한국정보과학회
KOREAN INSTITUTE OF INFORMATION SCIENTISTS AND ENGINEERS

제 19 권 제 1 호
Vol. 19 No. 1

SOFTWARE ENGINEERING SOCIETY

소프트웨어공학회 KIPS

2017

제19회 한국 소프트웨어공학 학술대회 논문집

Proceedings of the 19th Korea Conference on Software Engineering (KCSE 2017)

- 일시: 2017년 2월 8일(수) ~ 2월 10일(금)
- 장소: 강원도 평창 한화리조트(휘닉스파크점)

주최: 한국정보과학회, 한국정보처리학회
주관: 한국정보과학회 소프트웨어공학 소사이어티
한국정보처리학회 소프트웨어공학 연구회
한국전자통신연구원

후원: ㈜솔루션링크, ㈜코스콤,
㈜비트컴퓨터, ㈜모아소프트,
소프트웨어공학엑스퍼트그룹㈜, T3Q㈜,
STA 테스팅컨설팅㈜, 슈어소프트테크㈜,
TTA 소프트웨어시험인증연구소
SW 상시모니터링기술연구단
무인자율 및 적응형 소프트웨어센터

초대의 글

소프트웨어 공학인의 축제인 제19회 한국 소프트웨어 공학 학술대회(KCSE 2017)에 참가하러 오신 여러분을 환영합니다.

기업, 연구소 및 학계에서 활동하고 계신 소프트웨어공학 관련 전문가들의 모임인 한국정보과학회의 소프트웨어공학 소사이어티와 한국정보처리학회의 소프트웨어공학연구회에서는 여러 산업 분야에서 소프트웨어 공학의 중요성이 날로 부각되고 소프트웨어 공학 기술이 국가 발전에 중요한 부분으로 인식되는 시점에서 소프트웨어 공학 기술의 발전과 적용을 확대하고자 산, 학, 연이 협력하여 한국 소프트웨어 공학 학술대회를 개최하게 되었습니다.

이번 학술대회에서는 특히 "**제4차 산업혁명 시대의 소프트웨어공학 기술**"을 주제로 산, 학, 연 각계에서 제출한 논문 75편이 발표됩니다. 또한 Agile 개발환경, 소프트웨어공학과 오픈소스 소프트웨어, 소프트웨어 아키텍처 분석과 평가, 객체지향 분석 설계, 블록체인, 소프트웨어 아키텍처 복원을 포함한 총 6개의 유익한 튜토리얼이 준비되어 있습니다.

아울러 소프트웨어 공학 발전을 위해 애쓰고 계신 저명인사 3분의 기조연설도 있습니다. 본 학술대회가 소프트웨어 공학의 학문적 발전과 소프트웨어 산업기술 발전의 큰 장이 되는 만큼, 활발한 학술 및 기술 교류는 물론이거니와 격의 없는 토론이 진행될 수 있도록 여러분의 적극적인 참여를 부탁 드립니다. 이번 행사를 위해 수고해 주신 준비 위원들과 후원해 주신 기관 여러분께 진심으로 감사를 드립니다.

한국정보과학회 소프트웨어공학소사이어티 회장 강성원
한국정보처리학회 소프트웨어공학연구회 운영위원장 박용범

학술대회 준비 위원회

공동 대회장: 강성원 교수(KAIST), 박용범 교수(단국대)

조직위원장: 고인영 교수(KAIST)

조직위원: 백종문 교수(KAIST), 서주영 교수(아주대), 염근혁 교수(부산대), 유준범 교수(건국대),
이병정 교수(서울시립대), 이정원 교수(아주대), 정우성 교수(서울교대)

학술위원장: 이관우 교수(한성대)

학술위원: 김문주 교수(KAIST), 김순태 교수(전북대), 박용범 교수(단국대),
박찬진 박사(Nemustech), 배경민 교수(포항공대), 서영석 교수(영남대),
유철중 교수(전북대), 윤일철 교수(한국뉴욕주립대), 윤회진 교수(협성대),
이선아 교수(경상대), 이우진 교수(경북대), 이은서 교수(안동대), 이지현 교수(전북대),
이찬근 교수(중앙대), 정인상 교수(한성대), 조은숙 교수(서일대),
채흥석 교수(부산대), 최윤자 교수(경북대), 홍신 교수 (한동대), 홍장의 교수(충북대)

문의사항 연락처

학술대회 홈페이지 : http://www.sigsoft.or.kr/KCSE2017/
조 직 : 고인영 교수 (iko@kaist.ac.kr, 042- 350-3547)
학 술 : 이관우 교수 (kwlee@hansung.ac.kr, 02-760-5864)

* KCSE 2017 프로그램

2 월 8 일 (수)			
시간	**행 사 내 용**		
12:00–13:00	**KCSE 2017 등록**		
	튜토리얼 T1: Agile 개발환경 좌장: 이병정 교수(서울시립대) 장소: 세미나실 1	**튜토리얼 T2: 소프트웨어 공학과 오픈소스 SW** 좌장: 김순태 교수(전북대) 장소: 세미나실 2	**튜토리얼 T3: 소프트웨어 아키텍처 분석과 평가** 좌장: 이선아 교수(경상대) 장소: 세미나실 3
13:00-14:30 (90 분)	제목: Agile 개발환경의 중요성과 구축방안 김웅환 기술고문(SK Planet)	제목: SW 개발시 오픈소스 SW 활용과 소프트웨어공학 고려사항 송상효 교수(성균관대)	제목: 소프트웨어 아키텍처 분석과 평가 금창섭 책임(한국전자통신연구원)
14:30-14:40	**휴식**		
	튜토리얼 T4: 객체지향 분석 설계 좌장: 이정원 교수(아주대) 장소: 세미나실 1	**튜토리얼 T5: 블럭체인** 좌장: 김순태 교수(전북대) 장소: 세미나실 2	**튜토리얼 T6: 소프트웨어 아키텍처 복원** 좌장: 이선아 교수(경상대) 장소: 세미나실 3
14:40-16:10 (90 분)	제목: 사례 중심으로 소개하는 OOAD 이병정 교수(서울시립대)	제목: 블록체인 기초 이론과 활용 박용범 교수(단국대)	제목: 소프트웨어 아키텍처 모듈-뷰 복원을 위한 자동화 기법 이찬근 교수(중앙대)
16:10-16:20	**휴식**		
	개회식 장소: 그랜드홀 2		사회: 고인영 조직위원장(KAIST)
16:20-16:40 (20 분)	개회사: 강성원 회장(한국정보과학회 소프트웨어공학소사이어티), 박용범 위원장(한국정보처리학회 소프트웨어공학연구회) KCSE 2017 프로그램 소개: 이관우 학술위원장(한성대)		
	기조연설 I 장소: 그랜드홀 2		사회: 고인영 조직위원장(KAIST)
16:40-17:30	**하원규 박사(한국전자통신연구원)**		
	기조연설 II 장소: 그랜드홀 2		사회: 이관우 학술위원장(한성대)
17:30-18:20	**김인성 대표(엠포렌식센터)**		
18:30-19:30	**석 식**		

2 월 9 일 (목)

시 간	행 사 내 용				
	논문 발표 A				
	A1: SW 품질 1 좌장: 배경민교수 (포항공대) 장소: 그랜드홀 2	**A2: 도구 및 개발환경** 좌장: 이선아 (경상대) 장소: 세미나실 1	**A3: 보안 1** 좌장: 백종문 (KAIST) 장소: 세미나실 2	**A4: SW 테스팅 1** 좌장: 고인영 (KAIST) 장소: 세미나실 3	
09:00-10:40 (100 분)	소프트웨어 품질 측정을 위한 데이터 종합에서 역상관 방법 [단편논문] 안종선, 강성원 (KAIST) 소프트웨어 품질 가시화 전략 - 원격 개발 환경과 서비스 거버넌스 및 빅데이터를 활용한 End to End 모니터링 [후원업체] 박병훈(티쓰리큐(주)) 코드 수정 단위의 소프트웨어 결함 예측을 위한 정규화 기법 비교 분석 [단편논문] 정주상, 백종문(KAIST) Usability 와 Security 의 트레이드 오프 제거를 위한 품질 속성 요소 연결 지침 [단편논문] 노우리, 이석원 (아주대학교)	커널용 메모리 깨짐 검출기 [산업체논문] 우충기, 권진만, 이승훈, 이학봉, 권재욱 (㈜삼성전자) Toolchain 을 이용한 System Model 생성, 검증 자동화 제안 [단편논문] 이승민, 박용범 (단국대학교) 코드 클론을 효율적으로 관리하기 위한 시각화 방안 [일반논문] 김하영, 최은만 (동국대학교) 웨어러블 어플리케이션 개발을 위한 안드로이드 BLE API 에뮬레이터와 확장성에 관한 연구 [단편논문] 문현아, 박수용 (서강대학교), 최광훈 (전남대학교)	실시간 파일행위 모니터링을 통한 랜섬웨어 침해복구 연구 [단편논문] 김재열 (고려대학교) 화이트리스트 기반 감염 프로세스 검출 및 감염원 추적 [단편논문] 김시론, 박용범 (단국대학교) 데이터 통계를 이용한 국가 사이버위협 경보 수준 판단 [우수단편논문] 서형준, 조민경, 이상운, 배병철 (한국전자통신연구원 부설연구소) AOP 를 활용한 모바일 클라이언트와 서버간 민감 정보 전송의 동적 모니터링 방법 [단편논문] 최윤석, 최은만 (동국대학교)	국방무인기체계 SW 시험을 위한 DO-178C 적용방안 연구 [산업체논문] 류인수 ((주) 모아소프트) TPC-DI 기반의 ETL 솔루션 성능 테스트 사례 [산업체논문] 김상기, 강건희 (한국정보통신기술협회) 벤처/중소기업의 테스트 조직을 위한 한국형 테스트 성숙도 모델 시범 적용 사례 [산업체논문] 김기두 (한국정보통신기술협회), 김영철 (홍익대학교) 소프트웨어 변경의 테스트 범위를 정의하기 위한 산출물 정보 분석 [일반논문] 최효린, 이병정 (서울시립대학교), 이정원 (아주대학교)	**워크샵**: **SW 상시 모니터링 기술 연구단** 장소: 세미나실 6 (09:00-12:30)
10:40-10-50	**휴식**				

	논문 발표 B				
	B1: 분석 및 평가 좌장: 배경민교수 포항공대) 장소: 그랜드홀 2	**B2: 유지보수** 좌장: 이선아 (경상대) 장소: 세미나실 1	**B3: 보안 및 안전성** 좌장: 백종문 (KAIST) 장소: 세미나실 2	**B4: SW 품질 2** 좌장: 이관우 (한성대) 장소: 세미나실 3	
10:50-12:30 (100 분)	추상 도달가능성 그래프 기반 소프트웨어 모델체킹에서의 탐색전략 고려방법 [우수논문] 이낙원, 백종문 (KAIST) A Unified Approach for UML Based Safety Oriented Level Crossing Using FTA and Model Checking [일반논문] Anit Thapaliya, Gihwon Kwon (경기대학교) 감정 분석 기반의 사용자 피드백을 이용한 클라우드 서비스 평가 기법 [우수단편논문] 윤동규, 김웅수, 박준석, 염근혁 (부산대학교) 매쉬업 개발자를 위한 매쉬업 유사도 기반 서비스 추천 방법 [최우수논문] 김현승, 고인영 (KAIST)	코드 가독성 측정을 위한 소프트웨어 특징 목록 [단편논문] 최상철, 김순태 (전북대학교) 객체 지향 프로그램(C++)을 위장한 절차식(C) 패러다임 자동 식별화 구축 [단편논문] 변은영, 손현승, 장우성, 김영철 (홍익대학교), 김영수 (정보통신산업진흥원) N-그램 모델 기반의 코드 변경 추천 시스템 [단편논문] 김태현, 강성원 (KAIST), 이선아 (경상대학교), 금창섭 (한국전자통신연구원) 소프트웨어 유지보수 효율 향상 지원을 위한 의사코드 및 소스코드 양방향 자동 변환 연구 [학부논문] 권혁무, 장현수, 박동민, 서영석 (영남대학교)	모바일 아키텍처를 고려한 애플리케이션 보안 취약점 진단 방안 [단편논문] 김명근, 최은만 (동국대학교) 지능형 지속위협 생애주기를 고려한 보안요구사항 명세 방법 [단편논문] 김승준, 이석원 (아주대학교) 결함 분류체계와 휴먼 에러 분류체계를 적용한 SW 구현 단계 FMEA 수행 체계 [단편논문] 최이수, 한동준, 한혁수 (상명대학교) STPA 를 활용한 ISO 26262 기반의 안전분석 체계수립 [박사논문] 도성룡 (현대오트론), 한혁수 (상명대학교)	안드로이드 버그 분류: 이슈 설명서 기반 버그 담당자 선정 자동화에 대한 연구 [산업체논문] 최성옥, 조재호, 민모란, 민상윤 (KAIST 소프트웨어대학원) 컴포넌트 단위 학습을 이용한 버그 담당자 추천 [단편논문] 조충기, 김영민, 이기성, 이찬근 (중앙대학교) I-BL 기술의 성능 향상을 위한 버그리포트 품질 예측 및 자동 재구성 기술 [우수논문] 김미수, 안준, 이은석 (성균관대학교) 딥 러닝을 이용한 버그 담당자 자동 배정 연구 [우수논문] 이선로, 김혜민, 이찬근 (중앙대학교)	**워크샵**: **SW 상시 모니터링 기술 연구단** 장소: 세미나실 6 (09:00-12:30)
12:30-13:30	**중식**				

	논문 발표 C			
	C1: 리펙토링 & 유지보수 좌장: 홍장의 교수(충북대) 장소: 그랜드홀 2	**C2: 요구공학** 좌장: 김순태 교수(전북대) 장소: 세미나실 1	**C3: 설계 & 모델링** 좌장: 이정원 교수(아주대) 장소: 세미나실 2	**C4: 패턴추출 & 빅데이터** 좌장: 이찬근 교수(중앙대) 장소: 세미나실 3
13:30-15:10 (100 분)	파일들의 패키지 관계를 반영한 Change Coupling 추출 및 이를 이용한 리팩토링 대상 추천 [학부논문] 조영준, 허민재, 이찬근 (중앙대학교) 효율적인 리팩토링을 위한 조직 특성의 품질 지표 기반 우선순위 선정 자동화 [단편논문] 박지훈, 손현승, 박보경, 이진협, 김영철 (홍익대학교) Hadoop Platform 에서 UML 모델 기반 영상 처리 소스 코드 자동 생성 방안 [단편논문] 고미은, 박용범 (단국대학교) 테스트로부터 소스 코드로의 추적성 향상을 위한 소프트웨어 행위 모델 활용 [일반논문] 백승석, 이병정 (서울시립대학교), 이정원 (아주대학교)	인공지능 소프트웨어 개발을 위한 요구공학 프레임워크: 인지컴퓨팅 시스템을 중심으로 [일반논문] 안우람 김재홍, 김경모(KAIST 소프트웨어대학원), 민상윤(KAIST) 안전 요구사항 추출을 위한 안전성 분석 지침 개발 및 적용 사례 [단편논문] 정대희, 권기현 (경기대학교) 소프트웨어 요구사항 진화에서의 추적성 관리의 어려움 조사 [일반논문] 박종열, 류승희, 정세린, 이선아 (경상대학교) 오디오 컨텍스트의 라이프 시맨틱 분석 프레임워크 [최우수논문] 정한터, 김수동 (숭실대학교), 라현정 ((주)스마트랩)	JAVA 오픈 소스의 재사용성 및 확장성 향상을 위한 소프트웨어 설계 패턴 추출 기법 [일반논문] 최용석, 김두환, 홍장의 (충북대학교) 디지털 건강 분석 솔루션을 위한 소프트웨어 플랫폼 설계 [일반논문] 라현정(스마티랩), 정한터, 임성민, 김수동(숭실대학교) 기능 분석과 아키텍처 패턴을 이용한 자율주행 자동차 소프트웨어 아키텍처 설계 방법 [일반논문] 김순겸, 김두환, Nazakat Ali, 홍장의 (충북대학교) 행위 모델링 및 조합을 위한 객체지향성 [학부논문] 김희언 (세종대학교)	Cascade K-Medoids 기반 자바 메소드 식별과 패턴 자동 식별 기법 [단편논문] 김태영, 김순태, 이정휴 (전북대학교) 딥러닝을 활용한 주행 패턴 특징 추출 연구 [단편논문] 이주영, 장기태 (한국과학기술원), 이홍석, 정희진 (한국과학기술정보연구원) 추상구문트리의 연어 관계 분석을 통한 Java API 패턴 추출 및 추천 시스템 개발 [최우수논문] 권찬우, 황상원, 남영광 (연세대학교) 빅데이터 분석 방법론 [후원업체] 유태빈 ((주)코스콤)
15:10-15:20	**휴식**			

<table>
<tr><td></td><td colspan="2">논문 발표 D</td></tr>
<tr><td></td><td>D1: SW 중심 대학 (초청 발표 세션)
좌장: 이병정 교수 (서울시립대)
장소: 세미나실 1</td><td>D2: 우수 국제학술대회 발표논문 (초청 발표 세션)
좌장: 홍장의 교수(충북대)
장소: 그랜드홀 2</td></tr>
<tr><td>15:20-16:50
(90 분)</td><td>경북대학교 SW 중심대학사업단의 SW 가치확산 활동 사례 연구
정원일(경북대)

SW중심대학사업 수행사례- 성균관대학교 사례
이은석(성균관대)

비전공자를 위한 컴퓨팅사고력 교육과정-고려대학교 운영사례-
김현철(고려대)</td><td>Automated Model-Based Android GUI Testing using Multi-level GUI Comparison Criteria [ASE 2016]
Young-Min Baek and Doo-Hwan Bae (KAIST)

A Theoretical Framework for Understanding Mutation-Based Testing Methods [ICST 2016]
Donghwan Shin and Doo-Hwan Bae (KAIST)

Location-Based Web Service QoS Prediction via Preference Propagation for Improving Cold Start Problem [ICWS 2015]
Kwangkyu Lee, Jinhee Park, and Jongmoon Baik (KAIST)

An Efficient Resource Allocation Approach Based on a Genetic Algorithm for Composite Services in IoT Environments [ICWS 2015]
Minhyeop Kim and In-Young Ko (KAIST)</td></tr>
<tr><td>16:50-17:00</td><td colspan="2">휴식</td></tr>
<tr><td></td><td colspan="2">기조연설 III 사회: 이관우 학술위원장(한성대)
장소: 그랜드홀 2</td></tr>
<tr><td>17:00-18:00</td><td colspan="2">인호 교수 (고려대학교, 한국블록체인학회 회장)</td></tr>
<tr><td>18:00-21:00</td><td colspan="2">우수논문상, 공로상, 감사장 수여식 사회: 고인영 조직위원장(KAIST)
만찬 사회: 이병정(서울시립대)
장소: 그랜드홀 1</td></tr>
</table>

2 월 10 일 (금)

시 간	행 사 내 용			
	논문 발표 E			
	E1: 스마트 시스템 좌장: 김순태 교수(전북대) 장소: 그랜드홀 2	**E2: SW 테스팅 2** 좌장: 홍장의 교수(충북대) 장소: 세미나실 1	**E3: SW 프로세스** 좌장: 이찬근 교수(중앙대) 장소: 세미나실 2	**E4: 블록체인** 좌장: 이정원 교수(아주대) 장소: 세미나실 3
9:30-11:35 (125 분)	QoS 요구사항 만족을 위한 온톨로지 기반 태스크 매치메이킹 기술 [일반논문] 김민협, 고인영 (KAIST) 포그 컴퓨팅 환경에서 서비스 QoS 를 고려한 자동화된 서비스 인스턴스 선택 방법 [단편논문] 권정현, 고인영(KAIST) IoT 기반의 실시간 가축 건강 및 번식 관리를 위한 모바일 어플리케이션 개발 [우수단편논문] 김희진, 오세은, 최병주 (이화여자대학교), 안세혁 ((주)유라이크코리아) 다중 조명 제어 기기간의 최적화를 위한 자가 적응 의사 결정 방법 [단편논문] 김현우, 이의종, 백두권 (고려대학교)	시험 자동화도구를 적용한 소프트웨어 통합시험 커버리지 달성방안 [산업체논문] 장정훈, 이기영, 이원택, 강유선, 류인수 (㈜모아소프트) SILS 및 런타임 모니터링을 활용한 레거시 S/W 시나리오 테스팅 및 속성검증 사례연구 [우수단편논문] 박민규, 서보영, 조현승, 이호정 (LG 전자), 장훈 (현대자동차) 차량용 SW 의 HiL 테스팅을 위한 빅데이터 압축 수집 기법 [일반논문] 신종환, 최기용, 이정원 (아주대학교), 이병정 (서울시립대학교)	무기체계 SW 신뢰성 관리를 위한 프로세스 연구 [산업체논문] 서달미, 손근태, 서형오, 오정섭 (NSE), 김태현, 박삼준 (국방과학연구소) 한국형 테스트 성숙도 모델을 위한 경량화 연구 [우수단편논문] 박보경, 장우성, 김영철 (홍익대학교), 김기두 (한국정보통신기술협회), 이근상 (전북테크노파크), C. R. Carlson (IIT) 이슈 관리 프로세스의 정량적 평가 프레임워크 [단편논문] 오승원, 전은진, 한혁수 (상명대학교) ACMI Data Link Design and M&S Results [산업체논문] 심인보, 오지현, 김천영, 지철규 (국방과학연구소)	소비되지 않은 출력값 (UTXO) 추출 기법을 통한 블록체인 사이즈 축소 방안 [단편논문] 이동영, 박수용 (서강대학교) 블록체인 기반 전자투표 시스템 [단편논문] 이우승, 고동휘, 고덕윤, 박수용 (서강대학교) 스마트 계약 기반 탈 중앙 전자 투표 시스템 [단편논문] 노승학, 인호 (고려대학교) 모바일 디바이스 배터리 소모 분석 기법 평가 [우수단편논문] 송지영, 조치우, 지은경, 배두환 (KAIST)

	스마트 냉장고 상호작용 시 사용자 개입을 최소화하기 위한 스마트 스토리지 시스템 설계 및 프로토타입 개발 [학부논문] 박준희, 손희석, 김재헌, 이동만 (KAIST)	SW 테스트 수준 측정 기반의 테스트 중심 품질관리 프레임워크의 실무 활용 효과에 대한 연구 [산업체논문] 김관호, 권원일 (STA 테스트컨설팅), 정수진, 김명주, 유경훈, 이노원 (정보통신산업진흥원)	소프트웨어 개발 조직의 특성에 관한 연구 : 현상과 특성에 관한 조사 [일반논문] 김윤수, 김재욱, 이원영, 김태호, 민상윤 (KAIST 소프트웨어대학원)	소프트웨어 정의 네트워크 환경에서 애플리케이션 QoS 보장을 위한 모니터링 아키텍처 [단편논문] 황제승, 김웅수, 염근혁 (부산대학교), 박준석(부산대학교 물류혁신네트워킹연구소)
11:35-12:00 (25 분)	**폐회식** 장소: 그랜드홀 2			사회: 고인영 조직위원장(KAIST)

KCSE 2017 튜토리얼

튜토리얼 T1: Agile 개발환경

- 일시: 2월 8일(수) 13:00~14:30
- 장소: 세미나실 1
- 제목: Agile 개발환경의 중요성과 구축방안
- 연사: 김웅환 기술고문 (SK Planet)

- 튜토리얼 초록:

  Mobile, cloud 등 점점 더 복잡해져가는 Software 개발 요구사항을 대응하기 위하여 Agile한 개발환경이 중요한데, 그 방법으로 architecture 측면에서 MSA, 개발 infra측면에서 DevOps, culture측면에서 Scrum과 같은 agile process가 global하게도 hot trend로 자리잡고 있습니다. 이에 대한 중요성, 국내외 사례와 구축방안 그리고 각 측면에서 어떻게 서로 연관성이 있는지에 대하여 소개하고자 합니다.

- 김웅환 본부장 약력:

  2013-현재 SK Planet Tech Infra개발본부장, 현 기술고문
  2003-2013 Yahoo, Media Engineering Senior Director
  1990-2003 삼성SDS

튜토리얼 T2: SW공학과 오픈소스 소프트웨어

- 일시: 2월 8일(수) 13:00~14:30
- 장소: 세미나실 2
- 제목: SW 개발시 오픈소스 SW활용과 소프트웨어공학 고려사항
- 연사: 송상효 교수(성균관대)

- 튜토리얼 초록:

  오픈소스SW의 OSS분류/ OSS 비지니스/ OSS기반 소프트웨어개발 / OSS개발 프로세스 등에 대해서 전반적으로 발표를 진행한다. 소프트웨어 개발 시 사용하는 패키지 툴과 프로그램 코드를 적절하게 사용하는 방법을 알아보고, 특히 SW개발 시 참고해야 할 오픈소스SW 라이선스를 소개한다. 개발 단계별 오픈소스SW의 고려사항 및 검증을 설명하고, 마지막으로 공개된 오픈소스SW 코드를 사용 시에 발생할 수 있는 보안에 대해서 살펴본다.

- 송상효 교수 약력:

  2015년~현재 성균관대학교 산학협력중점교수
  2016년~현재 오픈소스소프트웨어재단 이사
  2014년~현재 전자정부 민간협력포럼 위원
  2012년~2015년 한국공개소프트웨어협회 회장
  2013년~현재 빅데이터전문가협의회 부의장
  2007년~현재 NEA OSS Promption Forum WG2(인력양성분과) 위원

튜토리얼 T3: 소프트웨어 아키텍처 분석과 평가

- 일시: 2월 8일(수) 13:00~14:30
- 장소: 세미나실 3
- 제목: 소프트웨어 아키텍처 분석과 평가
- 연사: 금창섭 책임연구원 (한국전자통신연구원)

- 튜토리얼 초록:

  소프트웨어아키텍처 평가는 시스템 개발의 초석이 되는 후보 아키텍처가 올바른지 분석하고 확인하는 과정을 의미한다. 본튜토리얼에서는 소프트웨어의 특징, 소프트웨어 아키텍처 분석과 평가 개요, 그리고 대표적인 시나리오 기 아키텍처 분석/평가 기법인 SAAM, ATAM, CBAM을 사례와 함께 소개한다.

- 금창섭 책임연구원 약력:

  1994년-현재 한국전자통신연구원 네트워크연구본부 PL(책임연구원)
  2013년 KAIST, 전산학 박사
  2005년 카네기멜론대학교, 소프트웨어공학 석사
  연구분야: 소프트웨어아키텍처, 5G 서비스 아키텍처, 인텔리전스 서비스플랫폼,
  모바일 엣지 클라우드

튜토리얼 T4: 객체지향 분석 설계

- 일시: 2월 8일(수) 14:40~16:10
- 장소: 세미나실 1
- 제목: 사례 중심으로 소개하는 OOAD
- 연사: 이병정 교수 (서울시립대)

- 튜토리얼 초록:

  본 강의에서는 사례 중심 객체지향 소프트웨어 분석/설계 방법을 소개한다. 먼저 요구사항으로부터 코드에 이르는 각 단계를 표준 객체지향 모델링 언어 UML을 사용하여 적용하는 과정을 배운다. 또한 설계 품질을 향상시키기 위하여 디자인 패턴을 설계/코드에 적용하는 과정을 살펴본다. 사례 중심으로 내용을 설명하고, 학습한 방법에 따라 프로젝트에서 작성되는 요구사항 정의서, 설계서 사례를 소개한다.

- 이병정 교수 약력:

  1986. 03 ~ 1990. 02 : 서울대학교 계산통계학과 (학사)
  1996. 03 ~ 1998. 02 : 서울대학교 전산과학과 (석사)
  1998. 03 ~ 2002. 02 : 서울대학교 컴퓨터공학과 (박사)
  2002. 03 ~ 현재 : 서울시립대학교 컴퓨터과학부 교수
  1990. 01 ~ 1998. 01 : ㈜현대전자 연구원

튜토리얼 T5: 블록체인

- 일시: 2월 8일(수) 14:40~16:10
- 장소: 세미나실 2
- 제목: 블록체인 기초 이론과 활용
- 연사: 박용범 교수(단국대)

- 튜토리얼 초록:

  블록체인은 무결성을 보장하는 보안 알고리즘을 기반으로 한다. 이러한 무결성은 상호 정보교환의 신용으로 이용될 수 있다. 본 튜토리얼은 블록체인 관련 기초 암호 이론을 간단히 살펴보고 순수한 형태의 블록체인을구성하는 방법을 소개한다. 이를 통해 암호학 또는 보안에 대해 자세한 지식이 없더라도 SE 분야에서 블록체인을 다룰 수 있는 기초 지식을 알려주는 것을 목표로 한다. 튜토리얼 후반은 이러한 블록체인이 어떻게 핀테크(FinTech)에 활용 될 수 있는지를 토론해보고 성문법을 따르는 우리나라 현실에 적용하는 것의 어려움에 대한 인사이트(Insight)를 전달한다..

- 박용범 교수 약력:

  1993 ~ 현재 단국대학교 교수
  1991.6 N.Y. Polytechnic (현 NYU-POLY), 전산학 박사
  1987.2 N.Y. Polytechnic (현 NYU-POLY), 전산학 석사
  연구분야: Intelligent Software Engineering, Artificial Neural Network, Secure Software Developments

튜토리얼 T6: 소프트웨어 아키텍처 복원

- 일시: 2월 8일(수) 14:40~16:10
- 장소: 세미나실 3
- 제목: 소프트웨어 아키텍처 모듈-뷰 복원을 위한 자동화 기법
- 연사: 이찬근 교수 (중앙대)

- 튜토리얼 초록:

  많은 소프트웨어 프로젝트에서 명시적인 소프트웨어 아키텍처와 설계 정보가 부재하거나 그들이 적절히 갱신되지 않아 유지보수 활동에서 어려움을 겪는 경우가 허다하다. 본 튜토리얼은 아키텍처 모듈-뷰를 자동적으로 복원하기 위한 기존의 연구와 근간 원리를 소개한다. 소스코드에 담긴 구조적 정보와 어휘적 정보를 추출하고 클러스터링 기법을 사용하여 모듈-뷰를 생성하는 기본적인 접근 방법, 이와 관련한 도구들, 그리고 최근의 연구 동향을 살펴본다.

- 이찬근 교수 약력:

  2007~현재 중앙대 컴퓨터공학부 조교수 및 부교수
  2005~2007 인텔 소프트웨어 엔지니어 (Oregon, USA)
  2005 Univ. of Texas at Austin, Computer Science 박사
  1998 KAIST, 전산학 석사
  1996 중앙대, 컴퓨터공학 학사
  연구분야: Software Architecture, Event-based Systems, Real-time Systems

KCSE 2016 기조연설

기조연설 I

- 일시: 2월 8일(수) 16:40~17:30
- 장소: 그랜드홀 2
- 제목: 제4차산업혁명 최전선! 삼라만상이 SW로 작동하는 시대
- 연사: 하원규 박사(한국전자통신연구원)

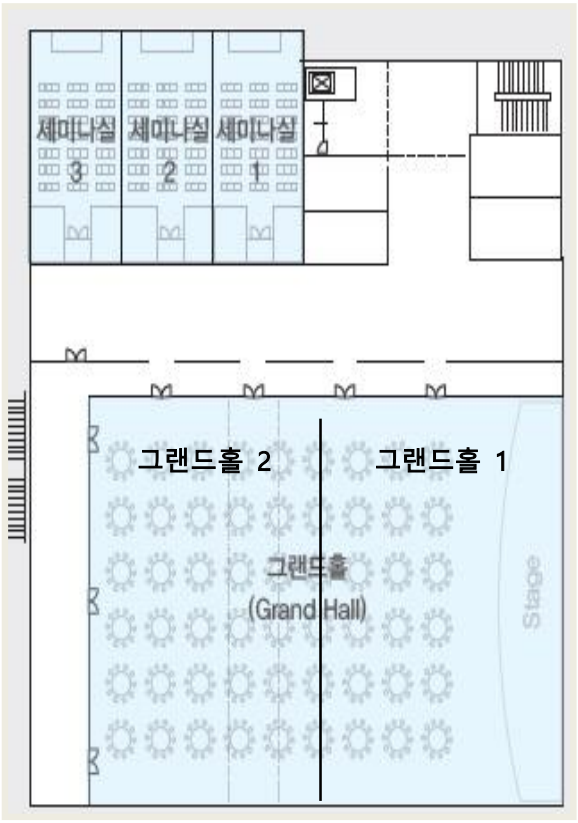
기조연설 II

- 일시: 2월 8일(수) 17:30~18:20
- 장소: 그랜드홀 2
- 제목: IT 전문가의 사회적 역할
- 연사: 김인성 대표 (엠포렌식센터)

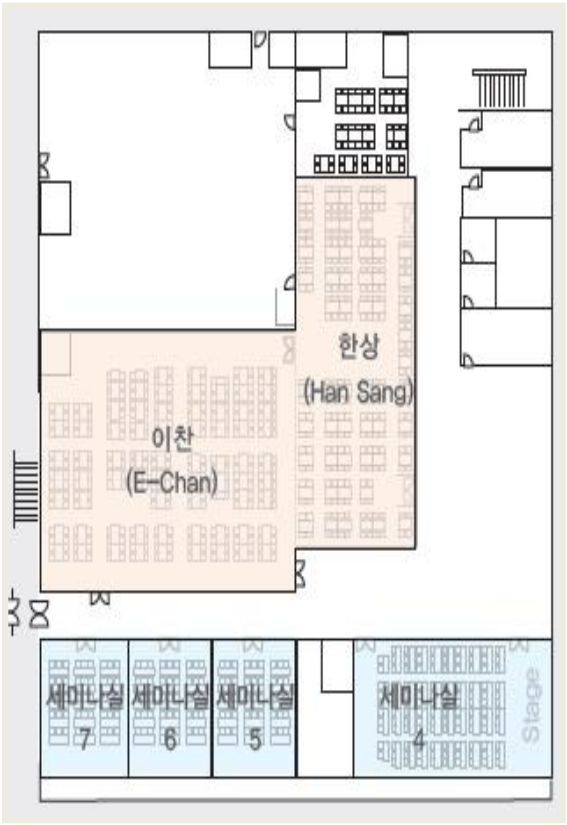
기조연설 III

- 일시: 2월 9일(목) 17:00~18:00
- 장소: 그랜드홀 2
- 제목: 블록체인: 4차 산업혁명의 핵심 인프라
- 연사: 인호 교수 (고려대학교)

행사장 및 식당 위치



1 F



B 1 F

* 휘닉스파크 내 한화리조트 위치(레드동)

소프트웨어 품질 측정을 위한 데이터 종합에서 역상관 방법

안종선 O 강성원
한국과학기술원
jsahn@kaist.ac.kr, sungwon.kang@kaist.ac.kr

Decorrelation Method in Data Aggregation for Measuring Software Quality

Jongsun Ahn O Sungwon Kang
KAIST

요 약

소프트웨어 품질을 측정할 때, 복잡도, 응집도 등과 같은 개발 산출물의 메트릭 등을 이용하면 소프트웨어 품질 측정을 자동화할 수 있다. 하지만 측정치 사이에서 상관도가 존재하면 평가가 잘못 이루어질 수 있다. 본 논문은 메트릭 사이의 상관도가 존재할 경우, 평가가 잘못 이루어질 수 있음을 몬테카를로 시뮬레이션을 통해 보이고 상관도를 제거한 결과가 정확히 평가될 수 있음을 보인다.

1. 서 론

소프트웨어 품질을 측정할 때, 복잡도, 응집도 등과 같은 개발 산출물의 메트릭을 측정하고 데이터 종합인 메트릭들을 가중합(Weighted Sum) 모델이나 가중곱(Weighted Product) 모델을 하여 다른 소프트웨어와 비교하는 다기준의사분석(Multi-Criteria Decision Analysis)를 수행한다[Blin 01]. 하지만 메트릭들 사이에 상관도가 존재하면 다른 소프트웨어와 비교할 때, 평가가 잘못 이루어질 수 있다.

소프트웨어 품질 측정에 다기준의사분석이 사용된 연구들은 측정값들 사이의 상관도에 대한 분석을 수행하지 않았다. 기존 방법들은 PROMETHEE나 ERECTRE와 같은 다기준의사분석을 포함하는 방법을 제안하였다. 소프트웨어간 품질 비교 시에 평가가 잘못 이루어질 가능성이 존재한다. 따라서, 메트릭들 사이에 상관도를 제거하는 역상관 기법 적용이 필요하다.

본 논문은 데이터 종합에 가중합 모델을 사용하여 소프트웨어와 비교할 때, 메트릭들 사이에 상관도가 존재하면 비교평가가 잘못 이루어질 수 있음을 몬테카를로 시뮬레이션을 통해 보인다. 그리고 상관도를 제거한 결과가 정확히 평가될 수 있음을 시뮬레이션을 통해 보인다.

본 논문은 2장에서 품질 측정 모델에 대해 정의하고 3장에서 소프트웨어 품질 측정을 위한 개발 산출물로부터 측정치 사이에 상관도가 소프트웨어간 품질 비교에 영향을 줄 수 있음을 확인한다. 4장에서는 상관도를 제거하는 방법을 제안하여 정확한 평가가 가능함을 보인다. 끝으로 5장에서는 관련 연구를, 6장에서는 결론을 기술한다.

2. 품질 측정 모델

품질은 다차원 벡터 공간[Jung04]으로 나타낼 수 있으므로 이를 품질 공간으로 정의하고 다음과 같이 가중합 모델의 품질 측정 모델을 정의한다.

$$f: \mathbb{R}^M \rightarrow \mathbb{R}^N \quad (1)$$

$$\mathbf{p} = f(\mathbf{q}) = \mathbf{A}\mathbf{q} \quad (2)$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NM} \end{bmatrix} \quad (3)$$

선정한 품질 속성의 직접 측정 가능한 품질 속성이 N개이고 품질 공간은 M 차원임을 정의한다. 품질 공간을 정의역, 품질 속성 공간을 공역으로 하는 선형 변환 f를 식 (2)로 정의할 수 있다. 행렬 A의 크기는 N × M이다.

위에서 정의한 모델을 가중합 모델로 만드는 선형변환 g를 정의하면 다음과 같다. y는 가중합 모델로 계산된 결과이다. w는 가중치 벡터이다.

$$g: \mathbb{R}^N \rightarrow \mathbb{R}^1 \quad (5)$$

$$y = g(\mathbf{p}) = \mathbf{w}^T\mathbf{p} = \mathbf{w}^T\mathbf{A}\mathbf{q} \quad (6)$$

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}^T \quad (7)$$

선형 변환 g는 모든 품질을 종합하여 나타내므로

공역이 1차원이다.

3. 데이터 종합 방법과 상관도

문제를 정의하기 위해서 두 가지 측정 대상 소프트웨어 A, B가 존재할 때 B보다 A를 선호한다고 가정한다. 소프트웨어 A에서 측정된 값은 $p_{1,A}$, $p_{2,A}$, $p_{3,A}$로 정의하고 소프트웨어 B에서 측정한 값은 $p_{1,B}$, $p_{2,B}$, $p_{3,B}$로 정의한다. 따라서 각 소프트웨어를 측정한 결과는 아래와 같이 나타낼 수 있다.

$$y_A = w_1 p_{1,A} + w_2 p_{2,A} + w_3 p_{3,A} \quad (8)$$

$$y_B = w_1 p_{1,B} + w_2 p_{2,B} + w_3 p_{3,B} \quad (9)$$

식 (8), (9)의 w는 해당 속성의 가중치이다. B보다 A를 선호하는 조건을 가정하여 보면 첫 번째 속성이 제일 중요도가 크고 두 번째 속성은 두 번째 중요도를 갖고 세 번째 속성이 가장 낮은 중요도를 갖는다고 하였을 때, 다음 다섯 가지 조건을 정의할 수 있다.

$$p_{1,A} > k p_{1,B} \quad (10)$$

$$p_{2,A} < p_{2,B} \quad (11)$$

$$p_{3,A} < p_{3,B} \quad (12)$$

$$w_1 > w_2 > w_3 \quad (13)$$

$$E[p_2 p_3] = a \quad (14)$$

식(10)은 가장 중요도가 있는 속성이 소프트웨어 A에서 k배 만큼 크다고 가정한 것이다. 따라서 중요도가 크고 값이 크므로 소프트웨어 A는 B보다 선호될 수 있다. 식 (11)과 (12)는 소프트웨어 A가 B보다 좋지 않음을 보여준다. 식 (13)은 위에서 설명한 속성의 중요도의 크기를 나타낸다. 식 (14)는 두 번째 속성과 세 번째 속성이 a만큼 상관관계가 있음을 나타낸다. a크기는 −1에서 1사이의 값을 갖는다. 위의 조건일 때, $y_A < y_B$ 인 결과가 나온다면 품질에 대한 평가가 정확하다고 볼 수 없다. 위의 조건을 가지고 몬테카를로 시뮬레이션을 한 결과는 다음과 같다. 품질 속성은 표준 연속 균등 분포를 가지는 확률 변수로 가정하였다.

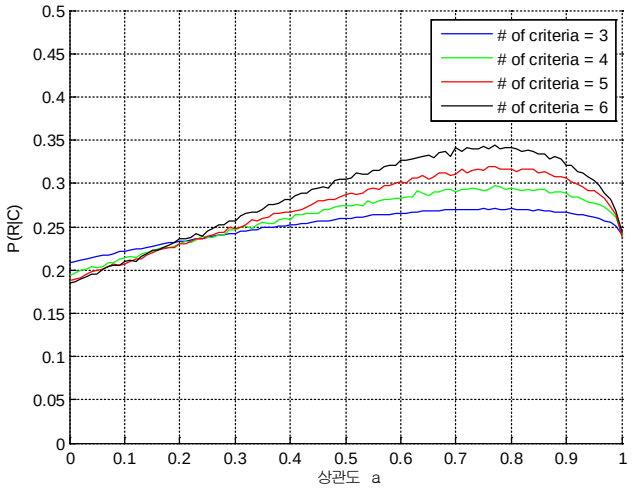


그림 1. 상관도에 따라 사건 R이 발생할 확률

표 1. 그림 1에서 사용된 가중치

속성의 수	가중치
3	w_1=0.5, w_2=0.3, w_3=0.2
4	w_1=0.5, w_2=0.2, w_3=0.15, w_4=0.15
5	w_1=0.5, w_2=0.2, w_3=0.1, w_4=0.1, w_5=0.1
6	w_1=0.5, w_2=0.15, w_3=0.1, w_4=0.1, w_5=0.1, w_6=0.15

그림 1은 $y_A < y_B$ 인 결과를 사건 R로 정의하고 위의 조건을 만족하는 집합을 C라고 했을 때 사건 R이 어떤 확률로 나타나는지 확인한다. 표 1은 그림 1에서 사용된 속성의 개수와 가중치 값을 보여준다. 속성의 개수가 많아지고 상관도가 커질 수록 사건 R이 많이 발생하였다. 따라서 속성 사이의 상관도는 소프트웨어의 품질을 잘못 평가할 수 있게 만든다.

4. 상관도를 제거하는 데이터 종합 방법

상관도가 문제 원인에 있으므로 상관도를 제거하는 방법을 제안한다. 상관도를 제거하는 방법은 역상관(decorrelation)이 있다. 품질 속성 모델의 행렬 A의 역행렬을 곱하여 속성들이 직교하게 하는 방법이다.

$$y_{m1} = \mathbf{w}_{m1}^T \mathbf{p} = \left(\left(\mathbf{A}^{-1}\right)^T \mathbf{w}\right)^T \mathbf{p} = \left(\left(\mathbf{A}^{-1}\right)^T \mathbf{w}\right)^T \mathbf{A}\mathbf{q} = \mathbf{w}^T \mathbf{A}^{-1} \mathbf{A}\mathbf{q} = \mathbf{w}^T \mathbf{q} \quad (15)$$

$$\mathbf{w}_{m1} = \left(\mathbf{A}^{-1}\right)^T \mathbf{w} \quad (16)$$

식 (15)에서와 같이 품질 공간의 속성은 직교하게 된다. 따라서, 식 (16)은 직교하게 만드는 가중치를 나타낸다.

그림 2은 역상관을 적용하지 않은 결과이고 그림 3은 역상관을 적용한 결과이다. 결과에서 잘못 평가할 확률이 낮아짐을 확실하게 알 수 있다.

행렬 A는 직접 알 수 있는 정보가 아니므로 A를 추정해야 한다. 추정하는 방법은 측정한 값을 가지고 공분산 행렬 R을 추정하고 촐스키 분해(Cholesky Decomposition)을 수행하여 행렬 A를 추정 할 수 있다.

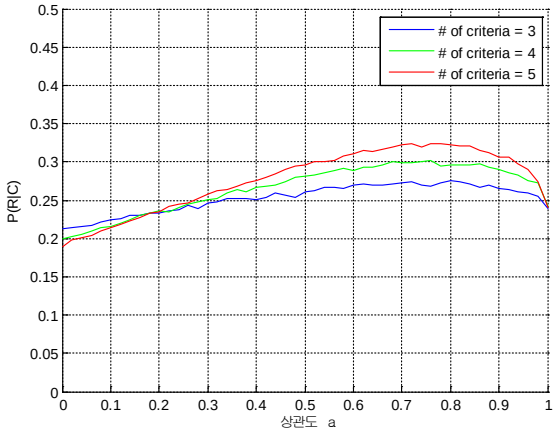


그림 2. 역상관 적용하지 않을 때의 사건 R이 발생할 확률

그림 4와 그림 5는 Cholesky Decomposition을 수행하여 추정한 공분산 행렬로 역상관을 수행한 결과이다. 그림 4의 그래프는 샘플 100개로 추정한 결과이고 그림5의 그래프는 소프트웨어 샘플 10개로 추정한 결과이다. 이것은 적은 소프트웨어 샘플로도 효과가 있음을 나타낸다.

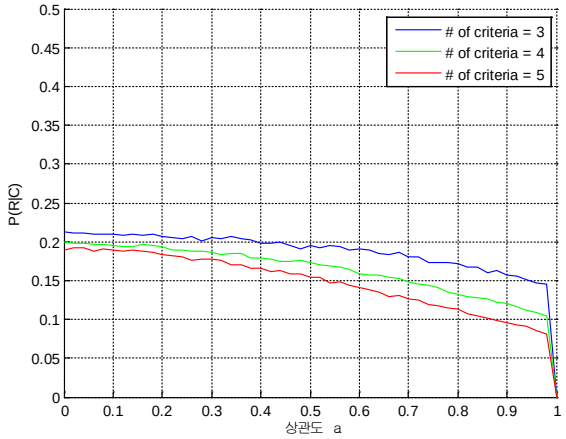


그림 3. 역상관 적용했을 때의 사건 R이 발생할 확률

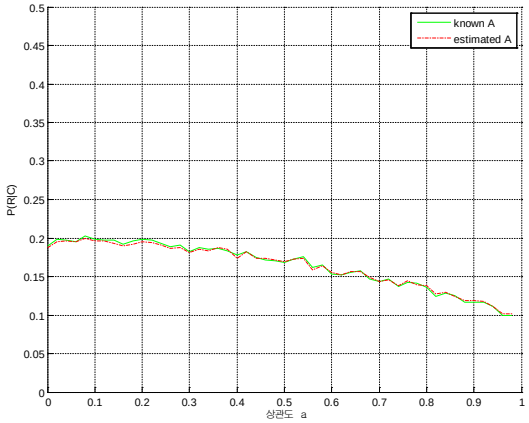


그림 4. 100개의 샘플로 공분산 행렬을 추정했을 때의 결과

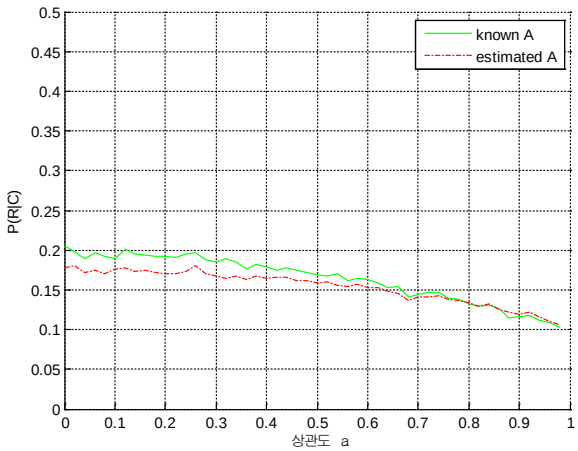


그림 5. 10개의 샘플로 공분산 행렬을 추정했을 때의 결과

5. 관련연구

기존 연구에 대한 조사는 ISO/IEC 25000 표준은 계층적 품질 모델을 제안하고 있으므로 계층적 품질 모델을 가지는 연구를 조사하였다. 해당하는 연구는 [Koscianski 99], [Blin 01], [Banisya 02], [Chang 07], [Halim 11], [Yang 12], [Ossadnik 13] 등이 있다. [Koscianski 99]의 연구는 가중합 모델을 사용하였으며 AHP로 종합하였다. 계층의 최하위 속성은 전문가와 툴을 이용하여 측정하였으나 속성 사이의 상관 관계를 고려하지 않았다. [Blin 01]과 [Ossadnik 13]의 연구는 속성 측정을 모두 전문가에게 의존하였고 간단한 예제로 수행결과를 나타내었다. [Banisya 02]의 연구는 가중합 모델을 사용하여 계층적 품질 모델에서 상위 품질을 종합하여 나타내었으나 가중치 선정의 이유가 존재하지 않아 속성 사이의 상관 관계를 고려하였는지 알 수 없다. [Chang07]의 연구는 Fuzzy AHP를 사용하여 기존 연구들과 다른 종합 방법을 제안하였지만 속성 측정 방법이 전문가에게 의존하여야 하므로 기존 방법들과 같이 한계가 극명하다. [Halim 11]의 연구는 AHP와 PROMETHEE를 융합하는 방법을 제안하였고 수행결과를 예제로 보여주었으며 민감도 분석을 수행하였다. PROMETHEE를 사용하여 다른 속성들과 상관 관계를 고려하였지만 해당 상관 관계는 관찰하여 알아낸 상관관계가 아닌 전문가가 판단하여 적용하는 상관 관계이기 때문에 전문가의 판단이 매우 중요하다. 그러므로 해당 연구는 도구의 자동화를 더욱 어렵게 만들었다. [Yang 12]의 연구는 Fuzzy Logic을 사용하여 가중합 모델이 아닌 비선형 모델을 제안하였다. 하지만 Fuzzy Logic은 설계 시에 전문가 모델이 반드시 필요하므로 새로운 측정 모델을 만들 때마다 전문가가 필요하므로 도구화가 어렵다.

기존 연구들의 특징은 전문가로부터 품질 측정을 하므로 측정 비용이 증가한다. 개발 산출물로부터 직접적으로 측정한다면 측정의 자동화를 통해 비용 절감이 가능하지만 개발 산출물로 측정된 측정 값의 특과 각 속성의 사이의 상관 관계에 대한 분석이 필요하다.

6. 결 론

본 논문은 소프트웨어 품질을 측정하기 위하여 자동화된 방법을 위해서 계층적 품질 모델을 적용하였다. 계층적 품질 모델에서 각 품질 모델은 상관도가 존재할 수 있고 소프트웨어 품질을 평가하는 것에 영향을 줄 수 있음을 보였다. 그에 대한 해결 방법으로 역상관을 적용하였으며 직접 알 수 없는 정보인 공분산 행렬도 측정된 소프트웨어 품질 속성 값으로부터 추정하여 효과가 있음을 보였다. 직접적으로 소프트웨어를 측정하지 않고 확률 모델을 사용하여 결과를 나타났었음으로 추후 연구로써 실제 소프트웨어를 측정하여 제안한 방법이 효과가 있음을

보여야 한다.

7. 참고문헌

[Kan 02] S. H. Kan, *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[Jung 04] H.-W. Jung, S.-G. Kim, and C.-S. Chung, "Measuring software product quality: A survey of ISO/IEC 9126," *Software, IEEE*, vol. 21, no. 5, pp. 88-92, 2004.

[Chang 07] C.-W. Chang, C.-R. Wu, and H.-L. Lin, "Integrating fuzzy theory and hierarchy concepts to evaluate software quality," *Software Quality Journal*, vol. 16, no. 2, pp. 263-276, Nov. 2007.

[Yang 12] H. Yang, "Measuring Software Product Quality with ISO Standards Base on Fuzzy Logic Technique," in *Affective Computing and Intelligent Interaction*, Springer, 2012, pp. 59-67.

[Blin 01] M.-J. Blin and A. Tsoukiàs, "Multi-criteria methodology contribution to the software quality evaluation," *Software Quality Journal*, vol. 9, no. 2, pp. 113-132, 2001.

[Ossadnik 13] W. Ossadnik and R. Kaspar, "Evaluation of AHP software from a management accounting perspective," *Journal of Modelling in Management*, vol. 8, no. 3, pp. 305-319, 2013.

[Koscianski 99] A. Koscianski and J. Candido Bracarense Costa, "Combining analytical hierarchical analysis with ISO/IEC 9126 for a complete quality evaluation framework," in *Software Engineering Standards, 1999. Proceedings. Fourth IEEE International Symposium and Forum on*, 1999, pp. 218-226.

[Bansiya 02] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," Software Engineering, IEEE Transactions on, vol. 28, no. 1, pp. 4-17, 2002.

[Halim 11] A. Halim, A. Sudrajat, A. Sunandar, I. K. R. Arthana, S. Megawan, and P. Mursanto, "Analytical Hierarchy Process and PROMETHEE application in measuring object oriented software quality," in 2011 International Conference on Advanced Computer Science and Information Systems, 2011, pp. 165-170.

소프트웨어 품질 가시화 전략

- 원격 개발 환경
- 서비스 거버넌스 (정적 분석)
- 빅데이터를 활용한 E2E 모니터링 (동적 분석)

품질 가시화 필요성

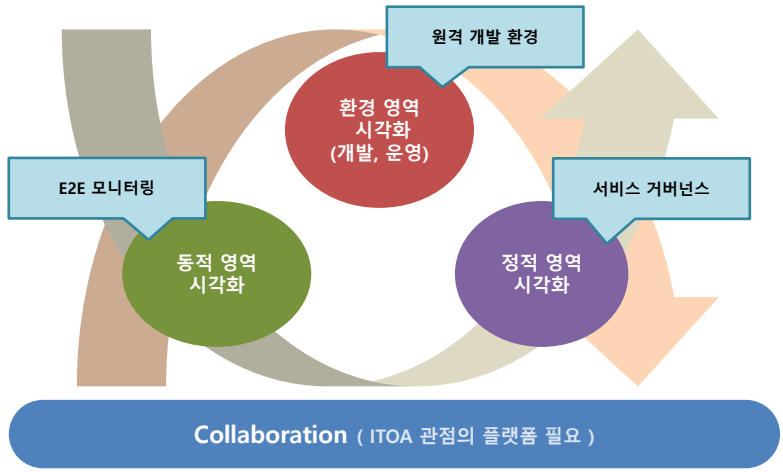
● 다음과 같은 목표 달성에 있어서 고객사와 개발사 간에 어떤 생각의 차이가 있을까요?

출처 : http://www.etnews.com/201301070406

- 고객사의 생각은 ?
- 개발사의 생각은 ?
- 생각이 다른 이유는?
- 어떻게 해야 할까요?

품질 가시화 방안

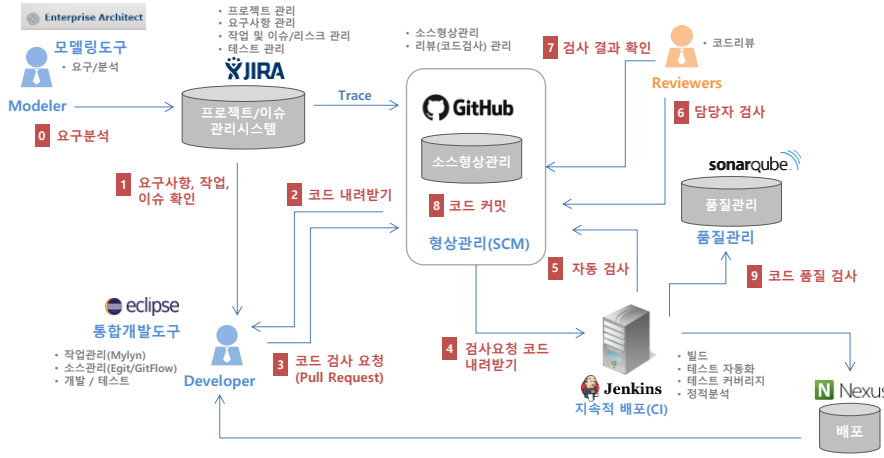
● 품질 가시화를 위해 3가지 영역에서의 시각화가 필요하며 각 영역은 Collaboration을 통해 전 영역의 품질 가시화를 완성할 수 있음



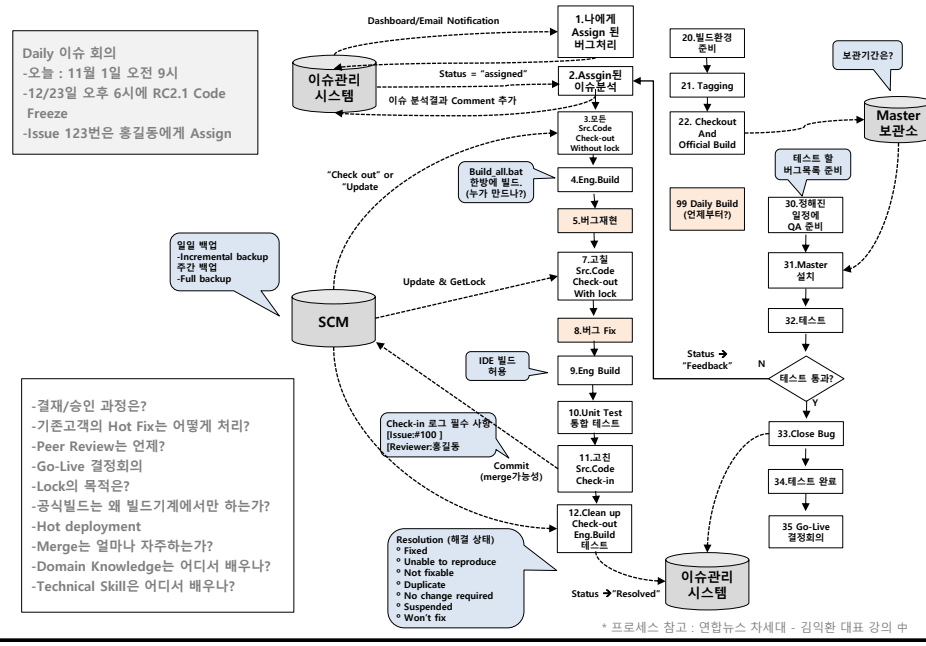
환경 영역 시각화 - 원격 개발 환경

원격 개발 환경 - 통합개발 및 관리 환경 (JIRA, GitHub)

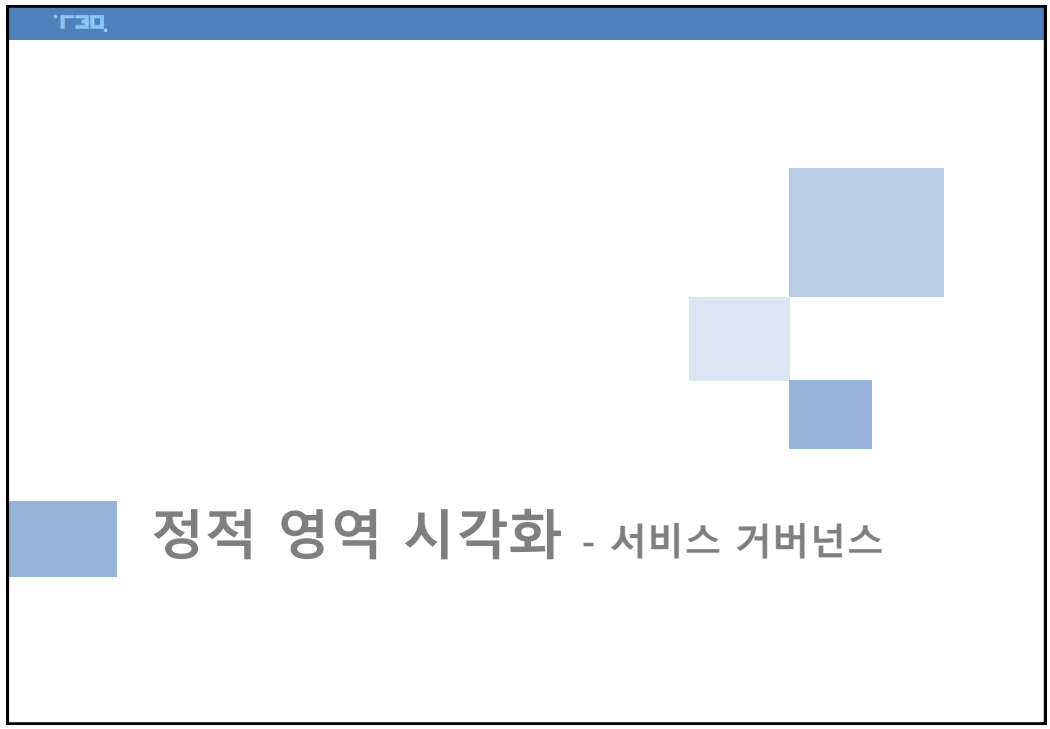
검증된 통합 개발 및 관리 환경을 수립해야 하며, 향후 운영 환경으로 이관될 수 있어야 함
1. 얻고자 하는 가치를 정의하고, 2. 가치를 얻기 위한 프로세스를 정의하고,
3. 가치를 얻기 위해 효율적으로 사용될 수 있는 프로세스 지원 도구 선정, 4. 도구를 활용한 프로세스 재정의
5. 도구 활용 기법 및 템플릿 작성 6. 교육, 7. Do It!

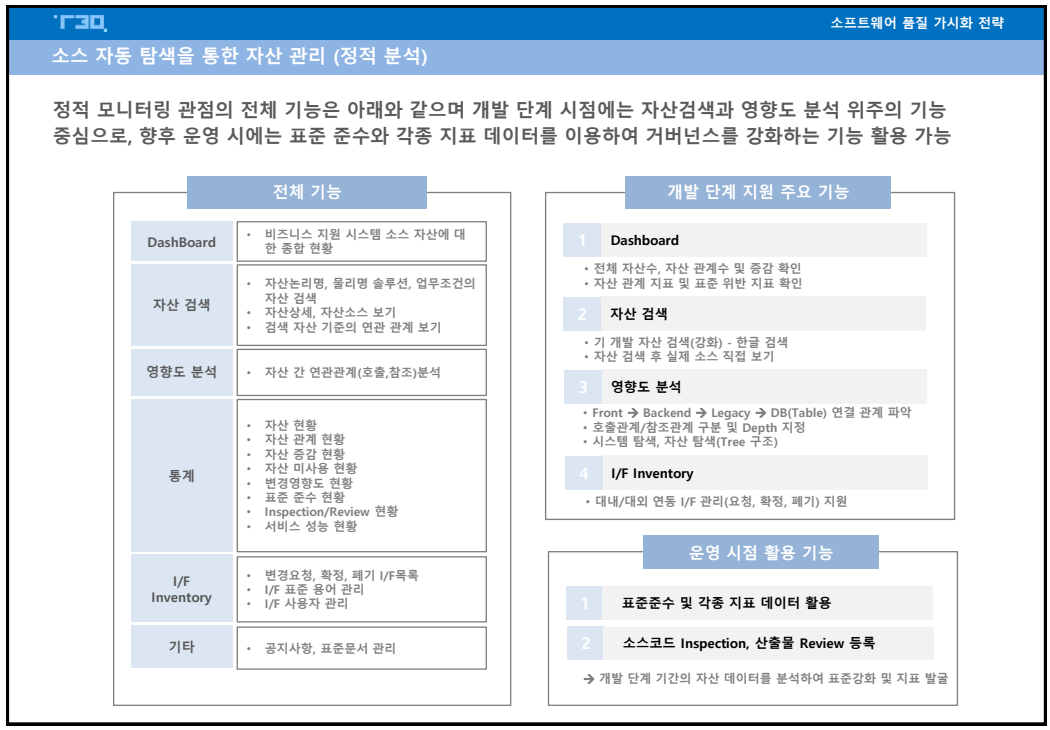


원격 개발 환경 - 구현 / 빌드 / 테스트 절차



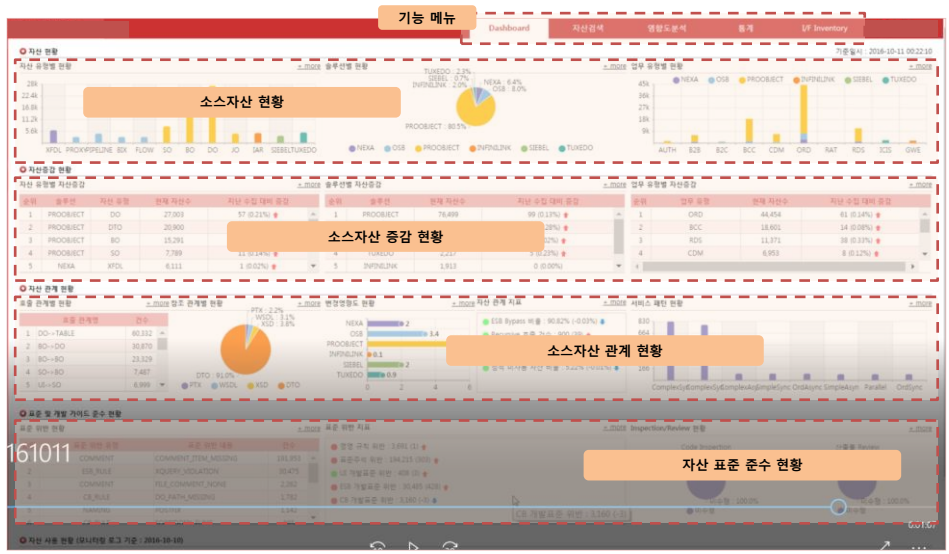
* 프로세스 참고 : 연합뉴스 차세대 - 김의환 대표 강의 中


정적 영역 시각화 - 서비스 거버넌스


소프트웨어 품질 가시화 전략
소스 자동 탐색을 통한 자산 관리 (정적 분석)
정적 모니터링 관점의 전체 기능은 아래와 같으며 개발 단계 시점에는 자산검색과 영향도 분석 위주의 기능 중심으로, 향후 운영 시에는 표준 준수와 각종 지표 데이터를 이용하여 거버넌스를 강화하는 기능 활용 가능
전체 기능
DashBoard
비즈니스 지원 시스템 소스 자산에 대한 종합 현황
자산 검색
자산논리명, 물리명 솔루션, 업무조건의 자산 검색
자산상세, 자산소스 보기
검색 자산 기준의 연관 관계 보기
영향도 분석
자산 간 연관관계(호출,참조)분석
통계
자산 현황
자산 관계 현황
자산 증감 현황
자산 미사용 현황
변경영향도 현황
표준 준수 현황
Inspection/Review 현황
서비스 성능 현황
I/F Inventory
변경요청, 확정, 폐기 I/F목록
I/F 표준 용어 관리
I/F 사용자 관리
기타
공지사항, 표준문서 관리
개발 단계 지원 주요 기능
1 Dashboard
• 전체 자산수, 자산 관계수 및 증감 확인
• 자산 관계 지표 및 표준 위반 지표 확인
2 자산 검색
• 기 개발 자산 검색(강화) - 한글 검색
• 자산 검색 후 실제 소스 직접 보기
3 영향도 분석
• Front → Backend → Legacy → DB(Table) 연결 관계 파악
• 호출관계/참조관계 구분 및 Depth 지정
• 시스템 탐색, 자산 탐색(Tree 구조)
4 I/F Inventory
• 대내/대외 연동 I/F 관리(요청, 확정, 폐기) 지원
운영 시점 활용 기능
1 표준준수 및 각종 지표 데이터 활용
2 소스코드 Inspection, 산출물 Review 등록
→ 개발 단계 기간의 자산 데이터를 분석하여 표준강화 및 지표 발굴

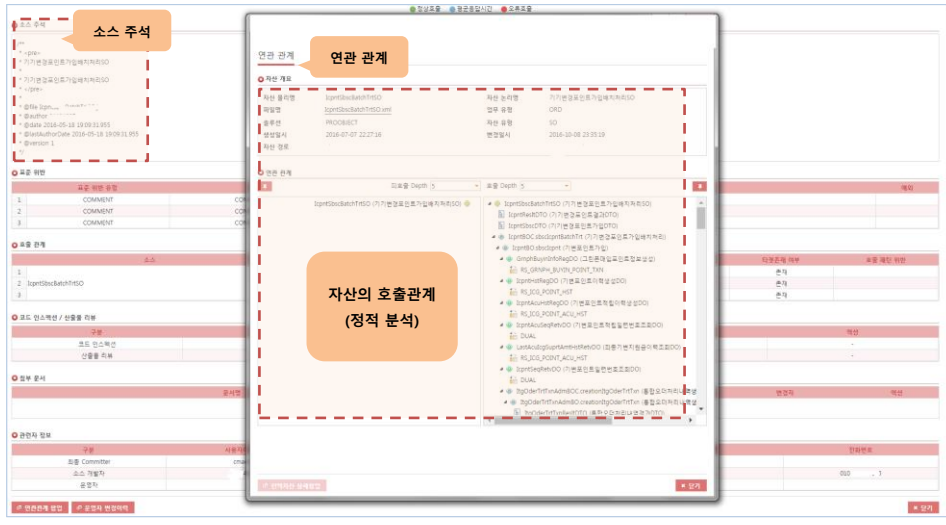
정적 영역 시각화 1

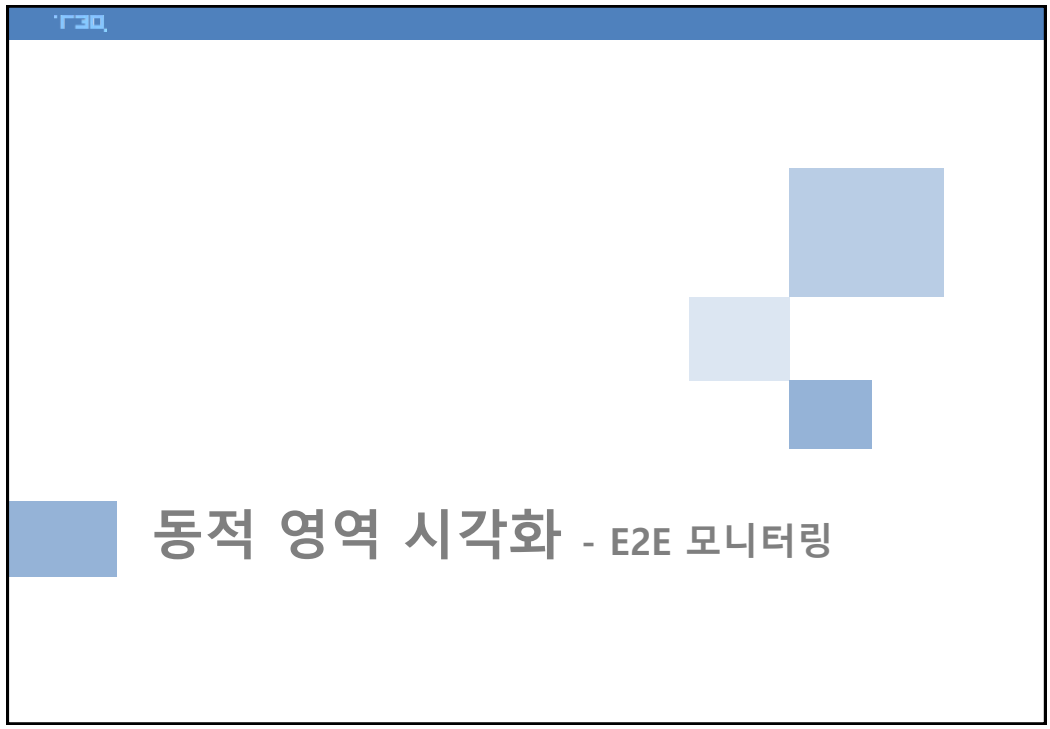
소스 자동탐색을 통한 소스자산관리 Dashboard – 배포된 App. 기준으로 소스 실시간 분석

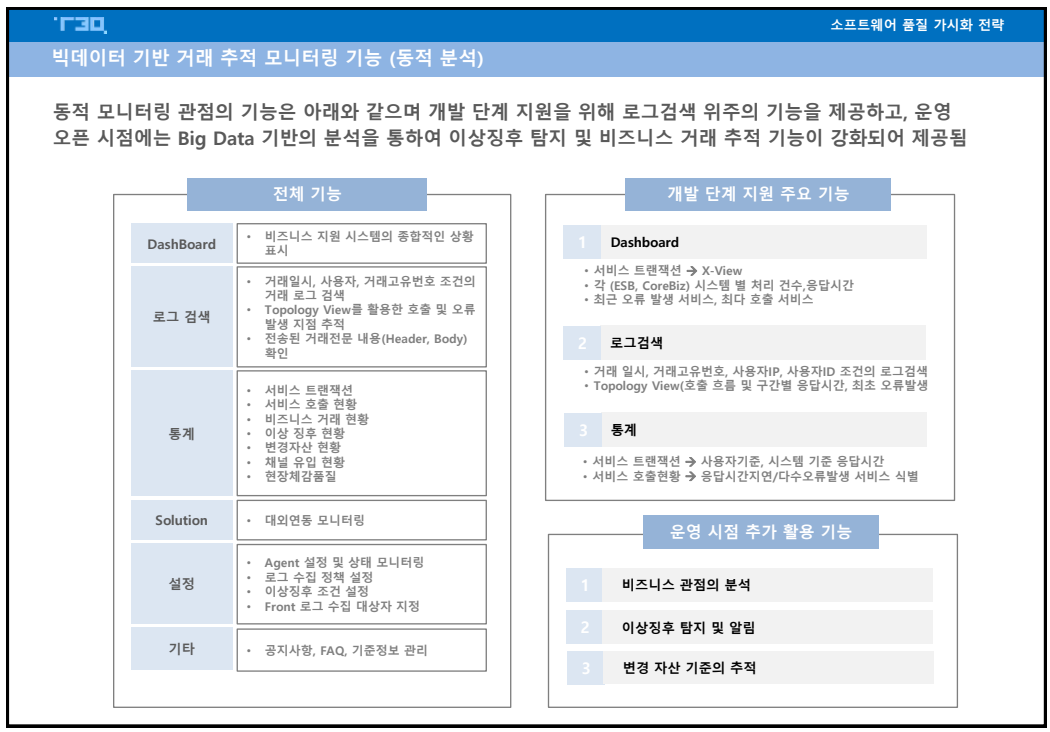


정적 영역 시각화 2

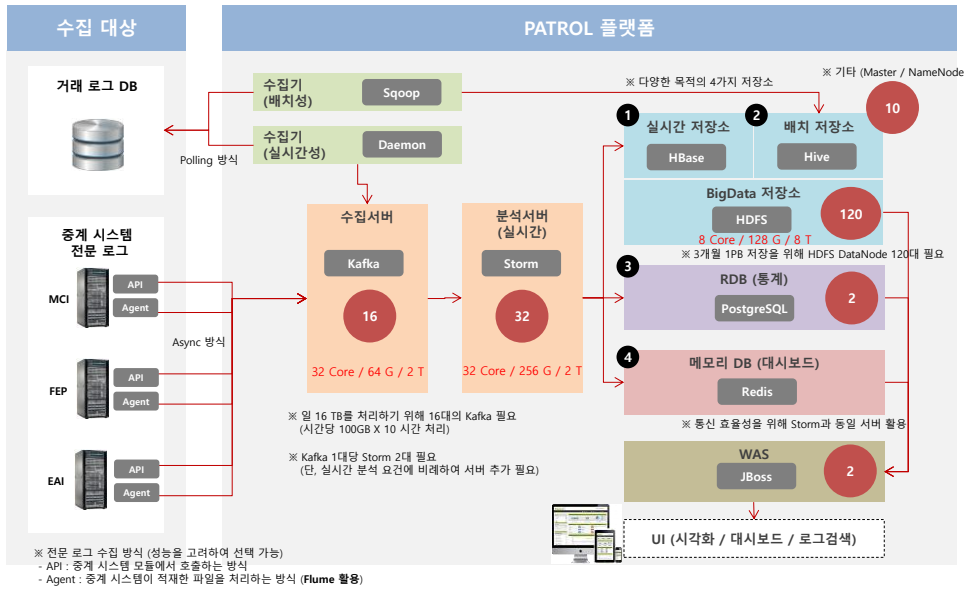
실시간 거래 모니터링 Dashboard – 실시간 구간 선택 → 특정 거래에 대한 Topology View (상세) → 특정서비스 선택 시 → 정적 분석(소스코드/연관관계/호출관계/품질 등의)과 연계하여 소스 구조적 정보를 보여 줌.




동적 영역 시각화 - E2E 모니터링

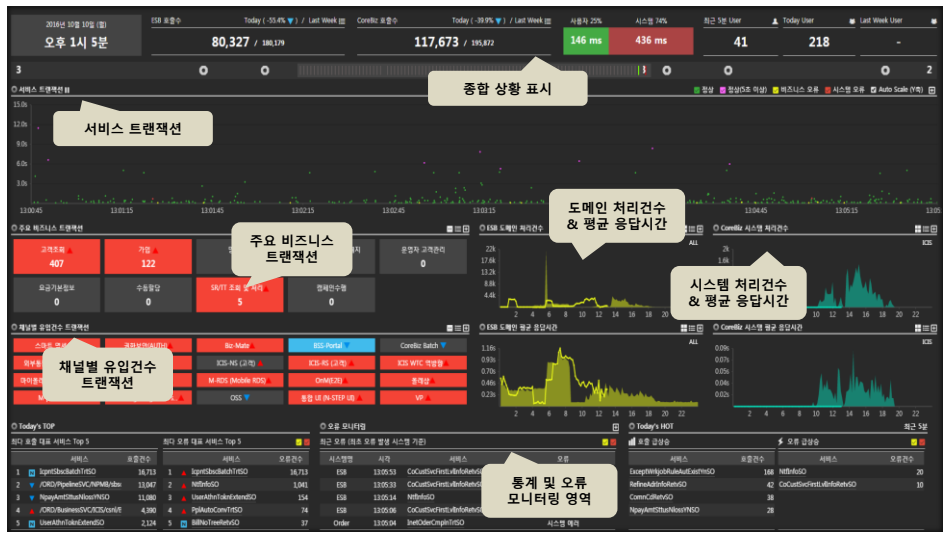

소프트웨어 품질 가시화 전략
빅데이터 기반 거래 추적 모니터링 기능 (동적 분석)
동적 모니터링 관점의 기능은 아래와 같으며 개발 단계 지원을 위해 로그검색 위주의 기능을 제공하고, 운영 오픈 시점에는 Big Data 기반의 분석을 통하여 이상징후 탐지 및 비즈니스 거래 추적 기능이 강화되어 제공됨
전체 기능
DashBoard
비즈니스 지원 시스템의 종합적인 상황 표시
로그 검색
거래일시, 사용자, 거래고유번호 조건의 거래 로그 검색
Topology View를 활용한 호출 및 오류 발생 지점 추적
전송된 거래전문 내용(Header, Body) 확인
통계
서비스 트랜잭션
서비스 호출 현황
비즈니스 거래 현황
이상 징후 현황
변경자산 현황
채널 유입 현황
현장체감품질
Solution
대외연동 모니터링
설정
Agent 설정 및 상태 모니터링
로그 수집 정책 설정
이상징후 조건 설정
Front 로그 수집 대상자 지정
기타
공지사항, FAQ, 기준정보 관리
개발 단계 지원 주요 기능
1 Dashboard
서비스 트랜잭션 → X-View
각 (ESB, CoreBiz) 시스템 별 처리 건수,응답시간
최근 오류 발생 서비스, 최다 호출 서비스
2 로그검색
거래 일시, 거래고유번호, 사용자IP, 사용자ID 조건의 로그검색
Topology View(호출 흐름 및 구간별 응답시간, 최초 오류발생
3 통계
서비스 트랜잭션 → 사용자기준, 시스템 기준 응답시간
서비스 호출현황 → 응답시간지연/다수오류발생 서비스 식별
운영 시점 추가 활용 기능
1 비즈니스 관점의 분석
2 이상징후 탐지 및 알림
3 변경 자산 기준의 추적

E2E PATROL 아키텍처 적용(안) 구성 (일 16 TB 생성 기준)



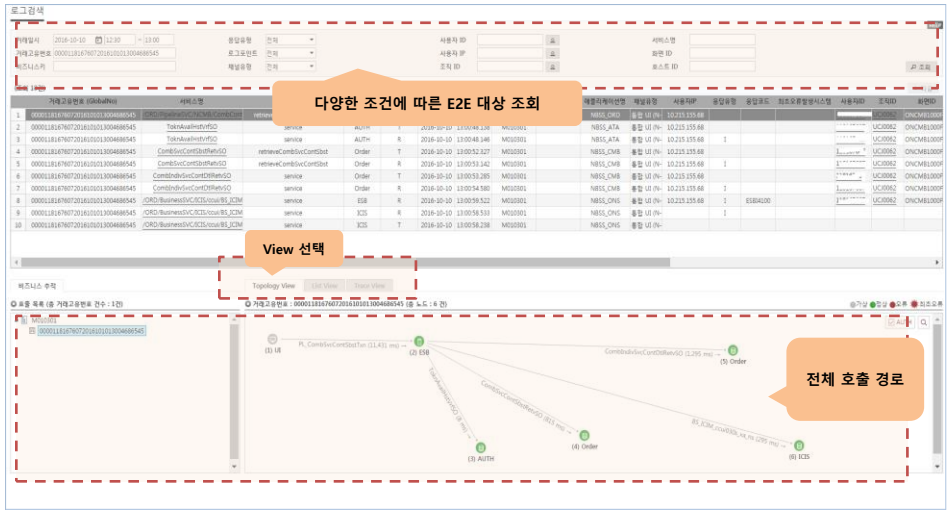
동적 영역 시각화 1

실시간 거래 모니터링 Dashboard – 표시 항목



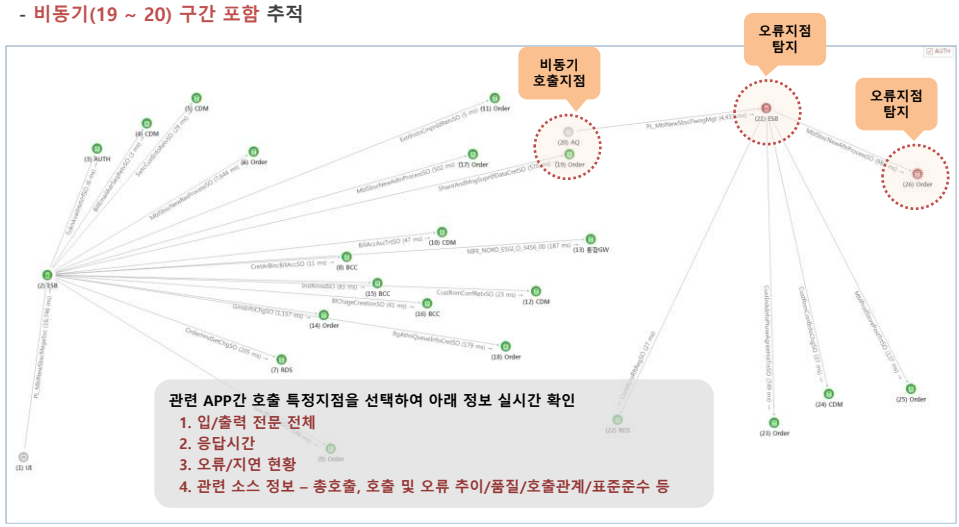
동적 영역 시각화 2

실시간 거래 모니터링 Dashboard – 실시간 구간 선택 혹은 조회 → 특정 거래에 대한 Topology View
- UI → G/W → ESB → Core Service → 대내/외 호출 전 구간에 대한 추적 뷰



동적 영역 시각화 3

실시간 거래 모니터링 Dashboard – 실시간 구간 선택 → 특정 거래에 대한 Topology View (상세)
- UI → G/W → ESB → Core Service → 대내/외 호출 전 구간에 대한 추적 뷰
- 비동기(19 ~ 20) 구간 포함 추적



질문과 답변

Q & A

성심껏 답변 드리겠습니다.

감사합니다

http://www.t3q.com

코드 수정 단위의 소프트웨어 결함 예측을 위한 정규화 기법 비교 분석

정주상 ○ 백종문

한국과학기술원 전산학부

대전광역시 유성구 대학로 291

jsjjung@kaist.ac.kr, jbaik@kaist.ac.kr

A Comparative Analysis of Normalization Techniques For Change Level Software Defect Prediction

Jusang Jung○ Jongmoon Baik

KAIST, School of Computing

291, Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea

1. 서 론

소프트웨어 결함 예측은 품질 보증 활동 중 하나로 소프트웨어 구성 요소에 결함이 있을 확률을 예측하는 기법이다. 이는 시간과 개발 인력, 비용 등의 제약 사항이 존재하는 실제 개발 환경에서, 모든 소프트웨어 구성 요소를 테스트하거나 코드를 검토하는 것이 실질적으로 불가능에 가깝기 때문에, 결함이 존재할 확률이 높은 구성 요소를 선택하여 테스트 순서 등의 개발 자원을 할당하는 과정에 이용된다.

소프트웨어 결함 예측은 소프트웨어 구성 요소에 대한 정의에 따라 모듈 단위의 결함 예측과 코드 수정 단위의 결함 예측으로 나눌 수 있다. 모듈 단위의 결함 예측은 구성 요소를 소스 코드 패키지, 코드 파일 등으로 정의하고, 각 소스 코드 구성 요소 안에 결함이 존재할 확률을 예측한다. 결함 예측을 위한 척도와 소스 코드 구성 요소 단위에 따라 패키지, 파일, 함수 단위 등 다양한 연구들이[1, 2] 소개되었다.

반면에 코드 수정 단위의 결함 예측은 오늘날 대부분의 개발 조직이 소프트웨어 형상 관리 도구를 이용하여 소프트웨어를 개발하는 현상을 따라 구성 요소를 각 코드 수정으로 정의하고, 각 코드 수정 안에 결함이 존재할 확률을 예측한다. 코드 수정 단위 결함 예측은 실제 소프트웨어 개발 과정을 잘 나타내고, 형상 관리 도구를 통한 자동화와 코드 리뷰 대상의 규모가 모듈 단위의 결함 예측보다 작다는 등의 장점을 가지고 있으며, 최근 많은 연구들이[3, 4, 5, 6] 소개되고 있다. 하지만 기존 연구들 중에는 정규화 기법에 대해 충분한 설명없이 사용되거나, 정규화가 생략된 연구들이[4, 5] 존재한다.

데이터 정규화는 데이터 분석 연구 분야의 기본 과정에 포함되는 과정으로 정규화 기법에 따라 예측 모델의 성능에 영향을 줄 수 있다. 따라서 데이터의 특성과 적용 도메인을 고려하여 정규화 기법을 선택하는 것이 필요하다. 코드 수정 단위 결함 예측에서 정규화 기법의 활용에 대한 문제를 해결하기 위해 본 논문은 코드 수정 단위 결함 예측에서 사용되고 있는 정규화 기법인 로그 변환 (Logarithm Transformation)과 최소-최대 정규화 (Min-Max Normalization) 기법을 적용하여 결함 예측 모델을 생성하고, 오픈소스 프로젝트를 대상으로 각 모델의 예측 성능 비교를 수행한다. 기존 정규화 기법들의 성능 비교를 통해 각 정규화 기법의 특징과 코드 수정 단위 결함 예측 분야에서 적합한 정규화 기법에 대해 논의한다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구에 대해 소개하고, 3 장에서는 정규화 기법에 대해 설명한다. 4 장에서는 실험 설계와 결과를 분석하고, 5 장에서는 결론에 대해 논의한다.

2. 관련 연구

본 장에서는 코드 수정 단위 결함 예측과 각 연구별 정규화 기법에 대한 관련 연구를 소개한다.

Kim [5] 은 12개의 오픈 소스 프로젝트를 대상으로 코드 수정 단위 결함 예측을 수행하였다. 하지만 정규화 기법을 적용하지 않았고, 사용한 척도, 분류기, 코드 수정에서 결함을 확인하는 기법 등에서 본 연구와 차이점이 존재한다.

Shihab [4] 은 대규모 상업용 모바일 소프트웨어를 대상으로 코드 수정 단위의 결함 예측을 수행하였다.

하지만 단일 프로젝트에 대한 사례 연구로 진행되었기 때문에 여러 프로젝트를 대상으로 일반화를 할 수 없다는 한계점을 지니고 있고, 정규화 기법을 적용하지 않았다는 차이점이 존재한다.

Yang [6] 은 6개의 오픈소스 프로젝트를 대상으로 코드 수정 단위의 결함 예측을 수행하였다. 결함 예측 모델을 만드는 과정에서 딥러닝 (Deep Learning) 알고리즘을 사용하여 모델을 생성하였으며, 정규화 기법으로는 최소-최대 정규화를 사용하였다.

Kamei [3] 는 6개의 오픈소스 프로젝트와 5개의 상업용 프로젝트를 대상으로 코드 수정 단위의 결함 예측을 수행하였다. 해당 논문에서는 코드 수정 단위의 결함 예측 연구들을 집대성하여 Just-In-Time Defect Prediction 이라는 이름으로 정리하였다. 본 논문과 가장 유사한 관련 연구이며, 정규화 기법으로는 로그 변환을 사용하였다.

3. 데이터 정규화 기법

이번 장에서는 데이터 정규화 기법에 대해 설명한다. 데이터 정규화 (Data Normalization)는 수집된 원시 데이터가 예측 모델을 생성하기에 적합하지 않은 형태를 띄고 있는 경우, 예측 모델의 성능에 영향을 미치기 때문에 이를 보완하는 기법으로 다양한 기법이 아래와 같이 존재한다.

3.1 Z-Score 정규화

Z-Score 정규화 기법은 표준 점수라고도 알려져 있으며, 각 데이터가 표준 편차 상에서 어느 위치를 차지하는가를 보여주는 수치이다. 표준값 $\mathrm{z} = \frac{x-\mu}{\sigma}$ 으로 나타낼 수 있으며, 이 때 x 는 원수치, μ 는 모집단의 평균, σ 는 모집단의 표준 편차를 나타낸다.

하지만, 코드 수정 단위 결함 예측에서 수집되는 척도의 원시 데이터들은 그림 1에서와 같이 편향되어 분포하는 경우가 많다. 즉, 표준화를 위한 가정인 정규 분포를 만족시키지 못하기 때문에 코드 수정 단위 결함 예측 연구에서는 Z-Score를 이용한 정규화를 적용할 수 없다. 따라서 본 연구에서는 Z-Score 기법을 제외한다.

3.2 로그 변환 (Log Transformation)

데이터 편향 (Data Skewness) 문제는 그림 1과 같이 코드 수정 척도 데이터가 값의 범위가 높거나 낮은 영역에 다수의 데이터가 분포하는 것을 의미한다. Kamei [3], Shihab [7]을 비롯한 코드 수정 단위 결함 예측 연구에서는 로그 변환 필터링 기법을 적용하여 데이터 편향 문제를 완화하였다. 그림 1의 원시 데이터에 로그 변환 정규화 기법을 적용한 결과는 그림 2와 같이 정규 분포에 가까운 형태를 나타내고 있다.

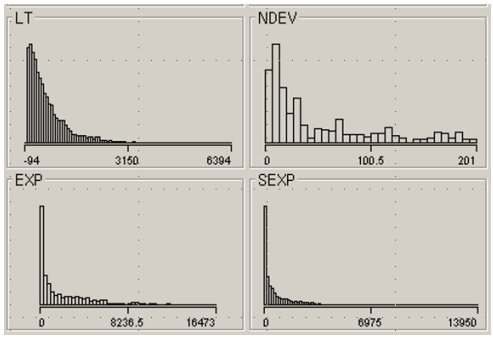


그림 1 코드 수정 단위 결함 예측 척도 원시 데이터 분포

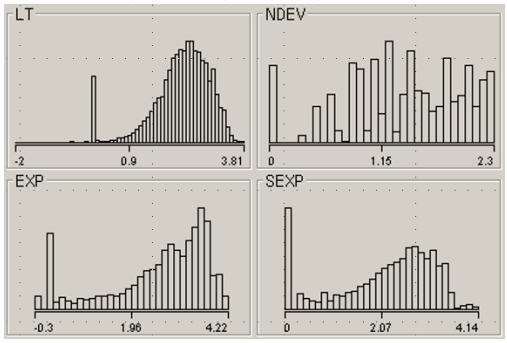


그림 2 로그 변환 정규화 기법 결과

3.3 최소-최대 정규화 (Min-Max Normalization)

최소-최대 정규화 기법은 Feature Scaling의 가장 간단한 방법 중 하나로 각 척도의 값을 특정 범위로 표현할 수 있다는 장점을 갖는다. 특정 범위로 표현되는 값 $x' = \frac{x-\min(x)}{\max(x)-\min(x)}$ 으로 정의된다. Min 과 Max 는 각 척도의 최대, 최소값을 나타내고, x' 은 0 에서 1 사이의 값으로 표현된다.

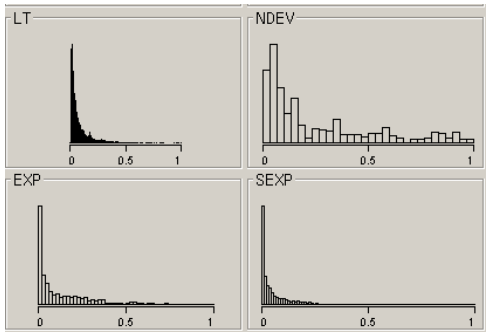


그림 3 최소-최대 정규화 기법

최소-최대 정규화 기법은 대표적으로 뉴럴 네트워크 (Neural Network) 모델 학습에서 요구하는 입력값의 범위를 만족시키는 과정에서 사용되며, 적용 결과는 그림 3과 같다. 그림에서 데이터의 영역은 0-1 사이로 감소하였지만, 데이터 분포는 원시 데이터와 동일하게 분포한다.

4. 실험

이번 장에서는 정규화 기법에 따른 코드 수정 단위 결함 예측 모델의 성능을 평가하기 위해 수행한 실험을 설명한다. 이어지는 절에서는 실험 설계, 실험 결과와 결과에 대한 분석을 수행한다.

4.1 실험 설계

본 연구의 실험에 사용한 데이터셋은 Github로부터 선정된 4개의 오픈 소스 프로젝트로 표 1과 같다.

표 1 실험 데이터셋 오픈소스 프로젝트 요약

Project Name	Repository Created Date	# of Commit	Language	# of defect change (%)
Tensorflow	2015.11.01	11120	Python	1604 (15.37)
Docker	2013.01.3	29145	go	4067 (13.95)
Socketio	2011.05.01	1605	Java Script	160 (9.96)
Django	2005.07.10	23458	Python	9979 (42.54)

본 실험에서 사용한 데이터셋은 개발언어, 코드 저장소 규모, 프로젝트 진행 기간 등에 대하여 다양한 프로젝트를 선정하여 정규화 기법에 따른 일반적인 특성을 확인하고자 하였다. 코드 수정 단위 결함 예측 연구에 따라 사용한 데이터셋이 다르기 때문에 새로운 오픈소스 프로젝트를 선정하고, 각 기법을 기반으로 예측 모델을 생성하여 성능 비교를 수행하였다.

본 실험은 정규화 기법에 따른 코드 수정 단위 결함 예측 모델의 성능 차이를 비교 분석하는 것을 목표로 한다. 예측 모델은 Kamei [3] 연구에서 소개된 코드 수정 단위 결함 예측 모델을 기반으로 재구성하였다. 정규화 기법은 3장에서 소개한 바와 같이 로그 변환과 최소-최대 정규화 기법을 이용하여 각각 원시 데이터를 필터링한 예측 모델을 생성하였고, 동일한 오픈소스 프로젝트를 대상으로 예측 성능을 측정하였다.

성능 평가의 지표로는 정밀도 (Precision), 재현도 (Recall), F1 점수, AUC (Area Under Curve)를 사용하였다. 정밀도는 결함으로 식별한 코드 수정 중 실제 결함인 것의 비율을 나타내고, 재현도는 전체 결함인 코드 수정 중 찾아낸 결함 코드 수정의 비율을 나타낸다. F1 점수는 두 지표를 하나의 수식으로 표현한 것이다. AUC 는 의학계에서 진단 검진의 성능을 평가하는 지표로서, 분류의 결과를 결정하는 기준값 (Threshold Value)을 변경하면서 예측 모델의 분류 성능을 평가한다.

성능 평가의 검증은 교차 검증 (Cross Validation) 을 수행하였다. 전체 데이터셋을 10개로 나누고, 그 중 9개를 학습 데이터 (Training Data)로 모델을 생성하고, 1개를 평가 데이터 (Test Data)로 모델의 성능을 측정하였다. 10회 라운드의 교차 검증을 5번 반복하여 데이터셋 당 50개의 결과를 도출하였다.

성능 평가의 통계적 분석은 Student t-test 와 Cliff Delta 효과 크기 (Effect Size) 분석을 수행하였다. Student t-test 는 두 집단의 평균이 통계적인 관점에서 유의미한지를 확인하는 시험으로 p-value 가 0.05보다 작을 때 두 집단의 평균이 다르다는 것을 의미한다. 효과 크기 (Effect Size) 분석은 두 집단의 얼마나 더 우세한지를 나타낸다. 본 연구에서 수행한 Cliff Delta의 경우 두 집단의 결과값을 짝비교를 통해 한 집단이 다른 집단보다 값이 큰 경우와 작은 경우를 기반으로 계산한다. Cliff Delta 값에 대한 해석은 아래 표와 같다.

표 2 Cliff Delta 효과 크기 분석의 해석

Cliff Delta Value	Interpretation
<0.147	Negligible
0.147 ~ 0.33	Small
0.33 ~ 0.474	Medium
0.474 ~ 1	Large

4.2 실험 결과

실험 결과의 분포는 그림 4와 같다. 그림에서 L 은 로그 변환을 적용한 예측 모델, MM 은 최소-최대 정규화를 적용한 예측 모델의 분포를 나타낸다. 예측 모델의 성능은 프로젝트에 의존적이므로, 본 연구에서는 두 정규화 기법의 예측 성능에 대한 상대적인 비교를 수행하였다. 그림 4의 분포에서는 Socketio, Tensorflow, Docker 3개의 프로젝트에서 로그 변환을 적용한 예측 모델의 성능이 더 높게 측정된 것으로 나타난다.

실험 결과의 통계적 유의미성을 분석한 결과는 표3과 같다. 그림 4의 분포도에서 명확한 성능의 차이를

L : Logarithm Transformation, MM : Min-Max Normalization

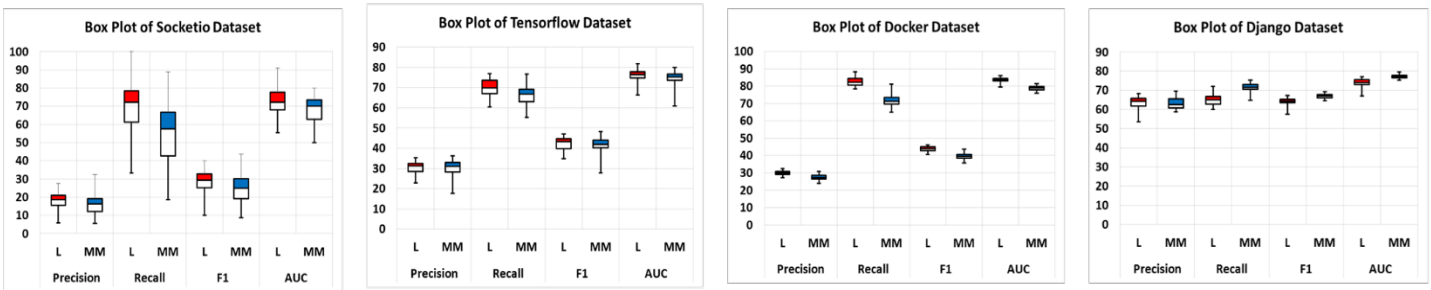


그림 4 실험 결과 Box Plot

표 3 실험 결과 분석

- >Medium
- < Medium
- Small
- >Medium (Neg)

	Socketio				Tensorflow				Docker				Django			
	Prec	Recall	F1	AUC	Prec	Recall	F1	AUC	Prec	Recall	F1	AUC	Prec	Recall	F1	AUC
Avg (Log)	18.27	69.62	28.66	71.89	30.47	69.96	42.37	76.12	29.89	82.52	43.86	83.54	63.46	64.96	64.13	73.94
Avg (Min-Max)	16.24	55.37	24.82	68.17	30.32	66.33	41.39	74.85	27.35	71.81	39.59	78.83	63.18	71.39	66.95	77.36
T-test p-value	5.3E-02	4.2E-06	8.2E-03	1.4E-02	8.3E-01	1.4E-04	1.8E-01	5.1E-02	3.8E-14	1.8E-32	1.1E-22	1.2E-32	6.4E-01	1.3E-24	2.5E-14	7.0E-18
Cliff Delta	0.2516	0.5068	0.328	0.2788	-0.0268	0.424	0.1448	0.2344	0.78	0.9768	0.9328	0.9816	0.092	-0.9184	-0.8328	-0.9184

나타낸 Socketio, docker 프로젝트에서는 모든 지표에 대해 p-value 가 0.05보다 작게 측정되어 통계적으로 두 집단의 평균이 다르다는 것이 확인되었다.

Tensorflow 프로젝트의 경우 비슷한 분포를 보인 지표에서는 p-value 가 0.05 보다 크게 측정되어 통계적 유의미성을 발견하지 못하였다. Cliff Delta 또한 Socketio, Tensorflow, Docker 프로젝트에서는 일부 지표를 제외한 대부분의 지표에서 로그 변환 기법이 최소-최대 정규화 기법보다 좋은 성능을 나타냈다.

반면에 Django 프로젝트에서는 최소-최대 정규화 기법을 적용했을 때 더 높은 예측 성능을 얻을 수 있었다. 정밀도를 제외한 성능 지표에서 t-test의 p-value 가 0.05보다 낮게 측정 되었고, Cliff Delta 에서도 차이를 확인할 수 있었다.

4.3 실험 결과 토의

본 연구의 실험에서 Django 프로젝트를 제외한 데이터셋에서 로그 변환 기법이 더 좋은 성능을 얻은 이유는 데이터 편향 (Data Skewness) 인 것으로 보인다. 즉, 로그 변환 기법이 최소-최대 기법보다 데이터 편향 문제를 완화시키는데 더 적합하기 때문에 우월한 코드 수정 결함 예측 모델이 생성된 것으로 판단된다. 최소-최대 정규화 기법은 그림 3에 나타난 것과 같이 값의 범위는 0-1 사이로 조정되었지만, 데이터 분포 자체가 달라지지 않기 때문에 데이터 편향 문제로 인해 예측 모델의 성능이 차이가 나는 것으로 보인다.

Django 프로젝트에서 상반된 결과가 얻어진 것에 대한 원인은 데이터셋과 성능 지표에 따라 코드 수정 단위 결함 예측 모델에 영향을 줄 수 있는 변수가 다양하기 때문에 일반화된 하나의 결론을 주장하기는 어렵다. 하지만, Django 프로젝트는 전체 코드 수정 중 결함이 발견된 코드 수정이 42.54 %로 많은 부분을 차지하고 있다. 이러한 차이가 Django 프로젝트에서 반대되는 결과가 얻어진 원인 중 하나일 수 있다.

5. 결론

본 논문에서는 코드 수정 단위 결함 예측 연구에서 사용되는 데이터 정규화 기법 중 로그 변환과 최소-최대 정규화 기법을 각각 적용한 결함 예측 모델을 생성하고, 오픈소스 프로젝트를 대상으로 성능 평가를 비교 분석하였다. 실험 결과 코드 수정 데이터의 왜곡 문제를 해결할 수 있는 로그 변환 정규화 기법이 4개 중 3개의 오픈소스 프로젝트에서 통계적으로 우월한 예측 성능이 나타났다. 따라서 코드 수정 단위 결함 예측 연구에서는 로그 변환 기법이 더 적합한 것으로 보인다.

본 연구의 추후 연구 방향으로는 추가적인 오픈소스 프로젝트를 대상으로 실험을 통한 일반화와 코드 수정 단위 결함 예측을 위한 새로운 척도를 제시하는 것이다.

참고문헌

[1] Yu, T-J., Vincent Yun Shen, and Hubert E. Dunsmore. "An analysis of several software defect models." IEEE Transactions on Software Engineering 14.9 (1988): 1261-1270.

[2] Moser, Raimund, Witold Pedrycz, and Giancarlo Succi. "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction." 2008 ACM/IEEE 30th International Conference on Software Engineering. IEEE, 2008.

[3] Kamei, Yasutaka, et al. "A large-scale empirical study of just-in-time quality assurance." IEEE Transactions on Software Engineering 39.6 (2013): 757-773

[4] Shihab, Emad, et al. "An industrial study on the risk of software changes." Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 2012.

[5] Kim, Sunghun, E. James Whitehead Jr, and Yi Zhang. "Classifying software changes: Clean or buggy?." IEEE Transactions on Software Engineering 34.2 (2008): 181-196.

[6] Yang, Xinli, et al. "Deep Learning for Just-In-Time Defect Prediction." Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on. IEEE, 2015.

[7] Shihab, Emad, et al. "Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project." Proceedings of the 2010 ACM-IEEE Int'l Symp on Empirical Software Engineering and Measurement. ACM, 2010.

Usability와 Security의 트레이드 오프 제거를 위한 품질 속성 요소 연결 지침

노우리 $_1$ ○ 이석원 $_2$

아주대학교 컴퓨터공학과$_1$ 아주대학교 소프트웨어학과$_2$
yuwooloe@ajou.ac.kr$_1$, leesw@ajou.ac.kr$_2$

The Guidelines for Linking Quality Factors of Usability and Security to Reduce Trade-offs

Woori Roh$_1$ ○ Seok-Won Lee$_2$

Ajou University Dept. of Computer Engineering$_1$ Ajou University Dept. of Software$_2$

요 약

본 연구는 소프트웨어 개발 초기 과정에서 Usability와 Security 사이의 발생하는 트레이드 오프를 미리 예측할 수 있도록 도와주는 품질 속성 요소 연결 지침을 제안한다. 9개의 Usability 품질 속성 요소, 9개의 Security 품질 속성 요소 그리고 15개의 Criteria를 사용하여 총 4 단계의 과정을 통해 도출된 품질 속성 요소 연결 지침은 관련 있는 요소들을 보여주고 트레이드 오프 없이 Usability와 Security 향상에 도움을 주는 지침을 제공한다.

1. 연구 배경 및 문제 정의

소프트웨어 개발 과정에서 Usability에 대한 중요성이 높아지고 있다. Usability는 단순히 사용자가 임무를 수행하는 과정에서 속도나 정확도를 높이는 것뿐만 아니라 사용자의 안전과 소프트웨어의 보안에도 관련이 있기 때문이다.

Usability와 Security의 트레이드 오프에 대한 연구가 다양하게 진행되어 오고 있지만 기본적으로 Usability와 Security가 서로 상충되는 특성을 가지고 있기 때문에 많은 개발자와 소프트웨어 공학자들이 두 품질 속성 요구사항이 분리되어 있다고 생각한다. Usability를 높이기 위해 대표적으로 고려되는 내용은 간단하고 단순한 진행 절차와 빠른 속도인 반면, Security를 높이기 위해 대표적으로 고려되는 속성으로는 모방 방지를 위한 복잡성이기 때문이다. 기존의 연구들은 트레이드 오프를 해결하기 위해 어느 한 쪽의 기능을 향상시키기 위한 기술적인 내용에 초점을 맞춰왔다. 하지만 Usability는 의도하지 않은 오류를 최소화하는데 목적을 두고 Security는 바람직하지 않은 행동을 막거나 완화시키는 것을 목적으로 두기 때문에 [1] 두 개의 품질 속성 요구사항은 같은 선상에서 다뤄져야 한다.

본 논문에서는 두 품질 속성 요구사항 측정에 사용되는 품질 속성 요소와 Criteria를 세부적인 단위와 요소로 구체화하여 그들 사이의 관계를 규명하고 관계들의 속성을 통하여 트레이드 오프를 감소시킬 수 있는 Criteria를 선정한다. 이를 통해 트레이드 오프 없이 두 품질 속성 요구사항을 향상시키는 지침을 제안할 수 있다.

2. 관련 연구 및 배경 지식

본 절에서는 기존에 사용되고 있는 Usability와 Security Metrics에 대한 조사를 바탕으로 두 관점의 Metrics에서 각각 중요하게 고려되는 품질 속성 요소들을 정리하고 선정하여 다음 연구에 기반이 될 개념들을 소개한다. Metrics는 Usability와 Security의 측정 가능한 척도를 명확하고 구체적이게 제공한다 [2]. 소프트웨어의 질을 높이기 위해서 측정 가능성의 여부를 판단하는 것은 중요하기 때문에 [3] Metrics과 관련된 품질 속성 요소를 수집하였다.

2.1 Usability 품질 속성 요소

본 절은 Usability를 측정하기 위하여 사용되는 품질 속성 요소와 Criteria에 대한 공신력 있는 자료를 활용하여 국제 표준 기구(ISO)에서 사용하는 Usability 정의 [4] [5]와 [6]의 연구에서 설명하는 내용을 토대로 구성한다. 대표적인 Usability 품질 속성 요소를 표 1에 정리했다.

표 1 Usability 품질 속성 요소

품질 속성 요소	정의
Efficiency	사용자가 목표를 달성하기 위해 특정 환경에서 적절한 양의 자원을 소비할 수 있게 하는 기능
Effectiveness	사용자가 정확하고 완전하게 구체적인 목표를 수행하게 하는 기능
Satisfaction	사용자의 주관적인 관점에서 사용의 편안함과 수용성을 의미
Productivity	사용자와 시스템이 소비하는 자원과 관련하여 성취된 효율성의 수준
Learnability	특정 목표를 달성하는데 필요한 기능을 얼마나 쉽게 습득할 수 있는지에 대한 기준
Safety	소프트웨어가 사람 또는 다른 자원 및 정보에 해를 끼칠 위험을 제한하는지에 대한 여부
Accessibility	장애가 있는 사람이 소프트웨어를 사용할 수 있는지에 대한 여부
Universality	다른 문화적 배경을 가진 사용자의 다양성을 수용하는지에 대한 여부
Usefulness	소프트웨어가 실제 문제를 수용 가능한 방식으로 해결할 수 있는지에 대한 여부

2.2 Security 품질 속성 요소

Security Metrics는 악의적인 공격으로 인한 피해 또는 손실 가능성을 수치화 하는 데에 초점을 두고 있다 [7]. 품질 속성 요소를 추출하기 위하여 Security Metric에 대한 논문 [3] [2] [8]과 Security에 대한 정의를 다룬 논문 [9]과 책 [10]을 참고하였다.

표 2 Security 품질 속성 요소

품질 속성 요소	정의
Confidentiality	승인되지 않은 정보 공개 방지
Integrity	부적절한 시스템 변경 방지
Availability	요청 시 서비스를 제공하는 기능
Safety	사용자와 환경에 치명적인 결과 방지
Correctness	시스템 서비스가 지정된 대로 보장
Efficiency	소프트웨어가 메모리 및 프로세서 사이클과 같은 시스템 자원을 사용하는 기준
Reproducibility	하나 이상의 오류를 일으킨 문제의 활성화 패턴을 식별하는 기능
Scalability	모든 사용자에게 전달되는 전반적인 서비스 품질을 저하시키지 않으면서 증가하는 사용자 수를 처리할 수 있는 능력
Accountability	작업을 수행한 사람의 신원 정보 가용성 및 무결성

2.3 Usability와 Security의 Criteria

[6]은 Usability 품질 속성 요소를 측정 가능한 25개의 Criteria로 세분화하였고 [11]은 이를 Security 문제와 연결하여 Usability와 Security의 트레이드 오프를 제거에 초점을 맞췄다.

[8]의 저자는 Security 전문가를 상대로 한 설문과 인터뷰를 통해 19가지 주요 품질 Criteria을 소개하고 그들 사이의 관계를 도출했다. 저자는 19가지의 Criteria 중에서 Meaningfulness, Correctness, Measurability, 그리고 Usability 4개의 중심 Criteria를 선정하였다. 여기서 의미하는 Usability는 Security를 위한 하나의 품질 Criteria 이지만 그 의미는 본 논문에서 정의한 Usability와 다르지 않다.

이를 통해 Sustainability for the task, Resource utilization, Resource safety, Time behavior, Minimal action, Information security, Operability, Error management, Consistency, Self-descriptiveness, Insurance, User-friendly, Authentication, Accuracy, 그리고 Controllability 총 15개의 Criteria를 선정하였다.

3. 제안 접근법

본 연구에서는 앞 장에서 소개한 관련 연구들의 내용을 참고하여 표 1과 표 2에서 정리한 Usability와 Security 관점의 품질 속성 요소를 세부적인 단위와 다양한 요소들로 구체화한다. 그리고 두 품질 속성 요구사항의 Criteria 간의 관계를 규명하고 그 관계들의 속성을 통하여 서로를 향상시킬 수 있는 중요한 품질 속성 요소와 Criteria를 추천함으로써 트레이드 오프 제거를 위한 지침을 제공한다.

3.1 4단계 과정

제안하는 접근법에 대한 과정은 표 3에 정리되어 있다. 총 4단계로 나누어 진행된다.

표 3 제안하는 4단계 접근법 과정

단계 1	Usability와 Security에서 중요하게 다뤄지는 품질 속성 요소와 관련 Criteria 선정
단계 2	각 품질 속성 요소의 정의와 Metrics를 통해 관련된 Criteria로 세분화
단계 3	세분화 된 Criteria를 그 속성에 따라 {make, help, hurt, break}의 4가지로 서로 연결
단계 4	연결된 관계를 통해 트레이드 오프 없이 Usability와 Security를 향상시킬 수 있는 품질 속성 요소 추천

단계 1은 Usability와 Security의 품질 속성 요소와 Criteria를 선정하고 규명하는 단계이다. 관련 연구, 관련 도서 그리고 표준 등을 통해 품질 속성 요소와 Criteria의 정의를 수집하고 그 안에서 핵심 키워드를 추출한다.

단계 2는 각 품질 속성 요소를 Criteria로 세분화하는 과정이다. 관련 연구 및 Metrics를 참고하여 품질 속성 요소를 더욱 세부적인 단위와 다양한 요소들로 구체화한다.

단계 3에서는 세분화되고 분석된 각 품질 속성 요소들이 그 관계의 특징에 따라 i* 프레임워크의 {make, help, hurt, break} 4가지 종류로 연결된다. 4개의 요소들은 각각 'make'는 '++', 'help'는 '+', 'hurt'는 '-', 그리고 'break'는 '--'로 표시된다.

단계 4에서는 단계 3에서 정리된 품질 속성 요소들과 Criteria의 관계를 통해 두 품질 속성 요구사항을 트레이드 오프 없이 향상시킬 수 있는 품질 속성 요소의 묶음을 도출한다.

3.2 품질 속성 요소 연결 초기 지침

Usability 품질 속성 요소 9개와 Security 품질 속성 요소 9개를 15개의 Criteria를 사용하여 3.1에서 소개한 4단계 중 단계 3까지 진행한 결과는 표 4와 같다.

18개의 Usability와 Security의 품질 속성 요소와 15개의 Criteria는 단계 1에서 선정 및 도출되었다.

단계 2와 단계 3에서 품질 속성 요소와 Criteria의 관계가 정의되었다. 예를 들어, Usability 품질 속성 요소인 Efficiency는 그 정의에 따라 사용자의 목표 달성 의지와 자원 활용을 포함하고 Metrics에 따라 시간, 단순한 동작, 작동성, 그리고 피드백과 연관이 있음을 확인할 수 있다. Usability 품질 속성 요소인 Effectiveness는 '정확하고 완전하게 구체적인 목표를 수행'의 정의와 사용자가 수행한 전체 task의 수와 사용자가 성공적으로 수행한 task의 수를 사용해 계산하는 측정법에 따라 목표를 지속할 수 있음, 지속적임, 정확함 그리고 완전함과 긍정적인 관계에 놓인다.

단계 4에서는 단계 3을 통하여 도출된 표 4에서 각 행에 표시된 Criteria에 대하여 '+'의 관계로 표시된 품질 속성 요소를 찾음으로써 트레이드 오프 없이 Usability와 Security를 향상시킬 수 있는 품질 속성 요소들의 묶음을 도출할 수 있다. 예를 들면, 표 4의 네 번째 행에 있는 'Time behavior'에 긍정적 관계를 가지는 품질 속성 요소는 Efficiency(U), Productivity, Availability, 그리고 Efficiency(S)이므로 이 품질 속성 요소 묶음은 트레이드 오프

표 4 품질 속성 요소와 Criteria 연결

Criteria	Usability 품질 속성 요소									Security 품질 속성 요소								
	Efficiency	Effectiveness	Satisfaction	Productivity	Learnability	Safety	Accessibility	Universality	Usefulness	Confidentiality	Integrity	Availability	Safety	Correctness	Efficiency	Reproducibility	Scalability	Accountability
Sustainability for the task	++	++	+		+		+	+				+		+	+		++	
Resource utilization	++			++					++			++			++		+	++
Resource safety						++				++	++		++			+		++
Time behavior	++			+								+			++			
Minimal action	+		+	+	++		++											
Information security						++				++	++		++			+		++
Operability	++	+	+	++		+	+	+	++	+	+	++	+	++	++	+	++	
Error management			+			++			++		++		++			++		
Consistency		+			++	+	+	+		++	++		+	++		+	++	+
Self-descriptiveness					++	-	++	++		-	--		-					
Accuracy		++				++			++	++	++	++	++	++		++	++	
User-friendly			++		++	-	++	++		-	-		-					
Authentication						++				++	++		++					++
Insurance						++				++	++	+	++	++	+	++	+	++
Controllability					+		++	++	++		+	+		+		+	+	

관계 없이 Usability 와 Security를 향상시킬 수 있다. 반면 열두 번째 행의 'User-friendly'와 긍정적인 관계를 가지는 품질 속성 요소 Satisfaction, Learnability, Accessibility, 그리고 Universality와 부정적인 관계를 가지는 품질 속성 요소 Safety(U), Confidentiality, Integrity, 그리고 Security(S)의 묶음은 트레이드 오프를 발생시킬 수 있다.

3.3 향후 연구 방향

본 절에서는 제안하는 접근법의 향후 연구를 어떻게 진행할지 이야기 하고자 한다. 본 연구에서는 보편적인 Usability와 Security의 품질 속성 요소와 Criteria를 수집하였다. 본 연구의 결과를 더 정확하게 도출할 수 있도록 도메인을 지정하여 시스템의 특성을 고려하는 품질 속성 요소와 Criteria를 선정할 필요가 있다. 그리고 본 연구는 품질 속성 요소와 Criteria의 관계를 각각의 정의 및 Metrics를 통해 도출하였기에 {make, help} 관계는 쉽게 도출되었으나 {hurt, break} 관계는 도출하기 쉽지 않다는 단점이 있다. 이를 보완하기 위해서 관련 연구와 정의, 그리고 실제 Usability와 Security를 동시에 고려한 시스템의 특성을 분석한 내용을 추가하여 제기된 단점을 보완할 수 있다. 차후에 보완된 접근법을 통해 소프트웨어 개발 과정의 향상 정도를 측정할 수 있는 사례 연구가 진행되어야 한다.

4. 결론

본 연구는 소프트웨어 개발 초기 과정에서 Usability와 Security 사이의 트레이드 오프 발생을 유발하는 품질 속성 요소를 소개하는 품질 속성 요소 연결 지침을 제안하였다. 총 4단계의 과정을 통해 도출된 품질 속성 요소 연결 지침은 어떤 품질 속성 요소가 트레이드 오프를 발생시키는지 그리고 어떤 품질 속성 요소가 서로 비슷한 양상을 띄우는지 보여준다. 이 관계를 통하여 서로 긍정적 관계에 있는 품질 속성 요소들을 묶음으로써 Usability와 Security의 트레이드 오프 없이 두 품질 속성 요구사항을 향상시킬 수 있는 지침을 제시할 수 있다. 추후에는 품질 속성 요소들의 관계를 확장하기 위하여 3.3에서 제안한 내용을 바탕으로 본 연구를 보완할 예정이다.

Acknowledgement

이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(2013M3C4A7056233)

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW특성화대학원 지원 사업의 연구결과로 수행되었음"(R0346-15-1017)

참고 문헌

[1] R. Kainda, I. Flechais and A. W. Roscoe, "Security and Usability: Analysis and Evaluation," *5th International Conference on Availability, Reliability and Security*, 2010.

[2] A. Jaquith, Security Metrics: Replacing Fear, Uncertainty, and Doubt, Addison-Wesley Professional, 2007.

[3] D. Mellado, E. Fernandez-Medina and M. Piattini, "A Comparison of Software Design Security Metrics," in *Proceedings of the Fourth European Conference on Software Architecture*, Copenhagen, 2010.

[4] *Ergonomic requirements for office work with visual display terminals – Part 11: Guidance on usability*. International Standard ISO 9241-11:1998(en), 1998-03.

[5] *Systems and software engineering - Systems and software quality requirements and evaluation (SQuaRE) - Measurement of quality in use.* International Standard ISO/IEC 25022:2016, 2016-06.

[6] A. Seffah, et al., "Usability Measurement and Metrics: A Consolidated Model," *Software Quality Journal*, vol. 14, pp. 159-178, 2006.

[7] O. S. Saydjari, "Is Risk A Good Security Metric?," in *QoP '06 Proceedings of the 2nd ACM workshop on Quality of protection*, Alexandria, 2006.

[8] R. M. Savola, "Quality of Security Metrics and Measurements," *Computers and Security*, vol. 37, pp. 78-90, 2013.

[9] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11-33, 2004.

[10] I. Sommerville, Software Engineering 9th edition, Pearson, 2010.

[11] C. Braz, A. Seffah and D. M'Raihi, "Designing a Trade-Off Between Usability and Security: A Metrics Based-Model," *Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction*, Rio de Janeiro, 2007.

커널용 메모리 깨짐 검출기

: Memory Corruption Detector for Kernel

우충기*, 권진만**, 이승훈**, 이학봉**, 권재욱**

*㈜ 삼성전자, KAIST 소프트웨어대학원
**㈜ 삼성전자
chungki.woo@kaist.ac.kr, {jm0508.kwon, sh91.lee, hakbong5.lee, jook.kwon}@samsung.com

요약: 메모리 깨짐 문제는 원인파악이 힘들어 개발 기간을 지연시키는 큰 문제점들 중 하나 이다. 본 연구에서는 메모리 깨짐 이슈들을 유형별로 분류하고 특히 커널(Kernel)에서 발생하는 문제들에 대해 실시간으로 원인을 검출하는 방법을 제안하고 이를 도구(Tool)로 구현 후 현업에 적용하여 그 효과를 확인하였다.

핵심어: 메모리 깨짐 검출기, 커널, Instrumentation

1. 배경

1.1 메모리 깨짐 의미와 원인

메모리 깨짐 이란 유효하지 않은 메모리 영역을 쓰거나 읽어서 발생하는 문제점이다. 그림 1. 과 같이 메모리 깨짐의 원인은 크게 두 가지로 나눌 수 있다. DDR 메모리 Error 와 같은 하드웨어 적인 원인으로 인해 발생하는 것과 소프트웨어 즉, 프로그램 버그에 의해 발생하는 것으로 분류 할 수 있다. 본 연구에서는 소프트웨어에 의해 발생하는 문제들에 대해 다루었다. 소프트웨어에서 주로 발생하는 문제 유형 여섯 가지를 그림 1. 의 오른편에 나열하였다.

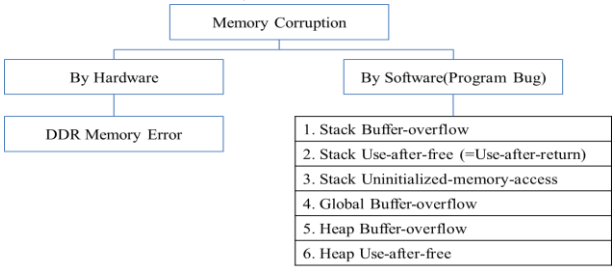


그림 1. 메모리 깨짐의 원인

1.2 Memory Corruption 문제 유형들

앞서 언급된 소프트웨어에 의해 발생하는 메모리 깨짐의 문제 유형 여섯 가지 들에 대한 예제 코드를 그림 2. 에 나열 하였다. 해당 문제 유형들은 크게는 Buffer-overflow, User-after-free, Uninitialized-memory-read 와 같이 세가지 유형으로도 분류 할 수 있고 문제가 발생하는 메모리 영역들에 따른 분류도 가능하다.

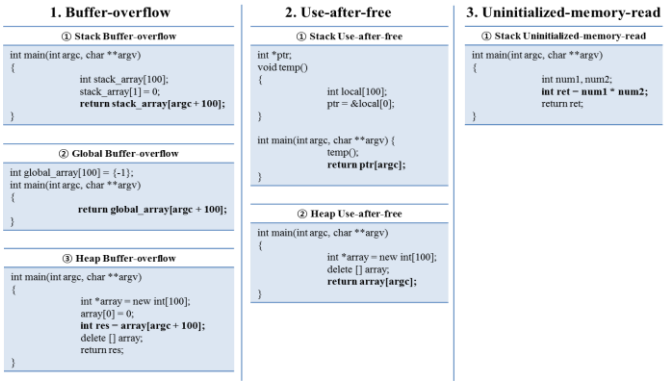


그림 2. 메모리 깨짐 문제 유형들

즉, 지역 변수 할당에 사용되는 스택(Stack) 영역 관련 문제, 전역 변수 와 정적 변수 할당에 사용되는 글로벌(Global) 데이터 영역 관련 문제, 동적으로 메모리 할당 시 사용되는 힙(Heap) 영역과 관련하여 발생하는 문제들로 분류 할 수 있다.

1.3 메모리 깨짐 문제의 심각성

이러한 메모리 깨짐 문제는 메모리가 깨지는 장소와 시점이 메모리 깨짐이 실제 원인이 되어 문제로 나타나는 장소, 시점과 달라 원인 파악이 굉장히 힘들다. 이로 인해 현업에서는 심각한 개발 기간 지연을 일으키고 있다. 그림 3. 과 같은 코드들의 경우 실행 시점에는 문제가 없으나 추후 어떤 문제를 일으킬 지 알 수 없는 코드들이다.

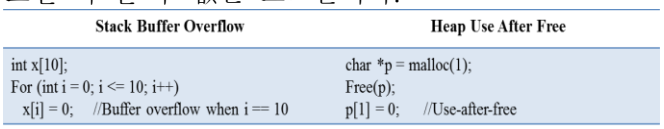


그림 3. 실행 시점에는 문제가 없는 코드들

1.4 커널 메모리 깨짐 이슈의 심각성

이런 커널 메모리 깨짐 이슈는 유저 메모리 깨짐 이슈 보다 훨씬 심각한 문제이다. 예를 들어 그림 5. 의 왼쪽과 같이 유저(User) 메모리 깨짐의 경우 만약 하나의 유저 프로세스에서 메모리 깨짐 발생시 가상 메모리 메커니즘에 의해 여파(Side Effect) 가 메모리 깨짐이 발생한 프로세스 하나로 고립(Isolation) 되게

된다. 하지만 커널 메모리깨짐 이슈의 경우 유저, 커널 가릴 것 없이 시스템 전체에 영향을 주게 되고 최악의 경우 마이크로 소프트사의 윈도우즈 시스템의 블루 스크린 문제와 같이 시스템을 더 이상 실행할 수 없는 상태에 빠뜨리게 한다.

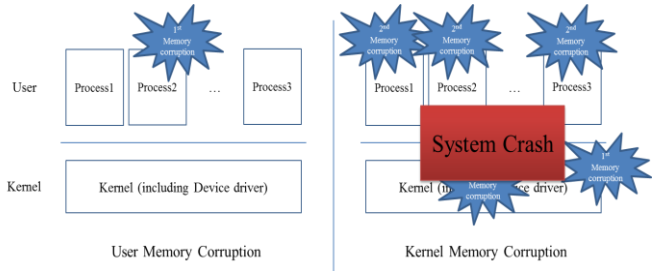


그림 5. 커널 메모리 깨짐 이슈의 심각성

1.5 현업 메모리 깨짐 이슈 발생 횟수

그림 6. 는 2014, 2015 년 제품 개발 시 실제로 발생한 커널 메모리 깨짐 문제들 22 건을 앞에서 언급한 소프트웨어에 의해 발생하는 메모리 깨짐 문제 유형 여섯 가지로 분류해보았다. 모든 유형의 문제가 골고루 발생하였으나 그 중 힙과 관련된 메모리 깨짐 문제가 전체 발생한 문제들의 71% 를 차지 하는 것을 확인 하였다.

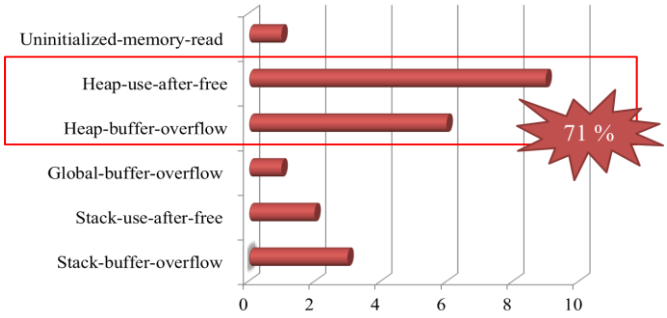


그림 6. 커널 메모리 깨짐 문제들 22 건

그림 7, 8. 은 2014 년, 2015 년 제품 양산시 실제로 발생했던 디바이스 드라이버 문제, 파일 시스템 문제들로써 2014 년에 발생했던 문제의 경우 해결기간에 1 주, 2015 년에 발생했던 문제는 해결기간에 2 주나 걸려 제품 개발 기간 지연으로 이어졌던 문제들이었다.

1. Mali device driver 에서 Descriptor table 을 함수인자인 init_entries 만큼 할당

```
mail_descriptor_mapping *mali_descriptor_mapping_create(int init_entries, int max_entries)
{
    map->table = descriptor_table_alloc(init_entries);
    if(NULL! = map->table) {
```

2. map->current_nr_mappings 에 할당된 entry 개수 저장

```
map->current_nr_mappings = init_entries;
```

3. mali_descriptor _mapping_allocate_mapping() 함수에서
old descriptor table 을 new descriptor table 로 복사 할 때 Heap Buffer Overflow 문제 발생

```
_mali_osk_errcode_t mali_descriptor_mapping_allocate_mapping(mali_descriptor_mapping * map, void * target, int
*odescriptor)
{
    ...
    map->current_nr_mappings += BITS_PER_LONG;
    ...
    mali_osk_memcpy(new_table->mappings, old_table->mappings, map->current_nr_mappings * sizeof(void*));
```

그림 7. 2014 년 디바이스 드라이버 문제

emmcfs_destroy_bnode() 함수 내에서 bnode 가 destroy 된 다음 다시 사용됨

```
int btree_remove(struct vdfs_btree *tree, struct emmcfs_generic_key *key,
        emmcfs_put_bnode(next_bnode);
{
...
    err = emmcfs_destroy_bnode(bnode);
    if(err)
        return err;

    EMMCFS_BUG_ON(bnode->node_id ==
            emmcfs_btree_get_root_id(tree));

    if(level!= emmcfs_btree_get_height(tree)) {
        err = btree_remove(tree, key, level + 1);
        if(err == -ENOENT)
...
}
```

그림 8. 2015 년 파일시스템에서 발생문제

2. 관련 연구 & 문제점

2.1 기존의 메모리 깨짐 검출기

이러한 메모리 깨짐 문제에 대한 원인을 쉽게 파악하고 해결 하기 위해 연구된 기존 도구들이 이미 존재하고 있으나(그림 9.) 문제는 이런 도구들의 특징들을 봤을 때 모두 유저 쪽 메모리 깨짐에 대해서만 대응 되는 것을 확인 할 수 있다.

본 연구에서는 이를 해결하기 위해 커널을 위한 메모리 깨짐 감지도구인 KMCD(Kernel Memory Corruption Detector) 즉, 커널용 메모리 깨짐 검출기를 제안하고 구현하였다.

도구	동작방식	특징
Valgrind[1]	Dynamic binary instrumentation	Support user-space only, Worst Performance, Much Used Memory
Purify[2]	Dynamic binary instrumentation	Support user-space only, Worst Performance, Much Used Memory
Insure++[3]	Compile-time instrumentation	Support user-space only, Low Performance, Much Used Memory
Address Sanitizer[4]	Compile-time instrumentation	Support user-space only, Low Performance, Much Used Memory

그림 9. 기존 메모리 깨짐 검출기들

3. 문제 해결 방법 제안

3.1 커널용 메모리 깨짐 검출기

커널용 메모리 깨짐 검출기는 아래 그림 10 과 같이 구성된다. 우선 Instrumentation 이란 기술을 통한 메모리 깨짐 발생 감지 기법을 사용하며 유저가 아닌 커널 지원을 목표로 하며 구현된 검출기가 임베디드(Embedded)라는 제한된 자원(Resources) 환경 하에서 제품의 무거운 소프트웨어와 함께 동작해야 하므로 제품의 무거운 소프트웨어 함께 동작해야 하므로 적은 메모리 사용량과 좋은 성능을 가지도록 하며 다양한 유형의 메모리 깨짐 문제를 감지 할 수 있어야 하는 요구 사항들을 만족 하도록 하였다.

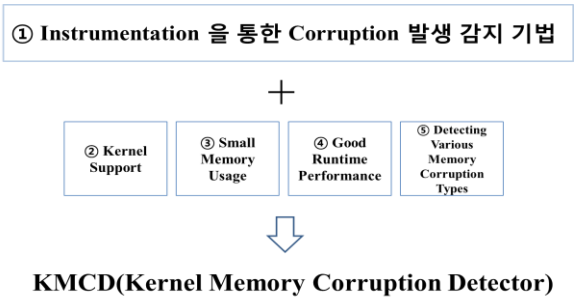


그림 10. 커널용 메모리 깨짐 검출기

3.2 Instrumentation 을 통한 메모리 깨짐발생 감지 기법

핵심이 되는 기술인 Instrumentation 을 통한 메모리 깨짐 감지 기법은 아래 그림과 같이 크게 두 가지 단계로 수행된다. 우선 메모리 할당 해제 시 할당 해제 정보를 별도의 메모리(Shadow Memory) 에 기록하게 하는 Poisoning 단계(Step 1) 와 메모리에 읽고 쓰는 순간 해당 메모리 주소에 대한 유효성을 체크하여 유효하지 않은 영역에 접근일 경우 알림을 주도록 하는 역할을 수행하는 Reference Validation 단계(Step 2)이다.

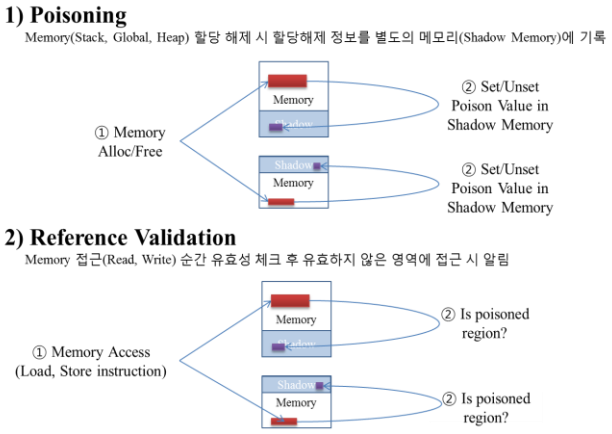


그림 11. Corruption 발생감지 기술

Poisoning 단계의 경우 상황에 따라 아래와 같은 Poison Value 들을 땅에 지뢰를 깔 듯 Shadow Memory 에 기록하게 된다. Reference Validation 단계에서는 Shadow Memory 에 기록된 Poison Value 를 보고 해당 메모리가 유효한지를 판단하게 되는 것이다.

```
/* Stack poison value */
#define KMCD_STACK_LEFT          0xF1
#define KMCD_STACK_MID           0xF2
#define KMCD_STACK_RIGHT         0xF3
#define KMCD_STACK_PARTIAL       0xF4
#define KMCD_FREE_STACK 0xF0

/* Global poison value */
#define KMCD_GLOBAL_REDZONE      0xF6

/* Heap poison value */
#define KMCD_SLAB_REDZONE        0xFD
#define KMCD_KMALLOC_REDZONE     0xFC
#define KMCD_KMALLOC_FREE        0xFB
#define KMCD_SLAB_FREE           0xFA
#define KMCD_SHADOW_GAP          0xF9
#define KMCD_VMALLOC_REDZONE     0xF8
#define KMCD_VMALLOC_FREE        0xF7
```

그림 12. Shadow memory 에 기록되는 Poison Value

※ Instrumentation 의 의미와 필요성

Instrumentation 이란 프로그램에 코드를 삽입하여 기존 프로그램의 동작을 변경 하는 것을 의미한다. 그림 13.과 같이 Instrumentation 방식은 Instrumentation 에 의해 삽입 되는 코드의 삽입 시점에 따른 분류 또는 삽입 방법에 따른 분류가 가능하다.

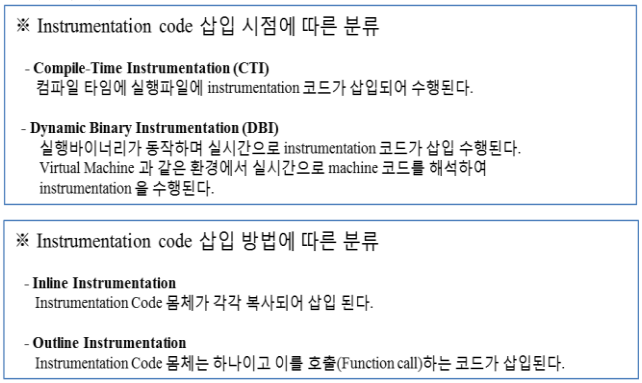


그림 13. Instrumentation 방식 분류

이러한 Instrumentation 기술이 반드시 필요한 이유는 메모리 깨짐이 발생하는 순간에 대한 즉시감지 (Prompt Detection) 가 가능하기 때문이다.

3.3 Poisoning 단계

Poisoning 단계의 경우 Instrumentation 과정을 거치면 그림 14. 와 같이 메모리 할당 이후 해당 메모리 영역과 매핑이 되는 Shadow Memory 영역에 Poison Value 를 기록 하는 코드가 삽입된다. Poison Value 는 3.1.1 에 언급되었다시피 어떤 메모리 영역과 관련된 메모리 깨짐 인지를 내부적으로 판단하기 위해 별도로 만든 값이며 할당 해제 되는 메모리 종류에 맞는 값을 사용하게 된다.

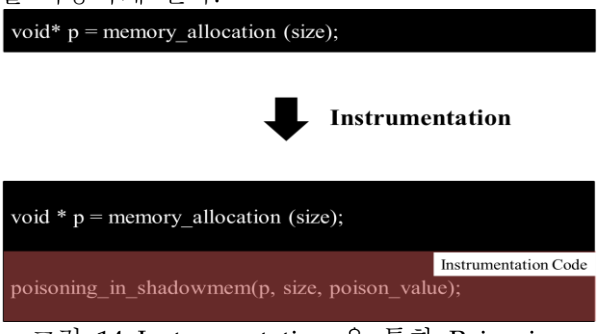


그림 14. Instrumentation 을 통한 Poisoning

3.4 Reference Validation 단계

Reference Validation 단계의 경우 Instrumentation 과정을 거치면 그림 15 와 같이 메모리 접근(읽기, 쓰기) 이전 해당 메모리 영역과 매핑 되는 Shadow Memory 영역에 기록되어 있는 Poison Value 를 보고 해당 영역이 유효 한지를 우선 판단하고 유효 하지 않을 경우 사용자에게 해당 정보를 리포팅 해주는 코드가 삽입이 된다.

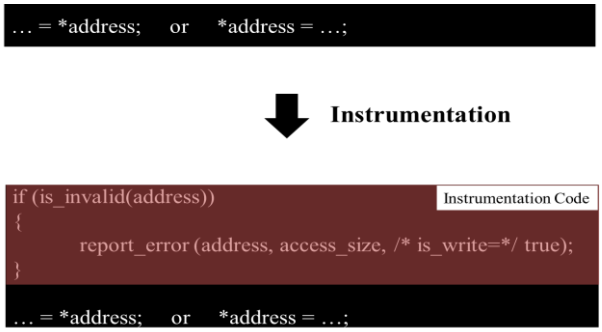


그림 15. Instrumentation 통한 Reference Validation

3.5 커널 지원

커널 지원(Kernel Support)을 위해서 Poisoning, Reference Validation 을 수행하는 코드를 커널에 구현되어야 한다. 예를 들어 커널이 사용하는 힙 메모리에 대해서는 각 힙 메모리 할당자 들에 대한 후킹(Hooking) 이 필요하다.

3.6 적은 메모리 사용량

적은 메모리 사용량(Small Memory Usage) 을 위해서는 3.1.1 에서 설명한 Outline Instrumentation 수행하도록 한다. 이를 통해 Inline instrumentation 에 비해 메모리 사용량을 줄일 수 있다. 또한 Shadow Memory 크기 최적화를 통해 메모리 사용량을 줄일 수 있다.

3.7 좋은 실행 성능

좋은 성능(Good Runtime Performance)을 위해서는 3.1.1 에서 설명한 CTI(Compile Time Instrumentation) 방식을 사용 하도록 하였다. 이를 통해 DBI(Dynamic Binary Instrumentation) 방식에 비해 실행 성능을 높일 수 있다. 또한 Instrumentation 에 의해 삽입되는 코드 자체에 대한 최적화를 통해 성능을 높일 수 있다.

3.8 다양한 메모리 깨짐 유형 검출

앞서 언급된 여섯 가지 깨짐 유형들의 검출 지원 여부를 기존의 메모리 깨짐 검출기 들과 비교해 봤을 때 구현된 커널용 메모리 깨짐 검출기의 경우 여섯 가지 메모리 깨짐 유형을 모두 감지 가능하도록 하였다.(Detecting Various Memory Corruption)

Memory Corruption Type	Valgrind	Address Sanitizer	KMCD
1. Stack Buffer-overflow	**No**	Yes	Yes
2. Stack Use-after-free (=Use-after-return)	Yes	**No**	Yes
3. Stack Uninitialized-memory-read	Yes	**No**	Yes
4. Global Buffer-overflow	**No**	Yes	Yes
5. Heap Buffer-overflow	Yes	Yes	Yes
6. Heap Use-after-free	Yes	Yes	Yes

그림 16. 감지 가능한 메모리 깨짐 유형 비교

4. 구현

앞서 언급했던 Poisoning 과 Reference Validation 에 대해 각각 구현을 해주었다.

4.1 Poisoning

Poisoning 의 경우 스택, 글로벌 데이터 영역, 힙의 할당 해제 지점에 Shadow Memory 에 기록 하는 코드가 실행되도록 해야 하는데 스택, 글로벌 데이터 영역의 경우는 컴파일러 내에 구현한 Instrumentation 기능을 사용하여 커널에 구현한 'Shadow Memory 에 기록하는 코드'가 수행되도록 하였으며 힙의 경우 커널 내 동적 메모리 할당자에 대한 가로채기(Hooking) 을 통해 Shadow Memory 에 기록하는 코드가 수행되도록 하였다.

Memory 종류 별 Poisoning 방법	
Stack	할당해제 지점(= 함수호출, 리턴 시점)에 Poisoning 코드가 실행되도록 구현. ⇒ Compiler 의 Instrumentation 을 통해 Kernel 에 구현된 Shadow Memory 에 기록하는 코드가 수행되도록 함. (CTI)
Global	할당 지점(=프로그램 시작 시점)에 Poisoning 코드가 실행되도록 구현. ⇒ Compiler 의 Instrumentation 을 통해 Kernel 에 구현된 Shadow Memory 에 기록하는 코드가 수행되도록 함. (CTI)
Heap	할당해제 지점(Heap 할당해제 함수 호출 시점)에 Poisoning 코드가 실행되도록 구현. ⇒ Kernel 내 동적메모리 할당자에 대한 후킹(Hooking) 을 통해 Kernel 에 구현된 Shadow Memory 에 기록하는 코드가 수행되도록 함.

그림 17. Instrumentation 을 통한 Poisoning

4.2 Reference Validation

Reference Validation 의 경우 스택, 글로벌 데이터 영역, 힙 공통으로 메모리 접근 시 마다 해당 주소에 대한 유효성을 판단하기 위해 컴파일러가 Instrumentation 을 통해 커널에 구현한 'Shadow Memory 를 보고 유효성을 판단하는 코드'를 수행하게 하였으며 유효하지 않은 메모리 접근 시 개발자에게 리포트 하는 코드를 구현해 주었다.

Memory 종류 별 Reference Validation 방법	
Stack	1. Memory Access 마다 주소에 대한 Reference Validation 코드가 실행되도록 구현. ⇒ Compiler 가 Instrumentation 을 통해 Kernel 에 구현된 Shadow Memory 를 보고 유효성 판단하는 코드를 수행하게 함. (CTI)
Global	
Heap	2. 유효하지 않은 메모리 접근 시 개발자에게 리포트 하는 코드 구현. ⇒ Kernel 에 리포트 함수 구현.

그림 18. Instrumentation 을 통한 Reference Validation

전체적으로 Instrumentation 동작을 수행하는 컴파일러 쪽과 Instrumentation 에 의해 삽입된 코드들에 의해 수행되는 커널쪽 코드를 구현해주었다.

4.3 구현 결과

아래와 같이 스택 Buffer Overflow 를 일으키는 예제 코드가 있을 때 커널용 메모리 깨짐 검출기 적용을 하게 되면 Instrumentation 을 수행하는 기능이 구현된 컴파일러를 통해 컴파일 과정을 거치게 되면서 생성되는 오브젝트 파일 내에 그림 19. 와 같이 Poisoning, Reference Validation 과정을 수행하는 코드를 호출하는 Instrumentation 코드가 삽입되게 된다.

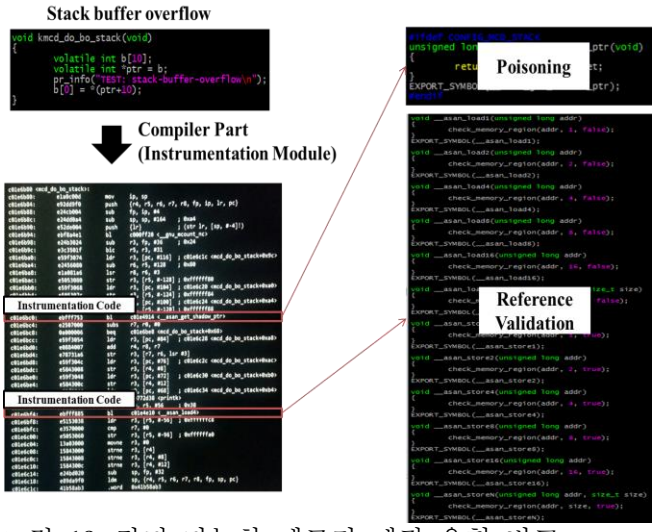


그림 19. 감지 가능한 메모리 깨짐 유형 비교

그림 20. 은 커널에서 구현된 커널용 메모리 깨짐 검출기 기능을 On/Off 할 수 있는 메뉴를 제공하는 커널 설정 화면이다.

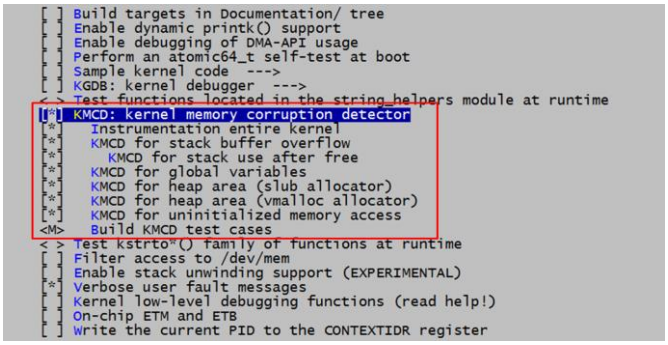


그림 20. 커널 기능 On/Off 설정 메뉴

5. 테스트

5.1 테스트 환경

테스트 환경은 아래와 같은 하드웨어 사양 하에 진행하였다.

CPU : ARM A9 Quadcore
System Memory Size : 1.7Gbyte
Kernel Version : 3.10.28 LTS(Long Term Stable)
Compiler : GCC 4.9

그림 21. 테스트 환경

5.2 테스트 케이스(Testcase)

우선 앞서 언급했던 소프트웨어에 의해 발생하는 메모리 깨짐 문제 유형 여섯 가지에 대해 각각 간단한 테스트 케이스 코드들을 작성하였다.

그림 22. 테스트 케이스 코드

5.3 테스트 케이스 실행 결과

커널용 메모리 깨짐 검출기가 적용된 환경에서 위 5.2 에서 작성된 힙 Buffer Overflow 를 일으키는 테스트 케이스 코드를 수행하게 되면 메모리 깨짐이 발생한 즉시 그림 23. 과 같은 로그가 출력됨을 확인할 수 있다. 해당 로그를 통해 개발자는 메모리 깨짐이 발생한 위치와 깨진 메모리가 할당된 경로 그리고 메모리를 깨는 경로를 정확히 확인 할 수 있으며 이를 통해 메모리 깨짐 문제의 원인을 쉽게 파악하고 해결 할 수 있다.

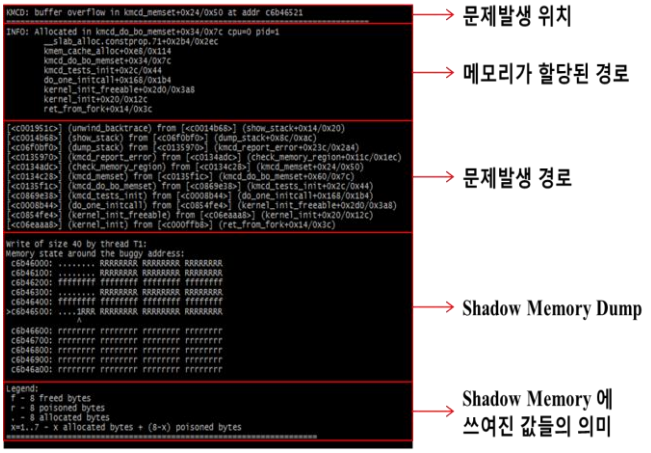


그림 23. 메모리 깨짐 검출순간 출력로그

5.4 기능 추가에 따른 바이너리 이미지 크기 변화

그림 24. 와 같이 기능이 추가됨에 따라 오브젝트 바이너리 크기는 커지게 된다. 원인은 Instrumentation 에 의한 추가적인 코드 삽입으로 인해서 이다. 커널 자체 이미지 바이너리 크기와 드라이버 바이너리 크기를 더해 32Mbyte 이하여야 하는 제약사항이 존재하는데 검출 기능을 모두 추가해도 해당 제약사항을 만족하는 것을 확인 할 수 있다.

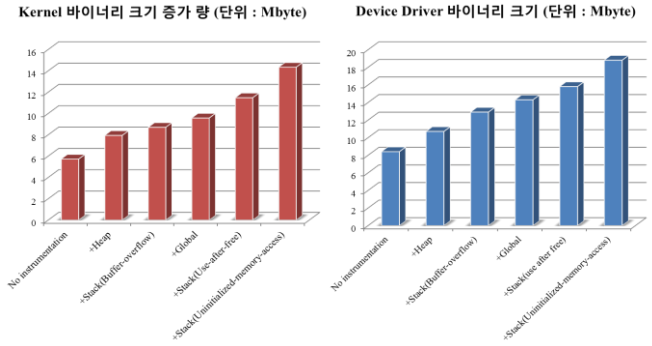


그림 24. 바이너리 이미지 크기(Mbyte) 변화

5.5 기능 추가에 따른 바이너리 이미지 크기 변화

기능 추가에 따른 메모리 사용량 크기가 증가하기 때문에 그림 25. 와 같이 부팅 직후 잔여 메모리 양이 기능 추가에 따라 줄어들게 된다. 원인은 Shadow Memory 로 인한 추가 메모리 할당과 5.4 에서 설명한 커널, 드라이버 자체 이미지 바이너리 크기의 증가 때문이다.

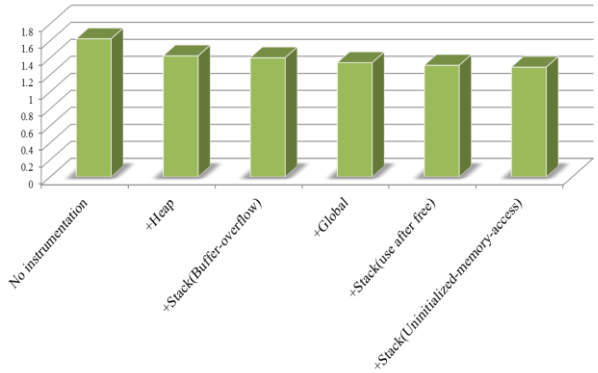


그림 25. 부팅 직후 잔여 메모리 양 (단위 : Gbyte)

5.6 기능 추가에 따른 바이너리 이미지 크기 변화

기능 추가에 따른 성능 변화도 아래 부팅 시간이 기능 추가에 따라 점점 늘어 나는 것을 확인 할 수 있다.

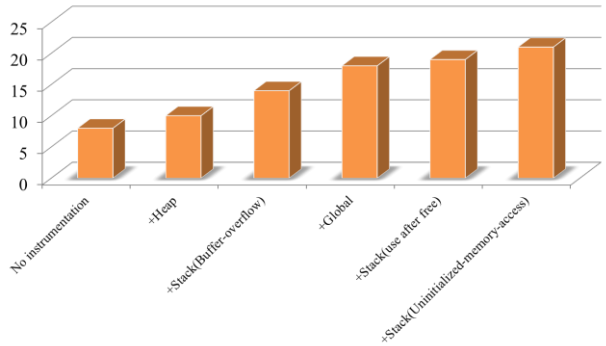


그림 26. 부팅 성능 (단위 : 초)

5.4 ~ 5.6 에서 설명된 메모리 사용량과 성능에 대한 비용을 최소화 하는 것이 중요는 하지만 커널용 메모리 깨짐 검출기 도구가 제품과 함께 탑재되어 나가는 것이 아닌 디버깅(Debugging) 용도이기 때문에 제품 출시 전 제품 통합 테스트가 가능한 수준이면 되므로 문제가 없는 상황이다.

5.7 솔루션 최종 적용 결과

앞서 1.6 에서 언급한 제품 개발 시 발생했던 커널 메모리 깨짐 문제들 22 건을 대상으로 해당 솔루션을 적용해 보았을 때 총 22 건 중 18 건이 해당 도구로 검출 가능한 것을 확인하였으며 최종 적으로 81% 의 검출 성공률을 보여준다. 검출 되지 않는 4 건의 경우 Overflow 에 의한 문제들 중 메모리를 선형적으로 접근하여 깨는 것이 아닌 메모리를 건너뛰어 접근하여 깨는 경우 일부 검출되지 않는 것으로 확인되었다. 이는 할당된 메모리 양 옆으로 위치하는 Poison Value 를 무한정 깔 수 가 없기 때문인데 이는 메모리 사용량 제약으로 인해 어쩔 수 없는 부분이다.

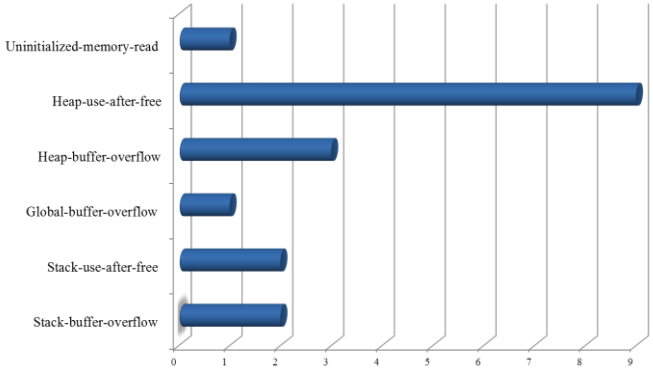


그림 27. 솔루션 최종 적용 후 검출 가능했던 문제점

6. 결론

기존 소프트웨어에 의해 발생하는 메모리 깨짐 문제점을 여섯 가지 유형으로 분류하였고 Instrumentation 기법 기반의 KMCD(Kernel Memory Corruption Detector) 즉, 커널용 메모리 깨짐 검출기를 제안하고 구현하였다. 핵심 단계인 Poisoning 과 Reference Validation 단계를 컴파일러와 커널 수정을 통해 구현하였고 기존 제품 환경에 해당 솔루션 적용 후 기존 확인된 문제점에 대해 81% 의 문제점 검출률을 달성하였다. 해당 솔루션을 이용해 실제 17 년 제품 양산시 기능 적용 및 문제점 검출 진행 중이며 장기적으로는 개발자 자주 검증 프로세스에 적용하여 해당 솔루션 적용을 강제할 예정이다. 이를 통해 커널 에서 발생하는 메모리 깨짐 이슈를 줄여 개발 기간 단축에 크게 기여 할 것으로 예상된다.

참고문헌

[1] Nethercote, Nicholas, and Julian Seward. "Valgrind: a framework for heavyweight dynamic binary instrumentation." ACM Sigplan notices. Vol. 42. No. 6. ACM, 2007.

[2] Hastings, Reed, and Bob Joyce. "Purify: Fast detection of memory leaks and access errors." In proc. of the winter 1992 usenix conference. 1991.

[3] Erickson, Cal. "Memory leak detection in c++." Linux Journal 2003.110 (2003): 8.

[4] Serebryany, Konstantin, et al. "AddressSanitizer: A fast address sanity checker." Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12). 2012.

Toolchain을 이용한 시스템 모델 생성, 검증 자동화 제안

*이승민 ○, **박용범
*단국대학교 컴퓨터학과, **단국대학교 소프트웨어학과
{Seungmin-Lee, ybpark}@dankook.ac.kr

Constructing Toolchain for the Automatic Generation and Verification of System Model

*Seungmin Lee ○, **Young B. Park
*Dankook University, Dept of Computer Engineering,
**Dankook University, Dept of Software Engineering

요 약

소프트웨어가 다양한 분야에서 복합적으로 운영되고 있다. 복합적으로 운영되는 소프트웨어는 목적 달성을 위해 서로 협력하는 구조로 설계된다. 설계는 협력을 구성하기 위하여 시스템 간 다양한 이벤트를 처리할 수 있도록 표현되어야 한다. 이러한 설계를 위해 설계 단계(Design phase)에서 시스템과 시스템의 동작에 대한 산출물 생성에 많은 시간이 소요된다. 또한, 이를 검증하는데 더욱 많은 시간이 소요된다. 이러한 문제를 해결하기 위하여 그레이-박스(Gray-Box)기반 요구사항 명세기법 연구와, 스테이트 다이어그램(State Diagram) 생성 자동화에 대한 연구가 진행되었다. 본 논문은 설계 단계의 산출물인 시퀀스 다이어그램(Sequence Diagram)을 이용하여 시스템 모델(System Model)로 변환과 검증을 자동화 하는 Toolchain을 제안하였다.

1. 서 론

다양한 소프트웨어가 등장함에 따라 여러 시스템을 복합적으로 구성하여 사용하는 시스템이 많아지고 있다. 대표적인 예로 사물인터넷(IoT), IT Ecosystem 등의 기술이 있다[1]. 이러한 기술들은 여러 시스템이 유기적으로 협력하여 목적 달성을 위해 동작하는 시스템이다[2, 3].

이렇게 시스템이 서로 협력함에 따라 시스템과 시스템이 서로 협력하기 위하여 다양한 이벤트를 처리할 수 있는 설계를 하여야 한다. 즉, 기존의 단일 시스템의 설계와 다르게 협력하는 시스템의 설계는 다양한 이벤트를 처리를 표현하기 위하여 복잡해졌다.

그 결과, 다양한 이벤트 표현을 위해 설계 단계의 시퀀스 다이어그램과 스테이트 다이어그램 작성에 많은 시간이 소요되게 된다. 특히, 스테이트 다이어그램은 객체의 이벤트에 대한 모든 상태 변화를 표현할 수 있기 때문에 작성에 더욱 많은 시간이 필요하게 되고, 이로 인하여 검증에도 많은 시간이 필요하게 된다[4, 5]. 이러한 문제를 해결하기 위하여 그레이-박스 기반 요구사항 명세 기법 연구[6]와 시퀀스 다이어그램을 통해 시스템 모델을 생성, 검증하는 연구가 진행되었다[7].

본 논문에서는 기존의 연구된 내용을 바탕으로 시퀀스 다이어그램을 시스템 모델로 변환과 검증을 자동화 하는 Toolchain을 제안한다. 2장에서는 임베디드 시스템의 요구사항 명세 기법과 그레이-박스 기반 시퀀스 다이어그램, 스테이트 다이어그램에 대한 내용과 시스템 모델에 대한 내용을 설명한다. 3장에서는 시퀀스 다이어그램에서 시스템 모델 변환 자동화와 이를 통한 Toolchain 구성 방안을 설명한다. 4장에서는 제안의 타당성과 향후 연구에 대하여 설명한다.

2. 관련 연구

본 논문에서는 시스템 모델로 변환과 검증을 자동화 하는 Toolchain을 구성하기 위하여 기존에 연구가 진행된 임베디드 시스템을 위한 그레이-박스 기반의 소프트웨어 요구사항 명세기법을 이용하였다. [6]에서는 블랙-박스(Black-Box) 관점의 명세와, 화이트-박스(White-Box) 관점의 명세를 이용하여 그레이-박스 명세 기법을 제안하였고, 그레이-박스 명세기법의 결과로 시퀀스 다이어그램과 스테이트 다이어그램을 동시에 표현한 산출물과, 시스템의 내부, 외부 간 상호작용을 명시한 산출물을 구성할 수 있었다.

3. 설계 단계의 산출물 검증 자동화 Toolchain

설계 단계의 산출물 검증 자동화를 위해서 사용할 산출물을 정의해야 한다. 본 논문에서는 그레이-박스 기반 명세 기법에서 사용하는 시퀀스 다이어그램을 사용하였다. 그레이-박스 기반 명세 기법에서 사용하는

시퀀스 다이어그램은 단순히 시퀀스 다이어그램만 표현한 것이 아니라, 스테이트 다이어그램을 동시에 표현하였기 때문에 이를 이용하여 메시지 흐름에 따른 유한한 상태의 객체의 변화를 표현할 수 있다. 이렇게 표현된 다이어그램은 유한 상태 기계(finite-state machine)의 형태를 나타내는 시스템 모델로 변환이 가능하다. 유한 상태 기계의 경우 상태 검증을 위한 여러 도구들이 많이 존재한다. 이러한 이유로 본 논문에서 하고자 하는 변환과 검증 자동화 Toolchain 구성에 있어 검증 자동화 부분에 대한 도구를 선정하는데 구체적인 대상을 선정할 수 있다.

3.1 모델 변환

모델 변환 목적은 시스템이 유한한 상태에서 제대로 동작하는지 검증하기 위함이다. 이러한 검증 과정을 진행하기 위하여 그레이-박스 명세기법의 시퀀스 다이어그램과 스테이트 다이어그램 이용하여 시스템 모델로 변환하고, 검증을 진행한다.

3.2. Toolchain의 구성 방안

모델 변환을 위해 시퀀스 다이어그램과 스테이트 다이어그램을 작성하는 도구(Diagram Tool)와 시스템 모델을 검증하는 도구(System Model Verify Tool)가 필요하다[그림1].

각 도구들은 도구에서 작성된 산출물을 저장할 때 어떠한 파일 형태(.xml, .txt, .mdj, …)로 저장된다. 저장된 파일의 형태가 동일하더라도, 도구에서 지원하는 형식에 따라 인식되지 않을 수 있다. 이러한 점을 해결하기 위하여 도구 간 지원 형식을 연결해주는 모델 변환 도구(Model Converter Tool)를 구성하였다[그림2].

XmlController는 xmlConvertingFileOpen()을 이용하여 Diagram Tool의 파일을 읽어오고, makeXML()을 이용하여 System Model Verify Tool에서 사용할 수 있는 형태로 변환을 해준다. makeXML()은 데이터 변환 작업을 하기 위해서 Diagram 작성 도구의 데이터를 데이터 형태와 역할에 따라 LifeLineProperty와, MessageProperty에 각각 나눠서 저장된 데이터를 이용하여 System Model로 구성 가능한 형태로 변환을 진행하고, 최종적으로는 변환된 데이터를 XML 파일로 구성함으로써 시스템 모델 검증 도구가 사용할 수 있는 형태로 구성해준다[그림3].

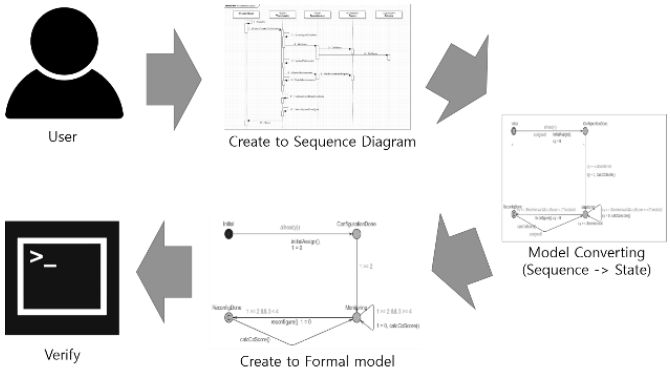


그림1. Toolchain 구성 방안

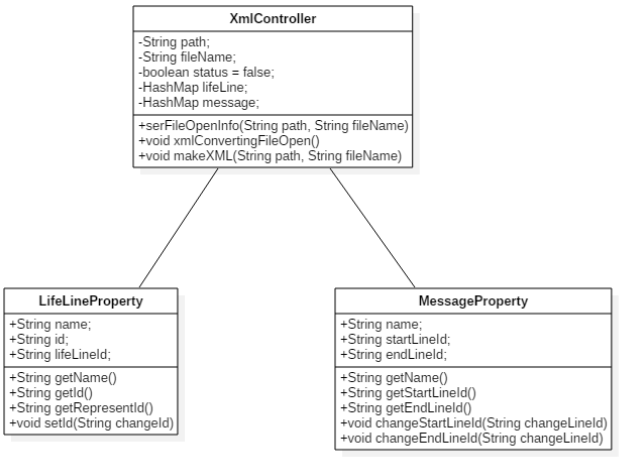


그림2. Model Converter 구성 방안

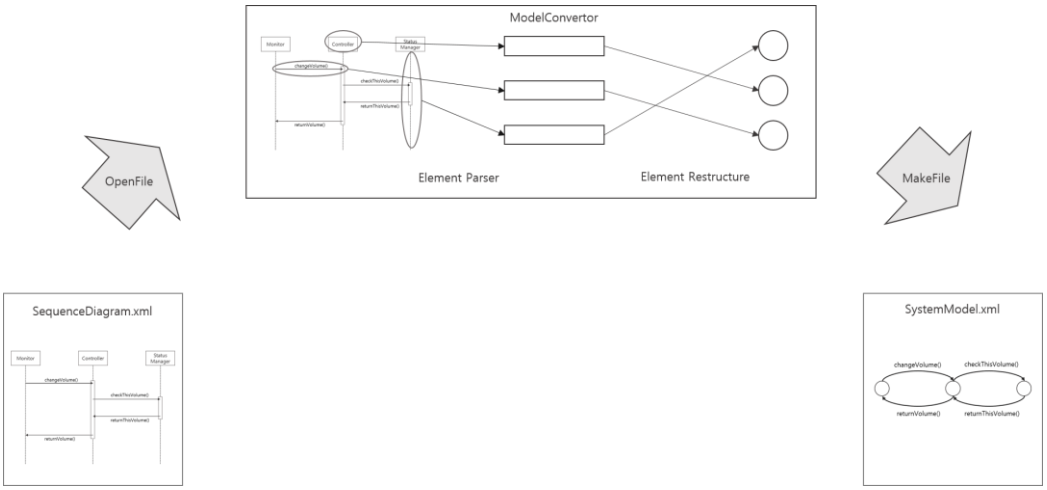


그림3. Model Converter Process

4. 제안의 타당성 및 향후 연구

본 논문에서는 그레이-박스 기반 요구사항 명세 기법에서 사용한 시퀀스 다이어그램을 기반으로 시스템 모델의 생성과 검증을 자동화 하는 Toolchain을 제안하였다. 또한, 다이어그램 작성 도구와 다이어그램을 시스템 모델로 변환해주는 모델 변환 도구(Model Convertor Tool), 시스템 모델 검증 도구의 역할과, 구성을 보여줌으로써 Toolchain의 구성 가능성을 확인하였다.

그러나, 시스템 모델 검증 도구에서 사용하는 명령어에 대한 연구가 부족하여 Model Convertor의 구성과 시스템 모델을 검증하기 위한 명령어에 대한 연구가 진행되어야 한다.

향후 연구로는 제안한 Toolchain을 구성하고, 구성한 Toolchain을 통해 실제 모델을 대상으로 검증을 진행할 것이다.

Acknowledgement

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No.2012M3C4A7033348).

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 지역신산업선도인력양성사업 성과임(No.NRF-2016H1D5A1909989)

참고문헌

[1] A. Ukil, J. Sen, S. Koilakonda. Embedded Security for Internot of Things. Emerging Trends and Applications in Computer Science (NCETACS), 2011 2nd National Conference on. IEEE, pages 1-6. 2011.

[2] Park, Soojin, Lee Seungmin, and Young B. Park. "A Reference Architecture Framework for Orchestration of Participants Systems in IT Ecosystems." Advances in Computer Science and Ubiquitous Computing. Springer Singapore, 2015. 883-889.

[3] Lee, Seungmin, Young B. Park, and Soojin Park. "A case study of Self-adaptive Software in the dynamic reconfiguration of IT Ecosystem." 2016 International Conference on Big Data and Smart Computing (BigComp). IEEE, 2016.

[4] Kopetz, Hermann. "The complexity challenge in embedded system design." 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). IEEE, 2008.

[5] Drusinsky, Doron. Modeling and verification using UML statecharts: a working guide to reactive system design, Runtime Monitoring and Execution-based Model Checking. Newnes, 2011.

[6] Park, Soo-Jin, and Soo-Yong Park. "A Gray-Box based Software Requirements Specification Method for Embedded Systems." Journal of KIISE: Software and Applications 38.9 (2011): 485-490.

[7] Lima, Vitor, et al. "Formal verification and validation of UML 2.0 sequence diagrams using source and destination of messages." Electronic Notes in Theoretical Computer Science 254 (2009): 143-160.

코드 클론을 효율적으로 관리하기 위한 시각화 방안

김하영, 최은만
동국대학교 컴퓨터공학과
h.3i3.young@gmail.com, emchoi@dongguik.edu

A Visualization Method for Efficient Code Clone Management

Ha Young Kim, Eun Man Choi
Department of Computer Science Dongguk University

요 약

코드 클론은 프로그램 내에서 원시 코드의 일부가 유사하여 중복된 코드로 판별 된 것이다. 코드 클론은 개수가 증가할수록 소스 코드의 크기가 커지고, 최소 두 개 이상의 클론 조각을 찾아서 업데이트 해야하는 단점이 있다. 따라서 코드 클론을 관리하는 것은 소프트웨어 유지보수 단계에서 당연히 고려되어야 할 요소이다. 본 연구는 코드 클론 관리에 필요한 정보를 수집하는 CCDI와 시각화 도구인 Clone-GUIde를 구현하였다. 구현한 도구를 통해 사용자가 코드 클론을 효율적으로 관리할 수 있는 방법에 대해 제안한다.

1. 서 론

코드 클론(code clone)은 프로그램 내에서 원시코드의 일부가 유사하여 중복된 코드로 판별된 것이다. 클론은 개발자가 원본 소스 코드를 재사용할 때 생성된다[1]. 코드 클론이 있으면 전체 코드 량이 늘어나 가독성을 저하시키고, 기능 수정이 필요할 때 유지보수가 어려워진다[8]. 하지만 분리 개발된 대규모 소프트웨어에서 코드 클론은 단시간에 없애거나 리팩토링 하는 것은 현실적으로 불가능하다. 실제 소프트웨어 개발 현장에서는 시간에 쫓겨 설계를 고칠 틈이 없고 리뷰가 잘 이뤄지지 않기 때문이다.

따라서 코드 클론을 한 곳으로 모아 정보를 관리할 필요가 있고, 거시적인 클론의 통계나 변동 추세, 위치, 클론 사이의 관계 등을 파악할 수 있어야 한다. 하지만 이러한 정보는 코드나 텍스트 파일에서는 나타내기가 어렵고 시각화 방법이나 메트릭 기반 레포트를 함께 제공해야 한다[3]. 결국, 클론 관리를 위해서는 클론 관리 관점에서 어떤 데이터가 필요한지 분석하고 이를 잘 시각화한 연구가 필요하다.

본 논문에서는 코드 클론을 관리할 때 필요한 데이터를 찾고, 효율적으로 시각화하는 방안을 모색하였다. 기존 클론 시각화 도구들이 제공하는 데이터들에는 무엇이 있는지 분석하고, 관리 활동에서 추가로 필요한 데이터들을 프로젝트에서 추출하였다. 그리고 이 데이터를 효율적으로 표현할 수 있는 방안은 무엇인지 탐구하여 시각화 도구로 구현하였다.

논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 기술하고, 3장에서는 코드 클론을 관리하기 위해 제안한 시각화 구현을 설명한다. 4장에서는 기존 시각화 도구와 비교 평가를 하며, 5장에서는 결론과 향후 연구에 대해 논의한다.

2. 관련 연구

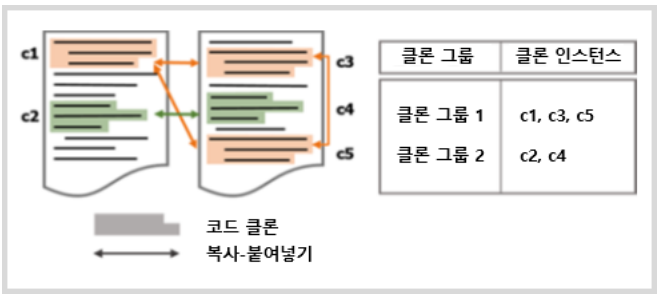
코드 클론은 10년 이상 연구된 분야이다. 1994년부터 2013년까지 진행된 관련 연구들을 조사한 결과 코드 클론 분석이 43%, 탐지가 39%, 관리는 15%, 도구 평가는 3%의 비율을 차지하였다[1]. 이 중 본 연구는 코드 클론 관리에 해당한다.

2.1 코드 클론 용어

이 절에서는 설명할 코드 클론 용어는 클론 탐지 도구에서 검출되는 결과를 표기할 때 사용된다.

1) **코드 단편 (Code Fragment)** : 코드 단편은 연속된 코드 라인으로 구성되어 있으며, 빈 칸이나 주석은 포함된 경우도 있고 그렇지 않은 경우도 있다. Tuple(f,s,l)로 나타낸다. f는 file name, s는 start line number, l은 total number of lines를 뜻한다. 코드 단편이 끝나는 라인은 s+l-1 로 구한다.[3]

2) **코드 클론** : 코드 단편 두 개 이상이 어떠한 유사성 판단 기준에 의해 같다고 판단된 것이다[3]. [그림 1]은 코드 클론을 기준으로 클론 그룹과 클론 인스턴스라는 트리 구조로 표현할 수 있다.

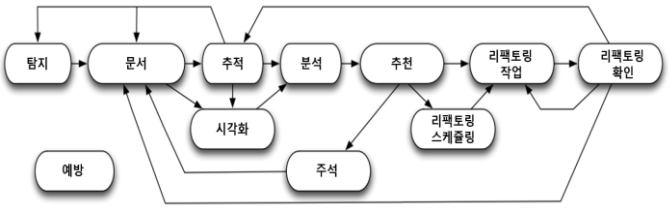


[그림 1] 코드 클론 정의 [1]

- 클론 그룹 : 원시 코드를 기준으로, 같은 코드 단편 내용을 공유하는 클론 인스턴스 집합이다.
- 클론 인스턴스 : 클론 그룹 내부에 있는 클론 인스턴스는 동일한 코드 단편 내용과 코드 상에서 자신의 위치를 갖는다.

2.2 코드 클론 관리

코드 클론 관리란 클론을 탐지하거나, 회피하거나 혹은 제거하는 모든 프로세스 활동을 의미한다. 코드 클론을 관리하는 활동은 [그림 2]와 같으며, 다음 세 가지로 압축할 수 있다.[1]



[그림 2] 코드 클론 관리 작업흐름도[1]

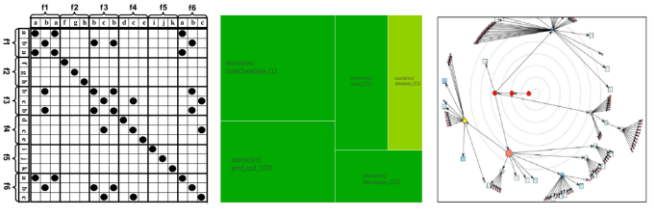
1) **코드 클론 탐지** : 코드 클론 탐지 도구를 이용하여 소스 코드에 존재하는 클론을 식별하는 활동이다. 탐지된 클론들의 위치와 내용은 문서에 저장한다.
2) **코드 클론 진화 추적** : 개발 중 코드가 변하면서 변경되는 코드 클론 진화를 추적하는 활동이다. 변경되는 코드 클론 정보는 문서에 업데이트 한다.
3) **코드 클론 리팩토링** : 코드 클론을 메소드나 컴포넌트, 혹은 애스펙트 안에 병합시켜 제거하는 작업이다. 리팩토링 할 수 있는 코드 클론을 분류하고, 이를 주석이나 리팩토링 스케쥴링에 포함시킨다.

[그림 2]에서 시각화는 위 코드 클론 관리 활동을 지원하는 연결고리 역할을 한다. 다음 절에서는 시각화에 대해 자세히 알아본다.

2.3 코드 클론 시각화

코드 클론 시각화는 탐지 도구로 코드 클론을 검출한 후, 검출한 결과를 도표, 차트 등을 이용해 시각적으로 제공한다.

2.3.1 도표 기반 코드 클론 시각화



산포도 - Gemini[9] 트리맵 – SonarQube[12] 계층종속성 그래프 – VisCad[5]

[그림 3] 도표 기반 시각화 방법과 도구

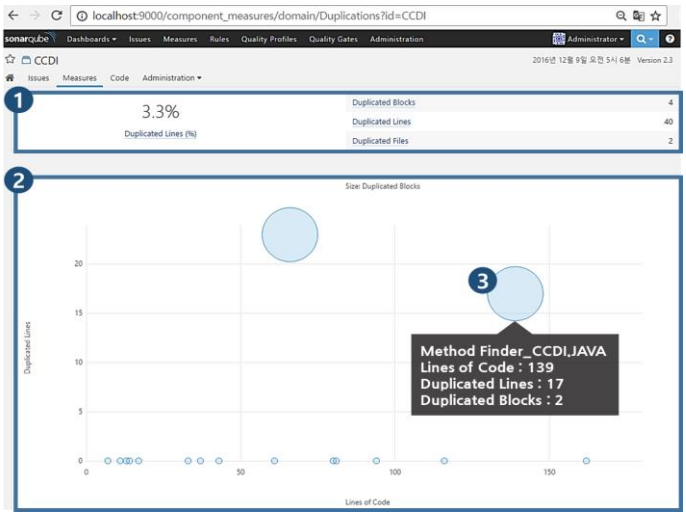
코드 클론 시각화 도구에서 지원하는 도표는 [그림 3]처럼 주로 산포도, 트리맵, 계층 종속성 그래프를 사용한다. 이 도표들은 파악한 코드 클론 결과를 질량으로 비교하여 나타내는 목적으로 사용된다.

시스템에서 코드 클론이 밀집된 위치를 파악 하는 데는 도표로 보는 것이 편리하다. 하지만 코드 클론 간 관계에 대한 추가 정보는 표시되지 않기 때문에 리팩도링 판단이 불편한 단점이 있다. 또한 도표는 시스템이 커질수록, 서브 그래프에서 나타난 중복 표현이 그래프의 복잡도를 높인다. 따라서 도표를 잘 세분화하여 볼 수 있는 나눌 수 기능이나 필요 없는 데이터는 감출 수 있는 기능이 반드시 필요하다.

2.3.2 이클립스 플러그인 기반 코드 클론 시각화

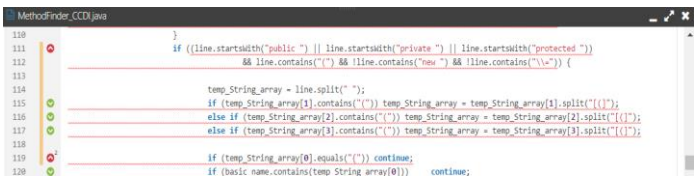
1) Sonar Qube [12]

Sonar Qube는 소프트웨어 품질을 시각화할 때 사용한다. Sonar Scanner로 코드를 분석한 후, 분석한 결과는 웹에서 확인한다.



[그림 4] Sonar Qube 중복 코드 시각화 화면

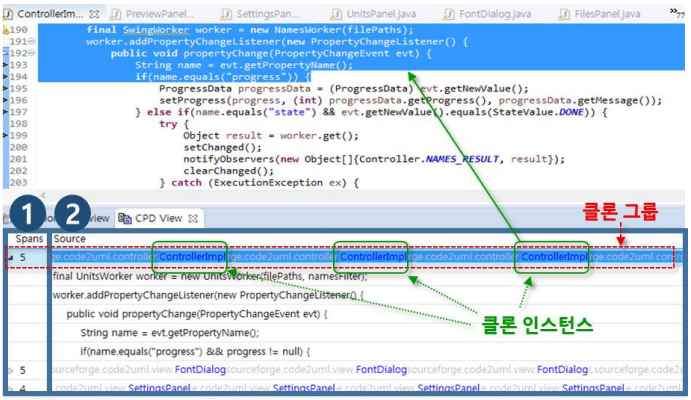
①번 뷰는 프로젝트에서 중복코드가 차지하는 비율을 %로 보여주고, 중복된 라인수와 블록, 파일 개수를 알 수 있다. 각각의 메트릭 정보는 하이퍼링크로 연결되어 하위 정보를 확인 할 수 있다. % 를 클릭했을 때는 클론이 있는 목록과 파일 간 트리맵을 제공한다. 트리맵을 클릭했을 때 이벤트는 [그림 5]와 같은 소스 코드 창을 하단에 띄운다. 오른쪽의 메트릭들도 클론이 있는 파일 리스트와 연결되고 소스 코드를 자세하게 확인할 수 있다.



[그림 5] Sonar Qub 소스 코드 창에서 코드 클론 표시

2) PMD [11]

PMD는 정적 분석 도구이다. PMD에서 CPD(Copy and Paste Detector) 기능은 클론을 검출하여 텍스트 기반으로 시각화 한다. 시각화 화면은 [그림 6]와 같다.



[그림 6] PMD-CPD의 중복 코드 탐지 화면

분석 결과는 이클립스 화면 하단에서 CPD view에 나타나며 상단의 소스 코드 뷰와 연동된다. CPD view는 ① spans과 ② source 메뉴로 나뉜다. ① spans 메뉴는 검출된 코드 클론의 라인 수이다.

② source에서 spans 숫자 바로 옆의 한 줄은 클론 그룹이다. 그리고 각각의 클론 인스턴스에 링크를 걸어 클론이 있는 소스 위치로 이동할 수 있게 하였다. 코드 뷰에 나타내는 클론은 컬러-코딩(color-coding) 방법으로 구분한다. 검출된 클론은 이클립스 IDE창에서 수정이 가능하지만 도표 시각화는 지원하지 않는다.

소스 코드 레벨에서 코드 클론을 표시하는 방법은 코드 클론 간의 상속 관계를 발견하기 어렵다. 소스 코드는 시각화에서 볼 수 있는 가장 하위 레벨로 리팩토링 판단에서 꼭 확인해야 할 정보이지만, 추상화를 보기에는 부적절하다. 따라서 적절한 시각화와 연계하는 것이 중요하다.

3. 코드 클론 시각화 도구 설계 및 구현

3.1 구현 동기

코드 클론을 관리하는 입장에서 시각화하기 위해서는 중복된 코드에서 의미 있는 결과를 도출하고자 하는 경영적인 시각과 데이터를 분석하는 기술적인 영역을 동시에 고려해야 한다. 코드 클론 관리를 고려한 시각화 방법을 위해 2.2절에서 설명한 코드 클론 관리활동을 시각화 조건에 추가하였다.

- 조건-1. 코드 클론을 없앨 수 있는 리팩토링 판단을 돕는다.
- 조건-2. 코드 품질을 관리하기 위해 코드 클론 진화를 지원한다.

위 조건에 필요한 코드 클론 관련 정보는 <표-1>에 정리하였다.

<표-1> 관리에 필요한 코드 클론 데이터 특징

코드 클론 관리 활동	클론의 특징	내 용
코드 클론 탐지	자료	크기(토큰, 라인 수), 복잡도, 출현 빈도
	위치	위치, 모듈 단위에서 분포(커버리지)
코드 클론 진화 추적	동향과 패턴	변경 횟수, 클론 진화, 변화의 방향성, 속도
코드 클론 리팩토링	시스템 내에서 연관	변경 영향도, 연관 관계, 계층 관계
	시스템 외부와 연관	클론 제작팀과 연관

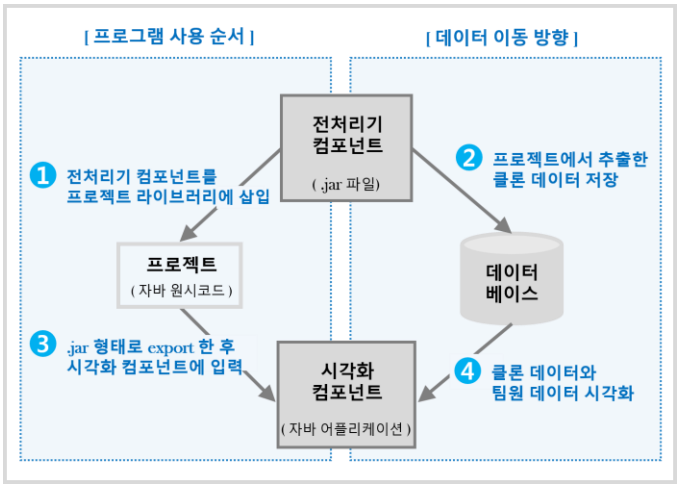
관련 연구에서 알아본 시각화 도구들은 <표-1>의 클론 탐지 부분을 표현하는 데 집중되었다. 그렇기 때문에 코드 클론 관리활동을 전부 지원하지 못했다. 이러한 기존 시각화 도구 단점을 보완하기 위해 코드 클론을 제거할 수 있는 리팩토링 지원과 코드 클론 진화를 추적할 수 있는 정보 축을 추가하였다. 추가한 시각화 정보는 다음과 같다.

- 코드 클론 리스트를 클래스 다이어그램과 연동
- 코드 클론을 만든 개발자 추출
- 코드 클론이 생성된 시간 추출

다음 절에는 위 시각화 정보를 추가하여 본 논문에서 구현한 프로그램을 소개하겠다.

3.2 코드 클론 관리 시각화 프로그램 전체 구조

본 논문에서 구현한 코드 클론 관리 시각화 프로그램은 [그림 7]처럼 두 개의 컴포넌트와 데이터베이스로 이뤄졌다. 프로그램이 지원하는 언어는 JAVA이다.



[그림 7] 코드 클론 관리 시각화 프로그램 구조

[그림 7]에서 표기된 1~4까지의 숫자는 프로그램이 구동되는 순서이다. 좌측은 사용자 시점에서 프로그램을 사용하는 순서이고, 우측은 프로그램을 사용함에 따라 이동하는 데이터의 방향이다.

1) 전처리기 컴포넌트

프로젝트에서 코드 클론과 이를 생성한 개발자 목록을 검출한 후에 데이터베이스에 저장하는 역할을 한다. 코드 클론 검사 도구와 데이터베이스에 관련된 외부 API들을 포함한다.

본 논문에서 제안하는 전처리기 컴포넌트는 CCDI (Code Clone Data Inspector)로 칭하겠다. CCDI는 분석 대상이 되는 프로젝트의 외부 라이브러리에 추가하여 사용한다. 그 이후, 메인 함수에서 [그림 8]의 코드를 추가하면 코드 클론 데이터를 탐지하여 저장하는 메소드를 호출하게 된다.

```
CodeClone c = new CodeClone();
  try {
      c.SaveCodeCloneData(true, true);
  } catch (Exception e) {
      e.printStackTrace();
  }
```

[그림 8] CCDI 실행 코드

위 그림에서 SaveCodeCloneData 메소드는 검출된 코드 클론 결과를 저장한다. 첫 번째 파라미터는 xml 파일에 저장 여부, 두 번째 파라미터는 데이터베이스에 저장 여부이다.

2) 시각화 컴포넌트

데이터베이스에 저장된 코드 클론 데이터를 시각화 한다. 본 논문에서 제안하는 시각화 컴포넌트는 Clone-GUIde(Code Clone Guide)로 칭한다. Clone-GUIde는 자바 어플리케이션 형태로 실행된다. 이 컴포넌트에서 제공하는 뷰는 3.1절 구현 동기에서 소개한 코드 클론 관리 활동을 지원한다.

3) 데이터베이스

전처리기 컴포넌트에서 코드 클론 데이터가 저장되며 저장된 코드 클론 데이터는 시각화 컴포넌트에서 사용된다.

3.3 코드 클론 전처리기 - CCDI

3.3.1 CCDI 요구사항

본 논문에서는 코드 클론을 만든 개발자를 파악하기 위해 자바 어노테이션[1]을 사용하였다. CCDI에는 미리 정의된 @TypeHeader 어노테이션이 있고, 이 어노테이션은 author 속성을 가진다.

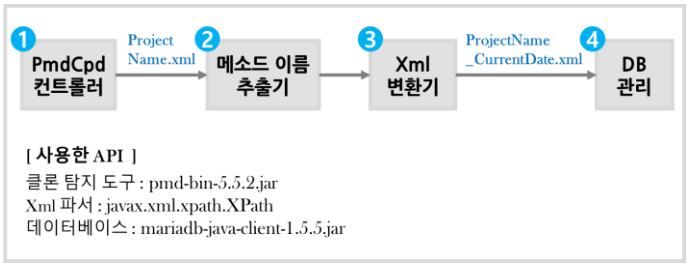
분석 대상이 되는 프로젝트에 CCDI를 추가하면, CCDI 내부에 정의된 @Type Header 어노테이션을 사용할 수 있다. 개발자는 [그림 9]처럼 메소드 이름 위에 @TypeHeader와 이 어노테이션의 author 속성에 값을 입력한다.

```
@TypeHeader (author = "Ha-Young")
public String ModifyXml (String xml_path){ ... }
```

[그림 9] @TypeHeader를 추가하는 코드

또한 코드 클론을 생성한 시각을 파악하기 위해, 프로젝트에서 어노테이션을 읽어 데이터베이스에 저장 하는 기능을 구현하였다.

3.3.2 CCDI 구조 및 구현



[그림 10] 코드 클론 전처리기 프로세스

1) PmdCpd 컨트롤러

본 논문에서는 클론 탐지 도구로 얻은 결과를 다시 시각화 프로그램에 입력하는 과정을 단축하기 위해, 클론 검사기를 자동으로 설치하는 기능을 전처리기 프로그램 내부에 구현하였다.

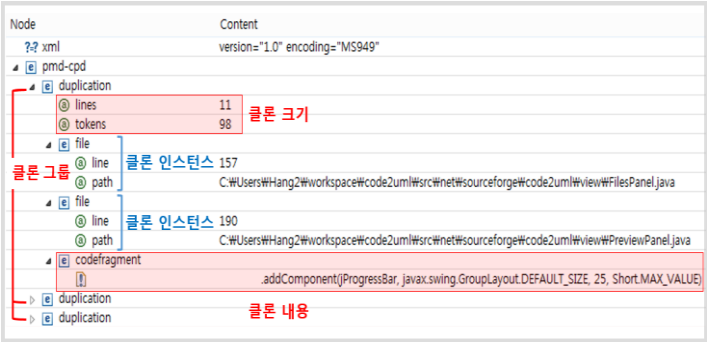
PmdCpd 컨트롤러는 프로젝트에 클론 검사기를 설치한다. 클론 검사 도구는 관련 연구에서 소개한 Pmd 도구에 포함된 Cpd(copy and paste detec tor)모듈을 사용하였다. Cpd를 실행시키면 프로젝트 내부에 CodeCloneData 디렉터리를 생성하고, 이 폴더에 코드 클론 검출 결과를 저장한다. Cpd.bat 파일을 실행시키는 명령어는 [그림 11]과 같다.

```
Cpd.bat --files path₩₩CodeCloneData₩₩
--minimum-token 30 --format xml >>
path₩₩CodeCloneData₩₩ProjectName.xml;
```

[그림 11] Cpd.bat 실행 명령어

[그림 11]에서 --minimum token은 코드 클론으로 판단하는 최소 토큰 개수이다. 본 논문에서는 도구에서 제공하는 기본 값인 30 token으로 설정하였다. Cpd는 연속된 코드 토큰이 30개 이상 중복되면 클론으로 판단한다. 검출된 결과가 저장될 파일 포맷은 —format xml로 지정하였다. 검출 결과인 ProjectName.xml 파일의 구성은 [그림 12]과 같다.

[1] 자바 소스 코드에 붙일 수 있는 메타 데이터(meta data)로 라벨처럼 사용한다.



[그림 12] Cpd 실행 결과인 ProjectName.xml 구조

위 그림에서 <duplication>태그는 클론 그룹의 개념과 대응하고, 이 태그 내부의 <file>태그는 클론 인스턴스 개념과 대응한다.

2) 메소드 이름 추출기

이 모듈은 PmdCpd 컨트롤러에서 넘겨받은 Project Name.xml 파일과 원본 소스 코드에서 클론이 속한 메소드 이름과 생성한 개발자 데이터를 추출한다.

코드 클론 리팩토링은 메소드 단위로 이뤄지기 때문에, 코드 클론이 위치한 메소드의 파악은 매우 중요하다. 하지만 코드 클론 검사 결과는 [그림 12]처럼 path가 파일 이름 까지만 추출되고, 코드 클론이 시작하는 위치인 line 값으로 구분하는 것을 볼 수 있다. 따라서 본 논문에서는 프로젝트 소스 코드와 코드 클론 검출 문서에서 코드 클론이 속한 메소드 이름을 추출하는 기능을 구현하였다. 이 모듈은 아래 두 가지 경우를 판단하여 메소드 이름을 추출한다.

- code fragment 내부에 메소드 이름이 있는 경우
- code fragment 내부에 메소드 이름이 없는 경우

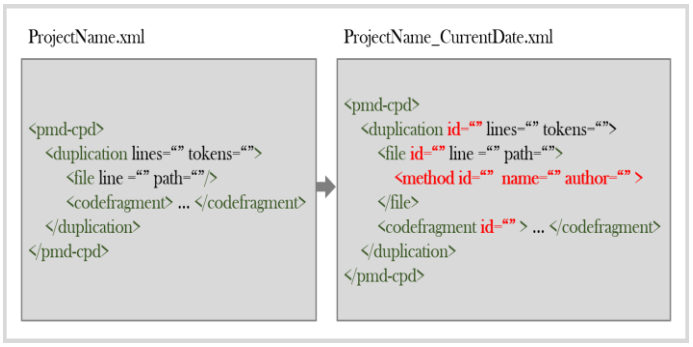
첫번째 경우에는 검출된 코드 클론 문서(Project Name.xml)에서 code fragment 안의 코드 내용으로 찾아낼 수 있다. 하지만 2)번의 경우는 프로젝트에서 직접 클론의 위치를 찾아 비교하는 추가 작업이 필요하다. [그림 12]의 line과 path 속성을 입력 받아 찾는다. 먼저 path에 위치한 파일을 불러온 후, 코드 클론이 시작하는 line을 기준으로 가장 가까운 메소드 이름을 탐색한다.

또한 이 모듈은 3.3.1항에서 요구하였던 @Type Header 어노테이션에 있는 author 속성을 수집한다. 추출한 개발자 이름 및 메소드 이름은 xml 파일과 데이터베이스 저장 모듈로 넘어가서 저장된다.

3) Xml 변환기

ProjectName.xml을 수정하여 ProjectName_Current Date.xml로 저장한다. Xml 변환기를 거치기 전과 후의 파일 비교는 [그림 13]과 같다. 기존 파일에서 클론 인스턴스인 <file> 태그 내부에 <method> 태그를 추가하고, name과 author 속성을 수집하였다.

- <method>의 name 속성 : 코드 클론이 속해 있는 메소드 이름을 추가한다.
- <method>의 author 속성 : 코드 클론을 제작한 개발자를 관리한다. 프로젝트에서 개발자 정보가 담긴 어노테이션을 읽어와 해당 메소드가 코드 클론이면 개발자 이름을 추가한다.



[그림 13] XML Modifier 전, 후의 xml 파일 변화

4) DB 관리

데이터베이스 연결과 코드 클론과 관련된 데이터를 저장한다. 이 모듈은 코드 클론 동향을 파악하기 위해 날짜 데이터를 관리하고 하루 중에서 마지막으로 컴파일 한 코드 클론 데이터만 남기고 삭제한다.

3.4 코드 클론 시각화 도구 – Clone-GUIde

Clone-GUIde는 CCDI에서 저장한 클론 데이터를 시각화 한다. 3.1장 구현 동기에서는 기존 클론 시각화 연구를 통해 본 연구에서 개발한 시각화 도구가 제공해야 할 기능을 고려하였다. 그 조건을 반영하여 Clone-GUIde에서 시각화 할 내용은 다음과 같다.

- 조건-1. 코드 클론을 없앨 수 있는 리팩토링 판단을 돕는다.

코드 클론 리팩토링은 클론을 지워도 프로그램 동작에 이상이 없도록 다른 코드, 함수, 모듈 간 관계를 고려해야 한다. 즉, <표-1> 코드 클론 리팩토링 특징 중 클론과 시스템 내부 연관에 해당한다. 본 논문에서는 프로젝트 내의 코드 클론 분포 현황을 나타내는 방법으로 클래스 다이어그램 형태의 뷰를 선택하였다. 클래스 다이어그램은 클래스 간의 상속과 연관 관계를 잘 나타내기 때문에 리팩토링 설계가 용이하다. 따라서 이 방법은 코드 클론을 소스 코드 내부에서 표기하는 것보다 모델링 분석 시간이 짧으며 직관적으로 리팩토링이 필요한 부분을 감지할 수 있다.

- 조건-2. 코드 품질을 관리하기 위해 코드 클론 진화를 지원한다.

코드 클론은 코드 클론을 생성한 개발자와 관계를 갖는다. 개발자가 코드 클론을 언제, 얼마나 생성했는지 파악하는 것은 <표-1> 코드 클론 특징 중 진화 추적

및 코드 클론과 시스템 외부 연관에 해당한다.

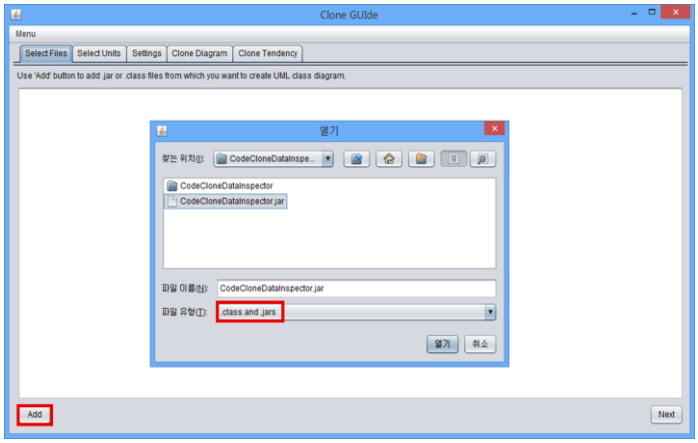
코드 클론 진화를 파악하면 관리자는 인력 관리와 함께 프로젝트에서 코드 클론이 리팩토링된 현황을 파악할 수 있다. 반면 개발자는 코드를 생성할 때 이미 존재하는 코드 클론을 탐지하여 비교할 수 있기 때문에 새로운 코드 클론을 생성하는 것을 방지 할 수 있다.

3.4.1 리팩토링 지원 뷰 구현 내용

이 뷰는 3.4절에서 조건-1을 구현하였다. 리팩토링 지원 뷰에서 클래스 다이어그램을 구현하는 부분은 Code2Uml[13] 도구를 사용하였다. Code2Uml은 소스코드를 클래스 다이어그램으로 변환해주는 오픈 소스 프로그램이고, JAVA만 지원한다.

아래에서 설명할 4개의 항목 중 1)~3) 항목은 Code2Uml에서 기본으로 제공하는 클래스 다이어그램 설정과 관련된 기능이다. 4) 항목은 본 논문에서 제안하는 리팩토링 지원 시각화 아이디어를 구현하였다. 분석한 프로젝트는 3.3 절에서 설명한 CCDI 를 예시로 테스트하였다.

1) Select File 탭 : 분석할 프로젝트 선택



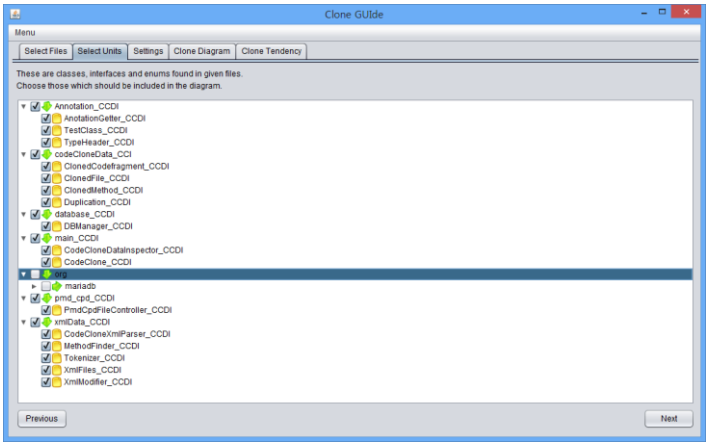
[그림 14] 분석할 프로젝트 선택 화면

[그림 14]의 Select Files 탭에서 하단의 add 버튼을 눌러 분석할 프로젝트를 추가한다. 프로젝트는 .jar 파일이나 .class 파일 형식만 분석할 수 있다.

2) Select Units 탭 : 클래스 다이어그램으로 나타낼 패키지 및 파일 목록 선택

전 단계에서 선택한 프로젝트 내부의 패키지 및 파일을 트리 형태로 생성한다. 이 단계에서는 클래스 다이어그램으로 표현하고 싶은 파일을 선택할 수 있다. 프로그램 전체를 클래스 다이어그램으로 나타내면 화면에 표현하는 정보가 많아서 복잡해진다. 따라서 사용자가 확인하고 싶은 부분만 선택하는 기능이 필요하다.

[그림 15]에서는 CCDI의 패키지 중, 데이터베이스와 관련된 MariaDB[14] API는 체크를 제거하였다.



[그림 15] 분석할 패키지 및 파일 선택 화면

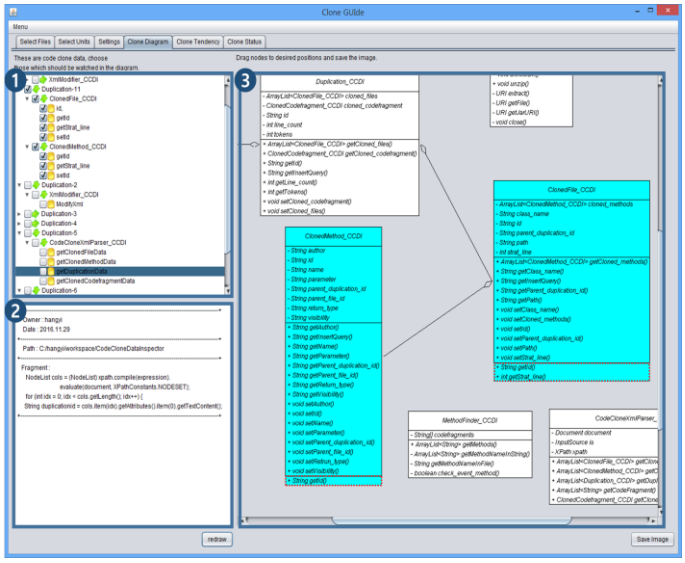
3) Settins 탭 : 클래스 다이어그램 설정

Settings 탭에서는 앞 단계에서 선택한 파일을 나타낼 클래스 다이어그램의 세부 설정을 제공한다. 클래스 다이어그램에 표기할 속성을 체크 박스에서 고를 수 있다. 또한 클래스 다이어그램의 기본 색상과 글씨체 등을 지정한다.

4) Clone Diagram 탭

Clone Diagram 탭은 조건-1을 구현한 화면으로, 클래스 다이어그램에 코드 클론을 표현한다. [그림 16]은 크게 세 가지 뷰로 나누었다.

①번 뷰는 데이터베이스에서 가장 최근에 저장된 코드 클론 정보를 가져와서 체크 리스트로 보여준다. ②번 뷰는 ①번 리스트에서 선택한 코드 클론의 정보를 표시하고, ③번 뷰는 ①번 리스트에서 선택한 코드 클론을 클래스 다이어그램에 표시한다.



[그림 16] 클래스 다이어그램에서 코드 클론 분포

[그림 16] 좌측에 배치된 ①번과 ②번 뷰는 체크 트리와 텍스트로 시각화 하였다. ①번 뷰의 코드 클론 리스트는 클론 그룹 - 클론 인스턴스 - 메소드 순서의 트리 구조로 구성되었다.

①번 뷰 리스트에서 선택한 클론은 컬러 코딩으로 표시되고, 선택한 클론과 관련된 추가 정보를 ②번 뷰에서 표시한다. ②번 뷰에 표시되는 정보는 클론을 개발한 개발자명과 클론을 생성한 날짜, 클론의 경로 그리고 코드 내용을 나타내었다.

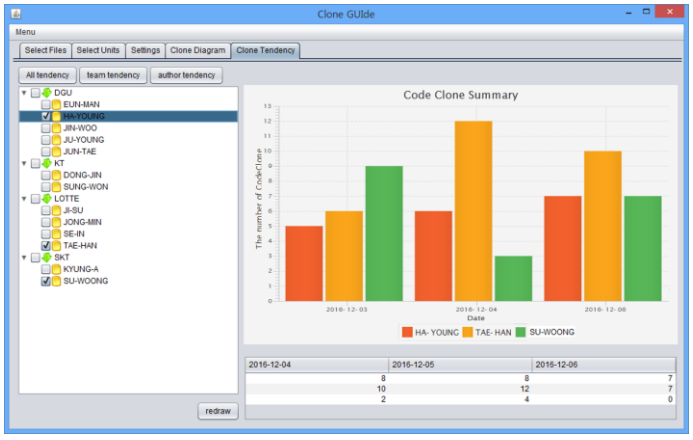
[그림 16] 우측에 배치한 ③번 뷰는 ①번 코드 클론 리스트에서 클론 그룹을 선택하고 하단의 redraw 버튼을 누르면 정보를 갱신한다. ①번 뷰에서 선택한 클론 그룹 Duplication-11 내부에는 클론 인스턴스인 ClonedFile_CCDI 클래스와 ClonedMethod_CCDI 클래스가 있다. 선택한 클론 그룹은 ③번 뷰의 클래스 다이어그램에서 색이 변경되고, 클래스 다이어그램 하단에 코드 클론 메소드가 추가로 표시된 것을 확인할 수 있다. 그리고 하단 우측의 Save Image 버튼을 누르면, ③번 뷰에 나타난 클래스 다이어그램을 이미지 파일로 저장한다.

3.4.2 코드 클론 개수 변화 추적 뷰 구현

1) Clone Tendency 탭

이 뷰는 3.4절에서 조건-2를 구현하였다. 프로젝트에 참여하는 개발자 및 팀에서 코드 클론을 언제, 얼만큼 제작했는지 파악하고 관리할 수 있도록 구현하였다. 크게 세가지 메뉴 버튼을 제공한다. 메뉴 별 코드 클론 변동 추이를 막대 그래프와 표로 파악할 수 있도록 구현하였다.

- All Tendency : 한 프로젝트에서 생성한 코드 클론 현황 확인
- Team Tendency : 팀 별로 생성한 코드 클론 확인
- Author Tendency : 개발자 별로 생성한 코드 클론 현황 확인



[그림 18] 코드 클론 제작팀 관리 시각화

4. 평가 및 분석

평가 및 분석은 본 연구에서 제안한 시각화 도구가 사용자의 코드 클론 시각화 실험 관리 판단에 도움이 되었는지를 판단하기 위한 목적으로 이루어진다. 평가를 수행하는 평가자는 시각화 도구를 사용할 수 있는 관리자 혹은 개발자 6명을 대상으로 하였다.

기업에서 코드 클론을 관리하는 전문가를 찾기 어려웠기 때문에, 학교 관계자 중 개발 및 관리 경력이 평균 2년이고 리팩토링에 대한 사전 지식이 있는 사람들로 선정하였다. 평가지는 평가 항목에 대한 점수 표시와 서술로 구성된다. 점수는 항목에 아주 적합하면 5점을, 아주 부적합하면 1점을 부여하는 방식으로 1~5 점까지 선택한다.

비교 평가 대상이 되는 Sonar Qube는 2장에서 소개한 기존 시각화 도구들과 비교 결과, 가장 관리에 적합하다고 판단되었기 때문에 선택하였다.

<표-2> 코드 클론 관리 방법 평가 결과

평가 항목		*평가 점수*
코드 클론 관리 활동	**상세 내용**	
코드 클론 탐지	크기(토큰, 라인 수)	5
	내용(코드) 빈도	4.6
	리팩토링 순위	4.2
	복잡도	3.5
	평 균	4.8
코드 클론 진화 추적	위치	4.8
	모듈 단위에서 분포(커버리지)	3.8
	코드 클론 진화 (변경 횟수, 변화 방향성, 속도)	3.7
	평 균	4.1
코드 클론 리팩토링	변경 영향도, 연관 관계, 계층 관계	5
	개발자 팀과 연관	3.7
	평 균	4.3

<표-3> 코드 클론 관리 도구 평가 결과

평가 항목	*Clone GUIde*	*Sonar Qube*
탐지 (코드 클론 내용 및 위치)	4	3.3
진화 추적 (변하는 코드 클론 파악)	3.5	3.5
리팩토링 (코드 클론 제거 활동)	4.5	2.5

<표-4> 코드 클론 시각화 평가 결과

평가 도구	*평가 항목*		*점수*
Clone GUIde	텍스트	코드 클론 리스트에서 관련된 정보만 표기	4.1
	도표	클래스 다이어그램에 메소드 단위로 표기	4.6
Sonar Qube	텍스트	소스 레벨에서 다른 취약점들과 함께 표기	4.3
	도표	코드 클론 분포도와 트리맵을 제공	2.6

첫 째로 코드 클론 시각화 도구에서 제공해야 할 정보와 관리 활동 만족도를 평가할 수 있다. <표-2>에서 코드 클론 탐지가 가장 높은 필요성을 보였고, 세부 항목을 보면 코드 클론의 정확한 위치 및 크기와 코드 내에서 코드 클론의 변경 영향도 및 계층 관계 파악이 중요함을 알 수 있었다. 도구 비교 평가 <표-3>에서는 탐지와 리팩토링 부분에서 Clone GUIde가 비교적 높은 점수를 받았다.

둘 째는 제공하는 시각화 요소가 관리에 도움이 될 수 있는지를 검증하였다. 특히 <표-3>에서 클래스 다이어그램의 정적 뷰가 시각화 평가 항목 중 가장 높은 점수를 받은 것을 미루어 클래스 다이어그램이 코드 클론을 시각화할 때 적합한 방법임을 알 수 있었다.

평가 후, 전체적인 피드백 의견으로는 Clone-GUIde는 버튼 사용을 줄여 직관적인 이벤트를 원하는 등 화면 구조 및 UI에 개선을 느끼는 의견이 많았다. 반면 Sonar Qube는 시각적인 레이아웃과 화면 배치 완성도가 뛰어나지만 잘 구성된 화면에 비해 사용법이 어려워 정보 간의 관계를 찾기 힘들다는 평이 대부분이었다.

5. 결론 및 향후 연구

본 논문에서는 클론 관리 활동에 필요한 정보를 고찰하고 시각화 도구에 반영함으로써 기존 코드 클론 관리에 사용되는 시각화 방법을 보완하였다.

CCDI에서는 코드 관리에 필요한 정보 축들을 수집하였다. 코드 클론 리스트를 클래스 다이어그램과 연동, 코드 클론을 만든 개발자, 생성된 시간 등의 새로운 정보축을 수집하였다. Clone-GUIde는 CCDI에서 모은 정보를 바탕으로 코드 클론과 클래스 다이어그램을 연동하였고, 코드 클론을 생성과 관련된 내용을 보여주는 시각화 방법을 제안하였다.

하지만 시각화 방법 중 코드 클론 개수 변화 추이 파악 뷰는 코드 클론을 만든 개발자와 생성한 시간 등의 데이터를 수집하여 시각화 하는 한계점이 있다. 전체적인 코드 클론 개수의 변화 방향성만 파악할 뿐, 클론 자체의 진화 추적은 제공하지 못하였다. 현재의 클론과 과거에 발견되었던 동일한 클론의 정확한 연관성을 파악할 수 있는 기준이 없기 때문이었다. 따라서 현재와 과거의 동일 클론의 연관성을 찾는 것은 코드 클론 진화를 추적하는데 가장 중요한 연구과제라고 할 수 있다.

[참고 문헌]

[1] Chanchal K. Roy, Minhaz F. Zibran, Rainer Koschke, "The vision of software clone management: Past, present, and future (Keynote paper)," 2014 Software Evolution Week – IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), pp. 18-33, 2014.

[2] Chanchal K. Roy, James R. Cordy, Rainer Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Journal Science of Computer Programming, Vol. 74, No. 7, pp. 470-495, 2009.

[3] Muhammad Asaduzzaman, Chanchal K. Roy, Kevin A. Schneider, "VisCad: flexible code clone analysis support for NiCad," Proceedings of the 5th International Workshop on Software Clones, pp 77-78, 2011.

[4] M. Fowler, Refactoring: Improving the Design of Existing Code, 1997.

[5] Asaduzzaman Muhammad, Visualization and analysis of software clones, Doctoral dissertation, University of Saskatchewan Saskatoon, 2012.

[6] Chanchal K. Roy, James R. Cordy, "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization," ICPC '08 Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension, pp. 172-181, 2008.

[7] Simone Livieri, Yoshiki Higo, Makoto Matushita, "Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder," 29th International Conference on Software Engineering (ICSE'07), pp. 106-115, 2007.

[8] Tibor Bakota, Rudolf Ferenc, Tibor Gyimothy. "Clone smells in software evolution," 2007 IEEE International Conference on Software Maintenance, pp. 24-33, 2007.

[9] Toshihiro Kamiya, Katsuro Inoue, Yasushi Ueda, Shinji Kusumoto, "Gemini: Maintenance Support Environment Based on Code Clone Analysis," Proceedings of the 8th International Symposium on Software Metrics, pp. 67, 2002.

[10] Md Sami Uddin, Varun Gaur, Carl Gutwin, Chanchal K. Roy, "On the comprehension of code clone visualizations: A controlled study using eye tracking," 2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 161-170, 2015.

[11] CPD, https://pmd.sourceforge.net/cpd.html

[12] SonarQube, http://www.sonarqube.org/

[13] Code2UML, https://sourceforge.net/projects/code2uml/

[14] MariaDB, http://mariadb.org/

웨어러블 어플리케이션 개발을 위한 안드로이드 BLE API 에뮬레이터와 확장성에 관한 연구

문현아 ○ 박수용
서강대학교 컴퓨터공학과
{hamoon,sypark}@sogang.ac.kr

최광훈
전남대학교 전자컴퓨터공학과
kwanghoon.choi@jnu.ac.kr

A Study on Android BLE API Emulator and Its Extension for Wearable Apps

HyeonAh Moon ○ Sooyong Park
Dept. of Computer Science
Sogang University

KwangHoon Choi
Dept. of Elect. and Computer Science
Chonnam National University

요 약

모바일, 웨어러블 기기, 사물인터넷에서 BLE(Bluetooth Low Energy) 기반 통신을 많이 활용하고 있다. BLE 연동 안드로이드 어플리케이션을 개발할 때 반드시 하드웨어가 있어야 하는 문제점이 있다. 본 연구에서는 확장 가능한 안드로이드 BLE API 에뮬레이터를 설계 및 개발하였고, BLE 기기 용 어플리케이션에 적용하여 실제 하드웨어가 없이 개발할 수 있음을 확인하였다. 제안한 BLE API레벨 에뮬레이터는 BLE 기반 통신 시나리오를 소프트웨어적으로 구현만 하면 마치 기기가 있는 것처럼 어플리케이션을 연동할 수 있는 장점이 있다. 두 가지 다른 BLE 기기에 적용하여 이러한 확장성을 확인하였다.

1. 서 론

웨어러블을 위한 어플리케이션을 개발할 때 기기와 연동해서 테스트하는 어려움이 있다. 더욱이 시장 선점을 위해 빨리 출시하려는 경우 웨어러블 기기와 어플리케이션을 동시에 개발해야 하므로 더욱 복잡한 테스팅 방법이 필요하다.

웨어러블 기기와 어플리케이션을 실행하는 모바일 기기를 연결하기 위해 BLE(Bluetooth Low Energy) 통신 방법을 많이 사용하고 있다. 웨어러블과 연동하는 어플리케이션을 테스트하려면 BLE 통신 방법을 고려해야 한다.

상용 웨어러블 기기인 맥박 측정기(HRM3200)를 위한 전용 안드로이드 어플리케이션을 실제 개발하면서 미완성 기기의 안정적이지 못한 BLE 통신 상황에서 테스트하고, 펌웨어 변경으로 자주 통신 규약이 변경되어 테스트에 큰 어려움이 있었다.

기존 개발 방법에서는 이렇게 불완전한 웨어러블 하드웨어 리비전이나 BLE 테스트 킷으로 최소한의 연동 테스트만 가능했다. 예를 들어, BLE 통신에 의존하는 사용자 인터페이스를 테스트하려면 아직 완성되지 않은 BLE 기능을 대체할 모방 코드를 매번 작성해야 한다.

본 연구에서는 웨어러블 하드웨어 없이도 안드로이드 어플리케이션의 개발을 지원하는 안드로이드 BLE API 에뮬레이터를 설계 및 구현한다. 이 에뮬레이터를 활용하여 [그림 1]의 상용 맥박 측정기(HRM3200), BLE 테스트킷(BoT-CLE110)과 각각 연동하는 안드로이드 어플리케이션 개발에 적용하였다. 이 하드웨어 기기들 없이도 BLE 기반 연동 시나리오 대로 어플리케이션이 잘 동작함을 확인하였다. 실제 하드웨어와 연동하려면 어플리케이션의 소스코드를 단지 14~16줄 만 수정하면 된다.

그림 1. HRM3200과 BoT-CLE110의 테스트 킷

BLE 통신 방법은 안드로이드, iOS 모바일에 기본 탑재되어 이미 많이 사용되고 있고, 사물 인터넷과 웨어러블의 중요한 통신 방법으로 활용되고 있기 때문에 이 논문에서 제안한 안드로이드 BLE API 에뮬레이터의 활용도는 매우 높을 것이다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 살펴보고, 3장에서는 개발한 안드로이드 BLE

API 에뮬레이터의 특징과 실제 적용 결과를 제시한다. 마지막으로 4장에서는 결론과 향후 연구 방향에 대해 논의한다.

2. 관련 연구

BLE(Bluetooth Low Energy)는 모바일 폰과 같은 중심 기기와 하나 이상의 주변 기기들 사이의 통신을 위한 무선 통신 프로토콜이다. 기존 블루투스 프로토콜보다 단순한 구조를 갖는 블루투스 저전력 프로토콜은 전송 데이터양이 많지 않은 IoT(사물인터넷) 분야와 저전력이 필수적인 무선 웨어러블 장치에 많이 이용되고 있다 [1,2,3,4].

BLE는 범용속성(GATT: Generic ATTribute) 프로파일을 통해 배터리 레벨, 맥박 수치 등의 다양한 데이터를 주고받는다. GATT프로파일 기반 BLE통신은 일반적으로 5단계로 이루어진다.

- 주변 BLE 기기 스캔
- 찾은 BLE 기기에 연결
- 연결된 BLE 기기가 제공하는 서비스 검색
- Notification 방식으로 데이터 수신을 반복
- BLE 기기 연결 해제

하드웨어와 소프트웨어를 동시에 개발할 때 아직 개발 중인 하드웨어 기기를 에뮬레이션하여 소프트웨어를 개발하는 방법은 특히 임베디드 시스템 등에서 많이 연구되어 왔다 [5,6,7,8,9].

그럼에도 불구하고, BLE 기반 안드로이드 어플리케이션 개발을 위한 BLE 에뮬레이션 라이브러리가 아직 개발되어 있지 않다 [10].

3. 확장 가능한 안드로이드 BLE API 에뮬레이터

안드로이드 BLE API 에뮬레이터의 구조 및 특징을 설명하고, 실제 적용 사례에 대해 논의한다.

3.1 구조 및 특징

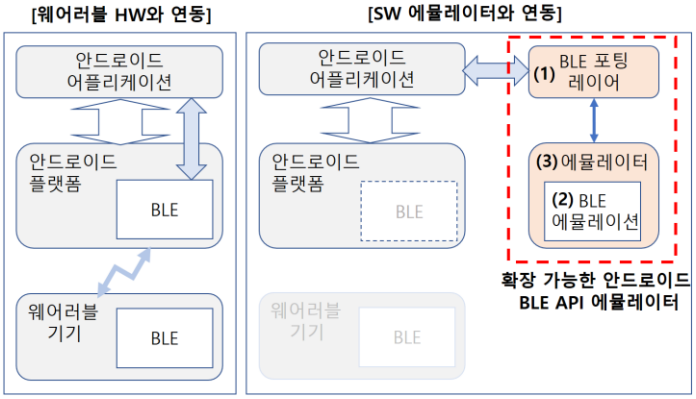


그림 2. 웨어러블 기기와 어플리케이션의 연동

이 논문에서 설계한 확장 가능한 안드로이드 BLE API 에뮬레이터의 구조는 [그림 2]와 같다.

제안한 구조는 크게 BLE 포팅 레이어, BLE 에뮬레이션 확장 모듈, 에뮬레이터로 구성된다. (1) BLE 포팅 레이어는 안드로이드 어플리케이션이 안드로이드 플랫폼의 BLE API 대신 구현한 에뮬레이션 API와 통신하기 위해 필요하다. (2) BLE 에뮬레이션 확장 모듈은 각 웨어러블 기기의 BLE 기반 통신 규약을 소프트웨어로 구현한 모듈이다. (3) 에뮬레이터는 BLE 에뮬레이션 확장 모듈을 마치 하드웨어 장치인 것처럼 어플리케이션과 연결시킨다.

[그림 2]의 안드로이드 어플리케이션, BLE 포팅 레이어, 에뮬레이터를 통해 BLE의 GATT 프로파일 기반의 데이터 통신을 에뮬레이션한다. 이 과정을 [그림 3]의 순차 다이어그램으로 설명한다. 이 순차 다이어그램에서 어플리케이션과 포팅 레이어가 동일한 스레드에서 동작하고, 에뮬레이터는 다른 스레드에서 실행된다. 두 스레드는 서로 메시지를 주고 받으며 비동기적 BLE 통신을 에뮬레이션한다. 주변의 BLE 기기를 검색할 때 에뮬레이터 스레드를 생성한다.

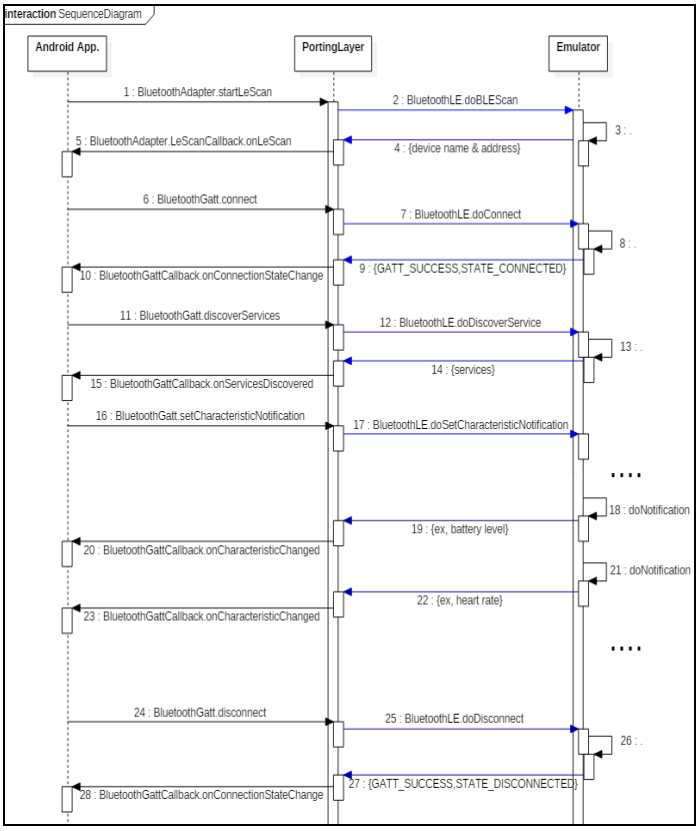


그림 3. 안드로이드 BLE 에뮬레이터 순차다이어그램

순차 다이어그램에서 언급한 BluetoothAdapter 클래스는 BLE 기기를 찾는 기능을 제공하고, BluetoothGatt 클래스는 BLE의 GATT 프로파일을 통해 연결된 BLE 기기에 데이터를 보낼 때 사용하고, BluetoothGattCallback 클래스는 반대로 데이터를 받을

때 사용한다. BluetoothLE 클래스는 BLE 에뮬레이터를 구성하는 클래스로 스레드 간 통신을 제공한다.

[그림 3]의 순차 다이어그램에서 BLE의 GATT 프로파일 기반의 데이터 통신을 에뮬레이션하는 과정은 다음과 같다.

- 주변 BLE 기기 스캔: 1~5번
- 찾은 BLE 기기에 연결: 6~10번
- 연결된 BLE기기가 제공하는 서비스 검색: 11~15번
- Notification 방식으로 데이터 수신을 반복: 16~17번, 18~23번
- BLE 기기 연결 해제: 24~28번

예를 들어, 주변 BLE 기기를 스캔할 때 BluetoothAdapter 클래스의 startLeScan 메소드를 호출하면(1번) BLE 포팅 레이어에서 에뮬레이터의 BluetoothLE의 doBLEScan 메소드가 호출되어(2번) 에뮬레이션하는 BLE 기기의 이름과 주소를(3번) 포팅 레이어를 거쳐(4번) BluetoothAdapter.LeScanCallBack 인터페이스의 onLeScan을 통해 어플리케이션에 전달한다(5번). 찾은 BLE 기기에 연결, 서비스 검색, 기기 연결 해제 과정도 이와 유사하게 진행한다.

Notification 방식으로 BLE 기기로부터 데이터를 수신하기 위해서 먼저 포팅 레이어의 BluetoothGatt 클래스의 setCharacteristicNotification 메소드를 호출하면(16번) 에뮬레이터의 BluetoothLE 클래스의 doSetCharacterisitcNotification메소드가 호출되어 BLE 기기에서 데이터가 변경될 때 마다 Notification하도록 설정한다(17번). BLE 기기 에뮬레이터는 데이터가 변경되면 BluetoothLE 클래스의 doNotification 메소드를 호출하고(18번) 포팅 레이어를 거쳐(19번) 어플리케이션의 BluetoothGattCallback 클래스의 onCharacteristicChanged 메소드를 호출하여 변경된 데이터를 전달한다(20번). 이 과정을 데이터가 변경될 때마다 반복한다.

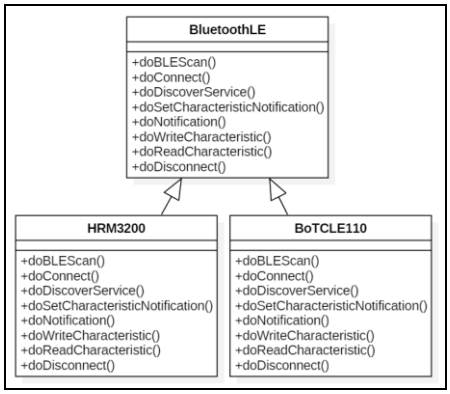


그림4. 클래스 상속으로 확장된 BLE 기기 에뮬레이션

웨어러블 기기의 BLE 통신 규약에 따라 [그림 2]의 (2)를 자유롭게 확장할 수 있는 구조이다. HRM3200 또는 BoT-CLE110 Test kit에서 사용하는 BLE 통신 규약에 맞추어 (2)번 BLE 에뮬레이션 확장 모듈을 개발하면 각 하드웨어 기기의 에뮬레이터가 된다. 어떤 확장을 하더라도 (1)과 (3)은 변경없이 재사용 가능하다.

[그림 4]는 BLE 기기 에뮬레이터의 기반 클래스 BluetoothLE를 상속하여 HRM3200 클래스와 BoTCLE110 클래스를 정의하는 것을 보여준다. 각 파생 클래스는 해당 BLE 기기의 통신 규약을 지원하는 확장된 에뮬레이터이다. doWriteCharacterisitc를 비롯하여 설명하지 않은 메소드들은 BLE 통신 규약을 지원하기 위해 필요한 것이다. 지면 관계상 자세한 설명을 생략한다.

제안한 방법은 다음과 같은 장점이 있다. 첫째, BLE 연동 어플리케이션도 안드로이드 기기 에뮬레이터에서 개발할 수 있다. 일반적으로 안드로이드 어플리케이션을 개발할 때에는 개발 툴(Android SDK)에 포함된 안드로이드 기기 에뮬레이터를 활용하지만 BLE 연동 기능은 이 에뮬레이터에서 제공하지 않는다.

둘째, BLE 통신 웨어러블 하드웨어 개발과 동시에 안드로이드 어플리케이션 소프트웨어를 개발할 수 있다. 이러한 환경에서 BLE API 에뮬레이터를 사용해 완성되지 않은 하드웨어 대신 원하는 통신 규격에 맞추어 자유롭게 작성할 수 있다.

셋째, 제안한 BLE API 에뮬레이터를 사용한 안드로이드 어플리케이션은 향후 실제 기기와 연동할 때 소스 코드를 거의 수정하지 않고 유연하게 적용할 수 있으므로 최종 어플리케이션 완성을 위한 시간을 절약할 수 있다.

3.2 적용 예

개발한 안드로이드 BLE API 에뮬레이터를 두 가지 실제 BLE 기반 기기에 적용하였다.

표 1. BLE 기반 기기

HRM3200	-H3System 개발 -웨어러블 맥박 측정기
BoT-CLE110	-㈜칩센 개발 -BLE 통신 테스트킷

각 BLE 기기의 통신 규약에 따라 [그림 2]의 (2)번 BLE 에뮬레이션 모듈을 작성하였다. HRM3200의 경우, 맥박 측정 시작과 종료를 BLE 통신으로 측정기에 명령을 내리는 시나리오와 측정기에 저장된 맥박 데이터를 모바일로 다운로드 하는 시나리오를 지원하는 통신 규약을 구현하였다. BoT-CLE110의 경우 테스트킷에 부착된 센서로부터 온도, 스위치 눌림 상태 등을 모바일로 읽는 시나리오를 지원하는 통신 규약을 구현하였다.

이와 같이 어떠한 BLE 기반 통신 규약도 BLE 에뮬레이션 모듈로 구현할 수 있기 때문에 안드로이드

어플리케이션과 연동하는 기기의 BLE 기능을 에뮬레이션 할 수 있다.

안드로이드 BLE API 에뮬레이터와 연동해 안드로이드 어플리케이션을 개발한 후, 실제 하드웨어와 연동하도록 하기 위해서는 어플리케이션의 매우 적은 소스 코드만을 수정하면 된다. 두 BLE 기기에 적용한 결과, [표 2]와 같이 기존 코드 대비 약 0.2%로 매우 낮은 비중을 차지함을 확인하였다.

표 2. 안드로이드 BLE API 에뮬레이터 사용하는 경우와 하드웨어를 사용하는 경우 소스 코드 수정이 필요한 라인 수에 대한 실험 결과

	소스 코드 총 라인수	수정 라인수	
		Java 문장	Import 문
HRM3200	7503	2	12
BoT-CLE110 Test kit	5982	2	14

[표 2]에서 보여지듯 실제로 수정이 필요한 Java 문장은 2문장으로 최소 수준의 변경만으로 가능했다. 그 외의 import 문을 수정한 것은 어플리케이션이 사용하는 안드로이드 BLE API를 [그림 2]의 (1)번 BLE 포팅 레이어에서 지원하는 해당 API로 대체하기 위해 필요하였다.

지금까지 BLE API 에뮬레이터를 사용해서 하드웨어 없이 안드로이드 어플리케이션을 개발할 수 있다는 점을 강조했지만 향후 테스팅 분야에도 활용 가능할 것으로 판단된다. 첫째, BLE 기반 복잡한 통신 규약을 고려할 때 에뮬레이터를 활용하면 더 쉽게 반복적으로 테스트할 수 있다. BLE 기반 서비스가 복잡하고 다양해지면서 (예: HRM3200의 맥박 데이터 다운로드) 안드로이드 BLE API는 간단하지만 주고 받는 데이터 형식이 복잡해지는 경향이 있다. 아무리 복잡한 시나리오도 제안한 BLE API 에뮬레이터 기반 클래스를 상속받아 각 기기의 통신 규약에 맞는 시나리오를 기술하기만 하면 된다.

둘째, 웨어러블 기기의 경우 실제 사람이 기기를 착용하고 안드로이드 어플리케이션의 스트레스 테스트를 하는 것을 쉽게 소프트웨어적으로 대체할 수 있다. 상대적으로 테스트 비용도 낮고, 다양한 스트레스 테스트를 할 수 있을 것이다.

4. 결론 및 향후 연구

본 논문에서 확장 가능한 안드로이드 BLE API 에뮬레이터를 설계 및 개발하였고, 두 가지 BLE 기기용 어플리케이션을 실제 하드웨어가 없이 개발할 수 있음을 확인하였다. 이후에 하드웨어 기기와 연동하더라도 에뮬레이터로 개발한 어플리케이션의 소스 코드를 거의 수정하지 않아도 되는 장점이 있었다. 특히 BLE 기반 서비스가 복잡하더라도 BLE API 에뮬레이터에 쉽게 소프트웨어적으로 구현할 수 있다. 마지막으로, 모바일과 사물인터넷에서 BLE 통신을 이용하는 사례가 많아서 이 연구 결과의 활용도가 높을 것이다.

향후 BLE 연동 어플리케이션 개발에서 테스트 방법으로 발전시키고자 한다. 예를 들어, BLE 기기의 통신 규약 명세로부터 테스트 케이스를 자동 생성하거나, BLE 통신 데이터를 랜덤하게 생성해 어플리케이션의 강건성 테스트 연구를 수행하려 한다.

참고문헌

[1] Bluetooth Low Energy Specification AdoptedDocuments.https://www.bluetooth.org/en-us/specification/adopted-specifications/

[2] McGrath, Will; Etemadi, Mozziyar; Roy, Shuvo; Hartmann, Bjoern, Fabryq: Using Phones as Gateways to Prototype Internet of Things Applications Using Web Scripting, EICS '15 Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, 164-173, 2015.

[3] 이주형, 한문석, 비콘 서비스를 사용한 보행자 안전 신호감지시스템의 설계, 정보과학회 컴퓨팅의 실제 논문지 제22권 제11호, 576-582, 2016.11

[4] 이주원, 김우생, Bluetooth Low Energy 기술을 이용한 위치기반 모바일 사물인터넷 제어 시스템, 2015년 한국컴퓨터정보학회 동계학술대회 논문집 제23권 제1호, 231-233, 2015.1

[5] 송현근,윤준호,이종현, 외부 시스템 Mocking 을 통한 웹서버 확장 모듈 테스팅 기법, 한국정보과학회 2011가을 학술발표논문집, 제38권 제2호(B), 128-131, 2011.11

[6] CMock 함수 호출을 흉내내는 라이브러리, https://github.com/ThrowTheSwitch/CMock

[7] Java의 Mockito, http://site.mockito.org/

[8] Freeman, Steve and Mackinnon, Tim and Pryce, Nat and Walnes, Joe, jMock: Supporting Responsibility-based Design with Mock Objects, Companion to the 19th Annual ACM SIGPLAN Conference on Object oriented Programming Systems, Languages, and Applications, 4-5, 2004

[9] Xiaoyin Wang, An Empirical Study on the Usage of Mocking Frameworks in Software Testing Shaikh Mostafa, 14th International Conference on Quality Software, 127 - 132, 2014

[10] http://stackoverflow.com/questions/38436370/hermatic-testing-for-ble-based-android-application?rq=1

--

이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임
(No.NRF-2014R1A1A2053446)

실시간 파일행위 모니터링을 통한 랜섬웨어 침해복구 연구

김재열*
고려대학교 컴퓨터정보통신대학원
pento@hanmail.net

A Study on Ransomware Recovery by Real-time File Behavior Monitoring

Jae-Yeol Kim*
Dept. of Computer Information & Communication, Korea University
pento@hanmail.net

요 약

최근 악성코드는 단 시간 내 변종이 재생산되는 속도가 백신에 의해 대응되는 속도보다 빠르게 나타나고 있다. 고전적인 방어 기법인 시그니쳐를 기반으로 대응방법은 한계가 존재하고 한다. 따라서 백업기반 사전방어 기법들이 연구되고 있다. 2,000년대로 접어들면서 해킹의 목적이 블랙마켓과 결합되면서 하나의 산업으로 발전하고 있다. 특히, 랜섬웨어는 대상파일을 암호화 하는 특성으로 다른 종류의 악성코드처럼 치료나 삭제를 통해 감염대상 파일을 복구할 수 있는 방법이 불가능하다. 따라서, 본 논문에서는 행위기반으로 랜섬웨어의 동작을 파악하고 실시간 예측백업을 통해 랜섬웨어를 침해 이후에도 복구할 수 있는 방법을 연구하는데 목적이 있다.

1. 서 론

2015 년 기준으로 신종 악성코드는 하루 30 만건 이상이 신규로 발생하고 있다[1]. 이 처럼 대량의 신종악성코드를 처리하는 것은 현실적으로 불가능하다. 보안 연구분야에서 대량으로 발생하는 신종 악성코드에 대한 대응이 가장 중요한 기술적 방향으로 연구되고 있다. 클라우드 기반으로 악성코드를 처리하는 방식으로 진단 율 향상에 혁신적인 발전을 하게 되었다

랜섬웨어가 등장하기 전까지는 99% 정도의 진단 율과 행위기반 기법으로 충분히 대응이 가능하였다. 왜냐하면, 일부가 시스템이 악성코드에 감염되더라도 빠른 시간 내 치료가 가능하기 때문에 대응이 가능한 상황으로 인식되었다. 하지만, 최근 유행하는 랜섬웨어의 가장 큰 특징인 대상파일 암호화로 인해 악성코드 감염에 대한 패러다임이 바뀌게 되었다. 랜섬웨어는 Anti-Virus System 에서 누락된 감염된 파일들은 복구 방안이 존재하지 않는 다른점이다. Anti-Virus System 시스템은 아무리 기술적으로 진화를 이루더라도 신종 악성코드를 효과적으로 방어할 수 없고, 항상 복구의 문제는 존재하게 된다.
따라서, 본 연구를 통해 실시간으로 랜섬웨어에 침해된 문서파일을 복구 하는데 연구목적이 있다.

2. 관련연구

2.1 랜섬웨어 행위유형

랜섬웨어를 연구한 결과 랜섬웨어 공격시점에 나타나는 행위를 유형별로 분류하였다.

표 1. 랜섬웨어 행위분류표

구분	ID	행위유형
F(File)	F1	파일 암호화
	F2	파일명 변경
	F3	파일 삭제
	F4	파일 생성
S(System)	S1	RunKey 변경
	S2	윈도우 설정변경
	S3	볼륨쉐도우 변경
	S4	시스템잠금
P(Process)	P1	System Process Injection
N(Network)	N1	CNC Server 접속

본 분류는 2016 년 발생한 랜섬웨어 표본의 행위분류를 통해 항상 발생하는 행위단위를 표본 조사하였다.

2.2 행위유형별 표본조사

2016 년 1 년간 발생한 94 종류의 랜섬웨어 표본을 조사한 결

과 행위유형별 '표 2' 와 같이 행위분포를 확인할 수 있다,

표 2. 랜섬웨어별 행위분포

구분	ID	분포
F(File)	F1	94%
	F2	77%
	F3	3%
	F4	3%
S(System)	S1	3%
	S2	14%
	S3	1%
	S4	3%
P(Process)	P1	1%
N(Network)	N1	10%

본 분석에서는 F1(암호화) 행위를 대상으로 파일백업을 수행한다고 가정한다면 약 6%정도는 F1 을 수행하지 않는 것으로 예상된다.

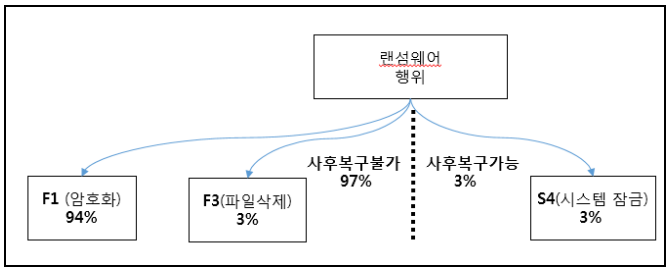


그림 1. 랜섬웨어 행위분류 구성도

상기 '그림 1' 을 통해 암호화를 수행하지 않는 총 6 건의 케이스를 분석해본 결과 MBR 암호화 또는 스크린락커에 해당한다. 따라서, 본 케이스는 원본데이터가 암호화 된 상태가 아니라서 사후 복구가 가능하다. 따라서, 파일이 실제 F1 과 F3 행위에 따라서, 원본이 침해되기 직전에 원본파일을 백업할 수 있다면 아직 알려지지 않은 신종 랜섬웨어 침해에 대해 문서보호가 가능할 것으로 본다.

3. 랜섬웨어 대응모델

3.1 파일엔트로피 판단법

'파일엔트로피 판단법' 은 암호화를 판단하는 통계적 방법으로 C. Shannon 이 제안한 엔트로피 수식을 이용해 정보의 양을 기반으로 암호화를 판별하는 방법이 있다[2].

$$H(X) = \sum_{i=1}^{n} P(i) \log 2P(i) \qquad (1)$$

수식(1)을 적용해 암호화된 실행파일을 측정한 결과 Average Entropy 7.2 이상의 경우로 측정되었다[3]. 본 연구를 통해 측정해본 결과 문서파일 암호화는 Average Entropy 5.3 으로 나타났다.

3.2. 파일헤더 판단법

'파일헤더 판단법' 은 랜섬웨어가 암호화를 수행한 경우 헤더가 바뀌기 때문에 헤더 유효성만 가지고도 정상파일 유무를 판단하는 방법이다. 다만, 파일헤더를 그대로 두고 나머지 영역만 암호화를 수행한 경우 정상파일로 잘 못 판단할 수는 있다. 현재까지는 조사된 랜섬웨어는 '파일헤더 판단법' 으로 대응이 가능하다.

4. 기존 대응방식의 문제점

랜섬웨어는 한 번 침해되면 복구할 수 없다는 특성을 가지고 있다. 따라서, 100%에 근접한 정확도를 가지고 있어야 한다. '파일엔트로피 판단법' 은 90%내외의 정확도를 기록하기 때문에 백신이 무력화된 상황에서 수행되는 백업으로서는 낮은 정확도로 보인다. 또한, '파일헤더 판단법' 은 성능이나 수행시간 측면에서 효율적으로 보이지만, 만약 랜섬웨어가 파일헤더를 보존하고 문서의 중간부분만 암호화하는 경우 쉽게 무력화 된다는 단점을 가지고 있다.

5. 본 논문 제안요약

본 논문에서는 거의 대부분의 랜섬웨어 특징인 암호화 수행 이전에 랜섬웨어 의심행위를 선별적으로 실시간 백업하는 방안을 연구하는 것을 목적으로 한다. 즉, 랜섬웨어가 대상파일을 침해 하려는 바로 직전에 행위를 판별해서 복사본을 안전한 공간에 복사하는 방식을 말한다. 행위단위에 대한 표본조사 결과를 토대로, '표 2' 에서 언급한 F1,F3 행위를 통해 파일변조가 수행되는 파일은 '그림 2' 절차에 따라 백업을 수행하도록 한다.

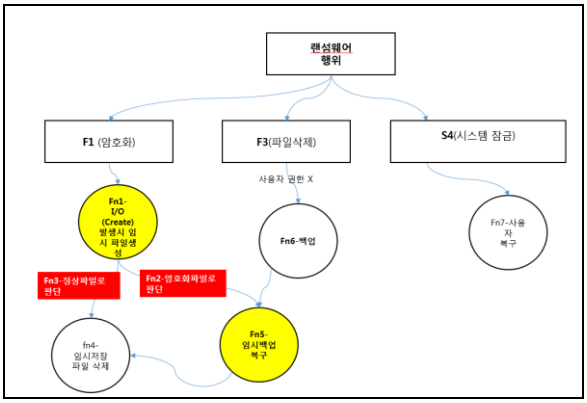


그림 2. 파일 I/O 기반 침해행위탐지 모델

6. 수행모델

상기 모델의 기본적인 구성은 파일 I/O 가 발생하는 시점에 임시로 파일을 저장한다. 저장된 임시파일과 쓰여진

파일과 과를 비교해서 암호화 행위인지 판단하는 기본적 구조를 가지고 있다. 여기서 파일 쓰기 I/O 의 발생시점은 아직 원본파일로 변경 전을 의미한다. '그림 2' 에서 랜섬웨어 행위를 3 가지 단계로 정의하고 F1, F3, S4 에 대해 각각의 행위유형에 적합한 대응계획을 정의할 수 있다.

6.1. 'F1' 암호화 행위판별

본 논문에서 암호화 행위판별은 침해결과 파일에 대한 파일패턴을 검출하는 방식으로 판별하게 된다.
즉, 정상적인 파일의 경우는 해당파일 헤더부분에 정의된 파일에 대한 포멧을 정의함으로써 검증하게 된다. 암호화 판단은 헤더식별, 포맷검증 룰로 판단한다.

검증 룰	방법	대상
파일포멧 판단 법	파일 포맷 유효성 확인	포멧이 공개된 문서파일
파일헤더 판단 법	보호하고자 하는 문서의 헤더 유효성 확인방법	문서파일
엔트로피 판단 법	문서파일 엔트로피 7.2 이상인 경우 암호화 파일로 판다.	일부 파일 조각으로 검증가능.

표 3. 문서파일 암호화 판단법

'파일포멧 판단법'은 파일변경이 완료된 파일의 포멧을 보고 정상적인 파일인지 구분하는 방식으로 판단한다. 아래 MS Office 계열파일의 경우 4byte 헤더값으로 문서포멧을 인식하고 CRC 4byte 를 통해 유효성을 판단할 수 있다. 따라서, 암호화를 비롯해서 정상적이지 않는 경우 CRC 비교를 통해 판단할 수 있다. 또한, CRC 가 없는 경우는 파일의 Offset 값이나 사이즈를 보고 판단할 수 있다.

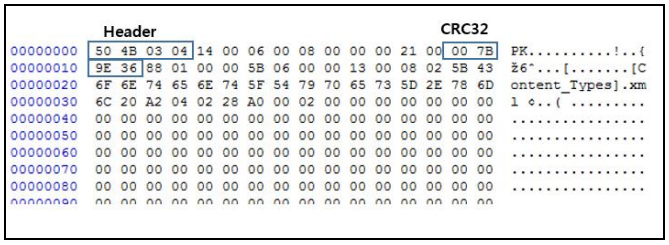


그림 3. MS-Office 계열 파일포맷 판단법

6.2. 'F3' 파일삭제 행위판별

2016년 이전까지는 랜섬웨어가 추구하는 목적에 따라 원본파일을 복구할 수 있는 방안을 제공하였다. 2016년 하반기부터 파일을 삭제하고 사용자를 속이는 새로운 형태의 랜섬웨어가 나타나기 시작했다. '표2'에서는 3%의 작은 비중이긴 하지만 암호화 보다 더 강력한 형태로 보인다. 조사자료 중 2건은 테스트단계 유출로 추정된다. CryptoFinancial(RansCam)은 사용자를 의도적으로 속이기 위해 제작되었다. 또한, 완전삭제를 통해 원천적으로 복구가 불가능하도록 처리되었다. 다만, 암호화가 동반되는 경우 "5.2.1 'F1' 암호화 행위판별"을 통해 복구가 가능하지만, CryptoFinancial(RansCam)처럼 암호화를 수행하지 않는 경우는 다른 형태의 복구 룰을 시도해야 한다. 현재까지는 1건의 삭제형태가 나타났지만 향후 많은 형태로 나타날 가능성이 있다. 아직 삭제만을 수행하는 경우는 시도는 거의 없어서 행위를 특정할 수 없는 경우가 많다

표 5. 삭제행위 판단법

검증 룰	방법	대상
행위 판단 법	사용자권한 이외 권한	문서파일
	완전삭제시도 (2회이상)	

3%의 'F3' 파일삭제시도 랜섬웨어에서 동일한게 검출된 동작은 완전삭제 동작이다. 따라서 동일한 파일을 2 회이상(2 pass) 이상 완전삭제를 수행할 경우 랜섬행위로 유추할 수 있다. 완전삭제의 경우 일반적으로 사용자가 수행한다. 또한, 일반적인 삭제의 경우 복구가 가능하기 때문에 완전삭제 시도는 필수적인 행위로 볼 수 있다.

6.3. 'F3' 파일삭제 행위판별

랜섬웨어에 동작 중 암호화 행위를 동반하지 않고 시스템 잠금 후 비용을 요구하는 경우가 존재한다. 이런 경우는 크게 2 가지 동작으로 분류할 수 있다.

1) MBR 변조 및 암호화
2) 스크린 잠금

상기 2 가지 경우 모두 원본데이터를 복구할 수 있다는 특징을 가지고 있다. 따라서, 'S4' 행위의 복구 방안은 사용자 수동복구로서 시스템적으로 관여하지 않는다. 또한, "1)번 MBR 변조 및 암호화" 의 경우 최신 백신에서 거의 대부분 방어 룰을 가지고 있기 때문에 향후 추가적으로 공격기법이 고도화되거나 변종이 발생할 가능성이 희박하다.

7. 제안의 검증

본 논문에서는 유사기법의 방식과 비교분석을 통한 검증을 수행하도록 한다. MBR 변조의 경우는 대부분의 백신에서 이미 방어처리가 되어있는 상황으로 다시 공격형태가 발생할 가능성이 희박하다. 또한, 발생하더라고 MBR 이외의 데이터 영역을 보존되는 형태라서 충분히 복구가 가능하다. 상기 이유로 인하여 본 논문은 암호화 행위를 중심으로 연구가 되었고 비교실험 또한 암호화를 중심으로 실험하였다.

비교의 대상은 3 개의 모델을 대상으로 하고 진단율을 비교하였다. 검증샘플은 총 4 개로 아래와 같은 특징을 가지고 있다. Cerber 는 암호화가 수행되면 랜덤파일명과 확장자로 가변경된다. 완벽한 암호화를 위해 암호화 진행 시 관련프로세스를 종료시킨다. 샘플로 사용된 Cerber 은 2016 년 3 월에 나

타났으며 확장자가 '.hta'로 변경되는 특징이있다.

1) Tesla Crypto 는 국내에 많이 유포된 랜섬웨어로 취약한 웹페이지 및 이메일 첨부파일로 배포된다. 확장자는 ecc,micr 등으로 변경된다.

2) JuicyLemon 은 2016 년 8 월경 유포된 전형적인 암호화 랜섬웨어다.이메일 첨부파일로 'RESTORE FILES.txt'나 Victims-ID.txt 로 첨부되고 실제로는 JavaScript 파일로 감염시킨다.

3) CryptoXXX 는 2016 년 4 월경 보고되었으며 암호화 수행후 랜덤하게 확장자가 변경된다. regsvr32.exe 으로 실행되고 시작프로그램에 등록하는 특성을 가지고 있다.

표7. 파일헤더 판단법

파일수	샘플	정확도(%)	속도(sec)
1290	Cerber	100	1.763
786	Tesla Crypto	100	0.76
1426	JuicyLemon	100	0.75
1274	CryptoXXX	100	2.74

표8. 파일포멧 판단법

파일수	샘플	정확도(%)	속도(sec)
1290	Cerber	100	1.872
786	Tesla Crypto	100	0.92
1426	JuicyLemon	100	0.87
1274	CryptoXXX	100	2.9

표9. 파일엔트로피 판단법

파일수	샘플	정확도 (%)	속도 (Sec)	엔트로피 값
1290	Cerber	85.2	7.4	5.17
786	Tesla Crypto	97.4	0.5	5.4
1426	JuicyLemon	99	4.4	5.59
1274	CryptoXXX	85.7	4.1	5.16

8. 결론

본 논문에서 연구하는 목적은 아직 알려지지 않고, 기존 백신이 대응할 수 없는 상태에 많은 피해가 발생하고 있다. 따라서, 랜섬웨어의 절대적인 특성을 파악해서 백신이 놓치는 경우도 최종적으로 문서를 복구할 수 있는 최후의 방법으로 연구가 시작되었다. 따라서, 판단의 정화도가 100%에 근접해야 의미가 있다고 볼 수 있다.

'파일엔트로피 판단법'의 경우 통계적 기법으로 90% 내외의 정확도를 가진 것으로 실험결과가 나왔다. 따라서, 본 연구에서 추구하는 100% 근접한 수지에는 맞지 않다고 볼 수 있다. '파일헤더 판단법'의 경우는 빠르고 정확해 보이기는 하지만, 잠재적으로 발생할 수 있는 헤더를 보존한 중간데이터 암호화의 경우 쉽게 무력화 된다는 단점이 있기 때문에 적용하기 힘들 것으로 보인다.

결론적으로, '파일포멧 판단법'을 활용한다면 100% 가까운 정확도를 바탕으로 사전에 정의된 확장자에 대해 침해된 문서를 가장 효율적으로 복구할 수 있는 것으로 연구되었다.

참고문헌

[1] 카스퍼스키랩.(2016)." 2015년에 매일 31만개의 신종 악성코드 탐지" 카스퍼스키 홈페이지, http://news.kaspersky.co.kr/news2015/12n/151210.htm.(2016-1-3방문)

[2] Thomas M. Cover and Joy A. Thomas, "Elements of Information Theory", Second Edition. Wiley Interscisence, pp. 1-16, 2006

[3] Han, S., Lee, K., Lee, S.: Packed PE File Detection for Malware Forensics. In: 2nd International Conference on Computer Science and its Applications, CSA, Korea, 2009

화이트리스트 기반 감염 프로세스 검출 및 감염원 추적

김시론 ○ , 박용범
단국대학교
rlatlfhs@gmail.com ○ , ybpark@dankook.ac.kr

Whitelist-based Detecting infected processes and tracking infections

Siron Kim○ , Yongb Park
Dankook University

요 약

오늘날 컴퓨터의 활용구역은 점점 넓어지고있다. 또한 컴퓨터의 활용을 위협하는 악성코드의 위협도 기하급수적으로 증가함에 따라 컴퓨터를 보호하기위한 노력은 점점 더 많이 요구되고있다. 기하급수적으로 증가하는 악성코드에 대해 모든 경우의 수를 계산하고 그에 따른 방안책을 마련한다는 것은 현실적인 어려움을 겪을 수 밖에 없다. 따라서 본 논문에서는 시시각각 증가하는 많은 수의 악성코드가 사용중인 프로세스들을 감염시키더라도 가용성이 낮지만 차단율이 높은 화이트리스트를 기반한 차단시스템을 이용하여 이러한 프로세스들이 실행되지 못하도록 막고, 감염원을 찾아내는 방법을 제안하고자 한다.

핵심어 : 화이트리스트 , 감염 프로세스 검출 , 감염원 추적

1. 서 론

기술의 발전에 따라 점점 더 컴퓨터는 사회 전반적인 분야에 걸쳐 사용이 늘어나고있다. 하지만 이를 악용하기위한 악성코드의 숫자도 늘어나고 있다. McAfee 2016.06 보고서에 따르면 매 분기마다 새로운 악성코드가 꾸준히 나타나고 있으며, 이에 따라 2016년 Q1분기에 이미 누적 악성코드의 숫자는 550,000,000건을 돌파하였다[1].

악성코드는 계속 증가되는 양상을 보인다. 수많은 악성코드를 사전에 정의하고 차단한다는 것은 불가능에 가깝다고 볼 수 있다. 이에 따라 본 논문에서는 사용자가 이미 사용하고있거나 사용할 프로세스들을 화이트리스트로 작성하고 이 화이트리스트를 통해 이미 사용하는 프로세스들이 감염된 상태로 실행되지 않도록 시스템을 안전하게 보호하고자 한다.

사전에 정의된 악성코드를 차단하는 것을 넘어, 화이트리스트를 통해 감염된 프로세스를 발견하고 해당프로세스의 정보를 가진 로그의 분석을 통해 감염원을 찾고자 한다. 따라서 본 논문에서는 화이트리스트를 작성하여 인가된 프로세스만 실행되는 시스템에 대해 먼저 설명하고, 화이트리스트에서 거부된 프로세스를 발견하여 해당 프로세스가 리스트 내에 있었음에도 불구하고 거절 될 경우 감염 됐다고 판단 후 로그 분석을 통해 감염원을 발견 하고자 한다.

2장은 본 논문의 연구배경으로 화이트리스트, 악성코드 검출 , 감염원 추적에 대해 알아보며 3장에선 본 논문에서 제시하는 아이디어를 모니터링, 대조, 분석 총 3단계에 걸쳐 이야기해보겠다. 4장에서는 본 논문의 예상결론과 발생하게 될 한계점 및 문제점을 이야기하고 향후 연구에 대해 이야기하겠다.

2. 연구배경

2.1 Whitelist

화이트리스트는 허용된 목록만 허가 시키고 그 외에는 전부 비 허용하는 방식이다.

허용된 목록만 허가시키는 특성상 화이트리스트는 가용성이 낮은 특징을 가지고 있다.

본 논문에서는 사용자가 사용할 정상 프로세스의 특징을 통해 만들어진 화이트리스트를 이용하여 악성코드에 감염된 프로세스를 구별하기위해 사용할 것이다[2].

2.2 Malware Detection

악성코드 탐지는 악의적인 동작을 감지하여 시스템을 보호하는 역할을 한다.

악성코드 탐지에는 정상 기반 탐지 기법과 비정상 기반 탐지기법이 존재하는데 본 논문에서는 화이트리스트를 이용한 비정상 기반 탐지기법을 이용하여 악성코드 탐지 기법을 사용한다.

비정상 기반 탐지기법은 프로그램이 정상적이지 않은 경우를 탐지하는 기법으로 화이트리스트에서 허용되지 않은 경우를 비정상으로 보고 그에 따른 대처를 하도록 제안하였다[3].

2.3 Tracking Infection

감염원 추적은 정상 프로세스가 악성코드에 의해 감염 됐을 경우 해당 프로세스를 감염시킨 감염원을 찾고자 하는 행위이다.

본 논문에서의 감염원 추적은 감염된 프로세스와 관련된 로그파일을 분석하여 감염원을 특정하고 감염시간대를 추정하는데 초점이 맞춰 있다.

3. 제안 아이디어

본 논문의 제안 아이디어는 실시간으로 프로세스 테이블을 감시한다.

모니터가 프로세스 테이블을 감시하는 도중에 일어난 테이블의 변화에 따라 새로이 추가된 프로세스를 확인하고 해당 프로세스를 화이트리스트 상의 프로세스와 비교검사를 통하여 악성코드에 감염이 되었는지 검사한다.

검사결과에 따라 정상으로 처리 후 그대로 실행명령을 내리던가, 혹은 악성코드에 감염 됐다고 판단하여 악성코드 감염원을 발견하기위한 로그 분석 단계에 접어들도록 한다.

3.1 프로세스 테이블 감시

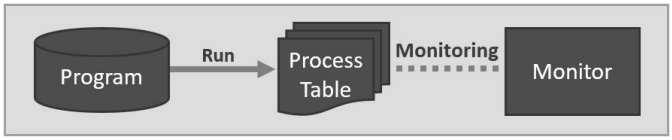


그림 1 Process Table Monitoring

그림 1의 Process Table Monitoring은 Process Table을 감시하기 위한 전체 구성을 나타내고있다.

Monitor를 통해 실시간으로 Process Table의 변화를 지켜보며 새로운 프로그램이 실행 되어 메모리상에 적재된 후 Process Table에 새로운 프로세스로서 등록되면 Monitor는 Process Table에 새로 등록된 프로세스에 대해 검사 작업을 수행하도록 명령을 내린다.

이에 따라 새로이 등록된 Process를 검사 하기 위해 Process Table Analysis에 분석명령을 내리게 된다.

3.2 프로세스 테이블 변화에 따른 처리

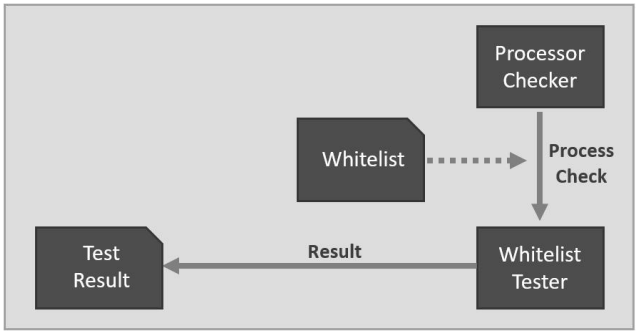


그림 2 Process Table Analysis

그림 2의 Process Table Analysis는 앞단계에서 프로세스 테이블의 변화가 감지됨에 따라 분석신호를 받게 되고, 이에 따라 검사를 시작하게 된다.

명령을 받으면 먼저 Process Checker는 Process Table에 새로이 등록된 Process를 선택하고 선택된 Process는 Whitelist와 함께 Whitelist Tester로 전송된다.

Whitelist Tester는 전달받은 Process와 Whitelist의 정보를 비교하여 해당 프로세스가 Whitelist내에 있는 프로세스 정보와 부합하는지 아닌지를 결정 후 Test Result를 생성한다.

생성된 Test Result에 따라 Process Table Analysis는 새로 추가된 Process가 정상이라고 판단되면 그대로 실행 되도록 한다.

만약 비정상으로 판단되면 해당 프로세스가 악성코드에 감염되었다 판단하고 서비스는 그 즉시 Process Log Manager를 호출하여 해당 프로세스의 로그를 기반하여 감염시기나 감염원을 도출하도록 명령한다.

3.3 Log Analysis

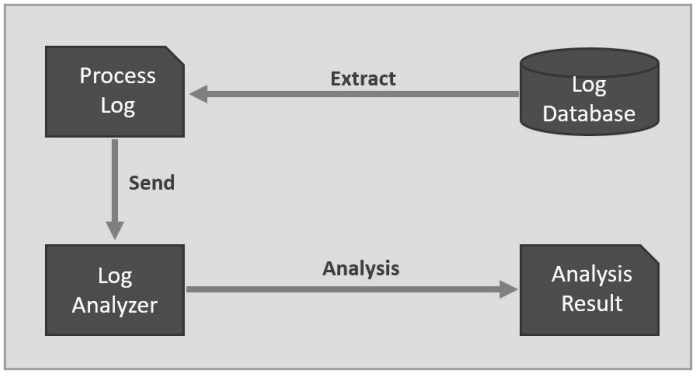


그림 3 Process Log Manager

3.2의 Process Table Analysis 단계에서 악성코드로 의심된 프로세스가 발견 시 그림 3의 Process Log Manager가 동작한다.

먼저 Process Log Manager는 요청을 받으면 해당 프로세스와 관련된 로그파일을 Log Database 에서 찾아서 Log Analyzer에 전달한다.

Log Analyzer는 전달받은 해당 로그에서 가장 최근에 정상적으로 실행된 기간을 탐색한다.

비정상으로 발견된 기간을 지정 후 해당 기간내에 감염된 프로세스에 접근한 프로세스나 감염된 프로세스가 사용한 파일 리스트의 분석을 통해 해당 프로세스를 감염시킨 감염원을 특정하고, 해당 감염원이 접근한 시간대를 분석하여 감염시기를 특정해 낸다.

분석이 끝나면 Process Log Manager는 해당 분석을 Analysis Result 형태로 정리 후 사용자에게 결과를 보여준다.

4. 결론 및 향후연구

본 논문에서는 리스트기반 변조 프로세스 검출 및 감염원 추적에 대해 다루었다. 따라서 화이트리스트를 기반으로 사용될 프로세스가 감염된 상태로 실행되지 않게 하고 감염될 경우 이를 추적할 수 있는 기반의 구현을 통해 늘어가는 악성코드에 대응하고자 하였다.

제안 아이디어의 예상결과는 일반적인 프로세스 감염의 경우 화이트리스트에서 차단은 잘 이루어 지겠지만 문제점은 감염된 프로세스의 로그분석 과정에서 정확한 감염원과 감염시기가 결정될지 미지수이며, 사용될 프로그램이 업데이트 시 바뀐 데이터 값에 대한 처리가 포함되지 않으므로[4], 업데이트를 판단 할 수 있도록 로그분석 과정에서 업데이트 판단이 가능하도록 제작되어야 할 것이다.

또한 은닉된 프로세스의 경우 프로세스 테이블에 등록되지 않으므로[5], 이러한 은닉 프로세스가 감염 시 탐지문제가 발생하며, 악성코드가 화이트리스트 내의 프로세스 고유 값, 예를 들어 해쉬값을 이용 시 해당 해쉬값과 동일하도록 변조를 꾀한다면 본 제안 아이디어는 한계점을 가질 수 있을 것이다.

따라서 향후연구는 제안한 아이디어를 실제로 구현하고 앞서 말한 문제점과 한계점들에 대한 극복방안에 대해 연구할 계획이다.

Acknowledgment

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 지역신산업선도 인력양성사업 성과이며 (No.NRF-2016H1D5A1909989),

미래창조과학부 및 한국인터넷진흥원의 "고용계약형 정보보호 석사과정 지원사업"의 연구결과로 수행되었음 (과제번호 H2101-17-1001).

5. 참고문헌

[1] McAfee, “맥아피 연구소 위협 보고서”, 2016년 6월호, 42p

[2] 박성식, “화이트리스트 기반 전사적 IT자원 보안 관제 시스템”, 한국정보처리학회 추계학술발표대회 논문집, 23권, 2호, 461p

[3] Idika, Nwokedi, and Aditya P. Mathur. "A survey of malware detection techniques." Purdue University 48 (2007).

[4] 이대성, “소프트웨어 업데이트 유형별 위협요소와 안전성 강화를 위한 화이트리스트 구성방안” , 한국정보통신학회 논문지, 18권, 6호, 4p

[5] 고미은, “커널 오브젝트 핸들 테이블을 이용한 화이트리스트 기반의 은닉 프로세스 탐지 방법” , 한국정보보호학회 하계학술대회 논문집, 26권, 1호 , 180p

데이터 통계를 이용한 국가 사이버위협 경보 수준 판단

서형준 ○ 조민경 이상운 배병철

한국전자통신연구원 부설연구소

hjseo@nsr.re.kr, minkyoung@nsr.re.kr, sangounlee@nsr.re.kr, bcbae@nsr.re.kr

Decision of Alarm Level of National Cyber Threat Using Statistical Data

H.J. Seo○ Minkyoung Cho Sang-Oun Lee B.C. Bae

The Attached Institute of ETRI

요 약

국가사이버안전관리규정에 근거하여 국가 차원의 사이버위협 경보가 단계별로 발령되고 있다. 현재 발령되고 있는 사이버위협 경보는 사이버 침해사고와 피해발생 가능성으로 판단되고 있으나 판단을 위한 근거 데이터의 기준이 명확하게 제시되지 못하는 어려움을 가지고 있다. 이러한 문제를 해결하기 위하여 본 연구에서는 국가 차원의 사이버위협 데이터 통계를 이용하여 사이버위협 경보 수준을 판단하기 위한 연구를 수행하였다. 연구에 사용한 데이터는 국가 차원의 사이버위협 데이터로 기존의 연구에서 가지는 데이터 한계를 극복하였으며 통계 분석 방법인 K-평균 클러스터링 방법을 적용하여 점수화하는 방안을 제시하였다. 본 연구에서 제안한 방법을 통하여 현실적인 데이터에 근거한 사이버위협 경보 수준을 동적으로 판단할 수 있을 것으로 기대한다.

1. 서 론

국가 사이버위협 경보는 국가사이버안전관리규정 제11조에 근거하여 파급영향, 피해규모 등을 고려하여 정상, 관심, 주의, 경계, 심각의 단계로 규정하고 있다[1]. 사이버 침해사고의 기준에 따라 위기 경보를 발령하는 부분은 원칙적으로는 간단하게 보이지만 실제적으로 적용하는 데 있어서는 몇 가지 문제점을 가지고 있다[12]. 사이버 침해사고의 기준이 분야별로 상이하여 공통된 기준을 사용하고 있지 않았다. 2016년 하반기부터 국내 사이버 침해사고를 담당하는 기관들에서 공통으로 사용할 수 있는 국가 사이버 침해사고 통계 기준을 마련하고 이를 적용하는 연구가 뒤늦게나마 진행되고 있다[2]. 또 다른 문제점으로는 무엇보다도 각 단계의 경보가 발령하는 원칙은 가지고 있으나 이 경보를 어느 정도의 기간까지 유지하는지에 대한 기준이 명확하지 않은 점이 존재한다. 국가 사이버위기 경보를 피해발생 가능성의 관점에서 보았을 때는 조금 더 불확실한 면이 존재한다. 위험도가 낮은 웜, 바이러스, 해킹기법, 보안 취약점과 높은 웜, 바이러스, 해킹기법, 보안 취약점을 구분하는 기준이 존재하지 않으며 주의 단계로 상향 발령되기 위한 조건인 다수 기관으로 확산될 가능성이나 경계 단계의 대규모 피해로 발전될 가능성을 판단하는 근거가 불확실하다는 점이다. 이러한 문제들을 해결하기 위해서 본 연구에서는 데이터 통계를 기반으로 사이버위협 경보 단계를 판단하는 방안을 제안하고자 한다. 본 논문의 2장에서는 국가 사이버위협 경보 수준 판단과 관련된 기존 연구를 고찰하고 3장에서는데이터 통계치 분석을 이용한 경보 수준을 판단하는 방안과 이를 적용한 결과를 나타낸다. 4장에서는 결론과 향후 연구에 대한 내용으로 3장에서 기술된 데이터의 점수화를 통해서 경보 수준을 판단하는 방법과 추가적으로 추진되는 연구를 포함하여 기술하였다.

2. 기존 연구

기존의 문호건 등에 의해서 수행된 연구에서는 네트워크 전체위협 중에서 실제 네트워크 내부 자산에 영향을 주는 위협의 비율을 통해 위험도를 산정하고 위험도가 특정 임계치를 넘는 경우에 경보를 생성하도록 하였다[3]. 김성원 등의 연구에서는 문호건의 연구에서도 제시한 바 있는 자산의 중요도, 위협, 취약성 항목들을 확장하여 가치, 가능성, 통제, 요구의 항목을 추가하였다[4]. 이기혁 등의 연구에서는 위험도를 자산의 중요도, 위협, 취약성의 곱으로 계산하는 방법을 이용하여 최종적인 점수를 산출하였다[5]. 김기영 등이 수행한 연구에서는 공격위험도(30점), 1일 보안이벤트 발생빈도(20점), 피해기관 중요도(10점), 피해시스템 중요도(10점), 트래픽 이상 유무(20점), 피해복구 난이도(10점)를 통해 좀 상위 차원의 위험도를 산정한다[6]. 전덕조 등에

의해서 수행된 연구에서는 경보의 관점에서 사이버 위협 예측을 수행하지 않았지만 빅데이터 기술을 활용한 모델을 제시하고 있다[7]. 기존의 연구들에서 나타난 문제점을 종합하면 국가 차원에서 사이버위협 경보를 판단하는 데는 한계가 있으며 실제적인 데이터를 이용하지 못했다는 한계가 존재한다. 따라서 본 연구에서는 기존의 연구의 한계를 해결하기 위해서 본 연구에서 수행되는 데이터는 실제 데이터를 사용한 연구를 수행하였으며 국가 차원에서 전체적이고 장기간인 데이터를 분석하여 사이버위협 경보 수준을 판단할 수 있도록 하였다. 특히 사이버 침해사고가 발생하지 않은 정상과 관심 단계에서 위협이 발생할 수 있는 가능성을 판단하는데 실제 데이터 통계치를 분석하여 위협 경보 수준을 발생할 수 있도록 하였다.

3. 데이터 통계치 분석을 이용한 경보 수준 판단

네트워크 유해 트래픽 탐지 분야에서 주성분 분석, 시계열 예측, 서포트 벡터 머신, 클러스터링 등의 통계 분석 방법을 차용한 연구들이 수행되어 왔다[8]. 이 중 클러스터링 방법은 데이터들의 유사성을 비교하여 비슷한 속성을 가지는 데이터들을 하나의 그룹으로 묶는 기법으로 국가 차원의 사이버위협 경보를 발령하기 위한 수준을 판단하는데 사용할 수 있다. 즉, 국가 차원에서 사이버위협에 대한 다양한 데이터가 존재할 때, 이 데이터들을 몇 개의 클러스터링으로 분류하여 위협의 수준으로 사용할 수 있는 점에 착안을 한 것이다. 클러스터링 방법은 비 계층적 클러스터링과 계층적 클러스터링으로 나뉘는데 본 연구에서 차용한 비 계층적 클러스터링 방법은 분할 영역 K개를 지정하여 데이터를 분할하는 방법이다[9]. 국가 차원의 사이버위협 수준을 점수화하는데 5단계나 5점으로 하는 경우에는 영역(K)을 5로 분류하면 되고 10단계나 10점으로 하는 경우에는 영역(K)을 10으로 분류하면 된다. 물론 분류된 영역을 점수화하는 단계에서 5점 척도나 10점 척도로 사용하지 않고 좀 더 큰 단위의 점수를 부여하거나 넓은 범위의 분류를 사용할 수도 있다. 국가 차원의 사이버위협 경보를 발령하기 위한 수준 판단에서 데이터를 분석하여 각각의 수준을 잘 구분할 수 있는 클러스터링과 수준 판단을 위한 다른 파라메터(Parameter)들과의 상대적인 가중치에 따라서 영역과 점수화는 변경할 수 있다.

본 연구에서는 2016년 1월1일부터 5월31일까지 국가 차원의 악성코드 감염규모, 분야별 탐지 현황 데이터, 7월 31일까지의 특정국가 공격 건수에 K-평균 클러스터링을 적용하여 각 항목별로 10점 단위의 점수를 부여할 수 있도록 하였다. 경보발령을 위한 최종적인 점수는 각 항목의 정규화에 가중치를 곱하여 산출하게 된다.

4. 결 론

본 연구에서는 국가차원의 악성코드 감염규모, 분야별 탐지현황, 특정 국가의 공격 건수에 K-평균 클러스터링을 적용하여 정량적인 점수를 획득할 수 있는 방안을 제안하고 이를 실제 데이터에 적용하였다.

본 논문에서 K-평균 클러스터링에 사용한 데이터는 2016년에 발생한 데이터를 사용하였기 때문에 이후에 추가적으로 축적되는 데이터를 사용하여 배점을 최신화할 예정이지만, 점수화를 하는 방법은 같은 방법이 적용되어 배점이 된다. 또한, 국가 사이버위협 경보 수준 판단을 위해서 국가 차원에서 수집할 수 있는 데이터 항목의 추가적인 발굴과 기존의 연구에서 제시된 바 있는 비정량적인 데이터의 영향[10, 11] 등은 향후에 추가적으로 연구할 예정이다.

[참고문헌]

[1] 국가사이버안전관리규정, 대통령훈련 제316호, 2013.9.2. 시행

[2] 서형준 등, 존 게링의 방법을 이용한 국가 사이버 침해사고 통계 개념 정립 방안 연구, IT서비스학회 추계학술대회, pp. 289-292, 2016. 11.

[3] 문호건 등, 취약점과 위협의 상관성 분석을 통한 네트워크 위험 조기경보 시스템 설계, 정보보호학회지 15(1), pp. 23-32, 2005. 2.

[4] 김성원 등, 네트워크 기반의 실시간 위험관리를 위한 위험분석 및 평가 방법 연구, 정보과학회 학술발표논문집 34(1D), pp. 29-34, 2007. 6.

[5] 이기혁 등, 사이버 환경에서의 침해사고대응을 위한 위험도 산정 및 실시간 경보생성에 대한 연구, 정보보호학회지 18(5), pp. 112-124, 2008. 10.

[6] 김기영 등, 사이버 위협 사전인지를 위한 위협 정량화 기술, 정보보호학회지 22(8), pp. 15-20, 2012. 12.

[7] 전덕조 등, 빅데이터 기술을 활용한 사이버 위협 예측 분석 모델, 정보기술학회논문지 12(5), pp. 81-100, 2014. 5.

[8] 신동혁 등, K-평균 클러스터링을 이용한 네트워크 유해 트래픽 탐지, 통신학회논문지 41(2), pp. 277-284, 2016. 2.

[9] G. Munz, et. al., Traffic anomaly detection using k-means clustering, GI/ITG Workshop MMBnet 2007, Hamburg, Germany, 2007. 9.

[10] 서형준 등, AHP를 이용한 국가주도의 사이버위협 예측 시스템, 정보과학회 학술발표논문집, pp. 149-151, 2015. 6.

[11] 김완주 등, 사이버 방어작전 프레임워크 기반의 공격그룹 분류 및 공격예측 기법, 정보과학회논문지: 컴퓨팅의 실제 및 레터 20(6), pp. 317-328, 2014. 6.

[12] 국가사이버안전센터, http://www.ncsc.go.kr

AOP를 활용한 모바일 클라이언트와 서버간 민감 정보 전송의 동적 모니터링 방법

최윤석○, 최은만
동국대학교 컴퓨터공학과
ysmanse@gmail.com, emchoi@dgu.ac.kr

A Method of Dynamic Sensitive Data Monitoring Between Clients and Server Using Aspect-Oriented Programming

Yunseok Choi○, Eun Man Choi
Dongguk University

요 약

모바일 애플리케이션은 사용자의 개인정보 등 보안에 민감한 데이터를 많이 사용하고 있으며 보안 위협에 쉽게 노출된다. 따라서 모바일 보안 취약성의 감지 및 해결 방법에 대한 연구가 필요하다. 이 논문에서는 모바일 통신의 핵심이 되는 클라이언트와 서버 사이의 보안성을 모니터링 하는 방법을 제안한다. 일반적인 보안 모니터링 방법은 시스템의 형상에 따라 모니터링 하기 위한 특별한 준비가 필요하지만 본 논문에서는 AOP를 활용하여 클라이언트와 서버의 시스템 형상에 관계 없는 효과적인 모니터링에 대한 방법을 제안한다. 클라이언트에서 서버에 데이터를 보내는 지점을 Point Cut으로 선언하여 독립적인 애스팩트 클래스를 생성한다. 이러한 AOP를 활용하면 개발된 소스코드의 수정 없이 애플리케이션 내의 모든 통신에 대한 모니터링을 가능하게 하여 개인정보 등 보안에 민감한 데이터들이 어떤 형태로 서버에 전송되고 있는지 모니터링 할 수 있다.

1. 서 론

2007년 iOS를 탑재한 아이폰의 출시에 이어 안드로이드, 바다, 윈도우와 같은 다양한 플랫폼의 스마트폰이 출시되었다. 현재는 세계적으로 스마트폰의 보급률이 70%에 육박하며 스마트폰 애플리케이션을 통해 PC로 수행할 수 있는 작업을 거의 모두 수행할 수 있다[1]. 이에 따라 스마트폰 애플리케이션을 통한 민감한 정보 및 개인 정보 침해 사고는 해마다 늘어나는 추세이며 사용자의 개인 정보보호에 대한 위협은 지속적으로 심화되고 있다.

대부분의 애플리케이션은 서버와 통신을 통해 사용자에게 여러 기능을 제공한다. 이 때 사용자의 개인정보 등 보안에 민감한 많은 데이터들이 특별한 조치를 통하지 않고 전송되는 과정에서 많은 보안 사고가 발생하게 된다. 이같은 취약점은 국제 웹 애플리케이션 보안 연구 단체(OWASP)에서 2016년 모바일 애플리케이션 취약점 TOP10 안에 '민감한 정보의 평문 전송'으로 선정될 정도로 많은 애플리케이션의 취약점으로 발견된다[2].

사용자의 민감 정보는 신상정보나 의료정보, 생체정보, 금융 재산 정보와 같은 정보주체의 사생활을 현저히 침해할 우려가 있는 개인의 정보들이다. 이같은 애플리케이션 사용자 개인의 민감정보들은 사용자의 허가를 통해 수집 되어야 하며 수집된 데이터들은 안전하게 전송되고 보관되어야 한다.

모바일 애플리케이션에서 민감 정보의 평문 전송을 점검하는 방법은 크게 정적 분석과 동적 분석 방법이 있다. 정적 분석은 애플리케이션을 실행시키지 않고 소스코드를 분석한다[3]. 정적 분석은 소프트웨어의 개발 단계에서 주로 사용된다. 그렇기 때문에 개발된 소프트웨어의 소스코드 내에 산재되어 있는 서버 통신과 관련된 모든 코드를 찾아내어 평문 전송에 적합하지 않은 데이터라는 것을 판단하고 이를 수정하기란 매우 어렵다.

한편 실시간으로 소프트웨어를 구동하여 확인하는 동적 분석 방법을 사용하면 비교적 간단하다. 민감한 데이터를 입력 받아 사용되는 기능적인 부분을 설계서에서 확인하여 해당 기능을 실제로 수행하여 민감한 데이터가 서버로 어떻게 전송되는지 쉽게 확인할 수 있다. 동적 분석 방법은 모의 해킹 툴인 BurpSuite를 활용하여 배포된 모바일 클라이언트 애플리케이션에서 전송되는 데이터를 확인 하는 방법이 있다.

별도의 툴을 사용하는 방법은 먼저 프록시 서버를 구성하고 모바일 클라이언트에서 구성된 프록시로 연결하여 통신할 수 있도록 모바일 클라이언트용 인증서를 발급하여 설치해야하는 여러 복잡한 과정이

필요 하다. 또 동적 분석 방법으로 소스 코드 상에서 서버와 통신하는 부분 모두를 찾아 로깅 기능의 코드를 삽입하여 모니터링 하는 방법이 있다. 이와 같은 동적 분석 방법은 정적 분석에 비해 정확도가 높은 반면 분석 기능이 있는 코드를 수정하는데 많은 비용과 노력이 요구된다[4].

본 논문에서는 iOS 애플리케이션에서 서버에 전송되는 데이터를 유지보수성과 확장성이 뛰어난 AOP(Aspect-Oriented Programming)를 이용하여 개발된 소스코드의 수정 없이 구현 가능한 동적 모니터링 방법을 제안한다.

2. 관련 연구

2.1 AOP(Aspect-Oriented Programming)

AOP는 사용자의 핵심 요구사항을 핵심 모듈과 별개로 보조적인 횡단 모듈로 분리할 수 있다[5]. 횡단 모듈로 따로 분리된 애스팩트 클래스를 활용하여 개발된 소스코드에 영향을 미치지 않고 분리 개발 및 기능 추가가 가능하다. 그림 1은 AOP의 대표적인 핵심 관심과 횡단 관심의 예 이다. 핵심 관심으로는 계좌이체, 입출금, 이자 계산과 같은 소프트웨어의 주요 핵심 기능이다. 횡단 관심은 모든 핵심 관심에 필요한 로깅, 보안, 트랜잭션이 각 핵심 기능에 횡단으로 흩어져 있다. AOP는 이러한 핵심 관심에 흩어져 있는 횡단 관심에 대한 코드를 독립적으로 구현하고 실제 핵심 관심의 기능의 소스코드를 수정하지 않고 각 핵심 관심 기능에 필요한 횡단 관심 기능을 추가 할 수 있다.

본 논문에서는 iOS 애플리케이션에서 각 클래스에 산재되어 있는 서버 통신 기능을 핵심 관심으로 정의하고 통신에 보내지는 데이터를 로깅하는 기능을 횡단 관심으로 구현하여 민감 데이터의 평문 전송에 대한 모니터링을 구현한다.

그림 1 AOP의 핵심관심과 횡단 관심의 예

2.2 Method Swizzling

iOS는 아이폰용 애플리케이션을 개발하는데 Objective-C를 사용한다. C언어에서 스몰토크 스타일의 메세지 구문을 추가한 객체 지향 언어로써 맥킨토시 운영체제인 OS X 와 아이폰 운영체제인 iOS에서 사용되고 있다[6]. 이러한 C언어 기반의 Objective-C의 경우 런타임의 동적인 특성을 갖고있다. 그러한 특성 때문에 런타임상에서 실행되는 메서드를 교체 혹은 확장 하는 것이 가능하다. 이를 Objective-C에서는 메서드 스위즐링이라 부르며, 이러한 메서드 스위즐링 방법을 활용하여 Objective-C에서 AOP를 편리하게 적용할 수 있다[7,8]. 본 논문에서는 메소드 스위즐링을 보다 편리하게 사용할 수 있도록 제공해주는 Aspects라는 Objective-C 라이브러리를 사용하여 실험을 진행한다.

3. AOP를 이용한 데이터 전송 모니터링 방법

3.1 iOS의 HTTP 데이터 전송 모듈

iOS 에서 서버와 HTTP 통신을 위해 SDK 에서 제공하는 NSURLConnection 오브젝트 클래스를 사용해야 한다. NSURLConnection은 herader, body, url, method를 서버 구성에 따라 정확하게 설정하여 사용해야 통신이 가능하다. 이러한 번거로움으로 많은 개발자들은 NSURLConnection 오브젝트 클래스를 직접 사용하지 않고 좀 더 편리하게 사용할 수 있도록 만들어진 통신 관련 오픈소스 라이브러리를 사용한다. 그림2와 같이 이러한 오픈소스 라이브러리는 NSURLConnection을 랩핑하여 기본적인 header와 body 정보를 설정해주어 개발자는 통신에 필요한 URL과 Parameter들만 입력하여 사용 한다. 이와 같이 다양한 환경의 iOS 애플리케이션 소스코드에 동일하게 적용 가능 하도록 AOP를 활용하여 서버와 HTTP 통신의 핵심 부분인 NSURLConnection 오브젝트 클래스를 위빙 포인트로 사용 한다. 그림3과 같이 NSURLConnection 클래스에서 HTTP 데이터 전송 요청에 사용되는 initWithRequest:delegate:Immediately 메서드를 포인트컷으로 선언하여 iOS 모바일 클라이언트에서 데이터가 어떻게 전송 되는지 모니터링 한다.

그림 2 오픈소스 라이브러리의 구성

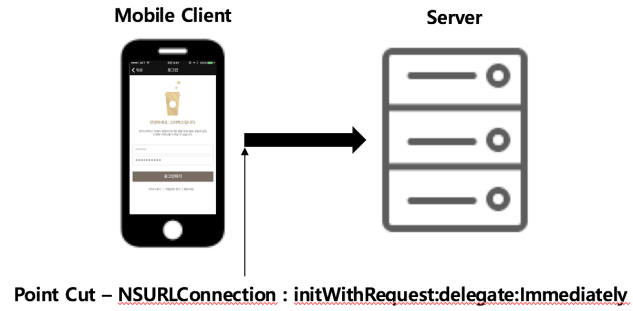


그림 3 iOS의 HTTP 데이터 전송 모듈에 대한 AOP 적용 방법

3.2 모니터링 애스팩트 포멧

본 논문에서는 iOS 에서 AOP를 사용하기 위해서 메소드 스위즐링을 활용하여 사용하기 쉽게 만들어진 Aspects 라이브러리를 사용한다[9]. JAVA에서 사용하는 AspectJ 와 마찬가지로 간단하게 활용할 수 있다. 그림 4 와 같이 제공되는 aspect_hookSelector 를 통해 포인트 컷 지점을 지정하고 withOption을 통해 조인포인트를 지정할 수 있다. 블록 구문으로 이루어진 부분은 AspectInfo로 호출된 메서드에서 사용된 파라메터들의 정보들이 리턴 되어 어드바이스 구현 시 사용이 가능하다.

```
NSURLConnection aspect_hookSelector:
    @selector(initWithRequest:delegate:startImmediately:)
          withOptions:AspectPositionBefore
          usingBlock:^(id<AspectInfo> aspectInfo) {
                    //Advice
          } error:NULL];
```

그림 4 Aspects 포맷

3.2 모니터링 애스팩트 구현

횡단 관심사로 분리한 iOS 애플리케이션에서 서버로 데이터 전송 시 데이터 로깅에 대한 부분은 그림 5 과 같이 구현하였다. 애스팩트 클래스가 한번만 호출되도록 싱글턴으로 작성을 하였고 모든 통신이 일어나는 때를 감지하기 위해 NSURLConnection 오브젝트 클래스의 initWithRequest:delegate: Immediately 메서드를 포인트컷으로 활용하였다. iOS SDK에서 HTTP 데이터 전송을 위해서는 NSURLConnection 클래스의 initWithRequest 메소드를 사용해야 한다. 해당 클래스와 메소드를 포인트컷으로 선언 함으로써 기 개발된 소스코드가 개발 편의를 위해 어떠한 오픈소스 라이브러리를 사용하여 통신 모듈 기능을 개발하였다 하더라도 모니터링이 가능하다. AspectInfo 오브젝트 클래스 변수로 리턴되는 포인트컷 메서드 호출 시 사용되는 파라메터들을 로깅하기 위해 aspectInfo.arguments 배열에서 서버로 요청된 URL과 모든 파라메터들을 로그에 남기도록 구현하였다.

```
@implementation AspectManager
+ (AspectManager *) sharedInstance {
   static AspectManager *instance = nil;
   static dispatch_once_t onceToken;
   dispatch_once(&onceToken, ^{
      instance = [[self alloc]init];
   });
   return instance;
}
- (id)init {
   self = [super init];
   if(self != nil) {
      [self addOriginRequestMonitoringAspects];
   }
   return self;
}
- (void)addOriginRequestMonitoringAspects {
   [NSURLConnection aspect_hookSelector:
      @selector(initWithRequest:delegate:startImmediately:)
      withOptions:AspectPositionBefore
      usingBlock:^(id<AspectInfo> aspectInfo) {
         NSLog(@"#####################################");
         NSMutableURLRequest *request =
         [aspectInfo.arguments objectAtIndex:0];
         if (request) {
            NSString* bodyStr;
            bodyStr = [[NSString alloc] initWithData:request.HTTPBody
            encoding:NSUTF8StringEncoding];
            NSLog(@"HTTP Body datas : %@", bodyStr);
            NSLog(@"URL : %@", request.URL.absoluteString);
         }
         NSLog(@"#################################");
         } error:NULL];
}
@end
```

그림 5 전송 데이터 모니터링을 위한 코드 구현

4. 실험 결과

본 실험에서는 현재 서비스 중인 스타벅스 애플리케이션에 구현된 애스팩트 클래스를 추가하여 진행하였다. 스타벅스 애플리케이션은 사용자 편의 기능으로 리워드 관리 기능과 음료 및 메뉴 주문 기능을 제공한다. 그 중 로그인과 스타벅스 카드 등록 시 서버에 데이터를 어떻게 전송하는지 확인하였다. 그림 6은 로그인이 수행될 때 표시된 전송 데이터 로그, 그림 7은 스타벅스 카드 등록 시 표시된 전송 데이터 로그이다. 카드 등록과 로그인이 수행될 때 사용자의 아이디, 비밀번호, 카드 번호 등 민감한 사용자 정보들이 어떻게 전송되고 있는지 표시하였다. 두 기능들이 수행될 때 서버로 전송되는 데이터가 평문으로 전송되고 있다는 것을 알 수 있다.

```
2016-12-20 02:58:07.711095 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(38)>
#####################################################################
2016-12-20 02:58:07.711416 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(46)> HTTP Body datas : {"app":
{"userId":"ysmanse","password":"chldbstjr1","deviceId":"4100bcb24bd94161a
f80b8f8897b40ce","pushId":"0","osType":"1","osVer":"10.1.1","loginType":"
M"}}
2016-12-20 02:58:07.711624 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(47)> URL : https://msr.istarbucks.co.kr:6443/appif/auth/
login.do
2016-12-20 02:58:07.712271 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(50)>
#####################################################################
```

그림 6 로그인 데이터 전송 로그

```
2016-12-20 03:05:43.623885 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(38)>
##################################################################
2016-12-20 03:05:43.624055 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(46)> HTTP Body datas : {"app":
{"pin":"12345678","cardNumber":"1234567890123456","cardNickname":"카드 등
록"}}
2016-12-20 03:05:43.624154 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(47)> URL : https://msr.istarbucks.co.kr:6443/appif/card/
registration.do
2016-12-20 03:05:43.624237 Starbucks_iOS_Renewal[353:36637] <0x17000e6a0
AspectManager.m:(50)>
##################################################################
```

그림 7 카드 등록 데이터 전송 로그

서버 전송 기능을 핵심 관심으로 설정하고 전달되는 데이터 로깅을 횡단 관심으로 구현하여 모듈화 하였다. 모니터링을 위해 구현된 코드의 라인 수는 33라인이며 표1과 같이 소프트웨어에 산재 되어 있는 모든 통신 수행 코드에 로깅 기능을 추가 하는 방법보다 추가된 코드와 수정된 클래스가 압도적으로 줄어든 것을 알 수 있다. 여러 클래스에 산재 되어 있는 통신 모듈을 찾아내어 직접 소스코드를 수정하였다면 개발자의 실수로 인한 기능 누락과 많은 비용이 소모되었을 것이다. 또한 클라이언트와 서버 간의 데이터 통신 모니터링에 주로 활용되는 Burp-Suite는 프록시 구성과 iOS 에서 구성된 프록시에 접근하기 위해 CA 인증서를 설치해야 하는 번거로움을 그림8 같이 AOP를 활용하여 구현된 클래스만 추가하여 빌드만 수행하면 별도의 설정할 필요 없이 간결하게 모니터링 할 수 있도록 구현하였다.

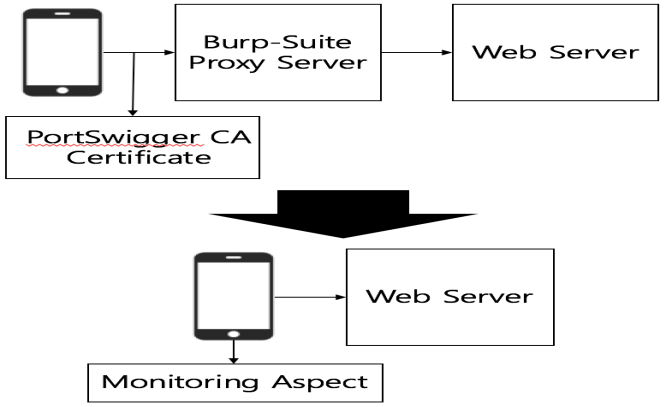


그림 8 Burp-Suite활용과 AOP활용 비교

소프트웨어 전반에 추가해야하는 기능은 해당하는 모든 클래스를 일일이 살펴보고 추적하여 필요 지점마다 직접 코드를 추가 및 수정을 해야하는 번거로움이 있다. 이러한 작업들로 소프트웨어의 품질이 저하되고, 이해 가능성이 까다로우며, 개선의 어려움으로 유지보수 비용도 증가하게 된다.

표 1. 서버 전송 데이터 모니터링 방법 비교

모니터링 방법	추가 라인	변경 클래스	추가 클래스
일반적인 로깅 코드 추가	273	99	0
AOP 활용	33	0	1

5. 결론 및 향후 연구

본 논문에서는 AOP를 활용하여 모바일 클라이언트와 서버간 실시간 데이터 통신 모니터링을 효율적으로 구현하는 방법을 제안 하였다. 본 논문에서 제안한 방법과 실험을 통한 결과는 다음과 같다. AOP활용하여 애스팩트 클래스를 기 개발된 소스코드의 수정 없이 구현하여 유지보수성을 향상 시켰다. 또한 코어 통신 모듈을 포인트컷 하여 같은 플랫폼의 어떠한 소스코드에 이식 하여도 동작하도록 구현하여 이식성을 향상 시켰다.

향후 진행할 연구로는 클라이언트에서 전송되는 데이터의 모니터링 뿐만 아니라 전송되는 데이터를 자동 암호화 할 수 있도록 구현 할 것이다. 암호화된 데이터를 수용하기 위해 서버에도 개발된 소스코드에 영향 없이 클라이언트로부터 받은 암호화된 데이터를 Decrypt 할 수 있는 애스팩트 클래스를 구현하여 보안성을 확보할 수 있도록 연구를 진행할 계획이다.

참 고 문 헌

[1] Gertner, “Worldwide market share smarphone operating systems”, http://www.gartner.com/newsroom/id/3415117.

[2] OWASP, “The list bellow represents a release candidate of the OWASP Mobile Top Ten 2016”, https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10.

[3] D.Binkley, “Source Code Analysis: A Road Map”, 2007 Future of Software Engineering, pp.30-45

[4] 최은만, 서광익, “보안 취약점 검사를 위한 AOP 기반의 동적 분석”, 한국정보과학회논문지, pp.2-4, 2010.

[5] Kiczales, Gregor, G Lamping A Mendhekar, C Maeda, C Lopes, J Loingrier, J Irwin, Aspect-Oriented Programming, Proc. Of ECOOP, Vol.1241, pp.220-242, 1997.

[6] Stephen G. Kochan, Programing Objective-C,Addiison-Wesley Professional, 2013.

[7] Kyle Richter, Joe Keeley, iOS Components and Frameworks : Understanding the Advanced Feateres of the iOS SDK, pp.300-312, 2013.

[8] NSHipster, “Method Swizzling”, http://nshipster.com/method-swizzling/.

[9] Steipete, Aspects, https://github.com/steipete/Aspects.

국방 무인기체계 SW 시험을 위한 DO-178C 적용방안 연구

류인수 ○

㈜모아소프트, 광운대학교 방위사업학과 박사과정

isryu@moasoftware.co.kr

A Study on the Application Method of DO-178C for the Defense UAV System SW Test

Ryu In Soo○

MOASOFT, Department of Defence Acquisition Program Graduate School of Kwangwoon University

요 약

본 연구에서는 무인기체계에 탑재되는 소프트웨어 개발 시 반드시 준수해야 하는 방위사업청 '무기체계 소프트웨어 개발 및 관리 매뉴얼'과 DO-178C의 소프트웨어 시험관련 규정을 살펴보고, 이 두가지 지침과 표준을 충족하는 소프트웨어 개발 단계별 소프트웨어 시험준비 및 시험실시 방안을 제시하였다.

1. 서 론

전 세계 무기체계 중 무인기체계의 대표격인 글로벌 호크는 이라크 전에서 미사일 포대, SAM 발사대, 전차 등 핵심표적 정보를 미군에게 제공함으로써 미군의 승리를 이끈 주요 무기체계로 인정받게 되었다. 이처럼 현대전에서 무인기체계는 전쟁수행의 핵심 무기체계로 발전해 가고 있다. 최근 우리나라도 중고도 무인기체계 개발 등 무인기체계 개발에 박차를 가하고 있다.

무인기체계는 임무 중요(Mission Critical) 체계로 소프트웨어 시험은 '무기체계 소프트웨어 개발관리 매뉴얼'(이하 관리 매뉴얼)과 'DO-178C'을 준수해야 한다. 따라서 이 두가지 지침과 표준을 충족하는 소프트웨어 시험방안이 요구되어 '국방 무인기체계 SW 시험을 위한 DO-178C 적용방안'을 연구하였다.

2. 본 론

무인기체계에 탑재되는 소프트웨어는 '군용항공기 비행안전성 인증에 관한 법률'에 따라 감항인증 기준을 준수해야 하고,[1] 무기체계로 분류되므로 방사청의 '관리 매뉴얼'규정을 만족하여야 한다. 따라서 본 논문에서는 방사청 '관리 매뉴얼'과 DO-178C의 소프트웨어 시험관련 규정을 알아보고 소프트웨어 개발 단계별 소프트웨어 시험준비 및 시험실시 방안을 제시하였다.

2.1. 무인기체계 개발 관련 국내 지침 및 국제 표준

2.1.1. 국방 '무기체계 소프트웨어 개발 및 관리 매뉴얼'의 소프트웨어 시험 관련 사항

방사청의 '관리 매뉴얼'은 방위력개선사업으로 획득되는 소프트웨어의 체계적인 개발 및 관리를 위한 프로세스와 산출물 작성표준 등을 규정한 문서이다. '관리 매뉴얼'에 명시된 소프트웨어 개발 단계별 시험 관련 사항은 다음과 같다.[2]

- 소프트웨어 개발 계획 수립단계에서는 소프트웨어 개발계획서에 신뢰성 시험 및 보안성 시험을 포함한 소프트웨어 시험 방안 시험 환경 구축 방안과 동적 시험 기준 설정 방법을 반영해야 한다.
- 소프트웨어 요구사항 분석단계에서는 소프트웨어 품질요소 기술사항에 신뢰성 확보활동 요구사항을 요구사항명세서에 기술해야 한다.
- 소프트웨어 구조/상세설계단계에서는 소프트웨어통합시험계획서(초안)을 작성해야 한다.
- 소프트웨어 구현단계에서는 방사청 코딩규칙을 준수하고, 신뢰성 시험을 지속적으로 수행하여 결함을 사전에 제거하며, 소프트웨어 신뢰성 시험에 대한 세부계획을 작성해야 한다.
- 소프트웨어 시험평가 단계의 개발시험평가에 적용하는 표 1과 같이 정적/동적시험 기준을 충족하여야 한다.

표 1 정적/동적시험 기준

구 분		평가 기준
정적 시험	코딩 규칙 검증	결함이 검출되지 않아야 한다. 결함 검출 시 소프트웨어 소스코드 수정 후 재시험하여 결함 존재 여부를 확인한다. 단, 거짓경보는 결함에 포함하지 않는다. 소스코드 메트릭은 6종에 대한 기준값을 충족해야 한다.
	취약점 점검	
	소스코드 메트릭	
동적 시험		설정된 시험기준(문장, 분기, MC/DC)에 대한 코드 실행률 100% 달성 여부

- 규격화 단계에서는 시험평가단계에서 결함수정 등으로 소프트웨어가 변경된 경우 소프트웨어 속성정보(파일명, 크기, 라인수, 체크섬, 수정날짜 등), 기술문서 보완사항을 제출해야 한다. 시험평가결과 결함 또는 보완요구사항이 있는 경우 원인을 분석하여 보완하고, 변경된 소프트웨어에 대한 신뢰성시험을 실시한 후 그 결과를 소프트웨어 통합시험결과서에 반영하여야 한다.

2.1.2. DO-178C의 소프트웨어 시험 관련 사항

항공용 소프트웨어와 관련된 개발 표준은 DO-178C, DO-278A가 있다. 항공기에 탑재되는 소프트웨어 개발은 DO-178C를 적용하고, 지상용 항공관제장비에 탑재되는 소프트웨어 개발은 DO-278A를 적용한다.

DO-178C에서 소프트웨어 시험은 소프트웨어가 달성요건(Objectives)을 만족하는 것을 입증하고, 시스템의 안전성 평가 결과에 따라 고장상태로 이어질 수 있는 오류가 제거된 것을 입증하기 위해 사용된다고 정의하고 있다. 소프트웨어 시험 목적을 달성하기 위해 사용되는 소프트웨어 시험 관련 활동(Activities)은 그림 1과 같다.[3]

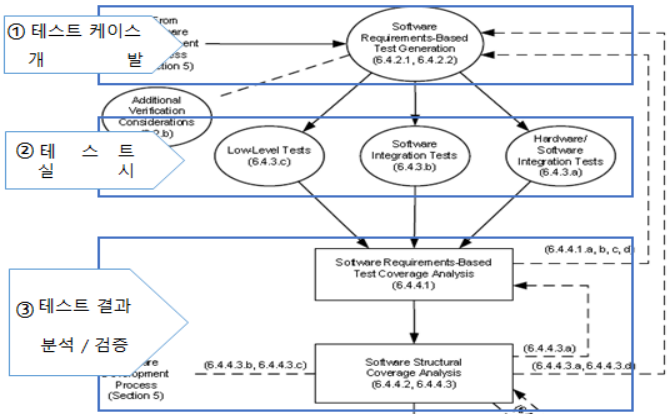


그림 1 소프트웨어 시험 관련 활동(Activities)

DO-178C의 소프트웨어 시험 활동(Activities)은 크게 3가지 단계로 구분하여 설명할 수 있다.

1단계는 테스트 케이스를 개발하는 단계로서 요구사항 명세서와 설계기술서를 기초로 요구사항별 시험방법을 식별하여 테스트 케이스를 작성하는 과정이다.

2단계는 시험을 실시하는 단계로서 1단계에서 작성한 테스트 케이스를 이용하여 하위수준 시험, 소프트웨어 통합시험, 하드웨어/소프트웨어 통합시험을 실시한다.

3단계는 시험결과 분석과 검증하는 단계로서 2단계에서 실시한 요구사항기반의 시험 커버리지를 분석하여 커버리지가 미달하면 1단계로 돌아가 테스트 케이스를 개발하고 추가적인 시험을 실시한다.

소프트웨어 요구사항기반 시험은 하위수준시험(Low-level testing), 소프트웨어 통합시험(SW integration test), 하드웨어/소프트웨어 통합시험(HW/SW integration test)으로 구분하고 있다.

하위수준시험(Low-level testing)은 하위수준 요구사항이 실행되는지 확인하기 위한 시험이며, 소프트웨어 통합 시험은 소프트웨어 요구사항과 구성요소, 소프트웨어 구성요소와 소프트웨어 구조를 확인하기 위한 시험이고, 하드웨어/소프트웨어 통합시험은 대상 컴퓨터 환경에서 소프트웨어가 올바르게 동작하는지 확인하기 위한 시험이다.

소프트웨어 구조기반 커버리지 분석은 요구사항기반의 시험방법에 의해 실행되지 않은 코드구조와 컴포넌트 간의 인터페이스를 분석한다. 요구사항기반 테스트 케이스만으로 코드구조가 완전하게 시험되지 않을 수 있으므로 구조기반 커버리지 분석을 추가적으로 수행하게 된다.

2.2. 무인기체계 소프트웨어 개발 단계별 소프트웨어 시험준비 및 시험실시 방안

무인기체계 소프트웨어는 무기체계 소프트웨어 개발 프로세스, DO-178C 시험활동, 무기체계 소프트웨어 시험 평가항목에 따라 시험하여야 한다. 무인기체계 개발 시 무기체계 소프트웨어 개발 프로세스에 맞게 DO-178C와 무기체계 소프트웨어 시험평가 항목을 적용할 수 있도록 그림 2에 '무기체계 개발 프로세스와 DO-178C 시험 활동 비교표'를 제시하였다.

그림 2의 비교표에 따라 요구사항 분석 단계부터 소프트웨어 통합시험 단계까지 개발 단계별 소프트웨어 시험준비 및 시험실시 방안을 세부적으로 제시하였다.

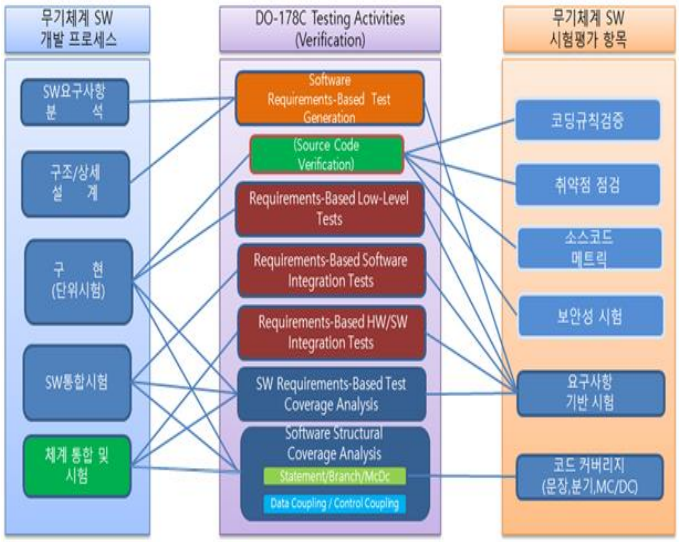


그림 2 무기체계 개발 프로세스와 DO-178C 시험활동 비교표

2.2.1. 소프트웨어 요구사항 분석단계

이 단계에서 방위사업청의 '관리 매뉴얼'은 소프트웨어개발계획서(SDP)에 소스코드 구현 시 준수할 코딩 규칙 및 소스코드 품질 메트릭 기준 값 등 소프트웨어 신뢰성 확보 활동 계획을 포함해야 한다. 소프트웨어요구사항명세서(SRS)의 요구사항별 시험가능성, 시험방법 등을 검토 및 분석해야 하며, 소프트웨어 요구사항 항목 중 '소프트웨어 품질요소'에 소프트웨어 신뢰성 확보활동을 위한 요구사항을 포함해야 한다.[2]

이 단계에서 DO-178C는 요구사항 단계의 산출물 검증(Verification of Outputs of Software Requirements Process)에 대한 달성요건(Objective)은 표 2와 같이 적용한다.[3]

표 2 요구사항 분석 단계의 달성요건

	Objective	Applicability by Software Level			
	Description	A	B	C	D
1	High-level requirements are compatible with target computer.	○	○		
2	High-level requirements are verifiable.	○	○	○	
비 고	○ : The objective should be satisfied. Blank : Satisfaction of objective is at applicant's discretion				

1번 항목은 상위수준 요구사항에 대한 시험환경구축과 관련이 있는 사항으로 운용환경에 맞게 시험환경을 구축해야 한다.

2번 항목은 상위 요구사항에 대한 시험 유형 개발에 관련된 사항으로 상위 요구사항을 시험할 수 있도록 테스트 케이스를 작성해야 한다.

2.2.2. 소프트웨어 구조 및 상세 설계 단계

이 단계에서 방위사업청의 '관리 매뉴얼'은 소프트웨어 구조 및 상세 설계단계에서 소프트웨어 상세설계 문서를 기초로 소프트웨어통합 시험계획서(STP) 초안을 작성한다. 소프트웨어 신뢰성시험 계획을 소프트웨어통합 시험계획서(STP)에 포함하고, 신뢰성시험 및 평가방법 및 절차, 기준을 설정해야 한다.[2]

이 단계에서 DO-178C는 요구사항분석 단계에서 작성한 테스트 케이스에 추가하여 설계기술서에서 도출할 수 있는 테스트 케이스를 작성하고, 요구사항 명세서와 설계기술서의 양방향 추적성을 확인해야 한다.[3]

표 3 소프트웨어 구조 및 상세 설계 단계 달성요건

	Objective	Applicability by Software Level			
	Description	A	B	C	D
1	Low-level requirements comply with high-level requirements.	●	●	○	
2	Low-level requirements are accurate and consistent.	●	●	○	
3	Low-level requirements are compatible with target computer.	○	○		
4	Low-level requirements are verifiable.	○	○		
5	Low-level requirements conform to standards.	○	○	○	
6	Low-level requirements are traceable to high- level requirements.	○	○	○	
비 고	● : The objective should be satisfied with independence ○ : The objective should be satisfied. Blank : Satisfaction of objective is at applicant's discretion				

1번 항목과 3번 항목은 하위 요구사항과 소프트웨어 아키텍처 시험환경 구축과 관련이 있는 사항으로 운용환경에 맞게 시험환경을 구축해야 하며, 이를 위해 DO-178C 6.3.2항목의 활동(Activity)을 적용해야 한다.

2번 항목과 4번 항목은 하위 요구사항과 소프트웨어 아키텍처 테스트 케이스 구축 관련 사항으로 테스트 케이스를 작성해야 하며, 이를 위해 DO-178C 6.3.3항목의 활동(Activity)을 적용해야 한다.

5번 항목은 소프트웨어 아키텍처의 표준준수 여부를 시험하는 항목이다. 이때 DO-178C 6.3.3항목의 활동(Activity)을 적용해야 한다.

6번 항목은 소프트웨어 파티션 상호간 간섭 등으로 인한 결함이 발생하지 않도록 소프트웨어 통합 시험에서 고려해야 하는 요소이다. 이를 위해 DO-178C 6.3.3 항목의 활동(Activity)을 적용해야 한다.

2.2.3. 소프트웨어 구현 단계

이 단계에서 방사청의 '관리 매뉴얼'은 "무기체계 소프트웨어 코딩규칙"을 준수하고, 자체적으로 소프트웨어 단위시험을 실시한 후 단위시험 결과를 바탕으로 소프트웨어통합시험계획서를 최신화하고 소프트웨어 통합시험절차서를 작성한다.

소스코드 검증(Verification)에 필요한 코딩규칙, CWE 취약점 점검, 소스코드 메트릭 및 보안성 시험과 관련된 보안 코딩 규칙 등에 위배되지 않도록 코딩하고, 개발자가 자체적으로 시험 및 분석활동을 실시한다.[2]

이 단계에서 DO-178C는 하위수준 요구사항에 대하여 요구사항기반 시험을 실시하고, 이에 대한 시험 커버리지를 분석한 후 소프트웨어 등급에 따라 구조기반 커버리지 분석(문장, 분기, MC/DC)을 실시한다. 시험 및 분석과정에서 미흡한 사항이 발생하면 충족할 때까지 시험 유형 추가, 소스코드 수정 등의 과정을 반복한다.

코딩단계의 달성요건(Objective)별 활동(Activity)은 소프트웨어 등급분류 결과에 따라 1~9번 항목까지 표 4와 같이 적용한다.[3]

표 4 소프트웨어 구현 단계 달성요건

	Objective	Applicability by Software Level			
	Description	A	B	C	D
1	Source Code complies with low-level requirements.	●	●	○	
2	Source Code complies with software architecture.	●	○	○	
3	Source Code is verifiable.	○	○		
4	Source Code conforms to standards.	○	○	○	
5	Source Code is traceable to low-level requirements.	○	○	○	
6	Source Code is accurate and consistent.	●	○	○	
7	Output of software integration process is complete and correct.	○	○	○	
8	Parameter Data Item File is correct and complete	●	●	○	○
9	Verification of Parameter Data Item File is achieved.	●	●	○	

2.2.4. 소프트웨어 통합시험 단계

이 단계에서 방위사업청의 '관리 매뉴얼'은 '소프트웨어통합 시험계획서'에 따라 소프트웨어 통합시험을 실시한다.

방위사업청에서 요구하는 동적시험은 구조기반 커버리지 분석이며, 문장시험, 분기시험, MC/DC 3가지 방법으로 커버리지 100%를 달성하여야 한다. 개발자의 의도된 코드 또는 방어코드 등의 불가피한 사유로 코드 실행률 100% 미 충족하는 부분은 사유를 제시하여야 한다.[2]

DO-178C에서 소프트웨어 통합시험은 소프트웨어 요구사항 간의 상호관계와 소프트웨어 아키텍처 요구사항의 구현에 관한 것이다. 요구사항기반의 소프트웨어 통합시험은 소프트웨어 구성요소가 서로 정확하게 상호작용하고 소프트웨어의 구조와 요구조건을 충족하는지 검증해야 한다.[3]

표 5 소프트웨어 통합시험 단계 달성 요건

	Objective	Applicability by Software Level			
	Description	A	B	C	D
1	Executable Object Code complies with high-level requirements.	○	○	○	○
2	Executable Object Code is robust with high-level requirements.	○	○	○	○
3	Executable Object Code complies with low-level requirements.	●	●	○	
4	Executable Object Code is robust with low-level requirements.	●	○	○	
5	Executable Object Code is compatible with target computer.	○	○	○	○
비 고	● : The objective should be satisfied with independence ○ : The objective should be satisfied. Blank : Satisfaction of objective is at applicant's discretion				

이 단계에서 방사청의 '관리 매뉴얼'은 통합된 체계가 체계 요구사항에 부합하는지 확인하고, 체계통합 시험결과서를 작성한다. 소프트웨어통합 시험단계에서 소프트웨어 동적시험을 미실시한 경우에는 소프트웨어를 하드웨어 장비에 탑재하여 동적 시험을 수행한다.[2]

DO-178C에서 하드웨어와 소프트웨어 통합시험(HW/SW integration test)은 실제 하드웨어(Target)환경에서 소프트웨어 동작과 관련된 결함원인을 찾아내고 상위의 기능이 적절히 구현되었는지 검증해야 한다.[3]

또한 소프트웨어 요구사항기반 시험 실행률 분석(software requirements-based test coverage analysis)을 요구한다. 이 분석은 앞 단계에서 실시한 요구사항기반 시험에서 소프트웨어 요구사항이 잘 구현되었는지, 추가적인 시험 유형 필요성을 분석하기 위한 것이다.

소프트웨어 요구사항기반 시험 실행률 분석(software requirements-based test coverage analysis)에 이어 소프트웨어 구조기반 시험 실행률 분석(software structural coverage analysis)을 실시한다. 이 분석은 컴

포넌트간의 인터페이스를 포함하여 요구사항기반 시험에서 어떤 코드구조(code structure)가 시험되지 않았는지 확인한다.

3. 결 론

무인기체계는 군용항공기 비행안전성 인증에 관한 법률에 따른 감항인증 기준과 방사청의 '관리 매뉴얼'에서 규정하는 소프트웨어 시험 기준을 모두 충족하여야 한다. 본 논문에서는 DO-178C와 방사청 '관리 매뉴얼'을 모두 충족하도록 소프트웨어 시험준비 및 시험실시 방안에 대한 절차 정도의 기준을 제시하였다. 향후 각 단계별 세부 시험활동에 대한 추가적인 연구가 요구된다.

[참고문헌]

[1] 군용항공기 비행안전성 인증에 관한 법률[법률 제11559호, 2012.12.18)

[2] 무기체계 소프트웨어 개발 및 관리 매뉴얼(2016), 방위사업청

[3] RTCA Inc, (2011), "Software Considerations Airborne Systems and Equipment Certification", Document RTCA/DO-178C

TPC-DI 기반의 ETL 솔루션 성능 테스트 사례

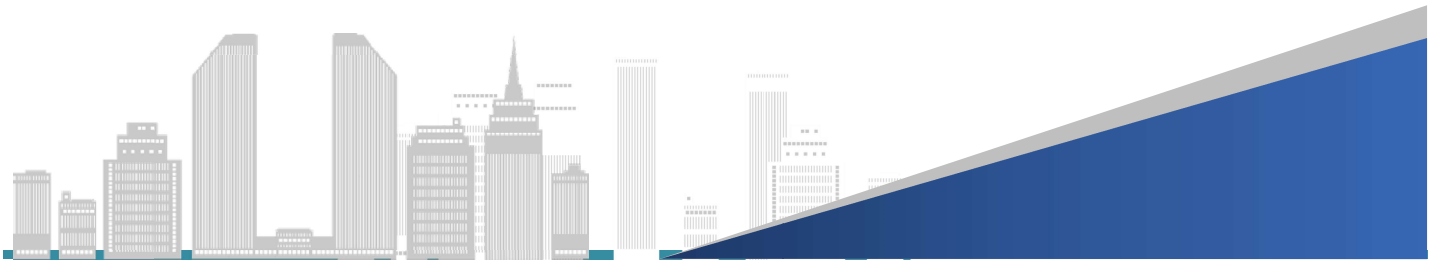
한국정보통신기술협회 김상기, 강건희

2017. 2.

목차

Ⅰ 개요

- ETL 정의 및 목적
- ETL SW 분류
- ETL 솔루션 평가의 현상 및 문제점



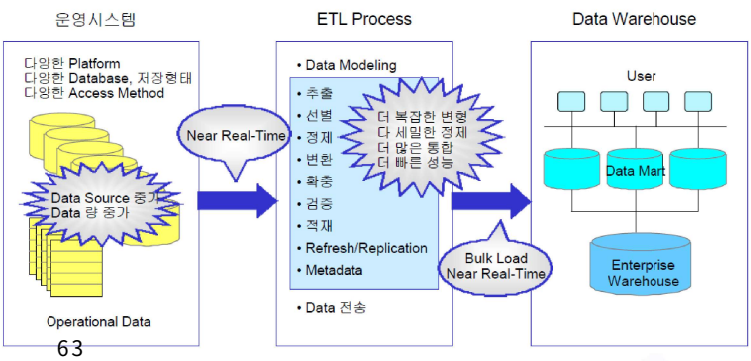
ETL 정의 및 목적

ETL 정의

데이터의 추출(Extraction), 가공(Transformation), 로딩(Loading)의 약자로 데이터를 소스 시스템에서 추출하여 목표 시스템에 로드시켜 정제작업까지 이르는 전 과정을 의미함.

ETL 목적

원시 데이터를 획득하고 데이터 변환, 정제하여 목표 시스템으로의 전송, 관리 및 스케줄링 하는 것.
데이터의 추출 주기 및 양, 로드 속도, 데이터의 질, 과거 데이터의 형식, 사용자의 요구 조건 등에 따라 달라짐.



ETL SW 분류

ETL SW 분류

구분	특징
프로그래밍(in-house coding)	▪ 숙련된 프로그래머가 필요하다. ▪ Offline 방식이나 online 방식 가능 ▪ 장점은 각각 사용자시스템에 맞는 ETL을 할 수 있다. ▪ Aggregation 까지 할 수 있다 ▪ 개발하는 시간이 오래 걸린다 ▪ 빠르게 디자인 할 수 있다 ▪ 개발 know-how가 필요하다
대규모 Loading/Unloading	▪ DBMS 업체가 제공하는 기능 ▪ 단순 추출, 데이터 등록 ▪ 타 도구와 연계
게이트웨이 툴	▪ 일반적인 copy 기능과 빠른 속도로 전송 ▪ 제한적인 정제 및 통합
데이터 정제/통합 툴	▪ 데이터 정제 기능 ▪ 타 도구와 연계

ETL 솔루션 평가의 현상 및 문제점

다양한 ETL 솔루션의 등장

데이터 통합 및 이동의 중요성에 따라 고객 요구에 맞는 다양한 기능을 개발한 ETL 솔루션이 시장에 출시됨.

표준화된 ETL 솔루션 평가 방법의 필요성

데이터 획득(통합, 이동 등) 시 다양한 시나리오에 대한 표준화된 평가 방법의 부재

복잡하고 다양한 경우의 수를 대비한 테스트 데이터 준비의 한계

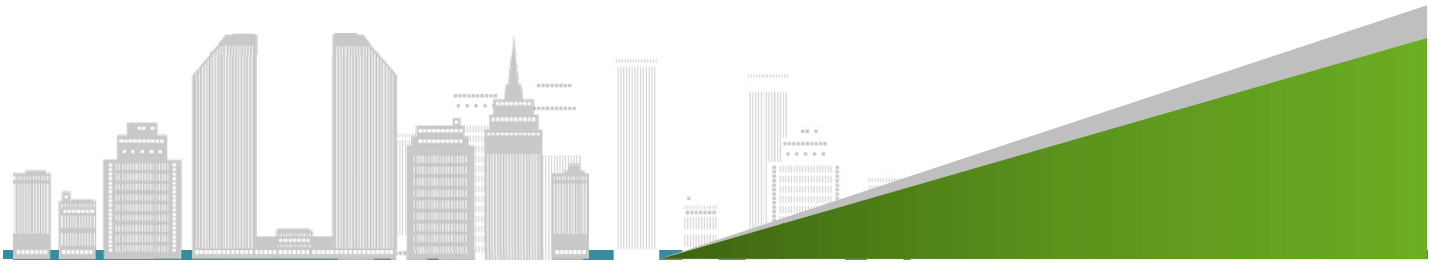
테스트 환경 제약 조건(서버, OS 등)에 따른 환경 구축의 어려움

방안 연구

기업의 ETL 솔루션 선택
- 의사 결정 시 활용
- 기능/성능 비교

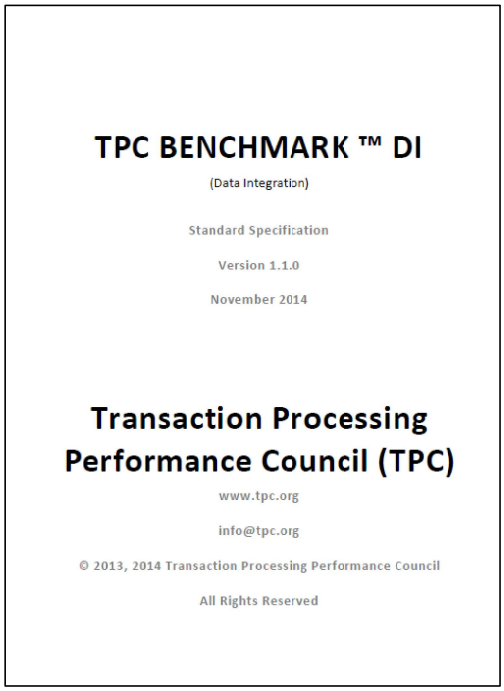
Ⅱ 관련 연구

- TPC-DI
- 해외 연구 논문



TPC-DI

TPC-DI 개요


TPC BENCHMARK ™ DI

(Data Integration)

Standard Specification

Version 1.1.0

November 2014

Transaction Processing Performance Council (TPC)

www.tpc.org

info@tpc.org

© 2013, 2014 Transaction Processing Performance Council

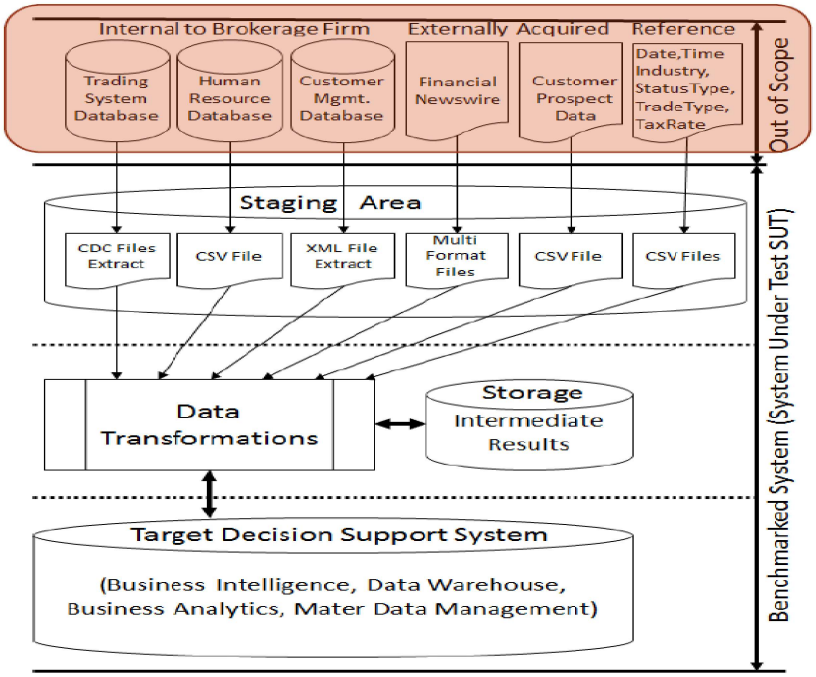
All Rights Reserved

- 데이터 이동 및 통합하는 도구의 성능 테스트 활용
- 다양한 데이터 통합(DI) 도구의 비교를 위한 표준 방법 제공
- TPC-DI-Tool을 통하여 필요한 크기만큼의 데이터웨어하우스(DW) 데이터를 준비할 수 있음
- 다음과 같은 특징이 있음
  - 대량 데이터의 조작 및 로딩
  - Error 오류 검사
  - 데이터 유형 변환
  - 집계 연산
  - 데이터 업데이트
- http://www.tpc.org/tpcdi/default.asp

* TTA 표준용어 기준 번역

TPC-DI

TPC-DI 범위



➔ TPC-DI-Tool을 통해 생성
> java –jar DIGen.jar

➔ 생성된 파일을 타겟시스템에 적재하기 위한 명세를 제공함

해외 연구 논문

TPC-DI: The First Industry Benchmark for Data Integration

Meikel Poess
Server Technologies
Oracle Corporation
Redwood Shores, California
USA
mpoess@oracle.com

Tilmann Rabl,
Hans-Arno Jacobsen
Middleware Systems
Research Group
University of Toronto
Canada
tilmann.rabl@utoronto.ca
jacobsen@eecg.toronto.edu

Brian Caufield
InfoSphere DataStage
IBM
San Jose, California
USA
bcaufiel@us.ibm.com

ABSTRACT

Historically, the process of synchronizing a decision support system with data from operational systems has been referred to as Extract, Transform, Load (ETL) and the tools supporting such process have been referred to as ETL tools. Recently, ETL was replaced by the more comprehensive acronym, data integration (DI). DI describes the process of extracting and combining data from a variety of data source formats, transforming that data into a unified data model representation and loading it into a data store. This is done in the context of a variety of scenarios, such as data acquisition for business intelligence, analytics and data warehousing, but also synchronization of data between operational applications, data migrations and conversions, master data management, enterprise data sharing and delivery of data services in a service-oriented architecture context, amongst others. With these scenarios relying on up-to-date information it is critical to implement a highly performing, scalable and easy to maintain data integration system. This is especially important as the complexity, variety and volume of data is constantly increasing and performance of data integration systems is becoming very critical. Despite the significance of having a highly performing DI system, there has been no industry standard for measuring and comparing their performance. The TPC, acknowledging this void, has released TPC-DI, an innovative benchmark for data integration. This paper motivates the reasons behindits development, describes its main characteristics including workload, run rules, metric, and explains key decisions.

1. INTRODUCTION

The term data integration (DI) covers a variety of scenarios, predominantly data acquisition for business intelligence, analytics and data warehousing, but also synchronization of data between operational applications, data migrations and conversions, master data management, enterprise data sharing and delivery of data services in a service-oriented architecture context, amongst others. Each of these scenarios requires the extraction of data from one or multiple source systems and data transformation and writing the data to one or more target systems.

While small DI deployments tend to be implemented using collections of customized programs or database procedures, medium to large sized DI deployments are usually implemented using general purpose DI tools. These deployments often must integrate data from many disparate data sources with various formats requiring complex formatting and data transformations prior to loading into one or more target systems. General purpose DI tools can significantly increase developer productivity by providing commonly used functionality for system connectivity and for standard data transformations. They further improve the availability and maintenance of the DI processes by visualizing connections, transformations and progress of running tasks. A non-exhaustive list of commercially available tools includes, for example, Ab Initio[1], IBM InfoSphere Information Server for Data Integration[2] Microsoft SSIS[3], and Oracle Warehouse Builder[4].

Ever since vendors started implementing and marketing general purpose DI tools, they started making competitive and performance claims. With no standard DI benchmark available, today's situation is similar to that of 1980s, when many system vendors due to the the lack of standard database benchmarks practiced what is now referred to as *benchmarketing*, a practice in which organizations make performance claims based on self-designed, highly biased benchmarks. Today, a large number of *world record* claims have been made for DI systems (e.g., [8, 10, 12]). These are of no value to customers who would like to evaluate DI performance across vendors. Having realized this void the Transaction Processing Performance Council (TPC) released the first version of its data integration benchmark, TPC-DI, in January 2014[5]. TPC-DI is modeled using the data integration processes of a retail brokerage firm, focusing on populating a decision support system with transformed data from a variety of desparate systems, including a trading system, internal Human Resource (HR) and Customer Relationship Management (CRM) systems. The mixture and variety of operations being measured by TPC-DI are not designed to exercise all possible operations used in DI systems. And they are certainly not limited to those of a brokerage firm. They rather capture the variety and complexity of typical tasks executed in a realistic data integration application that are characterized by:

Proceedings of the VLDB Endowment, Vol. 7, No. 13
Copyright 2014 VLDB Endowment 2150-8097/14/08.

[1] http://www.abinitio.com/
[2] http://www-03.ibm.com/software/products/en/infosfoserverfordatainte
[3] http://technet.microsoft.com/en-us/library/ms141826.aspx
[4] http://www.oracle.com/technetwork/developer-tools/warehouse/overview/introduction/index.html
[5] http://www.tpc.org/tpcdi/default.asp

TPCDI: The First Industry Benchmark for Data Integration,
Meikel Poess Server Technologies Oracle Corporation
Redwood Shores, California USA

Benchmarking ETL Workflows

Alkis Simitsis[1], Panos Vassiliadis[2], Umeshwar Dayal[1],
Anastasios Karagiannis[2], Vasiliki Tziovara[2]

[1] HP Labs, Palo Alto, CA, USA,
{alkis, Umeshwar.Dayal}@hp.com
[2] University of Ioannina, Dept. of Computer Science, Ioannina, Hellas
{pvassil, ktasos, vickit}@cs.uoi.gr

Abstract. Extraction–Transform–Load (ETL) processes comprise complex data workflows, which are responsible for the maintenance of a Data Warehouse. A plethora of ETL tools is currently available constituting a multi-million dollar market. Each ETL tool uses its own technique for the design and implementation of an ETL workflow, making the task of assessing ETL tools extremely difficult. In this paper, we identify common characteristics of ETL workflows in an effort of proposing a unified evaluation method for ETL. We also identify the main points of interest in designing, implementing, and maintaining ETL workflows. Finally, we propose a principled organization of test suites based on the TPC-H schema for the problem of experimenting with ETL workflows.

Keywords: Data Warehouses, ETL, benchmark.

1 Introduction

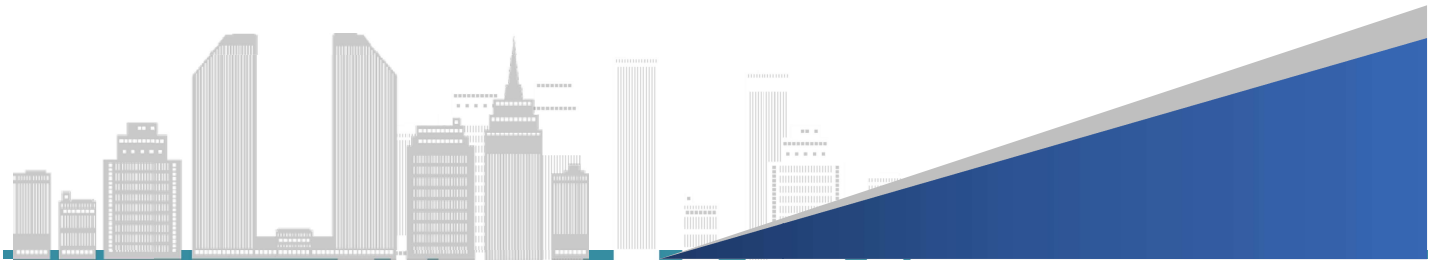
Data warehousing is a technology that enables decision-making and data analysis in large organizations. Several products are available in the market and for their evaluation, the TPC-H benchmark has been proposed as a decision support benchmark [16]. TPC-H focuses on OLAP (On-Line Analytical Processing) queries and it mainly deals with the data warehouse site. Another version termed TPC-DS has been around for the last few years, but this version is still in a draft form [11, 15]. TPC-DS considers a broader picture than TPC-H including the whole flow from the sources to the target data warehouse. However, it partially covers the data warehouse maintenance part, considering only simple mechanisms for inserting and deleting tuples.

To populate a data warehouse with up-to-date records extracted from operational sources, special tools are employed, called *Extraction – Transform – Load* (ETL) tools, which organize the steps of the whole process as a workflow. To give a general idea of the functionality of these workflows we mention their most prominent tasks, which include: (a) the *identification* of relevant information at the source side; (b) the *extraction* of this information; (c) the *transportation* of this information to the Data Staging Area (DSA), where most of the transformation usually take place; (d) the *transformation*, (i.e., customization and integration) of the information coming from multiple sources into a common format; (e) the *cleansing* of the resulting data set, on the basis of database and business rules; and (f) the *propagation* and loading of the data to the data warehouse and the refreshment of data marts.

Benchmarking ETL Workflows
Alkis Simitsis, Panos Vassiliadis, Umeshwar Dayal

III 본 론

- TPC-DI 데이터 모델 선정
- 테스트 시나리오 및 데이터 설계
- 테스트 환경 구축
- 성능 테스트 수행



TPC-DI 데이터 모델 선정

TPC-DI 소스 스키마

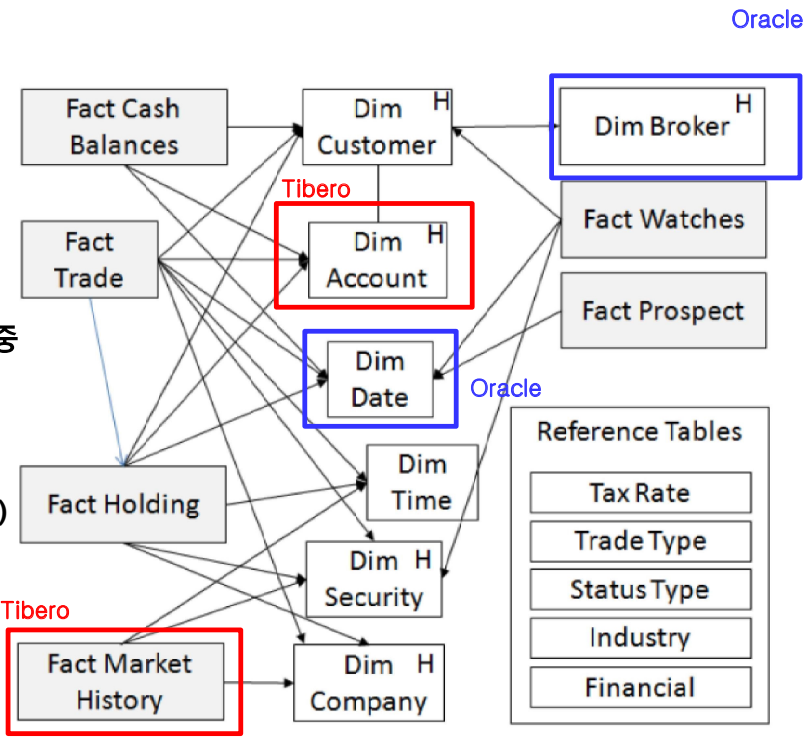
- 18개 다른 테이블
- 다양한 종류의 포맷
  - CSV (Comma Separated)
  - CDC (Change Data Capture)
  - Multi Record
  - DEL (Pipe Delimited)
  - XML
- Historical Load와 Incremental Load 구분
- BMT 시 전체를 사용하지는 않음

Source Table	Format	H	I
Account.txt	CDC		✓
CashTransaction.txt	DEL/CDC	✓	✓
Customer.txt	CDC		✓
CustomerMgmt.xml	XML	✓	
DailyMarket.txt	DEL	✓	✓
Date.txt	DEL	✓	
Time.txt	DEL	✓	
FINWIRE	Multi-record	✓	
HoldingHistory.txt	DEL	✓	✓
HR.csv	CSV	✓	
Industry.txt	DEL	✓	
Prospect.csv	CSV	✓	✓
StatusType.txt	DEL	✓	
TaxRate.xt	DEL	✓	
TradeHistory.txt	DEL	✓	
Trade.txt	DEL/CDC	✓	✓
TradeType.txt	DEL	✓	
WatchItem.txt	DEL/CDC	✓	✓

TPC-DI 데이터 모델 선정

TPC-DI 타겟 스키마

- 6개 Fact 테이블
- 7개의 Dimension 테이블
- 5개의 Reference 테이블
- BMT 수행 시에는 소스 스키마 중 선택된 일부 파일로 생성 가능한 일부 테이블만을 대상으로 함. (2개의 다른 종류의 DBMS 사용)



테스트 시나리오 및 데이터 설계

시나리오 설계

1) ETL 솔루션의 주요 기능 평가 항목
   - 데이터 조작 및 로딩
   - 데이터 오류 검사
   - 데이터 유형 변환, 집계
   - 다양한 형식의 데이터 소스 지원 여부 (TXT, CSV, XML, DB 등)

2) ETL 솔루션의 주요 성능 평가 항목
   - 데이터 변환 처리 속도 (예, XML 문서의 파싱 등)
   - 대용량 데이터의 적재 속도

➔ 위 평가 항목이 포함될 수 있도록 TPC-DI의 데이터 모델 선정하고, 기능 평가항목이 포함된 성능 시나리오를 개발함.

테스트 시나리오 및 데이터 설계

데이터 설계

1) 데이터 파일 결정

- 입력 데이터 타입을 CSV, TXT, XML 파일로 한정

2) 데이터 사이즈 결정

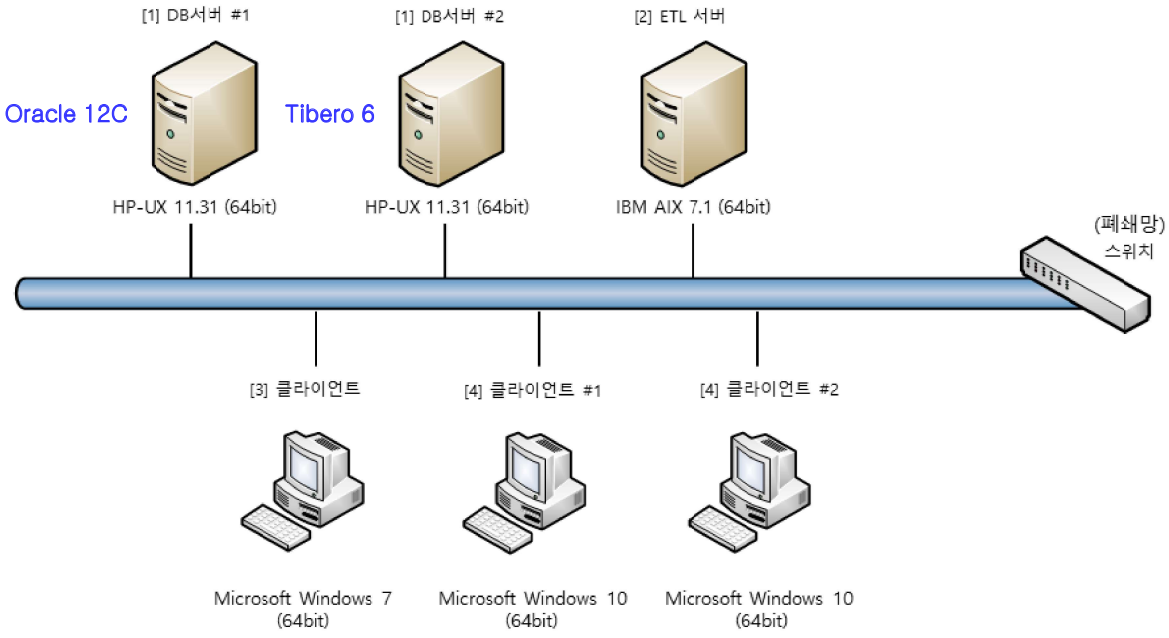
파일 명	Size in Bytes	Number of rows
Date.txt	3,420 KB	25,933 건
HR.csv	415,460 KB	1,425,155 건
CustomerMgmt.xml	3,065,435 KB	3,047,000 건
DailyMarket.txt	31,059,354 KB	541,550,324 건

➔ **1시간 이내로 측정하기 위한 테스트 데이터 사이즈 선정**

: TPC-DI Scale Factor를 1000으로 세팅

테스트 환경 구축

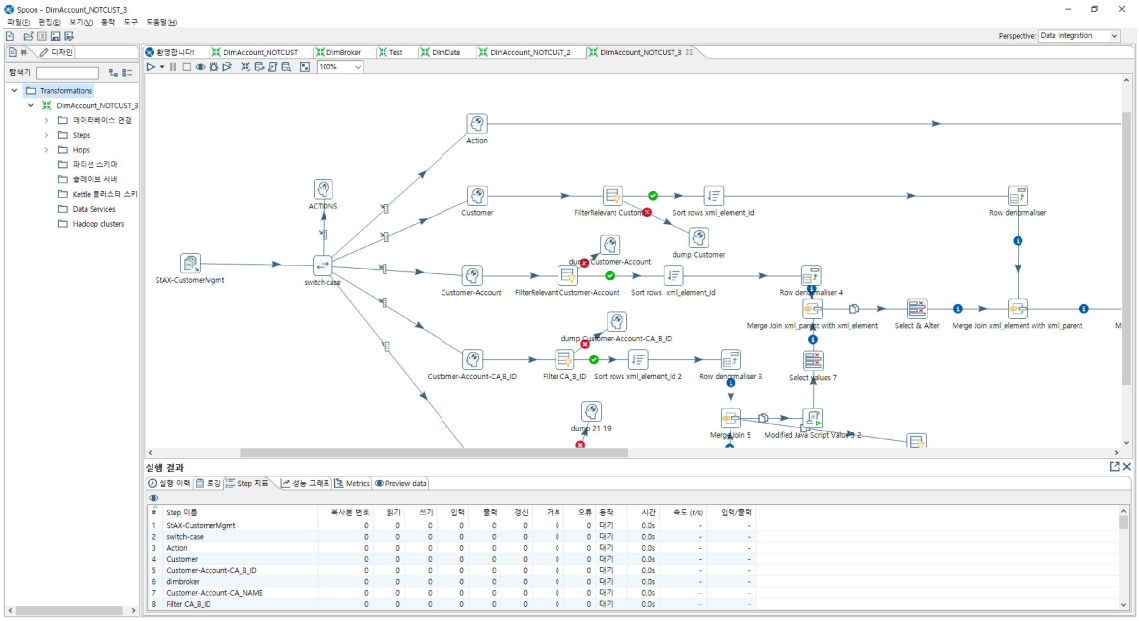
테스트 환경



테스트 환경 구축

사전 검증

오픈 소스 ETL 툴을 사용하여 테스트 환경에 대한 검증 및 성능 테스트를 위한 소요 시간을 예측함.



성능 테스트 수행

시나리오

1. DimDate 테이블을 생성 후 Date.txt 파일의 모든 데이터를 적재한다.

2. DimBroker 테이블을 생성 후 HR.csv 파일의 데이터 중 다음 규칙에 따라 데이터를 선별하고, 선택된 데이터만 적재한다.
   - EmployeeJobCode 가 314가 아닌 Record만 선별
   - 데이터 중 오류값인 '-' 이 포함된 Record는 제외

3. DimAccount 테이블을 생성 후 CustomerMgmt.xml 파일을 다음 규칙에 따라 파싱하여 데이터를 선별하고, 선택된 데이터만 적재한다.
   - ./@ActionType 이 'NEW' or 'ADDACCT' 인 Record만 선별

4. FactoMarketHistory 테이블을 생성 후 DailyMarket.txt 파일을 DBMS의 BulkLoad 기능을 활용하여 모든 데이터를 적재한다.

성능 테스트 수행

테스트 결과

평가 항목		A 업체	B 업체	C 업체
기능	데이터 조작 - 1:1 매핑 - 1:N 매핑 - M:N 매핑	P	P	P
	데이터 오류 처리 기능	P	P	P
	데이터 유형 변환 및 집계	P*	P	P
	리포트 기능	P	P	P
	모니터링 기능	P	P	P*
성능	데이터 변환 처리 시간	X분 XX초	X분 XX초	X분 XX초
	대용량 데이터 적재 시간	XX분 XX초	XX분 XX초	XX분 XX초

P : 합격 (모든 기능이 동작), P* : 부분 합격 (일부 기능 동작이 미흡)

*** 본 테스트 결과는 실제 BMT 결과 중 발생된 수행 결과를 일부 분석하여 논문의 활용이 필요한 부분에 대하여만 발췌하여 정리한 결과로써 실제 BMT 결과가 아님. 또한 위 평가 항목도 일부이거나 상위 레벨로 추상화하여 정리한 결과로 실제 평가 항목은 이보다 상세함.**

Ⅳ 결론

- 요 약
- 향후 계획

요 약

ETL은 데이터의 추출(Extraction), 가공(Transformation), 로딩(Loading)의 약자로 데이터를 소스 시스템에서 추출하여 목표 시스템에 로드시켜 정제작업까지 이르는 전 과정을 의미함.

본 논문에서는 다양한 ETL 솔루션 등장에 따라 해당 솔루션의 품질 및 성능을 비교/평가하기 위한 표준화된 기준이 필요하여, 이에 대한 연구를 통해 TPC-DI에서 정의된 데이터 모델 중 일부를 선택하고, 3개의 제품에 적용하여 비교/평가를 한 사례를 제시하였음.

* 본 연구는 ETL 솔루션에 대한 테스트 수행 시 자체 연구하였던 내용을 바탕으로 작성하였으며, 한국정보통신기술협회(TTA) 기업의 의견이 아닌 개인의 의견으로 기술한 내용임.

향후 계획

2016년 가트너 조사 Report

Figure 1. Magic Quadrant for Data Integration Tools

Source: Gartner (August 2016)

➔ 가상화, 클라우드 서비스 데이터에 대한 통합 지원으로 솔루션이 버전 업되고 있음.
(디지털 / IoT 데이터 통합, iPaaS, 셀프 서비스 데이터 준비, 빅 데이터, 데이터 거버넌스 및 데이터 보안 등)

2016. Magic Quadrant for Data Integration Tools

https://www.gartner.com/doc/reprints?id=1-3CUJXZO&ct=160727&st=sb

➔ 개선된 ETL 솔루션의 기능에 따른 평가 기준 업데이트가 지속적으로 필요함.

TTA 한국정보통신기술협회
Telecommunications Technology Association

벤처/중소기업의 테스트 조직을 위한 한국형 테스트 성숙도 모델 시범 적용 사례

김기두[O], 김영철[OO]

[O]한국정보통신기술협회, [OO]홍익대학교 컴퓨터정보통신공학과 소프트웨어공학연구실

[O]kdkim@tta.or.kr, [OO]bob@selab.hongik.ac.kr

Real Trial Practice of a Simplified Test Maturity Model for Test organization of Small & Medium sized Companies

Kidu Kim[O], R. Young Chul Kim[OO]

[O]Telecommunications Technology Association, [OO]SE Lab., Hongik University

요 약

본 연구에서는 국내 중소기업에 소프트웨어 고품질을 위해, 개발한 한국형 테스트 성숙도 모델을 기업에 적용가능 여부 검증을 하고자 한다. 즉, 기존에 개발된 소프트웨어 성숙도 모델들이 국내 중소 소프트웨어 개발 조직에 비용과 기간 부분에서 적합이 어려워, 기존 TMM의 모델 경량화 작업을 통해 한국형 테스트 성숙도 모델을 개발하였다. 이를 실제 A, B 회사의 개발 조직에 적용하여 경량화된 모델의 적용 가능여부를 확인하고, 적용 결과 기반으로 한국형 테스트 성숙도 모델 개선 사항을 도출하였다.

1. 서 론

예전 하드웨어만, 또는 소프트웨어만 동작하는 시스템에서 최근 소프트웨어와 하드웨어가 함께 동작하는 융/복합 시스템들이 개발되고 있다. 그로 인해, 전체 시스템에서 소프트웨어가 제품에서 차지하는 비중이 높아짐으로써, 제품 품질에서 소프트웨어가 차지하는 비중 또한, 높아졌다. 산업분야별로도 사용자의 생명과 직결되는 의료, 자동차, 국방/항공 등의 분야에서 소프트웨어가 차지하는 비중은 45% 이상을 차지하고 있다[1].

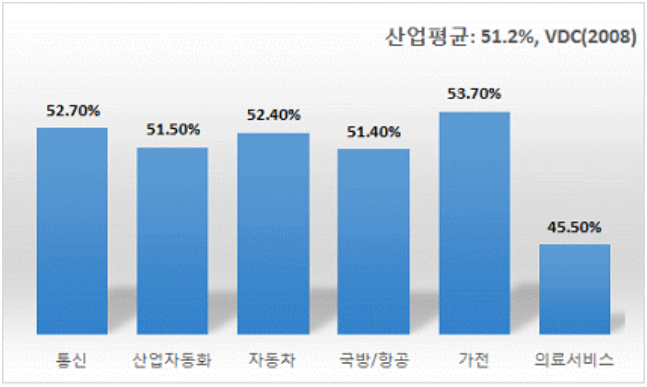


그림 1. 산업별 소프트웨어 비중[1]

그림 1은 산업별 소프트웨어 비중을 보인다. 소프트웨어가 차지하는 비중이 높아지는 만큼 소프트웨어 품질의 중요도에 대한 인식 또한 많이 높아졌다. 소프트웨어 품질을 향상하기 위해 소프트웨어 개발 관련 중소 기업별로 다양한 방법을 고민하였다. 대표적인 방법으로 개발자에 의지한 품질 향상으로, 개발자가 코드 개발 단계에서 품질을 향상하는 방법이다. 하지만, 이 방법은 개발 일정과 많은 업무에 쫓기는 국내 개발현실에서는 어렵다. 오히려, 조직 차원에서 품질 향상을 위한 테스트 절차, 방법, 도구 등을 갖춰 개발하는 경우 일정한 수준의 품질을 확보할 수 있다. 이를 위해, 한국정보통신기술협회(TTA)에서는 국내 중소 기업의 품질 수준 진단 및 개선을 위해 국내 중소 기업에서 활용 가능한 한국형 테스트 성숙도 모델을 개발하였다[2, 3]. 본 연구는 2015/2016년 홍익대학교 소프트웨어공학 연구실과 협력으로 개발된 한국형 테스트 성숙도 모델을 국내 중소기업에 시범 적용하여 개발된 모델의 문제점 파악 및 개선하는데 있다.

2. 관련 연구

2.1 한국형 테스트 성숙도 모델

한국형 테스트 성숙도 모델은 기존의 IIT(일리노이 공대)의 TMM(Test Maturity Model)[4]을 기반으로 국내 중소기업의 개발 현실을 반영하여 개발된 모델이다. 경량화 된 테스트 성숙도 모델은 국내 중소기업 현황을 고려하여 기존의 TMM에서 필수 항목 도출, 적합성 판단, 모델 보완의 과정을 통해 개발되었다. 그림 2와 같이 한국형 테스트 성숙도 모델의 구조는 한국형

테스트 성숙도 모델의 근간이 되는 TMM과 동일하다. 다만, 테스트 프로세스 진단 및 개선을 위해 TPI next[5] 모델이 함께 적용되었다. 시범 적용 당시 레벨 4~5가 개발 중으로 시범 적용에는 2015년에 개발된 한국형 테스트 성숙도 모델 레벨 1~3이 활용되었다.

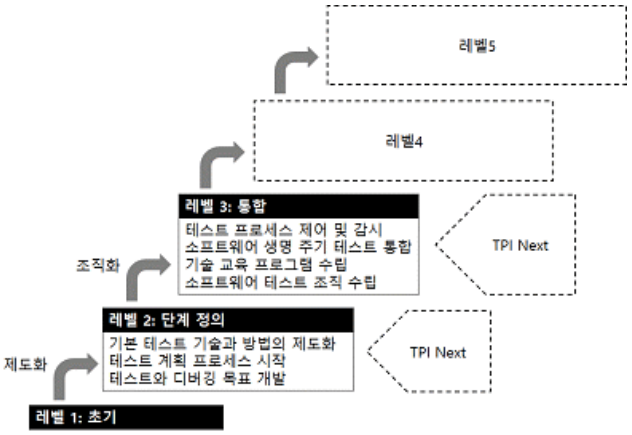


그림 2. 한국형 테스트 성숙도 모델 구조[2]

레벨 1(초기)은 소프트웨어 품질과 관련된 활동 대부분이 수행되지 않는 단계로, 개발자가 테스트의 역할과 병행하며, 버그 해결을 위한 최소한의 활동만 수행한다. 레벨 2(단계 정의)는 테스트를 위한 별도의 조직이 존재하지 않아 소규모로 진행이 되며, 모든 테스트 활동이 소스코드 기반으로 수행된다. 레벨 3(통합)은 테스트 활동이 가능한 조직이 별도로 구성이 되며, 개발 생명주기와 함께 테스트 활동이 함께 이루어지는 단계이다. 그림 2는 한국형 테스트 성숙도 모델 구조도 입니다.

TMM과 한국형 테스트 성숙도 모델은 성숙도 목표, 부목표, 세부 평가 항목으로 구분되는데, 경량화 작업을 통해 세부평가항목이 TMM에 비해 줄어들었다. 세부 평가 항목만 줄어듦으로써, 레벨 별로 달성해야 하는 목표의 기준을 변경하지 않고, 심사를 요청하는 개발 조직의 산출물 및 심사 조직의 활동들을 축소 시킬 수 있었다. 표 1은 한국형 테스트 성숙도 모델 평가 항목 관련 수이다

구분	TMM			한국형 테스트 성숙도 모델		
	레벨2	레벨3	총합	레벨2	레벨3	총합
성숙도 목표	3	4	7	3	4	7
성숙도 부목표	13	12	25	14	12	25
세부평가항목	91	118	209	61	79	140

표 1. 한국형 테스트 성숙도 모델의 평가 항목 수[3]

2.2 한국형 테스트 성숙도 모델 심사 모델

경량화 된 테스트 성숙도 모델을 소프트웨어를 개발하는 중소기업을 대상으로 심사 시 아래 그림 3의 한국형 테스트 성숙도 모델 평가 절차에 따라 진행하도록 되어 있다.

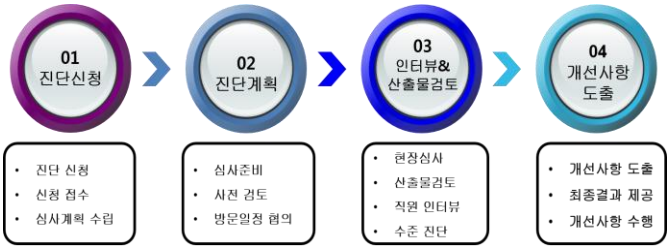


그림 3. 한국형 테스트 성숙도 모델 평가 절차

그림 3은 한국형 테스트 성숙도 모델 평가 절차입니다. 평가 절차는 진단신청, 진단계획, 인터뷰 및 산출물 검토, 개선사항 도출의 4단계로 구성된다. 평가 기업에서 한국형 테스트 성숙도 모델 평가를 신청 시, 진단 기관은 신청을 접수 후 심사계획을 수립한다. 진단 계획은 진단 기관이 평가 기업의 테스트 성숙도 진단을 준비하며, 방문일정을 협의한다. 평가 대상 기업은 지단 관련 자료를 진단 기관에게 제출해야 한다. 평가 기업의 자료들을 검토하기 위해서 진단 기관은 심사팀을 구성하며, 테스트 성숙도 모델 평가 전에 사전 검토를 진행한다. 인터뷰 및 산출물 검토 단계에서는 심사원들의 현장 심사를 통해 평가 기업의 테스트 성숙도를 진단한다. 기업에서 제출한 자료와 부족한 자료들을 분석하고, 평가 기업의 직원들과 인터뷰를 통해 테스트 성숙도 수준을 진단한다. 평가 후, 심사원들은 개선사항을 도출하여 평가 결과를 진단 대상 기업에 제출하여 심사 절차가 종료된다.

3. 한국형 테스트 성숙도 모델 시범 적용

3.1 시범 적용 절차

한국형 테스트 성숙도 모델 적용을 위해 우선적으로 시범 적용이 가능한 대상을 선정하였다. 적용 대상 선정 기준은 아래와 같다.

1) 중소기업
2) 테스트 조직 존재/미존재 여부
3) 대상 기업의 지원 가능 여부
4) 심사 후 개선 의지

우선 대기업이 아닌 중소기업을 대상으로 개발된 모델이기 때문에, 국내 중소기업으로 선정하였다. 그리고, 테스트 조직이 존재하는 경우 또는 존재하지

않는 경우에도 적용가능 여부 확인을 위해 테스트 조직의 존재 여부를 고려하였다. 대상 기업의 지원 가능 여부는 시범 적용 시 가장 중요한 항목으로 대상 기업에서 산출물 및 인터뷰 등의 지원이 미흡할 경우 심사 결과가 무의미 해질 수 있기 때문이다. 마지막, 심사 후 개선 의지는 심사 결과에 따라 조직이 갖고 있는 품질 관련 취약점을 인지하여 개선할 의지를 보유할 경우에만 심사에 대한 효과가 나타나지 때문이다. 위 고려사항을 반영하여 지역에 소재하는 중소기업 2곳(A, B)을 선정하여 시범적용을 진행하였다. A사의 경우 조직의 규모(30명 이하)는 적고 테스트 조직이 존재하지 않지만, 안정적인 사업 실적을 기반으로 기업 전반적인 개선을 위해 연구하는 조직이었다. B사는 A사에 비해 규모가 큰 기업(80명 이하)이며 솔루션 사업 및 다양한 SI사업을 수행하는 기업이었다. 소규모의 테스트 조직이 존재하며, 여러 SI 사업 경험으로 인해 조직의 품질 개선에 대한 의지가 높은 조직이었다. 시범 적용은 대상 기업(A, B)을 방문하여 진행하였으며, 한국형 테스트 성숙도 모델 심사 절차를 기반으로 아래 그림 4와 같이 진행되었다. 진행 기간은 20WD 전후로 진행되었으며, 20WD는 결과 발표까지 소요된 기간이다.

2016.XX.XX

진행 절차	내 용
산출물 수집	• 착수회의 • 프로젝트 산출물(XXX, XXXX, XXXXX 등) 수집 • 조직 관련 자료(조직도, 현장심사 자료) 수집 • 산출물 분석 영역 및 역할 분담
산출물 분석	• 산출물 및 조직 관련 자료 분석 • 테스트 활동 • 질의 항목 작성
인터뷰 진행	• 내용 : 레벨 별 세부평가 시 도출된 질의 항목에 대한 인터뷰 진행 • 일시 : 2016년 XX월 X일, XX시간 • 참석자 :
수준진단	• 산출물 분석 결과 및 인터뷰 내용을 기반으로 테스트 성숙도 수준 진단(레벨)
개선작업	• 분야별(테스트 조직, 테스트 프로세스, 테스트 기법, 테스트 환경, 테스트 관리 및 모니터링) 취약점 도출 • 취약점 개선 방안 도출
결과발표	• 결과 내용 발표 • 일시 : 2016년 XX월 XX일

2016.XX.XX

그림 4. 시범 적용 절차

그림 4는 시범 적용 절차를 보인다. 심사 기간 동안 통해 약 3,000건 이상의 산출물을 수집 및 분석하였다. 또한, 프로젝트 담당자, 관리자, 테스트 담당자 등의 인터뷰를 수행하였으며, 인터뷰 결과 또한, 수준 진단 결과에 반영되었다. 최종적으로 한국형 테스트 성숙도 모델 수준 진단 체크리스트를 통해 평가된 레벨을 기업에게 전달함으로써 종료되었다.

진단결과는 아래 [표 2]의 영역별로 정의하였다. 진단 영역은 기존의 한국형 테스트 성숙도 모델의 경우 기업에게 전달되는 항목과 평가 항목이 많아 평가 항목을 세부 항목들을 테스트 관련 영역별로 구분하여 정리한 내용이다. 실제로 인터뷰 수행 시 세부 평가 항목으로 질의를 진행한 경우 개발자 또는 테스터들이 이해하기 어려웠다. 반면에, 진단 영역별로 구분하여 인터뷰 및 결과 발표 시 업체의 이해도가 더욱 높았다. 표 2는 진단 영역을 언급한다.

진단영역	내용
테스트 프로세스	테스트 수행 목표를 위한 수행 조직 구성, 구성원 역할 및 책임, 달성을 위한 세부 업무, 각 업무의 수행 절차 및 결과물
테스트 관리 및 모니터링	조직에서 정의한 테스트 단계별 업무 수행 과정 및 수행 경과를 계획과 비교
테스트 환경	테스트를 수행하기 위한 사무 환경, 소프트웨어 및 시스템 환경
테스트 조직	테스트 프로젝트를 수행하기 위한 사람 또는 조직
테스트 기술	테스트 프로젝트 수행에 사용되는 다양한 이론 및 방법

표 2. 진단 영역

3.2 시범 적용 결과

한국형 테스트 성숙도 모델을 국내 중소기업(A, B)에 적용한 결과는 아래와 같다. 그림 5. A사 진단 결과을 보인다.

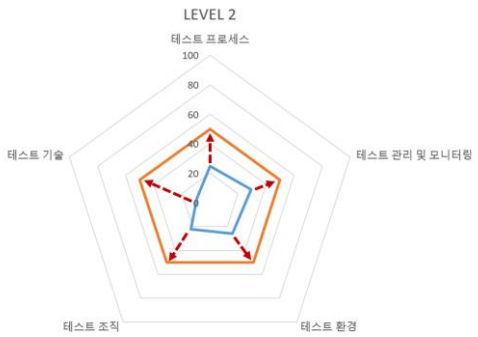
구분	진단 결과
A 사	

그림 5. A사 진단 결과

그림 6은 B사의 진단 결과를 보인다.

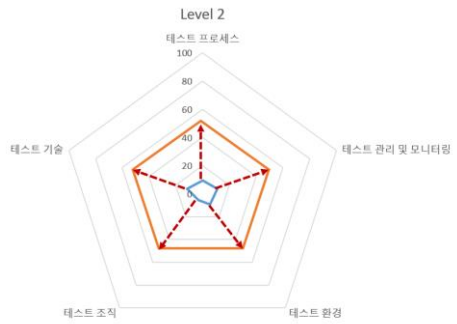
구분	진단 결과
B 사	

그림 6. B사 진단 결과

레벨 단위로 평가한 결과는 A사, B사 모두 레벨 2를 만족하지 못했다. 평가 결과에 대한 세부 점수 및 영역별 진단 내용은 대외비로 본 논문에서는 공개하지 않는다. 다면, 진단 결과 그래프에 표시된 것과 같이 A사의 경우에는 테스트 기술과 테스트 조직에 대한 보완이 필요했으며, B사의 경우는 테스트 조직(담당자) 중심으로 관련 인원의 보강이 필요하며, 전반적으로 낮은 수준으로 평가되었다. A, B사 공통적으로 테스트 조직 또는 담당자에 대한 인력 충원이 필요한 상태였으며, 테스트 능력 향상을 위해 교육 및 관련 지원이 필요한 상황이었다. 하지만, 시범 적용에 참여한 중소기업 모두 개선의 의지가 높았으며, A사의 경우 심사 후 관련 인력을 충원하였다. B사의 경우는 시범 적용 후 테스트 관련 교육 및 도구에 대한 조사를 수행하여 테스트 기법에 대한 보완할 수 있도록 준비하였다. 최종적으로 시범 적용 과정을 통해 수준 진단뿐만 아니라, 국내 중소기업에서 사용 가능하도록 단계별 샘플 문서가 필요함을 파악할 수 있었다.

5. 결론 및 향후 연구

본 논문에서는 국내 중소기업에 적용 가능한 한국형 테스트 성숙도 모델의 시범 적용하여 적용 가능 여부 및 진단 결과를 통해 개선 방안을 도출하였다. 개선 방안을 통해 테스트 활동과 관련하여 기업에 우선적으로 필요한 사항들을 도출하여 제안할 수 있었다. 또한, 기업 내에 존재하는 다양한 기준들을 재확인하여 테스트에 재사용될 수 있도록 지도할 수 있었다. 현재, 레벨 2~3만을 기준으로 심사를 수행하였으나, 차후 레벨 4~5를 반영하여 심사에 적용할 예정이다. 또한, 기업 규모 등의 기업 조건을 고려한 다수의 심사 대상을 확보하여 많은 시범 적용 사례 및 결과 분석을 통한 개선과 기업에서 쉽게 활용 가능한 샘플 산출물 개발이 필요하다.

참고 문헌

[1] 박일준, “창조경제 실현을 위한 소프트웨어 혁신 전략”, 정책과 이슈, 산업연구원, 2014
[2] 홍익대학교, 한국정보통신기술협회, "한국형 테스트 성숙도 모델 개발에 관한 연구", 2014
[3] 홍익대학교, 한국정보통신기술협회, "한국형 테스트 성숙도 모델 개발에 관한 연구", 2015
[4] Ilene Burnsteine, “"Practical Software Testing"", Springer, 2003
[5] Gerrit de Vries, “The What And How of Testing TPI next and TMap Next Related”, Sogeti, 2010
[6] 김기두, “테스트 프로세스 성숙도 향상을 통한 테스트 성숙도 모델(TMM) 개선에 관한 연구”, 홍익대학교 석사학위 논문, 2014
[7] Bo Kyung Park, So Young Moon, Kidu Kim, Woo Sung Jang, R. Young Chul Kim, C. R. Carlson, “Refining an Assessing Model for Simplified TMM”, 2016 PlatCon, 2016

소프트웨어 변경의 테스트 범위를 정의하기 위한 산출물 정보 분석*

최효린 [1O], 이정원 [2], 이병정 [1]

서울시립대학교 컴퓨터과학과 [1] 아주대학교 전자공학과 [2]

{yoinoichr2015, bjlee}@uos.ac.kr[1], jungwony@ajou.ac.kr[2]

Artifact information analysis for defining test scope of software change

Hyorin Choi[1O], Jung-Won Lee[2], Byungjeong Lee[1]

Dept. of Computer Science, University of Seoul[1],

Dept. of Electrical and Computer Engineering, Ajou University[2]

요 약

산업 전반적으로 소프트웨어의 규모가 거대하고 복잡해지면서 소프트웨어 테스트의 중요성이 대두되고 있다. 그러나 개발 프로젝트의 규모가 클수록 지속적인 개발을 수행하는데, 변경된 시스템에 대한 테스트 항목을 판단할 때 개발자의 직관에 의존하게 되는 문제가 있다. 본 논문에서는 테스트 명세 정보를 기반으로 소프트웨어 변경점에 대한 분석을 수행하고, 변경된 시스템의 테스트 범위를 정의하기 위한 추적성 정보 모델을 제안한다. 개발 프로젝트의 산출물과 관련된 테스트 스위트를 분석하여 시스템의 실행 정보를 식별하고, 사례 연구에서 실제 작성된 문서들로부터 데이터집합을 생성하는 과정을 보인다. 그리고 다른 연구와의 비교 분석을 통해 제안하는 추적성 정보 모델의 정성적 평가를 수행한다. 본 모델을 통해 소프트웨어 변경점으로부터 적절한 테스트를 설계할 것으로 기대한다.

1. 서 론

오늘날의 산업 전반적으로 소프트웨어의 필요성이 대두되면서, 만족시켜야 할 소프트웨어의 요구사항도 점차 복잡해지고 있다[1]. 고객들은 그들의 복잡한 요구들을 모두 만족하면서, 또 제대로 동작하기를 바라는데, 이에 따라 개발된 시스템의 신뢰성을 판단하는 소프트웨어 테스트 역시 중요해지고 있다.

소프트웨어의 규모가 커질수록 개발 프로젝트의 형태는 많은 개발자들이 협업을 거쳐 지속적인 개발을 수행하는 방향으로 변화되어왔다. 시스템이 잘 개발되기 위해서는 시스템의 구조나 행동양식에 대한 개발자 간의 이해를 일치시키는 것이 중요한데, 이를 도와주는 방법으로 OMG에서 제정한 UML 다이어그램[2]이 있다.

UML 다이어그램은 개발될 시스템의 구조를 다양하게 표현할 수 있다. 개발자들은 고객의 요구에 대한 이해를 기반으로 소프트웨어 요구사항 명세서(Software Requirement Specification, SRS), 소프트웨어 설계 기술서(Software Design Description, SDD) 등 개발 산출물을 통해 설계 내용을 다각적으로 기술한다[3].

모델 기반 테스트(Model-based Testing, MBT)는 설계 문서에 기술된 UML 다이어그램에 기반한 테스트 수행 방법으로서 제안되었다. 모델로부터 시스템의 실행 정보(Execution data)를 파악하고, 개발된 시스템의 테스트 전략을 수립한다. 모델 기반 테스트는 자동화된 테스트 설계를 지원하는데[4], 이를 통해 큰 규모의 시스템의 테스트 설계 작업에 소모되는 막대한 노력과 비용을 줄일 수 있게 되었다.

그러나 모델 기반 테스트는 미리 작성된 모델에 기반하기 때문에, 변화에 대한 능동적인 테스트를 수행하기에는 어려움이 있다. 소프트웨어 유지보수(Software Maintenance)를 위한 시스템 변경이나 새로운 기능의 추가, 결함 수정(Fault Fixing), 하드웨어(Hardware)의 교체로 인한 소프트웨어 변경[5] 등 새로운 요구사항에 대한 적절한 소프트웨어의 변경점(Change Point, CP)을 찾고, 기존 모델에 반영하여 새롭게 테스트해야 한다.

이를 개발자가 직접 수행한다는 것은 상당한 노력과

* 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보 컴퓨팅기술개발사업(NRF-2014M3C4A7030504)과 서울시의 재원으로 수행한 서울시 창조전문인력 양성사업(No.CAC1510)의 지원을 받아 수행된 연구임.

시간이 필요하다. 또한, 시스템의 변경을 성공적으로 수행하였더라도 그 변경점 역시 새로운 결함이나 시스템의 원치 않는 동작(Unwanted Behavior)을 야기할 수 있기 때문에, CP를 파악하고 CP가 영향을 끼칠 수 있는 범위를 분석하는 과정이 추가로 필요하다. 이러한 작업 역시 개발자의 판단에 의존하기엔 많은 위험성[6]이 내재되어 있으며, 변경된 시스템의 전체 신뢰도 하락을 유발한다.

연구 목적

따라서, 요구사항의 변경에 따른 CP의 변경 영향 분석(Change Impact Analysis, CIA)을 수행하고 변경된 시스템의 적절한 테스트를 설계하는 연구가 필요하다. 본 논문에서 제안하는 방법은 다음과 같다. 소프트웨어 설계 문서, UML 다이어그램, 그리고 테스트 스위트의 메타데이터를 분석하고, 이들의 연관성을 추적성 정보 모델(Traceability Information Model, TIM)로서 표현한다. 이를 기반으로 테스트 명세 정보(Test Specification)을 정의하고, CP를 특정하여 적절한 테스트 범위를 정의하는 방법을 소개한다. 사례 연구에서는 의미기반 테스트 지원 도구(Content-Based Testing system, CBTs)[7] 개발 프로젝트의 산출물에서 데이터집합을 추출하는 과정을 보여주어, 시스템의 설계 문서와 실제 개발된 시스템과의 연관성을 파악하고 관련된 테스트 스위트를 정의하는 과정을 살펴본다.

본 논문의 기여도는 다음과 같다.

- 제안하는 추적성 정보 모델을 통해 개발 산출물의 테스트 명세 정보를 분석할 수 있다.
- 소프트웨어 변경점을 특정하고, 변경된 시스템의 적절한 테스트 범위를 판단할 수 있다.

본 논문의 전체적인 구성은 다음과 같다. 2장은 논문의 이해를 돕기 위한 배경 지식에 대해 소개하고, 3장에서 관련된 연구를 요약한다. 4장에서는 TIM의 작성 방법과 본 연구에 필요한 TIM 구조를 제시하고, 5장에서 사례연구로서 CBTs의 개발 산출물에서 테스트 명세 정보를 분석하는 과정을 기술한다. 6장에선 본 논문의 정성적 평가를 수행한다. 마지막으로 7장에서 결론과 향후 연구방향을 정리한다.

2. 배경 지식

이 장에서는 본 연구의 이해를 돕기 위한 주요 배경 지식을 기술한다.

2.1 변경 영향 분석

변경 영향 분석이란, 소프트웨어 개발 프로젝트에서 필요에 따라 빈번하게 발생하는 소프트웨어 요구사항 변경(Software Requirement Change)을 시스템에 반영할 때, 실질적인 소스 코드(Source Code)의 변경이나 변수 값, 논리구조의 변경 등이 일어나는 요소들(Elements)을 분석하는 기술이다. 이는 소프트웨어 개발 분야의 가장 산업집적인(Industry-integrated) 부분이며, 소프트웨어 유지보수에 필요한 시간, 인력 비용과 노력을 산정할 수 있는 정량적 분석 방법으로써 현재에도 많은 연구가 진행되고 있다[9-12].

1986년, Horqwitz[13]가 제안한 SODOS(SOftware DOcumentation Support) 방법 이래로 요구사항의 변경을 분석하기 위한 여러 방법들이 제안되었다. 먼저, 전통적인 방법으로 정적 코드 기반 분석(Static code-based analysis) 방법이 있다. 이는 변수의 선언-사용 관계 또는 함수 호출 등, 소스 코드를 기반으로 영향이 미칠 범위를 특정하는데, 일반적으로 너무 많은 범위를 변경 가능성이 있는 영향 집합(Impacted Set, IS)으로 판단하여 활용하기에 어려움이 있었다. 정적인 방법을 보완하기 위해 변경된 소스 코드와 관련된 시스템의 실행 정보를 동적(dynamic) 방식으로 분석하여 영향 집합의 범위를 축소하는 방법이 제안되었지만, 이는 부정 오류(False-negative)가 발생할 가능성이 있다는 치명적인 문제가 있었다.

따라서, 최근에는 위 두 가지 방법의 단점을 보완하고 효과적으로 실제 변경 집합(Real Impacted Set, RIS)을 추정하기 위한 연구가 진행되고 있으며, 본 논문에서는 동적 분석에 모델 기반 테스트를 접목한 분석 방법을 제안한다. 요구사항과 관련 테스트 정보를 활용하여 시스템의 실행 정보를 분석하고, 이를 통해 RIS를 추정하고 관련된 테스트 스위트를 식별한다. 제시하는 방법에는 분석을 용이하게 하는 추적성 정보 모델이 있으며, 자세한 내용은 4장에 기술하였다.

2.2 추적성 정보 모델

추적성 정보란, 소프트웨어 개발에 전주기적으로 사용되는 문서 또는 소스 코드 간의 종속성을 나타내는 정보이다. 추적성 정보 모델은 이러한 정보를 정형적인 표현기법으로 기술함으로써, 상호 간 이해를 돕는 데 사용된다. 그림 1은 [5]의 TIM 일부를 기술한다.

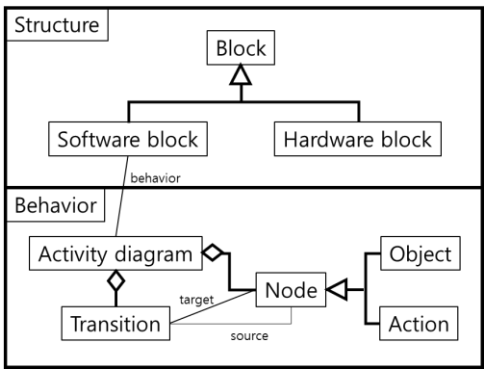


그림 1. 추적성 정보 모델의 예

시스템의 정적인 구조를 표현한 블록 다이어그램(Block Diagram) 의 소프트웨어 블록(Software Block)은

활동 다이어그램 (Activity Diagram)과 서로 활동 (Behavior) 관계를 가지는 것을 확인할 수 있다.

본 논문에서는 TIM을 테스트 명세 정보의 메타데이터 간 추적성을 표현하는데 활용하였다. 제안하는 TIM이 개발 산출물로부터 테스트 명세 정보의 분석을 돕고, 더 나아가 향후 연구로서 분석된 테스트 스위트의 정정 자동화에 활용될 것이라 기대한다.

3. 관련 연구

CP를 특정하는 방법은 다양한 논문에서 제안되고 있다. 이 장에서는 해당 연구들의 접근 방법들을 요약하고, 본 연구의 목적과 적합성을 판단해 본다.

[5]는 소프트웨어와 하드웨어 동작 모두를 포함한 시스템에서 적절한 영향 범위를 추정하기 위해 IR 기법을 적용한 방법을 제안하였다. 이 방법에서는 시스템의 구조적 모델과 행동 모델을 각각 SysML로 표현하고, 모델에 표현된 요구사항, 함수 명, 시스템 블록 명 등을 정보 검색의 데이터집합으로 활용하였다. 하지만 시스템의 구조적 모델과 행동 모델 간 연관성을 모두 상세하게 표현되어야 한다는 조건이 필수적이었다. 이는 개발 시스템의 규모가 커질수록 표현이 필요한 연관성 정보가 지수적으로 증가하기 때문에 실질적으로 활용하기엔 어려움이 있으며, 추가적인 테스트 설계를 수행하여야 하는 문제가 있다.

[10]은 [9]에서 제안된 코드 기반 분석에서 행위자가 시스템을 활용하는 조건 및 활동들을 'Obligation'이라는 개념을 제안하였다. 행위자가 시스템의 모든 동작을 일으키는 시점들을 실행 정보로서 파악한다는 점에서 동적인 코드 기반 분석의 부정 오류 문제를 해결할 방안으로서 효과적임을 보여준다. 그러나 테스트 설계는 시스템 동작의 시작점 이외에도 함수의 호출, 파일 참조 등 다양한 접근 위치에서 수행되어야 하기 때문에, 체계적인 테스트를 수행하기엔 식별되어야 할 시스템 실행 정보가 부족하다는 문제가 있다.

[11]은 소프트웨어 산출물의 정보들을 함축한 단어를 기반으로 CP를 추정하는 방법을 제안하였다. 모델이나 파일 명, 함수 명과 같은 정보들은 해당 단위를 설명할 함축적인 단어로 기술되어 있으며, 설명 역시 자연어로 기술된다는 점에 착안하여, IR 기법의 데이터집합으로서 소프트웨어 산출물들의 정보들을 수집하고 TIM과 같이 구성한다면 어떠한 질의(Query)가 주어질 때 관련성이 높은 요소들부터 차례대로(Ranked) 검색하고 이를 EIS로 만드는 방법을 제안한다. 이 방법은 본 논문에서 사용된 접근 방법과 유사하지만, [11]에서 구성된 데이터집합은 SysML로 표기가 완료된 프로젝트에 한해 적용가능하며, 또한 협업과정에서 각 개발자 간의 명명 형식이 달라진다면 정확도가 하락할 수 있다는 한계가 있다. 본 논문은 연구의 목적에 따라 [11]을 보완한 새로운 데이터집합을 제안한다.

[12]는 세 가지 주요 CIA기법들을 통합한 분석 방법을 제안하였다. 코드 기반 분석에 정보 검색 기법을 적용하면서, 데이터집합으로 시스템의 동작을 표현한 행동 모델과 시스템의 정적 정보를 가진 구조(Structural) 모델의 연관성 정보를 기반하였다. 추가로 오픈 소스 라이브러리로부터 개발 프로젝트의 요구사항 변경점과 관련될 확률이 높은 요소를 학습하였다. 제안된 기법은 각 기법의 단점을 상호 보완하여 CIA의 정확도를 효과적으로 상승시킬 수 있는 분석 방법이지만, 역시 테스트 설계를 추가적으로 수행해야 하며, 변경점을 특정하는 구성요소의 단위가 단순하다는 문제가 있다.

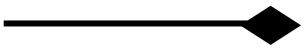
4. 테스트 명세 정보의 분석 방법

이 장에선 구조화된 테스트 명세 정보의 추적성 정보 모델을 제안한다. 우선, TIM의 표기 방법에 대해 기술하고, 각각의 메타데이터를 정의한다. 또, 산출물의 데이터집합을 어떻게 구성하는지에 대한 설명을 수도 코드로 표현하였다.

4.1 추적성 정보 모델 표기 방법

추적성 정보 모델의 표기 방법은 표 1로 정리하였다.

표 1. 추적성 정보 모델 표기법

TIM Element	Note
	Configuration
	Trace
	Type
	Component
	Dependency
	Mapping

항목(Configuration)은 정의된 정보에 대한 내용을 추적 요소로서 구성할 수 있을 때, 이를 사용한다. 빈 사각형 내부에 정의하고자 하는 정보를 적는다.

추적(Trace)은 TIM에서 기본적인 추적성을 나타내는 방법으로, 여러 항목 간에 방향성이 없는 연관성을 나타낼 때 주로 사용된다. 실선과 함께 간단한 자연어로 연관성에 대한 내용을 기술한다.

속성(Type)은 어느 항목이 다른 항목을 구성하는 속성값임을 의미한다. 구성요소(Component)는 속성과 비슷하게 사용되는데, 항목 간의 계층적 구조를 표현할 수 있다. 실질적인 추적관계를 구성하진 않지만, 의미적 연관성을 가질 경우 의존(Dependency)관계를 사용한다. 의존이 방향성을 가지면 맵핑(Mapping)을 사용한다.

4.2 테스트 추적성 정보 모델

본 논문에서는 소프트웨어 개발 전주기적으로 작성 및 관리되는 요구사항 명세 또는 설계 문서로부터 시스템의 실행 정보, 요구사항 분류 별 정보, 시나리오 등을 추출하고, 시스템 설계에 대해 정형화된 표현 방법으로 작성된 행동 모델 정보, 또 이를 통해 생성된 테스트 스위트 등 관련된 모든 정보들을 테스트에 필요한 명세 정보로 정의하였다. 정의된 정보들을 TIM 작성 방법으로 표현하면 그림 2와 같이 나타낼 수 있으며, 테스트 추적성 정보 모델이라 명명한다.

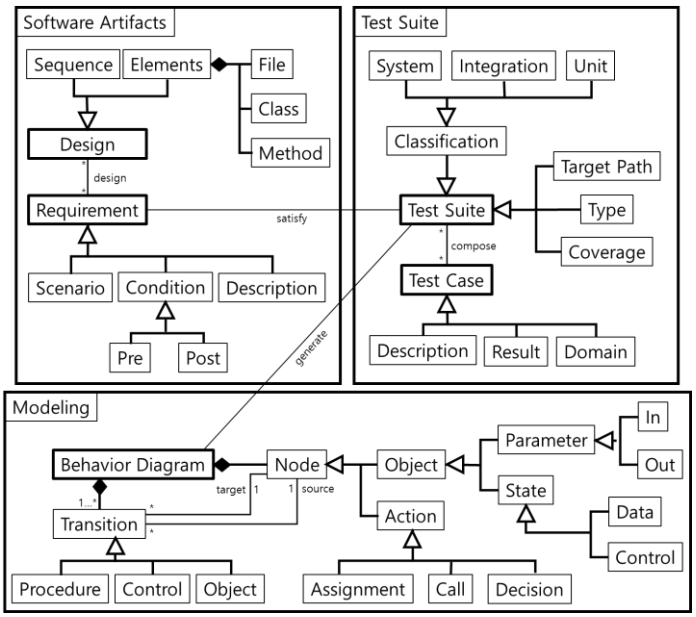


그림 2. 테스트 추적성 정보 모델

제안하는 TIM은 크게 세 부분으로 나뉘어진다. 아래 절에서 각 부분이 가지는 정보를 자세히 기술한다.

4.2.1 소프트웨어 산출물(Software Artifacts)

소프트웨어 산출물이란, 개발 프로젝트를 진행하면서 소프트웨어공학적 기법에 따라 작성되는 요구사항 명세서, 실제 개발될 시스템에 대한 상세한 내용을 서술적, 또는 정형적으로 기술하는 설계 기술서, 개발에 필요한 모든 정량적 요소들을 정의한 설계 명세서 등의 문서들을 총칭한다.

소프트웨어는 필요에 의해 개발된다[6]. 이러한 특성 때문에 소프트웨어 개발 프로젝트에는 고객의 막연한 필요를 구체화하는 작업이 필수적인데, 이는 개발자에 의해 요구사항(Requirement)으로 자세하게 기술된다. 이 때, 개발 설계에 따른 요구사항 분석 기법 에 따라 내용이 다양한 관점으로 기술되며, 명세서에서 필수로 작성되어야 할 3가지의 속성을 정의하였다.

시나리오(Scenario)는 시스템이 수행되어야 할 가장 기본적인 기능들에 대한 서술적 표현이다. 변경될 요구사항의 분석에서 기준이 되는 내용으로서, 가능한 한 상세하고 명확한 용어로 작성되어야 한다.

조건(Condition)은 해당 요구사항이 어떠한 시스템의 상태, 사용자의 입력 등의 조건이 필요한 것인지를 서술한다. 조건에는 두 가지 형태로 존재하는데, 사전 조건(Pre-Condition)과 사후 조건(Post-Condition)이 그 형태이다. 이들은 각각 해당 요구사항이 수행되기 이전 또는 이후에 수행되어야 하는 작업 또는 상태 등을 의미한다. 설명(Description)은 각 요구사항의 이해를 돕기 위해 자연어로 작성하는 항목이다.

이렇게 작성된 요구사항들은 실제 개발될 시스템의 각 부분들로서 설계(design) 작업을 수행하는데, 생성된 설계사항(Design)들은 요구사항과 다 대 다(n-to-n) 관계를 가진다. 설계사항은 두 가지 속성을 가진다. 요구사항을 하나의 전체적인 기능으로서 보는 시스템적 처리 과정(Sequence)에 대한 설계와, 개발에 사용되는 언어적 특성과 개발방법론에 따라 상세하게 분류되는 구성요소(Elements)적 설계가 있다. 특히 구성요소적 설계에서는 각 파일 명(File), 클래스 명(Class), 함수 명(Method)들이 각각 식별된다.

소프트웨어 산출물은 개발된 시스템에 대한 전반적인 이해를 증진시키는 정보를 포함한다. 이는 요구사항의 변경이 가지는 특성들을 분석하고, 이를 통해 CP를 특정하기 위해 필수적으로 포함되어야 할 정보이며, 테스트에 필요한 데이터집합으로서 기능한다. 요구사항들은 그 목적에 따라 테스트 스위트의 목적 경로와 관련되며, 만족(satisfy)관계를 가진다.

4.2.2 테스트 스위트(Test Suite)

테스트와 관련된 실질적인 사용 데이터를 메타데이터 형식으로 정의하였다. [15]는 소프트웨어 생명주기에 따른 테스트의 구분을 정의하였는데, 본 논문에서도 이 정의를 채용하면서 추가적으로 테스트를 수행하기 위한 개발자의 판단 정보를 포함한다.

테스트 스위트(Test Suite)란 특정 목적을 가지고 생성된 테스트 케이스들을 다루기 위한 하나의 단위다. 동일한 테스트 목적을 공유하기 때문에, 각 테스트의 구분(Classification)과 함께 행동 모델로부터 생성된 목적 경로(Target Path), 타입(Type), 그리고 커버리지(Coverage)를 포함한다.

테스트 케이스(Test Case)는 프로젝트에서 생성한 모든 테스트 케이스를 의미하며, 테스트 스위트와 다 대 다 포함(compose) 관계를 가진다. 설명(Description)과 테스트 결과(Result), 사용 가능한 입출력 값의 범위(Domain)를 속성으로 가진다.

테스트 스위트는 소프트웨어 변경점에 대한 재귀 테스트로서 활용되며, 추정 영향 집합(Estimated Impacted Set, EIS)에 대한 새로운 테스트 스위트를 식별하고, 생성할 때에서도 사용될 것이다. 시스템의 행동 모델로부터 모델 기반 테스트를 수행하기 때문에, 모델링 정보와 생성(generate)관계를 가진다.

4.2.3 모델링(Modeling)

본 논문의 이전 연구에서는 시스템의 행동 모델들이 표현하는 행동 양식으로부터 테스트 케이스 생성에 필요한 정보들을 식별해내는 연구를 진행하였다[8]. [8]의 제안 방법을 통해 소프트웨어 설계 기술서에서 사용되는 여러 행동 모델들이 공통된 정보를 포함하고 있다는 것을 확인하였고, 이를 일관성 있는 양식으로 표현해냄으로써 총 3가지의 모델로부터 동일한 테스트 경로를 탐색할 수 있다는 점을 증명하였다.

본 논문의 TIM에서는 이러한 연구를 바탕으로 행동 모델의 메타데이터를 위와 같이 정의하였다. 행동 모델에는 가장 기본적으로 2가지의 요소로 구성된다. 노드(Node)와 전이(Transition)인데, 표현 모델에 따라 각기 다른 속성들을 가진다.

노드는 기본적으로 시스템의 행위(Action)에 대한 내용을 의미한다. 시스템의 행위에는 할당(Assignment), 호출(Call), 분기(Decision) 3가지의 속성이 있다. 할당이란 시스템의 동작에 사용되는 파라미터 값 등에 임의로 값을 부여하는 행위, 호출은 함수나 변수를 메모리로 불러오는 행위, 분기는 어떠한 조건의 만족 여부에 따라 상이한 작업의 수행을 처리하기 위한 행위를 나타낸다.

또한, 노드는 파라미터 값의 입력 또는 출력을 나타낼 수 있으며, 시스템의 상태(State)에 대한 표현 역시 가능하다. 시스템의 상태 정보는 임의 객체(instance)의 데이터를 정의하거나, 분기문과 함께 시스템 제어를 수행하는 장치로서 사용된다.

전이는 노드와 노드 간의 연관성을 보여준다. 전이는 속성으로 처리 순서(Procedure), 제어(Control), 그리고 객체(Object)를 가진다. 처리 순서는 시스템의 노드를 거쳐옴에 따라 변화하는 상태 정보를 의미한다. 제어는 다음 작업이 진행되기 위해서 특정 조건을 만족시켜야 하는 경우, 이를 전이 조건의 형태로 정보를 포함한다. 마지막으로 전이 속성으로서의 객체는 전이 그 자체로서 어떠한 시스템의 동작을 표현할 때(예를 들면, 상태 다이어그램에서 상태 간 변화를 전이로서 나타내는 경우), 해당 정보를 포함하고 있음을 의미한다.

시스템 행동 모델이 내포하는 공통적인 정보는 위의 두 요소를 통해 식별이 가능하며, 설계로부터 테스트 케이스를 생성하기 위한 모델의 정보를 정의하였다.

4.3 테스트 명세 정보 분석 방법

이 절에서는 4.2절에서 정의된 테스트 명세 정보를 분석하고, 소프트웨어 변경이 일어날 위치를 추정하는 알고리즘을 소개한다. 알고리즘은 다음 그림 5와 같이 정리한다. 먼저, 분석할 시스템에 대한 산출물들을 입력 값으로 받아온다. 입력된 정보들은 분석을 위한 데이터집합으로 생성하는데, TIM에서 분류한 세 분야를 각각 하나의 집합으로 구성한다. SA(r,d)는 SRS, SDD에 기술된 요구사항과 설계 요소를 의미하고, M(n,t)는 모델의 노드와 전이, TS(tp,c,tc)는 테스트 스위트에서 목적 경로, 분류, 테스트 케이스를 의미한다.

그런 다음, 분류된 집합 간의 추적 정보를 구성한다. 이들 정보는 요구사항(SA(r))과 관련된 테스트 목적을 가진 테스트 스위트(TS(c))를 'satisfy' 관계로 정하고, 요구사항을 표현한 모델(M(n,t))에서부터 목적 경로 탐색 후 테스트 케이스(TS(tc))를 'generate' 관계로 설정한다.

Input Software Artifacts

1. **Create** initial dataset.
 1) Collect 'Software Artifact(SA)' requirement (r), design(d) from SRS, SDD, to SA(r,d)
 2) Collect 'Modelling(M)' node(n), transition(t) from all behavioral diagram, to M(n,t)
 3) Collect 'Test Specification(TS)' target path (tp), coverage(c), test case(tc) data using Path-wise Testing, to TS(tp,c,tc)
2. **Construct** Traceability Information.
 1) Get TIM elements, then Find 'satisfy' link between SA(r) and TS(c).
 2) Explore M(n,t), then Connect to TS(tp).
 3) Generate TS(tc) based on SA(d), and TS(tp).
3. If Query(w), **Find** all related SA(r,d) with 'satisfy' link.
4. **Get** change information of Query(w), then Apply to TS(tc).
5. **Create** Estimated Impact Set using SA, TS data.

Return EIS(SA, TS).

그림 3. 정보 분석 알고리즘

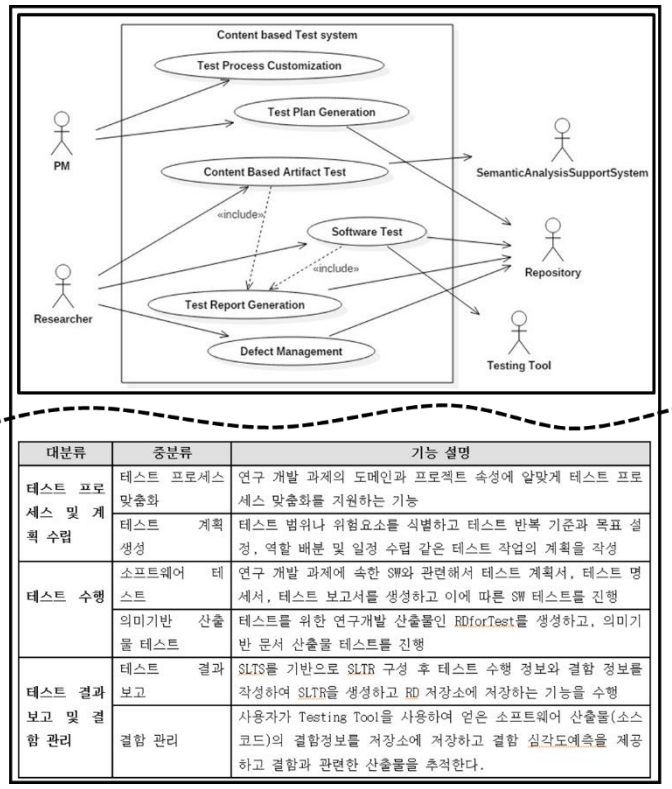
이렇게 데이터집합을 모두 식별했다면, CP를 특정할 수 있다. Query(w)는 요구사항의 변경을 자연어(w)로서 기술한 내용이며, 변경된 시스템의 CP를 추정하기 위해 우선 Query(w)와 관련된 모든 요구사항을 식별한다. 각 요구사항들은 연결된 'satisfy' 관계를 가진 모든 TS 정보를 가져오며, 이들의 분석을 통해 CP가 입력될 파라미터 값이 변경될 경우 TS(tc)에 반영한다. 반영된 모든 TS(tc)들은 EIS로서 SA와 함께 출력되고, 알고리즘을 종료한다.

5. 사례 연구

사례 연구로는 본 논문에서 제안하는 추적성 정보 모델을 실제 프로젝트에서 작성된 개발 산출물에 적용해 봄으로써 각 데이터집합이 잘 구성되는지 분석한다. 예시로서 사용되는 개발 프로젝트는 의미기반

테스트 지원 시스템(CBTs)[7]의 구현 프로젝트이며, 본 논문의 연구목적에 부합됨을 보이기 위해 일부 기능을 사례 연구로서 활용하였다.

CBTs의 소프트웨어 산출물 작성 과정은 다음과 같다. 시스템에 필요한 기능을 요구사항으로서 자연어로서 기술하고, 이를 사용사례(Use Case)로 분석한다. 그리고 각 사용사례의 사전, 사후 조건에 대한 내용, 간단한 설명, 행위자를 식별하고, 시퀀스 다이어그램(Sequence Diagram)으로 표현하였다.



대분류	중분류	기능 설명
테스트 프로세스 및 계획 수립	테스트 프로세스 맞춤화	연구 개발 과제의 도메인과 프로젝트 속성에 알맞게 테스트 프로세스 맞춤화를 지원하는 기능
	테스트 계획 생성	테스트 범위나 위험요소를 식별하고 테스트 반복 기준과 목표 설정, 역할 배분 및 일정 수립 같은 테스트 작업의 계획을 작성
테스트 수행	소프트웨어 테스트	연구 개발 과제에 속한 SW와 관련해서 테스트 계획서, 테스트 명세서, 테스트 보고서를 생성하고 이에 따른 SW 테스트를 진행
	의미기반 산출물 테스트	테스트를 위한 연구개발 산출물인 RDforTest를 생성하고, 의미기반 문서 산출물 테스트를 진행
테스트 결과 보고 및 결함 관리	테스트 결과 보고	SLTS를 기반으로 SLTR 구성 후 테스트 수행 정보와 결함 정보를 작성하여 SLTR을 생성하고 RD 저장소에 저장하는 기능을 수행
	결함 관리	사용자가 Testing Tool을 사용하여 얻은 소프트웨어 산출물(소스코드)의 결함정보를 저장소에 저장하고 결함 심각도예측을 제공하고 결함과 관련한 산출물을 추적한다.

그림 4. ‘CBTs’의 소프트웨어 산출물 예시

그림 4는 SRS에서 작성된 내용으로, 위에서 설명한 작성 과정의 상세한 내용이 기술된다. 사용자의 요구사항은 세부적인 기능에 대한 설명과 함께 유즈케이스 다이어그램 (Use-case Diagram)으로 나타낸다. 그리고 ‘대분류’, ‘중분류’, ‘기능’으로 단계적 구분을 수행하는데, 대분류는 사용사례를 총칭하며, 중분류는 사용사례로부터 기능적으로 분리된 단위, 그리고 기능은 설계를 위한 요구사항의 세부 단위다.

SDD에는 시스템의 구조에 대한 설계 내용을 작성하였다. 전체적인 시스템 구조의 설명과 SRS에서 분석된 단계적 분류를 각 패키지로 구성하고, 패키지의 구성요소들이 기술되며, 화면 및 소프트웨어 인터페이스 설계 등의 내용을 명세화한다. 개발에 대한 모든 설계를 다양한 관점에서 상세히 기술하기 때문에 막대한 양의 데이터집합으로 구성되어 있으며(한글 기준 약 200p 분량), 테스트와 관련된 정보로서 식별해야 할 데이터집합은 소프트웨어 아키텍처와 패키지 설계 내용이 된다. 산출물 구성에 대한 보다 자세한 내용은 [7]에서 기술한다.

표 2. CBTs의 소프트웨어 산출물

	Scenario	Use case	Actor	Function
SRS	14	6	5	41

	Sequence	File	Class	Method
SDD	38	107	52	250

표 2는 CBTs으로부터 테스트 추적성 정보로 사용될 정보로서 식별된 내용을 정리한다. 요구사항은 14개, 사용사례 6개, 행위자 역할 5 종류, 그리고 시스템 설계를 통한 41개 기능 (Function)으로 구성되었다. 시퀀스(Sequence)는 각 기능들이 수행되기 위해 사용되는 파일, 함수들의 순서를 의미하며, 총 38개의 시퀀스가 있다. 파일은 각 클래스와 상위 인터페이스, 데이터베이스 관련 파일의 수이며, 107개가 있다. 클래스와 함수는 각각 52개, 250개로 구성된다. 사례연구에서는 이 중, ‘테스트 계획 생성(UC2)’ 기능에 대한 정보를 수집하였다.

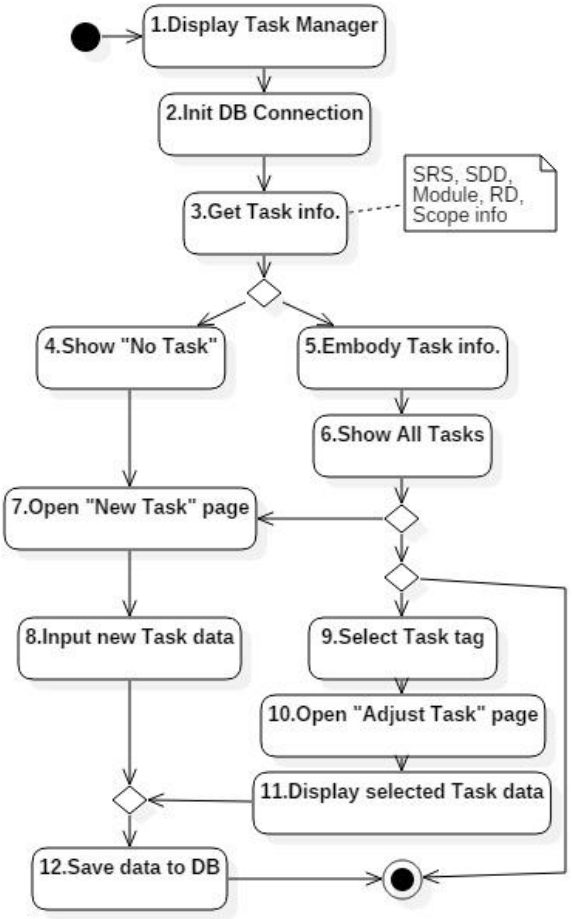


그림 5. ‘UC2’의 행동 모델

모델링과 테스트 명세 정보는 기술자의 테스트 설계 과정에 따라 작성되는 정보량에 상당한 차이가 있다. 본 사례연구에서는 'UC2'의 시스템 레벨 테스트 설계를 사용한다. 예시로 사용된 모델은 'UC2'의 사용사례와 관련된 정보들이며, SRS, SDD로부터 'UC2'와 관련된 정보를 각각 추출하고, 이들 행동 구조를 액티비티 다이어그램(Activity Diagram, AD)으로 나타내었다. 그림 5는 'UC2'의 모델 정보이다.

'UC2'와 관련된 기능 분류로는 총 5가지 기능이 있다. 'Task'에 대한 정보를 데이터베이스로부터 가져오기 위한 기능으로 그림 5에서 1,2,3번으로 나타나는 작업 'F-1', 조회된 'Task'가 없음을 화면 상에 보여주는 기능으로 그림 5의 4번 작업인 'F-2', 조회된 'Task'를 화면 상에 보여주기 위해 그림 5의 5,6번 작업을 거치는 'F-3'이 있다. 새로운 'Task'에 대한 정보를 입력 받는 기능인 'F-4'는 그림 5에서 7, 8, 12번으로 나타내며, 기존의 'Task' 정보를 수정하는 작업은 그림 5에서 9,10,11,12번으로 나타내며 'F-5' 기능을 구성한다. 작업 3번의 알림(Notation)은 해당 작업이 수행되기 위해 입력되어야 할 값(Parameter)으로, 'Task'는 SRS, SDD, Module, RD(연구문서), Scope로 구성되어 있다는 내용을 표현한다.

그림 5와 같이 'UC2'의 시스템적 행동 모델을 구성하였을 때, [8]의 테스트 설계 방법에 따라 테스트 스위트(TS)를 표 3과 같이 구성할 수 있다. 사용된 커버리지는 간단 경로 커버리지(Simple Path Coverage, SPC)이며, 탐색된 테스트 경로는 각 테스트 스위트의 시나리오가 된다. 시스템 테스트를 수행하기 때문에 각 테스트 케이스는 시나리오와 동일하므로 생략하였다.

표 3. 'UC2'의 테스트 스위트

TS.	Req.	Class.	Cov.	TP.	Func.
1	UC2	System	SPC	S-1-2-3-4-7-8-12-E	1,2,3
2	UC2	System	SPC	S-1-2-3-5-6-7-8-12-E	1,3,4
3	UC2	System	SPC	S-1-2-3-5-6-9-10-11-12-E	1,3,5
4	UC2	System	SPC	S-1-2-3-5-6-E	1,3

* TS. : Test Suite, Req. : Requirement, Class. : Classification, Cov. : Coverage, TP: Target Path, Func. : Function

'UC2'의 테스트 명세 정보는 총 4개의 테스트 스위트로 구성된다. 테스트하려는 요구사항은 'UC2'이며, 시스템 테스트를 수행한다는 점, 그리고 SPC를 사용한다는 점은 동일하다. 그러나 탐색된 경로들이 포함하는 기능(Func.)은 각기 다르며, SPC를 기준으로 탐색된 4개 경로가 각각 테스트 스위트로서 구성되었다. 모든 경로는 시작점(S)와 종료점(E)이 있고, 그림 5의 각 노드가 순차적으로 경로를 구성하고 있으며, 이는 표 3에서 정리하였다. 설계된 테스트의 구분이 '시스템 테스트'이므로, 테스트 경로가 포함하는 기능은 요구사항의 분류로서 구분된 기능들이다.

CP의 테스트 범위

'UC2' 기능에 대한 소프트웨어 산출물, 모델 정보, 테스트 정보를 데이터집합으로서 구성하였다. 이를 통해, 요구사항의 변경이 데이터집합의 어떠한 요소로서 판단할 수 있다. 만약, 'Task를 구성하기 위한 정보 중에서, Scope 정보를 제외한다'라는 요구사항의 변경이 일어날 경우, 관련될 정보들은 다음과 같다.

- 그림 5의 3번 작업
- 3번 작업이 속한 기능 분류 'F-1'
- 'F-1'이 속한 요구사항 'UC-2'

일반적으로 'Task'를 구성하는 정보가 변경될 경우, 'F-1'의 기능이 제대로 동작하는지 테스트한다. 하지만 실제로 'Task'의 정보가 활용되는 기능들은 그림 5의 정보에 따라 4개 경로에서 사용되는 'F-2', 'F-3', 'F-4', 'F-5' 기능들이 모두 관련되어 있다. 따라서 Scope 정보를 제외한 시스템에서 테스트 되어야 할 기능은 UC2의 모든 관련 기능들이며, 테스트 스위트 1~4번의 기능을 모두 통과해야 한다.

6. 평 가

본 논문에서 필요한 연구 과제(Research Question, RQ)를 제시하고, 관련 연구들과의 비교 및 분석을 통해 정성적으로 평가를 수행하였다.

RQ1. *추적성 정보 모델은 잘 표현되었는가?*

A. 제안하는 TIM에서 사용되는 메타데이터들은 이전 연구인 [8]로부터 테스트 설계에 필요한 사용성 분석을 완료한 내용에 기반한다. 소프트웨어 산출물의 경우, CBTs를 구현하기 위해 실제 작성된 SRS, SDD에서 추출할 명세 정보들을 식별하였다. 시스템의 구조를 여러 행동 모델로서 표현하였는데, 이는 테스트 명세에서 사용될 테스트 분류, 목적, 커버리지 등의 정보와 함께 테스트 케이스의 명세 정보로서 구성 가능하다. 각 테스트 케이스가 시스템 내부적으로 어떠한 동작, 부분에 기여하는지에 대한 연관성을 TIM으로 구축한다면, CP에 대한 IS의 범위를 한정하고 이와 관련된 테스트 케이스를 가져오거나 변경하기에 용이하며, 새로운 요구사항은 기존의 테스트 명세로부터 테스트 케이스 생성 전략을 세울 수 있게 한다. 따라서 제안하는 TIM은 본 연구의 목적에 부합하는 적절한 의미정보로서 구성되어 있다.

RQ2. *제안하는 방법이 CP의 테스트 범위를 판단하기 위한 적절한 정보를 제공하는가?*

A. CIA는 시스템의 행동 정보를 기반하며, CP가 시스템 동작에서 어떠한 영향을 끼치게 될지, 그리고 그 범위가 얼마나 되는지 분석하는 데에 사용된다. 따라서 행동 정보를 무엇으로 구성하는지에 따라 CIA 기법들이 나뉘어지며, 이 과제의 평가를 수행하기 위해 기술된 3가지 접근 방법과 본 논문의 비교분석을 수행하였다.

표 4. RQ2 비교분석 표

	Code [9]	Obligation [10]	Integrated [12]	본 논문
정보 구성	File + Method	Actor + Control	Model + OSL*	Model + Doc* + TS*
적용 기법	CIA	CIA + IR	CIA + IR + DM*	CIA + TIM + PT*
도구 지원	O	X	O	X
한계점	Overly FP* range	Lack of test info.	Depends on naming rule	Model Consistency

* OSL : Open Source Library, Doc : SRS, SDD, TS : Test Specification, DM : Data Mining, PT : Path-wise Testing, FP : False-Positive

Code-based CIA[9]

행동 정보를 분석하는 가장 기본적인 방법은 시스템의 소스 코드를 분석하는 것이다. 하지만 코드 기반 분석은 많은 긍정 오류(false-positive)이 발생하는 문제가 있으며, 주요 기능의 경우 시스템 절반 이상을 EIS로 추정하게 되는 문제점[6]이 있다. 따라서, 코드 기반 방법을 그대로 사용하는 건 CP를 특정하기에 효과적이지 않다.

Obligation[10]

행위자의 접근 위치(Access Point, AP)를 시스템의 행동 정보로서 보는데, 이는 코드 기반 분석의 문제점을 보완하지만, 행위자로부터의 입력이 적고 센서에 의한 자동 제어를 기반한 시스템의 경우 내부적 실행 정보를 식별하지 않게 되므로 테스트를 설계하기에 까다롭다. 따라서 행위자의 접근 위치가 상세하다는 특정 구조의 시스템에 대해서만 상세한 테스트 설계가 가능하다는 한계를 가진다.

Integrated[12]

이 방법은 IS의 추정에 코드 기반 방법에 정보 검색 기법과 오픈 소스 라이브러리의 업데이트 정보를 함께 고려한다. 시스템의 실행 정보와 변경점의 유형 및 영향 분석을 위해 정보 검색 기법과 데이터마이닝 기법을 적용하였는데, 이는 코드 기반의 CIA에는 용이하지만 테스트 설계에 대한 어떠한 정보도 포함하지 않으므로 추가적인 분석 과정을 필요로 한다. 또한 코드에서부터 추출된 데이터집합을 기반하는데, 파일 명, 함수 명과 같이 개발자의 명명 조건에 의존하는 위험을 내재하고 있으며, 적절한 CP를 특정할 때 방해요소로 작용된다.

본 논문

본 논문은 시스템의 실행 정보를 분석할 때 개발 문서와 행동 모델, 테스트 설계를 고려한다. 요구사항 변경에 맞는 CP를 찾고, 관련된 실행 정보를 가져올 때 관련 테스트 정보까지 가져온다. 테스트 케이스에는 변경사항을 반영하여 자동으로 수정하고, 새로운 기능은 행동 모델과 기존 테스트 환경을 기반하여 새 테스트 케이스로 구성할 수 있다. 요구사항 정보와 가장 밀접한 정보를 기반으로 CIA를 수행하기 때문에, 변경된 시스템의 테스트를 자동으로 설계할 수 있다는 점에서 다른 논문과 비교할 때 가장 효율적이다.

7. 결 론

본 논문은 테스트 명세 정보를 구성하는 데이터 집합을 추적성 정보 모델로 표현하고, 개발 산출물들을 분석하여 시스템의 실행 정보를 파악하여 소프트웨어의 변경점을 특정하는 연구를 수행하였다. 사용되는 데이터 집합은 SRS, SDD 등 소프트웨어 개발 산출물에 작성된 요구사항과 구현을 위한 설계 정보, 다양한 관점에서 작성된 시스템의 행동 모델, 그리고 이 두 정보를 기반하여 생성된 테스트 스위트이다.

사례연구로서 CBTs의 개발 프로젝트를 소개하였는데, 이는 실제 개발 프로젝트에서 작성된 산출물을 통해 본 논문의 제안 방법을 적용함으로써 본 연구의 실효성을 증거할 수 있었다. 또한 본 논문의 평가를 위한 RQ를 제시하고 다른 연구들과의 정성적 평가를 수행하였으며, 결과로 본 논문이 효과를 보였음을 확인할 수 있었다.

그러나 본 논문이 제안하는 방법은 개발자가 생성한 테스트 명세를 기반으로 시스템 실행 정보를 판단하기 때문에, 실제 변경이 일어난 부분이 행동 모델의 구조를 변경하게 되는 경우 이를 반영하여 일관성을 유지해야 하는 문제가 있다. 또한, 요구사항의 변경이 특정된 CP로 표현하려면 개발자의 판단이 필요하다는 잠재적 위험이 존재한다. 향후 연구에서는 소프트웨어의 변경이 어떠한 데이터집합에 변경을 일으키는지를 설계하는 연관 규칙을 정의하고, 또 분석된 내용을 모델의 구조에 자동으로 반영하는 도구를 구현할 계획이다.

참고 문헌

[1] P. C. Jorgensen, *Software Testing: a Craftsman's Approach*, CRC press, pp. 3-52, 2013.

[2] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, John Wiley & Sons, pp. 41-84, 2011.

[3] M. Utting, and B. Legeard, *Practical Model-Based Testing: A Tools Approach*, pp. 1-18, 2010.
[4] C. Mingsong, Q. Xiaokang, and L. Xuandong, “Automatic test case generation for UML activity diagrams,” In Proc. of the 2006 International Workshop on Automation of Software test, pp. 2-8, 2006.
[5] S. Nejati, M. Sabetzadeh, C. Arora, L. C. Briand, and F. Mandoux, "Automated change impact analysis between SysML models of requirements and design," In Proc. of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 242-253, 2016.
[6] W. Chen, "A Hybrid Software Change Impact Analysis for Large-scale Enterprise Systems," McMaster University, 2015.
[7] S. Song, A. Dashbalbar, J. Lee and B. Lee, "Test Framework Requirements to Verify Artifacts in Software R&D Project," International Journal of Software Engineering and Its Applications(IJSEIA), Vol. 10, No. 11, pp. 83-94, 2016.
[8] S. Back, H. Choi, J. Lee, and B. Lee, "Evolutionary Test Case Generation from UML-Diagram with Concurrency," International Conference on Computer Science and its Applications, Vol.421, pp. 674-679, 2016.
[9] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," Journal of Software Testing, Verification and Reliability, Vol. 23, No. 8, pp. 613-646, 2013.
[10] I. Rubab, S. Ali, L. Briand, and V. Le Traon, "Model-based testing of obligations," 14th International Conference on Quality Software, pp. 1-10, 2014.
[11] Y. C. Cavalcanti, I. do Carmo Machado, P. A. D. M. S. Neto, and E. S. de Almeida, "Towards semi-automated assignment of software change requests," Journal of Systems and Software, Vol. 115, pp. 82-101, 2016.
[12] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated impact analysis for managing software changes," 34th International Conference on Software Engineering, 2012.
[13] E. Horqwitz, and R. C. Williamson. "SODOS: a software documentation support environment: its definition." IEEE Transactions on Software Engineering, Vol. SE-12, pp. 849-859, 1986.
[14] R. Baeza-Yates, and B. Ribeiro-Neto, *Modern Information Retrieval: The Concepts and Technology behind Search*, ACM press, pp. 104-132, 2011.
[15] 진광희, 송상민, 이병정, 이정원, "소프트웨어 R&D 프로젝트의 상시 모니터링을 위한 테스트 계획 및 보고." 한국정보과학회 학술발표논문집, pp. 597-599, 2015.

추상 도달가능성 그래프 기반 소프트웨어 모델체킹에서의 탐색전략 고려방법

이낙원 ○ , 백종문

한국과학기술원

skrdnjs@kaist.ac.kr, jbaik@kaist.ac.kr

Controlling Traversal Strategy of Abstract Reachability Graph based Software Model Checking

Nakwon Lee ○ , Jongmoon Baik

Korea Advanced Institute of Science and Technology

요 약

본 연구에서는 추상 도달가능성 그래프 (ARG) 기반의 소프트웨어 모델체킹에서 그래프 탐색전략을 설정할 수 있는 새로운 방법을 제시한다. 본 연구에서는 기존 연구에서 제시된 효과적인 인코딩을 위한 탐색 기법을 사용하여 블록 인코딩 (Block Encoding)을 효과적으로 수행하는 동시에 인코딩된 후의 ARG에 대한 탐색 순서를 고려할 수 있는 이중 탐색전략 기법을 제시한다. 또한 인코딩된 ARG에서 탐색 순서의 변화가 모델체킹의 성능에 미치는 영향을 확인하기 위하여 제시하는 기법을 오픈소스 모델체킹 도구에 구현하고 벤치마크 실험을 수행하였다. 실험결과 블록 인코딩된 ARG에 대한 탐색전략이 달라지면 모델체킹의 성능이 달라지는 현상을 확인하였다.

1. 서 론

추상 도달가능성 그래프 (Abstract Reachability Graph: ARG) 기반의 소프트웨어 모델체킹 방법은 소프트웨어가 여러 실행 경로를 가지고 있는 것에 착안하여 복잡한 전체 소프트웨어를 각각의 실행 경로로 분리하고 단순해진 각각의 경로에 대하여 모델체킹을 수행하여 확장성을 향상시킨 기법이다.

ARG 기반 모델체킹 방식은 개별 경로에 대한 검증을 쉽게 수행할 수 있으나 경로의 수가 매우 많아지면 전체 경로를 모두 검증하는데 오랜 시간이 소모되는 문제가 있었다. 최근에는 모델체킹 성능의 향상으로 기존의 개별 경로에 대한 검증보다 더 큰 규모의 검증을 수행할 수 있게 되면서 경로를 몇 개씩 묶어 한번에 검증하는 블록 인코딩 (Block Encoding) 방식이 해결책으로 제시되었으며 성능의 향상을 보였다. 그런데 블록 인코딩 적용으로 인하여 인코딩된 ARG (eARG) 가 실질적인 모델이 되었음에도 불구하고 eARG의 특성을 반영한 탐색전략에 대한 고려가 이루어지지 않았다.

본 연구에서는 블록 인코딩을 사용한 ARG 기반의 모델체킹에서 인코딩된 eARG의 특성을 반영한 탐색전략을 적용할 수 있는 이중 탐색전략 기법을 제안한다. 제안하는 이중 탐색전략 기법은 블록 인코딩 과정에서 사용하는 지역 탐색전략과 인코딩된 eARG의 특성을 반영하는 전역 탐색전략의 두 가지 탐색전략을 가지고 있어 효과적인 블록 인코딩과 eARG 기반의 탐색전략을 동시에 지원할 수 있다. 또한 제안하는 이중 탐색전략 기법은 기존의 단일 탐색전략 구현방식과 유사하게 구현 가능하여 추가적인 계산의 오버헤드 없이 사용할 수 있다.

제안한 이중 탐색전략 기법은 실제 오픈소스 모델체킹 도구에 구현하였다. 구현된 도구를 이용하여 기존 단일 탐색전략 기법과 이중 탐색전략 기법의 비교 실험을 수행하였으며 eARG를 고려한 탐색전략 기법의 변화에 따른 모델체킹 성능의 변화 또한 실험을 통해 확인하였다.

2. 문제정의

ARG 기반의 모델체킹은 ARG를 만들어나가는 과정에서 프로그램에 대한 검증이 수행되므로 프로그램의 CFG를 어떻게 탐색할 것인지 결정하는 탐색전략이 전체 모델체킹의 성능에 영향을 미친다. 기존의 ARG 기반의 모델체킹 기법들은 사용한 탐색전략을 구체적으로 명시하고 있지 않아 탐색전략 선택에 따른 모델체킹 성능의 변화를 고민하지 않고 있거나, 특정 탐색전략만을 제시하고 탐색전략에 대한 추가적인 고민이 필요하다는 언급을 하는 정도에 그쳤다. 특히 블록 인코딩을 사용한 연구에서는 여러 연구에 걸쳐 다양한 인코딩 방식과 성능향상 효과등을 보여주고 있으나 탐색전략의 차이를 고민한 연구는 없었으며 탐색전략의 선택에 대하여 블록 인코딩과정의 효율성 즉, 두 관리노드 사이의 일반노드들을 효과적으로 탐색하기위한 전략을 선택하는 방식을 택하고 있다. 이 연구에서는 CFG 기반의 탐색전략인 Reverse Post Order (RPO) 기반의 탐색전략을 선택하고 있으나 이는 eARG의 특성을 고려한 탐색전략이 될 수

없다.

3. 이중 탐색전략 기법

새롭게 제안하는 기법에서는 노드를 담고있는 큐에 회색으로 표현된 추상 수식 관리노드와 일반노드가 함께 들어있을 때는 일반노드중에서만 노드를 선택하여 처리 과정으로 전달하고 전체 큐가 추상 수식 관리노드로 채워져 있을 경우에만 추상 수식 관리노드간의 우선순위를 고려한 탐색 전략으로 관리노드를 선택한다. 일반노드를 선택하기 위한 전략은 지역전략이라 부르고 추상 수식 관리노드를 선택하기 위한 전략을 전역전략이라 부르기로 한다.

이렇게 지역, 전역 탐색전략을 분리하는 방법을 사용하면 큐 안에 들어있는 모든 일반노드는 가장 최근에 선택된 추상 수식 관리노드로부터 도달 가능한 노드들만 존재하게 되며 서로 다른 추상 수식 관리노드로부터 도달가능한 일반노드는 동시에 존재할 수 없다. 결국 지역전략은 블록 인코딩을 효과적으로 수행할 수 있는 기존 탐색전략을 사용하고 전역전략에 대하여 여러가지 탐색전략을 적용함으로서 eARG에 대한 진정한 탐색전략의 고려가 가능하다.

4. 탐색전략 비교실험

본 연구에서 제안하는 eARG 기반 탐색전략 고려방안을 활용하여 서로 다른 탐색전략이 모델체킹의 성능에 미치는 영향을 실험을 통하여 분석하였다. 먼저 제안하는 기법이 기존의 탐색전략 미 고려방안에 비하여 성능의 차이가 없다는 것을 확인하기 위해 기존기법과 거의 동일한 전략으로 탐색이 진행되도록 전역 및 지역전략을 설정한 후 기존 기법과 성능 비교를 수행하였다.

전역 탐색 전략은 널리 알려진 그래프 탐색 전략인 DFS와 BFS 그리고 CSRPO 세가지 방법을 사용하였다. 전역 탐색전략 3종, 알고리즘 4종, 블록 인코딩 방식 2종의 총 24가지의 모델체킹 설정에 대하여 비교 실험을 수행하였다.

제안하는 기법은 오픈소스 모델체킹 도구인 CPAchecker에 구현하였다. 실험에 사용된 대상 프로그램은 Software verification competition (sv-comp) 에서 사용하는 Integer control flow 카테고리의 46개 벤치마크 프로그램을 사용하였다. 모델체킹의 성능 비교는 모델체킹을 수행하는데 소요된 cpu 시간을 비교하였다. 각 벤치마크 프로그램에 대한 모델체킹 시간은 900초의 시간제한으로 시행되었으며 3.3GHz 의 프로세서에서 8000MB 메모리 제한을 두고 수행되었다.

실험 결과 제안하는 이중 탐색전략 기법이 충분히 사용 가능한 수준의 기법이라는 것을 기존 기법과 비교를 통해 확인할 수 있었으며, 탐색전략간의 비교 결과 반복문 시작점 만을 관리노드로 사용한 경우보다 함수의 시작과 끝을 같이 관리노드로 사용한 경우에 성능의 변화가 많이 나타났으며 이는 관리노드의 수가 많고 경로의 수가 많아지기 때문에 탐색전략의 영향이 커지는 것으로 생각할 수 있다. 알고리즘을 기준으로 살펴보았을 때 PA 기법에서 성능의 변화가 큰 폭으로 나타났는데 PA 기법은 sv-comp와 같은 모델체킹 경진대회에서 높은 성능을 보여주는 최신의 기법으로 이러한 최신 기법에서 탐색전략의 변화만으로 성능의 차이를 발생시킨다는 것은 탐색전략이 모델체킹 성능 향상을 위한 한 방안으로 연구가치가 있다는 것을 의미한다. 또 다른 알고리즘은 IMPACT에 대해서는 기존에 알려진 바와 같이 FC의 효율적인 사용이 절대적으로 성능향상에 필요한데 FC를 사용하지 않는 경우에 탐색전략으로 대폭 성능의 향상을 보이는 경우가 있어 여러 알고리즘과 탐색전략의 관계에 대한 연구가 필요함을 알 수 있다. 벤치마크 프로그램의 기준에서 살펴보면 전반적으로 성능의 차이가 특정 프로그램에서 자주 발생하고 있으며 결국 프로그램의 특성에 따라 탐색 전략의 영향이 존재한다고 볼 수 있다.

각 실험은 모든 조건이 동일한 상태에서 전역 탐색전략만을 다르게 하여 진행 되었으므로 탐색 전략의 차이가 일부 프로그램, 알고리즘 하에서 상당한 성능의 차이를 유발한다는 것을 확인할 수 있다.

5. 결론 및 향후연구

본 연구에서는 ARG 기반의 모델체킹에서 블록 인코딩을 사용하는 경우에 eARG의 특성을 탐색전략에서 고려하기 위하여 이중 탐색전략 기법을 제안하였다. 이중 탐색전략 기법은 ARG에서 블록 인코딩될 일반노드들의 우선순위를 낮추고 eARG의 노드가 될 관리노드들의 우선순위를 높여 일반노드에 대한 탐색전략과 관리노드에 대한 탐색전략을 다르게 적용하는 방식으로 eARG의 특성을 탐색전략에 반영하였다. 일반노드에 대한 탐색전략을 지역전략, 관리노드에 대한 전략을 전역전략으로 사용함으로서 기존 연구들을 통해 블록 인코딩에 효과적으로 알려진 탐색 전략을 지역적략으로 사용하고 전역전략의 변경을 통해 eARG를 탐색하는 순서를 변경할 수 있게 되었다.

제안한 이중 탐색전략 기법을 오픈소스 모델체킹 도구인 CPAchecker에 구현하였으며 이중 탐색전략을 사용하지 않은 경우와 이중 탐색전략을 사용한 경우를 비교하여 이중 탐색전략의 효과를 확인하였으며 세 가지 전역전략을 여덟가지의 모델체킹 설정에 적용하여 벤치마크 실험을 통해 전역 탐색전략의 변화에 따른 모델체킹 성능의 변화를 확인하였다.

A Unified Approach for UML Based Safety Oriented Level Crossing Using FTA and Model Checking

Anit Thapaliya and Gihwon Kwon

Dept of Computer Science, Kyonggi University

anitdgret@gmail.com, khkwon@kgu.ac.kr

Abstract

This paper proposes a unified approach for UML based safety oriented railway level crossing using model checking and fault tree analysis. The main goal of this research is to show the possibility to combine the concept of traditional safety analysis technique FTA and formal verification technique model checking for UML based safety oriented railway level crossing system. In this paper we used UML for specifying, visualizing and constructing the application of level crossing with numbers of diagram in which some of the diagrams are used for modelling the finite state machine from the UML diagrams to the support model checking. Secondly, we considered traditional safety analysis technique FTA in order to generate safety properties within the level crossing system. Similarly, after the transformation of UML models into unified finite state machine and FTA to safety properties NuSMV model checker was used in order to satisfy our model as per the safety properties.

1. Introduction

The traditional software engineering methodologies commonly emphasized in designing and developing systems just to meet the certain functional requirements. But those methodologies barely focused on examining the possible consequences failure which could lead to the massive loss of life and properties in the critical software domain. Software's are massively used as monitoring and controlling components of real time system like railway level crossing systems. The problem is failure in those systems can cause direct loss of life and property hence there is the need of safety oriented level crossing which could minimize the potential harm to the human and environment. Though any software system is not completely safe, there is a need of better approach for developing safety critical system to encompass the number of safety challenge in the problem domain currently.

The problem domain chosen for this research was a level crossing system having a road and track intersection based on which we considered traditional safety analysis technique fault tree in order to generate the safety requirement model and the safety-properties [1]. Similarly, using the case study of level crossing we generated different diagrams based on the UML principles. Some of the diagram was then translated into unified finite sate machine in order to support the formal methods. Normally UML diagrams lacks of mathematical foundation and our NuSMV model checker only support the system model in terms of finite state machine. Lastly we proposed our model checking based verification of the level crossing system with models form UML diagrams and properties form fault tree analysis.

This paper is structured as mentioned further. We have first described the motivational factors behind the paper including the introduction of railway level crossing, UML, FTA in (section 2). In section 2.2, we described the traditional safety analysis technique fault tree for level crossing and generated safety requirement model and safety properties of level crossing. In section 2.3, level crossing system is described using UML based diagrams. Likewise, unified finite state machine is introduced in section 2.4 from the UML state transition diagrams in section 2.3. Similarly, in section 3 we combined safety properties from section 2.2 and system model from section 2.4 in order to introduce NuSMV model checking for verification of the level crossing system model.

"This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2016-R0992-16-1014) supervised by the IITP(Institute for Information & communications Technology Promotion)"

2. Background

Evidences from historical disasters in the field of

software engineering triggered the need of safety related considerations while designing and developing the critical software systems that could cause the undesired loos of life and property. As railways and roadways are among the safest ways to travel, this paper mandates the requirements for the design, construction, inspection, operation of railway level crossing. Level crossing constitutes of different level of safety considerations internationally but still, each year, around 400 people lose their life in European Union and over 300 people get physically injured in United States [2]. Hence, designing and developing critical system like level crossing needs specific safety requirement model in order to avoid train and vehicles collision for principal safety related objectives. This paper presents general guidelines for the management and operation of safe level crossing along with the extensive consultation operation of level crossing mechanism in the real world scenario in order to present the quantitative understanding of risks and responsibilities.

2.1 Level Crossing Case Study

A level crossing or grade crossing is an intersection point between a railway line and a road or a path at the same level, as opposed to the line crossing over or under using a bridge or tunnel [3]. In other words, level crossing is a term used for describing a section where railway track crosses the roadway. This case study is focused on the principal safety objectives of avoiding train and vehicles collision with some basic considerations like train do not stop at the level crossing area but it is mandatory for vehicles to clear the track before the train. To stop moving train in the level crossing is difficult because of braking capabilities of larger mass of the train. Secondly, for accident prevention and safety another requirement of the system is to close the gate while the train is in the level crossing area. Among the different types of level crossing accidents in the world, like train – vehicles collision, train – pedestrian's collision, our case study only focuses on the train and vehicles collision at the level crossing. This case study is concerned with a sensor based crossing controlling system used in the intersection area between the track and the road as illustrated in figure 1. The crossing consists of two sensors, two gates and one controller which control the overall working mechanism of the level crossing system. Sensor is used to detect the train in the level crossing area and the gates are used as the barriers for the roadway to block the entry of vehicles in the track while the train is passing. Similarly, the lights, red and yellow, are for the roadway traffic indicating track busy (red) and track free (yellow). This case study is the abstraction of railway management system where the communication channel and emergency situation are simply ignored for the sake of simplicity of the problem domain. In the next section we have designed our system as per the case study and principal safety objective of this paper.

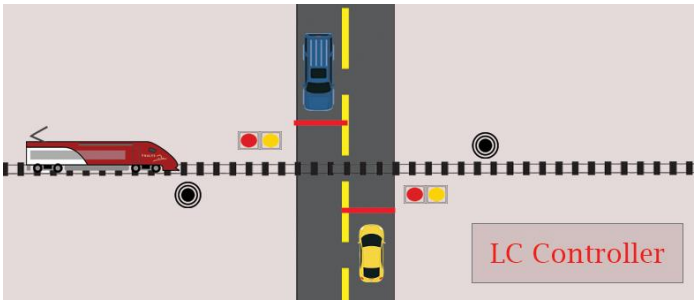


Fig.1 Railway Level Crossing

2.2 Fault Tree Analysis Safety Requirement Model

The importance of fault tree in developing the safety critical system like level crossing is to generate the basic cut sets of event that could cause a top level of failure. Fault tree analysis is the top down approach that provides a graphical representation of deductive failure using the Boolean logical gates representing how systems can fail. The connections between the basic events making the top event failure are carried out by using the logical OR-Gate and AND-gate. Based on our principle safety objectives to avoid collision we defined Train and Car collision as our top level event and generate the fault tree describing all the causes of undesirable events. The principal safety objective is to avoid vehicles passing through the crossing at the same time when train is passing. This objective is defined by the following requirements. (i) Gate must be closed when the train is crossing, (ii) cars or vehicles have enough time to pass the crossing when the gate is open, (iii) train do not stop at the level crossing. The main causes of the failure are more dependent on cars intention and more on the controller malfunctioning and sensors malfunctioning. The main causes of the failure are more dependent on cars intention and more on the controller malfunctioning and sensors malfunctioning. These events are illustrated in the fault tree analysis below but this case study only focuses on limited numbers of failure that could directly lead to collision.

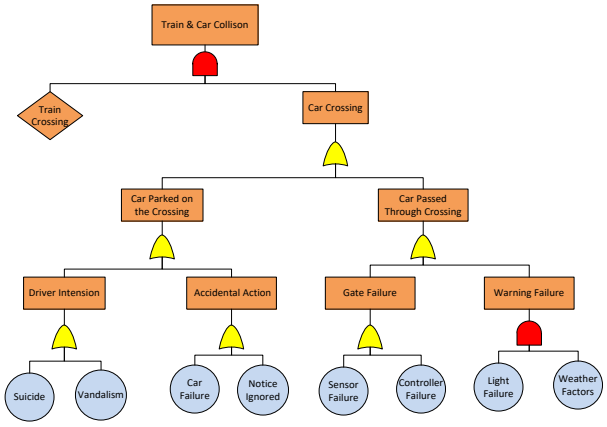


Fig.2 FTA of Level Crossing

The above Figure 2 represents the fault tree analysis of railway level crossing based on the principal safety objective of case study. Once the faults have been derived from the FTA at figure 2, it is possible to form a formal property specifying the controller behaviour for avoiding those faults. The process of generating safety properties from fault tree analysis is inspired by [1] where a safety formal property is extracted by using computational tree logic using the static gates. The main goal of this paper is to verify the level crossing model using model checking through a safe automation model of the system to check whether the properties satisfies the system model. Top level fault of a static gate is the combination of inputs and the general form of safety properties can be deduced by using the path quantifier A, a property that holds for all the paths, and state quantifier G, a property holds for all the states within the path. Figure 3 shows the combined expression from the fault tree analysis. According to [4] AG ¬ (Car Crossing & Train Crossing) can be depicted and used for checking the invariant safety properties that requires only present values of the controller.

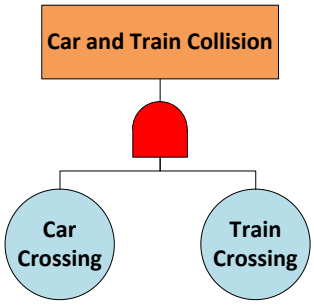


Fig.3 Safety Formal Properties

2.3 UML Modeling For Level Crossing

The Objective of this section is to model the proposed system. The lists of requirements that were gathered in the previous chapter are to be designed in this section using UML. Different UML diagrams are used to define the requirements. UML stands for Unified Modelling Language which is used in OOAD approach in software engineering. UML 2.2 contains fourteen different diagram divided into two different categories, structural and behavioural, of the system. Few of those diagrams which are used in this research are mentioned below:

.

2.3.1 System Architecture Design

The architectural design of the system is based on the real time working mechanism of the level crossing. Normally, the inbound sensor detects the train approaching and leaving motion from the level crossing and communicates with the controller in the rare end. The rare end controller receives the signals from the sensors and operates the gate in order to block or open the roadway movement respectively. The controller also operates the lights as per sensor data. Figure 4 describes the architectural design of the system based on the components within level crossing.

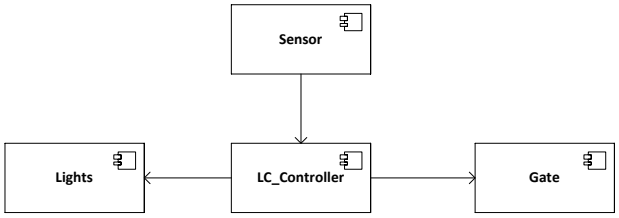


Fig.4 System Architecture of LC

2.3.2 UML Models of Level Crossing

Considering our case study we designed few, out of 14 UML, diagrams which are class diagram, use case diagrams, sequence diagrams, activity and state transition diagrams. A class diagram [5] represents the static building block of the system in object oriented view.

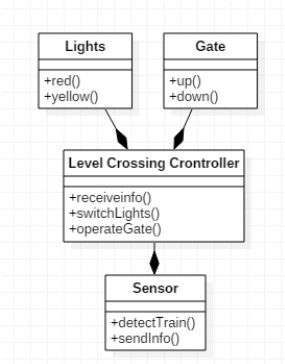


Fig.5 LC Class Diagram

A sequence diagram [6] is used to specify the operational scenarios of the level crossing as per our case study. It represents how objects interact and communicate with other objects. Figure 6 represents the level crossing sequence diagram including the external actors of the system as well the operational flow between the object involved within.

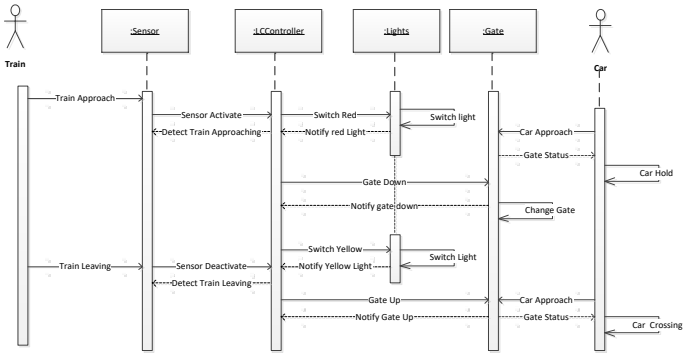


Fig.6 LC Sequence Diagram

Apart from class diagram and sequence diagram, we have designed use case diagrams to represent the interaction of actor to the system which shows the relationship between the components along with the activity diagrams for the better understanding of the dynamic aspects of the system involving the flow from one activity to another. But, to reduce the complexity of the paper, use case diagrams are ignored here. Figure 7 and Figure 8 describe the dynamic nature of level crossing in the case study requirements.

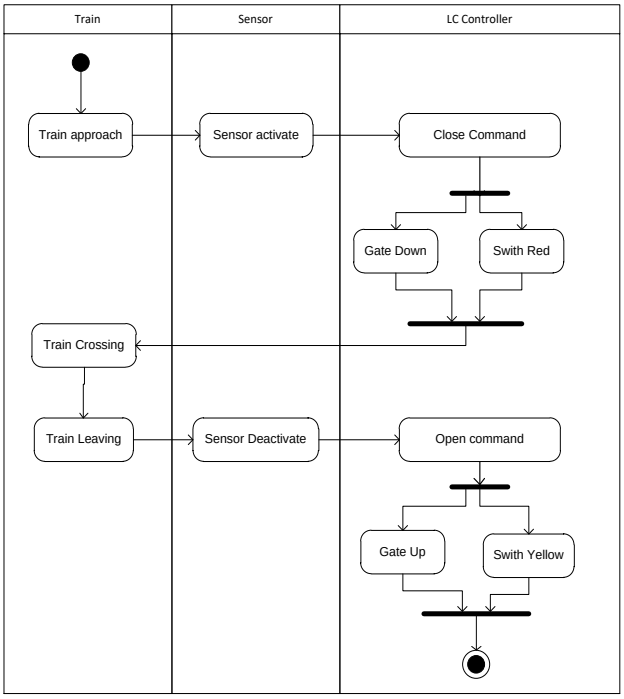


Fig.7 LC Activity Diagram

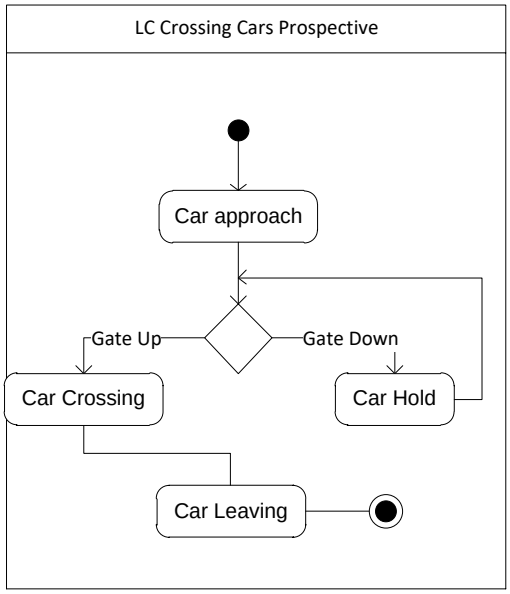


Fig.8 LC Car Activity Diagram

Besides activity diagram we have designed state transition diagram in order to convert UML diagram into finite state machine which is the input model for verifying our UML model through model checking. The next section will describe the transformation of UML based state transition diagram into unified finite state machine.

2.4 Transformation of State Transition Diagrams to Finite State Machine

State transition diagram [7] represents the model of the system including the state and transitions like events and actions. It shows the dynamic behaviour of the states in object oriented system. Likewise, finite state machine gives a computational model for a static as well as dynamic behaviour of system. The transformation of state transition diagrams is fully dependent on three major components which are action, inputs and transition. Hence, state diagram represents the how an object changes it state. This transformation is used in the FSM through the graphical representation. FSM is an abstract machine with the finite number of states from the initial state to the final state.

According to the theory of automata, A finite state machine 'M' is defined as $(Q, \Sigma, \delta\ q_0, F)$

Finite state machine is a mathematical model which denotes the finite states and transitions between states of system [8, 10]. FSM consists of five-element set: $FSM=(Q, \Sigma, \delta\ q_0, F)$, where: Q is a set of all possible states of system; Σ is a set of triggering events that the system may encounter; δ is state

transition functions which determine subsequent state that the system will be transferred to when triggering event occurs. q0 : Q ×Σ→ Q ; $q_0 \in$ Q, where: q0 is initial state; F ⊆ Q is a set of system end states, the system will end its behavior once it is transferred to any end state.From the above mentioned definition of finite state machine state transition diagrams are transformed into unified finite state machine.

In this section we have illustrated our case study UML state transition diagram into FSM. Six different FSM was generated as per the requirements from case study. Figure 9, below, represents the state transition diagram of the car or vehicles in the level crossing scenario. This state transition diagram is transformed into FSM in Figure 10 using the conversion process in Table 1.

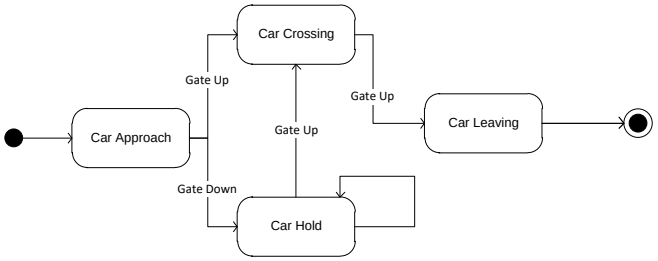


Fig.9 State Transition Diagram of Car

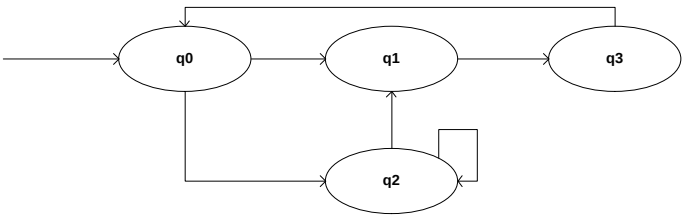


Fig.10 FSM of Car

State / Input	Gate Up	Gate Down
q_0	q_1	q_2
q_1	q_3	–
q_2	q_1	q_2
q_3	q_1	q_1

Table.1 FSM Conversion of Car

In table 1, state transition diagram of car is transformed into FSM. This conversion is inspired by [9] where UML model is validated by the finite state machine. Similarly, the conversion table shows the state diagram's equivalent FSM where q_0 denoted as the initial state as well as the final state. The transition occurred with some certain inputs as described in the Table 1.

The FSM of level crossing controller is directly dependent on the train movement and the sensors. Whenever the train approaches to the level crossing, the sensor detects the train's motion and, thus, communicates the information to the controller. The controller then, in order to close the level crossing gate, sends a close command input. This scenario is depicted in the state transition diagram which is the input for FSM. This paper presents only one finite state machine with table out of six generated during the research. FSM of sensor activities is similar to that of the lights controller and gate because they mainly depend upon the train's motion in the track.

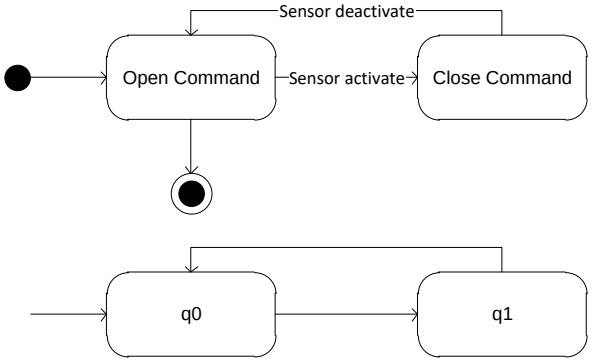


Fig.11 FSM of Controller

Similarly, the state diagrams of light, train and gate were also converted into corresponding finite state machine using the same process. State transition diagrams of light and gate depends upon the state transition diagram of controller. The transition of state in Light and Gate both takes the input from the controller in terms of open command and close command. Open command signifies that the track is free for level crossing whereas close command signifies that a train is either approaching or crossing the track. Figure 12 and 13 shows the state transition diagram and corresponding finite state machine of light and gate respectively.

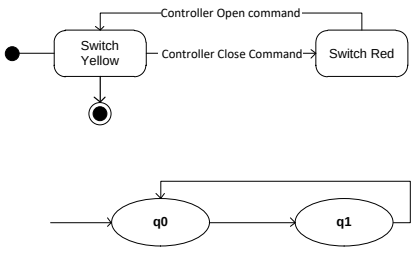


Fig.12 FSM of Light

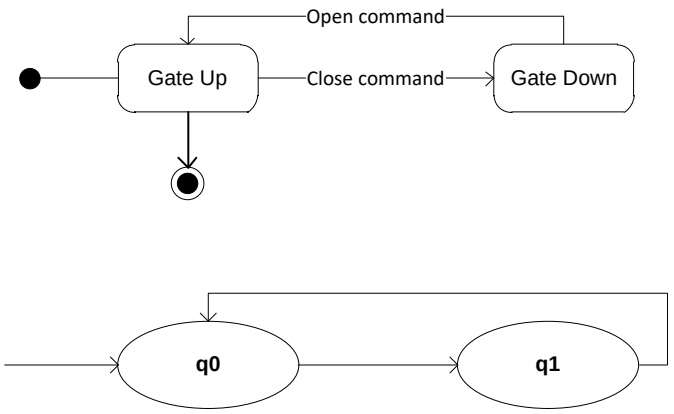


Fig.13 State Transition Diagram of Gate

FSM of train is based on the UML state transition diagram designed from the requirement that train don't stop at the level crossing. Hence, the state transition diagram of the train is composed of the different states of the its movement on the level crossing such as approaching, crossing and leaving the level crossing. Within this consideration we proposed FSM of train directly composing the state transition diagram of train without any input parameters as the train movement is independent. Figure 14 describe the train state transition diagram and resulting FSM.

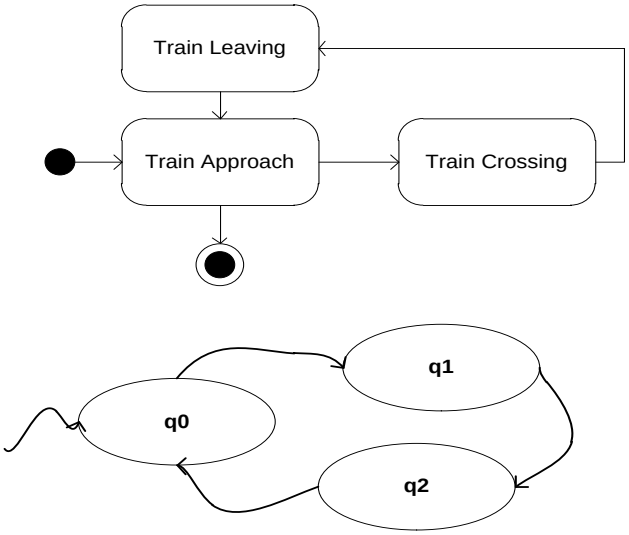


Fig.14 FSM of Train

3. Model Checking

In order to satisfy our principal safety objectives we are aiming at fully automatic tool support for the verification process that can be used in the early stages in the design phase. This research proposes the use of NuSMV model checking tool for verification of the design before implementation. Model checking is a formal method which can prove that a model can work properly within the certain specification. In other words, model checking technology analyzes the system model with the system properties for verification. In this paper, NuSMV [11] model checker is applied for the verification of principal safety objectives generated from traditional fault tree analysis and FSM of railway level crossing obtained from UML state transition diagrams.

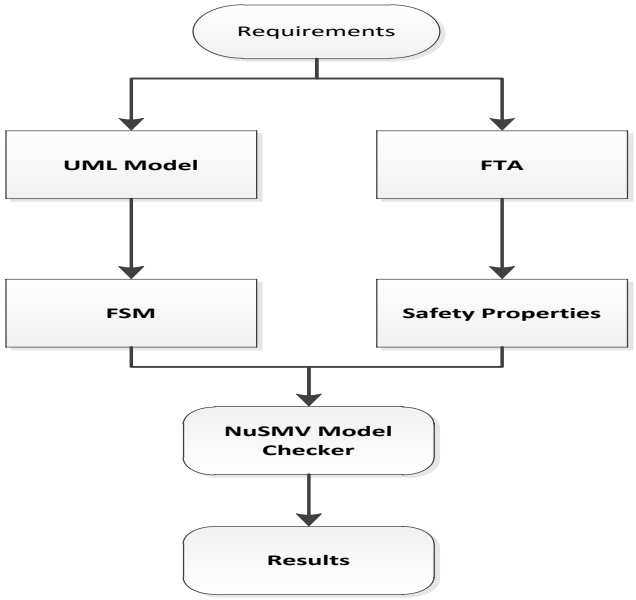


Fig.15 NuSMV Verification

Figure 15 represents the working mechanism of NUSMV verification for the case study. NuSMV is model checking tools for finite state machine against the specification written in the temporal logic such as computational tree logic. NuSMV has its own language description for the finite state machine based on the internal inputs of SMV model checking algorithms. In order to apply NuSMV model checking in this case study our FSM was translated into SMV internal representation. A state transition diagram was regarded as each module, and its corresponding state as the variables with the transition relation as per the finite state machine generated in earlier section [10]. The principle safety objectives generated from fault tree analysis is now regarded as keyword SPEC for SMV program which simply refers the specification for model checking.

After the formal conversion of FSM and safety properties into a language that supports NuSMV. Model checking is simulated and the result obtained is shown in figure below

Fig.16 NuSMV Result

The true result shows that the finite state machine obtained from UML state transition diagrams meets the safety requirements obtained from the traditional fault tree analysis of the railway level crossing case study.

4. Comparison

Formal verification with model checking has been applied before in the field of safety properties verification in critical software domain. Fault forecasting and fault removal for safety critical system can be minimized using the formal properties generated from the fault tree analysis [1]. Dynamic and temporal gates were used to model the safety properties from the FTA. But [1]does not show the effective modelling of system and its specification. Safety requirements are not sufficient for model checking, we need to specify our system model as per our requirements. Hence, this paper used UML principles to design system models. Similarly, [6] applied B method for the verification of UML related safety applications. In [9] the finite state machine was used for validation of UML diagrams. But these papers are only focuses on the verification of UML model and diagram not the formal verification of the UML and FTA.

Our safety related case study describe the effectiveness of using FTA and UML together for the formal verification using NuSMV based model checking. If we test the model for the existence of the basic faults we can determine whether the system application will cause the system level hazard. This is the major advantages of using FTA and UML together for the safety critical systems. FTA on railway level crossing specifies a set of weakest pre conditions for the occurrence of high level hazard i.e. Train and Car Collision. The negation of this condition forms a safety property as a input for model checking. UML is directly related to these concepts because it is indented for the specification of precondition and post condition information. But UML models are not suitable for NuSMV model checking. This approach required the transformation of UML to finite state machine, as the input model for NuSMV. Existing model based verification approaches focused on the UML models verification with respect to its requirements but this paper presents procedural ways to perform safety properties verification on the UML models as per the requirements. In other words, here safety properties verification was done through the corresponding finite state machines of the UML diagrams.

Lastly, this approach is effective for modelling the critical system in terms of requirements traceability because the counterexample in the model checking can be used for identification of faults or error in the resulting state transition diagrams.

5. Conclusion and Future Work

This paper concludes that it is possible to combine the advantages of UML, safety analysis technique and formal methods all together to design a critical safety related application. In this paper, we combined the different approaches of software engineering all together to propose an unfiled approach for UML based safety oriented critical system development using the traditional fault tree analysis and the formal aspects of model checking using NuSMV. According to the research, this unfiled approach is more promising for verification of safety properties extracted from traditional safety analysis FTA against the FSM obtained from the UML state transition diagrams.

Similarly, from this research it can also be concluded that traditional safety analysis can be used for along with UML modelling language for modelling the safety critical system. The graphical representation UML is very useful for converting the diagrams into correct finite state machine for the final verification of UML diagram model using the model checking tool. Lastly, a unified approach to verify UML based safety oriented level crossing system is presented with successful verification result.

Likewise, the future direction of this research can be extended into different decomposition of UML diagrams into finite state machine because this paper only focuses on the transformation of state transition diagram into the finite state machine. In addition to this, transformation of sequence diagrams into finite state machine is another challenge for future. Verification of those diagrams using model checking tools is possible. Model checking counterexamples can detect the inconsistency and abnormal behaviour of the sequence diagrams.

Lastly, traditional safety analysis technique can be improved by implementing more powerful techniques like Failure Mode Effects Analysis [12] along with traditional fault tree analysis. Normally, FMEA is performed in order to generate the top level of failure

for any critical system domain but in this research FMEA is not implemented because level crossing case study is more focused on avoiding the top level event like car and train collision at the level crossing track as the principle safety objectives.

References

[1] Israel Santiago Barragan, Jean-Marc Faure. FROM FAULT TREE ANALYSIS TO MODEL CHECKING OF LOGIC CONTROLLERS. 16th IFAC World Congress, Jul 2005, Praha, Czech Republic. CDROM paper n04596. <hal-00361602>

[2]European Union Agency For Railways, Railway Safety Performnace in the European Union Biennial Report 2016.

[3]Level Crossing. (n.d). retrieved December 28, 2016 from https://en.wikipedia.org/wiki/level_crossing

[4]enry, S. and J.M. Faure (2003). Elaboration of invariant safety properties from fault-tree analysis. Proceedings of IMACS-IEEE "CESA'03, 6 pages, July 9-12, Lille, France.

[5]Laurent, Audibert (2009) UML 2 de l'apprentissage à la pratique (cours et exercices). Editions Ellipses, ISBN 10: 2729852697.

[6]Jean Louis Boulanger, Brail Requirement Analysis, IADIS Virtual Multi Conference on Computer Science and Information Systems 2005.

[7] Blanc, Xavier & Isabelle, Mounier (28 Septembre 2006) UML 2 pour les développeurs cours avec exercices corrigés, Eyrolles ISBN : 2-212-12029-X

[8]Chenthati, D., et al. Verification of Web Services Modeled as Finite State Machines. in Mathematical/Analytical Modelling and Computer Simulation (AMS), 2010 Fourth Asia International Conference on. 2010.

[9]Vipin Saxena, Santosh Kumar, Validation of UML Class Model though Finite State Machine. International Journal of Computer Application (0975-8887) Volume 41-No. 19, March 2012.

[10]Congcong Chen, Fuping Zeng, Minyan Lu. Verification Method for Software Safety Requirement by Combining Model Checking and FTA, 2015 International Industrial Informatics and Computer Engineering Conference (IIICEC 2015).

[11]NuSMV 2.5 User Manual.

[12]Ben Swarup Medikonda, Anu A Gokhale and Seetha Ramaiah, FMEA and Fault Tree based Software Safety Analysis of a Railway Crossing Critical System, Global Journal of Computer Science and Technology, May 2011

감정 분석 기반의 사용자 피드백을 이용한 클라우드 서비스 평가 기법

윤동규 ○, 김웅수*, 박준석**, 염근혁*+
부산대학교 전기전자컴퓨터공학과, 부산대학교 물류혁신네트워킹연구소
{lodestar692, kus9010, pjs50, yeom}@pusan.ac.kr

Cloud Service Evaluation Techniques Using User Feedback based on Sentiment Analysis

Donggyu Yun○, Ungsoo Kim, Joonseok Park, Keunhyuk Yeom*+
Department of Electrical and Computer Engineering, Pusan National University
Research Institute of Logistics Innovation and Networking, Pusan National University

요 약

본 논문에서는 클라우드 서비스 브로커 (Cloud Service Broker, CSB) 에서 클라우드 서비스에 대한 사용자 피드백에 감정 분석 (Sentiment Analysis) 을 적용하여 서비스를 평가하는 방법을 제안한다. 기존의 평점 기반 서비스 평가 방법의 경우 사용자마다 평점을 매기는 기준이 다양하여 신뢰도가 떨어진다는 문제점이 존재한다. 본 논문에서 제시하는 방법에서는 평점뿐 아니라 기계 학습 기반의 감정 분석 결과를 이용한 보정 점수를 사용하여 기존의 평점 기반 서비스 평가의 단점을 보완한다. 또한 본 논문에서는 실제 클라우드 서비스 리뷰를 학습 데이터로 사용한 실험을 통해 감정 분석에 사용될 수 있는 다양한 기계 학습 알고리즘의 성능을 비교한 결과를 제시한다.

1. 서론

클라우드 서비스 브로커 (Cloud Service Broker, CSB) 는 클라우드 서비스 제공자와 사용자 사이에서 클라우드 서비스를 추천해 주며 계약을 중개해 주는 개념 또는 그러한 서비스를 지칭한다[1]. CSB에서 핵심적으로 고려되어야 하는 사항은 사용자의 기대를 충족시켜줄 수 있는 적합한 클라우드 서비스를 추천해 주는 것이며, CSB는 클라우드 서비스 추천을 위한 다양한 서비스 평가 메커니즘을 가질 수 있다.

본 논문에서는 기존의 대표적인 피드백 기반 서비스 평가 방법인 평점 기반 서비스 평가의 단점을 보완하기 위해 감정 분석 (또는 감성 분석, Sentiment Analysis) 을 적용하여 클라우드 서비스를 평가하는 기법을 제시한다. 즉 사용자가 작성한 리뷰에서 서비스에 대한 사용자의 감정을 추출하여 이를 기반으로 평점을 보정한다. 그리고 실제 클라우드 서비스 리뷰에 대한 감정 분석 실험을 수행하여 다양한 기계 학습 알고리즘의 성능을 비교한 결과를 제시한다.

2. 감정 분석 기반의 사용자 피드백 평가

2.1 기존의 평점 기반 서비스 평가 방법의 문제점

사용자로부터 서비스에 대한 피드백을 받는 가장 일반적인 방법으로는 평점을 입력 받는 방법이 있다. 가장 간단하게 사용자의 피드백을 평가에 반영할 수 있는 방법 중 하나로 쇼핑몰, 앱 스토어의 평가 시스템 등 다양한 분야에서 사용되고 있다. 그러나 이와 같은 평점 기반 평가 시스템은 아래와 같은 문제점을 가지고 있다.

- 5단계, 10단계의 세부적인 구분이 크게 의미가 없는 경우가 많다. 예를 들어 5점 만점의 평가인 경우 대다수의 사용자들이 서비스가 만족스러웠을 경우 5점, 불만족스러웠을 경우 1점을 매긴다[2].
- 사용자마다 평가 기준이 다르다. 평균적인 수준의 서비스에도 5점을 주는 사용자도 있는 반면 좋은 서비스에도 4점을 주는 엄격한 사용자도 존재한다.

본 논문에서는 이와 같은 문제를 해결하기 위한 방안으로 서비스 리뷰에 감정 분석 기법을 적용하여 서비스에 대한 사용자의 감정에 따라 평점에 보정을 가하는 방법을 제안한다.

2.2 감정 분석을 적용한 사용자 피드백 평가 기법

감정 분석이란 대상 (주로 텍스트) 으로부터 대상에 담겨있는 감정을 추출해내는 방법을 말하며, 일반적으로 데이터 수집-텍스트 전처리-모델링-검증 및 평가의 과정을 거쳐 이루어진다. 이 중 모델링 과정에서는 크게 기계 학습 기반의 방법과 사전 기반의 방법이 이용될 수 있는데[3], 본 논문에서는 기계 학습 기반의 방법을 이용한다. 기계 학습 기반의 방법은 사전에 없는 단어나 특정 도메인에서 자주 쓰이는 단어 또한 분석에 반영할 수 있다는 장점이 있어 많이 이용되고 있는 추세이다.

+교신 저자

"이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. NRF-2016R1D1A1B03935865)"

표 1 감정 분석 실험 결과 (F1 Score)

Feature / Algorithm	Unigram+TO	Unigram+ BTO	Unigram+ TF-IDF	Bigram+TO	Bigram+BTO	Bigram+ TF-IDF
Decision Tree	0.9818	0.9814	0.98	0.9812	0.9808	0.9812
Random Forest	0.9832	0.9826	0.9818	0.9832	0.9822	0.9828
Boosted Tree	0.9834	0.9834	0.9826	**0.9844**	0.9836	0.983
Logistic Regression	0.9784	0.9792	0.9796	0.9782	0.9778	0.9782
Support Vector Machine	0.9776	0.9804	0.9766	0.978	0.978	0.9758

2.1절에서 언급한 문제점을 해결하기 위해 이와 같은 감정 분석 방법을 이용한다. **그림 1**의 (a) 와 같이 평점이 3점으로 같은 두 리뷰에 감정 분석 모델을 적용했을 때 하나의 리뷰 텍스트는 '긍정적', 다른 하나의 리뷰 텍스트는 '부정적' 으로 분류되었다면, CSB 내부적으로는 같은 3점이 아니라 각각 3.5점, 2.5점이나 4점, 2점으로 처리할 수 있다. 또한 학습 알고리즘으로 확률 알고리즘을 이용한다면 **그림 1**의 (b) 와 같이 확률 값을 이용하여 보다 세부적인 보정 값을 부여하는 방법도 사용할 수 있다.

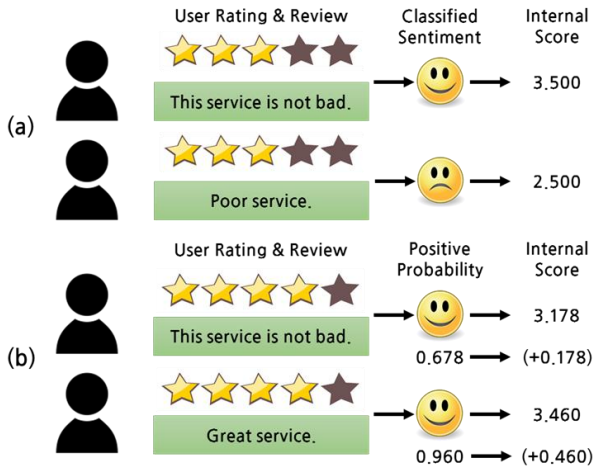


그림 1 감정 분석 결과를 적용한 평점 보정

3. 클라우드 서비스 리뷰에 대한 감정 분석 실험

실제 클라우드 서비스에 대한 리뷰를 대상으로 다양한 기계 학습 알고리즘에 대한 감정 분석 실험을 진행하였다. 실험에 사용된 리뷰 데이터는 Blue Page Dataset[4]으로, SaaS (Software as a Service) 타입의 클라우드 서비스에 대한 약 6,200개의 리뷰로 이루어져 있으며 각 리뷰는 텍스트와 5점 만점의 평점으로 구성되어 있다. 4~5점짜리 리뷰에는 '긍정적', 1~2점짜리 리뷰에는 '부정적' 레이블을 붙였으며 3점짜리 리뷰는 학습에 사용하지 않았다.

감정 분석 기법으로는 기계 학습 기반의 방법을 이용하였으며, Python 라이브러리인 NLTK와 Graphlab Create를 이용하여 텍스트 전처리와 모델링 과정을 수행하였다. 기계 학습 알고리즘으로는 지도 학습 기반 분류 알고리즘인 Decision Tree, Random Forest, Boosted Tree, Logistic Regression, Support Vector Machine을 사용하였다.

또한 학습에 쓰일 Feature로는 Term Occurrence, Binary Term Occurrence, TF-IDF를 Unigram, Bigram에 대하여 각각 계산하여 사용하였다. Unigram은 단어 하나를 단위로 출현 횟수를 세는 방법, Bigram은 단어 두 개를 단위로 세는 방법을 말한다.

성능 평가를 위하여 5-Fold Cross Validation을 통한 테스트를 진행하였으며 평가 척도로 분류 알고리즘의 평가에 많이 사용되는 F1 Score를 이용하였다.

감정 분석 실험을 진행한 결과는 위의 **표 1**과 같다. Boosted Tree 알고리즘에 Bigram 단위로 계산한 Term Occurrence를 Feature로 사용했을 때 가장 좋은 결과를 보였으며 그 외에도 전반적으로 Decision Tree 계열의 앙상블 알고리즘인 Random Forest, Boosted Tree가 뛰어난 성능을 보여주었다.

4. 결론

본 논문에서는 CSB에서 클라우드 서비스를 평가할 수 있는 방법 중 하나로 감정 분석 기반의 사용자 피드백을 이용한 서비스 평가 방법을 제안하였다. 기존의 평점 기반 사용자 피드백의 단점을 보완하기 위해 감정 분석 결과를 이용한 보정 방법을 제시하였으며, 다양한 기계 학습 알고리즘을 적용하여 실제 클라우드 서비스 리뷰에 대한 감정 분석을 수행한 실험 결과를 제시하였다.

5. 참고 문헌

[1] NIST Cloud Computing Standards Roadmap, https://www.nist.gov/sites/default/files/documents/itl/cloud/NIST_SP-500-291_Version-2_2013_June18_FINAL.pdf

[2] https://youtube.googleblog.com/2009/09/five-stars-dominate-ratings.html

[3] Walaa Medhat, Ahmed Hassan and Hoda Korashy, "Sentiment analysis algorithms and applications: A survey", Ain Shams Engineering Journal, Vol. 5, No. 4, pp. 1093-1113, 2014.

[4] Asma Musabah Alkalbani, Ahmed Mohamed Ghamry, Farookh Khadeer Hussain and Omar Khadeer Hussain, "Blue Pages: Software as a Service Data Set", IEEE International Conference on Broadband and Wireless Computing, Communication and Applications, pp. 269-274, 2015.

매쉬업 개발자를 위한 매쉬업 유사도 기반 서비스 추천 방법

김현승 ○, 고인영
KAIST 전산학부
a030603@kaist.ac.kr, iko@kaist.ac.kr

요 약

본 연구에서는 매쉬업 개발자의 성향을 분석하여 서비스를 추천할 수 있는 새로운 방법을 소개한다. 매쉬업 개발자들이 만든 매쉬업의 집합들에 대하여 추천 대상 개발자의 매쉬업 집합에 관한 유사도를 측정하고 유사한 매쉬업 집합들로부터 서비스를 추천할 수 있는 구체적인 방법을 설명한다. 그리고 ProgrammableWeb에서 가져온 매쉬업 데이터로 실험한 결과를 비교 분석하여 본 연구의 방법이 사용자 기반 협업 필터링 알고리즘보다 높은 정확도와 재현율을 보임을 확인한다.

1. 서 론

웹 서비스는 사용자의 로컬 컴퓨팅 리소스의 소비 없이 웹을 통해 서버의 컴퓨팅 자원을 사용하여 사용자가 원하는 기능을 제공한다[1]. 사용자는 필요한 기능을 직접 구현할 필요 없이 서비스를 검색하여 적절한 서비스를 선택, 접근하여 이용할 수 있다. 서비스의 이런 편리함과 유용성 때문에 웹 서비스는 차세대 웹 기술로 많은 인기를 누려왔다.

서비스의 종류와 수가 많아짐에 따라 사용자는 자신에게 필요한 서비스를 찾는 것이 어렵게 되었다. 이에 따라 서비스 추천(Service Recommendation)에 관한 연구의 필요성이 증대되었다[2]. 최근에는 서비스와 *서비스 매쉬업(Service Mashup, 단순한 표현으로 '매쉬업' 또는 'Mashup')*의 관계를 이용하여 서비스를 추천하는 방법이 연구되고 있다. 하지만 기존의 매쉬업 단위 서비스 추천 방식은 개별 매쉬업과 비슷한 매쉬업을 찾아서 그에 따라 서비스를 추천할 수는 있으나, 여러 서비스 매쉬업을 만드는 개발자의 성향을 분석하고 그에 맞는 서비스를 개발자에게 추천하지는 못하였다.

본 논문에서는 매쉬업 개발자를 위한 새로운 서비스 추천 방법을 소개한다. 이 방법은 서비스 추천 대상 개발자의 성향을 분석하고 비슷한 성향의 다른 개발자가 사용한 서비스를 추천하는 것이다. 이 방법으로 서비스를 추천 받은 개발자는 서비스 검색의 복잡성 문제를 피하여 개발 과정의 비용과 시간을 절감하고, 지금까지 사용한 적이 없는 새로운 서비스를 발견하고 활용할 수 있는 기회를 갖게 되며, 이를 통해 더 나은 서비스 매쉬업을 개발할 수 있을 것이다.

본 논문에서 제안하는 매쉬업 유사도 기반의 서비스 추천 방법을 우리는 *Recommender system for Mashup Developers (RMD)*라고 부른다.

2. RMD 서비스 추천 방법

이 추천 알고리즘의 가장 기본이 되는 첫 단계는 서비스 유사도 측정 단계이다. 이 단계에서는 사용자들이 개발한 매쉬업들에 사용된 서비스들 간의 유사도를 측정한다. 본 연구에서는 서비스 평가 점수 행렬(Rating Matrix)에서 두 서비스에 대한 평가 점수의 분포가 유사하면 두 서비스가 비슷한 기능을 제공할 가능성이 높을 것으로 가정한다. 서비스 점수 분포의 유사도를 구하는 방법으로는 코사인 유사도(Cosine Similarity)를 사용하였다. 본 연구에서 서비스는 그 서비스에 대한 사용자들의 평점 벡터로 정의한다. 두 서비스 s_a와 s_b의 코사인 유사도를 $Cosine(s_a, s_b)$라 할 때 서비스 유사도를 다음과 같이 측정할 수 있다.

$$SimS(s_a, s_b) = \frac{1 + Cosine(s_a, s_b)}{2} \quad (1)$$

두번째 단계인 서비스 매쉬업 유사도 측정 단계에서는 서비스 매쉬업 간의 유사도를 측정한다. 본 연구에서는 두 서비스 매쉬업에 사용된 서비스의 구성이 비슷하면 두 매쉬업이 비슷하다고 가정하였다. 서비스 매쉬업은 그 매쉬업을 구성하는 서비스의 집합으로 정의할 수 있다. 두 매쉬업 m_a와 m_b에 대하여 $|m_a| \leq |m_b|$ 일 때 그 유사도는 다음의 $SimMU$ 함수를 통해 구할 수 있다.

$$SimMU(m_a, m_b) = \frac{\sum_{s' \in |m_b|} max_{\, s \in m_a} SimS\,(s,\ s')}{|m_b|} \quad (2)$$

세번째 단계는 서비스 매쉬업 집합 유사도 측정 단계이다. 이 단계에서는 어떠한 개발자의 서비스 매쉬업 집합을 다른 개발자가 만든 다른 매쉬업 집합과 비교하여 그 유사도를 측정하고 가장 유사한 매쉬업 집합부터 순서대로 정렬한다. 서비스 매쉬업 집합 유사도는 두 매쉬업 집합을 이루는 매쉬업의 구성이

비슷할수록 높아진다. 서비스 매쉬업 집합은 그 집합에 해당되는 개발자가 만든 매쉬업의 집합으로 정의할 수 있다. 이때 두 서비스 매쉬업 집합 mu_a 와 mu_b 에 대하여 $|mu_a| \leq |mu_b|$ 일 때 그 유사도는 다음의 $SimMUS$ 함수를 통해 구할 수 있다.

$$SimMUS(mu_a, mu_b) = \frac{\sum_{m' \in mu_b} max_{\ m \in mu_a} SimMU\ (m,\ m')}{|mu_b|} \qquad (3)$$

마지막 단계는 서비스 추천 단계이다. 이 단계에서는 추천 대상 개발자의 서비스 매쉬업 집합과 가장 유사한 상위 N명의 개발자들의 서비스 매쉬업 집합들을 찾는다. N명의 개발자들이 사용한 서비스는 모두 추천 후보가 된다. 그리고 각 후보 서비스에 추천 대상과의 관련성에 대한 점수를 매긴다. 그리고 추천 대상 개발자가 사용한 적이 없는 서비스를 점수가 높은 순으로 추천한다. 후보 서비스에 대한 점수는 추천 대상이 사용한 서비스에 대한 유사도 평균인 서비스 점수와, 추천 대상이 개발한 매쉬업에 대한 유사도 평균인 매쉬업 점수를 가중합계하여 결정된다.

3. 실험 및 분석

본 연구에서는 실험을 실제상황과 비슷하게 진행하기 위하여 ProgrammableWeb[1]에서 데이터를 수집하였다. 이 실험에서 매쉬업 팔로워(Follower)가 매쉬업 개발자라고 가정하였다. 그리고 서비스에 대한 평점 정보가 없기 때문에 사용자의 서비스 팔로우 여부를 대신 사용하였다. 실험은 추천 대상의 매쉬업 집합을 매쉬업 등록 날짜 순으로 정렬한 뒤 반으로 나누어 과거의 것을 알고리즘에 입력하고 추천된 서비스를 나중에 사용했는지 여부를 확인하는 방식으로 진행되었다. 실험 변수는 매쉬업의 최소 크기와 매쉬업 집합의 최소 크기이다. 알고리즘 평가 척도로는 정확도(Precision)과 재현율(Recall)을 사용하였다. 비교 알고리즘은 사용자 기반 협업 필터링 알고리즘이다.

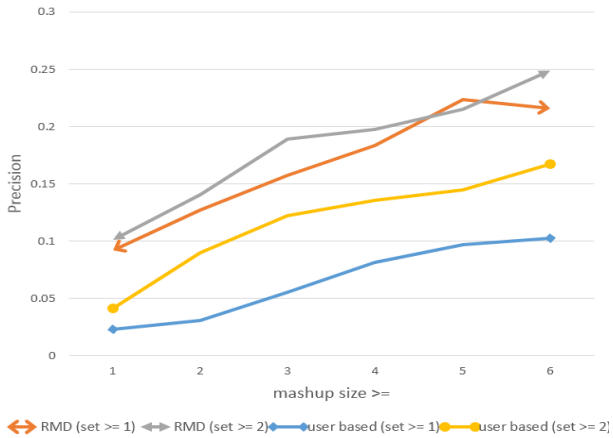


그림 1 RMD와 사용자 기반 협업 필터링의 정확도

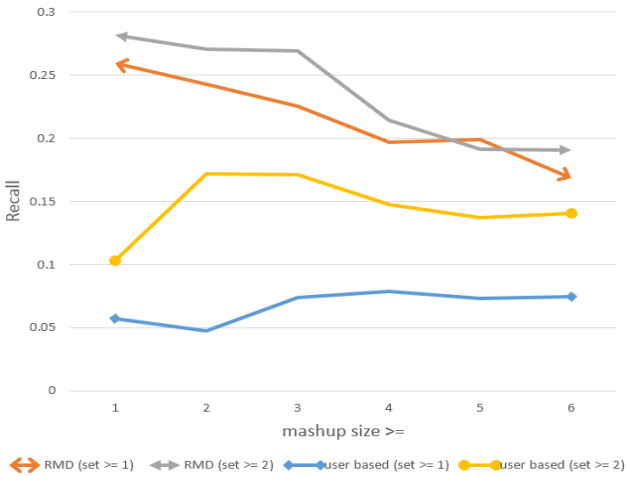


그림 2 RMD와 사용자 기반 협업 필터링의 재현율

실험 결과가 그림 1, 2에 정리되어 있다. 가로축은 매쉬업 최소 크기를 나타낸다. 각 꺾은선의 이름 옆 괄호 안에 표시한 것은 매쉬업 집합의 최소 크기를 의미한다. 매쉬업 최소 크기가 증가함에 따라 RMD는 전체적으로 정확도는 증가하고 재현율은 감소하는 추이를 보였고, 비교 알고리즘은 정확도는 증가하고 재현율은 전체적으로 조금 증가하거나 비슷한 수준을 유지하는 것을 보였다. 그리고 모든 구간에서 RMD가 비교 알고리즘 보다 높은 정확도와 재현율을 보였다.

4. 결론

본 논문에서는 서비스 매쉬업 개발자들의 성향을 분석하여 그에 맞는 서비스를 추천할 수 있는 방법을 제안하였다. ProgrammableWeb에서 매쉬업 팔로워 데이터를 바탕으로 본 연구에서 제안한 서비스 추천 방법이 실제 어떤 성능을 보이는지 실험하였다. 실험 결과 본 논문에서 제안하는 서비스 추천 방법(RMD)이 비교 알고리즘인 사용자 기반 협업 필터링에 비해 높은 정확도와 재현율을 보임을 확인하였다.

사 사

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.: 2016R1A2B4007585)

참고문헌

[1] Alonso, G., Casati, F., et al. “Web Services”, *Springer Berlin Heidelberg*, pp. 123-149, 2004.

[2] Liu, X., Fulia I., “Incorporating User, Topic, and Service Related Latent Factors into Web Service Recommendation”, *2015 IEEE International Conference on Web Services*, pp. 185-192, 2015.

[1] https://www.programmableweb.com/

코드 가독성 측정을 위한 소프트웨어 특징 목록

최상철, 김순태

전북대학교 소프트웨어공학과

114477aa@jbnu.ac.kr, stkim@jbnu.ac.kr

전라북도 전주시 덕진구 금암1동 백제대로 567

Software Feature List for Measuring Source Code Readability

Sangchul Choi, Suntae Kim

Dept. of Software Engineering, Chonbuk National University

567 Baekje-daero, deokjin-gu, Jeonju-si, Jeollabuk-do 54896, Republic of Koreat

요 약

소프트웨어 유지보수는 소프트웨어 life cycle 전체에서 드는 비용의 많은 부분을 차지하고 있다. 이 중 코드를 읽는 활동은 소프트웨어 유지보수에서 가장 많은 시간이 필요한 활동이다. 코드의 가독성은 사람이 코드를 얼마나 쉽게 읽을 수 있는지를 나타내는 지표로, 코드의 가독성은 코드를 읽는 활동의 시간에 큰 영향을 미치는 요소이다. 코드의 가독성을 측정하는 기존의 연구들은 코드가 읽기 쉬운지 아닌지 판단하는 것을 목적으로 했기 때문에, 코드의 가독성이 어느 정도 되는지를 알 수가 없었다. 본 논문에서는 코드의 가독성을 측정하는 모델을 수립하기 위한 특징 목록을 식별하고 향후 연구를 통해 사람들이 코드의 가독성 정도를 정확히 파악하는 데 도움을 주고자 한다.

1. 서론

소프트웨어 유지보수는 소프트웨어 life cycle 전체에서 드는 비용의 많은 부분을 차지하고 있다[1]. 이 중 코드를 읽는 활동은 소프트웨어 유지보수에서 가장 많은 시간이 필요한 활동이다[2]. 코드의 가독성은 프로그램 소스코드를 사람이 얼마나 쉽게 이해할 수 있는지를 나타내는 것으로 정의된다. 따라서, 코드의 가독성은 코드를 읽는 활동의 시간을 좌우하는 요소라 할 수 있고 소프트웨어 유지보수의 매우 중요한 요소 중 하나이다[2].

코드의 가독성을 측정하기 위한 연구는 이전에도 많이 진행되어 왔다. Buse는 LOC(Lines of code), 코드 내 식별자의 수, 주석의 수 등 코드의 정적인 특징들을 사용해 코드의 가독성을 측정하고자 했다[2]. 또, Halstead는 코드의 연산자와 피연산자를 사용한 몇몇 특징을 사용해 계산하는 값을 사용하여 코드의 가독성을 측정하고자 했다[3]. 또, Posnett은 기존의 모델에서 가장 설명력이 높은 특징을 선별해 모델을 축소하는 시도를 하기도 했다[4]. 이러한 연구들은 각각의 요소들을 사용해 주어진 코드가 사람이 읽기 쉬운지 아닌지를 판단한다. 즉, 위에서 소개한 방식으로 코드의 가독성을 알고자 한다면, 결과로는 코드가 읽기 쉬운지 어려운지로 판단된다. 코드의 가독성 판단 결과가 이분법적이기 때문에, 위에 소개한 방법으로는 자신의 코드가 얼마나 읽기 쉬운지를 알 수 없다.

본 논문에서는 코드의 가독성을 정량적으로 측정하기 위한 소프트웨어 특징 목록의 식별을 목적으로 한다. 코드의 가독성을 정량적으로 알 수 있다면 현재 코드의 가독성이 얼마나 되는지 객관적인 판단이 가능하다. 본 논문에서는 가독성을 측정하는 모델을 만들기 위한 소프트웨어 특징 목록을 만들어내는 과정을 소개한다.

2. 모델 수립

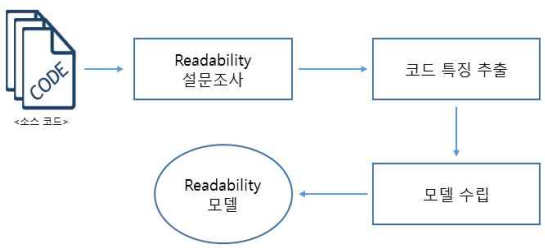


그림 1. 가독성 모델 생성 과정

본 논문은 **그림1**과 같이 세 단계를 통해 연구를 수행했다. 먼저, 사람들이 생각하는 코드의 가독성이 어떻게 되는지 알기 위해 설문조사를 했다. 다음으로, 설문조사에 사용된 코드로부터 특징들을 추출했다. 여기에는 LOC, 주석의 수 등이 포함된다. 이후, 설문의 결과를 바탕으로 코드의 가독성 점수를 측정하기 위한 모델을 수립했다.

2.1 설문조사

코드의 가독성은 사람이 판단한 사람이 코드를 이해할 수 있는 정도이기 때문에, 코드의 가독성을 보다 객관적으로 평가하기 위해 설문조사를 시행했다.

설문조사는 총 41명을 대상으로 진행되었다. 이들은 각각 20개 메서드에 대해서 1~5점사이의 가독성 점수를 매겼다. 이들의 경력은 1~3년이 22명, 3~6년이 10명, 7~10년이 3명, 10~15년이 2명, 15년 이상이 4명으로 분포되었다.

설문조사에 사용된 코드는 오픈 소스라이브러리의 코드들을 사용했다. antlr4, base-one, cglib, dbcp, fileupload, groovy, httpclient-, javacc의 소스코드 중 코드의 길이가 13~96인 메서드들만을 사용했고 2.2절에서 언급할 특징들이 되도록 고르게 분포하도록 선정했다.

2.2 코드의 특징 추출

코드의 가독성 점수를 측정하기 위해 코드의 특징들을 정의했다. 사용된 특징들은 LOC, 분기문의 수, 반복문의 수, 빈 줄의 수 등 [2,3,4]에서 사용된 특징 중 일부와 이를 확장하여 만들어낸 특징들을 추출했다. 추출된 특징들은 설문조사에서 사용될 코드를 선정하는데 사용됐고, 모델을 생성하는 데 있어 입력값으로도 사용됐다. **표1**은 코드에서 추출한 특징 목록의 일부이다.

2.3 모델 생성

본 논문에서는 설문조사 결과를 분석하여 코드 가독성 점수를 측정하는 모델을 만들어내기 위해 다중 선형회귀분석을 사용했다. 다중 선형 회귀분석은 종속변수와 둘 이상의 독립변수 간의 관계를 모델링 하는 방법이다[5]. 종속변수는 독립변수에 의해 나타나는 결과값이고, 독립변수는 결과값에 영향을 미치는 입력값이라 할 수 있다. 본 논문에서 사용된 독립변수는 2.2절에서 추출된 코드의 특징들이고 종속변수는 설문조사를 통해 얻어낸 가독성 점수이다. 분석할 특징들의 개수가 10개 이상이기 때문에 이 방법을 사용했다.

3. 모델 평가

모델을 생성한 후 multiple R-squared[6]을 사용해 모델의 설명력을 확인했다. 설명력이란, 만들어진 모델이 데이터를 얼마나 잘 표현하는가를 의미한다. 모든 특징을 사용한 모델의 설명력은 77.17%를 기록했다.

사용된 특징들의 수가 많아 사용되는 특징들의 수를 줄이기 위하여 단계선택(stepwise) 방식을 사용했다. 단계선택 회귀는 회귀를 통해 만들어진 모델을 최적화하는 알고리즘이다. 이 방법을 통해 생성된 모델은 7개의 독립변수만을 사용한 모델이고, 모델의 설명력은 74.59%를 기록했다. 아래 **표1**은 단계선택 방식을 적용한 결과로 선정된 코드 특징의 목록이다. **표1**의 가중치는 특징이 가독성에 미치는 영향을 뜻한다. 가중치가 음수이면 특징이 가독성을 낮추고 양수이면 특징이 가독성을 향상시키는 특징이라는 의미이다. 또, 가중치의 크기가 커질수록 해당 특징이 미치는 영향이 크다.

표 1. 주요 코드 특징 목록

특징	설명	가중치
LOC	코드 전체의 라인 수	-0.020
numOfComments	코드에 쓰인 주석의 수	0.040
numOfBlankLines	코드 전체에서 아무것도 쓰이지 않은 줄의 수	0.037
numOfBitwiseOperators	코드에서 사용된 &, | 등 비트 연산자의 수	-0.755
maxNestedControl	if문이나 for문 등 제어문이 제어문 내에서 중첩되어 사용되었을 때, 중첩된 수 중 가장 큰 수	-0.153
programVolume	소스코드가 품고 있는 정보량[3]	-0.001
entropy	코드의 어수선한 정도 혹은 복잡도[4]	-0.611

4. 결론 및 향후 연구

본 논문에서는 가독성을 측정하기 위한 특징 목록을 식별하였다. 이를 위해, 코드에서 추출할 수 있는 특징을 정의하고 이를 추출했다. 다음으로 추출된 특징들이 고르게 분포한 설문조사를 통해 사람들이 생각하는 코드의 가독성을 조사했다. 마지막으로, 설문조사의 결과를 바탕으로 코드의 특징들의 영향력을 알기 위한 모델을 생성했다.

기존의 연구들에서는 코드가 읽기 쉬운지 아닌지 만을 판단했다. 본 논문에서는 이를 점수화 하기 위한 특징 목록을 식별했다.

향후에는 설문조사와 데이터 분석을 하는 동안 식별된 추가적인 특징들을 정의하고 이를 통해 가독성을 정량적으로 측정하기 위한 메트릭을 만들어 낼 것이다. 또, 어떻게 하면 본 논문을 통해 만들어진 가독성 모델이 사람들에게 직접적으로 도움이 될지를 연구할 것이다.

5. 참고문헌

[1]B. Boehm and V. R. Basili. Software defect reduction top 10 list. Computer, 34(1):135{137, 2001.

[2]R. P. L. Buse and W. R. Weimer, Learning a metric

for code readability, Software Engineering, IEEE Transactions, vol. 36, no. 4, pp. 546-558, 2010.
[3]M. Halstead, “Elements of software science”, Elsevier New York, 1977.
[4] D. Posnett, A. Hindle, and P. Devanbu, “A simpler model of software readability”, in Proceeding of the 8th working conference on Mining software repositories, MSR, vol. 11, pp. 73-82, 2011.
[5] David A. Freedman, “Statistical Models: Theory and Practice", Cambridge University Press, 2009
[6] Investopedia, http://www.investopedia.com/terms/r/r-squared.asp, (available in 2017.01.26)

객체 지향 프로그램(C++)을 위장한 절차식(C) 패러다임 자동 식별화 구축

변은영 O1 손현승 2 장우성 3 김영수 5 김영철 4

홍익대학교 소프트웨어공학연구실

{eybyun[1], son[2], jang[3], bob[4]}@selab.hongik.ac.kr

정보통신산업진흥원 5

ysgold@nipa.kr[5]

Constructing an Automatic system for identifying the faked Procedural-Oriented Paradigm(C) within Object-Oriented Program (C++)

Eun Young Byun, Hyun Seung Son, Woo Sung Jang, Young S. Kim, Young Chul Kim

SE Lab., Hongik University

NIPA

요 약

대규모의 소프트웨어의 증가로 다수 개발자의 협업과 재사용성이 중요한 이슈이다. 바람직하지 않은 코드는 복잡도를 높이고 재사용성을 낮춘다. 이를 해결함으로써 소프트웨어의 품질과 유지보수성을 향상시키고자 한다. 본 논문은 객체 지향 프로그램(C++) 내에서 절차식(C) 패러다임 사용을 최소화하는 방법을 제안한다. 이를 위해 자동 식별 가시화 시스템을 제안하고 식별된 절차식 스타일의 리팩토링을 통해 소프트웨어의 복잡도를 낮추고 재사용성을 높여 품질을 향상시키고자 한다.

1. 서 론

객체 지향 프로그램은 속성과 메소드들의 그룹인 클래스로 구성된다. 하나의 객체는 특정한 작업과 연관된 구조로, 소프트웨어 복잡성을 낮추고 재사용이 가능하다[1,2]. 객체 지향 패러다임은 객체에 대한 충분한 이해가 필요하다. 하지만 대부분의 개발자는 객체 지향 프로그램을 절차식 패러다임으로 구현하는 실수를 한다[3]. 이것이 객체 지향 프로그램을 위장한 절차식 패러다임이다. C++의 경우, C와 이름이 비슷하듯이 유사한 부분이 많다[4]. 하지만 C++과 C는 객체 지향, 절차 지향이라는 결정적인 차이를 갖는다. 이 차이를 이해하지 못한다면 개발자는 C++에서 C패러다임으로 프로그래밍하는 실수를 할 수 있고 이 경우를 개발자가 스스로 인지하기는 쉽지 않다.

기존 연구[5,6]에서 소프트웨어 구조 가시화와 복잡도를 측정하는 툴을 개발하여 C++로 구현된 프로그램을 가시화했다. 그림1은 해당 툴에서 측정한 복잡도를 낮추기 위해 리팩토링을 수행하며 개선한 수치 리스트이다. 복잡도(Coupling)는 정적 분석 도구(CCheck)의 Style, Warning, Performance Violation이 연관성이 있고 이 항목들은 C패러다임을 포함한다. 따라서 리팩토링 과정에서 C패러다임을 사람이 직접 식별하고 수정한다. 수작업으로 수행되기 때문에 종종 식별하지 못하는 경우도 발생했다.

이 논문은 C++을 위장한 C패러다임을 정의하고, 프로그램의 전체 구조와 함께 C패러다임이 나타난 부분을 가시화하는 자동 식별 가시화 시스템을 제안한다. 이 결과를 통해 C패러다임을 자동으로 식별할 수 있다. C패러다임의 리팩토링을 통해 소프트웨어의 복잡도를 낮추면 품질 향상과 함께 객체 지향 특성인 재사용성을 높인다.

본 논문의 구성은 다음과 같다. 2장에서는 C++ 코딩 룰을 기반으로 C++을 위장한 C패러다임을 정의한다. 3장은 자동 식별 가시화 시스템으로 Tool-Chain 구조와 C패러다임 추출 가시화에 대해 기술한다. 마지막으로 4장에서는 결론 및 향후 연구를 언급한다.

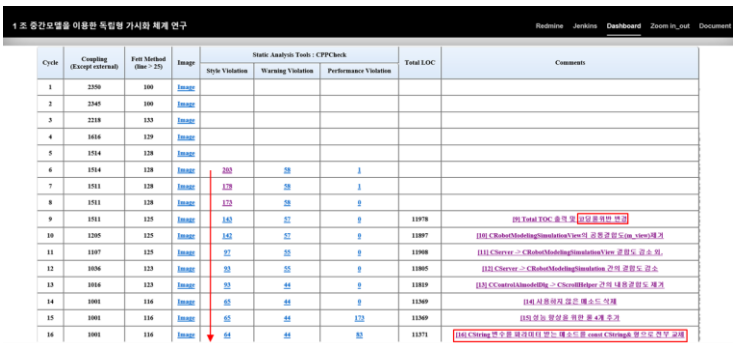


그림1 C++ 프로그램 복잡도 수치 리스트

† 이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548)

2. C++을 위장한 C패러다임

기존의 C++ 코딩 룰은 프로그램 코딩 시, 가장 바람직하지 않은 경우들을 대상으로 만들어 진다. 이 중에서도 많은 표준들은 C++에서 C패러다임의 사용을 최소화하기 위해서 정의된다[7]. C++을 위장한 C패러다임은 C++ 코딩 룰과 C++에 추가된 라이브러리를 기반으로 정의한다. 표1은 C++를 위장한 C 패러다임 항목이다.

	C 패러다임	C++ 패러다임
input/output	printf(), scanf()	std::cout, std::cin
메모리 할당	malloc(), calloc(), free()	new, delete
cast	(cast형)	static_cast<cast형>
struct	keyword 생략 X typedef사용	keyword 생략
enum	keyword 생략 X	keyword 생략
namespace	X	O
Boolean	X	O

표1 C++을 위장한 C패러다임 항목

◆ input / output

C는 문자열 입출력을 위해서 stdio 헤더파일의 printf(), scanf()함수를 사용한다. C++은 추가된 iostream 헤더파일의 cout, cin을 사용한다. cout, cin은 printf(), scanf()와 다르게 대상이 되는 변수형에 상관 없이 연산자 오버로딩을 통해 자동으로 바꾸어 준다. 따라서 printf(), scanf() 함수를 호출하는 경우, C패러다임으로 판단한다. 그림2는 C/C++ 패러다임 input/output 비교 예시이다.

C Paradigm

```
#include<stdio.h>

class A{
    void func(){
        printf("C Style");
        printf("printf function");
    }
}
```

C++ Paradigm

```
#include<iostream>

class A{
    void func(){
        std::cout << "C++ Style";
        std::cout << "cout function";
    }
}
```

그림2 C/C++ 패러다임 input/output

◆ 메모리 공간 할당

C는 메모리 할당을 위해서 malloc(), calloc(), free()를 사용한다. C++은 new, delete 키워드를 사용한다. C++은 경우에 따라 두 가지 방법을 모두 사용할 수 있지만 객체 생성 시에는 반드시 new와 delete를 사용해야 한다. 따라서 malloc(), calloc(), free()를 호출하는 경우, C 패러다임으로 판단한다. 그림3은 C/C++ 패러다임 메모리 할당 함수 비교 예시이다.

C paradigm

```
#include<iostream>

class A{
    void func(){
        int *i;
        i = (int*)malloc(sizeof(int));
        free(i);
    }
}
```

C++ paradigm

```
#include<iostream>

class A{
    void func(){
        int *i;
        i = new int;
        delete i;
    }
}
```

그림3 C/C++ 패러다임 메모리 할당

◆ cast

C는 적절한 헤더가 없을 시에는 타입에 대한 정보를 찾지 못해 컴파일 후에 오류가 발생한다. C++은 static_cast를 이용해 예상치 못한 오류를 방지할 수 있기 때문에 더 안전하다. 그림4는 C/C++ 패러다임 cast 비교 예시이다.

C paradigm

```
extern void Fun(Derived*);

class A{
    void Gun(Base* pb){
        Derived* pd = (Base*)pb;
        Fun(pd);
    }
}
```

C++ paradigm

```
extern void Fun(Derived*);

class A{
    void Gun(Base* pb){
        Derived* pd = static_cast<Derived*>pb;
        Fun(pd);
    }
}
```

그림4 C/C++ 패러다임 cast

◆ struct

C++은 C와 다르게 구조체 내에 함수의 선언이 가능하고 구조체 변수의 생성이나 함수 파라미터로 사용 시, struct 키워드를 생략할 수 있다. 따라서 구조체 변수 선언이나 함수 파라미터에서 struct 키워드를 사용한 경우, C 패러다임으로 판단한다. 그림5는 C/C++ 패러다임 struct 비교 예시이다.

C paradigm

```
class A{
    struct cStyle{
        …
    }
    void func(struct cStyle *c){
        struct cStyle s;
        …
    }
}
```

C++ paradigm

```
class A{
    struct cStyle{
        …
        void func(){
            …
        }
    }
    void func(cStyle *c){
        cStyle s;
        …
    }
}
```

그림5 C/C++ 패러다임 struct

◆ enum

struct 키워드와 마찬가지로 C++은 C와 다르게 enum 변수의 생성이나 함수 파라미터로 사용 시, enum 키워드를 생략할 수 있다. 따라서 구조체 변수의 생성이나 함수 파라미터에서 enum 키워드를 사용한 경우, C 패러다임으로 판단한다. 그림6은 C/C++ 패러다임 enum비교 예시이다.

C paradigm

```
class A{
    enum state {
        …
    }
    void func(enum state *s){
        enum state t;
        …
    }
}
```

C++ paradigm

```
class A{
    enum state {
        …
    }
    void func(state *s){
        state t;
        …
    }
}
```

그림6 C/C++ 패러다임 enum

◆ namespace

C++은 C에는 없는 namespace라는 새로운 키워드가 추가되었다. 동일한 이름을 가진 함수들의 충돌을 방지할 수 있다. C는 동일한 기능이지만 이름에 차이를 주기 위해서 앞에 식별할 수 있는 수식어를 붙여 사용한다. 수식어 구분은 대문자나 '_'로 구분한다. 따라서 동일한 기능의 이름 앞에 구분을 위한 단어를 포함하고 있는 메소드들이 있는 경우, C 패러다임으로 판단한다. 그림7은 C/C++ 패러다임 namespace 비교 예시이다.

C paradigm

```
class A{
    void aumoaPrintInfo(void){
        std::cout << "Made by Aumoa";
        return;
    }
    void libPrintInfo(void){
        std::cout << "Made by Lib";
        return;
    }
}
```

C++ paradigm

```
class A{
    namespace Aumoa{
        void PrintInfo(void){
            std::cout << "Made by Aumoa";
            return;
        }
    }
    namespace Aumoa{
        void PrintInfo(void){
            std::cout << "Made by Lib";
            return;
        }
    }
}
```

그림7 C/C++ 패러다임 namespace

◆ Boolean

C++은 C에는 없는 Boolean Type이 있다. C는 int형을 이용해 Boolean Type을 대체했다. 조건 식에 int형 변수만 있는 경우, C 패러다임으로 판단한다. 그림8은 C/C++ 패러다임 Boolean 비교 예시이다.

C paradigm

```
class A{
    void func(){
        int condition=1;

        while(condition){
            …
        }
    }
}
```

C++ paradigm

```
class A{
    void func(){
        bool condition=true;

        while(bool){
            …
        }
    }
}
```

그림8 C/C++ 패러다임 Boolean

3. 자동 식별 가시화 시스템

3.1 Tool-Chain 구성도

C++에서의 C패러다임을 자동 식별 가시화하기 위해서 연구실에서 개발한 xCodeParser와 오픈소스인 SQLite, Graphviz를 이용하여 Tool-Chain을 구현했다. 그림9는 Tool-Chain 구성도로 총 5단계로 나누어진다.

➢ Step1: 대상이 되는 소스 코드를 xCodeParser에 입력한다. 이 논문은 객체 지향 언어인 C++기반의 코드를 입력한다.

➢ Step2: 입력된 코드를 xCodeParser를 사용해 분석한다. 이 때, 각 파일에 대한 분석결과로 ASTM파일들이 생성되며 프로젝트, 클래스, 구조체, 멤버, 변수, 메소드 등에 대한 정보를 트리 형태로 저장한다.

➢ Step3: 데이터베이스 저장 단계는 ASTM 데이터를 원하는 구조로 구조화하여 데이터베이스에 저장한다. ASTM_content, ASTM_link, ASTM_local, ASTM_API 테이블을 생성하고, 이 데이터를 기반으로 C패러다임이 발견된 파일, 멤버와 C패러다임 종류 정보를 구조화하여 CCheck 테이블을 생성한다. 이 테이블로 코드의 어떤 부분에 어떤 C 패러다임이 발견되는지 확인할 수 있다.

➢ Step4: 구조 분석 단계는 데이터베이스에 저장된 정보를 DOT 문법에 맞게 재구성한다.

➢ Step5: 가시화 단계는 재구성한 정보를 Graphviz를 이용해 가시화한다.

3.2 C패러다임 추출 가시화

Tool-Chain을 통해 C++에서 C패러다임을 가시화한 결과는 그림 파일로 추출된다. 소스 코드를 구성하는 모든 클래스, 구조체, cpp파일에 대한 구조 정보를 노드로 가시화하고 컴포넌트(클래스, 구조체, cpp)간의 관계(상속 관계, 호출 관계, 객체 생성 관계)를 화살표로 가시화한다. 그림10은 가시화 노드, 관계의 종류를 나타낸다.

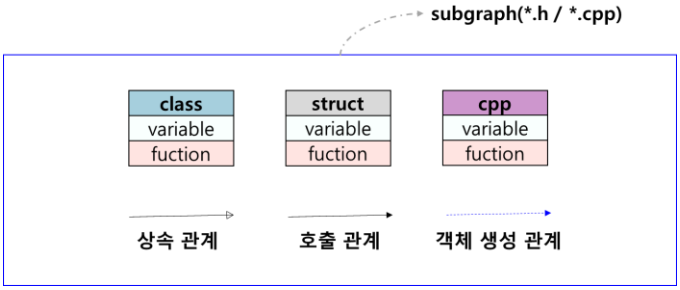


그림10 가시화 노드, 관계의 종류

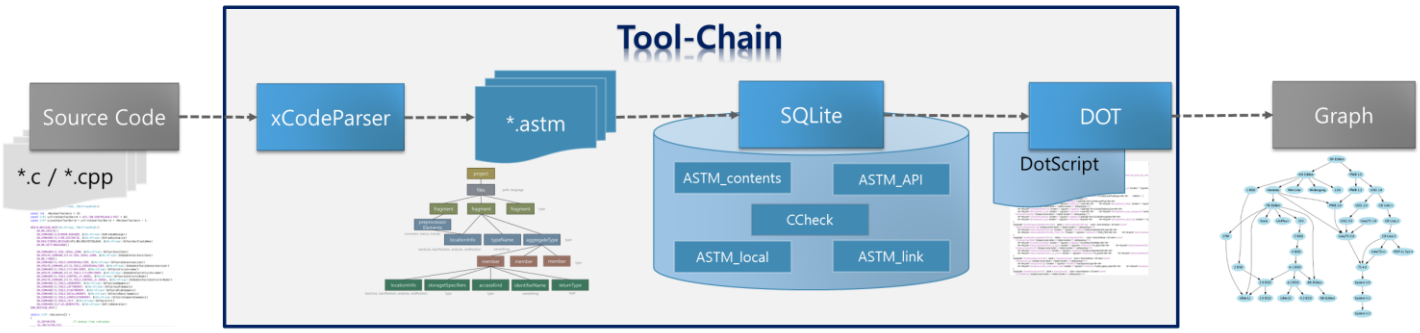


그림9 Tool-Chain 구성도

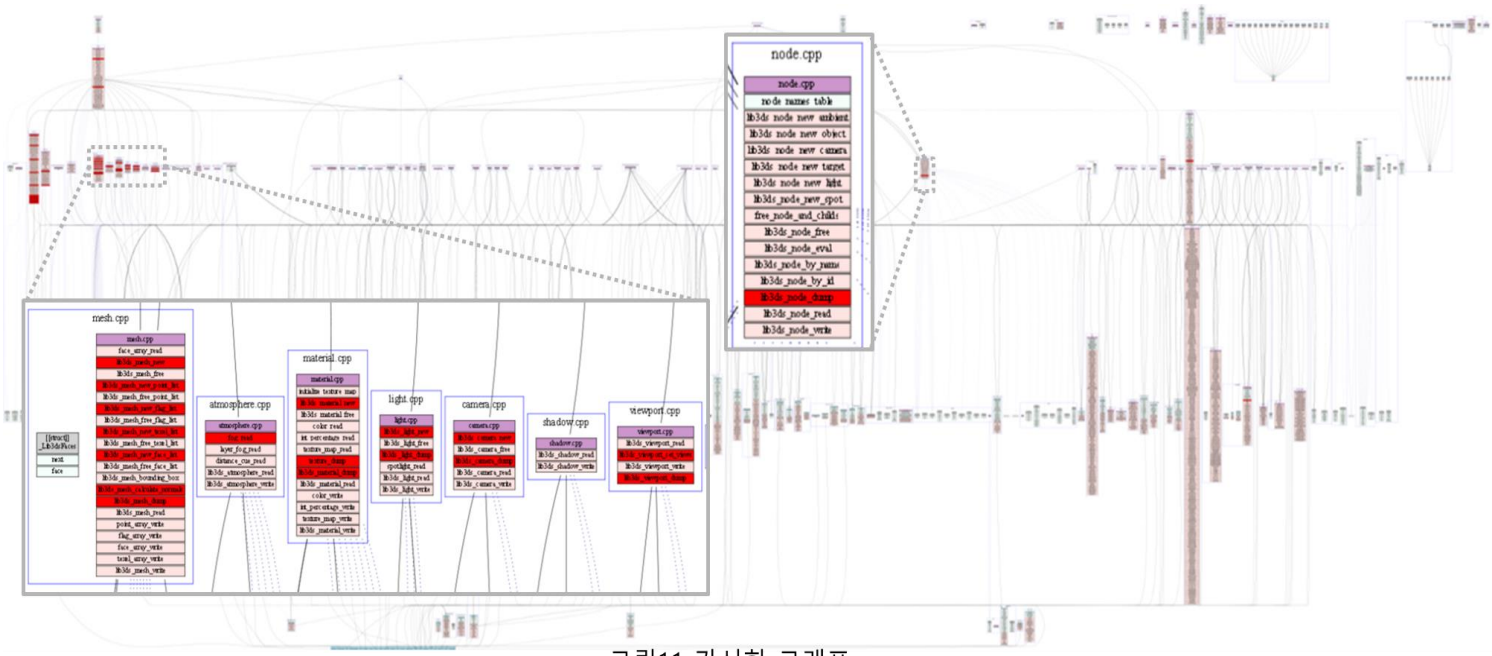


그림11 가시화 그래프

그림11은 RobotModelingSimulation코드를 Tool-Chain을 통해 가시화한 그래프이다. 해당 코드는 크고 복잡한 시스템이지만, 가시화를 통해 모든 컴포넌트들의 구조와 관계를 확인할 수 있다. C패러다임이 발견된 부분을 빨간색으로 표시하여 가시성을 높였다. 그림12는 CCheck 테이블로 어떤 C패러다임이 발견 되는지 그림에 도식화 하지 않았지만 이를 통해 확인 할 수 있다. 대표적으로 그림에서 node.cpp와 mesh.cpp에서 빨간색으로 표시한 부분과 CCheck 테이블에 데이터가 동일하고 C패러다임 종류를 알 수 있다. 해당 부분들을 리팩토링 하면 소프트웨어의 복잡도를 낮출 수 있고 재사용성을 높일 수 있다.

	mem_name	def_type	inout	namespace	bool	struct	enum	malloc	cast
₩RobotModelingSimulation₩lib3ds₩atmosphere.cpp	fog_read	Lib3dsBool				Y			
₩RobotModelingSimulation₩lib3ds₩camera.cpp	lib3ds_camera_new	Lib3dsCamera*						Y	
₩RobotModelingSimulation₩lib3ds₩camera.cpp	lib3ds_camera_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩chunk.cpp	lib3ds_chunk_debug_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩chunk.cpp	lib3ds_chunk_unknown	void	Y						
₩RobotModelingSimulation₩lib3ds₩chunk.cpp	lib3ds_chunk_dump_info	void	Y						
₩RobotModelingSimulation₩include₩drawstuff₩drawstuff.h	dsSimulationLoop	void				Y			
₩RobotModelingSimulation₩lib3ds₩file.cpp	lib3ds_file_new	Lib3dsFile*						Y	
₩RobotModelingSimulation₩lib3ds₩file.cpp	dump_instances	void	Y						
₩RobotModelingSimulation₩lib3ds₩light.cpp	lib3ds_light_new	Lib3dsLight*						Y	
₩RobotModelingSimulation₩lib3ds₩light.cpp	lib3ds_light_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩material.cpp	lib3ds_material_new	Lib3dsMaterial*						Y	
₩RobotModelingSimulation₩lib3ds₩material.cpp	texture_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩material.cpp	lib3ds_material_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩matrix.cpp	lib3ds_matrix_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new	Lib3dsMesh*						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_point_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_flag_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_texel_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_face_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_calculate_normals	void	Y					Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩node.cpp	lib3ds_node_dump	void	Y						

그림12 CCheck 테이블

4. 결론 및 향후 연구

본 논문은 객체 지향 프로그램을 절차식 패러다임처럼 구현하는 실수로 인한 소프트웨어의 복잡도와 재사용 문제를 해결하기 위해 C++에서의 C패러다임을 자동 식별 가시화 시스템을 언급한다. 추출된 부분을 객체 지향화하는 리팩토링을 통해, 소프트웨어의 복잡도를 낮추고 소프트웨어의 품질을 향상시키면서 객체 지향 언어의 특성인 재사용성을 높일 수 있다. 향후에는 C패러다임을 추가적으로 검출하고 식별한 C패러다임들을 자동으로 리팩토링 해주는 시스템을 구현할 예정이다.

참고문헌

[1] S Wiedenbeck, V Ramalingam, “Novice Comprehension of small programs written in the procedural and object-oriented styles”, International Journal of Human-Computer Studies, Vol.51, No.1, pp. 71-87, 1999

[2] 변은영, 박보경, 장우성, 김영철, “객체 지향 패러다임에서의 코드 재사용을 위한 응집도 레벨 식별 모범 사례”, 한국정보처리학회, Vol. 23, No.2, pp. 475-478, 2016

[3] R Fanta, V Rajlich, “Restructing legacy C code into C++”, ICSM, pp. 77-85, 1999

[4] Herbert Schildt, “C/C++ Programmer’s Reference”, McGraw-Hill, 2000

[5] 강건희, 손현승, 김영수, 박용범, 김영철, “SW 가시화 기반 리펙토링 기법 적용을 통한 정적 코드 복잡도 개선”, 한국정보처리학회, Vol. 21, No.2, pp. 646-649, 2014

[6] 변은영, 박보경, 장우성, 김영철, “가치 있는 모듈 식별을 위한 오픈 소스 기반 소프트웨어 현대화 시스템 구축”, 한국정보과학회, pp. 404-406, 2016

[7] H Sutter, A Alexandrescu, “C++ Coding Standards: 101 Rules, Guidelines, and Best Practices”, Pearson Education, 2004

N-그램 모델 기반의 코드 변경 추천 시스템

김태현 강성원 이선아 금창섭
한국과학기술원 경상대학교 한국전자통신연구원
{memo_less, sungwon.kang}@kaist.ac.kr saleese@gnu.ac.kr cskeum@gmail.com

Code Change Recommendation System based on the N-gram Model

Taehyun Kim Sungwon Kang Sunah Lee Changsup Keum
KAIST Gyeongsang National University ETRI

요 약

코드 변경 추천 시스템의 목적은 프로그래머가 코드 네비게이션에 소모하는 시간을 줄여 생산성을 증대시켜주는 것이다. 많은 변경 추천 시스템들은 리비전 히스토리 또는 동작 히스토리에서 연관규칙을 찾아내어 변경할 파일 또는 함수를 프로그래머에게 추천해준다. 하지만 연관규칙 분석법을 이용한 많은 변경 추천 시스템은 추천 정확도가 높지 않은 단점이 있다. 본 논문에서는 동작 정보 외에도 동작 순서 정보 역시 추천할 파일이나 함수를 특징 짓는데 도움을 줄 수 있을 것이라는데 착안하여, 시간순서대로 정렬된 동작 기록을 마이닝하는 N-그램 모델을 기반으로 한 코드 변경 추천 방법을 제시한다.

1. 서 론

코드 변경 추천 시스템(Code Change Recommendation System)은 프로그래머가 소프트웨어 진화 작업(software evolution task)을 수행할 때 그들에게 변경해야 할 파일 또는 함수를 추천해주는 시스템이다[1]. 프로그래머들은 소프트웨어 진화 작업을 할 때 변경할 파일을 찾는데 많은 시간을 투자하기 때문에, 변경해야 할 대상을 쉽게 찾을 수 있다면 소프트웨어 진화 작업에 드는 시간을 크게 줄일 수 있다.

히스토리 기반의 변경 추천에 관한 많은 연구들이 과거에 수행되어, 정적 분석 도구가 해주는 것 이상으로 프로그래머들에게 도움을 주어왔다. 관련 연구는 크게 두 가지의 패러다임으로 나뉘어진다. 첫 번째 패러다임에 속하는 연구들은 리비전 히스토리를 변경 추천에 이용한다. 예를 들어 Zimmerman et al. [4] 의 연구에서는 버전관리 시스템에 저장된 리비전 히스토리를 변경 추천에 이용하였다. 이러한 접근 방법은 높은 빈도로 함께 변경된 파일들 간의 연관규칙을 마이닝한다. 두 번째 패러다임에 속하는 연구들은 프로그래머의 동작 히스토리를 이용한다. Lee et al. [5]은 Eclipse Bugzilla 시스템에 저장된 Mylyn[6]으로 기록한 동작 히스토리를 변경 추천에 이용하였다. 이러한 접근 방법은 프로그래머가 과거에 보거나 변경한 파일 또는 메소드에 관한 정보로부터 연관규칙을 마이닝한다.

이 두 패러다임은 독립적으로 발전해왔지만 두 방법 모두 충분히 높은 정확도를 여전히 확보하지 못하고 있다. 또한 두 패러다임에 속한 많은 연구들이 연관성 규칙 기법을 사용한다는 공통점을 가진다. 연관규칙은 데이터의 출현 빈도만 고려하기 때문에 결과적으로 사용자 동작의 순서 패턴(sequential pattern)으로부터 얻을 수 있는 정보를 활용하지 못하는 한계점을 가진다. Zimmerman et al.이 활용한 리비전 히스토리는 소프트웨어 진화 작업 전후의 차이점에 관한 정보만을 제공한다. 이와는 달리 Lee et al. [5]이 활용한 동작 히스토리는 각 동작 기록에 시간의 기록(Time Stamp)도 함께 포함한다.

본 연구는 수정 동작이라는 한가지 정보만을 사용하는 [4]보다 관찰 동작이라는 추가적인 정보를 이용하는 [5]의 정확도가 좋다는 사실로부터 히스토리에 포함된 동작 순서라는 또 하나의 새로운 정보를 변경 추천에 이용하면 정확도 향상에 도움을 줄 수 있을 것이라는데 착안하여, 이러한 동작 순서를 이용하는 마이닝 기법을 제안한다. 이 중에서도 순서로 정렬된 기록을 엄격하게 마이닝하는 N-그램 모델(N-gram model) 기법을 적용해 변경 추천 모델을 만드는 방법과 해당 방법을 구현한 도구인 NCRS(N-gram model based Change Recommendation System)를 제안한다.

본 논문의 구성은 다음과 같다. 제2장에서는 변경

관련 연구들을 소개한다. 제 3장에서는 N-그램 모델 기반의 변경 추천 방법 및 해당 방법을 이용하는 추천 시스템에 대해 자세히 설명한다. 제 4장에서는 본 연구의 결론을 제시하며 향후 연구 방향에 대해 논의한다

2. 관련 연구

2.1 리비전 히스토리를 추천에 이용하는 연구

Gall et al.[7]은 CVS와 같은 버전 관리 시스템에 저장되어있는 리비전 히스토리로부터 어떤 파일들이 높은 빈도로 함께 변경되었는지를 분석하여 논리적인 커플링을 찾는다.

Zimmermann et al.[4]은 ROSE라는 툴을 개발하였다. ROSE는 버전 관리 시스템 중 하나인 CVS에서 제공하는 리비전 히스토리를 사용하는데, 이 기본적으로 제공하는 어떤 파일이 변경되었냐를 아는 수준에서 한 발짝 나아가 각 리비전 히스토리를 분석하여 어떤 파일에서 어떤 메서드가 변경 되었는지를 파악하고 트랜잭션(transaction)이라고 명명한, 변경점들의 집합을 생성한다. 마지막으로 트랜잭션으로부터 연관 규칙을 마이닝하여 추천 모델을 생성한다. 사용자로부터 ROSE에 어떤 메서드를 변경중인지에 대한 정보가 입력되면 ROSE는 해당 메서드와 연관된 메소드의 목록을 추출하여 신뢰도(confidence)가 큰 순서대로 추천한다.

위의 연구들은 공통적으로 CVS, subversion, 그리고 Git등의 버전 관리 시스템에서 작성한 리비전 히스토리를 이용한다. 리비전 히스토리를 사용하는 것에는 크게 두가지 한계점이 있다. 첫 번째 한계점은 리비전 히스토리는 프로그래머가 커밋을 하는 순간에 일괄적으로 작성 된다는 것이다. 이러한 특성 때문에 프래그래머의 활동이 어떤 시각에, 어떤 순서로 행해졌는지에 대한 정보가 누락된다. 두 번째 한계점은 리비전 히스토리는 오로지 변경점만을 기록한다는 사실이다. 서론에서 소개한 Bugzilla의 버그리포트에도 나와있듯이, 소프트웨어 진화작업에 드는 노력의 대부분은 코드를 변경 하는 행위가 아닌 변경해야 할 코드를 찾기 위해 여러 파일을 관찰하는 관찰하는 행위로 이루어져있다. 하지만 리비전 히스토리는 이러한 행위 기록을 포함하지 않는다.

2.2 동작 데이터를 마이닝하는 연구

2.1에서 설명한 리비전 히스토리의 한계를 극복하기 위해 많은 연구자들이 프래그래머의 동작정보를 마이닝하는 연구를 수행하였다. 단순히 어떤 파일이 어떻게 변경 되었는지를 알려주기 때문에 버전 관리 시스템마다 리비전 히스토리의 형식은 달라도 제공하는 정보 자체는 대동소이하던 것과 달리 본 절에서 소개하는 연구들은 연구자마다 독자적인 정의를 내리고 해당 정보만을 마이닝한다는 차이점이 있다.

Kersten et al.[6] 은 Eclipse 플러그인 Mylyn을 개발하였다. Mylyn은 Eclipse를 사용하는 프로그래머의 동작 히스토리를 자동적으로 기록하는 도구이다. Mylyn은 사용자의 동작을 파일 열기, 패키지(디렉토리) 전개(propagation), 관찰, 변경 네 가지로 정의하고 있으며, 관찰 및 변경 동작의 경우 파일 내부의 메서드 레벨의 정보까지 기록한다. Mylyn은 Eclipse에서 제공중인 이슈 추적 시스템(issue tracking system)인 Bugzilla에 채택되었으며, Eclipse IDE와 관련된 오픈소스 프로젝트들에 대한 소프트웨어 진화 과정을 기록하고 있다.

Kobayashi et al.[8]은 PLOG라는 도구를 개발하였다. 이 도구는 사용자 동작 중 관찰 및 변경 동작만을 파일 수준으로 기록한다. 특이점으로는 사용자의 동작 외에도 IDE에서 빌드한 프로그램을 실행시켰을 때 출력되는 표준 출력(stdout) 및 표준 에러(stderr) 또한 기록한다.

Gu et al.[9]은 IDE++라는 도구를 개발하였다. 해당 도구는 프래그래머의 동작을 최대한 세밀하게 기록하기 위해 사용자 동작을 키 스트로크, 파일 실행, 리팩토링, 파일 저장, 등 44종류로 분류하였다.

본 절에서 소개한 연구들은 동작데이터를 저마다의 방법으로 수집만 할 뿐 수집한 데이터로 변경추천을 하지 않는다.

2.3 동작 데이터를 추천에 이용하는 연구

Lee et al.[5]은 MI라는 도구를 개발하였다. MI는 ROSE[4]를 확장시킨 추천 시스템이다. ROSE가 리비전 히스토리를 사용함으로써 가지게 되는 한계점을 극복하기 위해 리비전 히스토리 대신 동작 히스토리를 이용해 관찰 동작에 관한 정보까지 이용해 변경추천을 하는 것을 목적으로 하며, 문맥을 구성하는 동작의 종류 및 개수에 에 대한 조건을 자유롭게 설정할 수 있다는 특징이 있다.

Zou et al.[10]은 높은 빈도로 레퍼런스 스위치가 일어나는 파일 사이에 논리적 결합을 co-change, co-view, 그리고 change-view 세 종류로 정의하여 파일레벨의 추천에 이용하는 연구를 수행하였고, Robbes et al.[11]은 비슷한 접근 방법을 가진 연구를 입도가 높은 동작 정보를 이용해 메서드 레벨에서 추천하는 연구를 수행하였다.

Maalej et al.[12]은 사용자 동작 데이터를 수집할 때 사용자가 사용하는 도구에 대한 정보도 함께 수집하여 추후 새로운 작업을 할 때 어떤 도구를 사용하는 것이 적절한지 추천해주는 방법에 관한 연구를 수행하였다.

본 절에서 소개하는 연구들의 경우, 동작데이터를 어떻게 정의하고 추천을 발생시키기 위한 문맥을 어떻게 정의하는지는 제각각 다르지만, 문맥과 추천 대상 사이의 논리적 결합은 관계가 발생하는 빈도를 통해 규칙을 정의하는 연관 규칙 기법을 이용해 정의한다는 공통점을 가지고 있으며, 서론에서 언급한 것처럼 동작데이터의 순서를 고려하지 않는다는 한계점을 가진다.

2.4 N-그램을 소프트웨어 공학 분야에 활용하는 연구

Hindle et al.[12]은 프로그래밍 언어와 자연어의 통계적 유사성을 입증하기 위한 연구를 수행하였다. 프로그램 코드와, 영어로 쓰여진 말뭉치로부터 N-그램을 추출해 각각 N의 크기에 따른 교차 엔트로피의 분포를 분석한 결과 영어와 프로그래밍언어 사이에는 높은 통계적 유사성이 있다는 것을 확인하였다.

Santos et al.[14]은 제 3자 API를 사용하는 소스코드에 N-그램 모델을 적용하여 API콜을 할 때 API문장을 완성하기 위해 입력해야 할 토큰들을 추천하는 연구를 수행하였다.

3. N-그램 모델 기반의 변경 추천 방법

3.1 동작 트레이스

N-그램 모델을 구축하기 위해서는 모델을 학습시키기 위한 데이터를 입력해야 한다. 본 연구는 N-그램 모델을 학습시키기 위해 입력하는 입력하는 데이터를 동작 트레이스라고 정의하고, 소프트웨어 진화 작업을 하는 동안 기록된 로그 파일 하나를 동작 트레이스의 기본 단위로 정의한다. 그림 1 은 사용자의 동작을 기록한 로그가 어떻게 동작 트레이스로 표현되는지를 보여준다.

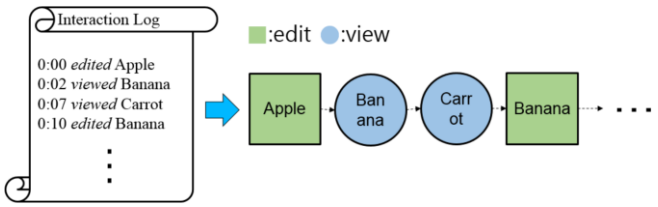


그림 1 동작 트레이스 생성

3.2 문맥 형성

문맥이라는 단어의 사전적인 의미는 '현재 상황을 파악하는데 쓰이는 정보'이다. 그러나 변경 추천에 관련된 연구에서는 문맥이란 용어를 '추천을 발생시키는 암묵적인 질의문'이라는 의미로 사용한다[1]. 본 연구는 문맥을 N 개의 연속된 사용자 동작 데이터와, 그 뒤에 최초로 수행되는 변경 동작 데이터의 묶음으로 정의한다.

$$C_j = \{a_j, a_{j+1}, ..., a_{j+n-1}\}$$

3.3 N-그램 모델 생성

본 연구에서 변경 추천을 위해 사용하는 N-그램 모델은 다음과 같은 쌍을 이용해 구축한다.

$$(C_j, RC) \equiv (\{a_j, a_{j+1}, ...a_{j+n-1}\}, FE(\{a_{j+n}, a_{j+n+1}, ...\}))$$

C는 추천후보(Recommendation Candidate)를 의미하고, 함수 FE(First Edit)는 인수로 주어진 동작데이터에서 첫 번째 변경 동작을 반환하며, 함수 FE의 인수로 주어진 동작 데이터의 집합은 동작 트레이스에서 문맥 이후의 데이터들을 가리킨다. 그림 2는 9개의 동작으로 이루어진 동작 트레이스로부터 3-그램 모델을 생성하는 과정을 나타낸 것이다. 그림 2의 화살표 왼쪽 부분은 9개의 동작데이터를 시간 순서대로 3개씩 묶고, 각 묶음과 그 이후에 처음 나타나는 수정 동작 데이터를 짝 짓는 모습을 나타내고, 화살표 오른쪽 부분은 짝지은 쌍들로 문맥-추천대상 쌍의 트리구조를 생성한 모습을 나타낸다.

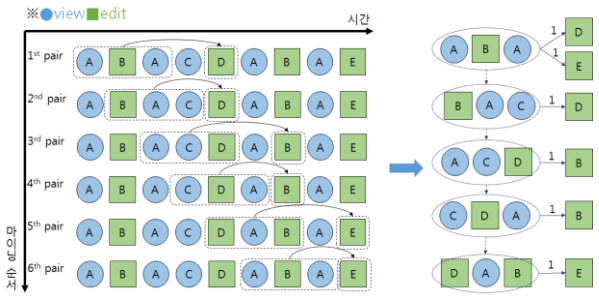


그림 2 N-그램 모델을 생성 하는 과정

3.4 변경 대상 추천

N-그램 모델에는 과거 데이터에서 나타나는 N-그램과 이어지는 변경 동작 및 그 빈도를 그림 2의 오른쪽에 나타나 있는 것처럼 빈도를 가중치로 가지는 가중 그래프(weighted graph)의 형태로 저장하고 있다. 문맥에 대한 변경 추천을 할 때 추천 후보들의 조건부 확률을 일일이 계산할 필요 없이 추천 후보들의 발생 빈도 순으로 정렬한 다음 사용자에게 추천해주면 된다.

그림 3은 사용자의 동작데이터를 입력 받아 추천대상을 반환하는 과정을 표현한 것이다. 사용자가 수행한 동작으로부터 ACD라는 3-그램의 문맥을 생성한 다음 N그램 모델에 질의한 결과 일치하는 것이 존재하며, 추천대상인 B를 반환해주는 것을 볼 수 있다. 사용자가 ACD를 관찰하거나 수정하는 동작을 수행 하자 해당 동작들로 문맥을 형성한 뒤, 해당 문맥이 N-그램 모델 내에 존재하는지 확인 한 뒤 해당 문맥의 유일한 추천 대상인 B를 사용자가 변경해야 할 대상이라고 추천해주는 모습을 나타낸다.

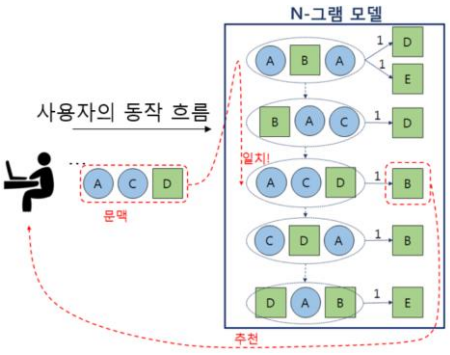


그림 3 N-그램 모델로부터 변경 추천을 받는 과정

3.5 N-그램 모델 갱신

사용자의 소프트웨어 진화 작업이 끝난 뒤 수행되는 N-그램 모델의 갱신은 다음과 같은 과정을 통해 이루어진다. 우선 4.4의 과정 동안 입력된 전체 동작 데이터를 이용해 동작 트레이스를 생성한다. 생성한 동작 트레이스에 3.2~3.3절에서 소개한 과정을 적용하여 문맥-추천 대상 쌍을 생성한다. 이 쌍들을 N-그램 모델과 비교하여, 문맥 추천대상이 둘 다 존재할 경우에는 가중치를 증가시키고, 문맥은 존재하나 추천대상이 존재하지 않을 경우에는 문맥에 해당하는 트리의 부모 아래에 추천대상에 해당하는 새로운 자식을 추가한다. 마지막으로 문맥과 추천대상 모두 존재하지 않을 때에는 문맥을 부모로, 추천대상을 자식으로 하는 새로운 트리를 추가한다.

그림 4는 앞에서 설명한 N-그램 모델의 갱신과정을 3-그램으로 예를 들어 나타낸 것이다. ABA의 경우 N-그램 모델 내에 존재하지만 추천대상에 해당하는 E가 존재하지 않기 때문에 새로운 자식으로서 추가되는 모습을 보여준다. BAC의 경우엔 추천대상에 해당하는 D도 함께 존재하기 때문에 출현빈도에 해당하는 가중치를 1 증가시킨다. 마지막으로 CDA의 경우 N-그램 모델에 존재하지 않기 때문에 추천대상인 B와 트리를 형성하여 N-그램 모델에 새롭게 추가된다.

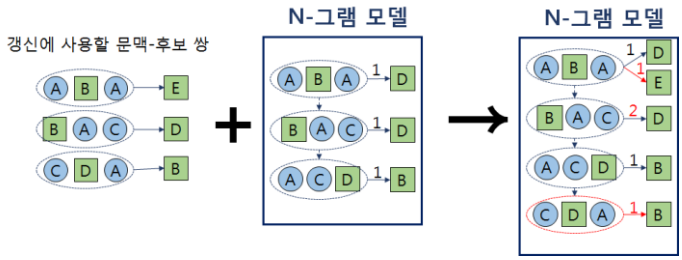


그림 4 사용자의 작업이 끝난 후 N-그램 모델을 갱신하는 과정

3.6 N-그램 모델 기반의 변경 추천 시스템의 구조

N-그램 모델 기반의 변경 추천 시스템(N-gram model based Change Recommendation System, NCRS)의 구조도는 그림 5와 같다. NCRS의 입력에는 두 종류가 있다. 첫 번째 입력은 NCRS를 적용할 프로젝트의 과거 동작 트레이스들이다.

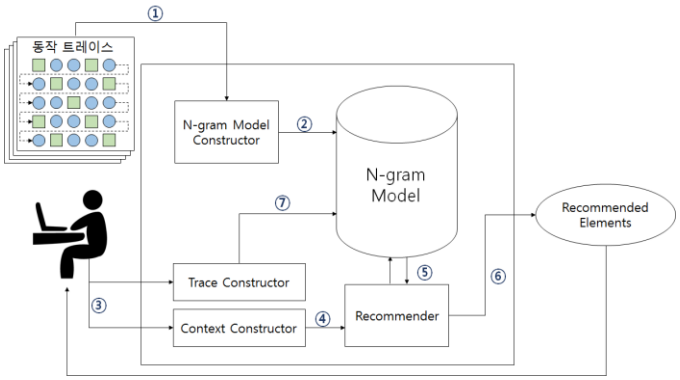


그림 5 NCRS의 시스템 구조도

4. 결론

본 연구는 기존 연구가 고려하지 않던 프로그래머의 동작 순서를 고려하는 변경 추천을 하기 위해 N-그램 모델을 사용하는 방법을 제안하였다. 향후 연구로는 본 연구에서 제안한 방법을 구현하고, 또한 다른 변경 추천 시스템과 성능 비교를 할 수 있는 프레임워크를 구현하여 기존 연구와 비교하는 실험을 수행할 예정이다.

참 고 문 헌

[1] M. Robillard, R. Walker, and T. Zimmermann, “Recommendation systems for software engineering,” IEEE Software., vol. 27, no. 4, pp. 80-86, Jul./Aug. 2010.

[2] “Bug 261613 – Java compare does not give focus to editor”, Eclipse Bugzilla. Eclipse Foundation, Feb. 2009.

[3] “Bug 241244 – [Status Handling] StatusManager.handle(status, StatusManager.BLOCK) does not block within Wizard.performFinish,” Eclipse Bugzilla. Eclipse Foundation, Feb. 2008.

[4] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, “Mining version histories to guide software changes,” Proc. ICSE, vol. 31, no. 6, pp. 429-445, 2005.

[5] S. Lee, S. Kang, S, Kim and M. Staats, “The Impact of View Histories on Edit Recommendations,” in IEEE TSE., vol 41, no. 3, pp. 314-330, 2015.

[6] M. Kersten and G. C. Murphy, “Mylar: a degree-of-interest model for ides,” in Proc. AOSD, 2005, pp.159-168.

[7] H. Gall, K. Hajek, and M. Jazayeri, “Detection of logical coupling based on product release history,” in Proc. ICSM, 1998, pp. 190-198.

[8] T. Kobayashi, N. Kato, and K. Agusa, “Interaction histories mining for software change guide,” in Proc. RSSE, 2012, pp. 73-77.

[9] Z. Gu, D. Schleck, E. T. Barr, and Z. Su, “Capturing and exploiting ide interactions,” in Proc. Symposium on New Ideas in Programming and Reflections on Software, 2014, pp. 83-94.

[10] L. Zou, M. Godfrey, and A. Hassan, “Detecting interaction coupling from task interaction histories,” in Proc. ICPC, 2007, pp. 135-144.

[11] W. Maalej, T. Fritz, and R. Robbes, “Collecting and processing interaction data for recommendation systems,” in Recommendation Systems in Software Engineering, M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, Eds. Springer BerlinHeidelberg, 2014, pp. 173-197.

[12] A. Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. "On the naturalness of software. In Proceedings of the 34th International Conference on Software Engineering (ICSE '12). IEEE Press, Piscataway, NJ, USA, 2012, pp. 837-847.

소프트웨어 유지보수 효율 향상 지원을 위한 의사코드 및 소스코드 양방향 자동 변환 연구

권혁무, 장현수O, 박동민, 서영석*

영남대학교 컴퓨터공학과

21211843@ynu.ac.kr, ddi00190@yu.ac.kr, 21211871@ynu.ac.kr, ysseo@yu.ac.kr

An approach to automated Bi-directional conversion for supporting efficiency improvement of software maintenance

Hyuck-Moo Gwon, Hyun-Soo JangO, Dong-Min Park, Yeong-Seok Seo*

Department of Computer Engineering, Yeungnam University

요 약

소프트웨어 개발 생명 주기 중 소프트웨어 유지보수는 소프트웨어 공학 커뮤니티들에서 항상 이슈가 되어 왔다. 소프트웨어 요구사항의 변경, 신기술의 출시, 오류 발생 예방 등에 따라 기개발된 소프트웨어의 유지보수가 반드시 필요하게 되지만, 거대한 소프트웨어 시스템 아키텍처를 쉽게 이해하기는 상당히 어렵기 때문에 많은 소프트웨어 산업 분야에서 유지보수에 투입되는 비용과 시간이 초기 개발보다 더 많이 필요로 하는 실정이다. 따라서 본 논문에서는, 기존 복잡한 소프트웨어에 대한 유지보수 작업시 설계자 및 개발자 이해도 향상에 도움을 줄 수 있는 한글기반 의사코드 자동 생성 기법 알고리즘을 연구한다. 또한 생성된 의사코드를 추가 및 변경하였을 시 해당 부분을 소스코드로도 변환시켜 줄 수 있는 기법을 제공하여 양방향 변환을 지원할 수 있도록 한다. 연구된 기법은 실제 산업체에서 근무하고 있는 개발자를 대상으로 평가를 진행하였고, 실질적인 활용 측면에서 다각도로 분석하였다.

1. 서 론

소프트웨어 프로세스상의 각 단계들 중 소프트웨어 유지보수 분야의 경우 학계와 업계에서 반드시 고려되어야 하는 부분이지만 난이도가 매우 높고 도전적인 작업으로 평가되어져 왔다. 소프트웨어의 경우 개발 후 요구사항의 변경, 오류 예방 등을 위해 변경이 지속적으로 일어나게 되는데 비용과 시간의 제한으로 인해 소프트웨어 설계품질을 고려하지 않고 변경하게 되고 이는 소프트웨어 에이징(Software aging) [1]이나 소프트웨어 붕괴(Software decay) [2]로 이어지게 된다. 즉, 시간이 지날수록 소프트웨어 유지보수성이 떨어지고 시스템의 복잡도는 증가하게 된다. 특히 과거부터 개발되어 현재까지 사용하고 있는 거대하고 복잡한 레거시 시스템(legacy system)의 경우, 유지보수성이 크게 저하되어 있기 때문에 유지보수 비용이 개발 비용에 비해 훨씬 상회한다. 레거시 시스템들은 초기 소프트웨어 설계자 및 개발자가 없고 설계 및 구현문서들은 존재하지 않거나 과거버전이기 때문에 최신 현황을 반영하고 있지 못하며 때로는 소스코드들 조차 패키지들로 분리되어 있지 못하고 하나의 폴더에 통째로 저장되어 있는 경우가 대부분이기 때문이다 [3,4,5].

이와 같은 문제를 해결하기 위해 소프트웨어 공학자들은 소프트웨어 유지보수 단계에서 많은 소스코드들로부터 전체적인 시스템 아키텍처를 보다 쉽게 이해할 수 있도록 하기 위해 높은 응집도(high cohesion)와 낮은 결합도(low coupling)을 가진 소프트웨어 모듈들을 클러스터링하는 연구들 [6, 7, 8, 9, 10]이 진행되고 있지만, 클러스터링 개수 설정하는 방법에 따라 모듈 품질의 차이가 크고 기법 활용을 위해서는 상당한 시간이 소비된다는 한계가 존재한다.

본 논문에서는 소프트웨어 이해도를 향상을 통한 유지보수 효율 향상 지원을 위해, 소프트웨어 설계자 및 개발자가 쉽게 활용할 수 있고 직관적인 방식의 이해가 가능한 한글기반 의사코드(pseudo-code) 자동 생성 기법 알고리즘을 연구한다. 이와 함께, 생성된 의사코드를 추가 및 변경하였을 시 해당 부분을 소스코드로도 변환시켜 줄 수 있는 기법을 제공하여 양방향 변환을 지원할 수 있도록 한다.

소프트웨어 유지보수 시 전체코드에 대한 의사코드를 파악할 수 있게 될 경우 복잡한 소스코드를 모두 탐색할 필요없이 상세설계를 검토할 수 있고 각 기능들 사이의 연관관계를 쉽게 파악할 수 있어 보다 짧은 시간에 전체적인 소프트웨어 이해도 향상을 이루어 낼 수 있다. 이는 코드 수정을 용이하게 해주고 최소의 노력으로 오류들을 바로잡을 수 있는 기회를 제공한다. 이와 더불어, 추가 및 수정이 발생한 의사코드는

O: 발표자
*: Corresponding author

자동적으로 소스코드로도 변환을 지원하여 반복적인 프로그래밍을 개선하고 주석(comment)작성에 대한 부담감을 덜어줄 수 있는 장점들을 얻어낼 수 있다 [11,12].

연구된 기법은 상세히 구현하여 도구화하였고, 실제 국내 소프트웨어 업체에 근무하고 있는 개발자를 대상으로 평가를 진행하여 활용 가능성을 보여주었다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구들에 대해 분석하고 3장에서는 본 연구에서 제시하고 있는 의사코드 및 소스코드 양방향 변환 기법 알고리즘에 대해 소개한다. 4장에서는 본 연구에서 수행한 실험설계와 과정에 대해 설명하고 결과에 대해 논의한다. 5장에서는 결론 및 향후 연구를 제시하고 본 논문을 마무리한다.

2. 관련 연구

소프트웨어 유지보수단계에서 소프트웨어 이해도 향상을 위한 의사코드 변환 연구는 다양한 언어별로 진행되고 있지만, 심도있는 연구의 필요성이 강조되고 있다. 기존 연구의 경우 공통적으로 특정 소스코드 문자열을 배열에 저장한 후 의사 코드로 변환시 각 단어들을 토큰(token)화하여 문자열로 재변환시켜 출력하는 방법을 사용하고 있고, 구현시 Map을 활용하기도 하였다 [13].

소프트웨어 공학 분야 최고의 국제학회 중 하나인 IEEE/ACM International Conference on Automated Software Engineering (ASE) 2015에 발표된 Fudaba et al. [14,15] 연구에서는 통계적 기계변환 (SMT: Statistlcal Machine Translation) 프레임워크를 기반으로 소스코드로부터 의사코드를 자동적으로 생성해주는 기법 알고리즘을 제안하였다. 파이썬(Python)언어 소스코드를 영어 또는 일어를 지원하는 의사코드로 변경할 수 있도록 하여 소스코드 이해도 향상에 도움을 줄 수 있도록 하였다. 그러나, 복잡한 조건식(괄호안에 두 개 이상의 조건이 동시에 들어가는 식)을 의사코드로 변환과정에 대한 방식이 없고 적용가능한 언어가 파이썬으로 제한적이며 영어 및 일어는 지원하지만 한글은 지원하지 않는다.

본 연구에서는 이전 연구들을 응용하여 기본적인 형식은 유지하되, C언어 소스코드를 라인별로 의사코드로 자동변환할 수 있도록 하였고, 반대로 의사코드를 다시 C언어 소스코드로도 자동변환할 수 있도록 제안하였다. 또한 기존 연구에서는 제시되어있지 않던 복잡한 조건식들에 대해 본 기법에서는 재귀(recursive) 함수를 기반으로 한 신규 알고리즘을 개발해 오류 없이 변환 가능하도록 하였으며 영어 및 한글을 지원할 수 있도록 제안하였다.

3. 의사코드 및 소스코드 양방향 자동 변환 기법 알고리즘

본 논문에서는 C언어 소스코드의 다양한 종류의 문장들 중 함수 선언문 및 종료문, 변수 선언문, 출력문, 반복문, 조건문에 대한 변환에 대해 구체적으로 살펴보고 본 기법에서 추가적으로 제공하는 함수 호출 관계 정보에 대해 소개한다.

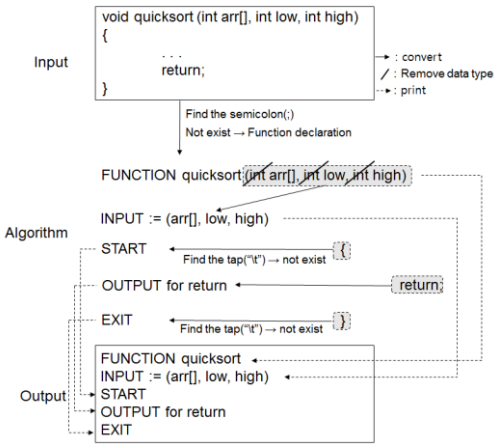
3.1 소스코드에서 의사코드로의 변환 기법

다양한 구문들을 대상으로 한 소스코드에서 의사코드로의 변환을 위해, 기본적으로 *Java*의 *Scanner Class*에 속해있는 *NextLine*함수를 사용하여 소스코드 한 행을 읽도록 하고 *String Class*의 *indexOf*함수를 사용하여 변환이 필요한 문자열을 검색한다. 검색 결과 변환이 필요한 문자열이 존재할 경우 *String Class*의 *replace*함수를 이용하여 해당 문자열을 의사 코드 형태로 변환 및 출력하도록 하였다.

3.1.1 함수 선언문 및 종료문

소스코드에서 읽어 온 문자열이 *int*나 *void*같은 형태를 포함하면서 세미콜론(;)이 없을 경우 함수의 선언 부분으로 식별하도록 하였다 (변수 선언과의 구분 용도). 함수의 선언 부분일 경우 우선 함수의 반환형은 통상적인 의사 코드 형태로 삭제하고 함수라는 것을 나타내기 위해 “FUNCTION”으로 표시하였다. 함수 이름의 경우에는 그대로 남겨두고, 이후 배열에 저장해 프로그램소스에서 함수의 사용을 판별하는지 검사하는데 사용하였고 매개변수의 경우는 함수의 입력값이므로 “INPUT” 키워드를 통해 명확히 나타내주었다.

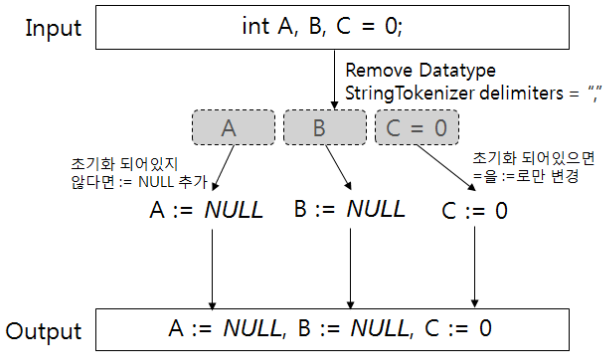
함수의 제일 처음과 마지막 중괄호는 각각 ***START, END***로 표시하여 함수의 시작과 끝을 표시하였다. *Scanner Class*의 *NextLine*함수로 읽어 온 문자열이 함수의 return문인 경우 *return*할 값이 있다면 그대로 적어주되 “OUTPUT”으로 표시하여 주었다. 만약 함수의 형태가 반환하는 값이 없는 *void*형이지만 사용자가 *‘return;’*으로 선언을 해주었다면 이는 사용자가 의도적으로 표시해 주었다고 판단하여 그대로 표시하도록 하였다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.



[그림 1] 소스코드 함수 선언문 및 종료문 변환 과정

3.1.2 변수 선언문

소스코드에서 읽어 온 문자열이 변수형을 포함하면서 세미콜론(;)을 포함하면 변수의 선언 부분으로 식별하였다. 해당 문자열에서 변수형은 지우고 *StringTokenizer Class*를 사용하여 세미콜론(;)과 콤마(,)로 구분하여 토큰을 생성하였다. 생성된 토큰에서 만약 '=' 기호가 포함되어 있다면 초기화된 변수로 '=' 기호를 ':='로 변환하고 '=' 기호가 없다면 ':=NULL'이라고 변경하여 초기화 되지 않았음을 표시해 주었다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.



[그림 2] 소스코드 변수 선언문 변환 과정

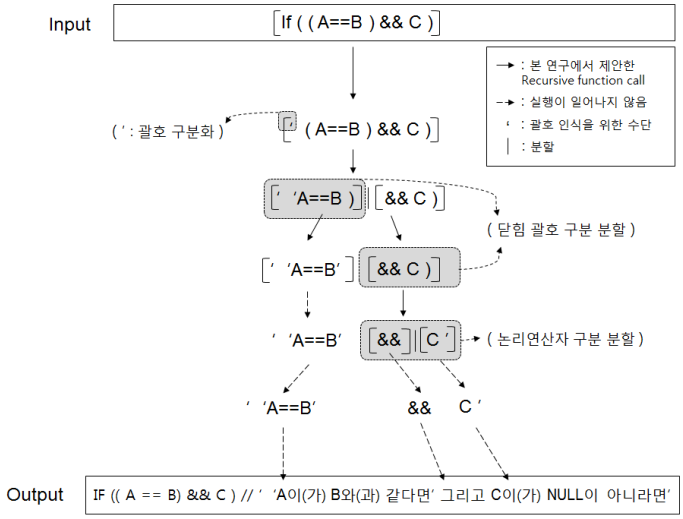
3.1.3 조건문

if, else if, else문과 같은 조건문의 경우 괄호 안에 두개 이상의 조건이 있을 경우 변환 알고리즘과 조건들을 한글 주석으로 생성하는 방법이 주요 이슈였다. 기존 연구들에서는 조건의 문자열을 트리 형태로 분할하여 저장한 뒤 영어로 변환해주는 형식을 취하였지만 [14,15], 다음 그림과 같이 괄호가 여러 개인 복잡한 조건식에 대해서는 실제 판별식이 명확히 제시되어 있지 않다.

```
if (( A == B ) && C )
if ((A != B) || ( C == D ))
```

[그림 3] 조건문 내의 괄호 조건 예

이러한 점에 대해 본 연구에서는 재귀함수 형태를 활용하여 해결할 수 있는 알고리즘을 개발하였다. 즉, 만일 입력된 조건문자열이 소괄호로 시작하면 소괄호를 작은 따옴표로 변환하고 다시 재귀 함수를 호출한다. 이후 *StringTokenizer Class*를 사용하여 '&&'이나 '||', ')'으로 token을 생성하고 다시 재귀 함수를 호출한다. 입력받은 문자열이 더 이상 토큰으로 구별되지 않으면 '=='이나 '>='과 같은 조건에 맞게 한글로 변환하도록 하였다. 만일 조건이 없이 변수만 들어올 경우 'C(이)가 *NULL*이 아닐 경우'와 같이 변환해주었다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.

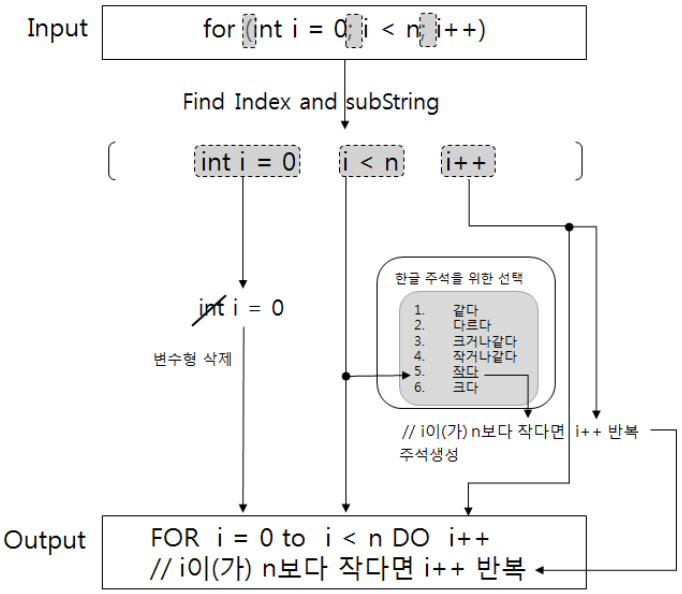


[그림 4] 소스코드 조건문 변환과정

if와 else는 대문자로 변환하고 조건 뒤에 *THEN*을 붙여 실행하는 것을 표시해주었다. 또한 중괄호를 *BEGIN, END*로 구분하여 시작과 끝을 표시하였다.

3.1.4 반복문

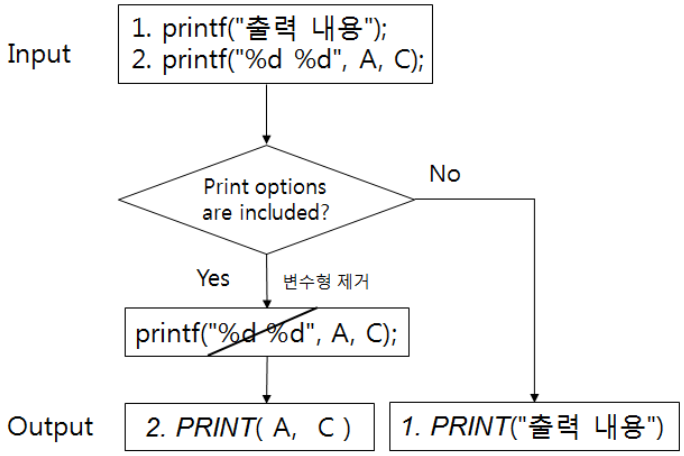
for, while, do while 문과 같은 반복문의 경우 괄호안에 조건이 있는 형태이기 때문에 기본적으로 조건문과 유사한 형태를 가진다. 따라서 조건문에서 수행한 것과 같이 복잡한 조건들에 대해서는 재귀함수 형태를 이용해서 반복적으로 식별하고 의사코드 형태로 변환하였다. 한글 주석 생성시에는 등호나 부등호를 식별하고 switch문을 활용하여 선택 형태로 구현하여 모든 조건에 대해 커버할 수 있도록 구현하였다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.



[그림 5] 소스코드 반복문 변환과정

3.1.5 출력문

함수를 사용하여 읽어온 문자열이 대표적인 출력구문인 *printf*함수라면 우선 2가지 상황을 식별하여 그에 맞게 변환과정을 수행해야 한다. 첫째, printf 함수 내에 문자열만 출력하도록 하는 경우와 둘째, printf 함수 내에 출력옵션(%d, %c 등)이 있어 변수의 값을 출력해주는 경우이다. 첫번째 경우에는 함수 내의 출력을 하고자 하는 문자열을 그대로 나타내었고, 두번째 경우에는 출력옵션은 제외하고 실제 출력 변수명을 표시하도록 하였다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.



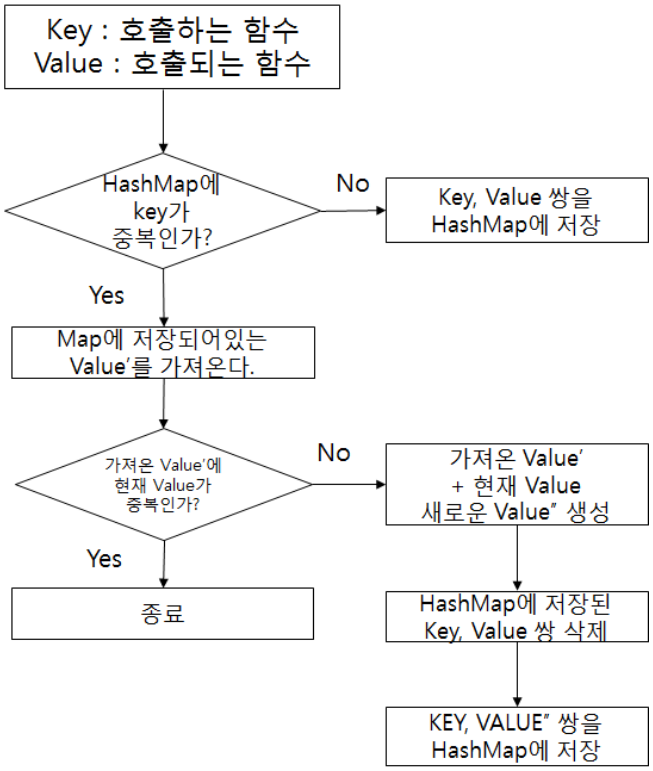
[그림 6] 소스코드 출력문 변환과정

3.1.6 추가 기능 (function call analysis)

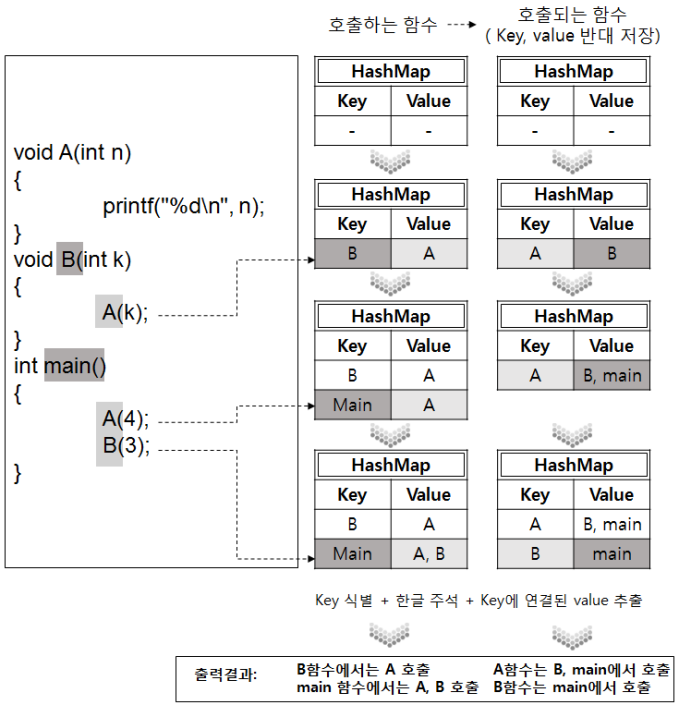
만일 소스코드 사이즈가 상당할 경우 의사코드들 간의 전체적인 관계를 보다 명확히 이해할 수 있도록 도움을 주기 위해, 본 연구에서는 함수간의 함수 호출(function call) 상관관계를 명시적으로 식별하여 제공해줄 수 있도록 하였다.

이를 위해 앞서 함수의 선언문에서 현재 함수의 이름을 변수에 할당하여 이를 함수를 호출하려는 함수로 보았다. 이 변수는 새로운 함수의 선언문이 나오면 변경이 된다. 그 이후 HashMap Class를 이용하여 Key에는 함수를 호출하려는 함수(현재 정의 중인 함수)를 Value에는 호출당하는 함수를 저장한다. 단, HashMap에서 Key가 중복인지 판단하여 중복이 아니라면 Key, Value쌍을 저장하고 중복인 경우 Key에 대한 Value값을 가져와 현재 호출당하는 함수가 이미 포함되어 있는지 확인한다. 이미 포함되어 있다면 과정을 종료하고 만일 그렇지 않다면 현재 호출당하는 함수를 Value에 추가해 새로운 Key, Value쌍으로 저장한다 (e.g., Key, Value1 Value2).

반대로 생각하여 Value와 Key의 값을 바꾸어 출력하면 각 함수가 어느 함수에서 사용 되었는지도 파악할 수 있다. 또한 함수 선언문 내에서 함수 호출이 있을 경우 현재의 함수와 같다면 재귀함수호출, 다르다면 함수 호출이라고 주석으로 명시해주었다.



[그림 7] 함수 호출 간계 분석 알고리즘



[그림 8] 함수 호출 관계 의사코드 도출 과정 예

위 그림의 예제를 보면 B에서는 A함수를 호출하고 main에서는 A와 B함수를 호출하고 있다. 각 호출 정보는 HashMap을 통해 순서대로 Key, value로 저장하고 서로간의 함수호출관계를 최종적으로 출력해줄 수 있도록 하였다.

```
void printResult(bool flag, int num)
{
        if (flag == true)
        {
                printf("소수가 아닙니다.\n");
        }
        else
        {
                printf("%d는 소수입니다.\n",num);
        }
}
void isPrimeNumber(int n)
{
        bool flag = false;
        for (int i = 2; i*i <= n; i++) {
                if (n%i == 0)
                {
                        flag = true;
                        break;
                }
        }
        printResult(flag, n);
}
```

```
FUNCTION printResult
INPUT := (flag, num)
START
        IF (flag == true) THEN          // 'flag이(가) true와(과) 같다면'
        BEGIN
                PRINT("소수가 아닙니다.\n")
        END
        ELSE THEN
        BEGIN
                PRINT(num)
        END
EXIT

FUNCTION isPrimeNumber
INPUT := (n)
START
        flag := false
        FOR i = 2 to i*i <= n DO i++// i*i이(가) n보다 작거나 같다면 i++ 반복
        BEGIN
                IF (n%i == 0) THEN  // 'n%i이(가) 0와(과) 같다면'
                BEGIN
                        flag = true
                        break
                END
        END
        printResult(flag, n)            // printResult 함수 호출
EXIT

isPrimeNumber함수에서는 printResult 호출
printResult함수는 isPrimeNumber에서 호출
```

[그림 9] 의사코드 생성 예제

3.1.7 수행 결과 예

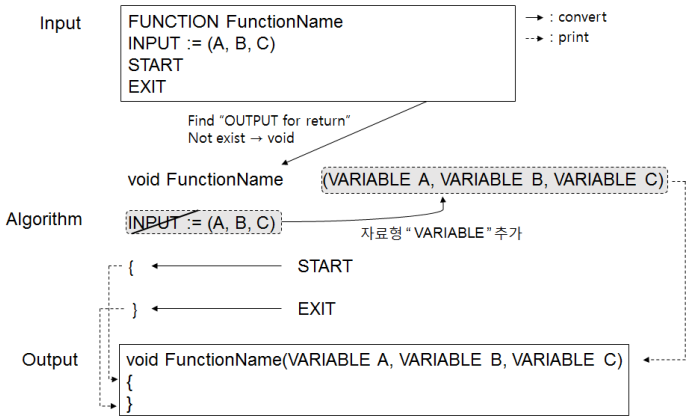
입력받은 숫자가 소수(Prime number)인지 아닌지 판별하는 예제 소스코드를 대상으로 제안된 기법을 통한 전체 의사코드 생성 결과를 나타내면 [그림 9]와 같다.

3.2 의사코드에서 소스코드로의 변환 기법

의사코드에서 소스코드로 변환하기 위해서는 기존 생성된 의사코드의 결과를 읽어 *StringBuffer Class*와 클래스 내의 *append*함수를 사용하여 하나의 함수를 한번에 출력하도록 하였다. 또한 주석은 해당 위치에 보존하도록 하였다.

3.2.1 함수 선언문 및 종료문

본 연구에서 활용하는 의사코드에서는 함수 선언인 경우 반환형이나 데이터타입이 생략되어 어떤 형태인지 미리 알기 어려워 정확한 타입으로 변환하기가 어려운 상황이 발생할 수 있다 (단, void형의 경우 *return*의 형태나 유무로 알 수 있다. *return*문이 *'return;'* 이거나 없을 경우 *void*형이다). 반환값이 있어 타입을 알 수 없는 경우 *'VARIABLE'*이라고 두어 사용자가 직접 변환할 수 있도록 하였다. 또한 매개변수의 경우에도 타입에 대한 정보가 없어 마찬가지로 *'VARIABLE'*로 두어 사용자가 변환할 수 있도록 하였다. 이는 함수 안의 변수에도 해당한다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.



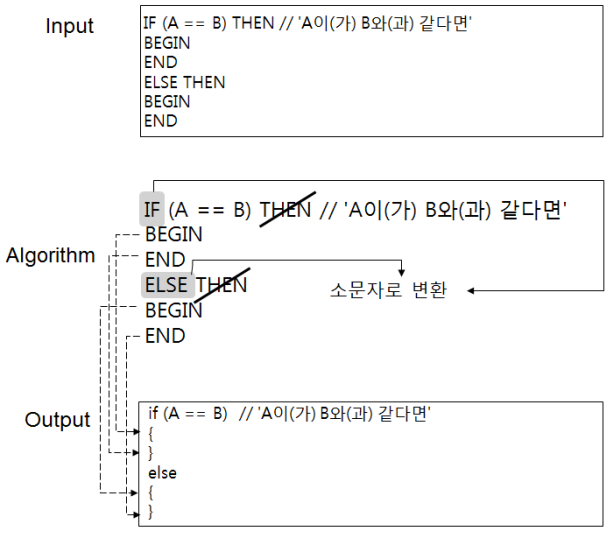
[그림 10] 의사코드 함수 선언문 및 종료문 변환과정

3.2.2 변수 선언문

본 연구의 의사코드의 경우 변수의 데이터 타입은 생략된 형태로 표현되기 때문에 변수 선언문에 대해서는 임의로 *'VARIABLE'*을 표현해준다.

3.2.3 조건문

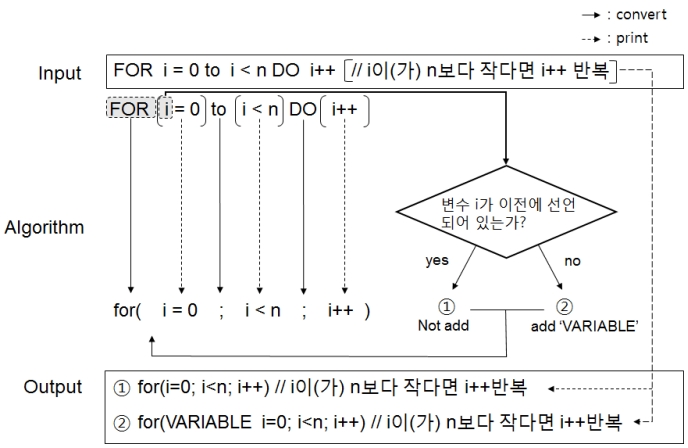
조건문의 경우 주석은 그대로 유지하되, 의사코드에서 대문자로 표기해주었던 IF, THEN, ELSE를 식별하여 소문자로 변경하고 BEGIN, END는 소괄호로 바꾸어 주었다.



[그림 11] 의사코드 조건문 변환과정

3.2.4 반복문

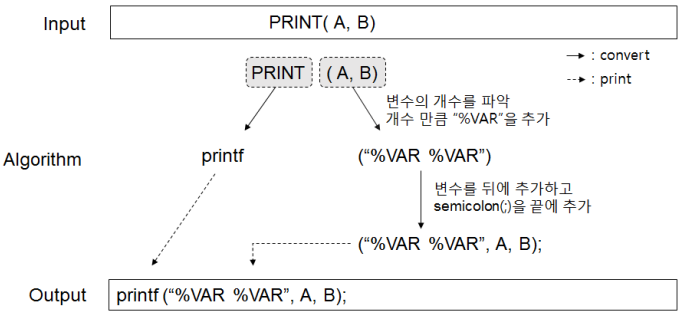
반복문인 for, while, do while문 중 *for*문의 경우, *for*문 안에서 참조변수를 직접선언하여 사용하는 경우와 *for*문 이전에 참조변수를 선언하여 두는 경우로 나누어질 수 있다. 이에 대해 본 연구에서는 참조변수의 이름을 검색하여 함수 안에서 해당 변수를 먼저 선언하지 않았으면 *'VARIABLE'*을 붙여 사용자가 변환할 수 있도록 하였다. 반복문의 경우에도 주석은 그대로 보존하였다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.



[그림 12] 의사코드 반복문 변환과정

3.2.5 출력문

출력문인 *printf*의 경우에서도 변수의 값을 출력하는 경우라면 변수의 형태에 맞게 형태를 지정해 주어야한다. 그러나 의사 코드(Pseudo Code)에서는 변수에 대한 정보를 알 수 없으므로 *printf*함수 안의 변수의 개수만큼 *'%VAR'*을 두어 사용자가 변경하도록 하였다. 다만 문자열만 있는 경우라면 그대로 출력하도록 하였다. 전체적인 실행 구성을 요약하면 다음 그림과 같다.



[그림 13] 의사코드 출력문 변환과정

3.2.6 수행 결과 예

의사코드를 대상으로 본 연구의 기법을 통한 소스코드 생성 결과는 [그림 14]와 같다.

4. 실험설계 및 결과

4.1 실험설계

본 연구 내용의 평가를 위해 소프트웨어 관련 금융기관 및 웹서비스 제공 업체 등 국내 5개의 소프트웨어 업체에서 11명의 개발자를 대상으로 설문을 진행하였다. 총 9개의 객관식 질문(1에서 5점 선택)과 1개의 주관식 질문으로 구성되어 있으며 요약하면 다음과 같다.

- 기법 알고리즘을 적용한 도구 완성도
- 소스코드에서 의사코드 변환 결과의 가독성
- 소스코드에서 의사코드 변환 결과의 부분적 활용가능성
- 의사코드 주석처리 정도
- 의사코드에서 소스코드 변환 결과의 완성도
- 의사코드에서 소스코드 변환 결과의 부분적 활용가능성
- 실제 활용 의사
- 개선 방향 및 기타의견

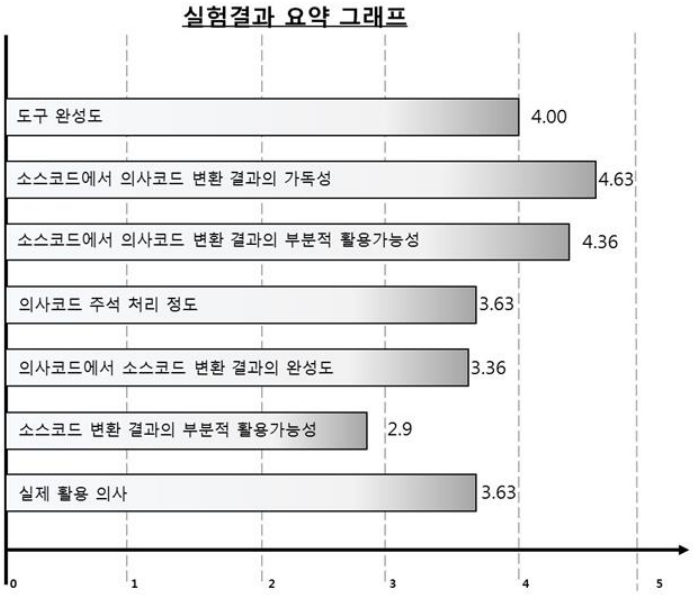
```
FUNCTION printResult
INPUT := (flag, num)
START
        IF (flag == true) THEN // 'flag이(가) true와(과) 같다면'
        BEGIN
                PRINT("소수가 아닙니다.\n")
        END
        ELSE THEN
        BEGIN
                PRINT(num)
        END
EXIT
FUNCTION isPrimeNumber
INPUT := (n)
START
        flag := false
        FOR i = 2 to i*i <= n DO i++// i*i이(가) n보다 작거나 같다면 i++ 반복
        BEGIN
                IF (n%i == 0) THEN  // 'n%i이(가) 0와(과) 같다면'
                BEGIN
                        flag = true
                        break
                END
        END
        printResult(flag, n)            // printResult 함수 호출
EXIT
```

```
void printResult(VARIABLE flag, VARIABLE num)
{
        if (flag == true) // 'flag이(가) true와(과) 같다면'
        {
                printf("소수가 아닙니다.\n");
        }
        else
        {
                printf( num);
        }
}
void isPrimeNumber(VARIABLE n)
{
        VARIABLE flag = 0;
        for(VARIABLE i=2; i*i<=n;  i++) //i*i이(가) n보다 작거나 같다면 i++ 반복
        {
                if (n%i == 0)    // 'n%i이(가) 0와(과) 같다면'
                {
                        flag = true;
                        break;
                }
        }
        printResult(flag, n);           // printResult 함수 호출;
}
```

[그림 14] 소스코드 생성 예제

4.2 실험결과 및 분석

전체적인 실험결과 및 분석은 다음과 같다.



[그림 15] 실험결과

“프로그램 완성도” (4.00)측면에서는 C언어에 구문들에 대한 해석 및 분석(parsing)이 확실하게 이루어져 변환이 가능하기 때문에 전반적으로 만족하는 평가를 보여주었다. 뿐만 아니라 “소스코드에서 의사코드 변환결과의 가독성” (4.63)항목에 대해서는 상당히 우수한 평가를 받았는데, 이는 앞서 소개한 구체적인 변환 알고리즘에 기반하여 복잡한 조건식들까지도 오류없이 변환가능하였으며 영어 외에 한글 주석을 추가로 지원하기 때문으로 분석되었다. 또한 “소스코드에서 의사코드 변환 결과의 부분적 활용가능성” (4.36)에서도 높은 평가를 받았는데, 생성된 의사코드를 활용하면 전체적인 프로그램 이해도를 높일 수 있는데 도움을 줄 수 있고, 기존 소프트웨어 시스템 유지보수 수행시 주요 핵심 부분들에 대해 부분적 의사코드를 도출하여 서로간의 커뮤니케이션 및 이해도 향상 용도로도 활용 가능할 수 있다고 평가받았다. “의사코드 주석 처리 정도” (3.63)에 대해서도 3점이상의 점수로서 보통이상으로 평가받았다.

반면, “의사코드에서 소스코드 변환 결과의 완성도” (3.36)측면에서는 상대적으로 점수가 낮았는데, 의사코드에서 확보할 수 있는 정보의 한계로 인해 소스코드로 변환시 즉각적으로 수행할 수 있는 완벽한 소스코드가 제공되지 않을 수 있기 때문으로 평가된다. 단, 의사코드 작성자에 따라 소스코드 변환을 위한 변수의 데이터 타입 등의 충분한 정보를 제공한다면 완벽한 소스코드로 변환시킬 수 있다.

“의사코드에서 소스코드 변환 결과의 부분적 활용가능성” (2.9)항목에서 가장 낮은 점수를 보여주었는데, 의사코드를 소스코드를 자동변환하게

되면 반복적인 프로그래밍을 개선하고 주석 작성을 지원해줄 수 있지만 실제로는 의사코드로 개발자들끼리 의견논의가 이루어진 후 완벽한 소스코드를 직접 만드는 것이 발생할 수 있는 오류를 미연에 방지하는데 도움이 될 수 있기 때문에 본 항목에 대해서는 가장 낮은 점수를 획득하였다.

마지막 “실제 활용 의사” (3.63)의 경우 평가자들의 의견을 종합해본 결과, 소프트웨어 유지보수시 실제 현업에서 쉽게 활용할 수 있는 도구나 방법들이 거의 없기 때문에 기존 소스코드를 한줄씩 읽어가며 이해해야 하는 경우가 자주 발생하므로 본 논문에서 제시한 연구는 소프트웨어 이해도 향상에 직접적인 도움을 줄 수 있는 가능성이 충분한 것으로 분석되었다.

5. 결론 및 향후 연구

본 논문에서는 소프트웨어 개발 생명 주기 중 소프트웨어 유지보수단계에서 소프트웨어 설계자 또는 개발자들의 이해도 향상을 지원할 수 있는 한글기반 의사코드 및 소스코드 양방향 자동 생성 기법 알고리즘에 대해 연구 제안하였다. 기존 연구들에서는 제시되어 있지 않았던 복잡한 조건식들에 대해서도 변환가능할 수 있도록 하기 위해 재귀함수를 활용한 신규 알고리즘을 개발하였고 영어 및 한글을 지원할 수 있도록 하였으며 효율향상을 위해 자동화를 지원하였다. 연구된 기법 및 도구는 실제 산업체에서 근무하고 있는 개발자를 대상으로 평가의견을 수집하였고 소프트웨어 이해도 향상 및 활용성 측면에서 충분한 가능성을 보여주었다.

본 논문의 연구를 기반으로 향후에는 의사코드 외에도 소프트웨어 전체적인 구조에 대한 시각화(visualization) 연구를 수행할 예정이다. 그리고, 본 논문의 실험을 수행하면서, 현업에서는 소프트웨어 유지보수 수행시 소스코드 한줄씩 읽어 분석하는 것은 매우 수동적이기 때문에 해당 소프트웨어에 대한 테스트케이스(testcase)를 살펴보고 전체 논리를 이해하려는 경향이 많아지고 있다는 피드백을 반영하여 이를 효율적으로 지원하기 위한 기법 및 도구를 연구개발 하고자 한다.

Acknowledgement

이 연구는 2017년도 영남대학교 학술연구조성비에 의한 것임.

참고 문헌

[1] D. Parnas, Software aging, The 16th International Conference on Software Engineering, p.279-287. IEEE Computer Society Press, 1994.

[2] M. Fowler and K. Beck, Refactoring: Improving the design of existing code, Addison-Wesley Professional, 1999.

[3] R. C. Martin, Agile Software Development: Principles, Patterns and Practices, Prentice Hall, 2003.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

[5] A.J. Riel, Object-Oriented Design Heuristics, Addison-Wesley, 1999.

[6] R. Naseem, O. Maqbool, and S. Muhammad, Cooperative clustering for software modularization, Journal of Systems and Software, 86, 8, p.2045-2062, 2013.

[7] P. Andritsos and V. Tzerpos, Information-theoretic software clustering, IEEE Transaction on Software Engineering, 31, 2, p.150-165, 2005.

[8] G. Bavota, A. Lucia, A. Marcus, and R. Oliveto, Using structural and semantic measures to improve software modularization, Empirical Software Engineering, 18, 5, p.1-32, 2012.

[9] A. Kuhn, S. Ducasse, and T. Grba, Semantic clustering: Identifying topics in source code, Information and Software Technology, 49, 3, p.230 - 243, 2007.

[10] K. Praditwong, M. Harman, X. Yao, Software module clustering as a multi-objective search problem, IEEE Transactions on Software Engineering, 37, 2, p.264-282, 2011.

[11] T. E. Bailey and Kris Lundgaard, Program Design with Pseudocode, Brooks/Cole Publishing Company, 1989.

[12] Lambert M. Surhone, Miriam T. Timpledon, and Susan F. Marseken, Pseudocode: Computer Programming, Algorithm, Programming Language, Declaration, Subroutine, Skeleton, Betascript Publishing, 2010.

[13] Jouni Smed and Harri Hakonen, Algorithms and Networking for Computer Games, Wiley, 2006.

[14] Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura, Learning to generate pseudo-code from source code using statistical machine translation, IEEE/ACM International Conference on Automated Software Engineering (ASE), p.574-584, 2015.

[15] Hiroyuki Fudaba, Yusuke Oda, Koichi Akabe, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura, Pseudogen: A Tool to Automatically Generate Pseudo-code from Source Code, Tool Demonstrations session in IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.

모바일 아키텍처를 고려한 애플리케이션 보안 취약점 진단 방안

김명근O, 최은만
동국대학교 컴퓨터공학과
xitcsk@gmail.com, emchoi@dgu.ac.kr

Assessment Method for Application Security Vulnerability Considering Mobile Architecture

Myeonggeun KimO, Eun Man Choi
Dongguk University

요 약

일반적으로 모바일 보안 취약점 진단하는 방법은 취약점의 특성을 분류하여 체크리스트를 만들고 이를 기반으로 하여 서로 다른 도구와 방법을 동원하여 진단하므로 실제 불필요한 작업을 반복한다는 단점이 존재한다. 이 논문은 모바일 앱의 구조와 특성을 고려하여 보안 취약점 진단 방법을 단계적으로 분석하는 방안을 제시한다. 본 논문에서는 모바일 취약점을 진단할 때 효율적인 진단 작업이 되기 위하여 진단자의 입장에서 취약점을 분류하고 실제 진단 순서에 따라 클라이언트, 전송 구간, 서버로 구분하여 취약점을 재분류하고 반복되는 작업을 최소화한다. 또한 아키텍처 상의 각 구간에서 모바일 보안 취약점을 진단하는 방법을 제시한다.

1. 서론

모바일은 2007에 출시된 아이폰을 기점으로 스마트폰으로 바뀌면서 보급이 빠르게 늘어나고 모바일이 대세를 이루고 있다. 2010년부터는 전 세계적으로 스마트폰의 출하 대수가 PC를 넘어서기 시작하였고 기존 웹 기반의 인터넷 서비스에서 모바일 앱 중심의 사용자 개개인에 맞춤화된 개인화 서비스가 증가하였다. 모바일 앱은 웹에 접속하지 않아도 스마트폰에 설치한 앱을 통해 사용자에게 보다 빠른 서비스를 제공한다. 이에 따라 모바일이 웹 시장을 넘어서 앱의 이용이 가속화될 전망이 높다[1].

이에 따라 모바일 보안 취약점에 대한 문제도 크게 대두되고 있는데, 초기에는 스미싱과 같은 공격을 통해 사용자의 스마트폰에 저장된 중요 정보를 빼돌리거나 소액 결제 사기 등이 많았다. 하지만 최근에는 악성코드를 이용한 정보유출이나 원격제어 등 모바일 취약점을 이용하여 고도화된 모바일 보안 위협이 발생하고 있다.

특히 모바일 앱 보안의 경우 국내 모바일 생태계의 82%를 차지하고 있는 안드로이드의 경우 6.0 버전의 보급률은 단 1%에 불과하며, 약 70%를 차지하고 있는 4.4 및 그 이하 버전의 사용자가 대부분이다. 즉, 보안에 취약한 모바일 운영체제를 사용하고 있는 사용자가 대부분이다.

이에 본 논문은 기존 안드로이드 모바일 앱에 대한 보안 취약점 진단 방법들을 분석하여 보안 진단 작업을 효율화 하기 위하여 진단자 입장에서 모바일 앱의 구조와 특성을 고려한 보안 취약점 진단 방안을 제시한다. 기존 모바일 취약점 진단 가이드의 경우 취약점 자체의 특성에 따라 체크리스트 항목을 분류하여 진단한다. 때문에 실제 해당 체크리스트 기반으로 작업하게 되고 진단 작업에서 불필요하고 같은 작업이 반복된다. 이에 따라 본 논문에서는 기존 취약점을 진단하려는 시스템의 아키텍처 지점의 위치에 기반하여 클라이언트, 전송 구간, 서버로 구분하여 취약점을 재분류하고 해당 취약점을 분석하는 방안을 제안한다.

2. 기존 모바일 앱 보안 취약점 진단 가이드

현재 국내에서 사용하는 모바일 앱의 보안 취약점을 진단하는 방안으로는 행정자치부와 한국정보보호진흥원(KISA)에서 발행한 “모바일 대국민 전자정부서비스 앱 소스코드 검증체계”와 “모바일 대민서비스 보안취약점 점검 가이드”가 있으며 금융감독원에서 시행하는 “스마트폰 금융 안전대책 이행실태 점검” 등이 있다. 해외에서는 OWASP(Open Web Application Security Project)에서 발행한 “OWASP Mobile Top 10”이 있다[2].

2.1 모바일 앱 소스코드 검증체계

행정자치부와 한국정보보호진흥원에서 발행한 가이드로 행정기관 또는 공공기관에서 개발한 모바일 대국민 전자정부서비스 앱을 대상으로 소스코드 수준의

보안 취약점을 사전에 진단하고 제거하기 위한 방안이다. 모바일 앱 개발 시 잘못된 소스코드로 인한 보안 취약점과 잘못된 기능 구현으로 발생할 수 있는 보안 취약점으로 분류를 나누어 보안 취약점을 진단할 수 있는 방법을 제시한다.

표 1 모바일 대국민 전자정부서비스 앱 가이드 라인

소스코드 보안취약점	1. 입력데이터 검증 및 표현
	2. 보안기능
	3. 시간 및 상태
	4. 에러처리
	5. 코드오류
	6. 캡슐화
	7. API 오용
기능 보안취약점	FV-1 : 임의기능 존재 여부
	FV-2 : 최소 권한
	FV-3 : 입력 값 유효성
	FV-4 : 중요정보 관리
	FV-5 : 플랫폼 보안 모델
	FV-6 : 상용/공개용 모듈
	FV-7 : 공개영역 취약점
	FV-8 : 보안공통기반 적용의 적절성
	FV-9 : 기타

소스코드 보안취약점 항목의 경우 앱 개발 과정에서 잘못된 코드로 인해 발생하는 취약점을 7개 대분류로 나누고 세부항목을 다시 47개 소분류로 나누어 취약점을 진단하며, 기능 보안 항목의 경우 앱의 소스코드보다는 기능에 초점을 맞추어 취약점을 9개의 대분류로 나누고 다시 25개로 나누어 세부 항목을 진단한다.

2.2 스마트폰 금융 안전대책

금융감독원에서 발행한 가이드로 스마트폰을 이용한 모바일 뱅킹 서비스를 제공하는 금융회사의 앱을 대상으로 모바일 앱에 대한 보안성을 검증하는 가이드이다. 모바일 금융 보안대책, 앱 위변조 방지대책, 기타로 3가지 영역을 분류하여 세부 항목 16개에 해당하는 모바일 보안 취약점 체크리스트를 제공한다[3].

표 2 스마트폰 금융 안전대책 이행실태 체크리스트

모바일 금융 보안대책	백신 프로그램 적용
	입력정보 보호대책 적용 여부
	금융정보 종단간 암호화 적용 여부
앱 위변조 방지대책	폰 임의개조 탐지 및 차단
	앱 무결성 검증 기술 적용
	코드 모듈 보호
기타	앱 취약점 점검
	앱 위변조 로그 기록
	멀티로그인 차단
	스마트폰에 중요 정보 저장 금지
	스마트폰 금융 거래기록 정보보관

모바일 뱅킹 서비스를 제공하는 금융 앱의 경우 필수로 진단해야되는 항목으로 구성된 체크리스트로 금융 앱의 보안성을 증명하기 위한 요소들로 구성되었다. 하지만 금융 앱의 보안성에 초점을 맞춘 만큼 일반적인 애플리케이션의 보안성을 완벽히 증명하기엔 부족한 점이 존재한다.

2.3 OWASP Mobile Top 10

OWASP에서는 3년을 주기로 웹과 모바일에 관한 10가지 중요 취약점을 선정하여 해당 취약점에 대한 가이드 라인을 발표하고 있는데, 모바일 보안 가이드는 현재 2016년 가이드까지 발표되었다. 진단 가이드 외에도 앱을 개발할 때 해당 취약점을 피해 안전하게 코딩할 수 있는 시큐어코딩 가이드 라인도 제공하고 있다. OWASP 탑 10에서 다루는 취약점은 다음과 같다[4].

표 3 OWASP Mobile Top 10(2016)

M1. 부적절한 플랫폼 사용
M2. 안전하지 않은 데이터 스토리지
M3. 안전하지 않은 커뮤니케이션
M4. 안전하지 않은 인증
M5. 불충분한 암호화
M6. 안전하지 않은 인증
M7. 클라이언트 코드 품질
M8. 코드 변조
M9. 역공학
M10. 불필요한 기능

OWASP Top 모바일 10은 모바일 취약점 가이드의 표준이라고 할 수 있는 가이드로 가장 최신 이슈를 잘 반영한 가이드로 볼 수 있다. 가이드에서 실제 제공하는 체크리스트의 경우 91개의 취약점 항목을 갖고 있다. 많은 취약점 항목을 갖고 있지만 취약점의 특성이나 환경적인 요소를 고려하지 않고 단순하게 모든 취약점을 나열하여 표시되어 실제 진단 시 반복된 작업 또는 불필요한 작업을 반복하게 된다.

3. 보안 취약점 진단 가이드 도출 과정

2장에서 살펴본 모바일 보안 취약점 진단 가이드들의 경우 모바일 취약점의 특성을 기준으로 분류를 나누어

체크리스트를 만들었기 때문에, 해당 가이드를 기준으로 실제 보안 담당자가 모바일 취약점을 진단할 때 비용과 시간적인 면에서 효율적이지 않다. 또한 국내에서 발행한 취약점 가이드들의 경우 공공기관이나 금융 서비스라는 한정된 앱을 대상으로 만들어졌기 때문에 그 점검 기준이 포괄적이지 못하다.

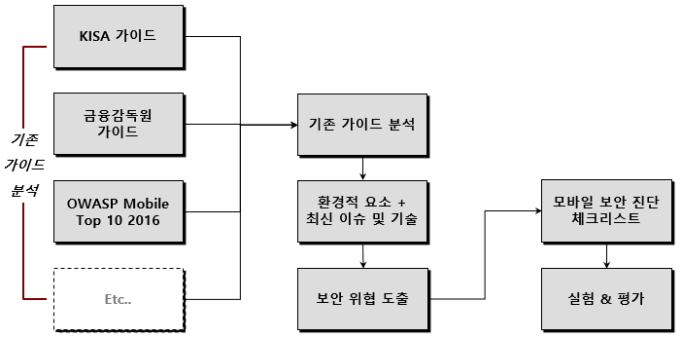


그림 1 보안 취약점 진단 가이드 라인 도출 과정

또한 기존의 가이드의 경우 취약점의 특성에 따라 취약점 항목을 분류하여 진단하기 때문에 취약점 진단 시 2가지 단점이 존재하는데, 첫 번째는 실제 취약점 진단 순서를 고려하지 않기 때문에 불필요한 작업이 반복된다는 점이고 두 번째는 취약점 자체 특성에 따라 분류하기 때문에 앱에서 발생한 취약점을 엮어서 실제 모바일 앱에 미치는 영향을 진단자 또는 보안 관리자가 판단하기 어렵다는 점이다.

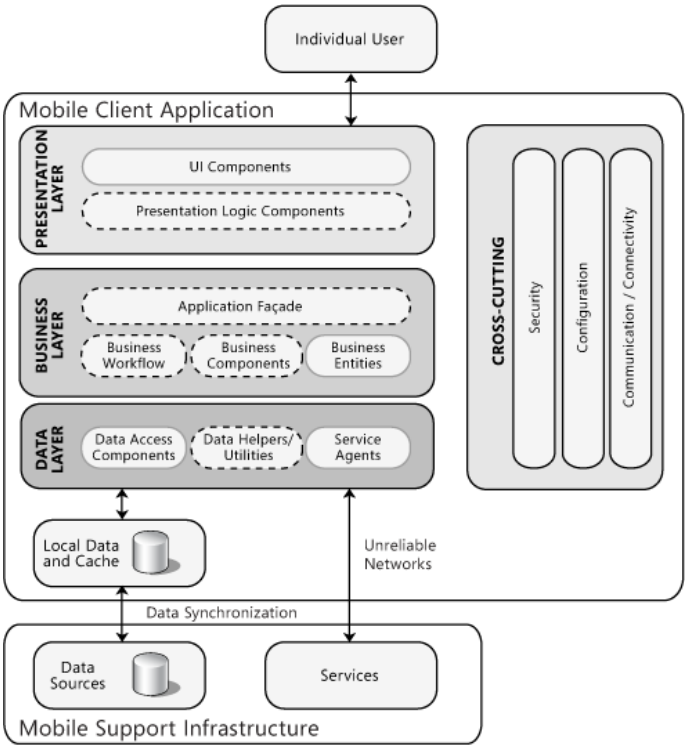


그림 2 모바일 애플리케이션 아키텍처

이러한 문제점 해결을 위해 본 논문에서는 모바일 아키텍처를 고려한 보안 취약점 진단 방안을 제안한다. 기존의 모바일 앱 보안 취약점 항목을 모바일 앱 아키텍처에 따라 클라이언트와 전송 구간, 서버로 구분하여 취약점을 재분류하고 앱 진단 순서에 따라 취약점을 정렬했다.

3.1 클라이언트에서 진단하는 보안 취약점

클라이언트 단계에서는 모바일 앱의 역공학, 코드 변조, 리팩토링 여부 등을 진단한 후 디컴파일한 코드를 통해 정적 분석을 수행한다. 정적 분석에서는 권한, 앱 컴포넌트, 앱에서 사용된 API 버전 등을 진단하고 소스코드 단계에서 확인할 수 있는 문제점들을 체크한다. 그 후 정적 분석 결과를 토대로 동적 분석을 수행하는데 모바일이라는 환경적 특성으로 인해 발생하는 취약점들을 주로 진단한다. 예를 들어 오프라인 인증을 통한 우회나 WebView, 런터임 분석을 통한 동적 디버깅 등을 진단한다.

3.2 전송 단계에서 진단하는 보안 취약점

전송 단계에서는 프록시 도구나 네트워크 트래픽 진단 도구를 사용하여 안전하지 않은 프로토콜의 사용 여부를 진단하고 그 과정에서 발생하는 SSL/TLS 통신의 암호화 강도를 진단한다. 또한 통신 과정에서 인증서의 유효성이나 프록시 도구로 CA 인증서를 자체 서명할 수 있는지를 진단한다. 마지막으로 디바이스에 대한 UDID를 통해 공격자가 확인 가능한 식별자 노출 등을 진단한다.

3.3 서버 단계에서 진단하는 보안 취약점

마지막으로 서버 단계에서는 서버 측 URL 및 IP에서 열린 포트를 확인하고 백엔드 서버에서 사용하는 기본 자격 증명에 대해 진단한다. 또한 입력 값에 대한 유효성이나 세션, 쿠키, 토큰 값에 대한 보안성 등을 주로 진단한다.

4. 모바일 취약점 진단 방법

일반적으로 취약점 진단자 입장에서 보안 진단을 하는 방법에는 크게 3가지가 존재하는데, 먼저 정보를 제공받지 않고 작업을 수행하는 블랙 박스 테스팅[5]과 애플리케이션에 대한 소스코드를 알고 있는 상태에서 진단하는 화이트 박스 테스팅[6], 그리고 일정량의 정보만 제공받은 상태에서 역공학과 같은 방법을 통해 진단하는 그레이 박스 테스팅이 존재한다.

모바일의 경우 일반적으로 취약점 진단자 입장에서는 그레이 박스 테스팅에 해당하는 진단을 수행할 때가

많은데, 안드로이드 앱의 경우 자바 기반으로 만들어졌기 때문에 난독화와 같은 보안 기술이 적용되어 있지 않다면, iOS에 비해 비교적 진단하기 쉬운 편이다.

4.1 정적 분석 및 동적 분석

정적 분석은 실제 앱을 구동하지 않고 APK로부터 소스코드를 추출하여 분석하는 방법으로 동적 분석에 비해 환경적인면에서 유리하지만 취약점을 잘못 탐색할 위험이 높다.

안드로이드 진단 시 APK 파일을 추출하여 클라이언트에 대한 보안 진단을 수행한다. APK 파일은 ZIP 파일과 유사한 형태를 지니며, 위 그림 2의 디컴파일 과정을 통해 자바 코드를 추출할 수 있다. 디컴파일을 통해 추출한 코드는 매니페스트 파일로부터 권한, 앱의 컴포넌트, 버전을 분석할 수 있으며, 자바 및 달빅 코드에서 클래스 간의 구문 분석을 수행할 수 있다. 앱이 NDK로 개발되었거나 자바 소스코드에 난독화 기술이 적용되어 있지 않다면, 디컴파일을 통한 소스코드 정적 분석을 통해 대부분의 취약점을 진단할 수 있지만 그렇지 않을 경우엔 동적 분석을 통해 취약점을 진단해야 한다.

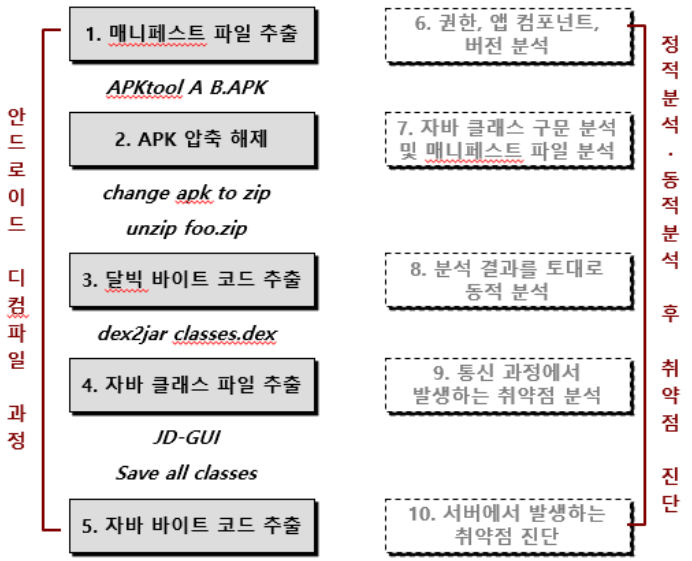


그림 3 모바일 앱 분석 단계별 수행하는 보안 진단

동적 분석은 실제 모바일 앱을 단말기 또는 가상머신에 설치하여 분석하는 방법으로 정적 분석에 비해 많은 시간과 하드웨어 자원을 필요로 한다. 또한 실제 모바일 앱을 구동하며 취약점을 분석하기 때문에 정적 분석에 비해 정확하게 취약점을 판별할 수 있다. 하지만 모든 영역을 탐지할 수 있는 테스트 케이스가 있지 않은 이상 정적 분석에 비해 상대적으로 취약점을 탐색하지 못할 위험이 높은 편이다.

가상머신을 사용하는 경우 다양한 디바이스 환경을 테스트할 수 있다. 또한 버전에 따라 발생할 수 있는 취약점이 다르기 때문에 일반적으로 보안 진단 시 가상머신을 많이 이용하는 편이다.

5. 결론

본 논문은 모바일의 아키텍처를 고려하여 모바일 앱에 대한 보안 취약점을 보다 효율적으로 진단하는 방안을 제시했다. 국내에서 사용하는 모바일 진단 가이드의 경우 공공기관이나 특정 서비스에 한정하여 진단 가이드를 만들었기 때문에 보다 포괄적인 모바일 앱에 대한 보안성을 체크하기 힘들었다. 또한 모바일 앱의 특성보다 취약점 자체의 특징을 기준으로 체크리스트 가이드를 만들어 배포하였기 때문에, 실제 진단자 입장에서 비용 및 시간적으로 효율적인 진단이 불가능했다. 본 논문에서 제안하는 방법은 실제 보안 담당자 입장에서 모바일 앱을 진단할 때 각 단계별로 진단해야되는 취약점과 분석 방법을 제안함으로써 기존 방법에 비해 비용과 시간을 절감시키고 모바일 앱의 잠재적 위협 요소를 제거하여 보안성을 높일 수 있는 방안을 제시하였다.

참고문헌

[1] 김경곤, 김휘강, “안드로이드 모바일 앱 보안진단 방법에 대한 연구”, 석사학위논문, 2015.

[2] 허환석, 강성훈, 김승주, “안전한 안드로이드 어플리케이션 개발을 위한 구현 단계별 보안성 검증 방안 제시”, 정보처리학회논문지, pp. 445-460, 2013.

[3] 금융감독원, “스마트폰 금융안전대책 이행실태”, 2012.

[4] OWASP, “OWASP Mobile Top Ten 2016”, 2016.

[5] Prithvi Bisht, Timothy Hinrichs, Nazari Skrupsky, Radoslaw Bobrowicz, V.N. Venkatakrishnan, “NoTamper: automatic blackbox detection of parameter tampering opportunities in web applications”, Proceedings of the 17th ACM conference on Computer and communications security, pp. 607-618, 2010.

[6] Riyadh Mahmood, Naeem Esfahani, Thabet Kace, Nariman Mirzaei, Sam Malek, Angelos Stavrou, “A whitebox approach for automated security testing of Android applications on the cloud”, IEEE, Automation of Software Test (AST), 2012 7th International Workshop on, pp. 22-28, 2012.

지능형 지속위협 생애주기를 고려한 보안요구사항 명세 방법

김승준 ○ 이석원*
아주대학교 컴퓨터공학과 ○, 아주대학교 소프트웨어학과*
dregonc@ajou.ac.kr○, leesw@ajou.ac.kr*

Advanced Persistent Threat Life-Cycle based Security Requirements Specification method

Kim Seung-Jun○ Lee Seok-Won*
Ajou University Dept. of Computer engineering○, Ajou University Dept. of Software*

요 약

사회-기술적 시스템(Socio-technical System, STS)에 대한 지능형 지속위협(Advanced Persistent Threat, APT)공격이 심각한 문제로 대두되면서, 지능형 지속위협에 대한 관심이 높아지고 있다. 이에 여러 방법들이 연구되었지만 시스템 설계 단계에서 지능형 지속위협을 생애주기마다 보안요구사항을 추천해주는 연구는 이루어 지지 않았다. 본 연구에서는 지능형 지속위협 생애주기를 목표 모델링(Goal modeling) 기법 중 하나인 KAOS(Keep All Objectives Satisfied)를 이용하여 목표 기반으로 재구성 하고, 위협 구성요소 관계모델을 통해 지능형 지속위협을 명세하고, 시스템 설계단계에서 적절한 보안 요구사항을 추천해주는 프레임워크에서 지능형 지속위협을 표현하는 방법을 제안한다.

1. 서 론

지능형 지속위협(Advanced Persistent Threat, APT) 공격은 바이러스(Virus)나 웜(Worm)과 같은 기존의 공격 방식과는 다르게 지속적이면서 변화하는 특성을 가졌기 때문에 예방 및 대처가 어렵다. 특히, 지능형 지속위협 공격은 인간과 소프트웨어, 하드웨어가 서로 긴밀한 관계를 가지는 사회-기술적 시스템(STS, Socio-Technical System)을 [1] 목표로 하는 경우가 많다. 사회-기술적 시스템은 은행, 발전소와 같은 이기종 시스템들과 연결되어 필수적인 임무(Mission Critical)를 수행하기 때문에, 사회-기술적 시스템에 대한 보안 결함 이슈는 매우 심각한 결과를 초래한다. 대표적인 사례로, 2010년에 스턱스넷(Stuxnet) 감염으로 인한 이란 핵 발전소 장애를 통해 [2] 그 심각성을 알 수 있다. 2012년에 발표된 포네몬 인스티튜트(Ponemon Institute)의 조사에 따르면 지능형 지속위협 공격에 의한 피해 복구 비용은 발생 건수당 약 550만 달러가 넘는다 [3].

이러한 배경에서 보안을 향상시키는 방법은 다양하며, 그 중 한가지는 시스템의 설계 단계에서 적절한 보안 요구사항 도출하여 시스템에 구현하는 것이다. 김봉재의 연구 [4]는 사회-기술적 시스템에서 고려해야 할 여러 가지 요소들을 분석해 적절한 보안요구사항을 추천하는 추천 프레임워크를 제안한다. 그러나 위 연구에서 제안하는 추천 프레임워크에서는 지능형 지속위협에 대한 적절한 보안요구사항을 나타내지 못한다는 문제점이 있다. 지능형 지속위협이 생애주기(Life Cycle)를 가지고 있고, 생애주기 각 단계마다 다른 공격 목표(Malicious goal)를 가지고 있어 단계마다 다른 보안 요구사항이 요구되는데, 생애주기 단계마다 보안요구사항을 명세하기 위한 방법이 제시되어 있지 않기 때문이다.

본 논문에서는 지능형 지속위협 생애주기를 분석하고, 목표 기반 접근법 중에 하나인 KAOS(Keep All Objectives Satisfied)를 통해 공격의 공격 목표를 추출하여 이를 통해 지능형 지속위협 생애주기를 목표 기반으로 재구성 하고자 한다. 더 나아가, 재구성된 목표 기반 지능형 지속위협 생애주기의 각 단계별로 보안 요구사항과 위험 구성 요소들을 나타내고, 요소들간의 관계를 표현하는 방법을 제시하여 추천 프레임워크에서 지능형 지속위협에 적절한 보안 요구사항을 추천할 수 있도록 한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구 및 배경 기술을 소개하고, 3장에서는 지능형 지속위협을 추천 프레임워크에서 표현하기 위한 방법을 제안하며, 마지막 4장에서 결론 및 향후 연구 방향을 제시하였다.

2. 배경 기술

2.1. 지능형 지속위협

지능형 지속위협 공격은 차세대 보안위협 중 하나로서, 목표로 하는 특정한 대상(Target)을 정한 후, 은밀하게 대상의 내부에 장기간 머무르면서 대상의 취약점이나,

중요한 데이터 등을 탈취하는 방법이다 [5]. 미국국립표준기술연구소(US National Institute of Standards and Technology, NIST)에서는 지능형 지속위협에 대해 다음과 같이 정의하고 있다 [6]:

"지능형 지속위협은 (ⅰ) 목적을 달성하기 위해 오랜 기간 동안 반복적으로 수행되고; (ⅱ) 공격을 막기 위한 공격 대상의 대응에 적응하고; (ⅲ) 목적 달성을 위해 필요한 대상과의 상호작용을 적정 수준으로 유지하는 것으로 결정된다."

위의 정의를 통해 지능형 지속위협은 (ⅰ) 특정한 대상과 분명한 목적이 존재하고, (ⅱ) 잘 조직되고 리소스를 충분히 가진 공격집단이 있으며, 그리고 (ⅲ) 오랜 기간 동안 반복적인 시도를 하면서도, 은밀하고 보이지 않는 공격 기술을 이용한다는 특징을 가지고 있음을 알 수 있다. [7]

2.2. 보안 요구사항과 위험 구성요소

보안 요구사항은 품질속성 요구사항(Non-functional Requirement)의 일종이면서, 기능적 요구사항(Functional Requirement)와 제약사항(Constraints)을 포함하고 있다. 그림 1은 김봉재의 연구에서 [4] 제안하는 모델로서 보안 요구사항을 정의하고 명세하기 위해 보안 요구사항을 구성하는 요소들을 정의하고 위험 구성요소들간의 관계를 나타낸 것이다. 기존의 공통 평가 기준 일반 모델(Common Criteria General Model) [8]과 Lee의 연구 [9]에서 제시된 내용을 통해 대칭적인 개념으로 보안 목표(Security goal)와 공격 목표(Malicious Goal)를 추가하여 보안 요구사항과 위험 구성요소 관계모델을 구성하였다.

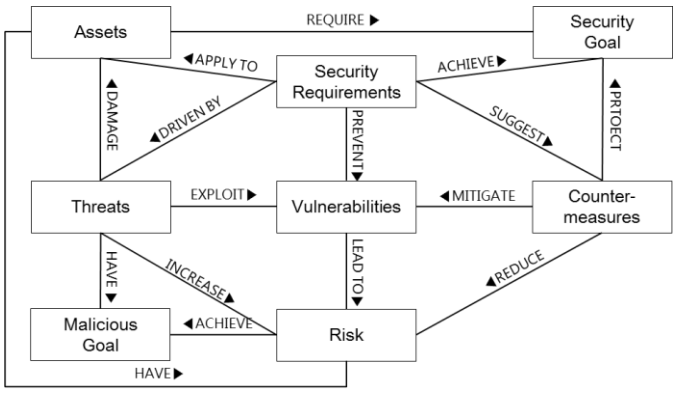


그림 1 보안 요구사항과 위험 구성요소 관계 모델 [4]

2.3. Keep All Objectives Satisfied (KAOS) 모델링

KAOS는 목표 기반 요구 공학 접근법(Goal-Oriented Requirement Engineering Approach)의 한 방법론으로 AND/OR을 이용한 트리 구조 모델로서 상위 계층은 Why에 대한 질문의 답을 도출하고, 하위 계층은 How에 대한 질문의 답을 도출해내는 모델링 방법이다. 본 연구에서 KAOS를 통해 지능형 지속위협 생애주기를 목표 기반으로 재구성하는 방법론을 제시하려고 한다.

3. 관련연구

본 절에서는 지능형 지속위협 공격에 대응하기 위한 선행 연구들을 소개한다.

이문구의 연구는 [10] 지능형 지속위협에 대응하기 위해 영역별 보안계층, 영역별 연계계층, 행위가시화 계층, 행위통제계층, 융합대응계층으로 총 5단계 계층으로 구성된 차세대 융합형 보안 프레임워크를 제안한다. 각 영역들은 관리적 보안, 물리적 보안, 기술적 보안을 모두 포함하면서 계층간 연계성을 가지도록 구성되어있다. 하지만 지능형 지속위협 생애주기의 각 단계에 따른 보안 요구사항은 얻을 수 없다는 한계점이 존재한다.

본 연구에서는 소프트웨어 공학적 접근 방법을 통해 지능형 지속위협의 보안 요구사항을 도출하고 적절한 요구사항을 추천하고자 한다.

김봉재의 연구는 [4] 소프트웨어 공학과 문제 도메인 온톨로지(Problem Domain Ontology)접근법 그리고 3계층 접근법을 통해, 사회-기술적 시스템을 노리는 여러 위협에 대한 종합적인 보안 요구사항을 도출해내는 프레임워크를 제시한다. 특히, 물리 계층, 정보-모델링 계층, 인지 계층으로 이루어진 PIC 3계층 모델을 통해 광범위한 지식과 모델을 통합하고, 위협 분석 및 위험 평가를 활용해 보안 요구사항을 이해한다. 그리고 온톨로지를 통해 보안 요구사항을 추천해준다. 하지만 지능형 지속위협을 현재 프레임워크에서 이해할 수 있도록 명세하지 못하였다.

본 연구에서는 지능형 지속위협의 다양한 공격 방식들을 물리 계층에서 구분하고, 정보-모델링 계층에서 공격방식과 생애주기와의 관계를 모델링 하여 그에 따른 추천 보안 요구사항을 얻고자 한다.

4. 제안 방법

본 절에서는 KAOS 모델링을 이용하여 지능형 지속위협의 생애주기를 목표기반으로 재구성하고, 재구성된 생애주기를 통해 각 단계마다 제안하는 위험 구성요소 관계모델을 이용하여 지능형 지속위협의 단계별 보안요구사항을 명세하는 방법을 제안한다.

4.1. KAOS 기반 지능형 지속위협 생애주기 재구성

그림 2는 지능형 지속위협의 생애주기를 나타낸 것으로, 각 단계는 지능형 지속위협 공격의 일부분이다. 각 단계는 목표로 하는 공격 목표(Malicious Goal)와 자산(Assets), 대응책(Counter-measures) 그리고 보안

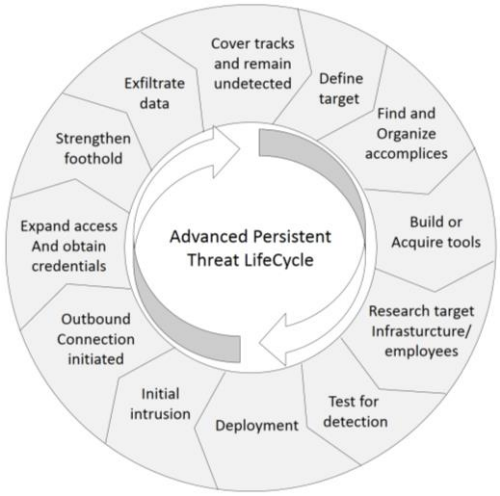


그림 2 지능형 지속위협 생애주기(APT Life Cycle)

목표 (Security goal)등 위험 구성요소가 다르기 때문에 그에 따른 보안 요구사항 또한 달라진다. 따라서 지능형 지속위협에 대한 더 깊이 있는 보안요구사항을 얻기 위해서는 생애주기의 각 단계마다 위험 구성요소관계 모델을 통해 단계별 보안요구사항을 추출 해야 한다. 하지만 그림 2에서 보이는 생애주기의 단계는 대상 정하기(Define Target)같은 목표(Goal)와 배치(Deployment)와 같은 기능(Softgoal)이 혼재되어 있어 모든 단계에서 위험 구성요소관계 모델을 나타낼 수 없다. 때문에 목표 기반으로 생애주기를 재구성해야 할 필요성이 있다. 이를 위해 비즈니스 골(Business goal)과 목적(Objective)을 가지런히 보여줄 수 있는 KAOS 모델링을 [7] 사용해 각 단계들을 목표와 기능을 구분하고 하나의 목표로 통합하여 지능형 지속위협 생애주기를 목표 기반으로 재구성한다.

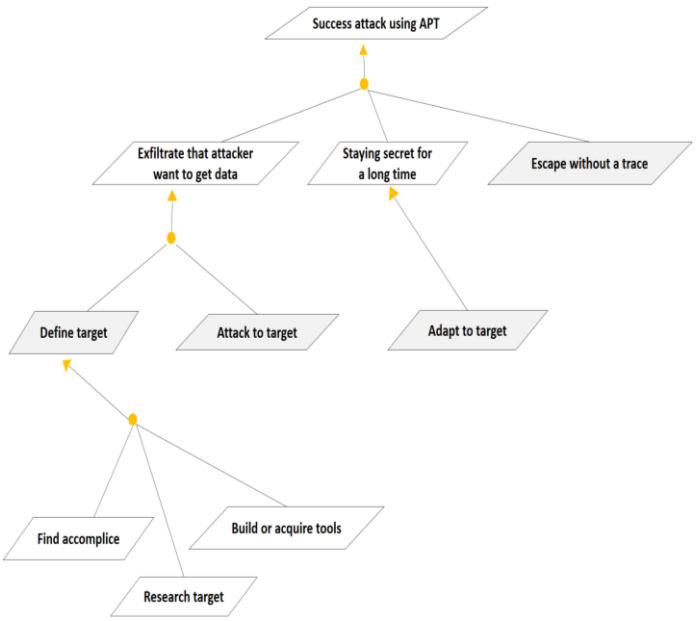


그림 3 지능형 지속위협 KAOS 모델링

지능형 지속위협을 분석하여 최종 목표(Goal)을 지능형 지속위협을 이용한 공격의 성공으로 정의하였고 그에 따른 하위 목표로 공격자가 원하는 데이터를 탈취하는 것, 시스템에 은밀하게 오랜 시간 동안 머무르는 것 그리고 흔적 없이 나오는 것으로 정의하였다.

그림 3에서 회색 박스로 표시된 목표들은 기존의 지능형 지속위협 생애주기에서 분석하여 추출한 목표들이다. 이 목표들을 KAOS 모델링에 적용시켜 그림 3과 같은 지능형 지속위협 KAOS 모델을 만들었다. 이를 기반으로 그림 4와 같은 KAOS기반 지능형 지속위협 생애주기로 재구성하였다.

그림 3에서 대상 정하기 목표에서 하위 구성요소들은 대상 정하기를 위한 기능들로 기존의 지능형 지속위협 생애주기에서 하나의 주기에 해당하는 요소들이다. 하지만 KAOS 모델링을 통해 대상 정하기의 하위 기능들로 통합되었다. 재구성된 생애주기를 통해 각 단계별로 향상된 위험 구성요소 관계 모델을 구성하여 보안 요구사항을 얻을 수 있다.

4.2. 지능형 지속위협 생애주기를 고려한 위험 구성요소 관계 모델

목표 기반으로 재구성된 지능형 지속위협 생애주기는 각 단계별로 적절한 보안 요구사항과 위험 구성요소들과의 관계를 모델링하여 나타낼 수 있다. 하지만 위험 구성요소 관계 모델 [4]에서는 생애주기의 각 단계를 표현해주고 다른 단계와의 연관성을 표현해줄 수 있는 구성요소가 존재하지 않는다. 각각의 단계가 연관성을 가지고 수행되어야, 지능형 지속위협으로 표현됨에 따라, 현재의 위험 구성요소 관계 모델로는 지능형 지속위협을 표현할 수 없다.

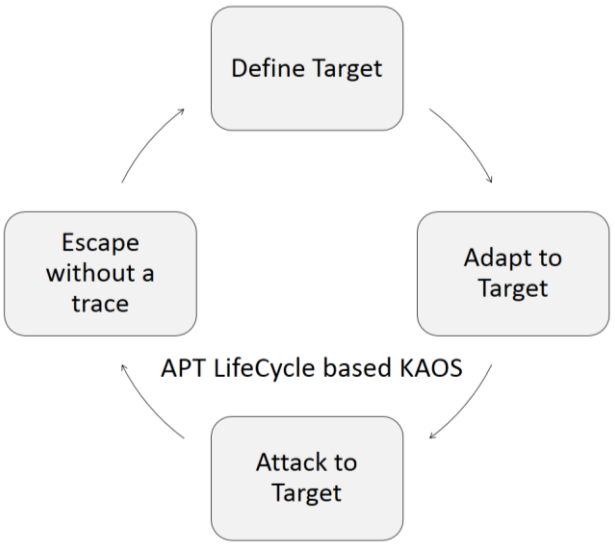


그림 4 KAOS기반 지능형 지속위협 생애주기

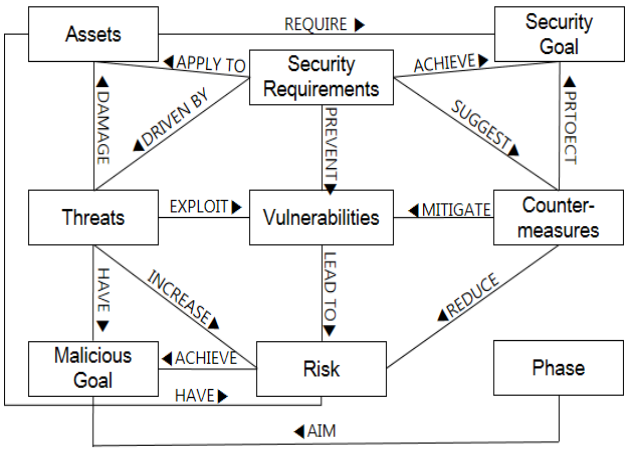


그림 5 지능형 지속위협 생애주기를 고려한 위험 구성요소 관계모델

본 연구에서는 지능형 지속위협 생애주기 각 단계마다 모델 사이의 연관성을 표현해줄 수 있는 구성요소를 추가한 향상된 위험 구성요소 관계 모델을 제안하여, 위험 구성요소 관계 모델들끼리 연관성을 나타내고 지능형 지속위협 생애주기를 표현 할 수 있도록 한다. 그림 5는 그림 1의 기존에 제안한 위험 구성요소 관계모델에서 지능형 지속위협 생애주기의 단계를 나타낼 수 있는 단계(Phase) 구성요소를 추가한 것이다. 지능형 지속위협 생애주기의 각 단계가 특정한 공격목표를 이루고자 한다는 관계를 '목표로 한다'는 AIM 관계로 표현하였다.

기존의 위험 구성요소 관계모델에서 지능형 지속위협에 대한 전반적인 보안 요구사항을 얻었다면, 향상된 위험 구성요소 관계모델에선 지능형 지속위협의 각 단계별로 위험 구성요소들과 그에 맞는 보안 요구사항을 얻을 수 있다. 또한 단계(Phase) 구성 요소들을 연결하여 지능형 지속위협에 대한 통합적인 보안 요구사항을 얻을 수 있고 추천 프레임워크에서 표현 할 수 있다.

5. 결론 및 향후 연구방향

본 연구에서는 지능형 지속위협 생애주기를 목표 모델링(Goal modeling)의 한 기법인 KAOS를 이용하여 지능형 지속위협 생애주기를 목표기반으로 재구성하고, 향상된 위험 구성요소 관계모델을 제안하였다. 재구성된 생애주기를 통해 각 단계별로 향상된 위험 구성요소 관계모델을 나타낼 수 있고, 각 단계마다 적절한 보안 요구사항을 도출해낼 수 있다. 더 나아가, 단계 구성요소를 통해 관계모델들을 연결하여 추천 프레임워크에서 지능형 지속위협을 표현할 수 있다. 이를 통해 지능형 지속위협에 대해서 통합적인 보안 요구사항 추천과 평가를 할 수 있다.

하지만 아직까지 본 연구에서는 제안한 방법에 대한 사례연구가 이루어지지 않았다. 향후 지능형 지속위협의 여러 공격 방식들을 향상된 위험 구성요소 관계모델을 모델링하는 사례연구를 실시하려한다. 또한 이렇게 얻어진 보안 요구사항들이 추천 프레임워크에서 다른 도메인마다 적절한 보안 요구사항으로 추천되는지 검증을 실시할 예정이다.

Acknowledgement

이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보 ·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(2013M3C4A7056233)
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW특성화대학원 지원 사업의 연구결과로 수행되었음"(R0346-15-1017)

참고문헌

[1] I. Sommerville, "Sociotechnical systems," in Software Engineering, Pearson, 2009, pp.263-288
[2] N. Falliere, L. O. Murchu and E. Chien, "W32.Stuxnet Dossier," Symantec, Cupertino, 2011
[3] Ponemon Institute, "2011 Cost of Data Breach Study: United States," Ponemon Institute, North Traverse City, 2012
[4] Bongjae Kim, Seok-Won Lee, "Analytical study of Cognitive Layered Approach for Understanding Security Requirements Using Problem Domain Ontology," APSEC, 2016
[5] E. Cole, "Advanced persistent threat: Understanding the danger and how to protect your organization," Syngress Media, 2012
[6] NIST. Managing Information Security Risk: Organization, Mission, and Information System View. SP 800-39, 2011
[7] F. Almisned and J. Keppens, "Requirements Analysis: Evaluating KAOS Models," Journal of Software Engineering and Applications, Vol. 3 No. 9, pp. 869-874, 2010
[8] Common Criteria, "Part 1: Introduction and general model," in Common Criteria for Information Technology Security Evaluation, Common Criteria, pp.38-44, 2012
[9] Seok-Won. Lee, Robin Gandhi, Divya Muthuraja, Deepak Yavagal, Gil-Joon Ahn "Building Problem Domain Ontology from Security Requirements in Regulatory Documents," in Workshop on Software Engineering for Secure Systems, pp. 43-50, 2006
[10] Moongoo Lee, Chunsock Bae. "Next Generation Convergence Security Framework for Advanced Persistent Threat," Journal of the Institute of Electronics and Information Engineers, 50(9), 92-99. 2013

결함 분류체계와 휴먼 에러 분류체계를 적용한 SW 구현 단계 FMEA 수행 체계

최이수 [1] 한동준 [2] 한혁수 [1]
상명대학교 컴퓨터과학과 [1], 상명대학교 산학협력단 [2]
choi.yisoo.hi@gmail.com, handongjoon@gmail.com, han.hyuksoo@gmail.com

SW FMEA Development Process at Implementation Stage based on Defect Taxonomy and Human Error Taxonomy

Yisoo Choi[1] Dongjoon Han[2] Hyucksoo Han[1]
Department of Computer Science, Sangmyung University[1]
Industry Academic Cooperation Foundation, Sangmyung University[2]

요 약

SW의 활용 분야가 다양해짐에 따라 SW 실패를 미리 파악하고 대비하는 것이 중요하다. 시스템의 실패를 예방하기 위해 전통적으로 FMEA가 많이 사용되어 왔고, SW 분야에서도 SW FMEA에 대한 연구가 활발히 진행되고 있다. 대부분의 SW FMEA 작성 방법에서 Failure Mode의 정의와 원인 분석은 전문가의 경험과 판단에 의존하고 있어서, 작성자의 수준에 따라 정확성과 일관성이 떨어지고, 결과의 편차가 심하다는 어려움을 겪고 있다. 이러한 문제를 보완하기 위하여, 본 논문에서는 SW Failure Mode를 정의하는 단계에서 결함 분류체계를 참조하고, 휴먼 에러 분류체계를 이용하여 Failure Mode의 원인을 분석하는 SW 구현 단계 FMEA 체계를 제안한다. 본 논문에서 제안한 체계를 위해 기존 연구와 SW의 실패 특성을 반영하여 결함 분류체계를 재정의하고 휴먼 에러 분류체계를 재구성하였다. 또한 실제 사례에 적용하여, 그 효용성을 입증하였다.

1. 서 론

SW 실패로 인한 사고는 인명, 재산 등에 심각한 피해를 발생시킬 수 있다. 이를 방지하기 위하여 SW 안전성을 관리하고 향상시킬 필요가 있다. SW의 안전성 확보를 위해서는 개발이 완료되기 이전에 잠재적인 SW 실패 요인을 파악하고 원인을 해결하기 위한 방안이 필요하다.

전통적으로 HW 분야에서는 실패를 예방하고 안전성을 향상시키기 위한 방법으로 FMEA(Failure Mode and Effect Analysis)가 널리 적용되어 왔다. FMEA는 시스템이나 장치 등의 실패 요인을 도출하기 위한 기법으로, 기기 혹은 부품 등에 Failure Mode가 발생하였을 때 시스템에 발생할 수 있는 위험 순위를 정하고 이에 대응할 수 있도록 하는 방법이다[1]. FMEA 방식에서는 표 1과 같은 워크시트를 사용하여 Failure Mode를 정의하고 그 영향도와 원인을 분석한다.

SW 분야에서도 실패 예방과 안전성 확보를 목적으로 SW FMEA에 대한 연구가 활발히 진행되고 있다. 안전 필수 시스템에 이용되는 표준인 ISO 26262와 A-SPICE 등에서 권고하는 HARA(Hazard Analysis and Risk Assessment) 방식에도 SW FMEA가 포함되어 있다. SW FMEA는 요구사항 단계 FMEA, 설계 단계 FMEA, 구현 단계 FMEA로 실행될 수 있다. 본 논문에서는 SW 구현 단계에서 FMEA를 적용하기 위한 체계를 구축하고자 한다.

일반적으로 FMEA 워크시트 작성에서 Failure Mode를 정의하고 원인을 분석하는 방법은 주로 전문가의 경험과 판단에 의존하거나 참여자들의 브레인스토밍(Brainstorming)을 통해 이루어진다[2]. 하지만 이러한 방식에서는 전문가의 수준에 따라 정확성과 일관성이 떨어지며, 결과의 편차가 심하다는 어려움이 있다. 브레인스토밍 또한 다양한 관점을 반영할 수 있으나 체계적이지 않기 때문에 발견된 Failure Mode의 완전성을 확보하기 어렵다. 그러므로 FMEA의 Failure Mode 작성과 원인 분석 작업을 체계화할 수 있는 방법이 요구된다.

본 논문에서는 결함 분류체계를 참조하여 SW Failure Mode를 정의하는 방안과 휴먼 에러 분류체계를 기반으로 원인을 분석하는 방안을 제시한다.

결함 분류체계는 SW 결함을 분류하는 체계로, 결함의 유형을 정의하고 그 유형과 연관된 속성들을 포함한다. SW 결함은 SW Failure로 이어지는 Mode를 나타내기 때문에 SW FMEA의 Failure Mode의 유형과 연결될 수 있다. 그러므로, 결함 유형은 SW Failure Mode의 작성단계에서 참조자료로 사용할 수 있다. 예를 들어, 작성자가 Failure Mode를 정의하는 단계에서 결함 유형 중 하나인 '데이터 접근 오류'

항목을 참조하여 '올바르지 않은 위치의 데이터 사용'을 Failure Mode로 정의할 수 있다.

SW 분야에서 휴먼 에러 분류체계는 개발자가 범하는 실수들의 분류체계이다. 결함의 주된 원인은 시스템 결함과 휴먼 에러인데, SW 개발은 인적 자원의 비중이 큰 인력 집약적 작업이기 때문에 결함의 70% 이상은 휴먼 에러로 인해 발생한다[7]. 그러므로 휴먼 에러 유형은 FMEA의 Failure Mode의 원인과 연계될 수 있다.

본 논문에서는 제안된 체계의 적합성을 높이기 위해 기존 결함 분류체계들과 SW의 실패 특성을 반영해 결함 분류체계를 재정의하였다. 또한, 기존의 휴먼 에러에 관한 연구와 SW 개발자의 실수에 기반하여 휴먼 에러 분류체계를 재구성하였다. 그리고 이를 적용한 SW 구현 단계 FMEA 수행 체계를 제안한다. 그 효용성을 입증하기 위해 K사의 이미지 인식 프로그램 개발 사례에 적용하였다.

2. 관련 연구

2.1. SW FMEA

FMEA는 전통적으로 부품의 오류로 인해 시스템 전체에 고장 발생 원인이 전파되는 것을 차단하기 위해 사용하는 기법이다. 이는 설계, 공정, 품질보증 등 각 부문에 산재한 안전성 문제점을 정량적으로 관리하기 위해 시스템 및 제품 구성 요소의 Failure Mode를 분석하여 시스템 자체의 영향을 평가하는 Bottom-up 방식이다[2].

SW FMEA는 SW 개발 단계가 진행됨에 따라 발견하기 어려워지는 SW Failure를 제품 개발 초기단계에서부터 발견하고 제거하기 위한 목적으로 활용된다. 이는 생명주기 단계 중 어느 시점에 적용되는지에 따라 요구사항 단계 FMEA, 설계 단계 FMEA, 구현 단계 FMEA로 나눌 수 있다[2]. 상위 단계 FMEA에서 식별된 원인이 하위 단계 FMEA의 Failure Mode로 연계되기 때문에, 상위 단계인 설계 FMEA에서 식별된 Failure Mode의 원인을 해당 단계에서 해결하지 못한 경우에는 하위 단계인 구현 FMEA에서 이를 해결할 필요가 있다. 본 논문에서는 SW 개발 단계 중 SW 구현 단계에 대해 FMEA를 적용하기 위한 수행 체계를 구축하고자 한다.

일반적으로 SW FMEA는 표 1과 같은 워크시트 기반으로 수행되며 주된 작업은 다음과 같다.

(1) Failure Mode를 정의한다.
(2) Failure Mode의 원인 및 Failure로 인한 영향을 파악한다.
(3) 심각도(Severity), 발생 가능성(Occurrence), 검출 가능성(Detection)을 측정한다.
(4) 이를 기반으로 RPN(Risk Priority Number) 값을 계산하여 위험 순위를 매긴다.
(5) 위험 순위가 높은 Failure Mode부터 각각의 해결방안을 정의한다.

표 1. 일반적인 FMEA 워크시트

Module (Component)	Failure Mode	Effect of Failure	Severity	Cause	Occurrence	Detection	RPN	Control Action (Improvement)	NEW RPN

이 작업은 전문가의 지식에 의존하거나, 참여자들의 브레인스토밍 방식으로 수행되기 때문에 그 일관성과 완성도가 떨어질 수 있다.

2.2. 결함 분류체계(Defect Taxonomy)

결함 분류체계는 결함에 대한 포괄적인 정보를 제공한다[1]. 결함 분류체계의 주요 속성으로는 결함 유형이 있다. 표 2는 Microsoft사의 결함 유형을 나타내고 있는데, 이 유형들은 결함이 시스템에 미치는 종류별로 분류 되어 있다.

표 2. Microsoft 사의 결함 유형[1]

Defect Type	Detail
Reliability	Incorrect result
	Igonored failure
	Unexpected failure
	Unexpected success
	Incomplete
Performance	Memory leak
	Resource leak
	Algorithm
	Inherent
	Excessive inputs
	Synchronization
Corruption	Self
	Objects
	Object organization
	Persistent
	Persistent organization
Disclosure	Internal data
	Stored user data
	Stored users data
	Deception
Functionality	Missing part
	Accessibility
	Complex
	Missing functionality

2.3. 휴먼 에러 분류체계(Human Error Taxonomy)

휴먼 에러는 사람에 의해서 수행되는 모든 작업에서 발생한다. 표 3은 기존 연구에서 파악한 휴먼 에러 유형을 요약한 내용이다.

표 3. 기존 연구의 휴먼 에러 유형

Autor	Human Error Type
Swain, Guttman[3]	omission error, commission error, extraneous act error, sequential error, timing error
Alphonse Chapanis[9]	task execution incomplete, task executed in wrong direction, wrong task executed, task repeated, task executed on wrong component, task executed too early/late/much/little, misread information
L.W. Rook[10]	ergonomic design error, produce error, verification error, install error, control error, treatment error
Schadler[11]	inadequate training, breakdown of communication and not accounting for all details of task
Reason[12]	slips-lapses-mistakes
Rasmussen[13]	skill-based, rule-based, knowledge-based error

SW 결함의 원인을 휴먼 에러 관점으로 다루는 연구는 주로 사람의 인지 모델을 중심으로 연구 되었다[5][6]. 하지만 인지 모델에 전적으로 의존하는 접근법은 인지 외의 이유로 발생하는 휴먼 에러에 대한 고려가 부족하다. SW 개발 과정에서는 인지 오류 외에도 설계 및 구현에 대한 지식 부족, 정의 되지 않은 프로세스의 수행, 도구의 잘못된 사용 등과 같이 결함을 유발할 수 있는 다른 요인들이 존재한다. 이러한 요인들도 SW Failure의 원인 분석에 고려되어야 한다.

3. 결함 분류체계와 휴먼 에러 분류체계의 재구성

본 논문에서는 SW 특성에 적합하도록 결함 유형을 재정의하였으며 SW Failure Mode의 원인을 적절하게 분석할 수 있도록 휴먼 에러 유형 분류체계를 재구성하였다.

3.1. 결함 분류체계 재정의

SW Failure는 SW의 결함이 시스템에 미친 영향의 결과라고 볼 수 있다. 그러므로 SW Failure Mode는 SW 결함의 특성과 연관된다. 이러한 특징을 반영하기 위해 Microsoft사의 결함 유형을 기반으로 하고 각 결함 유형에 속하는 상세 결함 항목들은 IEEE 1044와 IBM사의 ODC를 참조하여 재정의 하였다[1][4][8].

표 4. 재정의한 결함 분류체계

Defect Type	Defect
신뢰성 결함	결과값이 옳지 않음
	오류를 무시함
	오류를 인지 못함
	정상 동작을 오류로 인지함
	동작 수행을 완료하지 못함
성능 결함	메모리 누수가 발생함
	자원 누수(CPU, 네트워크 등)가 발생함
	알고리즘에 의한 속도 저하가 발생함
	HW 한계치로 인한 성능 한계가 발생함
	권고량이 초과로 입력됨
	동기화 오류(재시도, 타임아웃)가 발생함
데이터 접근 결함	내부 데이터의 비정상적 접근이 발생함
	사용자 데이터의 비정상적 접근이 발생함
기능 결함	구성 요소 누락(설정 파일, 자원)됨
	불필요한 기능이 개발(기능 시나리오 누락)됨
지역화 결함	언어 설정이 잘못됨
	국가권 설정이 잘못됨
준법(국가,지역) 결함	현지 규제를 준수하지 않음

3.2. 휴먼 에러 분류체계 재구성

Swain과 Guttman[3]의 연구에 따르면 휴먼 에러는 생략 에러, 실행 에러, 과잉 행동 에러, 순서 에서, 시기 에러로 분류된다. 이 중 시기에러는 구현 단계에서 SW Failure를 발생시키는 원인에 적합하지 않기 때문에 제외하고 표 5와 같이 휴먼 에러 분류체계를 재구성하였다.

표 5. 재구성한 휴먼 에러 분류체계

Human Error Type	Detail
생략 에러(omission error)	필요한 작업 내지 단계를 수행하지 않은 에러
실행 에러(commission error)	작업 내지 단계는 수행하였으나 잘못한 에러
과잉행동 에러(extraneous act error)	해서는 안 될 불필요한 작업 행동을 수행한 에러
순서 에러(sequential error)	작업 수행의 순서를 잘못한 에러
시기 에러(timing error)	주어진 시간 내에 동작을 수행하지 못 하거나 너무 빠르게 혹은 너무 느리게 수행하였을 때 생긴 에러

4. 결함 분류체계와 휴먼 에러 분류체계를 적용한 SW 구현 단계 FMEA 수행 체계

본 연구에서는 재정의한 결함 분류체계와 재구성한 휴먼 에러 분류체계를 적용하여 SW 구현 단계 FMEA 수행 체계를 개발하였다. 개발된 SW FMEA의 수행 프로세스는 그림 1과 같다.

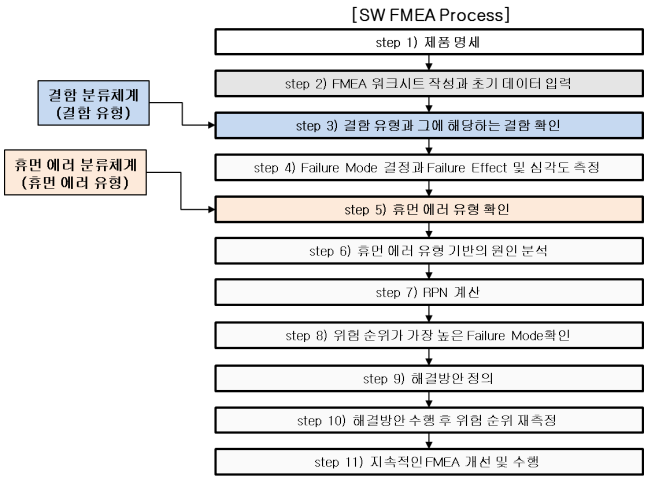


그림 1. 결함 분류체계와 휴먼 에러 분류체계를 적용한 SW 구현 단계 FMEA 수행 프로세스

SW FMEA 작성을 용이하게 하기 위해, 표 6과 같이 결함 분류체계와 휴먼 에러 분류체계를 참조할 수 있는 기능을 포함한 워크시트를 개발하였다.

5. 적용사례

본 연구에서 개발한 SW 구현 단계 FMEA 수행 체계를 중소기업인 K사에 적용하였다. K사는 임베디드 SW를 만드는 회사로 개발자 70명으로 구성된 개발 조직이다.

K사의 기능 모듈 1개를 선정하고 본 논문에서 개발한 SW FMEA 수행 체계를 적용하였다. 결함 분류체계의 결함 유형과 상세 결함 항목을 참조하여 주어진 모듈에 발생할 수 있는 Failure Mode를 표 7과 같이 도출하였다.

표 6. 결함 분류체계와 휴먼 에러 분류체계를 포함한 SW 구현 단계 FMEA 워크시트

Module (Component)	Defect Type	Defect	Failure Mode	Effect of Failure	Severity	Human Error Type	Cause	Occurrence	Detection	RPN	Control Action (Improvement)	NEW RPN

표 7. 사례에서 도출한 Failure Mode

Defect Type	Defect	Failure Mode
신뢰성 결함	결과값이 옳지 않음	인식한 홀로그램 값을 처리한 값이 옳지 않음
신뢰성 결함	오류를 무시함	오류를 수정하지 않은 홀로그램 값을 사용함
신뢰성 결함	오류를 인지 못함	오류가 포함된 홀로그램 값을 사용함
신뢰성 결함	정상 동작을 오류로 인지함	처리한 홀로그램 값을 권종 인식 모듈로 보내지 않음
신뢰성 결함	동작 수행을 완료하지 못함	처리한 홀로그램 값을 권종 인식 모듈로 보내지 않음
성능 결함	알고리즘에 의한 속도가 저하가 발생함	정사기의 지폐 인식 속도에 맞춰 홀로그램을 인식하지 못함
성능 결함	동기화 오류가 발생함 (재시도, 타임아웃)	이전에 인식한 홀로그램 값을 계속 사용함
데이터 접근 결함	내부 데이터의 비정상적 접근이 발생함	적합하지 않은 홀로그램 값을 불러옴
기능	불필요한 기능이 개발됨 (기능 시나리오 누락)	홀로그램 인식 외에 다른 부분도 인식함
지역화 결함	국가권 설정	홀로그램 위치가 잘못 설정됨

또한 본 체계의 휴먼 에러 분류체계의 휴먼 에러 유형을 참조하여 Failure Mode의 원인(Cause)을 분석하였다. 표 8은 '결과값이 옳지 않음'의 원인을 분석한 결과이다. '실행 에러' 유형에서는 두 가지 원인이 도출되었다.

표 8. '결과값이 옳지 않음'에 대한 원인

Human Error Type	Cause
생략 에러	필요한 코드를 생략 하였음
실행 에러	코드 로직 자체를 잘못 작성 하였음
	유닛 테스트를 잘못 수행 하였음
과잉행동 에러	불필요한 코드를 추가 하였음
순서 에러	공통 코드 구현 이전에 코드를 구현함

6. 결론 및 향후 연구

본 논문에서는 결함 분류체계를 재정의하고 휴먼 에러 분류체계를 재구성하여 이를 반영한 SW 구현 단계 FMEA 수행 체계를 제안하였다. 제안된 수행 체계의 기대효과로는 FMEA의 작성자가 Failure Mode 정의와 그 원인 분석을 체계적으로 수행하여 SW 실패로 인한 피해를 예방하고 안전성을 확보하는데 기여하는 것이다.

이번 논문에서는 휴먼 에러 분류체계를 Failure Mode의 원인 분석에 활용하는 용도로만 사용하였다. 추후 연구로는 Failure Mode의 결함 유형과 휴먼 에러 유형을 기반으로 Control Action을 제안하는 방안이 필요하다. 또한 요구사항 단계와 설계 단계에서 활용할 수 있는 SW FMEA 수행 체계를 개발해야 한다.

사사 (ack)

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음. (IITP-2016-R0992-16-1014)

참조 문헌

[1] McDonald, Marc, Robert Musson, and Ross Smith. "The practical guide to defect prevention". Microsoft Press, 2007.

[2] Softrel, LLC. "Effective Application of Software Failure Modes Effects Analysis". Quanterion Solutions, Inc. 2014.

[3] Swain, A.D. & Guttmann, "H.E., Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications". 1983, NUREG/CR-1278, USNRC.

[4] ISDW Group. "1044-2009-IEEE Standard Classification for Software Anomalies". IEEE, New York. 2010.

[5] Huang, Fuqun, Bin Liu, and Bing Huang. "A taxonomy system to identify human error causes for software defects". The 18th international conference on reliability and quality in design. 2012.

[6] Anu, Vaibhav, et al. "Development of A Taxonomy of Requirements Phase Human Errors Using a Systematic Literature Review: A Technical Report". Technical Report. 2016.

[7] Lee, K. W., Tillman, F. A. and Higgins, J. "literature survey of the human reliability component in a man-machine system". IEEE Transactions on Reliability. 1988.

[8] Kumar, Sharad. "Software Defect Prevention through Orthogonal Defect.". IRJA-Indian Research Journal. 2004.

[9] Morgan, Clifford T., et al. "Human engineering guide to equipment design". 1963.

[10] Rook Jr, L. W. "Reduction of human error in industrial production". No. SCTM--93-62 (14). Sandia Labs., Albuquerque, N. Mex.(USA). 1962.

[11] Mark Schadler. "Casual Analysis and Resolution". Milwaukee School of Engineering. 2009.

[12] Reason, James. "Human error". Cambridge university press. 1990.

[13] Rasmussen, Jens, and Keith Duncan. "New technology and human error". John Wiley & Sons. 1987.

STPA를 활용한 ISO 26262 기반의 안전분석 체계수립

도성룡 O 한혁수 OO
현대오트론 O 상명대학교 OO
SungRyong.Do@hyundai-autron.com O hshan@smu.ac.kr OO

Applying STPA to Establish a Safety Analysis System based on ISO 26262

Sung Ryong Do O Hyuk Soo Han OO
Hyundai Autron O SangMyung University OO

요 약

최근에 자동차 산업은 연비와 안전에 대한 법규가 강화되고, 고객의 첨단 기능요구에 따라 전기전자제어 시스템 적용이 급증하고 있다. 하지만 전기전자제어시스템의 증가는 차량의 설계 복잡도를 높이고, 전체적인 오동작을 증가시키고 있다. 차량의 기능안전성을 강조한 ISO 26262 표준은 개발 단계 별 HAZOP, FMEA, FTA 등 안전분석 기법 적용을 요구하고 있다. 하지만 이러한 기법은 시스템 간 상호작용에 따른 실패를 다루기에는 한계가 있다. 본 연구에서는 ISO 26262의 컨셉/시스템/소프트웨어 안전 요구사항 개발을 위해 STPA를 활용한 안전분석 체계수립 방안을 제시한다. 본 연구의 효용성을 검증하기 위해 스마트키 시스템에 적용하고, 기존 기법과의 비교 결과를 제시한다.

1. 서론

자동차 산업은 연비와 안전에 대한 법규가 강화되고, ADAS(Advanced Driver Assistance Systems)와 같은 첨단 기능에 대한 요구사항이 증가함으로써 전기전자제어시스템 적용이 급증하고 있다. Strategy Analytics의 2013년 보고서에 따르면, 차량 내 전기전자제어시스템이 차지하는 비율은 2030년에 50%, 원가 비율은 2020년에 70%까지 이를 전망이다[1].

하지만 전기전자제어시스템의 증가는 설계 복잡도를 높이고, 전체적인 오동작을 증가시키고 있다. 이에 선진업체들은 2011년 ISO 26262 기능안전 표준을 제정하였다[2]. ISO 26262는 개발 단계 별 HAZOP, FMEA, FTA 등 안전분석 기법 적용을 통해 안전 요구사항 식별 및 안전 설계를 요구하고 있다. 이들 기법은 연속적인 이벤트로 인과관계를 구성하는 Chain of Event 모델 방식으로 신뢰성 이론에 기반을 두고 있다. 하지만 Chain of Event 모델 기반의 기법들은 시스템 간 상호작용에 따른 실패를 다루기에는 한계가 있다. 이에 MIT의 Leveson 교수는 STAMP(System Theoretic Accident Model and Processes) 모델 기반의 STPA(System Theoretic Process Analysis) 기법을 제안하였다.

본 연구에서는 ISO 26262의 컨셉/시스템/소프트웨어의 안전 요구사항 개발을 위해 STPA 적용방안을 제시한다. 이를 위해, 시스템의 동작 과정 정의에 유용한 UseCase 기술서와 STPA 1단계를 적용한 Hazard 식별방안을 제시한다. 또한 기존 FMEA/FTA와 STPA 2단계 기반의 원인분석 방안을 제시한다. 본 연구의 효용성을 검증하기 위해 차량의 스마트키 시스템에 적용하고, 기존 기법과의 비교 결과를 제시한다.

2. 배경 지식

2.1 Chain of Event 모델기반의 안전분석 기법

Chain of Event 모델은 시스템 구성 요소들의 연속적인 실패 사건(Failure Event)들로 인해 사고가 발생한다는 이론이다[3]. 예를 들어, 브레이크 시스템에서 제어기가 고장 났다면, 이를 대체하는 제어기를 설계에 반영함으로써, 사고를 예방할 수 있다는 개념이다. 대표적인 Chain of Event 모델 기반의 안전분석 기법은 HAZOP, FMEA, FTA 등이 있다.

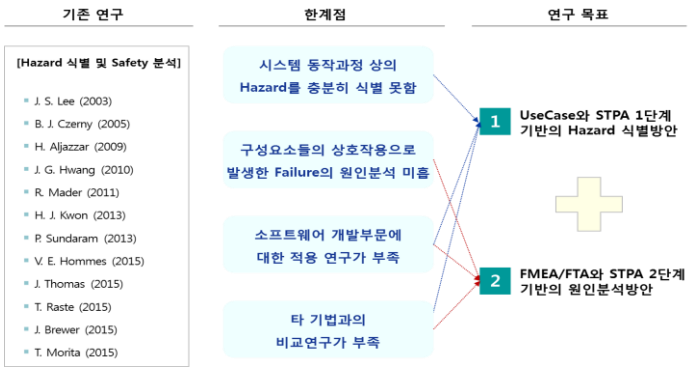
2.2 시스템 이론에 기반한 안전분석 기법 : STPA

Chain of Event 모델기반의 안전분석 기법은 기계장치가 주를 이루던 시대에 적합하다. 기계장치들로만 이루어져 있어 물리적으로 분리가 되고, 독립적으로 분석이 가능하였다. 또한 상대적으로 구성요소들 간의 상호작용이 단순하였기 때문에, 시스템 설계 오류들은 테스팅에 의해 대부분 발견되어 제거가 가능하였다.

하지만 최근의 전기전자제어시스템은 다양한 하드웨어와 제어 소프트웨어로 구성되어 있어 설계가 복잡하며, 더 이상 테스팅만으로는 오류 발견이 불가능해졌다[4,5]. 이에 Leveson 교수는 STAMP 모델 기반의 STPA 기법을 제안하였다. STPA 기법은 ① 대상 시스템의 제어구조도 작성, ② UCA(Unsafe Control Action) 식별, ③ 안전 제약사항 정의 ④ Causal Factor 도출의 네 단계로 구성되어 있다.

3. 기존연구 및 연구목표

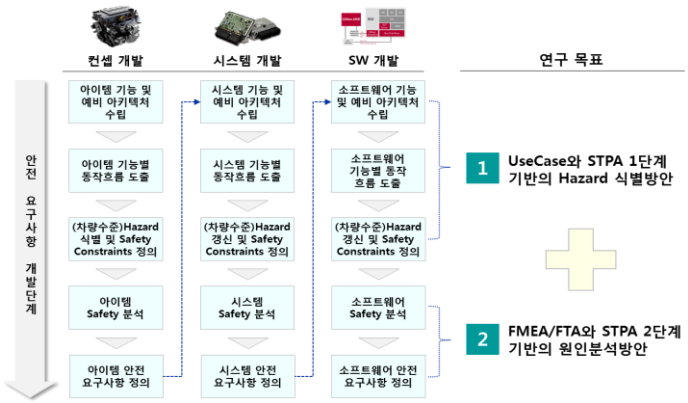
Hazard 식별 및 Safety 분석과 관련해서 PHA, HAZOP, FMEA, FTA, STPA 등 다양한 기법을 기반으로 연구가 진행되고 있다. 기존 연구를 분석한 결과 [그림 1]과 같이 한계점을 도출하였다. 본 연구에서는 기존 연구의 한계점을 보완하기 위해 STPA를 활용한 ISO 26262 기반의 안전분석 체계수립을 목표로 [그림 1]과 같이 두 가지 세부 목표를 수립하였다.



[그림 1] 기존 연구의 한계점 및 연구 목표

4. STPA를 활용한 ISO 26262 기반의 안전분석 프로세스

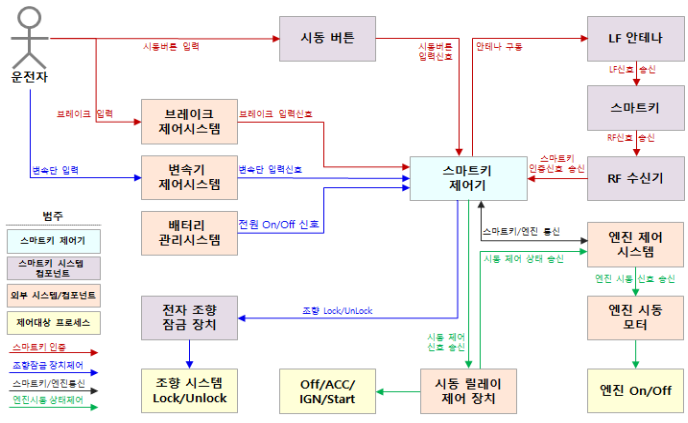
본 연구는 [그림 2]와 같이 ISO 26262 컨셉/시스템/소프트웨어 안전 요구사항 개발단계에서 STPA를 활용한 안전분석 프로세스 및 세부 활동을 제시한다. 또한 프로세스 활동 별 시스템의 규모 등 프로젝트와 조직의 상황에 맞게 조정할 수 있도록 필수와 선택으로 구분한다. 예를 들어, Hazard 식별 시 동작흐름 분석을 통해, 충분한 Safety Constraints가 도출된 경우에는 Safety 분석활동은 선택 적용이 가능하다.



[그림 2] STPA를 활용한 ISO 26262 기반의 안전분석 프로세스

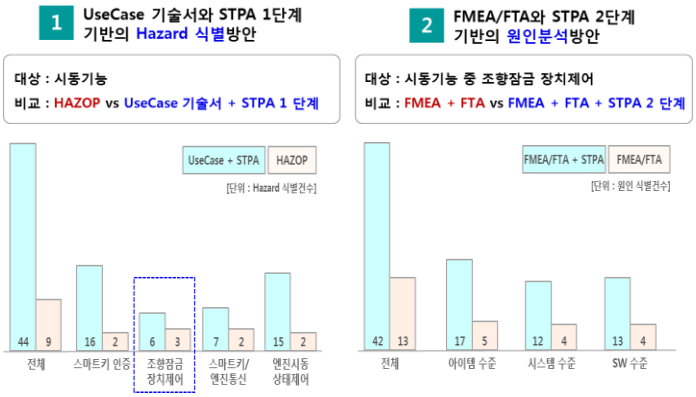
5. 적용 사례 및 연구 효과

본 연구의 효과를 검증하기 위해 스마트키 시스템에 적용한다. 예비 아키텍처 수립 결과는 [그림 3]과 같다.



[그림 3] 스마트키 시스템의 예비 아키텍처

본 연구 결과에 대해서 연구 목표 별로 기존 기법과 비교한 결과는 [그림 4]와 같다.



[그림 4] 연구목표 별 비교결과

6. 결론 및 향후 계획

본 연구에서는 STPA를 활용한 ISO 26262 기반의 컨셉/시스템/소프트웨어 안전 요구사항 개발을 위한 안전분석 체계를 수립하였다. 본 연구를 통해 다음의 효과를 확인할 수 있었다.

- UseCase 기술서를 통해 시스템 간 상호작용을 상세화 할 수 있으며, 그 결과로부터 시스템 동작과정 상의 구체적인 Hazard 식별이 가능하다. 또한 구체적으로 식별된 Hazard는 원인분석 단계에 활용이 가능하다.
- 시스템의 부분이 아닌 전체측면에서 원인 분석이 가능하다. 즉, 시스템 상호작용 과정 상의 잠재된 추가 원인도출이 가능하다.

그러므로 본 연구를 적용하는 조직에서는 개발 초기에 시스템 내 잠재하고 있는 더 많은 Hazard와 원인 그리고 안전 요구사항 도출이 가능하다. 결과적으로 시스템 품질 향상 및 품질 비용을 줄일 수 있을 것으로 기대한다.

하지만 본 연구는 자동차의 다양한 시스템 중, 스마트키 제어시스템 사례만 적용하였다. 또한 HAZOP, FMEA, FTA 등 일부 안전분석 기법과의 비교만 수행하였다. 이에 연구효과의 객관성을 확보하기 위해, 에어백, 엔진, 변속기 제어 시스템 등 자동차 분야의 복잡한 타 시스템에 적용하고, SWHA(SoftWare Hazard Analysis) 등 타 기법과의 비교를 수행할 예정이다.

참 고 문 헌

[1] Chris Webber., “Automotive Semiconductor Demand Forecast,” Strategy Analytics (2013).
[2] “ISO 26262 : Road Vehicles-Functional Safety,” ISO (2011).
[3] Benner, L., “Accident Investigations: Multilinear Events Sequencing Methods,” Journal of Safety Research (1975).
[4] Leveson, N., “Engineering a Safer World: Systems Thinking Applied to Safety,” MIT Press (2011).
[5] Leveson, N., “A New Accident Model for Engineering Safer systems,” Safety science, 42(4), 237-270 (2004).

안드로이드 버그 분류: 이슈 설명서 기반 버그 담당자 선정 자동화에 대한 연구

최성옥
카이스트 소프트웨어 대학원
ıewsungok2@gmail.com

조재호
카이스트 소프트웨어 대학원 / (주)LG 전자
allhave@kaist.ac.kr

민모란
카이스트 소프트웨어 대학원 / (주)LG 전자
minmoran@kaist.ac.kr

민상윤
카이스트 소프트웨어 대학원 / (주)솔루션링크
symin@kaist.ac.kr

Android Bug Triage: A Research on Automated Bug Owner Assignment Based on the Issue Description

Sung-Ok Choi
Graduate Program for Software Engineering

Jae-Ho Cho
Graduate Program for Software Engineering / LG Electronics Inc.

Mo-Ran Min
Graduate Program for Software Engineering / LG Electronics Inc.

Sang-Yoon Min
Graduate Program for Software Engineering / SOLUTIONLINK Co.

요 약

소프트웨어 기반의 새로운 융합 제품과 서비스가 증가함에 따라 소프트웨어 프로젝트 규모와 복잡도가 증가하고 있는 추세이다. 소프트웨어의 복잡도가 높아질수록 버그 또한 증가할 가능성이 상당히 커진다. 소프트웨어 개발 단계에서 발생하는 버그는 재 작업을 요함으로써 프로젝트 일정관리에 부담으로 작용하게 된다. 버그를 적합한 개발자에게 할당해주는 버그 분류는 일정 단축에 매우 중요한 요소이다. 특히, 버그가 대량으로 발생될 경우 수동으로 적절한 담당자를 찾는 것은 많은 시간이 소모된다.

본 논문에서는 안드로이드 이슈 추적 시스템에서 제공하는 버그 보고서를 기반으로 버그 담당자 선정의 자동화 방법을 제안한다. 먼저 과거 버그를 해결한 각 개발자를 분류한 후, 버그 분류에 적합한 데이터를 선정한다. *NLP*(Natural Language Processing) 알고리즘과 *TF-IDF*(Term Frequency- Inverse Document Frequency)를 활용하여 각 개발자의 문서 기반으로 선정된 데이터를 벡터 값으로 변경한다. 그리고 버그의 벡터 값과 각 개발자의 벡터 값의 코사인 유사도를 구하여 가장 높은 유사도를 갖는 개발자에게 버그를 할당한다. 연구 결과, 82.93%의 정확도를 보이는 버그 분류 모델을 도출하였다.

1. 서 론

최근 사용자의 요구사항이 다양해지고 새로운 융합 제품과 서비스가 증가함에 따라 소프트웨어 프로젝트 규모와 복잡도가 증가하는 추세이다. 소프트웨어의 복잡도가 높아질수록 버그 발생 또한 증가할 가능성이 상당히 커진다. 소프트웨어 개발 단계에서 발생하는 버그는 재 작업을 요함으로써 프로젝트 일정관리에 부담으로 작용하게 된다.

발생한 버그를 적합한 개발자에게 할당해주는 버그 분류는 일정 단축에 매우 중요한 요소이다. 특히, 버그가 대량으로 발생될 경우 수동으로 적절한 담당자를 찾는 것은 많은 시간이 소모된다.

버그 할당의 유형은 담당자에게 직접 통보되는 형태부터, 관리자를 통해 담당자에게 할당하는 형태까지 다양하다. 관리자가 있더라도 근본적으로 관리자가 기술을 잘 모르고 할당하는 경우도 적지않다.

어떠한 경우에도 버그는 가장 적절한 담당자에게 신속하게 할당되어야 해결도 빠르게 진행된다. 해결된다는 말은 버그가 수정되는 것만을 의미하는 것은 아니다. 보고된 버그가 중복된 버그이거나 정상동작으로 확인될 수도 있다는 것이다. 즉, 버그가 방치되지 않고 관리된다는 것에 의미가 있는 것이다.

현재까지 이클립스, 모질라 등의 오픈소스 프로젝트를 기반으로 버그 분류(이슈 담당자 선정하기)에 대한 연구[1]가 지속되어왔다. 2007년에 시작된 안드로이드는 다른 오픈소스 프로젝트에 비해서 역사가 짧고, 안드로이드 기기 제조사들이 안드로이드의 버그를 비공개함으로써 연구에 필요한 데이터 확보가 쉽지 않은 상황이었다. 이로 인해 안드로이드 기반의 프로젝트에서의 버그 분류에 대한 연구가 상대적으로 많이 이루어지지

않았다. 또한, 본 연구에서 실시한 온라인 설문조사 결과, 현업에서 부정확한 버그 분류로 인해 부서간의 *Silo* 현상 [1] 이 발생하고 있어 효율적인 안드로이드 버그 분류 시스템 정착이 시급함을 확인하였다.

1.1 설문 조사

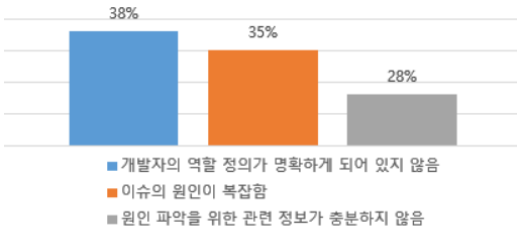
현재 버그 분류의 업계 실태를 파악하고 효율적인 버그 분류 방안이 필요하다는 본 연구의 타당성을 재확인한다.

1.1.1 설문 조사 대상

설문은 한국 주요 5개 기업의 IT분야 근무자를 대상으로 실시하였다. 기업 보안을 위하여 구체적인 기업명과 인원수는 논문에 기재할 수 없음을 밝힌다.
설문 대상자들은 주로 임베디드 및 플랫폼 영역에서 10.2년의 평균 경력을 가진 개발자를 대상으로 하였다.

1.1.2 설문 조사 내용 및 결과

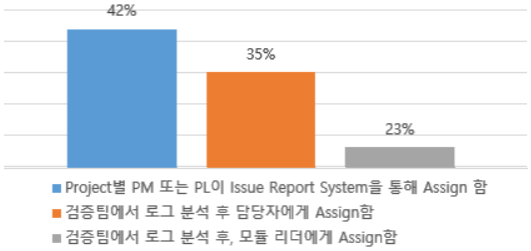
A. 현 소속사에서 버그 분류가 증가하는 주요 이유



<그림 1> 버그 분류 주요 이유 별 비중

첫번째 질문에 대한 응답으로 버그 분류가 각 회사에서 빈번히 일어나고 있으며 그 결과 많은 시간이 소모되고 있었다. 그 이유에 대해서는 각 개발자의 역할 정의가 명확하게 되어 있지 않아서 버그에 적합한 담당자를 찾기 힘들다는 것이 가장 주된 이유였다. 그리고 이슈가 단순하지 않고 복잡한 연계성을 갖기 때문이라는 의견도 있었다.

B. 현 소속사에서 버그를 분류하는 방법



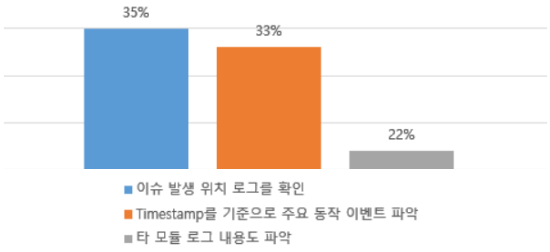
<그림 2> 버그 분류 방법 별 비중

두번째 질문에 대한 응답으로 프로젝트 별 프로젝트 매니저(PM) 또는 프로젝트 리더(PL)들이 검증팀으로부터 이슈를 보고 시스템을 통해 받고, 현상기반으로 각 파트장, 또는 실무자에게 할당한다는 응답이 가장 많았다. 그리고 검증팀에서 1차적으로 로그를 분석하여 이슈 담당자에게 직접 전달한다는 응답도 있었다.

C. 버그 분류에 대한 노하우

세번째 질문에 대한 응답은 소속사에 체계화되지는 않았지만 상대적으로 좋은 성과가 있었던 사례를 나타낸다. 크게 로그 분석을 하는 경우와 하지 않는 경우로 나눌 수 있었다.

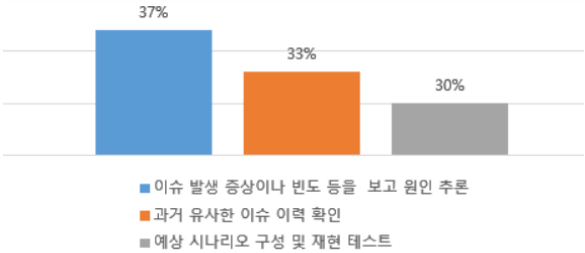
C-1. 로그 분석하는 경우



<그림 3> 개인의 노하우 별 비중
(로그 분석하는 경우)

로그를 분석하는 경우 이슈 발생 위치의 로그를 확인한다는 응답이 가장 많았다. 그리고 여러 모듈의 로그 분석 시 Timestamp를 기준으로 주요 동작의 이벤트를 파악한다는 응답도 있었다.

C-2. 로그 분석하지 않는 경우



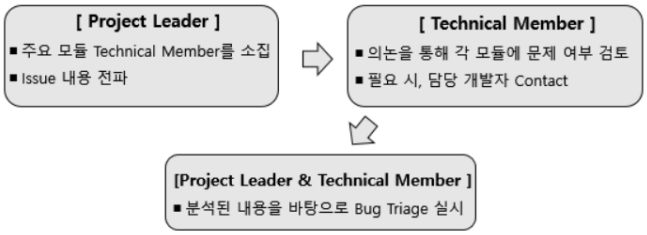
<그림 4> 개인의 노하우 별 비중
(로그 분석을 하지 않는 경우)

로그를 분석하지 않는 경우 이슈 발생 증상이나 빈도 등을 보고 원인을 추론한다는 응답이 가장 많았다. 그리고 과거 개발 중에 유사한 이슈 이력을 확인한다는 응답도 있었다.

[1] https://en.wikipedia.org/wiki/Silo

D. 현 소속사에 버그 분류에 대한 건의사항

네번째 질문에 대한 응답에서 각 개발자들이 버그 분류 개선을 위해 현 시스템에서 필요로 하는 프로세스가 무엇인지 알 수 있었다. 가장 많았던 의견으로는 <그림5>와 같이 프로젝트 리더와 모듈 담당자 사이의 협업을 통한 버그 분류에 대한 건의 프로세스였다.



<그림 5> 버그 분류에 대한 건의 프로세스

설문조사 결과, 각 기업의 시스템에서는 개발자가 만족할 만한 버그 분류가 시행되고 있지 않았다. 효율적인 버그 분류 정착이 시급함을 확인함과 동시에 본 연구의 타당성을 재확인 하였다.

본 논문에서는 *NLP*(Natural Language Processing) 알고리즘과 *TF-IDF*(Term Frequency- Inverse Document Frequency), 코사인 유사도로 다양한 실험을 진행하여 모델의 유효성을 확인한다. 그 결과 82.93%의 정확도로 버그 분류되는 것을 관찰할 수 있다.

1.2 본 논문의 연구방향

1.2.1. 버그 분류 자동화

안드로이드 이슈 추적 시스템의 버그 데이터를 활용하여 NLP(Natural Language Processing) 알고리즘 기반 버그 분류 자동화 모델을 제시한다.

1.2.2. 데이터 추가 수집 및 정제

안드로이드 이슈 추적 시스템에서 기본 제공하는 데이터 종류(Status, Summary 등) 외에 담당자-Comment, Reporter-Comment 를 추가 수집 및 정제하여 버그 분류의 정확도를 향상시킨다.

Empirical Case 초기 테스트 결과

- 기본 제공하는 데이터 (Summary)만 사용시 정확도: 약 55%
- 기본 제공하는 데이터 (Summary) + Owner-Comment + Reporter-Comment 사용시 정확도: 약 70%

1.2.3. 안드로이드 이슈에 최적화된 *Stop Words* 도출

주석 중 개발자들이 반복적으로 사용하거나 대부분의 기능에서 공통적으로 사용되는 단어 목록을 추출하여 제거함으로써 버그 분류 정확도를 향상시킨다.

- *Stop Words* 목록 제거 전 정확도: 36%
- *Stop Words* 목록 제거 후 정확도: 40%

본 논문의 구성은 2장에서 관련 연구를 설명하고 3장에서 담당자 선정 프로세스에 대해 언급한다. 4장에서는 실험 및 결과를 고찰한 뒤. 이어지는 5장에서 결론을 도출하고 마지막 6장에서 향후 연구에 대해서 기술한다.

2. 관련 연구

현재까지 이클립스, 모질라 등의 오픈소스 프로젝트를 기반으로 버그 분류에 대한 연구가 지속되어왔다. 다른 오픈소스 프로젝트에 비해 역사가 짧은 안드로이드는 그 버그가 공개되지 않아 연구에 필요한 데이터 확보가 어려웠고 이로 인해 버그 분류에 대한 연구는 상대적으로 많이 이루어지지 않았다. 관련 연구는 크게 비 안드로이드 버그 분류와 안드로이드 버그 분류로 구분한다.

2.1 비 안드로이드 버그 분류

Park Seong Hun의 연구에서는 코사인 유사도 이용하여 버그에 대한 개발자 추천 모델을 제안하고 있다. 이클립스의 오픈소스 프로젝트를 대상으로 수행된 개발자 할당 실험에서 기계학습 모델 기반의 할당 방법보다 제안하는 방법이 더 나은 결과를 얻은 것을 입증하고 있어, 이 두 모델을 비교하기에 유용하다. 하지만, 오픈소스 프로젝트에만 적용된 방법으로써 제안하는 방법을 다른 영역으로 확장해서 적용해 볼 필요가 있다.[1]

Gaeul Jeong의 연구에서는 오픈소스 결함 할당의 이력 정보에 Markov Chain 을 적용하고 개발자 별 재할당 확률 값에 따른 재할당 경로를 결정하는 모델을 제안하고 있다. 이 모델은 연관된 기능을 가진 개발자들의 그룹을 식별함으로써 새로운 작업을 위한 담당자들을 할당하는 데에 도움을 준다. 또한, 버그 Tossing이 필요할 때 자동으로 적합한 담당자들을 찾아주어 불필요한 Tossing이 줄어들고 있음을 증명하고 있다. 하지만 이 모델에는 더 이상 버그를 수정하지 않는 퇴직한 개발자의 정보는 고려하지 않는다는 한계점이 있다.[2]

Zhongpeng Lin의 연구에서는 *SVM*(Support Vector Machine) 기반 분류 알고리즘을 적용하여 중국어 버그 데이터로 실증 연구를 수행하였다. 그들은 중국어의 비 문자 데이터가 중국어 버그 할당에 가장 유용한 데이터임을 증명하고 있다. 하지만 분류 기법을 기반으로 하고 있는 이 논문에서의 접근법이 다른 중국 프로젝트에서 적용이 가능한지에 대해 증명할 타당도가 부족하다. [3]

John Anvik의 연구에서는 기계 학습을 이용한 Semi-Automated 버그 분류 방법을 제안하고 있다. 제안한 방법을 통해 이클립스와 파이어폭스의 소프트웨어 프로젝트에서는 50% 이상의 정확도를 달성했지만, gcc [2] 의 경우 6%의 낮은 정확도를 보였다. gcc 분야에서도 정확도를 높일 수 있는 방안에 대해서 깊이 있는 고찰이 필요하다.[4]

V.Akila의 연구에서는 목표-지향 경로 모델의 버그 분류보다 실제 경로 모델에 기반을 둔 농축 적응(Enriched Adaptive) 버그 분류 시스템의 재할당을 제안하고 있다. 제안된 시스템은 버그를 수정하는 과정에서 버그를 해결하는데 상당한 기여를 했을 중간 단계의 개발자가 수행한 모든 작업을 보존함으로써 주어진 버그에 대해 협력할 수 있는 최적의 개발자를 찾을 수 있음을 증명하고 있다.[5]

2.2 안드로이드 버그 분류

Pamela Bhattacharya 의 연구에서는 버그 보고서의 품질 기준을 정의하고 개발자 선정을 포함한 버그 해결 과정을 분석하는 실증 연구를 제시하고 있다. 상대적으로 실증 연구가 부족한 안드로이드 플랫폼 에서의 버그 보고서의 품질 수준을 분석하고 경향을 파악하는데 의미가 있는 연구이다. 보안 관련 버그가 비 보안 관련 버그보다 보고서 품질이 좋지만 버그 해결은 늦는 현상을 발견하였다. 하지만, 이에 대한 원인 분석이 뒷받침되지 않는다.[6]

Grzegorz Chrupala의 연구에서는 첫째로, 소프트웨어 프로젝트 별로 수집된 데이터로부터 시간에 따라 버그 성분 변화를 분석하였고 시각화된 자료를 제공함으로써 버그 성분 동향 분석에 유용하다. 둘째로, 온라인 학습 알고리즘 [3] 의 효율성을 평가하고 있다. 소프트웨어 프로젝트[4] 별로 알고리즘 [5] 을 적용 및 평가하여 주로 회귀 알고리즘의 성능이 좋음을 실험 결과로 보여주고 있다. 하지만 이유에 대한 설명이 부족하여 아쉬움이 남는다.[7]

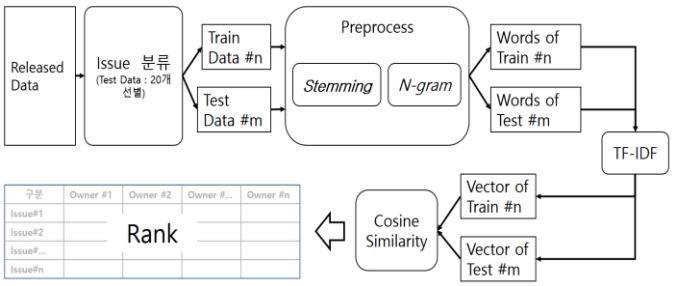
Alipour의 연구에서는 소프트웨어 품질, 소프트웨어 아키텍처, 시스템 개발 주제들의 문맥상 정보를 조사하여 적용함으로써 중복 버그의 탐지율을 높이는 방안을 제안하고 있다. 이는 기존 버그 분류 방법에서는 시도하지 않았던 문맥적 접근 방법으로서 의미 있는 정보이다.[8]

Victor Guana et al. 의 연구에서는 안드로이드 커뮤니티의 버그 관심도에 따라 보고자를 분류하였으며 선호도(Stars) 값이 높을수록 특정 영역(layer)이 개발자와 사용자에게 더 중요함을 확인하였다. 버그 보고자 분류 기준을 보고서 횟수와 주석 횟수로 설정하였으나 기준의 타당성이 설명되지 않아 연구 결과를 뒷받침하기 어려운 단점이 있다. 안드로이드 커뮤니티에서 가장 관심이 있는 영역이 프레임워크 임을 실험 결과를 통해 제시하고 있다. 하지만 안드로이드 어플리케이션 개발자들의 참여가 활발하다는 개인적인 의견을 전제로 하고 있어 타당성이 부족하다.[9]

Kevin Moran의 연구에서는 기존 버그 추적 시스템이 버그 해결을 위한 정보를 효율적으로 제공하지 못하여 FUSION이라는 자동완성기능을 지원하는 보고서 시스템을 제안하고 있다. 기존에 사용되고 있는 버그 보고서 시스템들을 분석하여 버그 보고서의 필수적인 정보들을 정리한 것은 버그 분류의 개선에 대한 좋은 자료이다. 하지만 자동화된 보고서 시스템이 도입되더라도 이슈 설명서 등 사람이 작성하는 부분은 모호성이 존재하기 마련인데 이를 해소할 수 있는 방안이 없다는 점에선 아쉬움이 남는다.[10]

3. 버그 담당자 선정 프로세스의 제안

본 연구에서는 기존 연구 사례[3],[4]를 참고하여 버그 담당자 선정 프로세스를 수립하였다. 각 개발자가 과거에 완료한 버그 설명서의 이력을 TF-IDF 과정을 통해 벡터 값으로 변환하고, 이것을 새로 생긴 버그 설명서의 벡터와 코사인 유사도를 계산하여 유사도가 가장 높은 개발자를 버그 담당자로 지정하는 방식이다. 워크 플로우는 4장 실험을 통해 보완 및 발전시킨다.



<그림 6> 워크 플로우

[2] http://www.gnu.org/software/gcc/index.html
[3] Window Frequency Baseline, SVM Minibatch, Bugzie 등
[4] Chromium, Android, Firefox, Launchpad
[5] Window, SVM, Perceptron, Bugzie, Regression

3.1 전처리

전처리는 크게 *Stop Words*, *Stemming*, *Lemmatization*, *N-gram* 과정을 거친다.

3.1.1 *Stop Words*

'*Stop Words*'란 자연어 데이터처리 전 또는 후에 제거되는 단어들을 지칭한다. 이를 위해서 *Scikit Learn API*를 사용하여 Common English 단어 목록을 먼저 제거한다. 그리고 안드로이드 오픈 소스 특성을 고려하여, 주석 중 개발자들이 반복적으로 사용하거나 대부분의 기능에서 공통적으로 사용되는 단어 목록을 추가적으로 추출한다. 단어 목록은 담당자가 버그 수정을 위해 필요로 하는 기본 정보를 요청하는 내용이다. 재현 경로, 재현 빈도, 기대하는 동작, 실제 동작에 대한 질문 등이 이에 해당하며, <그림7>과 같다.

```
cwords = """Steps to reproduce ₩n
What steps do others need to take in order to reproduce the issue themselves?₩n
Please explain the issue in detail.₩n

Frequency₩n
How frequently does this issue occur? (e.g 100% of the time, 10% of the time)₩n

Expected output₩n
What do you expect to occur?₩n

Current output₩n
What do you see instead?₩n
```

<그림 7> Android *Stop Words* 예시

3.1.2 *Stemming*[6]

'*Stemming*'이란 어형이 변형된 단어로부터 접사 등을 제거하고, 그 단어의 어간을 분리해 내는 것을 지칭한다. 예를 들면 automates, automatic, automation의 단어에서 automat과 같이 어간을 분리해 내는 것을 말한다.

<표 1>. *Stemming*과 *Lemmatization* 적용 예시

구분	Porter	Lancaster	Lemmatization
"presumably"	presum	presum	presumably
"multiply"	multipli	multiply	multiply
"provision"	provis	provid	provision
"wolves"	wolv	wolv	wolf
"owed"	owe	ow	owed
"are"	are	ar	are
"abacus"	abacu	abac	abacus

'*Stemming*'은 Porter Stemmer, Lancaster Stemmer 방법이 있다. 'Porter Stemmer'란 Martin Porter가 제작한 *Stemming* 알고리즘으로, 단어의 접미사를 제거하거나 다른 단어로 대체하는 역할을 한다. 'Lancaster Stemmer'는 Lancaster 대학교에서 개발한 *Stemming* 알고리즘으로, 'Porter Stemmer'와 유사하나 좀 더 성능이 좋다. 이 방법을 적용하는 가운데에 있어서 Natural Language Toolkit API를 사용한다.

3.1.3 *Lemmatization*

'*Lemmatization*'이란 '*Stemming*'과 매우 유사하나 단어 자체만을 고려하는 '*Stemming*'과는 달리 그 단어가 문장 속에서 어떤 품사로 쓰였는지까지 판단한다. 이 방법을 적용하는 가운데에 있어서 Natural Language Toolkit API를 사용한다.

3.1.4 *N-gram*

'*N-gram*'은 주어진 문자열을 연속된 N개의 단위로 절단하는 것을 지칭한다. 이를 위해서 *Scikit Learn API*를 사용한다.

<표2> *N-gram* 적용 예시

Input : "I want to see a movie"	
N-gram	Output
1-gram	"I", "want", "to", "see", "a", "movie"
2-gram	"I want", "want to", "to see", "see a", "a movie"
3-gram	"I want to", "want to see", "to see a", see a movie"

3.2 단어의 중요도 계산

단어의 중요도 계산을 위해서는 단어를 벡터로 변환하는 과정과 단어의 특정 가중치를 부여하는 것이 필요하다.

아래 식 (1)은 단어의 빈도수를 중요도로 변환하는 것이다.[11]

$$TF\text{-}IDF(w,d) = TF(w,d) \times IDF(w,D) \quad (1)$$

여기서,

w: 단어

d: 담당자의 설명서

D: 모든 담당자의 설명서

TF(*w*, *d*): 담당자의 설명서 *d* 에서 단어 *w* 의 빈도수

IDF(*w*, *D*): 모든 설명서 *D* 에서 단어 *w* 의 중요도

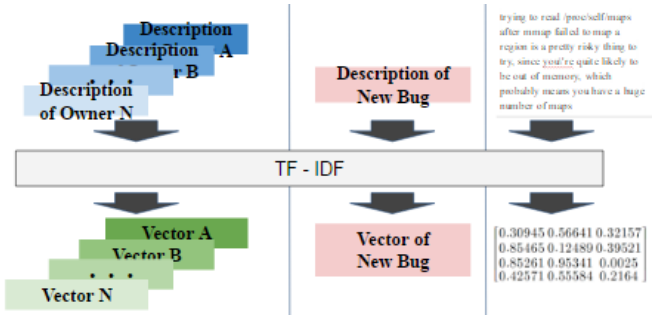
TF(Term Frequency)는 특정 단어가 특정 문서에 나타나는 빈도수를 나타낸다. IDF(Inverse Document Frequency)는 전체 문서에서 특정 단어의 중요도를 의미하고 식 (2)를 통해 값을 구한다.

$$IDF(w,D) = \log\left(\frac{|D|}{|w \in D : w \in d|}\right) \quad (2)$$

전처리 과정을 거친 담당자 설명서와 새로운 버그

[6] https://en.wikipedia.org/wiki/Stemming

설명서에 대해서 각각 *TF-IDF* 과정을 거쳐 담당자의 벡터와 새로운 버그의 벡터를 구한다.



<그림 8> *TF-IDF* 과정

3.3 버그 담당자 선정

분류 알고리즘의 후보로 *CART*[8]와 코사인 유사도[1]를 선정하고 그 실험결과를 근거로 코사인 유사도를 적용한다.

식 (3)은 0이 아닌 두 벡터의 내적을 계산하고 코사인 값을 측정한다.

$$\vec{a} \cdot \vec{b} = |a| \times |b| \times \cos(\theta)$$
$$\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{|a||b|} \quad (3)$$

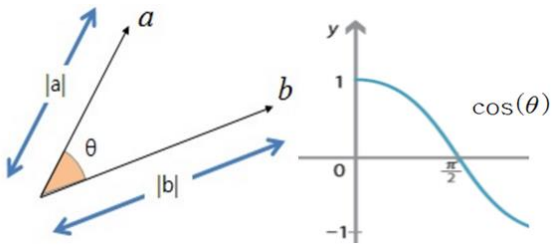
담당자의 설명서와 버그의 설명서의 코사인 유사도를 식 (4)로 측정한다.[12]

$$\text{코사인 유사도 } (d, b) = \frac{\sum_{i=1}^{n} d[i] \times b[i]}{\sqrt{\sum_{i=1}^{n} (d[i])^2} \times \sqrt{\sum_{i=1}^{n} (b[i])^2}} \quad (4)$$

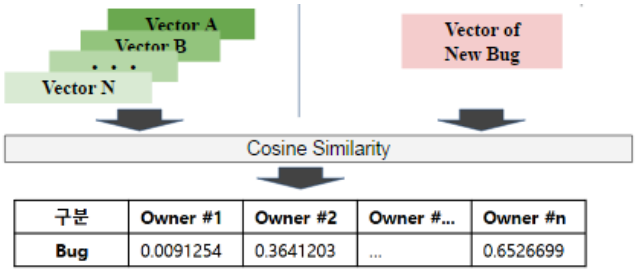
여기서,

d: 담당자의 설명서
b: 버그의 설명서
n: 전체 단어의 수
d[*i*]: 담당자의 *i*번째 단어의 *TF-IDF*
b[*i*]: 버그의 *i*번째 단어의 *TF-IDF*

코사인 유사도는 <그림9>에서와 같이 코사인 값이 1에 가까울수록 두 벡터의 유사도가 높다.



<그림 9> 두 벡터와 Cosine 곡선



구분	Owner #1	Owner #2	Owner #...	Owner #n
Bug	0.0091254	0.3641203	...	0.6526699

<그림 10> 코사인 유사도 구하는 과정

4. 실험 및 평가

효율적인 버그 분류를 위한 방법론을 수립하고 안드로이드에 최적화된 모델을 도출하고자 한다. 이를 위하여 최소한의 데이터를 활용하여 Trial Case Study를 먼저 수행하고, 완결성과 유효성이 검증된 데이터를 확보 및 추가하여 Empirical Case Study를 수행하였다.

4.1 실험 데이터

본 연구에 사용된 데이터는 안드로이드 오픈소스 프로젝트 기반의 버그 보고서 시스템인 안드로이드 이슈 추적 시스템에 등록된 버그로부터 획득 활용하였다.

A. 데이터 참조: *Nexus*

안드로이드 장치의 버그 분류에 대한 연구를 위해 안드로이드 관련 휴대폰인 *Nexus*가 언급된 버그로 한정하도록 하였다.

B. 버그 상태

총 14가지 상태 중 4가지 상태(Released, Future Release, Duplicated, WorkingAsIntended)의 데이터를 선정한다. 선정 기준은 버그 분류가 완료되어 더 이상 상태 변경이 없고(완결성), 이슈 설명서가 버그 해결에 유효한 내용의 비중이 높은(유효성) 데이터로 선정하였다.

<표 3> 이슈 상태 별 개수

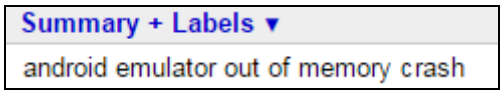
Status	Total	Declined	Duplicated	Future Release
개수	39,730	838	2,508	672
Status	NeedInfo	NotEnough Information	Obsolete	Question
개수	427	1,949	13,252	101
Status	Released	Spam	Unreproducible	UserError
개수	1,478	6,951	454	165
Status	Wrong Forum	New	WorkingAs Intended	
개수	5,450	4,729	756	

C. 이슈 설명서 정보 추가

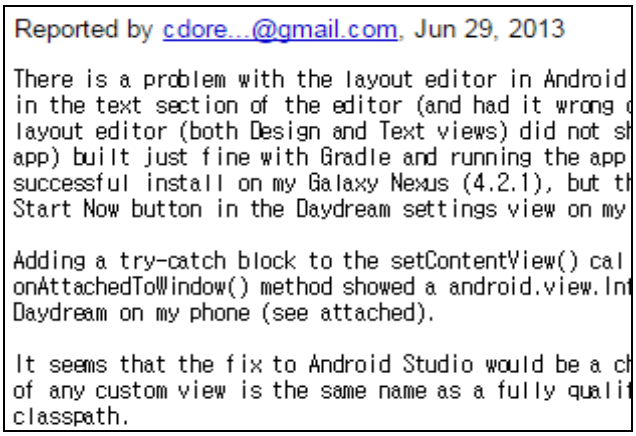
유사 연구[8]에서 쓰였던 Summary외에 Owner-Comments, Reporter-Comments를 추가한다. 주석을 사용한 이유는 충분한 문자 데이터로부터 버그 분류에 유용한 정보를 확보할 수 있기 때문이다.

<표 4> 이슈 설명서 종류에 따른 데이터 크기

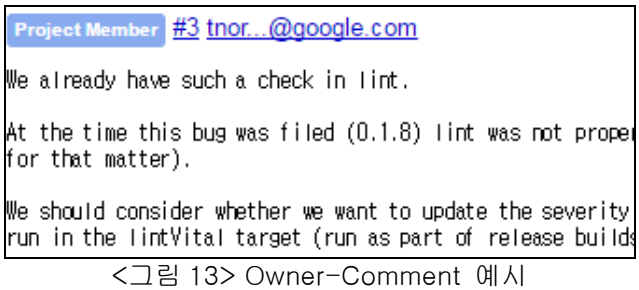
Status	Summary	Summary+Comment
	2,649KB	**76,342KB**
Released	918KB	32,746KB
Future Released	655KB	24,571KB
Duplicated	537KB	7,171KB
WorkingAsIntended	539KB	11,854KB



<그림 11> 안드로이드 이슈 추적 시스템의 Summary 예시



<그림 12> Reporter-Comment 예시



<그림 13> Owner-Comment 예시

4.2 Trial Case

4.2.1 실험 절차

Trial Case를 통해 기존 연구 사례[3],[4]들을 참고하여 실험을 거쳐 방법론을 정의하고 버그 분류 Trial 모델을 생성하고자 한다.

Trial Case의 워크 플로우는 <그림6>과 같이 문자 데이터 분석에서 보편적으로 사용되고 있는 방법인 전처리 과정(*Stop Words*, *Stemming*, *N-gram*)과 *TF-IDF*를 적용한다.

분류 알고리즘으로는 *CART*와 코사인 유사도를 후보로 선정하고 정확도 측정 결과 코사인 유사도가 우세하여 본 연구의 분류 알고리즘으로 선정한다.

<표 5> 알고리즘에 따른 정확도 비교

알고리즘	정확도
CART	5%
Cosine Similarity	30%

4.2.2 입력 데이터

Trial Case는 안드로이드 이슈 추적 시스템에서 제공하는 데이터중에서 담당자가 기록되고 상태가 Released인 데이터 중 Summary 내용만을 이용하여 버그 분류를 시도한다.

실험 데이터는 할당된 버그의 개수가 5개 이상인 담당자중에서 임의로 20개의 버그 추출한다.

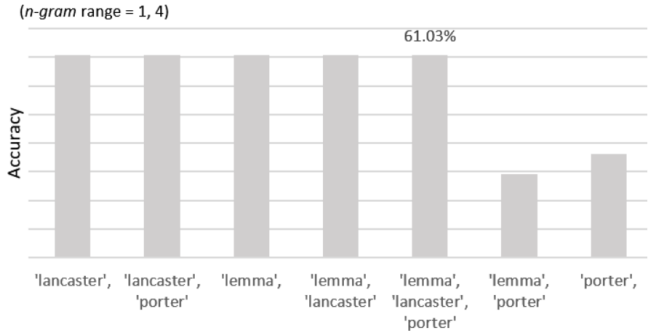
<표 6> Trial Case 에 사용된 데이터 현황

Data	Number of 데이터	Status	Used Information
Train Data	1,458 개	Released Data (Include Owner)	Summary
Test Data	20 개		

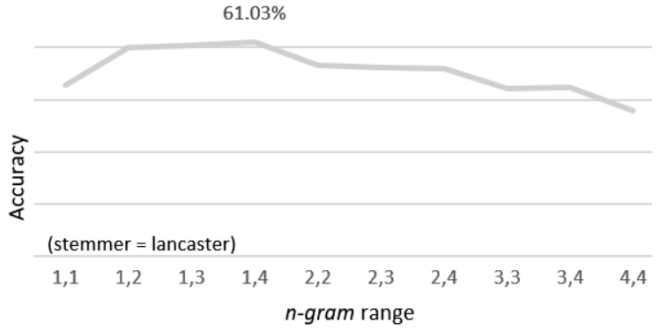
4.2.3 실험 결과 및 고찰

전처리를 위해 파이썬에서 제공하는 *Scikit Learn API* 중 *Stop Words*, *Stemming*과 *N-gram*을 활용한다.

<그림14>과 <그림15>는 *Stemming*과 *Lemmatization* 의 조합과 *N-gram* 범위에 따라 변화하는 정확도를 나타낸다.



<그림 14> *Stemming* 과 *Lemmatization* 의 조합에 따른 정확도



<그림 15> *N-gram* 범위에 따른 정확도

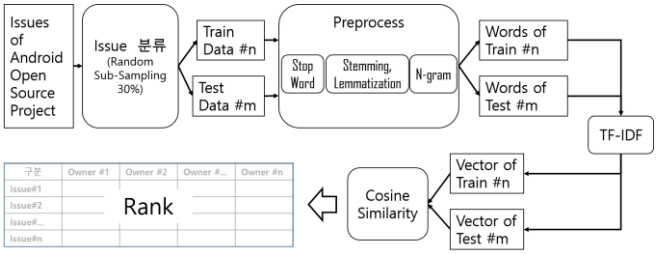
Trial Case에서 버그 분류 정확도는 최고 61.03%로서 본 연구에서 목표로 했던 80%보다 현저히 낮은 수준으로 측정된다. 사용한 데이터의 양이 충분하지 않아서 정확도에 대해서도 신뢰도가 낮을 것으로 판단된다.

이번 Trial Case에서의 워크 플로우는 신뢰도 향상을 위한 추가적인 작업이 필요하지만 안드로이드 버그 분류 과정에서 담당자 선정에 유의한 모델임을 확인한 것이다.

4.3 Empirical Case

4.3.1 실험 절차

Trial Case를 통해서 도출된 모델을 개선하여 <그림16>과 같은 워크 플로우를 제안한다.



<그림 16> Empirical Case 워크 플로우

4.3.2 입력 데이터

각 버그에 있는 Summary외에 정보 항목에서 Owner-Comment과 Reporter-Comment 데이터를 추가한다. 이슈 상태는 Released외에 FutureRelease, WorkingAsIntended, Duplicate 데이터를 추가하여 총 5,414개의 데이터를 확보하게 된다. 이 중 담당자 기록이 누락된 데이터를 제거한 2,636개를 활용하는데 테스트 데이터는 담당자 별 할당된 버그 수량을 고려하여 각 30%의 버그를 무작위로 샘플링한다.

담당자 69명 중 코사인 유사도가 가장 높은 1명과 이슈의 실제 담당자가 일치한다면 예측 성공한 것이다. 정확도는 791개의 테스트 데이터 중 워크 플로우를 수행하여 담당자를 예측하였을 때 실제 담당자와 일치한 개수의 비율로 측정하였다.

<표 7> Empirical Case 에 사용된 데이터 현황

Data	Number of Data	*Issue Status*	Used Information
Train	1,845 개	Released, FutureRelease, Duplicated, WorkingAsIntended (Include 담당자)	Summary, Owner-Comment, Reporter-Comment
Test	791 개		

<그림 16>의 워크 플로우와 데이터 형태를 고려했을 때 <표 8> 기준으로 계산해 보면 전체 경우의 수는 3,360 가지이다. (15×2×4×4×7 = 3,360 cases)

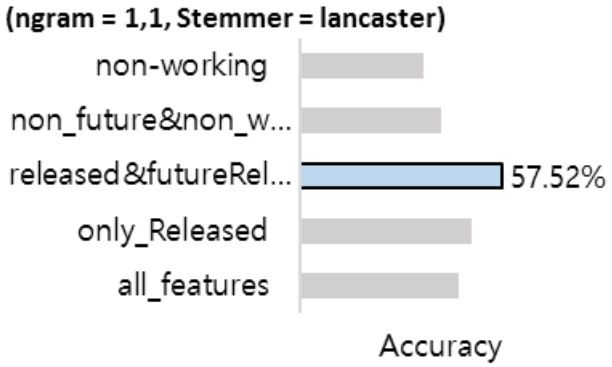
1회 Test시간은 74.27초, 총 테스트 시간은 약 70시간이 소요되었다.

<표 8> 모든 경우의 수

Issue Status	Information	*N-gram*		*Stemmer*
		Min	Max	
Released, FutureRelease, Duplicated, WorkingAsIntended	Summary + Owner-Comment, Summary + Reporter-Comment	1~4	1~4	*Lemmatization, Lancaster*
15case	2case	4case	4case	7case

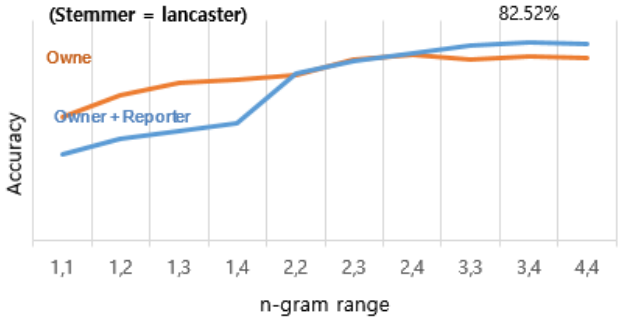
4.3.3 실험 결과 및 고찰

데이터가 많을수록 정확도가 높게 나타날 것이라는 예상과 달리, Released와 FutureRelease 데이터만을 사용했을 때 다른 그룹에 비해서 정확도가 월등히 높게 나타남을 확인할 수 있다. 그 이유는 <표7>에서와 같이 데이터 량이 추가 확보됨으로써 구축된 모델이 충분히 Train될 수 있었기 때문이다. 또한, WorkingAsIntended, Duplicated 그룹에는 설명서가 부실하거나 부적절한 부분이 많았고 주석 횟수도 충분하지 않았기 때문에 정확도 향상에 기여하지 못했던 것으로 분석된다.



<그림 17> 버그 이슈 상태에 따른 정확도

본 연구에서 제안한 방법에서는 버그 이슈 상태에 따른 정확도를 토대로 *N-gram* 범위에 따른 정확도가 측정된다.



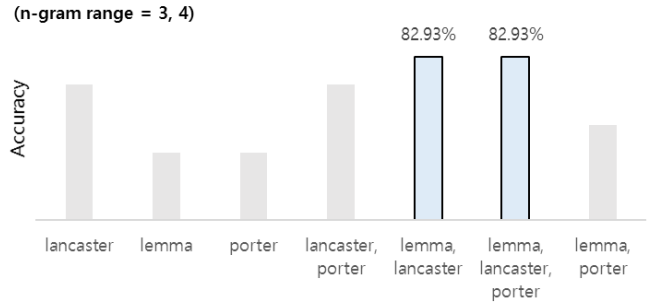
<그림18> *N-gram* 범위에 따른 정확도

<그림18>에서와 같이 안드로이드 이슈 설명서에서 가장 적절한 *N-gram* 범위는 (3,4)이다.

N-gram 범위를 (3,4)로 고정하고 *Stemming*과 *Lemmatization*의 조합에 따라 달라지는 정확도를 측정한 것이 <표9>이다.

<표 9> *Stemming* 과 *Lemmatization* 의 조합에 따른 정확도

Data	*N-gram* range	*Stemming, Lemmatization*	Accuracy
Owner + Reporter	3,4	*Lancaster*	82.52%
Owner + Reporter	3,4	*Lemmatization*	81.50%
Owner + Reporter	3,4	*Porter*	81.50%
Owner + Reporter	3,4	*Lancaster, Porter*	82.52%
Owner + Reporter	3,4	*Lemmatization, Lancaster*	82.93%
Owner + Reporter	3,4	*Lemmatization, Lancaster, Porter*	82.93%
Owner + Reporter	3,4	*Lemmatization, Porter*	81.91%



<그림 19> *Stemming* 과 *Lemmatization* 의 조합에 따른 정확도

3,360가지의 모든 경우의 수에 대해서 <표10>과 같은 Option을 적용하여 정확도 82.93%라는 결과가 도출된다.

<표 10> 안드로이드 이슈의 최적 파라미터

Status	Information	*N-gram*		*Stemming*
		Min	Max	
Released & FutureRelease	Summary & Owner-Comment & Reporter-Comment	3	4	*Lemmatization* & *Lancaster*

*Lemmatization*과 *Lancaster*를 함께 적용했을 때 정확도가 높게 나온 이유는 *Lemmatization*이 단어를 문맥적 의미 단위로 *Stemming*하는 방법이고, *Lancaster*는 *Porter*보다 빅데이터 분석에 호전적인 *Stemming* 방법이기 때문에 이 둘의 조합이 안드로이드 이슈 설명서 특징에 적합한 방법으로 작용한 것으로 해석된다.

5. 결론

본 연구는 안드로이드 오픈소스 프로젝트에서 최초 시도된 *NLP* 알고리즘을 적용한 버그 분류 자동화 방안에 대한 연구이다.

이클립스와 같이 버그 분류 에 관한 연구가 활발했던 OSP에 비해 안드로이드는 역사가 짧아서 버그 데이터 양이 적고 버그 분류 자동화에 대한 연구가 상대적으로 부족하였다.

본 연구에서는 안드로이드 버그 분류 자동화에 적합한 데이터 선정 방법(Summary, Owner-Comment, Reporter-Comment)과 안드로이드 이슈 설명서에서 공통적으로 제거되어야 하는 *Stop Words*를 제안한다.

다양한 실험을 통해 안드로이드 버그 분류에 가장 적합한 *Stemming과 Lemmatization, 그리고 N-gram* 범위(3,4)를 확인할 수 있었다.

이슈 상태 중 Released와 FutureRelease 데이터의 설명서가 버그 분류에 유용한 단어의 빈도가 높고 주석의 양도 상당량 존재했다. 실험 결과 82.93%라는 정확도를 도출해 냄으로써 본 논문에서 제안하는 모델이 안드로이드 영역에서의 버그 분류에 적절함을 확인할 수 있었다.

6. 향후 연구

TF-IDF 과정에서 TF 계산을 위해 문자 데이터를 활용함에 있어서 빈도가 높은 단어의 순위를 확인하고 버그 분류에 영향을 미치지 않는 단어를

*Stop Words*에 추가하는 방안이 지속 강구되어야 할 것이다.

또한 버그 설명서가 부실하거나 주석 횟수가 적은 경우에 대해서도 정확한 분류를 위한 시도가 계속 되어야한다.

아울러 연구 진행 중에 안드로이드 이슈 추적 시스템에서 활동하는 담당자들 중 특정 기간에 따라 버그 수정에 더 이상 참여하지 않거나 새로 참여하는 담당자를 다수 식별하였다. 이는 동일 기능에 대한 담당자가 빈번히 변경되는 현실을 보여준다. 동일 기능의 담당자가 변경될 경우에 대해서도 버그 분류의 정확도를 높일 수 있는 연구가 필요할 것이다.

7. 참고문헌

[1] S. H. Park, J. I. Kim, and E. J. Lee, "A Technique to Recommend Appropriate Developers for Bug Reported Bugs Based on Term Similarity and Bug Resolution History," *KIPS Transactions on software and Data Engineering*, vol. 3, pp. 511-522, 2014.

[2] G. Jeong, S. Kim, and T. Zimmermann, "Improving Bug Triage with Bug tossing graphs," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 111-120.

[3] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on Bug assingment automation using Chinese Bug Data," in *2009 3rd International Symposium on Empirical software Engineering and Measurement*, 2009, pp. 451-455.

[4] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this Bug?," in *Proceedings of the 28th international conference on software engineering*, 2006, pp. 361-370.

[5] V. Akila, G. Zayaraz, and V. Govindasamy, "Effective Bug Triage–A Framework," *Procedia Computer Science*, vol. 48, pp. 114-120, 2015.

[6] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of Bug reports and Bug fixing in open source android apps," in *software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, 2013, pp. 133-143.

[7] G. Chrupala, "Learning from evolving Data streams: online Triage of Bug reports," in *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 2012, pp. 613-622.

[8] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate Bug report detection," in *Proceedings of the 10th Working Conference on Mining software Repositories*, 2013, pp. 183-192.

[9] V. Guana, F. Rocha, A. Hindle, and E. Stroulia, "Do the stars align?: multidimensional analysis of Android's layered architecture," in *Proceedings of the 9th IEEE Working Conference on Mining software Repositories*, 2012, pp. 124-127.

[10] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing Bug reports for android applications," in *Proceedings of the 2015 10th Joint Meeting on Foundations of software Engineering*, 2015, pp. 673-686.

[11] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate Bug reports using natural language and execution information," in *Proceedings of the 30th international conference on software engineering*, 2008, pp. 461-470.

[12] P.-N. Tan, S. Michael, and V. Kumar, "Chapter 6. association analysis: Basic concepts and Algorithms," *Introduction to Data Mining. Addison-Wesley. ISBN*, vol. 321321367, 2005.

컴포넌트 단위 학습을 이용한 버그 담당자 추천

조충기 ○, 김영민, 이기성, 이찬근

중앙대학교 컴퓨터공학부

cndrldhQk@gmail.com, remnant1120@gmail.com, goory00@gmail.com, cglee@cau.ac.kr

Bug Triage via Component-based Learning

Choong-Ki Cho○, Young-Min Kim, Ki-Seong Lee, Chan-Gun Lee

School of Computer Science and Engineering, Chung-Ang University

요 약

소프트웨어 버그는 개발과 테스트 그리고 유지보수를 포함하는 소프트웨어의 전 생명 주기에서 끊임없이 발생한다. 버그를 효율적으로 처리하는 것은 소프트웨어 프로젝트의 성패에 중대한 영향을 미친다. 버그를 처리하기 위해 가장 먼저 수행해야 하는 작업은 해당 버그를 해결할 수 있는 적절한 개발자를 찾는 일이다. 많은 개발자가 참여하는 대규모 프로젝트에서는 하루에도 수많은 버그가 발생하기 때문에 이러한 버그 담당자 할당(Automated Bug Triage)은 큰 비용이 드는 어려운 과업이며, 또한 최근 여러 연구에서 버그 처리 비용을 증가시키는 주된 원인 중 하나로 보고되고 있다. 본 논문에서는 효과적인 버그 담당자 추천을 위해 컴포넌트 단위 학습을 사용하는 방법을 제안한다. 컴포넌트값을 기준으로 전체 버그 리포트를 나누고 각각 분할된 리포트 그룹들을 훈련 데이터로 사용하여 기계학습모델을 구축한다. 본 연구에서 수행한 사례연구결과 컴포넌트 단위 학습을 통한 개발자 추천 모델이 전체 데이터로 학습된 추천모델에 비해 최대 20%가 넘는 성능향상을 보인다는 사실을 확인할 수 있었다.

1. 서 론

버그는 개발과 테스트 그리고 유지보수를 포함하는 소프트웨어의 전 생명 주기에서 끊임없이 발생한다. 버그를 효율적으로 처리하는 것은 소프트웨어 프로젝트의 성패에 중대한 영향을 미친다. 효과적인 버그 관리를 위해 대규모 소프트웨어 프로젝트를 중심으로 버그 추적 시스템(Bug Tracking System)의 도입이 보편화 되고 있다. 해당 시스템은 버그 저장소(Bug Repository)를 포함하는데 그 저장소에는 발견된 버그가 해결될 때까지 발생하는 모든 관련 데이터들이 저장되며 또한, 그들을 열람하고 관리하는 기능을 제공한다.

버그 추적 시스템은 각 버그에 관한 정보를 버그 리포트(Bug Report)라는 형태로 관리한다. 버그 리포트는 여러 가지 항목(field)으로 구성되어 있으며 각 항목이 가질 수 있는 데이터 형태에 따라 크게 자연어 항목(Textual field)과 범주화 항목(Nominal field)으로 구분할 수 있다.

소프트웨어 공학 관련 학계에서는 버그 저장소를 활용한 다양한 연구들이 등장하고 있다. 그중 대표적인 것으로 자동 버그 담당자 할당(Automated Bug Triage)을 들 수 있다. 버그 담당자 할당은 식별된 버그에 관한 적절한 처리를 위해 가장 먼저 수행돼야 하는 작업으로 그 버그를 해결할 수 있는 적절한 개발자를 찾는 작업이다. 이는 얼핏 보면 간단한 문제라고 생각할 수 있으나, 대규모 프로젝트의 경우 하루에도 수십 건 이상의 버그 리포트가 생성되기 때문에 많은 개발자 중 해당 버그 리포트를 담당할 개발자를 찾는 일은 결코 쉬운 문제가 아니다. 또한, 여러 연구에서 버그 처리에 관한 비용 중 많은 부분이 버그 담당자 할당에 소비된다고 보고하고 있다 [1, 2, 7]. 자동 버그 담당자 할당 연구는 이러한 문제를 해결하기 위해 버그 저장소에 있는 버그 해결에 관한 이력 정보를 활용하여 효과적으로 버그 담당자를 찾는 기법들은 제안하고 있다 [1, 2, 4, 6, 7, 9, 12].

기존에 제안된 많은 관련 연구들은 기계학습 알고리즘을 사용하여 버그 담당자 할당 문제를 다루고 있다 [1, 2, 4, 6, 7]. 버그 저장소에 존재하는, 이미 해결된 버그 리포트의 텍스트 정보들을 사용해 기계학습모델을 구축한 다음 새로운 버그가 발생했을 때 적절한 개발자는 추천해주는 방식이다. 이러한 접근은 텍스트 마이닝 (Text mining)이나 정보 추출(Information retrieval)에서 사용되는 방식과 매우 유사하다.

또 다른 접근으로 버그 리포트의 범주화 필드를 활용하는 방법도 존재한다. Wang 등은 버그 리포트의 여러 항목 중 컴포넌트(Component)를 사용하여 버그 담당자를 추천하는 방법을 제안하였다 [11]. 버그 리포트의 컴포넌트 항목은 user interface 또는 network protocols과 같이 제품이 제공하는 특정 기능을

구현하기 위한 부품 혹은 부분을 의미한다. 소프트웨어의 경우에는 특정 기능을 제공하는 소스코드 모듈로 생각할 수 있다. 해당 연구의 저자들이 Eclipse Platform과 JDT 프로젝트의 버그 리포트를 분석한 결과, 한 개발자가 해결한 버그 리포트들의 컴포넌트 항목의 종류는 평균적으로 2개가 채 안 된다는 사실을 발견하였다. 이는 개발자들은 소수의 컴포넌트와 관련된 버그만을 수정하는 경향이 있다는 사실을 시사한다. 이와 같은 컴포넌트와 개발자 사이의 관계를 통한 담당자 추천은 복잡한 기계학습 알고리즘을 사용하지 않고도 의미 있는 담당자 추천 결과를 보여주었다.

학습모델을 통한 접근 방법은 버그 담당자 할당뿐만 아니라 소프트웨어 공학과 관련된 여러 문제 해결에 사용되고 있다. 예컨대, 다수의 소스코드 파일 중 버그가 발생할 파일을 예측하는 방법을 연구하는 소프트웨어 결함 예측(Defect Prediction) 분야는 기계학습모델을 활발히 사용하는 것으로 유명하다. 해당 분야에서는 2010년대 초반부터, 전체 데이터를 사용해 하나의 학습모델을 만드는 것보다 전체 데이터를 유사한 특성을 지닌 부분들로 나눈 다음, 각 부분의 데이터를 통해 학습모델을 구축하는 것이 유리하다는 연구가 발표되기 시작했다 [3, 8, 10]. 이러한 일련의 연구들은 소프트웨어 관련 데이터의 이질성(Heterogeneity)을 언급하며 이러한 이질적인 성질의 데이터를 통해 만들어진 학습모델은 견고한 성능을 보이기 어렵다고 지적하고 있다. 그들은 클러스터링과 같은 방법을 통해 전체 데이터를 서로 유사한 특성을 지닌 그룹으로 나눈 뒤 그것들을 사용해 학습모델을 구축하는 방법을 제안하고 있다.

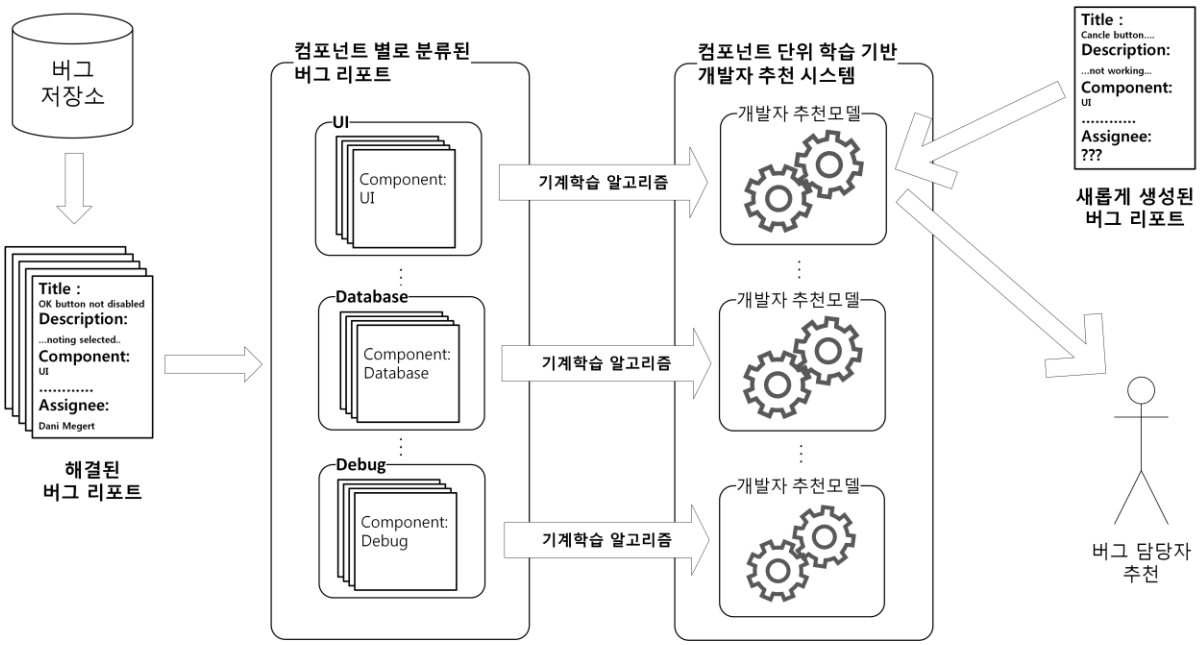


그림 1. 컴포넌트 단위 학습 기반의 버그 담당자 추천

본 논문에서는 효과적인 버그 담당자 추천을 위해 컴포넌트 단위 학습을 사용하는 추천 기법을 제안한다. 컴포넌트값을 기준으로 전체 버그 리포트를 여러 그룹으로 나눈 다음 각각 분할된 그룹들을 훈련 데이터로 사용하여 기계학습모델을 구축한다. 이러한 접근은 위에서 소개한 일련의 연구에서 밝혀진 것과 같이 [8, 3, 10], 전체 데이터를 유사한 특성을 지닌 여러 개의 부분으로 나누어 학습하여 소프트웨어 공학 데이터의 이질성을 극복할 수 있는 장점이 있다. 또한, Wang 등의 연구를 통해 밝혀진 사실 [11], 즉 개발자들은 대체적으로 한 두개의 컴포넌트와 관련된 버그들을 집중적으로 담당한다는 사실에 주목하면 버그 리포트라는 소프트웨어 공학 데이터는 컴포넌트 항목을 사용하여 그룹화하는 것이 매우 간단하면서 효과적인 방법이 될 수 있다. 본 논문에서 수행한 사례연구결과 컴포넌트 기반의 학습모델을 사용한 추천 기술은 전체 데이터로 한 번에 학습한 추천모델에 비해 최대 20% 이상의 성능 향상을 보인다는 점을 확인하였다. 본 논문의 공헌 점은 아래와 같다.

- 컴포넌트 단위 학습을 통해 효과적인 버그 담당자 할당 기법을 제시한다.
- 오픈소스 프로젝트를 통한 사례연구를 수행하여 제안하는 기법의 효과를 보고한다.

본 논문의 2장에서 제안하는 버그 담당자 할당 기법에 관해 기술하며, 3장에서는 오픈소스를 사용한 실험과 그 결과에 대해 보고한다. 마지막으로 결론 및 향후 연구를 제시하며 논문을 마친다.

2. 컴포넌트 단위 학습 기반 버그 담당자 추천

그림 1은 본 연구에서 제안하는 컴포넌트 단위

학습을 이용한 버그 담당자 추천의 개요와 적용 시나리오를 보여주고 있다. 먼저 버그 저장소에 존재하는 이미 해결된 버그 리포트를 수집하여 컴포넌트 항목의 값으로 분류한다. 컴포넌트 항목 값이 같은 버그 리포트들을 하나의 훈련 데이터 집합으로 사용해 기계학습 알고리즘을 적용한다. 그 결과 컴포넌트 단위로 생성된 여러 개의 개발자 추천모델이 생성된다. 새로운 버그 리포트가 생성되면 해당 버그 리포트의 컴포넌트 항목 값에 따라 대응하는 개발자 추천 모델을 사용해 적합한 버그 담당자를 추천해준다.

3. 실험

수행하는 실험은 컴포넌트 단위 학습을 통한 기계학습모델이 기존에 주로 사용되고 있는 방법, 즉 전체 데이터로 한 번에 학습된 모델보다 더 뛰어난 담당자 추천 성능을 보이는지 확인하는 것이다. 3.1절에서 수행된 실험의 설계에 관해 기술하며 3.2절에서는 실험 결과를 도시하고 그에 대한 분석을 제시한다.

표 1. 실험에 사용된 버그 리포트 (2002/01/01 ~ 2016/12/31)

프로젝트	컴포넌트 개수	참여 개발자 수	버그 리포트 개수
Eclipse JDT	6	45	7,421
Eclipse Platform	21	132	10,490

3.1 실험 설계

(1) 실험 데이터

표 1은 실험에 사용된 버그 리포트에 관한 정보를 나타낸다. 버그 리포트를 얻기 용이한 오픈소스 프로젝트인 Eclipse JDT와 Eclipse Platform의 버그 리포트를 사용해 실험을 수행하였으며 해당 버그 리포트는 두 프로젝트의 버그 추적 시스템인 Bugzilla를 통해 얻을 수 있다 [13]. 수집된 버그 리포트는 2002년부터 2016년 동안 발생한 것이며, 리포트의 상태가 resolved, verified 또는 closed인 버그 리포트만을 사용하였다. 또한, 버그 담당자 할당의 실효성을 높이기 위해 최근 2년 동안 단 1건의 버그 리포트도 처리하지 않은 개발자들이 포함된 리포트는 제외하였다.

(2) 기계학습 알고리즘

버그 담당자 할당에 관한 선구적인 연구에서 [1, 4] 기계학습 알고리즘으로 NB(Naïve Bayes)와 SVM(Support Vector Machine)이 사용된 이후로 여러 후속 연구에서 두 알고리즘이 널리 사용되고 있다. 본 실험에서는 위 두 가지 기계학습 알고리즘과 최근 연구에서 좋은 성능을 보인 바 있는 RF(Random Forest)를 사용해 실험을 진행하였다 [7]. 효과적인 실험을 위해 오픈소스 데이터 마이닝 도구인 WEKA에서 제공하는 구현을 사용하였다 [5].

(3) 특성 벡터 구성 및 전처리 작업

기계학습을 적용하기 위해 각 버그 리포트는 특성 벡터(Feature vector)로 표현되어야 한다. 본 실험에서는 리포트의 summary와 description항목에 존재하는 자연어 어휘들을 사용해 버그 리포트를 어휘 등장 빈도를 나타내는 특성 벡터로 표현하였다. 그리고 각 리포트를 해결한 개발자의 이름을 부류 특성(class feature)으로 설정하였다. 이는 정보 검색이나 텍스트 마이닝 분야에서 사용하는 방식과 유사하다. 특성 벡터를 구성하기 전에 버그 리포트에 등장하는 숫자나 특수문자 그리고 등장 어휘 중 'a' 혹은 'the' 와 같은 의미 없는 단어들을 제거하였다. 또한 모든 단어에 어간 추출(Stemming)을 적용하였다. 마지막으로, 구성된 특성 벡터에 적절한 가중치를 부여하기 위해 TF-IDF(Term Frequency-Inverse Document Frequency)를 적용하였다.

일반적으로 다수의 자연어 문서에는 수많은 단어가 존재한다. 따라서 위와 같은 과정으로 만들어진 특성 벡터는 매우 큰 차원을 가진다. 예컨대, Eclipse JDT 프로젝트의 경우 7,421개의 버그 리포트의 summary와 description 항목에 존재하는 단어의 개수는 위 과정을 모두 거친 후에도 2만 2천 개 이상이었다. 따라서 기계학습 알고리즘의 계산 시간과 성능을 위해 반드시 적절한 어휘 선택이 필요하다. 본 실험에서는 모든 단어 중 TF-IDF 값이 가장 큰 100개의 단어를 선택하는 간단한 방법을 사용하였다.

해당 실험은 컴포넌트 단위 학습의 효과를 입증하는 것에 집중하고 있기 때문에, 특별히 고도화된 전처리 기술을 사용하지는 않았다. 향후 연구의 일환으로 고도화된 텍스트 마이닝 기술들을 적용하면 더 좋은 결과를 얻을 수 있을 것으로 예상된다.

(4) 평가 방법

마지막으로 실험에서 사용한 성능 평가 방법과 척도에 관해 소개한다. 공정한 평가를 위해 기계학습 분야에서 널리 사용되는 k-겹 교차 검증(k-fold cross validation) 방법을 적용하였다. 전체 데이터 집합을 k개의 부분 집합으로 등분한 뒤 k-1 개의 부분을 통해 모델을 학습시키고, 나머지 1개의 데이터 부분으로 성능을 측정한다. k개의 모든 부분 집합으로 테스트한 결과를 얻기 위해 위 과정을 k번 수행한다. 측정된 k개의 성능 결과의 평균을 최종적인 성능으로 취한다. 본 실험에서는 k를 10으로 설정하였다.

버그 담당자 할당 모델을 평가하는 데 주로 사용되는 평가 척도는 Top-X 정확도(Accuracy)이다. 하나의 테스트 버그 리포트에 대해 담당자 예측 모델은 X 명의 담당자 후보 집합을 제안하며, 그중 실제 정답이 있는

경우에는 해당 테스트 리포트를 통과한 것으로 간주한다. 전체 테스트 버그 리포트 중 통과한 리포트의 비율을 계산하면 Top-X 정확도를 얻을 수 있다. X 의 값으로는 1, 3 그리고 5가 주로 사용된다. X 의 숫자가 작을수록 평가 기준이 엄격해 진다고 볼 수 있다. 본 실험에서는 Top-1 정확도를 통해 제안하는 모델의 성능 평가를 수행하였다.

표 2. 두 학습모델의 Top-1 정확도 비교

	Naïve Bayes	SVM	Random Forest
Eclipse JDT			
Global model	24.44	49.81	55.18
Proposed model	47.59	63.75	67.79
Eclipse Platform			
Global model	19.51	41.06	44.63
Proposed model	42.96	56.94	59.86

표 3. 컴포넌트 단위 학습모델의 메타 정보 및 Top-1 정확도.
† 표시는 버그 리포트가 너무 적어 학습모델을 구축 할 수 없는 컴포넌트를 나타낸다.

프로젝트	컴포넌트	개발자 수	버그 리포트 개수	NB	SVM	RF
Eclipse JDT	APT	8	67	43.28	40.30	47.76
	Core	28	4,359	37.37	58.59	63.80
	Debug	16	574	51.92	66.03	70.04
	Doc	9	76	40.79	26.32	40.79
	Text	12	763	72.48	77.20	77.72
	UI	20	1,582	62.71	73.45	75.35
Eclipse Platform	Ant	16	70	32.86	51.43	65.71
	Compare	9	200	73.50	79.50	82.00
	CVS	3	133	75.12	83.46	85.71
	Debug	19	153	45.10	56.21	60.13
	Doc	18	182	32.42	43.96	41.76
	IDE	33	253	21.34	29.25	32.81
	PMC	4	33	63.64	69.70	51.52
	Releng	22	1,453	71.13	80.14	82.06
	Resources	13	919	73.45	81.28	81.39
	Runtime	26	628	33.121	47.45	53.18
	Search	8	132	90.15	91.67	92.42
	SWT	37	903	21.71	40.10	44.52
	Team	17	255	68.24	76.08	80.00
	Text	30	644	60.25	72.05	71.89
	UI	105	4,358	26.69	45.50	48.99
	Update	2	21	85.71	85.71	90.48
	User Assistance	17	109	33.03	29.36	41.28
	Website	7	37	59.46	51.35	56.76
	†Incubator	1	1	-	-	-
	†Scripting	2	2	-	-	-
	†WebDAV	1	2	-	-	-

3.2 실험결과 분석

표 2는 본 연구에서 제안하는 컴포넌트 단위 학습을 통한 버그 담당자 추천모델(Proposed model)과 전체 데이터를 통해 학습된 추천모델(Global model) 사이의 성능을 비교하고 있다. 표의 숫자들은 Top1 정확도를 퍼센트로 나타낸 것이다. Eclipse JDT 프로젝트의 경우, 제안하는 모델이 global model에 비해 세 가지 기계학습 알고리즘 모두에서 약 12%에서 23% 정도의 성능 향상을 보이고 있다. Eclipse Platform 프로젝트에서도 대략 15%에서 22% 정도의 성능 향상을 볼 수 있다. 실험 결과에서 살펴볼 수 있는 또 하나의 흥미로운 결과는 NB보다 SVM 그리고 RF 알고리즘이 더 좋은 성능을 보인다는 점이다. 이는 최근에 보고된 성능 비교 연구의 결과와 일치한다 [7].

표 3은 실험에서 사용된 각 컴포넌트 단위로 학습된 추천 모델의 세부 정보를 보여준다. 해당 세부 정보는 프로젝트의 컴포넌트 이름과 그 컴포넌트에 속한 버그 리포트의 개수 그리고 해당 컴포넌트의 버그를 한 번이라도 해결한 이력이 있는 개발자의 수를 포함한다. 또한, 표 3은 각 컴포넌트 단위 추천모델의 Top-1 정확도를 포함하고 있으며 이들의 성능을 종합하면 표 2에 제시된 전체 Top-1 정확도를 얻을 수 있다.

Eclipse JDT 프로젝트는 총 6개의 컴포넌트로 구성된다. 따라서 각 컴포넌트에 속한 버그 리포트를 사용해서 6개의 컴포넌트 단위로 학습된 추천모델이 생성된다. Eclipse Platform은 총 21개의 컴포넌트를 가지고 있으며, 이에 따라 21개의 컴포넌트 단위 추천모델이 만들어 져야한다. 그러나 Incubator, Scripting 그리고 WebDAV 컴포넌트에는 각각 1개, 2개, 1개의 버그 리포트만이 존재한다(2016/12/31 기준). 따라서 위 3개의 컴포넌트로는 학습모델을 구축할 수 없었다. 이런 컴포넌트들은 비교적 최근에 프로젝트에 추가된 것으로 보인다.

표 3의 내용을 토대로 컴포넌트 단위 추천모델이 global model에 비해 높은 Top-1 정확도를 보이는 이유를 아래와 같이 추론할 수 있다. 전체 버그 리포트를 통해 구축된 global model은 수많은 개발자 가운데 가장 적합한 버그 담당자를 찾아야 한다. Eclipse Platform 프로젝트의 경우, global model은 132명 가운데 가장 적절한 개발자 1명을 찾아야 한다. 이는 훈련을 위한 버그 리포트가 충분히 많고, 아주 좋은 학습 알고리즘을 사용한다 하더라도 매우 어려운 문제이다. 실험 결과 역시, 가장 좋은 성능을 보이는 알고리즘(RF)의 경우에도 약 44% 정도의 상대적으로 낮은 정확도를 보이고 있다. 이에 비해 제안하는 추천모델은 이러한 문제를 컴포넌트 단위 학습을 통해 효과적으로 대처한다. 예컨대, Eclipse Platform 프로젝트의 Releng 컴포넌트 학습모델은 22명중 1명의 적절한 개발자를 찾는 문제를 풀고 있다. 이는 global model의 그것과 비교하면 굉장히 쉬운 문제이다. 실험 결과 해당 컴포넌트 모델은 80%가 넘는 정확도를 보이는 것을 확인 할 수 있다. 이는 전체 버그 리포트를 컴포넌트라는 버그 해결 활동과 매우 관련 깊은 정보를 통해 여러 그룹으로 분할하여 학습모델을 구축하는 것이 부류 특성의 개수를 매우 효과적으로 줄여주는 역할을 수행한다고 볼 수 있다. 따라서 제안하는 추천모델이 훈련 데이터의 감소에도 불구하고 global model에 비해 더 좋은 버그 담당자 추천 성능을 보이고 있다고 파악된다.

4. 결론 및 향후 연구

본 논문에서는 컴포넌트별 단위 학습을 통한 버그 담당자 추천 기법을 제안하였다. 컴포넌트는 버그 해결 활동과 매우 관련 깊은 정보이며 이를 통해 전체 데이터를 효과적으로 분할 할 수 있다. 본 논문에서 수행한 실험 결과는 제안하는 방법이 전체 데이터를 한 번에 학습하는 모델에 비해 약 12%에서 23%의 Top-1 정확도 향상을 보인다는 사실을 확인할 수 있었다.

향후 연구에서는 딥 러닝을 비롯한 다양한 학습 알고리즘을 적용해볼 계획이며, Eclipse 이외의 다른 오픈소스 프로젝트나 산업체 프로젝트의 버그 리포트를 활용할 예정이다. 또한, 본 논문에서는 버그 리포트를 특성벡터로 표현하는 과정에서 기초적인 텍스트 마이닝 기술만을 적용하였다. 차후 실험에서는 고도화된 텍스트 분석 기술을 적용해보는 것도 좋은 시도가 될 수 있을 것이다.

5. 감사의 글

본 연구는 한국연구재단 기초연구사업(과제번호 NRF-2014R1A2A2A01005519)의 지원을 받았습니다.

6. 참고 문헌

[1] Anvik J, Hiew L and Murphy GC. Who should fix this bug? Proceedings of the 28th international conference on software engineering, pp 361-370, 2006.

[2] Anvik J and Murphy GC. Reducing the effort of bug report triage: recommenders for development-oriented decisions. Trans Softw Eng Methodol, 20(3), pp 10:1-10:35, 2011.

[3] Bettenburg N, Nagappan M and Hassan AE. Think locally, act globally: Improving defect and effort prediction models. Proceedings of the 9th Working Conference on Mining Software Repositories, pp 60-69, 2012.

[4] Cubranic D and Murphy GC. Automatic bug triage using text categorization. Proceedings of the 16th international conference on software engineering & knowledge engineering, pp 92-97, 2004.

[5] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P and Witten IH. The WEKA data mining software: an update. SIGKDD Explor Newsl 11(1), pp 10–18, 2009.

[6] Jeong G, Kim S and Zimmermann T. Improving bug triage with bug tossing graphs. Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp 111–120, 2009.

[7] Jonsson L, Borg M, Broman D, Sandahl K, EldhPer S and Runeson P. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. Empirical Software Engineering, 21(4), pp 1533–1578, 2016.

[8] Menzies T, Butcher A, Marcus A, Zimmermann T and Cok D. Local vs. global models for effort estimation and defect prediction. Proceedings of the 26th International Conference on Automated Software Engineering, pp 343–351, 2011.

[9] Park J, Lee M, Kim J, Hwang S and Kim S. A cost-aware triage algorithm for bug reporting systems. Proceedings of the 25th AAAI conference on artificial intelligence, pp 139–144, 2011.

[10] Scanniello G, Gravino C, Adrian M and Menzies T. Class level fault prediction using software clustering. Proceedings of the 28th international conference on automated software engineering, pp 640–645, 2013.

[11] Wang S, Zhang W and Wang Q. FixerCache: unsupervised caching active developers for diverse bug triage. Proceedings of the 25th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2014.

[12] Zhang T and Lee B. A hybrid bug triage algorithm for developer recommendation. Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp 1088–1094, 2013.

[13] URL: https://bugs.eclipse.org/bugs/

I-BL 기술의 성능 향상을 위한 버그리포트 품질 예측 및 자동 재구성 기술

김미수 ○ 안준 이은석
성균관대학교 전자전기컴퓨터공학과
{misoo12, ahnjune, leees}@skku.edu

Quality Prediction and Automated Reformulation Technique of Bug Reports for Enhancing Performance of I-BL (Information Retrieval-based Bug Localization)

Misoo Kim○ June Ahn Eunseok Lee
Sungkyunkwan University

요 약

버그리포트는 소프트웨어의 운용 및 유지보수 단계에서 발생한 결함 정보를 담고 있는 문서로서 개발자가 해당 결함을 수정하기 위해 필수적인 정보이다. 이 때 개발자가 버그리포트를 해결하기 위해 결함을 추적하는 시간을 단축시키기 위한 정보검색기반 결함위치추적 기술들이 제안되었다. 그러나 정보검색에 유용하지 못한 내용들로 작성된 낮은 품질의 버그리포트가 등록 될 경우 결함위치추적의 성능이 크게 저하된다. 본 논문에서는 버그리포트의 쿼리로써의 품질을 예측하여, 예측된 품질에 따라 버그리포트의 내용을 자동으로 재구성하여 품질을 향상시키는 방법을 제안한다. 본 논문에서는 오픈 소스 프로젝트에 제안하는 방법을 적용하여 기존 정보검색기반 결함위치추적 기술들의 정확도를 평균 7.7% 향상 시켰다.

1. 서론

소프트웨어의 운용 및 유지보수 단계에서 테스터나 사용자에 의해 결함이 발생하게 되면 해당 결함을 발견한 사람에 의해 버그리포트가 작성된다[1]. I-BL 기술들은 정보검색 기술에 버그리포트를 쿼리로 활용하고 소프트웨어의 소스파일들을 문서 집합과 대응시켜 버그리포트와 관련된 결함 소스파일들을 검색하는 기술[2,3]이다. 해당 기술은 등록된 버그리포트가 정보검색에 유용하지 못한 내용들로 작성될 경우, 버그리포트가 낮은 품질의 쿼리로써 사용되기 때문에 정보검색 기술 자체의 성능뿐만 아니라 결함위치추적 정확도 또한 떨어질 수 있는 문제가 존재한다. 본 논문에서는 버그리포트의 쿼리로써의 품질을 예측하고 예측된 품질에 따라 버그리포트의 내용을 자동으로 재구성하여 위 문제를 해결하는 방법을 제안한다.

2. 제안 방법

2.1 버그리포트 품질 예측 방법

본 논문에서는 좋은 품질의 버그리포트는 I-BL 기술을 통해 결함 소스파일을 상위 10위내로 검색한 경우로, 그렇지 않은 경우를 나쁜 품질의 버그리포트로 정의한다. 본 논문에서는 버그리포트의 품질 예측을 위해, 쿼리 평가에서 일반적으로 사용되는 *AvgIDF*[4]와 초기에 검색된 상위 30개(Top-30)의 소스파일들의 패키지와 의존 정보를 사용하는 것을 제안한다.

(1)은 쿼리의 *AvgIDF*를 구하는 식이다.

$$AvgIDF(Q) = \frac{1}{|Q|}\sum_{q \in Q}(\log\frac{|D|}{df_t}) \quad (1)$$

*Q*는 버그리포트의 단어들의 집합, *q*는 *Q*의 개별 단어, *D*는 전체 문서집합, *dfq*는 단어 *q*를 갖는 문서들의 수이다.

소스파일들의 패키지는 서로 관련이 있거나 유사한 목적을 갖는 클래스의 집합으로써 버그리포트를 통해 획득한 검색 결과들이 같은 패키지에 존재한다면 유사한 목적에서 구현된 소스파일임을 의미한다. 제안하는 패키지 정보 기반 품질 예측을 위해 Top-30개의 파일에 대한 (2)와 (3)를 제안한다.

$$MaxPckgScore(Q) = max(n(\{p|p \in P\})) \quad (2)$$

$$NumPckgScore(Q) = n(\{p'|p' \in P'\}) \quad (3)$$

*Q*는 쿼리인 버그리포트, *P*은 Top-30개의 소스파일들이 위치된 패키지의 중복이 제거되지 않은 집합, *P'*은 중복이 제거된 패키지 집합, *n*은 집합 내 원소들의 수를 의미한다.

소스파일들간의 의존 정보는 소스파일들이 참조하거나 참조되는 관계를 설명한다. 해당 정보는 프로그램의 절차를 확인할 수 있는 정보로써, 초기에 검색되는 소스파일들 사이에 의존 관계가 있다면 해당 소스파일들은 프로그램 수행 시 유사한 목적을 달성하기 위해 함께 수행될 가능성이 높다. 이러한 가정에 근거한 의존 정보 기반 품질 예측을 위해 Top-30개에 대한 (4), (5), (6)과 (7)를 제안한다.

$$CallScore(Q) = n(\{c|c \in C\}) \quad (4)$$

$$CalledScore(Q) = n(\{cd|cd \in CD\}) \quad (5)$$

$$MaxCallScore(Q) = \max(n(Call(r), r \in R)) \quad (6)$$

$$MaxCalledScore(Q) = \max(n(Called(r), r \in R)) \quad (7)$$

*R*은 Top-30개의 소스파일들의 집합, *C*는 *R* 중 소스파일들을 참조하고 있는 소스파일의 집합, *CD*는 *R* 중 참조되는 소스파일의 집합, *Call*은 *r*이 참조하고 있는 소스파일의 수, *Called*는 *r*이 참조되고 있는 소스파일의 수이다.

본 논문에서는 Weka[5]의 인공신경망을 사용하여 버그리포트를 통해 계산되는 (1)~(7)과 버그리포트의 품질을 라벨로 학습하여, 새롭게 들어오는 버그리포트에 대한 품질 요소를 계산하고 기계학습을 통해 품질을 예측한다.

2.2 자동 버그리포트 재구성 기술

본 논문에서는 유의어 온톨로지를 사용한 쿼리 변환 기술과 해결된 버그리포트기반 쿼리 확장 기술을 제안한다.

유의어 온톨로지기반 쿼리 변환기술은 SEWordSim[6]과 WordNet[7]을 사용한다. 버그리포트에 소스파일에 존재하지 않는 단어들이 있다면, 해당 단어들에 대해 SEWordSim을 통해 유사한 단어를 추출하고 소스파일에 있는지 확인한 후 변경한다. SEWordSim을 통해 소스파일에 있는 유의어를 추출할 수 없다면 WordNet을 통해 유의어를 추출하여 소스파일에 있는지를 확인하여 변경한다. 그럼에도 적절한 변경할 단어를 선택할 수 없다면 WordNet을 통해 단어의 상위어가 소스파일에 있는지 확인하여 변경한다.

해결된 버그리포트기반 쿼리 확장 기술은 다음과 같다. 해결된 버그리포트와 수정된 Top-5개의 소스파일들에서 (8)통해 10개의 중요한 단어들을 추출한다. (8)은 단어 중요도를 계산하는 수식이다[4]. 추출된 단어 쌍은 2차원 단어빈도 행렬로 구성되고, 버그리포트가 해결될 때마다 단어 사이의 빈도를 증가시킨다. 새롭게 들어오는 버그리포트에 대한 단어들 중 중요도가 높은 단어들에 대해 생성된 단어빈도 행렬에서 지난 시간 빈도가 가장 높은 10개의 단어들을 선택하여 추가한다.

$$TfIDF(q,r) = tf(q,r) \times (\log \frac{5}{rf_q}) \quad (8)$$

문서 집합 *R*의 개별 문서 *r*에 존재하는 단어 *q*에 대한 단어 중요도를 계산하는 수식으로, *tfqr*는 *r*에 *q*가 나타난 횟수, *rfq*는 단어 *q*를 갖는 *r*문서들의 수이다.

3. 실험

3.1 대상 프로젝트 및 평가 방법

제안하는 품질 예측을 위해 10-fold cross-validation를 사용한다. 제안 기술을 통해 향상된 품질의 버그리포트는 I-BL 기술의 성능 향상 여부를 검증하기 위해 BLIA+[2]와 BugLocator[3]에 적용한다. 평가 방법과 실험 데이터는 BLIA+와 동일하며, AspectJ의 경우 BugLocator와 실험 데이터가 상이하여 결과에서 제외한다.

3.2 실험 결과

표 2는 실험 결과로 가장 좋은 값들은 밑줄로 표기한다. 제안하는 버그리포트 품질 평가 및 자동 재구성 기술은 기존 I-BL의 결함 추적 성능인 Top1, Top5, Top10, MRR, MAP에 대해서 평균 7.8%, 2.0%, 4.2%, 4.3%, 7.7%만큼 향상시켰다.

표 1 결함위치추적 평가 결과

프로젝트	방법	Top1 (%)	Top5 (%)	Top10 (%)	MRR	MAP
Aspectj	[2]	0.415	**0.711**	**0.806**	0.549	0.389
	제안+[2]	**0.426** (+2.6%)	0.708 (-1.1%)	**0.806** (+0.0%)	**0.552** (+0.6%)	**0.391** (+0.5%)
SWT	[3]	0.398	0.674	0.816	0.530	**0.450**
	제안+[3]	**0.449** (+25.6%)	**0.694** (+3.0%)	**0.867** (+6.3%)	**0.569** (+7.4%)	**0.450** (+0.0%)
	[2]	0.673	0.867	0.898	0.750	0.652
	제안+[2]	**0.684** (+1.6%)	**0.878** (+1.3%)	**0.908** (+1.1%)	**0.760** (+1.3%)	**0.653** (+0.2%)
Zxing	[3]	0.400	**0.600**	0.700	0.500	0.440
	제안+[3]	**0.500** (+25.0%)	**0.600** (+0.0%)	**0.750** (+7.1%)	**0.548** (+9.6%)	**0.569** (+29.3%)
	[2]	0.550	0.750	0.800	0.638	0.620
	제안+[2]	**0.600** (+9.1%)	**0.800** (+6.7%)	**0.850** (+6.3%)	**0.656** (+2.8%)	**0.674** (+8.7%)
평균 상승률		**+7.8%**	**+2.0%**	**+4.2%**	**+4.3%**	**+7.7%**

4. 결론

본 논문은 기존 I-BL 기술의 정확도를 하락시킬 수 있는 낮은 품질을 갖는 버그리포트들의 품질을 높이기 위한 품질 예측 기술과 자동 재구성 기술을 제안한다. 제안 기술을 적용하여 품질이 향상된 버그리포트들을 사용하여 I-BL 기술의 정확도가 향상되는 것을 확인하였다.

Acknowledge

이 논문은 2016년 정부(교육부)의 재원으로 한국연구재단-이공학 개인기초연구 지원 사업 (No. 2016R1D1A1B03934610), 미래창조과학부 및 정보통신 기술진흥센터의 SW중심대학 지원사업(No. R7115161006000 1002)의 연구결과로 수행됨

참고 문헌

[1] J.Zhang, X. Wang, D. Hao, B. Xie, and L. Zhang, "A survey on bug-report analysis," *Science China Information Sciences*, vol. 58, no. 2, pp. 1-24, 2015.

[2] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reposrt," *Information and Software Technology,* vol. 82, no. 1, pp. 177-192, 2017.

[3] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, pp. 14-24, 2012.

[4] D. Carmel, and E. Y. Tov, "Estimating the Query Difficulty for Information Retrieval," *Synthesis Lectures on Information Concepts, Retrieval, and Services,* vol. 2, no. 1, pp. 1-89, 2010.

[5] Weka: http://www.cs.waikato.ac.nz/ml/weka/

[6] Y. Tian, D. Lo, and J. Lawall, "SEWordSim: Software-specific word similarity database," in *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014),* pp. 568-571, 2014.

[7] WordNet: https://wordnet.princeton.edu/

딥 러닝을 이용한 버그 담당자 자동 배정 연구

이선로○, 김혜민, 이찬근
중앙대학교 컴퓨터공학부
서울시 동작구 흑석로 84
ssunno, kimkim94, cglee@cau.ac.kr

A Study on Automatic Bug Triage using Deep Learning

Sun-Ro Lee○, Hye-Min Kim, Chan-Gun Lee
Computer Science and Engineering, Chung-Ang University

요 약

많은 버그 담당자 자동 배정 연구들은 SVM, Naïve Bayes 등 고성능의 기계학습 모델을 이용하여 버그 리포트를 학습하는 방법을 제안하고 있다. 본 논문에서는 기계학습 활용 분야에서 좋은 성능을 보이는 딥 러닝을 버그 담당자 자동 배정에 적용한다. 실험 결과 딥 러닝 기반 기법이 활성 개발자를 대상으로 한 실험에서 기존의 기계학습 대비 24~69% 성능이 향상된 48%의 정확도를 보였으며 정밀도와 F-Measure 역시 각각 46%, 45%로 기존 알고리즘 대비 향상된 분류 성능을 보인다.

핵심어: 버그 담당자 자동 배정, 딥 러닝, 기계학습

1. 서 론

버그는 소프트웨어의 전반에서 발생하며 그 종류와 형태가 다양한데 규모가 크고 기능이 복잡한 소프트웨어는 하루에도 수많은 버그가 발생할 수 있다. 대형 프로젝트에서 매일 발생하는 수백 건에 달하는 버그 리포트를 분석하고 담당자를 배정하는 일은 인력적, 시간적 비용이 필요하며 프로젝트 전반에 걸친 지식이 요구된다. 그러나 프로젝트에서 해당 분야를 담당할 전문성을 가진 버그 선별자(Bug Triager)는 그 수가 적어 버그 리포트가 할당되지 못하고 대기하는 병목 현상이 발생하게 된다[2].

딥 러닝은 이미지 처리[4], 음성 인식, 자연어 처리[1]등에서 분야에서 기존의 기계학습 기법들보다 높은 정확도를 보이는 것으로 알려져 있다. 이에 본 연구에서는 최근 여러 분야에서 좋은 성능을 보이는 딥 러닝 알고리즘인 CNN(Convolutional Neural Network)을 이용하여 버그 담당자를 예측하는 방법을 제안한다. 본 논문의 기여점은 다음과 같다.

- 딥 러닝 기반 버그 담당자 자동 배정 시스템 제안
- 기존의 기계학습 기반 연구 대비 성능 향상

2. 관련 연구

2.1. 딥 러닝

딥 러닝은 최근 알고리즘 개선과 하드웨어의 발전으로 인한 처리량 개선, 높은 성능 등으로 주목받고 있다. 딥 러닝은 기계 학습 분야의 하나이며 DNN(Deep Neural Network), CNN(Convolutional Neural Network) 등 알려진 딥 러닝 알고리즘들을 활용한 최근 연구, 특히 자동 음성 인식과 컴퓨터 비전 분야에서 이전에 쓰이던 기계학습 알고리즘에 비해 좋은 성능을 보이고 있다.

2.2. 전처리와 벡터화

자연어 처리[3]는 자연어로 구성된 글을 컴퓨터가 이해할 수 있는 형태로 변환하거나 가공하는 것을 말한다. 품사 부착(POS Tagging), 어간 추출(Stemming), 불용어 제거(Stop Word Removal) 등이 이에 해당하며 이같은 기법들은 문서를 컴퓨터가 효율적으로 인식할 수 있는 형태로 변환하여 기계학습 등을 이용한 문서 처리에서 성능을 향상시킬 수 있다.

단어의 벡터화 기법은 문서에서 단어의 의미를 추출하고 숫자 형태(벡터)로 변환하여 컴퓨터가 처리 가능한 형태로 만든다. 이 기법은 단순한 색인 기반의 변환 방법에 비해 의미를 벡터 공간에 표현함으로써 단어의 관계를 고차원에서 표현할 수 있다.

3. 딥 러닝 기반의 버그 담당자 예측

본 연구에서는 딥 러닝을 이용하여 버그 담당자를 예측하는 방법을 제안한다. 제안하는 기법은 이전에 해결된 버그 리포트를 수집하고 딥 러닝으로 학습하며 학습 모델을 이용해 새 리포트에 적합한 담당자를 배정한다. 모델의 성능을 측정하고 기존의 기계학습과 비교하여 딥 러닝의 활용성과 적합성 등을 확인하기로 한다.

3.1. 데이터 수집 및 전처리

실험을 위한 데이터로 오픈소스 프로젝트인 Eclipse JDT, Eclipse Platform에서 2013-10-20부터 2016-10-20까지의 기간 동안 해결된 버그 리포트를 수집했다. 각각의 프로젝트에서 수집된 리포트는 1,465개, 3,512개 개발자 수는 70명, 164명이다. 수집된 리포트는 개발자 단위로 분류하고 특수문자 등을 제거한 뒤 리포트에 포함된 단어를 각각 벡터화 한다. 이후 CNN 모델이 요구하는 크기로 확장하며 분류 대상인 개발자 수에 따라 정답 리스트를 생성하고 버그리포트-정답 리스트 쌍을 구성한다.

3.2. 딥 러닝 모델 구축과 개발자 예측

본 연구에서는 [1]의 CNN기반 문서 분류 기법을 기반으로 다중 클래스 분류가 가능한 모델을 구축했다. 기존의 기계학습 알고리즘과 달리 CNN은 특징 선택(Feature Selection)기법이

표 1. 딥 러닝 모델과 기계학습 기법의 성능 측정 결과

모델명	프로젝트	전체 개발자 대상				활성 개발자 대상			
		Accuracy	Precision	Recall	F-Measure	Accuracy	Precision	Recall	F-Measure
CNN	**JDT**	**44.4%**	**40.8%**	**44.4%**	**40.5%**	**48.6%**	**47.6%**	**48.6%**	**46.3%**
CNN	**Platform**	**45.1%**	**38.5%**	**45.1%**	**40.2%**	**48.4%**	**43.8%**	**48.4%**	**43.9%**
NB	JDT	31.3%	34.3%	31.3%	31.2%	35.8%	40.7%	35.8%	36.6%
NB	Platform	32.4%	33.2%	32.4%	30.9%	35.4%	37.3%	35.4%	34.2%
SVM	JDT	35.7%	33.7%	35.7%	34.2%	39.3%	39.7%	39.3%	39.3%
SVM	Platform	35.9%	32.3%	35.9%	33.4%	38.7%	36.4%	38.7%	36.9%
C4.5	JDT	28.1%	26.3%	28.1%	26.9%	35.0%	34.1%	35.0%	34.3%
C4.5	Platform	25.1%	22.3%	25.1%	23.3%	28.7%	26.3%	28.7%	27.1%

분류 성능에 큰 영향을 미치지 않는다고 알려져 있다. 따라서 다양한 데이터가 혼재되어 있는 버그 리포트의 분류에서 기존의 기계학습 알고리즘 대비 CNN이 좋은 성과를 보일 수 있다.

학습에 사용된 CNN 모델은 256개의 필터를 사용하는 Convolution과 Max-pooling 레이어가 2단계로 구성되며 2종류의 필터를 사용, 총 512개의 Convolution 필터를 사용한다. 이후 fully-connected 레이어와 소프트맥스 회귀를 통해 정답 리스트 크기의 결과를 출력하게 되며 리스트에서 가장 큰 값과 매칭되는 개발자가 모델에 의해 추천된 결과가 된다. 그림 2는 제안하는 딥 러닝 모델의 구성과 버그 리포트를 학습하여 개발자를 예측하는 과정을 나타낸 것이다.

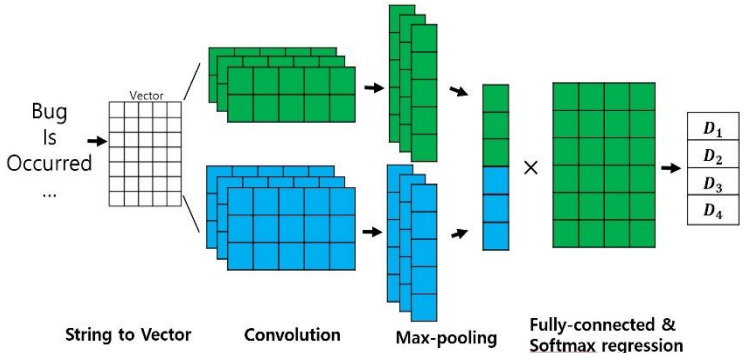


그림 2. 딥 러닝 모델 구조

4. 실험

4.1. 평가 방법

본 실험에서는 수집된 버그 리포트의 일부를 평가 데이터로 사용한다. 기계학습에서는 선정된 평가 데이터에 따라 측정 결과가 달라질 수 있으므로 본 실험에서는 10차 다중 교차 검증(10-fold cross validation) 기법을 적용하여 평가 데이터의 특성에 따른 측정 오차를 줄인다.

평가에 사용되는 항목은 정확도(Accuracy)와 정밀도(Precision), 재현율(Recall), F-Measure를 사용한다. 평가 항목 중 정밀도와 재현율, F-Measure는 분류 대상 각각에 대해 계산되므로 딥 러닝 모델의 분류 성능을 측정하기 위해서 가중치를 적용한 측정 방법으로 모델의 성능을 평가한다. 또한 대상 프로젝트에서 주도적으로 버그를 해결하는 활성 개발자가 상대적으로 많은 리포트를 해결하므로 대상 기간동안 10개 미만의 버그를 해결한 비활성 개발자를 제거한 실험을 진행한다. 본 연구에서는 제안하는 기법의 성능을 평가하기 위해 Naïve Bayes(NB), Support Vector Machine(SVM), C4.5 알고리즘을 대상으로 비교 실험을 진행했다.

4.2. 실험 결과

실험 결과 딥 러닝 예측 모델이 각각 44.7%, 48.5%의 정확도를 달성하여 전체 개발자 대상 실험과 활성 개발자 실험 대상 모두 가장 높은 성능을 보인다. 정밀도와 재현율 역시 타 모델 대비 10%p 이상 향상된 성능을 보였으며 F-Measure에서도 최소 6%p 이상의 성능 차이를 보이며 높은 성능을 보인다.

5. 결론 및 향후 연구

본 연구에서는 CV, ASR 등의 분야에서 좋은 성능을 보이는 딥 러닝 모델인 CNN(Convolutional Neural Network) 이용한 버그 담당자 자동 배정 기법을 제안하였고 기존의 기계학습 기법에 비해 24~69%향상된 정확도를 달성했다. 본 연구에서는 버그 리포트의 설명 정보만을 사용하고 있으나 관련 연구에서 제안되는 토싱 그래프 활용, 개발자 간 관계 분석, 토픽 기반 리포트 분석과 같은 기법들과 응용하여 성능 향상을 기대할 수 있다.

후속 연구로는 추가 정보(제목, 컴포넌트, 진화적 정보 등)를 활용하여 딥 러닝 기반 모델을 확장할 예정이다. 또한 다른 기법과의 비교를 통해 제안하는 기법의 성능을 평가하여 고성능의 버그 담당자 자동 배정 시스템을 제안하는 것을 목표로 한다.

감사의 글

본 연구는 한국연구재단 기초연구사업(과제번호 NRF-2014R1A2A2A01005519)의 지원을 받았습니다.

6. 참고문헌

[1] Kim, Yoon. "Convolutional neural networks for sentence classification", arXiv preprint arXiv:1408.5882, 2014.

[2] Anvik, J., Hiew, L., Murphy, G. C. "Who should fix this bug?" *In Proceedings of the 28th international conference on Software engineering*, ACM, pp. 361-370, 2006.

[3] Chowdhury, G. G., "Natural language processing", *Annual review of information science and technology*, 37(1), 51-89., 2003.

[4] Simonyan, K., Zisserman, A. "Very deep convolutional networks for large-scale image recognition.", arXiv preprint arXiv:1409.1556, 2014.

파일들의 패키지 관계를 반영한 변경 결합도 추출 및 이를 이용한 리팩토링 대상 선별

조영준, 허민재, 이찬근
중앙대학교 컴퓨터공학부
skywave@cau.ac.kr, elfel.cau@gmail.com, cglee@cau.ac.kr

Extracting change coupling by reflecting package relationships and recommending refactoring targets

Yeong-Jun Cho, Minjae Heo, Chan-Gun Lee
Chung-Ang University Computer Science & Engineering

요 약

본 연구에서는 소스 코드 변경 이력에서 연관된 리비전들 내에서 같이 변경된 파일들의 패키지 관계인 패키지 기반 동시 변경 정보를 추출하고 가공하는 방법에 대해서 논의한다. 그리고 이 패키지 기반 동시 변경 정보를 이용하여 리팩토링 대상 선별에 사용될 수 있는 변경 결합도를 계산하는 방법을 제안하며 DSM과의 결과 비교를 통하여 패키지 관계를 이용한 변경 결합도로 리팩토링 대상을 선별하는 것이 유효함을 보여주었다.

1. 연구 배경

분석 - 설계 - 구현 - 테스팅 - 운영 및 유지보수로 이루어진 소프트웨어 개발의 생명 주기에서 소프트웨어 유지 보수 단계가 가장 큰 비용을 차지한다는 것은 이미 잘 알려진 사실이다[1]. 따라서 전체적인 소프트웨어 운영 비용 감소를 위해 유지 보수 비용을 감소하는 것은 중요한 문제이며 소스 코드를 변경하되 소프트웨어의 외부 작동은 변경시키지 않고 내부 구조만을 변경하여 소프트웨어를 개선하는 리팩토링(refactoring)을 통하여 유지 보수의 비용을 줄일 수 있다는 것이 알려져 있다[2]. 하지만 '리팩토링을 어느 파일, 어느 모듈에 적용해야 하는가'를 발견하는 것도 어려운 것으로 알려져 있으며[3] 이를 위해 다양한 접근 방법이 연구되고 있다.

접근 방법의 한가지로 소스 코드 변경 이력에서 동시에 변경된 파일들의 관계를 이용하여 변경 결합도(change coupling)를 추출하고 이를 이용하여 소프트웨어 오류와의 상관관계를 밝혀낸 연구가 진행되었다[6]. 하지만 각 파일의 패키지 관계를 이용하지 않기 때문에 동일 패키지에 있는 파일들과 같이 자주 변경이 되더라도 결합도(coupling)가 높게 측정되는 한계가 있다.

본 연구에서는 이를 해결하기 위해 버전 관리 시스템에서 동시에 변경된 소스 파일들의 패키지 정보를 이용하여 패키지 기반 동시 변경 정보(package based co-change information)를 추출하고 이를 이용하여 계산된 변경 결합도로 리팩토링 대상을 선별하는 기법을 제시한다.

2. 관련 연구

리팩토링 대상을 체계적으로 선별하기 위하여 다양한 연구들이 진행되고 있다. 예를 들어 소프트웨어 변경 이력을 분석하여 어떠한 파일이 변경되었을 때 같이 변경되어야 하는 곳을 예측하거나[9] 소프트웨어를 분석하여 품질 지표를 계산한 후 이를 이용하여 리팩토링 대상을 찾는 연구들[10]이 이에 해당한다. 품질 지표로는 코드 길이, 제보된 버그 개수, 결합도등 다양한 값들이 사용된다.

결합도는 한 모듈이 정상적으로 작동하기 위하여 다른 모듈에 얼마나 의존하는지 정량적으로 나타낸 값이다. 이를 계산하기 위하여 다양한 방법들이 사용되는데 이 중 하나로 프로젝트 소스 코드에서 함수 시그니처, 모듈 임포트등 구조적 정보를 분석하여 계산되는 구조적 결합도가 있다[4].

최근에는 결합도 값을 구조적 정보가 아닌 소스 코드 변경 이력을 이용하여 추출하기도 하며 이 결과는 변경 결합도라고 부른다. 그리고 변경 이력에서 같이 변경된 파일들의 관계를 이용하여 다양한 방법으로 계산된 변경 결합도가 소프트웨어의 오류와 상관관계가 있다는 것도 밝혀졌다[6].

3. 실험 소개

본 실험을 진행하기 위해 버전 관리 시스템에서 패키지 기반 동시 변경 정보 추출 프로그램과 추출된 동시 변경 정보를 이용하여 변경 결합도를 추출하는 2개의 프로그램을 구현하였다. 프로그램의 규모를 줄여 유지보수를 쉽게 하고 프로그램의 범용성을 높이기 위해 기능을 나누어 별도로 프로그램을 완성하였다.

3.1 패키지 기반 동시 변경 정보 및 변경 결합도 정의

본 연구에서 정의하는 패키지 기반 동시 변경 정보란 '소스 코드 변경 이력에서 연관된 리비전(revision)들 내에서 같이 변경된 파일들의 패키지 관계 정보'이며 '패키지 관계 정보'로는 '동일 패키지에 포함된 파일 개수와 타 패키지에 포함된 파일 개수'가 사용된다. 이때 패키지 명은 파일의 디렉토리 위치와 같다고 가정하며 패키지 명의 일부가 일치하더라도 전체가 일치하지 않으면 다른 패키지로 처리한다.

- /package_1/file_1.java
- /package_1/file_2.java
- /package_1/sub_package/file_3.java
- /package_2/file_4.java

연관된 리비전(그룹) 내에서 동시에 변경 된 파일들이 위와 같은 경우 file_1.java 파일과 같은 패키지에 포함된 파일 개수는 'file_2.java' 1개이며 다른 패키지에 포함된 파일 개수는 'file_3.java', 'file_4.java' 2개이다.

한 파일에 대한 변경 결합도는 해당 파일이 포함된 모든 그룹에 대해 각각 다른 패키지에 포함된 파일의 비율을 계산하고 이들의 평균으로 정의한다. 파일의 비율은 0과 1사이의 실수이며 이의 평균을 계산하기 때문에 변경 결합도는 항상 0과 1사이의 실수로 계산된다. 다량의 다른 패키지 파일과 함께 바뀔수록 이 값이 커지기 때문에 이 값이 클수록 다른 모듈과의 의존 관계가 크다고 판단할 수 있으며 이는 리팩토링 대상이 되어야 함을 의미한다.

3.2 패키지 기반 동시 변경 정보 추출

변경 결합도를 계산하기 위해서는 먼저 파일별로 패키지 기반 동시 변경 정보와 관련된 데이터를 추출해야 하며 이를 위한 Java GUI/CUI 프로그램을 그림 1과 같이 개발하였다. 해당 프로그램에서 패키지 기반 동시 변경 정보를 추출하는 개괄적인 과정은 다음과 같으며 상세한 추출 과정과 조절 가능한 인자들은 각 과정별로 별도의 문단에서 서술한다:

- **수집 단계**: 사용자가 지정한 버전 관리 시스템의 모든 리비전 정보를 수집한다.
- **병합 단계**: 짧은 시간 내에 같은 사용자에게 생성된 리비전들을 그룹으로 병합하고 유효하지 않은 그룹들을 제거한다.
- **추출 단계**: 각 그룹 내에 포함된 파일들의 정보를 이용하여 파일별로 패키지 기반 동시 변경 정보를 추출한다.
- **내보내기 단계**: 생성된 패키지 기반 동시 변경 정보와 관련된 데이터들을 지정한 파일 형식의 파일로 내보낸다.

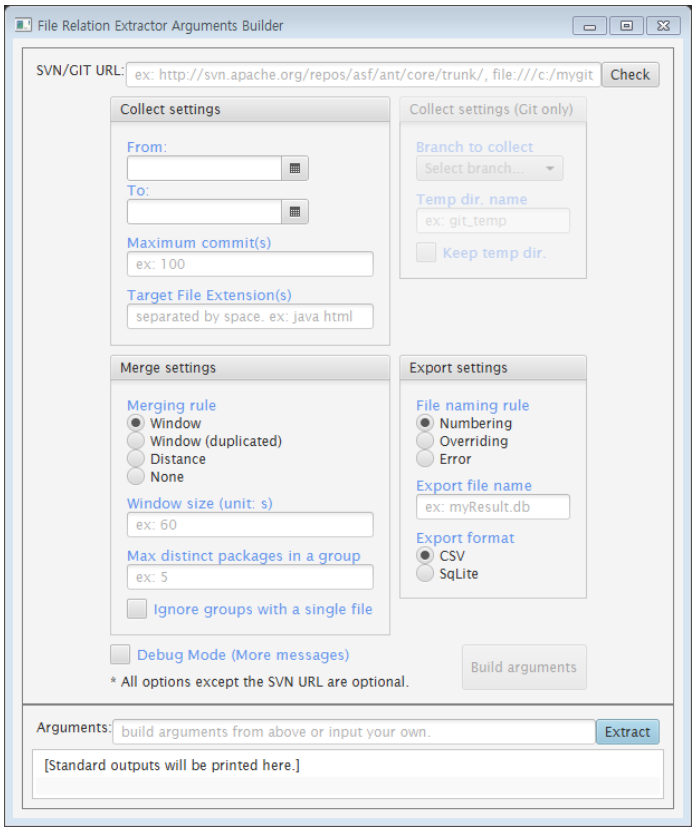


그림 1 패키지 기반 동시 변경 정보 계산을 위한 Java GUI 도구

3.2.1 수집 단계

수집 단계에서는 사용자가 지정한 로컬 혹은 원격의 SVN 혹은 GIT 저장소에서 리비전 정보들을 수집한다. 각 리비전 정보에는 리비전 생성 일자, 리비전 생성자, 변경된 파일 목록이 포함된다. 정보 수집 시 특정한 기간에 포함된 리비전만 지정할 수 있으며 변경된 파일 목록으로 원하는 확장자의 파일만 허용할 수 있다.

일반적으로 프로그래밍 언어별로 언어 고유의 확장자를 가진 파일들이 존재하며 이 파일들이 실질적인 소프트웨어의 구성 요소들이다. 그러므로 다양한 확장자의 파일이 사용되는 버전 관리 시스템에서 변경 결합도를 계산할 때 각 언어에서 사용되는 확장자 외의 파일들은 무시할 필요가 있다. 예를 들어 어떠한 java 파일과 이에 대한 명세가 포함된 html 파일이 한 리비전 내에 동시에 수정되더라도 html 파일로 인해 java 파일의 변경 결합도에 영향을 주어서는 안 된다.

그리고 추가로 추출 결과를 향상하기 위해 패키지 명에 특정 문자열이 포함된 경우 해당 파일을 필터링할 수 있다. 유닛 테스팅을 적용한 프로젝트의 경우 버전 관리 시스템 안에 주 언어로 작성된 모듈의 소스코드 외에도 주 언어로 작성된 모듈의 테스트 코드도 포함되어 있다. 하지만 이 파일들은 실질적인 소프트웨어의 구성 요소가 아니므로 변경 결합도 계산 시 제외가 되어야 한다. 일반적으로 이러한 파일들은

'test'라는 패키지 명을 포함하고 있으므로 패키지 명 필터를 이용하여 쉽게 제외할 수 있다.

3.2.2 병합 단계

병합 단계에서는 수집 단계에서 추출한 리비전들 중에서 짧은 시간 내에 한 사람에게 연속적으로 제출된 것을 그룹으로 병합하는 작업을 진행한다. 이는 본래 하나의 리비전으로 작업이 되어야 하나 여러 개의 리비전으로 나누어진 리비전들을 처리하기 위함이다.

여러 개의 디비전을 병합하기 위해서는 일정 시간 범위 안에 포함된 리비전들을 선택하는 타임 윈도우(time window) 방식을 사용한다. 타임 윈도우는 고정된 길이의 타임 윈도우(fixed time window)를 사용하는 방식과 슬라이딩 타임 윈도우(sliding time window)를 사용하는 방법 2가지를 제공하며 인자를 통해 어떤 방법을 사용할지 선택할 수 있다. 고정 길이 타임 윈도우 방식의 경우 아직 그룹에 포함되지 않은 가장 오래된 리비전을 기준으로 윈도우 크기 내에 포함된 모든 리비전을 하나로 묶는 것을 반복하는 방법이다. 슬라이딩 타임 윈도우 방식의 경우 위와 비슷하지만, 그룹 내에 새로이 포함된 리비전을 기준으로 타임 윈도우 내에 포함되는 리비전을 같은 그룹에 포함하는 작업을 새로운 리비전이 포함되지 않을 때까지 반복하여 수행한다[7].

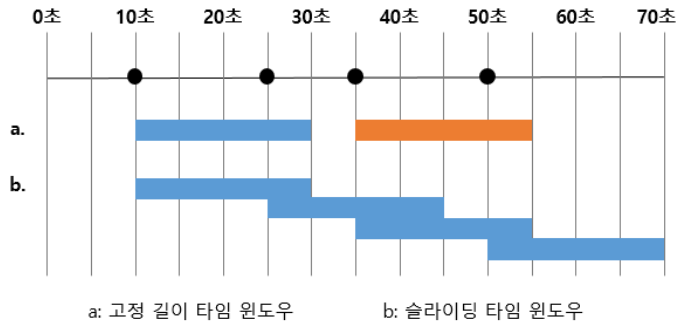


그림 2 고정 길이 타임 윈도우와 슬라이딩 타임 윈도우의 비교

그림 2와 같이 한 사람이 리비전을 10, 25, 35, 50초에 제출하고 윈도우 크기는 20초를 사용한다고 가정하자. 이 경우 고정 길이 타임 윈도우의 경우는 10, 25초의 리비전이 하나의 그룹으로, 35, 50초의 리비전이 다른 그룹으로 병합이 된다. 하지만 슬라이딩 타임 윈도우의 경우에는 모든 4개의 리비전이 하나의 그룹으로 병합이 된다.

위의 예에서 확인할 수 있듯이 동적인 방법을 사용하게 될 경우 본래 하나로 추정되는 리비전들을 하나의 그룹으로 묶을 가능성이 커지나 윈도우의 크기가 커지게 될 경우 그렇지 않은 리비전들도 병합을 하는 문제가 발생할 수 있다. 따라서 어떤 방법과 어떤 윈도우 크기를 사용하는지에 따라 실험 결과가 달라질 수 있다.

병합을 마친 후에는 동시 변경 정보 추출에 적합하지 않은 그룹들을 제외한다. 우선 파일이 1개로만 구성된 그룹의 경우 추출할 패키지 기반 동시 변경 정보가 없음으로 제외한다. 또한, 리팩토링 도구로 자동으로 소스코드를 수정한 리비전이나 여러 리비전들이 하나로 합쳐지는 병합 리비전의 경우 연관이 없는 다량의 파일들이 하나의 리비전에 포함될 수 있다. 이러한 리비전의 경우에도 결과의 품질을 떨어뜨릴 수 있으므로 지정된 숫자보다 많은 패키지가 포함된 그룹도 제외한다.

3.2.3 추출 단계

추출 단계에서는 그룹별로 파일들의 패키지 기반 동시 변경 정보를 추출하고 이를 파일에 따라 시간대별로 누적된 값을 저장한다.

3.2.4 내보내기 단계

내보내기 단계에서는 지금까지 생성한 데이터들을 CSV와 SQLite 중 지정된 파일 형식으로 저장한다. 생성된 데이터의 종류가 많으므로 단일 파일에 모든 데이터를 저장하는 것이 아니라 하나의 디렉토리 안에 데이터별로 파일을 저장한다. 내보내기 결과 생성되는 파일들과 내용은 다음과 같다.

- AccumulatedCounts: 소스 코드 변경 이력에 포함된 파일들에 대해 일자에 따라 누적된 패키지 기반 동시 변경 정보 값
- Groups: 각 그룹의 고유 ID 값과 생성 일시
- Files: 각 그룹에 포함된 파일들
- Revisions: 각 리비전의 소속 그룹ID, 생성 일시, 리비전 메시지(변경 내용)
- LatestRevision: 가장 최신 리비전에서 변경된 파일 목록
- Pairs: 모든 파일 쌍에 대해 함께 변경된 리비전 개수
- Options: 프로그램 실행 시 사용된 인자들

3.2 변경 결합도 추출

이전 프로그램을 실행하여 얻은 CSV형태의 패키지 기반 동시 변경 정보를 이용하여 변경 결합도를 계산하는 Java 프로그램을 구현하였다.

변경 결합도 추출의 목적은 리팩토링 대상을 찾기 위함이기 때문에 모든 파일에 대해서 변경 결합도를 계산하지 않고 이전 단계에서 생성된 LatestRevision 파일을 참조하여 가장 최신 리비전에 포함된 파일들에 대해서만 계산한다. 이 파일들에 대해 정의에 따라 각각 그룹 내에서 자신을 제외한 모든 파일에 대해 다른 패키지 파일의 비율을 계산하고 변경 결합도를 계산한다. 이를 의사 코드로 나타내면 다음과 같다.

```
foreach file in latest_revision:
    per_group_metric[file] = new list()
    foreach group in groups:
        if file not in group:
            continue
        s = same_package_in_group(group, file)
        o = other_package_in_group(group, file)
        per_group_metric[file].append(o / (s + o))
    change_coupling[file] = ave(per_group_metric[file])
```

4. 실험 결과

4.1 실험 환경

Github에 등록된 Hadoop 프로젝트에 대한 Git 버전 관리 시스템에서 0.19 브랜치[8] 에 대해 실험을 진행하였다. 패키지 기반 동시 변경 정보 추출 도구에서는 다음과 같은 인자를 사용하였다.

- **기간**: 모든 기간
- **허용 확장자**: java
- **파일에서 허용하지 않는 문자열**: test, example, docs
- **타임 윈도우**: 60초, 고정 길이 타임 윈도우
- 한 그룹당 최대 패키지 수: 5개
-

4.2 결과 데이터

실험 환경에 기록된 인자에 따라 패키지 기반 동시 변경 정보 추출 도구를 이용하여 패키지 기반 동시 변경 정보를 추출하였고 이에 대해 변경 결합도 추출 도구를 실행한 결과는 표 1과 같다. 단, 신뢰도가 떨어지는 결과를 제외하기 위하여 5개 이하의 집합에 포함된 파일들을 제외하였다. 또한, 결과 분석을 위하여 결과를 평균값 순으로 정렬한 후 상위 7개의 파일만 기재하였으며 DSM과의 비교를 위하여 파일마다 상위 패키지 이름을 추가로 기재하였다.

표 1 패키지 기반 동시 변경 정보 추출 실험 결과

parent	name	ave	cnt
hdfs	SRC/HDFS/*/HDFS/PROTOCOL/DATANODEINFO.JAVA	1	5
hdfs	SRC/HDFS/*/HDFS/TOOLS/DFSADMIN.JAVA	1	5
util	SRC/CORE/*/UTIL/STRINGUTILS.JAVA	0.988	5
hdfs	SRC/HDFS/*/HDFS/DFSCLIENT.JAVA	0.939	10
hdfs	SRC/HDFS/*/HDFS/DISTRIBUTEDFILESYSTEM.JAVA	0.928	7
fs	SRC/CORE/*/FS/FILESYSTEM.JAVA	0.775	6
hdfs	SRC/HDFS/*/HDFS/SERVER/DATANODE/DATANODE.JAVA	0.765	17

*: ORG/APACHE/HADOOP

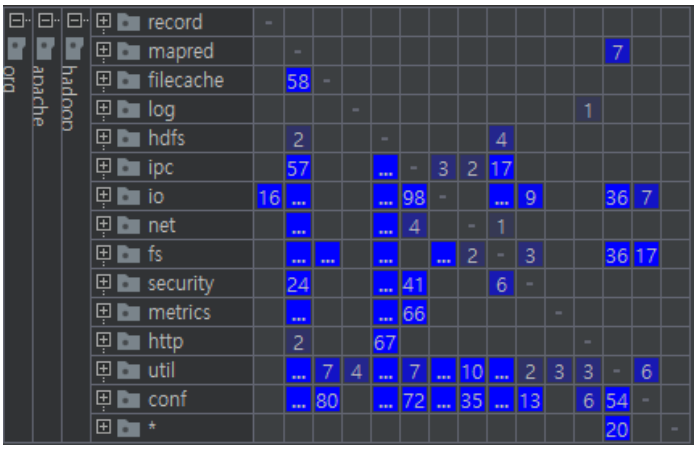


그림 3 실험 대상 프로젝트의 패키지 수준에서의 DSM

4.3 DSM과의 결과 비교

개발 및 유지보수 단계에서 구조적 결합도를 사용하는 소프트웨어 분석 도구로는 DSM(Design Structure Matrix)이 있다. DSM은 프로젝트의 소스 코드로부터 소프트웨어 구조를 추출한 후 각 모듈 간의 의존성을 행렬 형태로 표현하여 시각적으로 쉽게 모듈 간의 의존성을 파악할 수 있다[5]. 행렬의 각 행과 열은 하나의 모듈을 의미하며 각 셀은 세로줄에 해당하는 모듈의 가로줄에 해당하는 모듈에 대한 의존 수치를 나타낸다. 이 수치는 의존 관계가 클수록 더 높은 값이 사용된다.

Intellij IDE의 DSM 플러그인을 통하여 동일 프로젝트의 DSM을 패키지 수준에서 추출한 결과는 그림 3과 같으며 Intellij IDE의 Statistic 플러그인을 통하여 구한 패키지별 코드 줄 수(LOC)와 의존 합을 함께 정리한 결과는 표 2와 같다.

변경 결합도 추출 결과 중 가장 빈번하게 나타난 hdfs 패키지의 경우 코드 수에 비해 의존이 매우 적게 나타난다는 것을 확인할 수 있다. 따라서 변경 결합도를 이용하여 의존도가 높다고 추정된 파일은 단순히 파일과 코드가 많아서 선택된 것이 아니라는 것을 알 수 있다.

표 2 패키지 별 의존 합과 LOC 및 파일 수

패키지이름	의존 됨	의존 함	의존 합(a)	LOC(b)	파일 수	a / b
record	0	16	16	5474	49	0.002923
mapred	7	6020	6027	29506	208	0.204264
filecache	58	257	315	421	1	0.748219
log	1	4	5	106	1	0.04717
hdfs	6	3408	3414	23580	109	0.144784
ipc	225	288	513	1795	8	0.285794
io	2831	1264	4095	10325	84	0.39661
net	342	49	391	1458	13	0.268176
fs	4294	713	5007	10104	70	0.495546
security	224	27	251	262	3	0.958015
metrics	883	3	886	2994	22	0.295925
http	69	10	79	231	3	0.341991
util	1177	133	1310	2425	30	0.540206
conf	2105	30	2135	666	3	3.205706
합계	12222	12222	24444	89347	604	0.273585

변경 결합도 평균값이 가장 높게 나온 7개의 파일에 대한 개별 DSM에서의 의존 합과 코드 줄 수 결과는 표 3과 같다.

표 3 변경 결합도 상위 7개 파일의 의존 합과 LOC

이름	의존 합(a)	LOC(b)	a / b
DatanodeInfo	508	221	2.298643
DFSAdmin	181	549	0.32969
StringUtils	216	390	0.553846
DFSClient	973	2216	0.439079
DistributedFileSystem	346	275	1.258182
FileSystem	1975	782	2.525575
Datanode	1047	1034	1.012573
합계	5246	5467	0.959576

7개의 파일의 코드 줄 수에 대한 의존 합 비율이 전체 파일에 대한 비율인 0.273585보다 약 3.5배 높다는 것을 확인할 수 있다. 이를 통하여 변경 결합도가 높은 파일들이 구조적인 결합도가 높다는 것을 확인할 수 있으며 변경 결합도를 이용한 리팩토링 대상 선별이 유효함을 알 수 있다.

5. 결론 및 향후 연구 계획

이 연구를 통해 다양한 인자를 통하여 정제된 패키지 기반 동시 변경 정보를 추출할 수 있었고 이를 기반으로 하여 파일간의 패키지 관계를 이용한 변경 결합도를 구할 수 있었다. 그리고 다른 소프트웨어 구조 분석 도구인 DSM과의 비교를 통하여 이 연구에서 도출해낸 변경 결합도가 잠재적으로 문제가 있는 파일을 유효하게 선별해 낼 수 있음을 확인하였다.

향후 패키지 기반 동시 변경 정보 추출 도구에서의 인자들이 변경 결합도 계산 시 어떤 영향을 끼치는지에 대한 실험과 변경 결합도 추출 시 다른 수식을 이용한 실험을 진행하여 더욱 정확한 변경 결합도 추출을 시도할 수 있을 것이다.

또한, 본 연구에서의 결과 분석은 패키지 관계를 반영한 변경 결합도를 이용하여 리팩토링 대상을 추천하는 것이 유효함을 보여주었다. 하지만 패키지 관계를 반영하는 것이 그렇지 않은 결과와 어떤 차이가 있는지 설명하지 않고 있다. 따라서 패키지 관계를 이용하지 않을 경우의 실험 결과를 비교 대상으로 추가하여 패키지 관계를 반영하는 것이 어떠한 차이를 발생시키는지 확인할 것이다.

Acknowledgment

본 연구는 한국연구재단 기초연구사업(과제번호 NRF-2014R1A2A2A01005519)의 지원을 받았습니다.

참고문헌

[1] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of application software maintenance," Communications of the ACM, vol. 21, no. 6, pp. 466-471, Jun. 1978.

[2] J. Ratzinger, T. Sigmund, and H. C. Gall, "On the relation of refactorings and software defect prediction," pp. 35-38, Oct. 2008.

[3] T. Mens and T. Tourwe, "A survey of software refactoring," IEEE Transactions on Software Engineering, vol. 30, no. 2, pp. 126-139, Feb. 2004.

[4] L. C. Briand, J. W. Daly, and J. K. Wust, "A unified framework for coupling measurement in object-oriented systems," IEEE Transactions on Software Engineering, vol. 25, no. 1, pp. 91-121, 1999.

[5] A. Yassine, "An introduction to modeling and analyzing complex product development processes using the design structure matrix (DSM) method," Urbana, vol. 51, no. 9, pp. 1-17, 2004.

[6] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship betweenscoupling and software defects," 2009 16th Working Conference on Reverse Engineering, 2009.

[7] T. Zimmermann, "Preprocessing CVS data for fine-grained analysis," 'International Workshop on Mining Software Repositories (MSR 2004)' W17S Workshop - 26th International Conference on Software Engineering, 2004.

[8] apache, "Apache/hadoop," GitHub, 2016. [Online]. Available: https://github.com/apache/hadoop/tree/branch-0.19. Accessed: Nov. 9, 2016.

[9] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," IEEE Transactions on Software Engineering, vol. 31, no. 6, pp. 429-445, Jun. 2005.

[10] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," Proceedings Fifth European Conference on Software Maintenance and Reengineering, 2001.

효율적인 리팩토링을 위한 조직 특성의 품질 지표 기반 우선순위 선정 자동화

박지훈 ○ 손현승 박보경 이진협 김영철
홍익대학교 컴퓨터정보통신공학과 소프트웨어공학연구실
{pjh, son, park, ljh, bob}@selab.hongik.ac.kr

Tailoring an Automatic Priority based on quality metrics of an organization for Effective Refactoring

Jihoon Park○ Hyun Seong Son Bo Kyung Park Jin Hyub Lee R. Youngchul Kim
SE Lab, Hongik University

요 약

개발 완료된 소프트웨어도 필연적으로 추가적인 요구사항이나 변경할 것들이 생긴다. 이런 경우 유지보수에 투자되는 시간과 비용이 많이 차지한다. 하지만 국내 벤처/중소기업은 구현중심으로 유지보수에 대한 투자가 부족하다. 문제는 각 기업마다 조직 특성에 맞는 품질 지표나 그에 관련된 우선순위체계가 없는 경우가 많다. 이는 기업 조직에 알맞은 리팩토링 우선순위를 감별과 산출하는 기법을 제안한다. 본 논문에서는 리팩토링을 위한 Bad Smell기반 우선순위선정 프로그램 자동화를 구현하였다. 식별된 부분들을 리팩토링으로 효율적인 유지보수가 가능하다.

1. 서론

소프트웨어가 개발되어 사용자들이 사용하게 되면 필연적으로 추가적인 요구사항이나 수정해야 할 기능들이 생기기 마련이다. 이 중 유지보수에 들어가는 비용만 전체 개발비용의 절반 이상을 차지하고 있다[11]. 소프트웨어에 대한 연구 중 대부분이 소프트웨어 개발에 관련된 연구이다. 상대적으로 소프트웨어의 유지보수에 대한 연구는 많지 않은 실정이다[7,8]. 국내 중소기업의 경우 대부분 소프트웨어에 정해진 유지보수체계가 없는 상황이다. 대부분 유지보수 노력의 60%까지를 오류가 어디에 있는가를 식별하는데 사용한다[6].

본 논문에서는 이러한 문제들로 인하여 유지보수가 쉽지 않은 소프트웨어를 가시화한다. 가시화란 소스 코드로 존재하는 소프트웨어를 그래프 형식으로 변환하는 과정을 말한다[1,2,3,4]. 가시화된 소프트웨어에는 Bad Smell을 이용하여 리팩토링 우선순위를 산정할 수 있다. 각 기업에 따라서 더 우선시하는 부분에 중요도를 주어 Bad Smell기반 리팩토링 우선순위선정 자동화 도구를 구현하였다. 기업에서는 이 자동화된 도구를 이용하여 각 Bad Smell마다 레벨을 조정함으로써 리팩토링 우선순위를 산정할 수 있다.

2. 관련 연구

2.1 Refactoring

리팩토링은 기능은 그대로 유지한 채로 내부 구조를 개선하는 방법이다. 프로그램에서 버그가 발생할 확률을 최소화함과 동시에 코드의 가독성을 높일 수 있는 정형화된 방법이다. 본질적으로는 코드가 작성된 후 코드의 디자인을 개선하는 것이다[9].

사용자의 추가적인 요구사항들을 해결하다 보면 본래의 디자인과 달라질 것이다. 그렇기 때문에 잘 만들어진 디자인일지라도 유지보수를 하다 보면 망가질 가능성이 높다.

표 1 프로세스 수준 점수 산정 결과

영역	평가항목	대기업	중소기업
개발	고객 요구사항 관리	67.2	52.1
	분석	78.5	71.2
	설계	94	33.2
	구현	82.7	67.9
	테스트	100	89.9

표 1을 보면 중소기업의 경우는 소프트웨어프로세스(SP) 수준점수에서 설계 부분의 점수가 최하점수이다[10]. 그렇기 때문에 리팩토링이 필요하다. 리팩토링은 시스템을 구축하면서 디자인을 개선한다.

2.2 Bad Smell

Bad Smell은 마틴 파울러가 제시한 리팩토링이 필요한 순간들이다[9]. 표 2는 Bad Smell 중에서도 Method와 Class별로 잠재적 오류발생률이 높다고 판단된 Bad

† 이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548)

Smell항목이다.

표 2 잠재적 오류발생률이 높은 Bad Smell 항목

범위	항목	설명
Method	Long Parameter List	파라미터의 개수가 너무 많다.
	Message Chains	메서드의 호출연결고리가 너무 복잡하다.
	Feature Envy	다른 클래스의 속성을 너무 사용한다.
	Middle Man	클래스가 간단한 위임을 너무 많이 한다.
Class	Lazy Class	사용하지 않는 기능이 존재한다.
	Data Class	필드에 get, set 메서드만 존재한다.
	Large Class	클래스의 인스턴스 변수가 너무 많다.
	Refused Bequest	상속받은 메서드를 모두 사용하지 않는다.

3. 리팩토링 우선순위산정 가시화 프로그램

3.1 오픈 소스를 활용한 역공학

본 논문에서는 기존 연구의 결합도를 표시한 Class Diagram형식에서 Bad Smell 추출을 추가하였다[1,2,3,4].

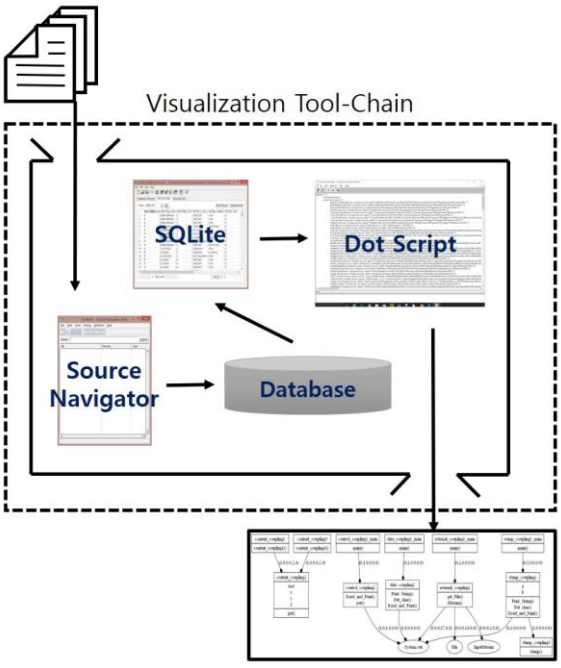


그림 1 Visualization Tool-Chain의 구성도

그림 1은 Tool-Chain의 구성도이다. 오픈 소스인 Source Navigator를 이용하여 JAVA 코드를 분석하면 생기는 여러 개의 SN파일 중 본 논문의 가시화 프로그램에 사용되는 파일은 총 6개이다. 추출된 파일의 데이터들은 재사용을 위해 새로 구성한 데이터베이스에 다시 저장한다. 데이터베이스에는 총 7개의 테이블이 있다. 6개의 SN파일들에 있던 정보들을 각각의 6개의 테이블마다 저장한다. 하나의 테이블을 더 생성하여 query문을 이용하여 추출한 결합도의 정보를 추가로 저장한다. 데이터베이스에 저장된 데이터들을 query문을 이용하여 리팩토링 우선순위가 높은 Bad Smell 항목을 찾아낸다. 각 Bad Smell은 사용자가 정한 기준에 따라 자유롭게 찾아낼 수 있다. 찾아낸 Bad Smell 항목들과 다른 정보들을 그래프로 그리기 위해 오픈 소스인 Graphviz를 이용한다. Graphviz는 Dot Script언어로 실행되기 때문에 정제된 데이터들을 이용하여 DotScript언어로 변환해주어야 한다. 본 논문에서는 이러한 과정들을 하나의 Tool-Chain으로 구축하여 자동으로 모든 과정이 진행되도록 하였다.

3.2 데이터베이스 구성

본 논문에서는 SNDB파일 들에서 추출해낸 데이터들을 재 정제하였다. 데이터베이스를 새로 만들어 저장하였고 다음은 그에 대한 설명이다.

그림 2는 클래스와 메서드의 정보가 저장된 데이터베이스이다. 4개의 테이블에 저장되어 있으며 각각 클래스, 인스턴스변수, 로컬변수, 메서드에 대한 데이터들이 저장되어 있다. SNDB_CL 테이블에는 클래스의 이름, 파일 경로, 클래스의 속성 등이 있다. SNDB_IV 테이블에는 인스턴스 변수가 소속된 클래스의 이름, 인스턴스변수의 이름, 파일 경로, 인스턴스변수의 속성 등이 있다. SNDB_LV 테이블에는 로컬변수의 이름, 파일 경로, 로컬변수의 속성 등이 있다. SNDB_MD 테이블에는 메서드가 소속된 클래스의 이름, 메서드의 이름, 파일 경로, 메서드의 속성, 메서드의 리턴 값, 메서드의 파라미터에 대한 정보 등이 있다.

Name	Object	Type
C:/Java/visualization/test/Test		
SNDB_CL	Table	
NAME	Field	TEXT
START_LINE_NO	Field	TEXT
PATH	Field	TEXT
END_LINE_NO	Field	TEXT
ATTRIBUTES	Field	TEXT
SNDB_IV	Table	
CLASS_NAME	Field	TEXT
VARIABLE_NAME	Field	TEXT
START_LINE_NO	Field	TEXT
PATH	Field	TEXT
END_LINE_NO	Field	TEXT
ATTRIBUTES	Field	TEXT
SNDB_LV	Table	
VARIABLE_NAME	Field	TEXT
START_LINE_NO	Field	TEXT
PATH	Field	TEXT
END_LINE_NO	Field	TEXT
ATTRIBUTES	Field	TEXT
SNDB_MD	Table	
CLASS_NAME	Field	TEXT
METHOD_NAME	Field	TEXT
START_LINE_NO	Field	TEXT
PATH	Field	TEXT
END_LINE_NO	Field	TEXT
ATTRIBUTES	Field	TEXT
RETURN_TYPE	Field	TEXT
ARGUMENT_TYPES	Field	TEXT
ARGUMENT_NAMES	Field	TEXT

그림 2 클래스와 메서드 정보 DB

그림 3은 각 모듈간의 호출, 상속, 결합도 정보다 저장된 데이터베이스이다. 3개의 테이블로 구성되어 있으며 각각 호출 정보, 상속 정보, 결합도 정보가 저장되어 있다. SNDB_BY 테이블에는 호출되는 클래스의 이름, 호출되는 메서드나 변수의 이름, 호출하는 클래스의 이름, 호출하는 메서드의 이름, 호출 형식, 파일 경로, 호출 간의 메서드 파라미터의 정보가 있다. SNDB_IN 테이블에는 서브클래스의 이름, 슈퍼클래스의 이름, 파일 경로, 클래스의 속성 등이 있다. SNDB_COUPLING은 앞의 6개의 테이블에 저장된 데이터들을 한번 더 정제한 테이블이다. 결합도 추출 프로그램에 의해서 SNDB파일과 다른 새 데이터들을 생산한다. 결합도에 관련된 모듈들에 대해서 호출하는 모듈, 호출당하는 모듈, 각 결합도에 대한 수치가 저장된다.

Name	Object	Type
SNDB_BY	Table	
REFERRED_CLASS_NAME	Field	TEXT
REFERRED_SYMBOL_NAME	Field	TEXT
REFERRED_TYPE	Field	TEXT
REFER_CLASS_NAME	Field	TEXT
REFER_SYMBOL_NAME	Field	TEXT
REFER_TYPE	Field	TEXT
ACCESS	Field	TEXT
LINE_NO	Field	TEXT
PATH	Field	TEXT
CALLER_ARGUMENT_TYPES	Field	TEXT
REFER_ARGUMENT_TYPES	Field	TEXT
SNDB_IN	Table	
SUB_CLASS_NAME	Field	TEXT
SUPER_CLASS_NAME	Field	TEXT
START_LINE_NO	Field	TEXT
PATH	Field	TEXT
END_LINE_NO	Field	TEXT
ATTRIBUTES	Field	TEXT
SNDB_COUPLING	Table	
CALLER	Field	TEXT
CALLEE	Field	TEXT
DATA	Field	TEXT
STAMP	Field	TEXT
CONTROL	Field	TEXT
EXTERNAL	Field	TEXT
COMMON	Field	TEXT
CONTENT	Field	TEXT

그림 3 각 모듈간의 호출, 상속, 결합도 정보 DB

3.3 우선순위 선정 프로그램

3.3.1 Long Parameter List

표 3 Long Parameter List 추출 설정 프로그램

```
Resultset lpl = statement.excutequery("
                    select ARGUMENT_TYPE from SNDB_MD");
while(lpl.next()) {
     String[] argument=lpl.getString("ARGUMENT_TYPE");
     if(argument.length > count) {
          return true;
     }
}
```

표 3에서 lpl은 SNDB_MD 테이블에서 ARGUMENT_TYPES값들을 가져온다. 가져온 값들은 파라미터의 정보이기 때문에 if문의 count값을 설정한만큼 Long Parameter List를 추출한다.

3.2.2 Middle Man

표 4 Middle Man 예시코드와 추출 프로그램

```
ResultSet mm = statement.excutequery("
                        select RETURN_TYPE from SNDB_MD where
                  RETURN_TYPE in (select REFERRED_CLASS_NAME
                                  from SNDB_BY where CLASS_NAME in
               (select [name] from SNDB_CL) and LINE_NO < 5)");
While(mm.next()){
     count ++;
}
return count;
```

표 4에서 mm은 SNDB_MD 테이블에서 5줄 미만의 클래스가 객체 형식으로 변수에 호출되는 RETURN_TYPE을 가져온다. 가져온 값들의 개수만큼 count를 추가한다. count를 리턴함으로써 count를 받는 곳에서 조건문을 설정한 만큼 Middle Man을 추출한다.

3.2.3 Large Class

표 5 Large Class 추출 설정 프로그램

```
Resultset lgc = statement.excutequery("
                    select count(*) as CNT from SNDB_IV where
          CLASS_NAME in (select [name] from SNDB_CL)");
while(lgc.next()){
     int cnt=Integer.parseInt("CNT");
     if(cnt > count) {
          return true;
     }
}
```

표 5에서 lgc는 SNDB_IV 테이블에서 인스턴스 변수의 개수를 가져온다. 인스턴스변수의 개수만큼 CNT가 저장된다. count 값을 설정한 만큼 Large Class를 추출한다.

3.2.4 Message Chains

표 6에서 mc는 SNDB_BY 테이블에서 호출되는 메서드의 이름을 가져온다. 이 과정을 한번의 값을 가져오는 과정이라고 볼 때 재귀함수로 설정하여 여러 번 수행한다. Message chain은 어떤 값을 가져오기 위해 여러 개여 객체를 거쳐야만 하는 경우이기 때문에 재귀호출을 통하여 거쳐갔던 객체의 개수를 count에 저장한다. count를 리턴함으로써 count를 받는 곳에서 조건문을 설정한 만큼 Message Chains를 추출한다.

표 6 Message Chains 추출 설정 프로그램

```
Public void re_getMC(String sN, String cN, int count){
    Resultset mc = statement.excutequery("
                    select REFERRED_SYMBOL_NAME,
                REFERRED_CLASS_NAME from SNDB_BY
          where REFERRED_SYMBOL_NAME LIKE 'get%'
     CLASS_NAME in (select [name] from SNDB_CL)");
    while(mc.next()){
        String rdsName
                =mc.getString("referred_symbol_name");
        String rdcName
                  =mc.getString("referred_class_name");
        count=re_getMC(rdsName, rdcName, count);
    }
    return count;}
```

3.2.5 Feature Envy

표 7에서 fe는 클래스를 호출하는 get-인 메서드를 체크한다. Cnt에 개수를 저장하고 count값을 설정함으로써 Feature Envy를 추출할 수 있다.

표 7 Feature Envy 추출 설정 프로그램

```
Resultset fe = statement.excutequery("
          select count(*) as CNT from SNDB_BY where
           REFERRED_SYMBOL_NAME LIKE 'get%' and
                                REFERRED_CLASS_NAME in
                          (select [name] from SNDB_CL)")
While(fe.next()) {
     int cnt=Integer.parseInt(fe.getString("CNT"));
     if(cnt > count) {
          return true;
     }
```

3.3 가시화된 Bad Smell

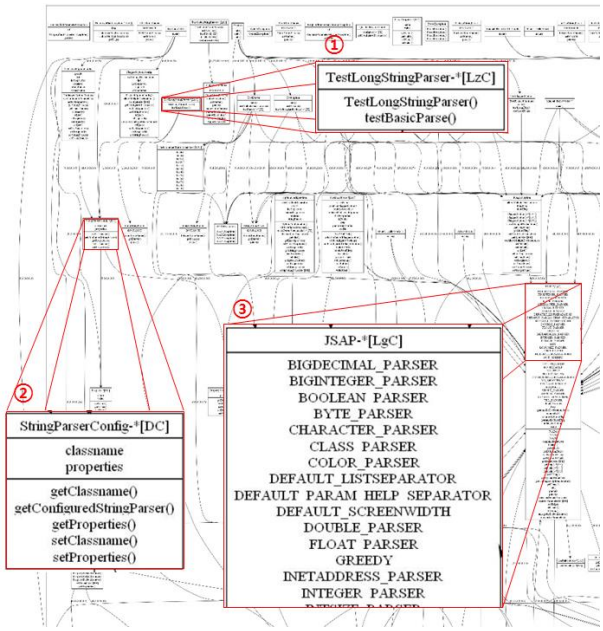


그림 4 추출한 BadSmell을 가시화한 그래프

그림 4는 오픈 소스인 Java Parser를 가시화한 그래프이다. ①번은 test가 아니면 일반적으로 사용되지 않는 클래스라는 것을 알 수 있다. ②번은 get-, set-메서드로만 이루어진 Data Class인 것을 알 수 있다. ③번은 인스턴스 변수가 일정 개수 이상이므로 중복된 코드가 존재할 가능성을 찾아볼 필요가 있다.

4. 결론

대부분의 중소기업들은 개발에만 투자를 집중한 나머지 조직의 품질 지표 기반의 유지보수 중요성에 둔감하다. 본 논문에서 제안한 가시화 시스템을 통해, 각 조직의 품질 지표 기반 맞춤형 리팩토링의 우선순위체계를 제안한다. 각 기업 조직의 맞는 지표 설정으로, 수작업으로 어려웠던 부분을 식별 가시화로 우선순위를 설정한다. 추가적으로 소스 코드를 해독 없이도 소스 코드를 쉽게 파악할 수 있다. 이를 통해 유지보수비용을 절감할 것으로 기대한다.

참 고 문 헌

[1] 박지훈, 권하은, 강건희, 이근상, 김영철, "코드 가시화를 통한 나쁜 코딩 습관 개선 방안 연구", 한국인터넷방송통신학회, 제 13권, 제 1호, pp.47-48, 2015

[2] 박지훈, 김영철, "나쁜 코딩 습관 개선을 위한 코드 가시화 연구", 한국정보처리학회, 제 23권, 제 2호, pp.497-500, 2016

[3] 박지훈, 장우성, 이진협, 손현승, 김영철, "확장된 나쁜 코딩 습관 식별 구현", 한국정보과학회, pp.401-403

[4] 강건희, 손현승, 김영수, 박용범, 김영철, "SW가시화 기반 리펙토링 기법 적용을 통한 정적 코드 복잡도 개선", 한국정보처리학회, 제 21권, 제 2호, pp.646-649, 2014

[5] 권하은, 박보경, 이근상, 박용범, 김영수, 김영철, "코드 가시화부터 모델링 추출을 통한 역공학 적용", 한국정보처리학회, 제 21권, 제 2호, pp.650-653, 2014

[6] 김지혁, 김창재, 류성열, "사례기반의 소프트웨어 유지보수 성숙도 모델 수립 방안", 정보과학회논문지, 제 36권, 제 9호, pp.718-731, 2009

[7] 조현훈, 황만수, 류성열, "효율적인 유지보수를 위한 C원시 코드 재구성에 관한 연구", 한국정보과학회 가을 학술발표논문집, 제 23권, 제 2호, pp.1573-1576, 1996

[8] 김창재, 박제원, "서비스기반의 소프트웨어 유지보수 성숙도 모델", Journal of KIIT, 제 12권, 제 5호, pp.173-184, 2014

[9] Martin Fowler, "Refactoring: Improving The Design of Existing Code", Addison-Wesley, 2002

[10] B.Boehm, Basil, "Software Defect Reduction Top 10 List IEEE Computer"

[11] NIPA, "SW Engineering 공학백서", SW 공학센터, pp.200, 2015

Hadoop Platform에서 UML 모델 기반 영상 처리 소스 코드 자동 생성 방안

고미은 ○ 박용범
단국대학교 컴퓨터학과
mieun0518@gmail.com, ybpark@dankook.ac.kr

Automatic Generation of UML Model-based Image Processing Source Code in Hadoop Platform

Mi-Eun Ko○ Young B. Park
Department of Computer Engineering, Dankook University
Yongin, South Korea

요 약

대용량의 영상을 검색하고 분석하기 위하여 Hadoop Map-Reduce를 이용한 대용량의 영상 데이터 병렬 처리 환경이 주목 받고 있다. Hadoop Map-Reduce는 대용량의 데이터를 처리하기 위한 플랫폼으로, Map-Reduce 작업을 병렬로 처리하여 복잡한 작업을 빠르게 처리 할 수 있다. 기존에 여러 영상 분석 기술과 Hadoop Map-Reduce를 접목하여 구현하려면 개발 시간이 많이 소요되고, 소스코드를 컴파일, 빌드 하기 전에 결과를 확인하기 힘들다. 이러한 문제를 해결하기 위해 본 논문은 플랫폼에 독립적인 개발을 위해 모델링 도구를 활용하여 모델을 만들고, 모델을 기반으로 소스 코드를 자동 생성하는 방안 제안한다.

1. 서 론

최근 소셜 네트워크, 스마트폰, CCTV 등에서 수집되는 영상 데이터의 양이 기하급수적으로 증가하고 있다. 이에 따라 영상 데이터를 실시간으로 인식하고 분석하여 활용하는 기술의 필요성이 증가하고 있다[1]. Hadoop은 병렬로 대규모의 데이터를 분산 처리, 분산 저장하는 자바기반의 오픈 소스 프레임워크로, Map-Reduce 작업을 서로 연결하여 복잡한 작업을 빠르게 처리할 수 있다[2][3]. Hadoop Map-Reduce Framework를 기반으로 하는 Hadoop Image Processing Interface(HIPI)와 영상 처리에 최적화된 라이브러리인 OpenCV는 Hadoop과 연동하여 대규모의 실시간 미디어 데이터를 분석, 활용할 수 있다. 그러나 기존에 여러 영상 분석 기술과 Hadoop Map-Reduce를 접목하여 구현하려면 기존에 작성된 알고리즘을 참고하여 개발자가 수작업으로 전체 코드를 작성해야 하기 때문에 개발 시간이 많이 소요된다. 그리고 소스코드를 컴파일, 빌드 하기 전에 결과를 확인하기 힘든 문제가 존재한다. Model Driven Engineering(MDE)은 UML Model을 통해 추상화를 제공하여 복잡한 소프트웨어 설계를 용이하게 하고, 모델과 도구를 결합하여 문제영역과 소프트웨어구현 사이의 차이를 줄이는데 목적이 있다. 모델 기반 코드 자동 생성은 시스템 검증 시간을 절약하고 수동 코딩으로 인한 에러 줄일 수 있고, 일부 코드를 자동생성 함으로써 생산성을 향상시킬 수 있다[4][5].

본 논문은 HIPI Map-Reduce 소스 코드를 자동화하기 위한 도구와 과정을 제안한다. 이를 위해 모델을 정의하기 위한 모델링 도구와 모델링을 돕기 위한 디자인 가이드, 그리고 소스코드 자동 생성을 위한 소스코드 생성기를 제안한다.

2. 관련연구

2.1 Apache Hadoop Version 2

Apache Hadoop Map-Reduce는 대용량 클러스터의 대량의 하드웨어에서 방대한 양의 데이터를 신뢰성(reliable), 내결함성(fault-tolerant)있는 방식으로 병렬(in-parallel)로 처리하고 응용프로그램을 손쉽게 작성할 수 있는 소프트웨어 프레임워크이다[3].

2.2 Apache Hadoop Map-Reduce

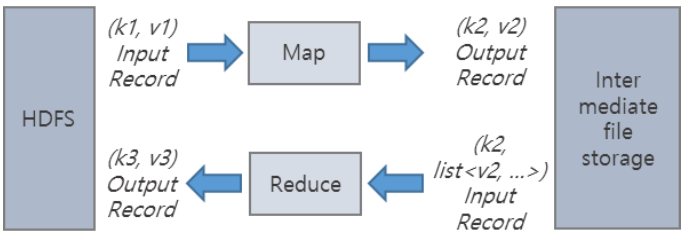


그림 1 Map-Reduce의 기본 동작[4]

Map-Reduce의 동작은 그림 2와 같다. 그림 2는 Map-Reduce Job에 대한 동작을 나타낸다. Map-Reduce Job은 Map과 Reduce의 두 단계로 구성되고, Map과 Reduce의 입출력 데이터는 모두 Key와 Value로 구성된다.

우선 Map의 입력 데이터 레코드 하나의 Key/Value 쌍을 각각 k1, v1라고 가정한다. Map 단계에서는 Map 입력 레코드 k1, v1을 Map을 통해 처리하여 새로운 Key/Value 쌍인 k2, v2로 변환한다. 이후 Map에서 출력된 레코드들에서 같은 key 값을 갖는 레코드들을 모아서 Reduce 단계의 하나의 입력으로 들어간다. Reduce 입력 레코드 k2, list<v2, … >는 Reduce를 통해 처리한 결과를 Reduce 출력 레코드 k3, v3로 변환하여 출력한다.

이러한 Map을 클래스로 구현한 것을 Mapper라 하고, Reduce를 클래스로 구현한 것을 Reducer라 한다[3][4][5].

2.3 Hadoop Image Processing Interface (HIPI)

HIPI는 버지니아 대학에서 대용량 영상 데이터에 대한 분산 컴퓨팅 처리를 위해 제공하는 Apache Hadoop Map-Reduce 라이브러리이다. HIPI는 Apache Hadoop Map-Reduce 프레임워크를 사용하여 영상 처리 인테페이스를 제공하고, 컴퓨터 비전 오픈 소스 라이브러리인 OpenCV와의 통합을 제공한다[6][7].

2.4 Model Driven Engineering (MDE)와 Model Driven Development (MDD)

MDE는 모델을 핵심 artifacts로 간주하고, 모델을 통해 추상화를 제공하여 복잡한 소프트웨어의 설계를 용이하게 한다. 해당 접근법은 체계적인 모델 변환을 지원하는 기술을 사용하여 문제 영역과 소프트웨어 구현 사이의 차이를 줄이는 것을 목적으로 한다. 그리고 모델과 도구를 결합하여 자동화된 개발을 목표로 한다[8][9].

MDD는 추상화를 시스템 설계에 제공함으로써 SDLC를 개선하는 소프트웨어 엔지니어링 방식이다. MDD는 시스템의 구성 및 통합과 관련된 문제를 해결하는 패러다임으로, 플랫폼 독립적인 SW 모델로부터 플랫폼 종속적인 SW 모델로 자동 변환한다. 또한 소스코드를 자동 생성하는 방법으로 원하는 플랫폼에 맞는 SW를 쉽고 빠르게 개발할 수 있다[10][11][12].

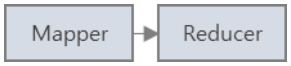
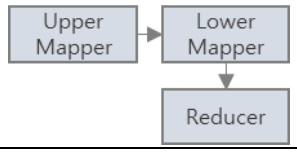
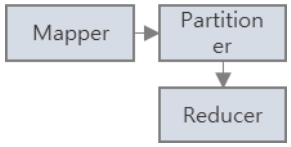
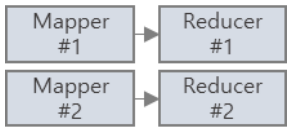
3. Hadoop Platform에서 소스 코드 자동 생성 방안

본 논문은 UML modeling과 Design guide를 통해 automatic code generation을 지원하는 방법 제안한다. 제안 모델은 크게 3가지 부분으로 나뉜다. HIPI Map-Reduce 적용을 위한 예제를 가이드해주는 HIPI Design Pattern Guide, 모델링 도구로 생성된 Model, 생성된 모델을 기반으로 소스 코드를 생성하는 Code Generator로 구성된다.

3.1 HIPI Design Guide

Hadoop Map-Reduce를 위한 디자인 가이드는 많이 제시되고 있다[13]. 그러나 기존에 제시된 Hadoop Map-Reduce 디자인 가이드를 HIPI에 바로 적용하기에 어려움이 있다. 따라서 영상 처리를 위한 Map-Reduce를 유형별로 정의하였고, 이에 따른 HIPI design guide를 제시한다. 제시된 가이드는 Class Diagram으로 가시화된 형태로, HIPI Map-Reduce를 하기 위한 기본 attribute와 method로 구성된다.

표 1 HIPI Design Guide 정의

명칭	설명	
Single Mapper Reducer		기본적인 Map-Reduce 패턴이다.
Chain-Mapper		단계적인 처리가 필요할 경우 사용한다.
Custom Partitioner		Map task의 출력 레코드를 특정 Reduce task로 보낼 수 있다.
Multi-Job		동시에 여러 개의 Job을 구동할 수 있다.

3.2 소스 코드 자동 생성 방안

모델 기반 개발 방법은 UML Modeling을 통해 구조적 관점과 행위적 관점을 modeling하고, 생성된 모델을 기반으로 소스코드를 자동 생성한다. 본 논문은 HIPI Map-Reduce 소스코드 자동 생성 방안을 제안한다. 제안 방법은 구조적 관점, 행위적 관점으로 2가지 관점으로 구성된다. 구조적 관점은 Class Diagram을 통해 정의되고, 행위적 관점은 Activity Diagram을 통해 정의된다. 이러한 제안 방법의 흐름은 그림 2와 같다.

제안 방법의 흐름은 다음과 같다. 우선 제시된 가이드를 바탕으로 모델을 작성한다. 작성된 모델 정보를 바탕으로 XML Document를 생성하고, 생성된 XML 정보를 분석하여 소스 코드 생성기에 전달한다. 소스 코드 생성기는 분석된 결과를 바탕으로 소스코드를 생성한다. 모델 작성시 제시된 가이드를 기반으로 기존에 존재하는 여러 영상 분석 모듈을 추가적으로 작성하여 기존 코드를 재사용 할 수 있다.

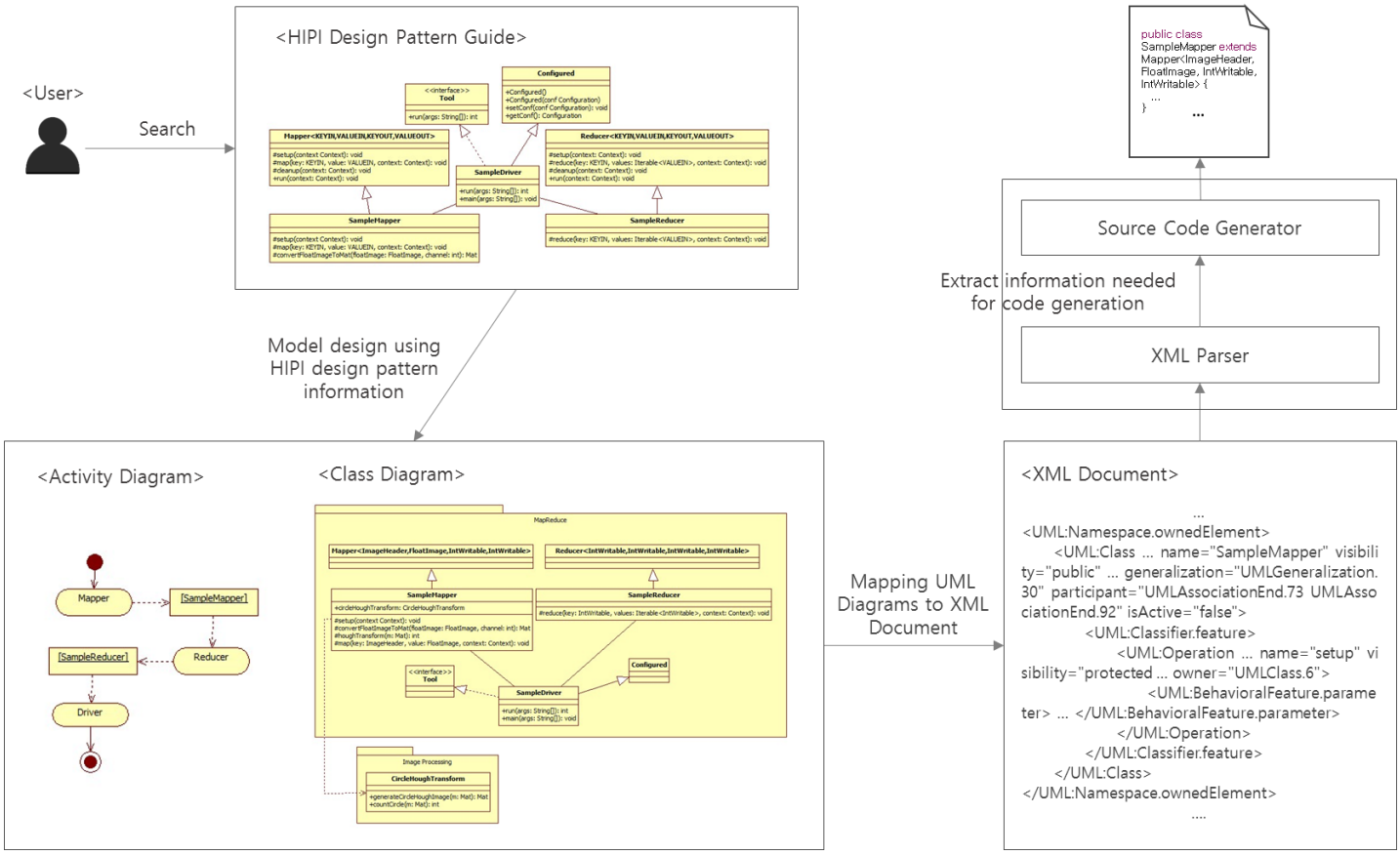


그림 2 HIPI Map-Reduce 소스 코드 생성

4. 결 론

모델 기반 개발 방법은 설계 품질을 향상시키고, 코드 재사용을 가능하게 한다. 모델 기반 자동 소스 코드 생성은 개발 시간을 단축키시고, 수동 코딩으로 인한 에러 줄일 수 있다. 본 논문은 UML을 통해 구조적 관점과 행위적 관점으로 모델을 정의하고, 모델링을 돕기 위한 디자인 가이드를 제시하였다. 그리고 작성된 모델을 기반으로 소스코드를 생성하는 방안을 제안하였다. 본 논문은 제시된 가이드와 모델 기반으로 소스 코드의 일부를 자동 생성하는 형태로, 향후 개발 흐름에 따라 모델 기반 소스 코드 자동 생성에 대한 연구가 필요하다.

Acknowledgement

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 지역신산업선도인력양성사업 성과임(No.NRF-2016H1D5A1909989)

본 연구는 미래창조과학부 및 한국인터넷진흥원의 "고용계약형 정보보호 석사과정 지원사업"의 연구결과로 수행되었음 (과제번호 H2101-17-1001)

참고문헌

[1] 고종국, 배유석, 박종열, 박경, “영상 빅데이터 분석기술 동향”, KIPS. Korea Information Processing Society review, Vol. 21, No. 3, pp. 59-67, May. 2014.
[2] The Apache Hadoop Software Foundation, “hadoop 2”, Nov. 2016, http://hadoop.apache.org/
[3] The Apache Hadoop Software Foundation, “hadoop 2 mapreduce”, Nov. 2016, https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Purpose
[4] 한기용, 『직접 해보는 하둡 프로그래밍』, 이지스퍼블리싱, p119-p121, p172-p182, 2013.
[5] The Apache Hadoop Software Foundation, “Apache Hadoop 2.6.0 API”, Nov. 2016, https://hadoop.apache.org/docs/r2.6.0/api/
[6] HIPI, University of Virginia, “HIPI : Hadoop Image Processing Interface,” Nov. 2016, http://hipi.cs.virginia.edu
[7] Chris Sweeney, Liu Liu, Sean Arietta, Jason Lawrence, “HIPI: A Hadoop Image Processing Interface for Image-based MapReduce Tasks”, 2011.
[8] Abilio G. Parada, Lisane B. de Brisolara, “A Model Driven Approach for Android Applications

Development", SBESC. Brazilian Symposium on Computing System Engineering, pp. 192-197, Nov. 2012.
[9] Juan de Lara, Esther Guerra, Ruth Cobos, Jaime Moreno-Llorena, "Extending Deep Meta-Modelling for Practical Model-Driven Engineering", Computer journal. Vol. 57, No. 1, pp. 36-58, Nov. 2012.
[10] "Developing Applications Using Model-Driven Design Environments" , A. Gokhale, G. Karsai, J. Sztipanovits, S. Neema,
[11] Juan de Lara, Esther Guerra, Ruth Cobos, Jaime Moreno-Llorena, "Extending Deep Meta-Modelling for Practical Model-Driven Engineering", Computer journal. Vol. 57, No. 1, pp. 36-58, Nov. 2012.
[12] Wikipedia, "Model-driven architecture", Oct. 2016, https://en.wikipedia.org/wiki/Model-driven_architecture
[13] Donald Miner, Adam Shook, "MapReduce Design Patterns", in O'REILLY, United States of America, 2012. pp. 19-223.

테스트로부터 소스 코드로의 추적성 향상을 위한 소프트웨어 행위 모델 활용*

백승석 O1, 이정원 2, 이병정 1

서울시립대학교 컴퓨터과학과 1 , 아주대학교 전자공학과 2

{baekss0409, bjlee}@uos.ac.kr1, jungwony@ajou.ac.kr2

Utilizing Software Behavioral Model to Enhance Traceability from Test to Source Code

Seungsuk Baek O1, Jung-Won Lee2, Byungjeong Lee1

Dept. of Computer Science, University of Seoul1,

Dept. of Electrical and Computer Engineering, Ajou University2

요 약

소프트웨어는 설계 단계, 구현 단계, 테스트 단계 등 다양한 단계가 밀접한 관련을 맺으며 완성되어 나가기 때문에 어느 한 산출물에서 변화가 발생하면 관련을 맺고 있는 다른 산출물 역시 동기화가 이루어져 관리 되어야 한다. 이 산출물들의 연관성을 관리하기 위한 일반적인 방법으로는 각각의 산출물에 식별정보를 부여하여 산출물간 추적성을 유지하고 필요시 이를 활용 할 수 있다. 그러나 관리의 실수나 무관심에 의해 잘못된 입력 또는 누락으로 식별정보가 더 이상 추적성을 유지시켜주지 못할 때는 곤란한 상황이 발생할 수 있다. 본 논문은 식별정보에 의한 방법이 소프트웨어 산출물간 추적성을 더 이상 지원해 주지 못할 때 추적성을 회복하기 위한 자동화된 방법을 제안한다. 또한, 본 논문에서는 테스트와 소스 코드의 추적성 향상을 위해 소프트웨어 행위 모델 정보를 활용하였으며 추적성 향상의 효과를 확인할 수 있었다.

1. 서 론

소프트웨어 개발과정에서 생성되는 산출물의 연관성을 추적하는 활동은 소프트웨어 공학 분야에서 매우 중요한 활동으로 볼 수 있다. 왜냐하면 소프트웨어는 어느 하나의 과정에 의해 완성되는 것이 아니며 요구 정의 단계, 설계 단계, 구현 단계, 테스트 단계, 유지 보수 단계 등 다양한 단계가 서로 밀접한 관련을 맺으며 완성되어 나가기 때문이다. 그렇기 때문에 어느 한 부분에서 변화가 발생하면 앞서 언급된 모든 부분에서도 동기화가 이루어져야 어떠한 문제가 생겼을 때 그 문제를 해결하기 위한 중요한 열쇠가 소프트웨어 추적성에 의지해야 하는 것 이라면 어떤 문제에도 현명하게 대처가 가능 할 것이다.

소프트웨어 산출물 추적성을 관리 하기 위한 일반적인 방법으로는 각각의 산출물 내에서 서로 내용이 연관된 부분에 대하여 주석이나 메모기능을 이용해 식별정보를 부여하여 관리하는 방법이 있을 수 있다 [1, 2]. 이렇게 추적성을 관리 하고 있는 조직에서 단위 테스트를 수행하였을 때 결함을 발견했다고 가정해 보자. 앞서 언급한 식별정보를 통해 신속히 관련된 모든 산출물을 추출하여 정정작업을 시도하고 내용을 동기화 시키는데 까지 인력 및 시간적 노력과 비용 소모가 크지 않을 것이다. 그러나 만일 관리의 실수나 무관심에 의해 산출물 간의 식별정보가 올바르지 않거나 누락된 경우 위에서 말한 과정처럼 신속히 대처하기도 어렵고 이로 인해 2차 피해를 입을 수도 있다. 따라서 본 논문은 식별정보에 의존하지 않고 각각의 산출물 내의 정보를 활용하여 단위 테스트 코드로부터 소스 코드로의 추적성을 확보하는 방법과 이를 향상시키기 위한 방법도 같이 제안한다.

본 연구의 학문적 기여도는 다음과 같다.

- 식별정보에 의존하지 않고 단위 테스트와 소스 코드 산출물 간의 자동화된 추적성을 제시한다.
- 소스 코드 산출물 간의 추적성을 향상시키기 위해

* 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업(NRF-2014M3C4A7030504)과 서울시의 재원으로 수행한 서울시 창조전문인력 양성사업(No.CAC1510)의 지원을 받아 수행된 연구임.

소프트웨어 행위 모델을 활용하여 좋은 성능을 확인하였다.

본 논문의 구성은 2장에서 산출물 간의 추적성을 연구한 다른 연구를 소개하고 3장에서 본 논문이 제안한 방법을 소개한다. 4장에서는 본 논문이 제안한 방법으로 사례연구를 진행하고 다른 연구에서 제안한 방법과 비교 검증을 실시한다. 5장에서는 사례연구 결과에 대한 토의를 진행하고 6장에서 결론을 맺는다.

2. 관련연구

소프트웨어 산출물 추적성 연구는 다양한 장소에서 여러 측면으로 연구 되어지고 있다. [3]은 Configuration management log를 활용하여 소프트웨어 소스 코드와 소프트웨어 요구사항 간의 추적성 회복에 대한 방법을 제시하였으며 [4]는 소프트웨어 R&D 프로젝트의 산출물 관리 및 재사용 지원을 위해 소프트웨어 원천기술 개발 사업 수명주기의 각 단계별 산출물의 유형화 과정을 적용한 추적성 방법을 제시하였다. 또한, [5]는 벡터 공간 모델을 개량하여 SysML로 표현된 요구정의서와 State Machine Diagram 그리고 Activity Diagram 간의 추적성 방법을 제시하였고 [6]은 버그리포트 내용과 소스 코드의 주석 내용이 서로 공유하는 정보를 이용해 산출물 간의 추적성을 제시하였다. 이렇듯 소프트웨어 산출물에 대한 추적성을 연구하기 위해 연구자들은 각기 다른 수단을 활용하여 연구를 진행한다. 예를 들면 형상관리 도구를 활용하기도 하며[3, 7] 정보검색 방법[5] 또는 기계학습 방법[8]을 활용하기도 한다. 또한 수단 뿐만 아니라 추적성 연구 대상 산출물 범위도 다양하다. 예를 들면 코드 산출물 간의 추적성[9, 10]에 초점을 맞춰 진행되기도 하며 소프트웨어 개발 계획서부터 인터페이스 요구사항 명세서, 소프트웨어 요구사항 명세서, 소프트웨어 시험 명세서, 소프트웨어 시험 결과서 등 넓은 범위의 추적성[11]에 초점을 맞춰 진행되기도 한다. 이처럼 추적성 연구는 다양한 대상과 수단 그리고 범위를 가지고 계속 연구 되어지고 있다.

다음으로 소개되는 연구는 식별정보에 의한 소프트웨어 산출물 추적 방법과 본 논문에서 제안하는 소프트웨어 테스트로부터 소프트웨어 소스 코드로의 추적성 연구와 유사한 연구이다.

[1]은 소프트웨어 산출물의 변경 이력을 통해서 산출물이 변경되는 과정과 산출물 사이의 연관 관계를 추적할 수 있는 방안을 제시하였다. 이를 위해 릴리즈 관리, 변경 관리, 형상 관리의 통합을 시도하였고 그 중 가장 중요한 릴리즈에 대한 추적은 릴리즈를 구성하는 각각의 기준선을 제시하고 해당 기준선에 속하는 산출물 간의 관계를 추적하는 것을 말한다. 이를 통해 하나의 릴리즈 안에서 존재하는 산출물 간의 추적성을 연결 짓는 목적을 달성했으며 Marcus가 제안한 추적성 관리를 위해 갖춰야할 요건들을 평가 대상으로 평가하여 좋은 결과를 확인한 것을 알 수 있다.

[10]은 단위 테스트와 소스 코드와의 추적성을 확보하기 위해 6가지 방법을 제안하였으며 명명 규칙 의도를 고려한 Naming Convention (NC) 방법, 테스트 케이스의 인스턴스를 활용한 Fixture Element Types (FET) 방법, 메소드 호출에 대한 변화를 이용한 Static Call Graph (SCG) 방법, 테스트 수행 구문의 위치를 분석해 활용한 Last Call Before Assert (LCBA) 방법, 개발자가 소스 코드 내에서 사용한 단어의 어휘를 분석하여 활용한 Lexical Analysis (LA) 방법, 버전 관리 시스템을 이용한 Co-Evolution (Co-Ev) 방법이 소개되었다.

3. 단위 테스트 코드와 소스 코드 산출물 추적성 향상 기법

본 장은 소프트웨어 단위 테스트시 발생한 결함에 대처하기 위해 사용자가 부여한 식별정보에 의존하지 않고 테스트로부터 소스 코드로의 추적성을 확보하고 더 나아가 이를 향상시키기 위한 기법을 단계적으로 소개한다. 자동화된 과정과 소프트웨어 행위 모델을 활용하여 추적성을 향상 시키는 것을 목표로 한다.

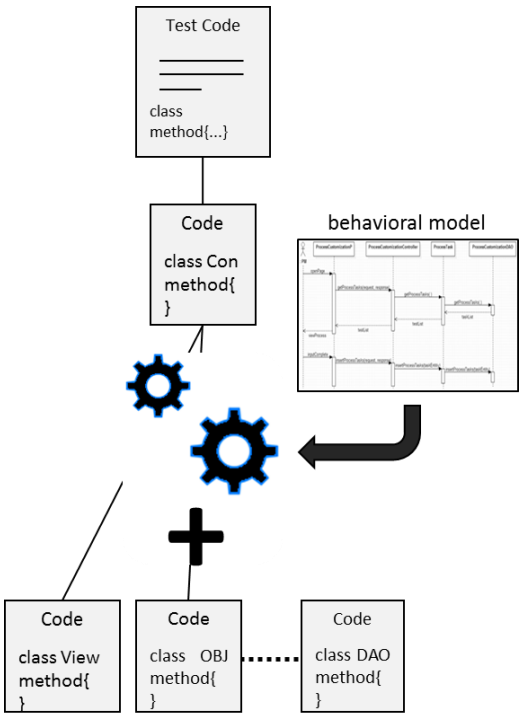


그림 1 테스트로부터 소스 코드로의 추적성

그림 1은 본 논문에서 궁극적으로 말하고자 하는 내용을 도식화 시킨 것 이다. 그림 1에서 알 수 있듯이 먼저 단위 테스트 소스 코드로부터 1차적인 추적성을

확보하고 소프트웨어 행위 모델을 이용하여 최종적으로 추적성을 향상시키는 것을 목표로 한다.

3.1 테스트로부터 테스트 대상 추출

첫번째 단계로는 단위 테스트 대상을 추출하는 것이다. 이를 위해 테스트 소스 코드를 추상 구문 트리(Abstract Syntax Tree)로 변환해 테스트가 수행되어지는 구문을 표 1과 같이 분류한 항목을 토대로 분석하여 테스트 대상 객체 및 테스트 대상 메소드를 추출한다.

표 1 테스트 수행 구문 분류

테스트 수행 구문	테스트 수행 목적
Assert ArrayEquals	배열 A와 B의 동일함을 확인
Assert Equals	객체 A와 B의 동일함을 확인
Assert False	조건 C가 False 임을 확인
Assert True	조건 C가 True 임을 확인
Assert NotNull	객체 A가 Null이 아님을 확인
Assert Null	객체 A가 Null 임을 확인
Assert NotSame	객체 A와 B가 동일한 객체가 아님을 확인
Assert Same	객체 A와 B가 동일한 객체임을 확인
Assert That	객체 A가 P라는 정규식에 부합함을 확인
Assert fail	동작 F가 예외를 발생시킴을 확인

표 1은 테스트 대상 객체와 메소드를 추출하기 위한 단위 테스트 수행 구문에 대한 기준을 제시한다. 표의 좌측은 테스트 수행 구문을 나타내고 우측은 테스트 수행 목적의 설명을 나타낸다.

이 단계의 구체적인 설명을 덧붙이면 최상위 계층인 클래스로부터 멤버 변수, 멤버 메소드를 추상화 시키고 다음으로 메소드 내부 객체까지 추상화로 표현하여 최종적으로 테스트 대상 객체와 테스트 대상 메소드를 추출하는 원리이다.

3.2 테스트 대상과 연관된 산출물 추출

두번째 단계는 첫번째 단계에서 추출한 테스트 대상 객체와 메소드를 정의하는 클래스와 객체의 메소드를 이용하는 비즈니스 로직을 포함하는 클래스를 추출하는 것이다. 이를 위해 테스트 대상 객체명 및 객체의 메소드명으로 정보검색(Information Retrieval) 방법을 이용해 클래스 파일을 모두 추출한다.

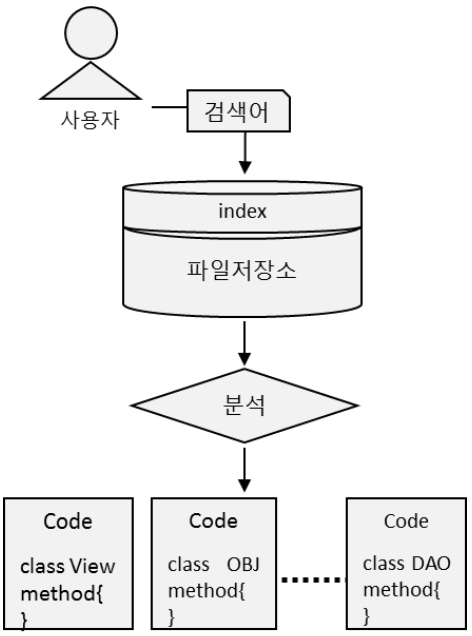


그림 2 정보검색 방법을 통한 파일 추출

그림 2는 불리언 모델을 사용한 정보검색 방법을 통해 테스트 대상과 연관된 클래스를 추출하는 과정을 도식화 하여 나타낸 것이다. 추가적으로 설명을 하면 테스트 대상 객체명과 객체의 메소드명을 검색어로 하여 소프트웨어 소스 파일이 저장되어 있는 파일저장소에서 연관된 클래스를 모두 추출하는 것이다.

아래는 파일 추출 검색에 사용된 불리언 모델에 대한 수식과 설명이다.

$$T = \{t1, t2, \dots, tn, \dots, tm\} \quad (1)$$

(1) T는 색인 단어 t로 구성된 집합

$$D = \{D1, D2, \dots, Dn, Dm\} \quad (2)$$

(2) D는 문서 D로 구성된 집합

$$Q = \text{query} \quad (3)$$

(3) Q는 검색어(query)를 나타냄

$$S = \{Dn | \text{query element of } Dn\} \quad (4)$$

(4) S는 검색어 Q와 일치하는 색인 단어 t를 포함하는 문서 집합

현 단계에서 테스트로부터 소스 코드로의 1차 추적성은 이루어졌다. 이 추적성의 범위를 더욱 확장하기 위해 본 논문은 소프트웨어 행위 모델을 활용하였다. 구체적인 내용은 다음절에서 설명된다.

3.3 추적성 향상을 위한 소프트웨어 행위 모델 활용

세번째 단계는 두번째 단계에서 추출한 클래스에 대한 소프트웨어 행위 모델 정보를 이용하여 소프트웨어 설계 문서와의 추적성도 일부분 확보하고 추적성에 포함시켜야 하는 클래스를 추가적으로 추출하는 작업이다. 본 논문에서는 시퀀스 다이어그램의 속성을 프로그래밍 언어로 제어가 가능한 자료 형태로 변환 하여 자동화된 방법으로 추출하였고 이용한 시퀀스 다이어그램의 속성은 아래 표 2 와 같다.

표 2 사용한 시퀀스 다이어그램의 속성

요 소	설 명
(1) 수명선	상호작용 중 개별 참여 객체를 나타내는 요소
(2) 행위자	시스템 외부에 있는 참가자
(3) 메시지	하나의 객체에서 다른 객체로의 송신 정보
(4) 실행 발생	작업을 실행하는 시간
(5) 반환 메시지	다른 객체로부터 수신 정보

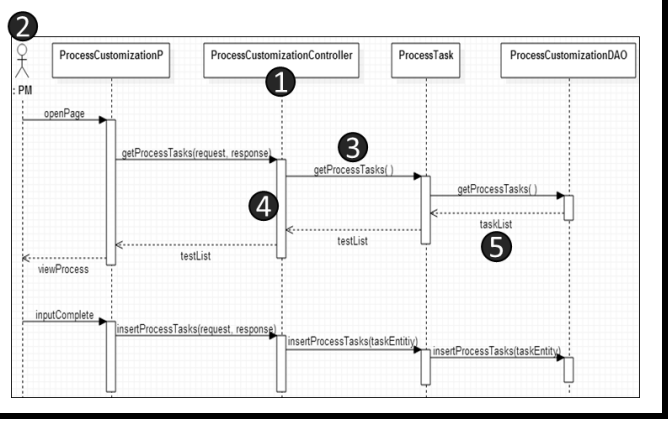


표 2는 추적성 향상을 위해 연구에서 사용된 시퀀스 다이어그램의 속성을 나타낸 것이다. 표의 좌측은 요소에 대한 명칭을 나타내고 요소의 형태는 표의 하단에 번호로 연결 지어 표현했다. 표의 우측은 해당 요소에 대한 설명을 나타낸다.

소프트웨어 행위 모델(시퀀스 다이어그램)을 사용하는 가장 중요한 이유는 객체 간 호출 관계가 명확히 표현되어 있어서 이를 통해 추적성에 추가적으로 포함시켜야 하는 파일을 발견하기 용이하기 때문이다. 덧붙여 설명하면 행위 모델은 어떠한 작업에 대해서 참여하는 모든 객체가 동작 순서에 따라서 수명선 형태로 표시되어 메시지에 의해 어떠한 방향성을 가지고 상호간에 동작이 이루어지는지 파악이 가능하다.

이를 통해 테스트 대상 객체의 메소드가 실행되면서 어느 자원과 어떠한 관련을 맺게 될지를 예상할 수 있다. 예를 들면 테스트 되어진 객체의 메소드가 메시지를 따라 계속 실행되면서 데이터베이스와의 상호작용을 하게 될 수도 있다. 이러한 경우에는 추적성을 데이터베이스 영역까지 확대하여 테이블 정보와 테이블 컬럼 정보, 더 나아가 사용자 정보까지 산출물 추적성을 확대 할 수 있다. 또한, 다른 방향으로의 방향성도 파악이 가능하므로 반대로 반환 메시지를 따라 계속 실행되면서 클라이언트 자원과 상호작용을 하게 되는 것을 발견 할 수도 있을 것이다. 이 내용을 메소드 수준에서 추적성과 클래스 수준에서의 추적성으로 세분화 하여 설명할 수 있으며 다음 내용에서 확인할 수 있다.

3.3.1 메소드 수준에서 추적성

두번째 단계에서 추출한 테스트 대상 객체 메소드를 기술하고 있는 부분을 가진 시퀀스 다이어그램을 통해 추적성을 찾는다.

이를 구체적으로 설명하면 테스트 되어진 메소드를 갖고있는 클래스는 이미 앞의 과정에서 모두 추출하였다 이 클래스들이 해당 메소드를 통해 직접적으로 상호작용을 하고 있는 클래스를 각각의 시퀀스 다이어그램에서 발견하여 각각의 추적성에 포함시킨다. 발견된 클래스와 또 상호작용을 하는 클래스가 표현되어 있으면 그 클래스 역시 추적성에 누적하여 포함시키고 이 클래스 역시 다른 클래스와의 상호작용이 있으면 동일하게 추적성에 누적하여 포함시키는 것이다. 각각의 시퀀스 다이어그램에서 이런 과정을 반복하면 테스트 대상 메소드를 통해 이루어지는 각 시나리오에 연관된 모든 클래스가 방향성을 가진 형태의 추적성을 가지게 된다. 이 모든 과정을 통해 얻어진 각각이 추적성을 수집하면 이게 곧 메소드 수준에서의 추적성이 이루어진 것이라 할 수 있다.

3.3.2 클래스 수준에서 추적성

메소드 수준에서의 추적성 작업이 완료되면 다음으로는 두번째 단계에서 추출한 테스트 대상

객체를 기술하는 시퀀스 다이어그램을 모두 추출하는 것이다. 이를 통해 테스트 대상 객체가 어플리케이션 전체에서 어떠한 역할로써 작용하는 모든 부분이 발견되는 것이다. 이는 곧 테스트 대상 메소드 수준으로의 발견과 함께 추가적으로 테스트 대상 객체의 테스트가 되어진 기능이 아니어도 테스트 대상 객체가 관여하는 모든 부분을 파악하여 추가적으로 추적성을 확장하여 포함할 수 있다.

4. 사례연구

앞서 설명한 기법을 진행중인 프로젝트에 적용하여 본 논문에서 제시한 기법이 얼마만큼 효용성이 있는지 사례연구를 통하여 살펴본다.

4. 1 테스트로부터 테스트 대상 추출

먼저 단위 테스트 코드를 추상 구문 트리로 변환하여 테스트 코드 내에서 테스트가 수행되는 구문을 분석하여 테스트 대상 클래스와 메소드를 추출한다.

```
@Test
public void testGetProcessTasks() throws Exception {

    ProcessTask processTask = new ProcessTask();
    List<TaskEntity> actual = processTask.getProcessTasks();
    Assert.assertEquals("", 2, actual.size());
}
```

그림 3 getProcessTasks 메소드 테스트

그림 3은 JUnit 테스트 프레임워크를 이용하여 ProcessTask 클래스의 getProcessTasks 메소드를 테스트하는 테스트 코드 이다. 그림 3에서는 assertEquals 메소드가 실행 되어지는 부분이 테스트가 수행 되는 부분이다. 이 부분을 통하여 ProcessTask 클래스가 테스트 대상 클래스 라는 것을 알 수 있고 getProcessTasks 메소드가 테스트 대상 메소드라는 것을 알 수 있다. 다음 단계를 위해 이를 추출한다.

```
@Test
public void testInsertProcessTasks() throws Exception {
    ProcessTask processTask = new ProcessTask();
    processTask.insertProcessTasks();
    Assert.fail("function check");
}
```

그림 4 insertProcessTasks 메소드 테스트

그림 4 는 JUnit 테스트 프레임워크를 이용하여 ProcessTask 클래스 insertProcessTasks 메소드가 예외를 발생시키는지 테스트하는 코드 이다. 그림 4 에서는 fail 메소드가 실행 되어지는 부분이 테스트가 수행되는 부분이다. 이 부분을 통하여 ProcessTask 클래스가 테스트 대상 클래스 라는 것을 알 수 있고 insertProcessTasks 메소드가 테스트 대상 메소드라는 것을 알 수 있다. 다음 단계를 위해 이를 추출한다.

4. 2 테스트 대상과 연관된 산출물 추출

다음으로는 정보검색 기능을 지원해주는 Lucene 라이브러리[12]를 이용하여 테스트 대상 객체를 정의하는 클래스와 테스트 대상 메소드가 사용되어지는 비즈니스 로직을 포함하는 클래스를 추출하는 단계이다.

```
Indexing to directory 'index'...
adding C:\target_data\DatController.java
adding C:\target_data\DatElement.java
adding C:\target_data\Dgm.java
adding C:\target_data\DgmDao.java
adding C:\target_data\DgmDaoInterface.java
adding C:\target_data\DgmInterface.java
adding C:\target_data\ProcessCustomizationController.java
adding C:\target_data\ProcessTask.java
adding C:\target_data\RelevanceRule.java
adding C:\target_data\SWTController.java
adding C:\target_data\TestingActivity.java
adding C:\target_data\TestingLevelData.java
adding C:\target_data\TestingOrganizationData.java
adding C:\target_data\TestingTask.java
346 total milliseconds
```

```
2 total matching documents
1. C:\target_data\ProcessTask.java
2. C:\target_data\ProcessCustomizationController.java
```

그림 5 정보검색을 이용한 파일 추출

그림 5는 Lucene 라이브러리를 이용해 파일 저장소의 파일들에 대한 색인작업을 완료하고 생성된 색인을 통해 ProcessTask 객체를 정의한 클래스와 그림 3과 그림 4의 테스트 대상 메소드를 사용하는 비즈니스 로직을 포함하는 클래스를 검색한 결과이다. 그림 5를 통해 다수의 검색 대상 파일들 중에서 두 개의 파일이 테스트 되어진 대상과 관계가 있는 클래스라는 것을 알 수 있다.

그럼 실제로 검색 결과로 출력된 ProcessTask 클래스와 ProcessCustomizationController 클래스를 살펴 보도록 한다. ProcessTask 클래스는 ProcessTask 객체를 정의한 클래스 이여 getProcessTasks 메소드와 insertProcessTasks 메소드를 가지고 있는 클래스이다. ProcessCustomizationController 클래스는 ProcessTask 객체를 생성하여 getProcessTasks 메소드와 insertProcessTasks 메소드를 호출하는 로직을 포함하는 클래스이다.

4. 3 추적성 향상을 위한 소프트웨어 행위 모델 활용

마지막으로 소프트웨어 설계 문서를 활용하여 앞에서 발견된 클래스에 대한 소프트웨어 행위 모델 정보[13]를 이용하여 추적성을 향상시키는 단계이다.

4. 3. 1 메소드 수준에서 추적성

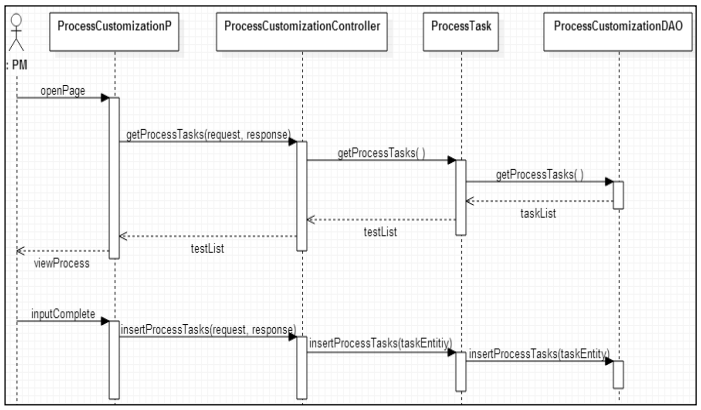


그림 6 연구에 사용한 시퀀스 다이어그램

그림 6는 소프트웨어 설계 문서에서 ProcessTask와 ProcessCustomizationController 클래스에 대해 기술한 시퀀스 다이어그램을 나타내며 시퀀스 다이어그램은 '프로세스 맞춤화' 에 포함된 'Task 목록 가져오기'와 '선택한 Task 적용' 기능들의 제어 흐름을 나타낸다. 이를 JSON(JavaScript Object Notation) 형태로 변환하여 자동화된 방법으로 모델 정보를 추출한다.

그림 6의 시퀀스 다이어그램을 통해서 확인 할 수 있듯이 ProcessCustomizationController 클래스와 ProcessTask 클래스는 getProcessTasks 메시지를 통해 방향성을 가지며 상호작용을 하고 있으며 ProcessTask 클래스 역시 메시지를 통해 ProcessCustomizationDAO 클래스와 상호작용을 하고 있다. 또한 ProcessCustomizationController 클래스는 ProcessCustomizationP 클래스와 메시지를 통해 상호작용을 하고 있다. 이렇게 추출된 클래스를 모두 정리하여 보면 다음과 같다.

사용자가 ProcessCustomizationP 클래스를 통해 Task 목록 가져오기를 요청하면 ProcessCustomizationController 클래스는 요청을 받아들여 테스트 대상 ProcessTask 클래스의 getProcessTasks 메소드를 통해 ProcessCustomizationDAO 클래스를 거쳐 데이터베이스에 저장된 정보를 가져온다.

이와 마찬가지로 사용자가 ProcessCustomizationP 클래스를 통해 선택한 Task 적용을 진행하면 ProcessCustomizationController 클래스는 요청을 받아들여 테스트 대상 ProcessTask 클래스의 insertProcessTasks 메소드를 통해 ProcessCustomizationDAO 클래스를 거쳐 데이터베이스에 선택한 Task를 저장한다.

이렇게 위와 같이 메소드 수준에서 추적성을 향상시켜 정의가 가능하다.

4. 3. 2 클래스 수준에서 추적성

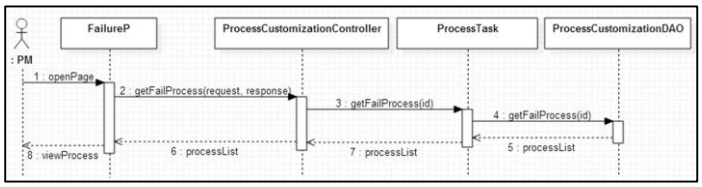


그림 7 테스트 클래스의 추가적인 시퀀스 다이어그램

그림 7은 테스트 대상인 ProcessTask 객체가 존재하는 시퀀스 다이어그램을 추출한 것이다. 이를 통해 테스트 대상 객체의 모든 기능에 대해 상호작용을 하는 추가적인 클래스를 발견할 수 있으며 메소드 수준에서 완성된 추적성 뿐만 아니라 클래스 수준으로 더욱 확장된 추적성을 완성시켜 나갈 수 있다. 그림 7에서 추출된 클래스에 대해서도 앞에서 설명한 것과 동일하게 메시지의 방향성에 따라 추적성을 정리하여 보면 다음과 같다.

사용자가 FailureP 클래스를 통해 실패한 프로세스 목록을 요청하면 ProcessCustomizationController 클래스는 요청을 받아들여 테스트 대상인 ProcessTask 클래스의 getFailProcess 메소드를 통해 ProcessCustomizationDAO 클래스를 거쳐 데이터베이스에 저장된 실패한 프로세스 목록을 가져온다.

메소드 수준에서의 추적성과 클래스 수준에서의 추적성을 함께 활용하면 테스트 대상 메소드와 객체에 어떠한 변화를 적용하고 연관된 클래스 역시 영향을 받을 때 이 클래스들을 효과적으로 추출하는데 용이할 것이라 판단된다.

4. 4 기존에 존재하는 기법과 비교를 통한 검증

앞서 사례연구를 통해 살펴본 테스트 코드(그림 3)와 발견된 클래스(표 3)를 이용하여 기존에 존재하는 기법에도 적용하여 본 논문에서 제안한 기법과 비교하여 추적성에 포함되어질 클래스가 어느정도 발견되는지 살펴본다. 이를 통해 본 논문에서 추적성 향상을 위해 제안한 방법의 효용성을 알아보고자 한다.

표 3 메소드 수준 추적성으로 발견한 클래스 및 메소드

비교 검증에 이용될 클래스와 메소드 목록	
클래스	ProcessCustomizationP
메소드	getProcessTasks

클래스	ProcessCustomizationController
메소드	getProcessTasks
클래스	ProcessTask
메소드	getProcessTasks
클래스	ProcessCustomizationDAO
메소드	getProcessTasks

표 3은 테스트 대상 메소드인 getProcessTasks 메소드를 이용해 본 논문에서 제안한 방법을 적용하여 발견한 클래스와 해당 클래스가 소유한 메소드를 나타낸 표이다. 사례연구를 통해 언급 되어졌듯이 getProcessTasks 메소드를 통해 방향성을 가진 추적성을 확인 할 수 있다. 이 내용을 가지고 [10]에서 제안한 방법과 비교를 시도한다.

[10]의 연구에서는 단위 테스트 코드로부터 구현 소스 코드와의 추적성을 연결하는 방법에 대하여 Naming Convention(NC) 방법과 Last Call Before Assert(LCBA) 방법 등을 소개하였다.

- Naming Convention(NC)

명명 규칙을 고려한 방법으로 테스트 케이스에서 테스트 될 대상의 앞이나 뒤에 'Test'라는 이름을 붙일 것이라 가정하여 추적성을 시도하는 방법이다. 그럼 그림 3의 내용을 이용하여 비교를 시도해보도록 한다.

① 테스트 코드에서 'testGetProcessTasks' 를 식별

② getProcessTasks 메소드와 ProcessTask 객체를 식별

③ ProcessTask 객체를 정의하는 클래스와 getProcessTasks 메소드를 호출하는 로직을 포함하는 클래스의 추적성을 진행

- Last Call Before Assert(LCBA)

Assert 메소드가 호출되기 바로 전에 일어나는 상황을 보고 helper class와 실제로 테스트 되어지는 객체를 구별하여 추적성을 시도하는 방법이다. 그럼 그림 3의 내용을 이용하여 비교를 시도해보도록 한다.

① 테스트 케이스 소스 코드에서 assertEquals 메소드를 식별

② assertEquals 메소드 전에 선언된 getProcessTasks 메소드와 ProcessTask 객체를 식별

③ ProcessTask 객체를 정의하는 클래스와 getProcessTasks 메소드를 호출하는 로직을 포함하는 클래스의 추적성을 진행

본 연구와의 비교 결과는 다음과 같다.

표 4 NC, LCBA 기법과의 비교 검증 결과

NC, LCBA 기법 비교 결과		
NC	클래스	ProcessTask
	메소드	getProcessTasks
	클래스	ProcessCustomizationController
	메소드	getProcessTasks
LCBA	클래스	ProcessTask
	메소드	getProcessTasks
	클래스	ProcessCustomizationController
	메소드	getProcessTasks

표 4의 내용을 보면 첫번째 방법인 NC 방법으로 테스트 대상 메소드인 getProcessTasks를 식별하고 이를 정의한 ProcessTask 클래스를 찾고 이 객체의 getProcessTasks 메소드를 호출하는 ProcessCustomizationController 클래스를 추적하였다. 그리고 두번째 방법인 LCBA 방법으로 테스트 수행 구문인 assertEquals 메소드를 이용해 테스트 대상 메소드인 getProcessTasks를 식별하고 이를 정의한 ProcessTask 클래스를 찾고 이 객체의 getProcessTasks 메소드를 호출하는 ProcessCustomizationController 클래스를 추적한 것을 확인할 수 있다.

5. 토 의

본 논문에서 제안한 방법에 대하여 타 연구와의 비교 검증을 실시한 결과를 분석하고 본 논문에서 제안한 방법의 한계점을 살펴본다.

5. 1 사례연구 결과에 대한 분석

표 4는 [10]의 연구에서 제안한 Naming Convention 방법과 Last Call Before Assert 방법 그리고 본 논문에서 제안한 방법을 비교 검증한 결과이다. 기존에 제안된 Naming Convention 방법을 이용하여 'Test'라는 단어가 포함된 메소드를 찾고 이를 바탕으로 하여

테스트 대상 클래스를 추적하여 소스 코드를 2개 발견할 수 있었다. 그러나 만일 'Test'라는 단어가 테스트 코드 상에 존재하지 않을 시 이 방법은 효과를 보지 못 할 것이다.

그리고 Last Call Before Assert 방법을 이용하여 테스트 수행 구문을 찾아 이를 바탕으로 하여 테스트 대상 클래스를 추적하여 소스 코드를 역시 2개 발견 할 수 있었다. 이 방법은 Naming Convention 방법보다 활용성이 좋지만 테스트 대상 까지만 추적성을 보였다. 마지막으로 본 논문에서 제안한 방법은 앞선 두 방법에서 발견한 소스 코드를 발견하였고 소프트웨어 행위 모델 정보를 이용하여 추가적으로 소스 코드를 2개 더 발견 할 수 있었다. 따라서 소프트웨어 행위 모델을 이용하면 소프트웨어 추적성의 성능이 향상됨을 확인 할 수 있다.

5. 2 한계점

본 논문은 테스트 코드로부터 소스 코드의 추적성 및 이를 향상시키는 방안을 제안하였고 소프트웨어 행위 모델을 이용하여 소프트웨어 산출물 간의 추적성을 향상시켰다. 사례연구를 통해 좋은 결과를 확인 할 수 있었지만 제안한 방법을 추가적으로 다른 프로젝트 대상으로도 적용하여 검증을 실시하면 이 또한 효용성을 시험해보는데 유익한 활동이 될 수 있다. 하지만 실제로 존재하여 운영중인 프로젝트의 테스트 코드나 소프트웨어 행위 모델을 구하기 위해서는 현실적인 어려움이 따른다 따라서 제안한 방법의 효용성을 더 시험하기 위한 추가연구가 필요하다.

6. 결 론

본 논문은 소프트웨어 테스트로부터 소스 코드로의 산출물간 추적성 및 추적성을 향상 시키는 방법을 제안하였다. 본 논문에서 제안한 방법으로는 테스트로부터 테스트 대상 객체와 메소드를 식별, 이를 자동화된 정보검색 방법으로 연관된 소스 코드를 발견해 추적성을 연결하고 추적성을 향상시키기 위해 소프트웨어 행위 모델(시퀀스 다이어그램) 정보를 JSON 형태로 변환하여 정보를 추출하여 나머지 소스 코드들의 발견과 함께 소프트웨어 설계 문서의 내용과 추적성도 일부분 확보 할 수 있었다. 따라서 본 논문이 제안한 방법이 소프트웨어 추적성을 향상시키는데 효용성이 있다는 것을 확인 할 수 있었다. 그러나 토의에도 언급되었듯 현실적인 한계가 있어 여러 프로젝트에 적용시켜 검증하기에는 어려움이 있어 추가적인 연구가 필요한 상황이다.

하나의 소프트웨어가 완성 되어지기 까지 여러 단계를 거쳐야 하고 각 단계에서는 서로 다른 종류의 산출물이 생성 될 것이다. 이들은 서로 관계를 맺고 있으므로 추적성 관리가 철저하게 이루어져야 한다. 그렇기 때문에 산출물들을 통합적으로 관리[14] 하기 위한 시도는 계속 진행 되어져야 한다.

현재는 연구가 초기단계에 있지만 앞으로 연구가 계속 진행됨에 따라서 향후에는 본 논문에서 제안한 방법을 기반으로 하여 부여된 식별정보에 의존하지 않고 자동화된 방법으로 추적성 관리를 하는 프레임워크도 구현 가능 할 것이라 기대한다.

참고문헌

[1] D. Y. Kim, and C. Youn, "Traceability Management Technique for Software Artifacts which Comprise Software Release," KIPS Transactions on Software and Data Engineering, vol. 2, no. 7, pp. 461-470, 2013.

[2] 한소연, 김주영, 류성열, "요구관리산출물과 위험관리산출물의 연계 방안," 한국정보과학회 학술발표논문집, vol. 34, no. 1B, pp. 108-113, 2007.

[3] R. Tsuchiya, H. Washizaki, Y. Fukazawa, M. KAWAKAMI, and K. YOSHIMURA, "Recovering traceability links between requirements and source code using the configuration management log," IEICE TRANSACTIONS on Information and Systems, vol. 98, no. 4, pp. 852-862, 2015.

[4] 최재영, 조영화, 홍장의, "소프트웨어 R&D 프로젝트의 산출물 관리 및 재사용 지원을 위한 문서 분류체계," 한국차세대컴퓨팅학회 논문지, vol. 11, no. 3, pp. 15-27, 2015.

[5] Y. Udagawa, "An Augmented Vector Space Information Retrieval for Recovering Requirements Traceability," In Proc. of International Conference on Data Mining Workshops, pp. 771-778, 2011.

[6] L. Moreno, W. Bandara, S. Haiduc, and A. Marcus, "On the relationship between the vocabulary of bug reports and source code," In Proc. of International Conference on Software Maintenance, pp. 452-455, 2013.

[7] 김대엽, 윤청. "단일 소프트웨어 산출물에 대한 추적성 향상 기법," 한국정보과학회 학술발표논문집, vol. 36, no. 2B, pp. 1-6, 2009.

[8] H. Asuncion, A. Asuncion, and R. Taylor, "Software traceability with topic modeling," In Proc. of International Conference on Software Engineering, vol.

1, pp. 95-104, 2010.

[9] M. Ghafari, C. Ghezzi, and K. Rubinov, “Automatically identifying focal methods under test in unit test cases,” In Proc. of International Working Conference on Source Code Analysis and Manipulation pp. 61-70, 2015.

[10] B. Van Rompaey, and S. Demeyer, “Establishing traceability links between unit test cases and units under test,” In Proc. of European Conference on Software Maintenance and Reengineering, pp. 209-218, 2009.

[11] A. Azmi, and S. Ibrahim, “Formulating a Software Traceability Model for IntegratedTest Documentation: a Case Study,” International Journal of Information and Electronics Engineering, vol. 1, no. 2, pp. 178, 2011.

[12] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in Action: Covers Apache Lucene 3.0*, Manning Publications Co., 2010.

[13] James Rumbaugh, Ivar Jacobson, and Grady Booch, *Unified Modeling Language Reference Manual*, Pearson Higher Education, 2004.

[14] 김병철, “Design of an Integrated Product Management System for Game Software Development,” 한국디지털정책학회, vol. 13, no. 12, pp. 319-324, 2015.

인공지능 소프트웨어 개발을 위한 요구공학 프레임워크: 인지컴퓨팅 시스템을 중심으로

안우람* 김재홍** 김경모** 민상윤***

* (주)삼성전자, KAIST 소프트웨어 대학원
** (주)LG전자, KAIST 소프트웨어 대학원
*** KAIST 전산학과
{rambar,jaehong.kim,gmkim,symin}@kaist.ac.kr

Requirement Engineering Framework for Artificial Intelligence Software Development: Focusing on Cognitive Computing System

요 약

인공지능 시스템은 기존 시스템과 구별되는 특징으로 인지기능적 특징과 데이터 관리적 특징을 가지고 있다. 기존 요구공학은 이와 같은 특징을 활용하지 않으므로 인공지능 소프트웨어에 적합한 새로운 요구공학 방법이 필요하다. 본 연구는 인공지능 소프트웨어의 요구사항을 도출하고 명세하는 요구공학 프레임워크를 제안한다. 사용자가 제시한 비즈니스 요구사항으로부터 시스템 수준의 요구사항을 구체화하기 위해 관계형 마인드 맵과 체크리스트를 활용하였으며 인지기능 요구사항과 데이터 요구사항을 반복적이며 점진적으로 도출하는 방법을 활용한다. 관계형 마인드 맵으로 도출된 요구사항은 이해관계자의 이해를 높이기 위해 클래스 다이어그램의 형태로 명세하는 방법을 제안한다. 논문에서는 요구공학 프레임워크를 적용한 시스템에 대한 사례 연구를 소개한다. 본 연구는 인공지능 소프트웨어의 요구사항을 효과적으로 도출 및 명세하여 개발 성공률을 높이는데 기여할 것이다.

1. 서 론

2016년 3월 구글이 개발한 인공지능(artificial intelligence) 바둑 프로그램 알파고(AlphaGo)가 한국 프로바둑기사 이세돌 9단과의 대국에서 4대 1로 승리하는 사건이 있었다. 세상은 이 사건을 두고 컴퓨터 인공지능이 인간을 넘어서는 초읽기가 시작된 사건으로 기억했고, 소프트웨어 산업계에서는 다시 한번 인공지능 기술에 대해 관심을 나타내기 시작했다.[1]

최근에 인공지능 기술은 학술적 연구 단계를 넘어 비즈니스에 적용 가능한 수준으로 빠르게 발전하고 있다. 과거 인공지능 기술은 기술적 한계로 인간의 인지 및 사고 능력에 미치지 못해 학술 연구 영역에서 벗어나지 못했으나, 최근 딥 러닝(deep learning) 기술의 발전으로 인해 일부 분야에서는 인간에 근접한 수준까지 발전하면서 상업적 활용 가능성이 증대되고 있다. 인공지능 기술은 범용성이 높은 대표적 융합기술로서 자동차, 금융, 의료, 유통 등 다양한 분야에 적용할 수 있으며 이를 기반으로 하는 제품 및 서비스에는 자율주행차, 로보어드바이저(robo-advisor), 의료 자문 서비스, 유통 수요 예측 그리고 개인용 로봇이 있다. 인공지능 기술을 통해 기존의 기술로는 구현할 수 없었던 새로운 제품 및 서비스들이 실용화될 전망이다.[2]

인공지능 소프트웨어는 데이터를 학습해 규칙을 생성하는 귀납법적인 접근으로 연역법적 패러다임의 전통적인 프로그램과는 큰 차이가 있다.[3] 따라서 인공지능 소프트웨어에는 전통적인 요구공학 방법을 적용하기 어렵다는 문제가 있다. 또한 소프트웨어 공학분야는 최근 특정 도메인 별로 관련 기법이 특화되는 추세를 가지고 있다. 인공지능 분야도 효과적 개발을 위한 특화된 소프트웨어 공학 연구가 필요하다.

본 연구에서는 IBM사의 인지컴퓨팅 플랫폼 “왓슨(Watson)”을 대상으로 기존 소프트웨어 대비 인공지능 소프트웨어의 특징에 대해 검토한다. 또한 도출된 인공지능 소프트웨어의 특징에 적합한 요구공학 프레임워크(requirement engineering framework)를 제안하여 요구사항을 효과적으로 도출하고 명세하는 방법을 제안한다. 인지컴퓨팅 시스템인 왓슨은 최신 인공지능 기술을 충실히 구현한 사례로서 본 연구에서

중점적으로 검토하기에 적합하다.

이를 위해 본 연구의 3장에서는 배경 지식으로 인지컴퓨팅 시스템의 두 가지 특징인 '인지기능적 특징'과 '데이터 관리적 특징'에 대해 정의하고, 관계형 마인드 맵(relational mind-map) 기법에 대해 소개한다. 4장에서는 인공지능 소프트웨어 개발을 위한 요구공학 프레임워크를 제안한다. 요구공학 프레임워크는 인공지능 소프트웨어의 요구사항을 도출하는 방법과 도출된 요구사항을 표현하기 위해 클래스 다이어그램으로 변환하는 과정을 포함한다. 5장에서는 본 연구에서 제안한 요구사항 프레임워크를 '쇼핑몰 무인상담 시스템'에 적용해보는 사례 연구를 소개한다. 6장에서 기존의 요구공학 접근법과 본 연구에서 제시한 요구공학 접근법을 비교하는 평가를 기술한다. 마지막으로 7장에서는 향후 연구방향과 본 연구의 결론을 제시한다.

2. 배경 지식

2.1 IBM 왓슨 그리고 인지컴퓨팅

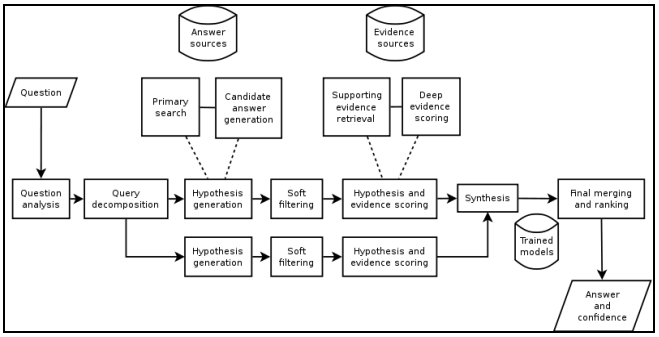
2011년 2월 IBM사의 인지컴퓨팅 시스템인 '왓슨(Watson)'이 제퍼디 퀴즈 쇼(Quiz Show Jeopardy)에서 인간과의 대결에 승리하는 것을 계기로 세상은 인공지능의 한 분야인 인지컴퓨팅 기술에 관심을 가지게 되었다.

인지컴퓨팅은 사람의 뇌가 가진 인지능력을 모방하여 만든 정보처리기술로서 사람의 인지기능은 지각, 행동, 언어, 시각, 기억, 학습, 추론, 의사결정 등을 포함한다. 따라서 인지컴퓨팅은 환경 인식과 행동, 시각 및 언어 처리 능력, 연상 기억, 유연한 학습과 추론, 안정적인 의사결정 등을 수행하는 정보처리기술로 정의될 수 있다.[4] 1997년 5월 IBM사는 인공지능 프로그램인 '딥블루(Deep Blue)'로 체스 게임에서 승리한 이후, DeepQA 프로젝트를 통해 자연어 형식으로 된 질문에 답할 수 있는 질의응답 (question answering) 인공지능 컴퓨터 시스템인 왓슨을 개발하기 시작한다.[5]

2.2 인지기능적 특징

그림 1의 IBM DeepQA 하이레벨 아키텍처(high-level architecture)를 살펴보면, 왓슨의 인지컴퓨팅 기능은 (1)질문을 인식하여, (2)보유하고 있는 지식(데이터)을 기반으로 질문에 대한 가설을 만들고 가설에 맞는 몇 가지 해답을 도출한 후, (3)지식 기반으로 해답 후보들을 평가하여, (4)확률적으로 가장 정확도가 높은 해답을 최종적으로 도출하는 과정으로 구성되어 있다. 이는 사람이 질문을 전달받았을 때, 사람의 두뇌가 인지기능과 지식을 기반으로 질문을 인지하고 해답을 만들어내는 과정을 유사하게 모방하여 설계한 것으로 볼 수 있다.

우리는 인지컴퓨팅 시스템의 첫 번째 특징으로 사람의 정보 처리 능력을 컴퓨터가 모방하는 인지기능으로 선정하였다.



(그림 1) DeepQA High-level 아키텍처

2.3 데이터 관리적 특징

DeepQA 하이레벨 아키텍처의 또 다른 주요 특징은 답변 출처(answer sources), 증거 출처(evidence sources), 학습 모델(trained model)등 데이터 소스(data source)에 대한 고려가 필요하다는 것이다.

왓슨의 경우 데이터 소스로 백과사전, 사전, 뉴스 기사, 문학작품을 포함하여 수백만 개의 문서를 사용했다.[6] 이 데이터는 제퍼디 퀴즈 쇼를 위한 데이터일 뿐 비즈니스 목표, 비즈니스 영역, 언어와 같은 요구사항이 변경되면 시스템을 구성하기 위한 데이터 소스를 전면적으로 재검토해야 한다.

인지컴퓨팅 시스템의 개발 관점에서는 인지기능에 어떤 데이터 소스가 필요한지에 대한 검토를 포함하여 데이터 수집, 저장, 가공, 보안, 분석, 폐기에 이르는 데이터 생애주기 전반에 대한 고려가 필요하다.

본 연구에서는 데이터라는 핵심요소를 단순히 데이터 소스 관점으로만 국한시키지 않고, 데이터 관리(data management) 관점에서 종합적으로 검토하기로 하였다.

2.4 인지기능적 특징과 데이터 관리적 특징의 상관관계

인지컴퓨팅 시스템 구성을 위해서는 먼저 해당 분야의 사람이 해당 업무를 처리하기 위해 수행하는 동작, 사고, 학습 등에 대한 고찰과 분석이 필요하다. 이를 바탕으로 인지기능으로 컴퓨팅 환경에서 활용 가능한 장치 또는 알고리즘을 선정하고, 해당 장치/알고리즘을 학습/구현하거나 동작시키기 위해 필요한 데이터 소스에 대해 검토하는 작업이 필요하다.

구글 딥마인드(DeepMind)의 알파고는 바둑판의 형세를 판단하기 위해 컴퓨터 비전(computer vision) 기술을 활용했고, IBM 왓슨의 경우 문제 인식을 위해

자연어처리(natural language processing)기술을 활용했다. 이와 같이 문제인식이나 학습에 활용 가능한 데이터 소스의 형태에 따라 인지기능으로 활용할 장치나 알고리즘이 변경되는 경우도 발생할 수 있다.

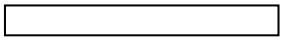
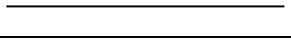
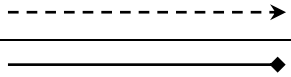
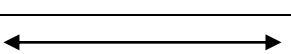
데이터 관리적 특징에서도 마찬가지로 인지기능과 상관관계가 있으므로 인지컴퓨팅 시스템 개발 시에 가용할 수 있는 인지기능에 따라서 데이터의 전처리가 필요하거나 필요한 데이터가 달라질 수 있다.

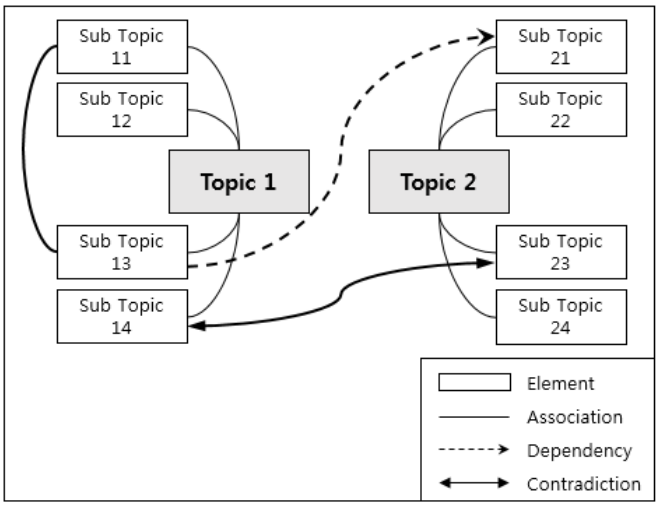
2.5 관계형 마인드 맵(Relational Mind-map)

마인드 맵은 정보를 시각적으로 체계화하여 정리하기 위해 사용하는 다이어그램이다. 노트 필기, 아이디어 도출, 프로젝트 계획, 문서 작성 등 많은 분야에서 마인드 맵을 활용하고 있다.

기존의 마인드 맵은 특정 요소를 하위 요소로 분할할 수는 있지만 두 개의 요소 간의 관계를 정의할 수 없기 때문에 새로운 관계 속성을 정의할 필요가 있다. 본 연구에서는 기존 마인드 맵을 확장하여 요소간의 관계를 다양하게 표현할 수 있는 관계형 마인드 맵을 창안하였다.

(표 1) 관계형 마인드 맵의 표기법

심벌	설명
	Element
	Association
	Dependency
	Composition
	Contradiction



(그림 2) 관계형 마인드 맵의 예

표 1에서 관계형 마인드 맵의 요소와 요소간의 관계를 표현할 수 있는 표기법을 정의하였다. 제안된 표기법을 통해 표현할 수 있는 관계의 속성은 연관(association), 의존(dependency), 합성(composition), 모순(contradiction)이다. 관계형 마인드 맵은 본 연구에서 요구사항을 구체화하고 명세화하는 핵심적인 도구로 활용할 것이다.

그림 2는 관계형 마인드 맵으로 요소간의 관계를 나타낸 예이다.

3. 관련 연구

3.1 데이터 관리 관련 연구

DAMA-DMBOK[7]과 SWEBOK[8]은 각각 데이터 관리와 소프트웨어공학에 대한 지식영역을 개념적 설명과 예시를 통해 체계적으로 요약 정리하였다. 여기서 소개하는 내용은 다양한 도메인에 적용하는 것을 목표로 하고 있기 때문에 다소 추상적이라는 한계를 가지고 있다.

Lavalle et al.[9]은 조직의 데이터 관리 역량을 평가할 수 있는 세 단계의 기준을 제시하였다. 조직의 데이터 관리에 대한 평가와 개선을 위한 방향성을 제공하였다는 점에서 의의가 있지만 데이터를 관리하기 위한 기술적인 부분에 대한 설명이 부족하다.

3.2 인지컴퓨팅 관련 연구

장병탁, 여무송[4]은 전통적인 정보 시스템과 인지컴퓨팅 시스템의 차이점을 설명하였고 지각과 행동지능과 관련된 인지컴퓨팅 연구사례를 소개하였다. 인지로봇에 대한 다양한 소개는 의의가 있었지만 인공지능 소프트웨어에 대한 깊은 고찰이 부족하다는 문제점이 있다.

장병탁, 이동훈[10]은 인지컴퓨팅의 한 분야인 컴퓨터 시각 및 언어 기술에 대해 신경언어학과 인지언어학과 연계한 연구를 소개하였다. 그러나 다양한 연구개발 사례에서 공통적인 특성을 도출하지 못했다.

장병탁, 김현수[11]은 뇌를 닮은 인지컴퓨팅으로 규칙기반 시스템(rule-based systems), 연결 모델, 연결론적 네트워크(connectionist network), 군집코딩(population coding), 동역학 시스템(dynamic system)에 사용되는 모델과 특징에 대해 소개하였다. 인지컴퓨팅 모델의 근간이 되는 데이터에 대한 고찰이 부족했다.

정성원, 권혁철[12]은 자연언어처리(natural language processing)를 위한 기계학습 단계에서 기존 언어학적 연구 및 인터넷 상의 대용량의 언어 자원을 활용한 기계학습 기법을 제안하였다. 어휘 데이터의 지속적인 갱신을 강조했지만 구체적인 관리 방법을 제시하지 않고 있다.

3.3 요구공학 관련 연구

요구사항을 이해관계자로부터 이끌어내기 위한 다양한 도출 기법이 존재한다. 전통적인 도출 기법으로는 질문지 기반의 설문조사, 인터뷰(interview), 유스케이스(use case) 등이 있으며, 조직 내에 보유하고 있는 다양한 문서를 분석하여 요구사항을 도출하는 방법도 있다.[13]

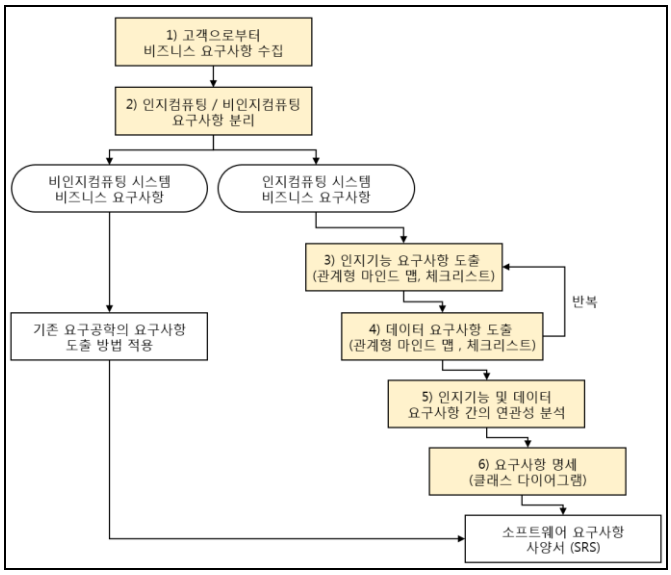
그룹 도출(group elicitation) 기법은 이해관계자들의 원활한 합의를 이끌어내는 것을 목적으로 하며 풍부한 의견을 종합하고자 할 때 유용하다. 대표적인 그룹 도출기법에는 브레인스토밍(brainstorming), 포커스 그룹(focus group), RAD/JAD 워크샵이 있다. [14]

Mylopoulos et al.[15]은 고객으로부터 전달받은 사용자 요구사항을 목표로 설정하고 이 목표를 작은 단위의 목표로 분해(decomposition)하며 구체화하는 방식을 제안한다. 사용자 요구사항을 단계적으로 구체화할 수 있다는 점에서 의의가 있지만, 도메인 특성에 맞는 특화된 방법을 제시하지 않는 한계가 있다.

Ohshiro et al.[16]은 다수의 이해관계자가 협력하여 고객의 요구사항으로부터 아이디어를 도출하고 도출된 아이디어를 그룹화 하고 연관관계를 정리하여 목표 그래프(goal graph)로 완성하는 방식을 제시한다. 이해관계자가 가지는 고유한 관점을 종합적으로 고려한 아이디어를 도출할 수 있다는 점이 장점이다. 하지만, 이 연구 역시 도메인 특성에 대한 고려가 없다.

4. 인공지능 소프트웨어 개발을 위한 요구공학 프레임워크

4.1 요구공학 프레임워크 개요



(그림 3) 인지컴퓨팅 시스템 요구공학 프로세스

그림 3은 본 연구에서 제안하는 인지컴퓨팅 시스템의 요구사항을 도출하고 명세하는 절차를 포함하는 전체적인 요구공학 프로세스를 보여준다. 요구공학 프로세스의 각 단계는 4.2 요구사항 도출 프로세스, 4.3 요구사항 도출 체크리스트 그리고 4.4 요구사항 명세 프로세스에서 다룰 것이다.

4.2 요구사항 도출 프로세스

1) 고객으로부터 비즈니스 요구사항 수집

인터뷰, 워크샵 등을 통해 고객으로부터 비즈니스 요구사항을 수집한다. 수집된 비즈니스 요구사항은 고객의 비즈니스 니즈(business needs)를 만족시키기 위한 목표로서 시스템 개발의 근거가 된다. 비즈니스 요구사항은 시스템 수준에서 아직 구체화 되지 않은 단계이므로 완전성(completeness)과 정형성(formality)이 낮다.

2) 비즈니스 요구사항을 인지컴퓨팅 요구사항과 비인지컴퓨팅 요구사항으로 분리

비인지컴퓨팅 요구사항, 즉 인지컴퓨팅 기능과 관련이 없는 요구사항에 대해서는 기존 요구공학 방법을 사용한다. 인지컴퓨팅 관련 요구사항은 본 연구에서 제시하는 요구공학 프레임워크를 사용한다.

요구공학 방법을 분리하여 적용하는 이유는 인지기능과 데이터가 핵심이 되는 인지컴퓨팅 소프트웨어는 전통적인 소프트웨어와는 다른 독특한 특징을 가지고 있기 때문이다.

3) 인지컴퓨팅 시스템의 인지기능 요구사항 도출

요구사항의 완전성과 정형성을 높이기 위해서 다소 추상적인 비즈니스 요구사항을 시스템 수준의 요구사항으로 구체화 하여야 한다. 요구사항을 구체화 할 때는 관계형 마인드 맵과 체크리스트를 활용한다. 관계형 마인드 맵은 요구사항을 창의적으로 구체화할 수 있는 장점이 있으며 체크리스트는 도출과정에서 고려해야 하는 사항을 체계적으로 점검할 수 있는 장점이 있다.

요구사항은 크게 인지기능 요구사항과 데이터 요구사항으로 구분하며 인지기능 요구사항을 먼저 구체화한다. '인지기능 요구사항 도출 체크리스트'는 4.3장에서 확인할 수 있다.

4) 데이터 요구사항 도출

인지기능 요구사항을 기반으로 데이터 요구사항을

구체화 하는 단계이다. 데이터의 생애주기적 관점에서 통합적으로 데이터 요구사항을 고려해야 하며 4.3장에서 제시하는 '데이터 요구사항 도출 체크리스트'를 활용한다.

인지기능과 데이터 요구사항을 서로 연관성이 깊으므로 인지기능 요구사항으로부터 데이터 요구사항을 발견할 수 있으며 그 반대인 데이터 요구사항을 통해 인지기능 요구사항을 발견할 수 있다. 따라서, 두 가지 요구사항은 반복적이며 점진적으로 구체화하여야 한다.

5) 인지기능 및 데이터 요구사항 간의 연관성 분석

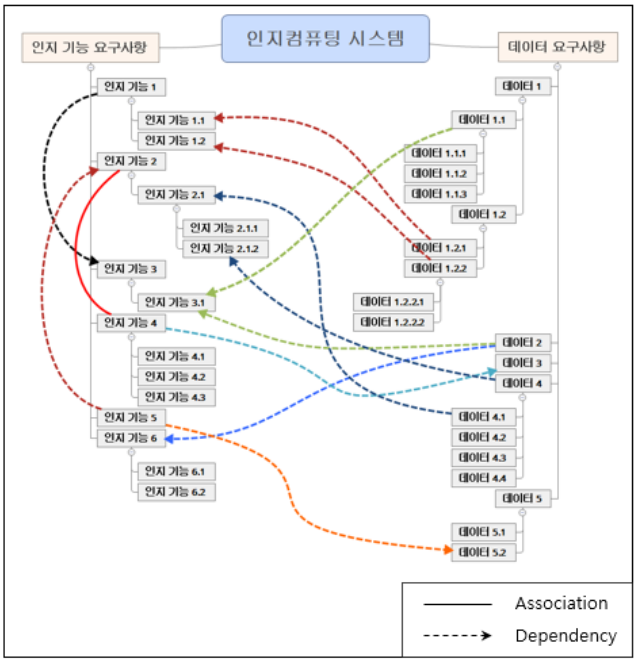
도출된 인지기능 요구사항과 데이터 요구사항을 분석하면 요구사항간의 연관, 의존, 모순, 합성 등과 같은 다양한 관계를 발견할 수 있다. 본 연구에서 제안한 관계형 마인드 맵을 활용하면 이와 같은 다양한 관계적 속성을 충실하게 표현할 수 있는 장점이 있다.

· 연관 관계(association): 두 개의 요구사항 사이에 연관관계가 있다면 두 요구사항이 서로 연관되어 있다는 의미이다.

· 의존 관계(dependency): 한 요구사항이 다른 요구사항을 참조하고 있으면 요구사항 사이에 의존 관계가 있다고 정의할 수 있다.

· 모순 관계(contradiction): 두 개의 요구사항 사이에 모순 관계가 있다면 둘 중 한 개의 요구사항을 제거함으로써 충돌을 해결할 수 있다.

· 합성 관계(composition): 복수 개의 요구사항이 한 개의 요구사항 안에 포함된다면 분할(decomposition) 수준이 다른 요구사항이 공존하지 않도록 한 쪽을 제거하거나 공존을 허용하는 방식 중에 선택할 수 있다.



(그림 4) 도출된 관계형 마인드 맵의 예

그림 4는 관계형 마인드 맵을 통해 인지컴퓨팅 시스템의 요구사항을 도출한 예이다.

4.3 요구사항 도출 체크리스트

1) 인지기능 체크리스트

본 연구에서는 Ferrucci[6]를 참고하여 인지기능 체크리스트를 표 2와 같이 도출하였다. 체크리스트를 이용하면 인지기능 요구사항을 도출할 때 필요한 기능을 누락 없이 작성하는데 도움을 받을 수 있다.

(표 2) 인지기능 요구사항 도출 체크리스트

<문제 분석>
A01. 음성인식(speech-to-text) 기능이 필요한가?
A02. 이미지 인식 기능이 필요한가?
A03. 텍스트의 의미를 이해하기 위한 자연어분석 기능이 필요한가?
A04. 실시간 처리기술(streaming processing)이 필요한가?
A05. 국가 언어간 번역 기능이 필요한가?
A06. 지원해야 하는 국가 언어는 어떤 것이 있는가?
A07. 질의는 어떻게 분류될 수 있는가?
A08. 질의로부터 구속 조건, 정의 부분, 제약 조건, 하위 요소, 예시를 어떻게 식별할 것인가?
A09. 질의 구문 분석 시스템은 어떤 알고리즘을 사용할 것인가?
A10. 질의 분석에는 어떠한 분석 시스템을 사용할 것인가?
A11. 질의 분석 중에 분석 시스템은 무엇을 이해하려고 시도하는가?
A12. 질의 분석의 결과가 어떠한 가중치로 의미를 도출하는가?
A13. 질의 분석의 결과적인 의미는 어떠한 데이터를 참조해야 하는가?
A14. 상황해석 시에 어떤 심층 분석 기법을 사용할 것인가?
A15. 상황해석 시에 어떤 통계 분류 기법을 사용할 것인가?
A16. 상황해석 시에도 운영 가설 설정이 필요한가?
A17. 상황해석 시에 어떤 알고리즘을 사용할 것인가?
<가설 설정 및 해답 후보 추론/예측>
B01. 가설 설정은 몇 단계로 이루어 지는가? (예, 왓슨의 경우 기본 검색과, 후보군 생성의 단계를 거침)
B02. 해답 후보는 곧 가설 중 하나에 의해 도출될 것이다. 해답 후보는 총 몇 개까지 도출할 것인가?
B03. 해답 후보는 최소한 얼마 이상의 신뢰도(정확도)를 필요로 하는가?
B04. 해답 후보를 도출하기 위한 데이터 소스에는 어떤 것들이 필요한가?
B05. 해답 후보의 신뢰도를 판단하기 위한 데이터 소스에는

어떤 것들이 필요한가? B06. 가설 설정을 위한 데이터 소스는 라벨을 가지고 있는가? B07. 가설 설정에는 어떤 기법을 활용하는가? B08. 예측/추론 알고리즘은 어떤 것을 사용할 것인가? B09. 예측/추론 알고리즘의 성능을 평가할 필요가 있는가?
<해답 후보 순위 선정> C01. 해답 후보의 점수는 어떻게 채점할 것인가? C02. 해답 후보의 점수 채점에 사용되는 데이터 소스에는 어떤 것들이 있는가? C03. 순위 알고리즘은 어떤 것을 사용할 것인가?
<해답 선정 및 답변> D01. 여러 의미에서 도출된 답변들을 병합할 필요가 있는가? D02. 있다면 답변 병합에 어떤 알고리즘을 활용할 것인가? D03. 병합된 답변들의 신뢰도와 순위는 어떤 알고리즘으로 측정할 것인가? D04. 도출된 답변은 어떤 형태로 출력할 것인가?

2) 데이터 관리 체크리스트

데이터 기반의 인지컴퓨팅 시스템을 개발하기 위해서는 데이터의 수집, 가공, 저장, 판단, 활용의 각 단계에서 발생할 수 있는 다양한 문제점을 해결해야 한다.[17] 본 연구에서는 DAMA-DMBOK[7]의 데이터 관리 체계를 참고하여 데이터 요구사항 체크리스트를 표 3과 같이 작성하였다.

(표 3) 데이터 요구사항 도출 체크리스트

<데이터 수집> E01. 필요한 데이터를 어떻게 확보할 것인가? E02. 수집 기준의 타당성이 확보되었는가? E03. 신뢰도가 낮은 데이터를 제거할 방법이 있는가? E04. 충분한 양의 데이터를 확보하였는가? E05. 최신의 데이터인가? E06. 중복 데이터를 어떻게 걸러낼 것인가?
<데이터 저장 및 가공> F01. 데이터를 저장할 매체는 무엇인가? F02. 비정형 데이터의 경우 어떤 전처리 과정이 필요한가? F03. 데이터의 형식은 무엇인가? 누락 데이터가 있는가? 있다면 어떻게 채울 것인가? F04. 이상치(Outlier)를 어떻게 정의하고 어떻게 처리할 것인가? F05. 데이터를 통합하기 위해 표현 단위를 일치시킬 필요가 있는가? F06. 데이터의 용량이 증가해 저장공간이 부족할 때 확장할 수 있는가? F07. 데이터가 보유한 정보를 조합해 새로운 정보를 생성할 수 있는가?
<데이터 보안> G01. 개인정보를 비식별화 하였는가? G02. 취득한 개인정보를 암호화할 필요가 있는가?
<데이터 분석> H01. 복잡한 정보를 한눈에 이해할 수 있도록 데이터를 시각화 해서 표현할 필요가 있는가? H02. 데이터분석을 통해 숨은 의미를 발견하였는가? H03. 추이 분석을 통해 데이터의 일관된 방향성을 확인할 수 있는가? H04. 상관 분석을 통해 두 개의 변수간의 상관성을 확인할 수 있는가?
<예측 분석 및 모델링> I01. 어떤 예측 모델을 사용할 것인가? I02. 트레이닝 수행 시 Training set 과 Validation set을 어떻게 분리할 것인가? I03. 데이터 타입에 적합한 모델이 필요한가?

4.4 요구사항 명세 프로세스

관계형 마인드 맵으로 작성한 요구사항은 이해관계자가 쉽게 이해할 수 있는 형태로 표현할 필요가 있다. 따라서, 본 연구에서는 소프트웨어 개발에서 자주 활용하는 UML의 클래스 다이어그램 형식으로 요구사항을 명세하는 방법을 제안한다.

클래스는 메소드(method)와 속성(attribute)을 가지고 있다. 도출한 요구사항 중 관련성이 깊은 인지기능 요구사항을 동일 클래스 내에 메소드로 정의한다. 또한 인지기능 요구사항에 의존 관계를 가지는 데이터 요구사항을 해당 클래스의 속성으로 정의하면 연관성이 높은 요구사항을 하나의 클래스에 표현할 수 있다. 하나의 클래스는 다수의 요구사항이 합쳐진 요구사항 그룹을 의미한다. 요구사항 그룹인 클래스는 다른 클래스와의 관계를 형성할 수 있으며 연관(association), 의존(dependency), 합성(composition), 집합(aggregation) 등의 속성으로 관계를 표현할 수 있다.

표 4는 OOP의 개념을 본 연구에서 제안한 요구공학 프레임워크의 개념과 매핑하여 표현한 것이다.

(표 4) OOP 개념과 요구공학 프레임워크 개념 매핑

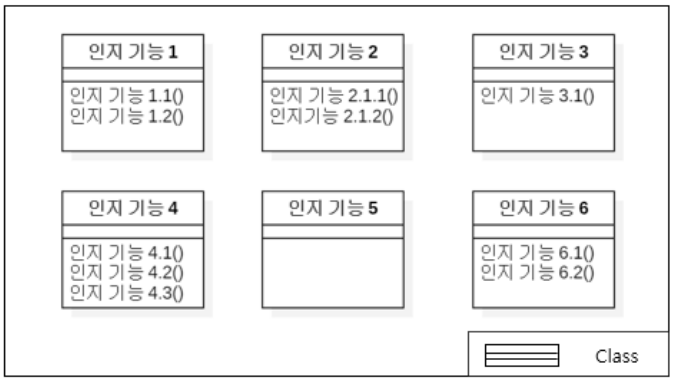
OOP 개념	요구공학 프레임워크 개념
클래스	요구사항 그룹
메소드	인지기능 요구사항
속성	데이터 요구사항
클래스간의 관계	요구사항 그룹간의 관계

1) 인지기능 요구사항 변환

관계형 마인드 맵을 통해 도출된 인지 기능 요구사항을 클래스 다이어그램으로 변환하는 단계이다.

상위수준의 인지기능 요구사항은 클래스로 정의하고 하위수준의 인지기능 요구사항은 클래스의 메소드로 정의한다.

그림 5는 그림 4에서 관계형 마인드 맵으로 도출된 6개의 인지기능 요구사항을 6개의 클래스로 변환하는 것을 나타낸다.

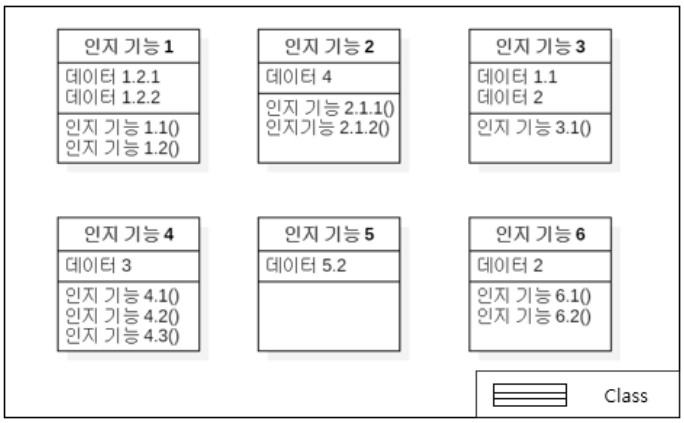


(그림 5) 인지기능 요구사항 명세

2) 데이터 요구사항 변환

관계형 마인드 맵으로 도출된 데이터 요구사항을 클래스 다이어그램으로 표현하는 단계이다. 관계형 마인드 맵에서 정의된 의존(dependency) 관계에 따라 데이터 요구사항을 클래스의 속성으로 정의한다.

그림 6은 그림 4에서 관계형 마인드 맵으로 도출된 데이터 요구사항을 6개의 클래스의 속성으로 할당하는 것을 나타낸다.

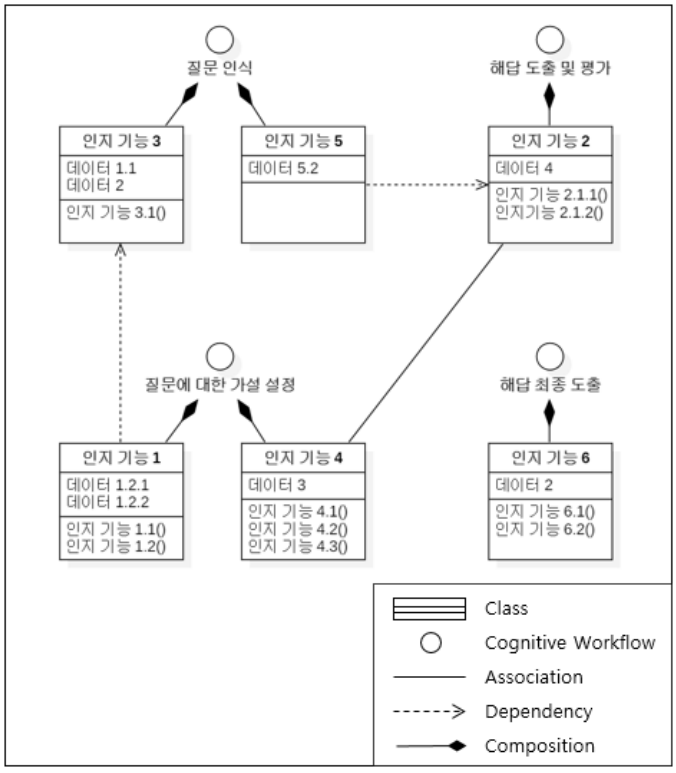


(그림 6) 데이터 요구사항 명세

3) 요구사항 그룹 간의 관계 설정

클래스 다이어그램은 클래스 간의 관계를 다양한 속성으로 표현할 수 있는 특징이 있다. 클래스는 다수의 요구사항이 결합된 그룹이며 그룹간의 관계를 다양한 형식으로 표현할 수 있다.

그림 7은 그림 4에서 도출된 인지기능 요구사항 간의 연관(association)과 의존(dependency) 관계를 클래스 다이어그램상에 표시한 것이다. 또한 인지기능의 공통 패턴에 맞게 인지기능 요구사항을 합성(composition) 관계로 나타낸 것이다.



(그림 7) 요구사항 그룹 간의 관계속성 명세

5. 사례 연구

“쇼핑몰 무인상담 시스템”에 해당 요구공학 프레임워크를 적용하여 사례 연구를 진행해 보았다.

1) 다양한 이해관계자로부터 비즈니스 요구사항 수집

콜센터로 전화를 걸거나 메신저로 문의하면 인공지능 로봇이 대답하는 무인상담 서비스를 하고 싶다. 고객은 음성이나 텍스트로 질의 할 수 있다. 국내 고객을 대상으로 하는 서비스이므로 한국어만을 지원한다.

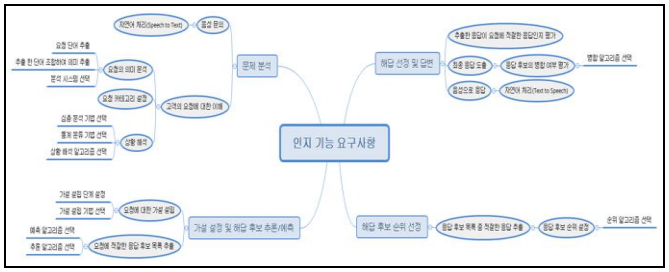
2) 수집된 요구사항을 인지컴퓨팅 요구사항과 비 인지컴퓨팅 요구사항으로 분리

- 비 인지컴퓨팅 요구사항
  - 메신저 / 전화 수발신 기능
- 인지컴퓨팅 요구사항
  - 사람이 대화하는 것을 기계가 이해하고 처리
  - 데이터를 학습해 사용자가 원하는 결과를 제시
  - 한국어를 이해하고 구사할 수 있어야 함

– 사용자 음성을 이해하고 결과를 음성으로 합성

3) 인지컴퓨팅 시스템의 인지기능 요구사항 도출

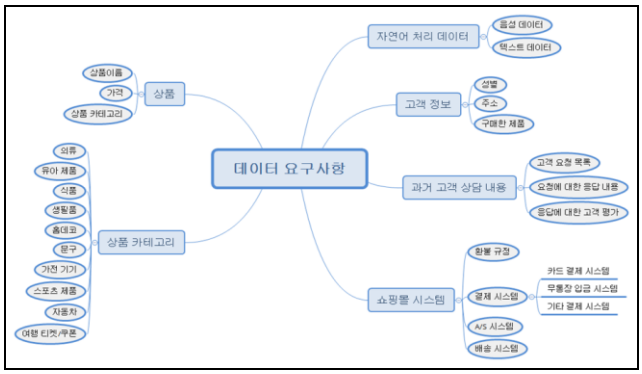
쇼핑몰 무인 응답 시스템의 인지컴퓨터 요구사항을 인지기능 요구사항과 데이터 요구사항으로 분리해 도출한다. 그림 8는 쇼핑몰 무인상담 시스템의 인지기능 요구사항을 도출한 관계형 마인드 맵이다. 4.3장에서 제안한 인지기능 요구사항 도출 체크리스트를 활용하여 인지기능 요구사항을 도출하였다.



(그림 8) 쇼핑몰 무인상담 시스템 인지기능 요구사항

4) 데이터 요구사항 도출

그림 9은 쇼핑몰 무인상담 시스템의 데이터 요구사항을 도출한 관계형 마인드 맵이다. 4.3장에서 제안한 데이터 요구사항 도출 체크리스트를 활용하여 데이터 요구사항을 도출하였다.

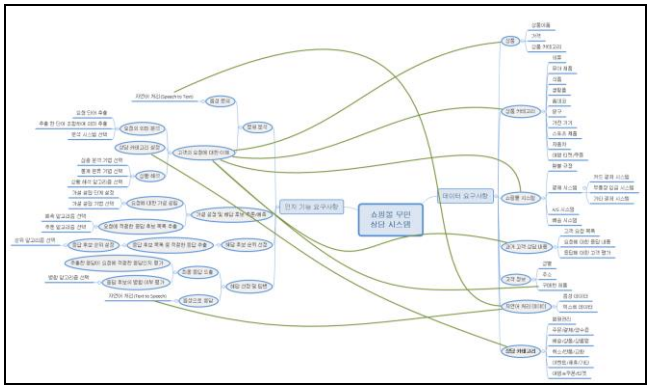


(그림 9) 쇼핑몰 무인상담 시스템 데이터 요구사항

5) 데이터 및 기능 요구사항 간의 상호 의존성 분석

인지기능 요구사항과 데이터 요구사항이 도출되면 서로 간의 상호 의존성을 분석하여 관계형 마인드 맵에 표시한다.

그림 10에서는 인지기능 요구사항과 데이터 요구사항 간의 상호 의존성을 분석하여 화살표로 표시하였다.

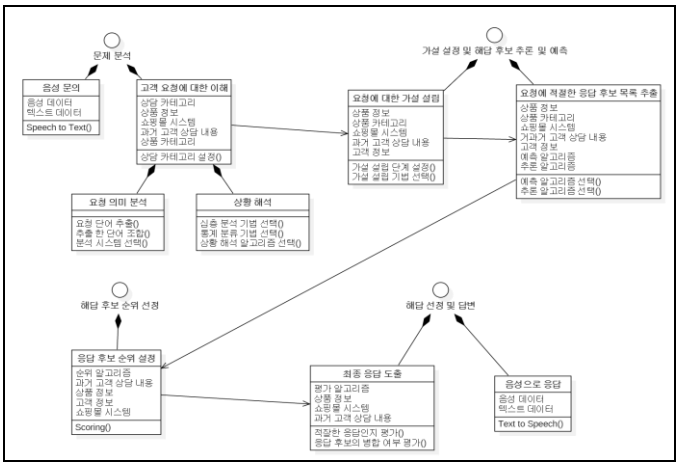


(그림 10) 쇼핑몰 무인상담 시스템 요구사항 도출

6) 인지컴퓨팅 시스템의 요구사항 명세

인지컴퓨팅 시스템 요구사항 도출이 완료되면 클래스 다이어그램을 이용하여 요구사항을 명세한다.

그림 11는 도출된 요구사항을 클래스와 화살표로 표시하여 쇼핑몰 무인상담 시스템의 요구사항을 명세하였다.



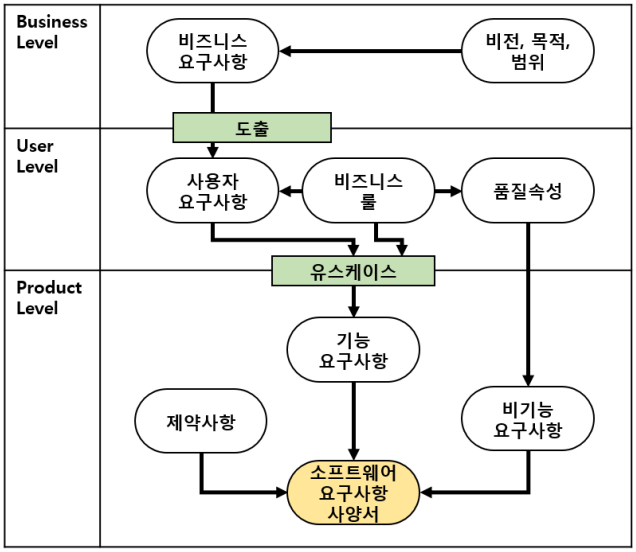
(그림 11) 쇼핑몰 무인상담 시스템 요구사항 명세

6. 평 가

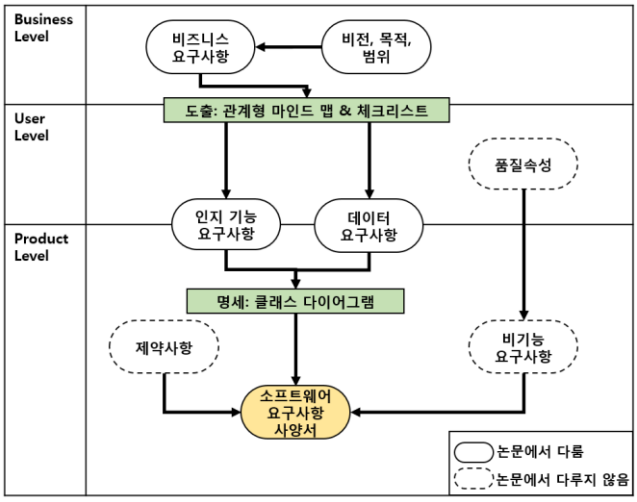
Wiegers[18]는 요구사항이라는 용어를 비즈니스, 사용자, 제품 관점의 요구사항으로 구분하여 정의하였고 각 요구사항이 구체화 되는 흐름을 그림 12과 같이 표현하였다. Wiegers의 프레임워크는 사용자 요구사항을 유스케이스(use case)의 형태로 기술하는데 유스케이스를 사용하면 요구사항을 추상적으로 포착하여 설계와의 갭이 크다는 단점이 있다.

그림 13은 본 연구에서 제시한 요구공학 프레임워크를 표현하고 있다. 이 프레임워크는 유스케이스 대신 관계형 마인드 맵과 체크리스트를 활용하여 시스템 수준의 요구사항을 도출하므로 유스케이스를 사용하는 방법보다 더 구체적이라는 장점이 있다. 또한 요구사항을 클래스 다이어그램으로

군집화 및 관계화하여 체계적으로 명세하기 때문에 이후 단계인 설계단계에서 활용도가 높다.



(그림 12) 기존 소프트웨어 요구공학 프레임워크



(그림 13) 본 연구에서 제시한 요구공학 프레임워크

7. 결론 및 향후 연구방향

인지컴퓨팅은 사람의 뇌가 가진 인지능력을 모방하여 만든 정보처리기술이다. 인지컴퓨팅 시스템은 특정 문제를 해결하기 위해 인공적인 인지능력을 활용하는 시스템으로 다른 도메인(domain)의 시스템과는 구별되는 특징을 가지고 있다. 첫 번째 특징은 사람의 인지기능과 유사하게 구현하는 인지기능적 특징이며, 두 번째는 인지기능을 위한 데이터의 관리적 특징이다. 현대의 소프트웨어는 규모가 방대하고 복잡도가 높아서 개발하고자 하는 시스템의 도메인에 맞는 특별한 개발 기법을 적용해야만 하는 상황에 이르렀다. 기존 요구공학 방법론을 인지컴퓨팅 시스템의 요구사항 도출 및 명세에 일괄적으로 적용하기 어렵다. 이러한 문제점을 해결하기 위해 인지컴퓨팅 시스템에 적합한 요구공학 프레임워크를 본 연구에서 제안하였다.

인지컴퓨팅 시스템을 위한 요구공학 프레임워크는 비즈니스 요구사항으로부터 시스템 수준의 요구사항을 구체화하기 위해 관계형 마인드 맵과 체크리스트를 활용하는 방법을 활용하였으며 인지기능 요구사항과 데이터 요구사항을 반복적으로 도출하면서 상호보완적으로 완성하는 방법을 제안하였다. 제안된 체크리스트를 활용하면 인지기능과 데이터의 요구사항을 보다 종합적이고 체계적으로 도출할 수 있는 장점이 있다.

시스템 수준의 요구사항이 기록된 관계형 마인드 맵은 클래스 다이어그램의 형태로 명세화하는 과정을 거치게 된다. 인지기능과 데이터 간에는 다양한 연관관계가 존재하며 부여된 관계의 특성에 따라 클래스 다이어그램의 속성(attribute)과 메소드(method)를 정의하게 된다. 이 과정을 통해 도출된 명세서는 이후에 설계 단계에서 활용할 수 있는 수준으로 도출되게 함을 목표로 하였다.

본 연구에서 제시한 요구공학 프레임워크를 개선하기 위해서 다음의 세가지 향후 연구방향을 제시한다.

1) 관계형 마인드 맵을 소개하면서 요소들 간의 관계를 나타낼 수 있는 관계속성을 정의하였는데 다양함이 부족하여 표현에 한계가 존재한다. 현실의 문제를 충실하게 표현할 수 있는 다양한 관계속성에 대한 추가적인 연구가 필요하다.
2) 관계형 마인드 맵 기법을 지원하는 체크리스트를 제시하였으나, 기술이 빠르고 복잡하게 변화하고 있으므로 본 연구에 제시된 체크리스트가 인지컴퓨팅의 각 특징들이 가진 모든 범주를 포함하는데는 분명한 한계가 있다. 관계형 마인드 맵에서 생각을 확장시키기 위한 도구에 대한 연구도 필요하다.
3) 요구사항을 도출할 때 비기능 요구사항에 대한 고려가 이루어지지 않았다. 비기능 요구사항은 설계와 구현 단계에 큰 영향을 미치는 요소로서 인지기능과 데이터 요구사항을 도출할 때 함께 고려되면 더 효율적일 것이다.

본 연구에서 제시한 요구공학 프레임워크를 사용해 인지컴퓨팅 시스템을 구축하면 효과적으로 요구사항을 도출할 수 있어 프로젝트 성공률을 높이는데 도움을 줄 것이며 이해관계자와의 의사소통을 원활히 하는데 기여할 것이다.

8. 참고문헌

[1] AlphaGo의 인공지능 알고리즘 분석, 소프트웨어 정책연구소, SPRi Issue Report (https://spri.kr/post/)

[2] 인공지능 업계 동향 및 인식조사 결과, 미래창조과학부

정보통신정보화 및 정책지원 사업(ICT통계조사 및 동향분석)의 연구결과, 정보통신기술진흥센터

[3] 이성호. "[신기술과 산업 지형의 변화 ①] 인지컴퓨팅." 과학기술정책 26.5 (2016): 16-23.

[4] 장병탁, 여무송. "Cognitive Computing I: Multisensory Perceptual Intelligence." 정보과학회지 30.1 (2012): 75-87.

[5] IBM 딥블루, IBM 왓슨, Wikipedia (https://ko.wikipedia.org/wiki/)

[6] Ferrucci, David, et al. "Building Watson: An overview of the DeepQA project." AI magazine 31.3 (2010): 59-79.

[7] Mosley, Mark, et al. DAMA guide to the data management body of knowledge. Technics Publications, 2010.

[8] Bourque, Pierre, et al. "The guide to the software engineering body of knowledge." IEEE software 16.6 (1999): 35.

[9] LaValle, Steve, et al. "Big data, analytics and the path from insights to value." MIT sloan management review 52.2 (2011): 21.

[10] 장병탁, 이동훈. "Cognitive Computing II: Machine Vision-Language Learning." 정보과학회지, 30.1 (2012.1): 88-100.

[11] 장병탁, 김현수. "Cognitive Computing III: Deep Dynamic Prediction." 정보과학회지, 30.1 (2012.1): 101-111.

[12] 정성원, 권혁철. "자연언어처리를 위한 기계학습." 정보과학회지, 25.3 (2007.3): 57-63.

[13] Nuseibeh, Bashar, and Steve Easterbrook. "Requirements engineering: a roadmap." Proceedings of the Conference on the Future of Software Engineering. ACM, 2000.

[14] Maiden, Neil AM, and Gordon Rugg. "ACRE: selecting methods for requirements acquisition." Software Engineering Journal 11.3 (1996): 183-192.

[15] Mylopoulos, John, Lawrence Chung, and Eric Yu. "From object-oriented to goal-oriented requirements analysis." Communications of the ACM 42.1 (1999): 31-37.

[16] Ohshiro, Kazuya, Kenji Watahiki, and Motoshi Saeki. "Integrating an idea generation method into a goal-oriented analysis method for requirements elicitation." 12th Asia-Pacific Software Engineering Conference (APSEC'05). IEEE, 2005.

[17] Agrawal, Divyakant, et al. "Challenges and Opportunities with big data 2011-1." (2011).

[18] Wiegers, Karl E. "Karl Wiegers describes 10 requirements traps to avoid." Software Testing & Quality Engineering 2.1 (2000).

안전 요구사항 추출을 위한 안전성 분석 지침 개발 및 적용 사례

정대희 ○, 권기현
경기대학교 컴퓨터과학과
jdhpen@kgu.ac.kr, khkwon@kgu.ac.kr

Developing Guidelines of Safety Analysis for Eliciting Safety Requirements and Its Application

Daehui Jeong ○, Gihwon Kwon
Dept of Computer Science, Kyonggi University

요 약

본 논문에서는 소프트웨어 안전성 분석의 중요성과 안전성 분석을 소개하고, 철도 분야라는 도메인에서 소프트웨어의 안전 요구사항 추출을 위한 안전성 분석 지침을 보인다. 지침은 사례로 선정한 철도 건널목 시스템 및 제어 소프트웨어 개발에 적용해 예시를 보인다. 안전성 분석을 하기 위해서는 위험원을 식별하고, 식별된 위험원의 위험도를 분석해야 한다. 본 논문에서는 PHA와 SHA, FMEA로 안전성 분석을 수행하였다.

1. 서 론

시스템에서 소프트웨어가 차지하는 비중이 늘어나며 소프트웨어로 인한 사고가 증가함에 따라 소프트웨어의 안전이 중요해지고 안전성 활동을 요구하게 되었다.

안전성 분석은 시스템 공학과 같은 분야에서 일찍이 사용 되어 오던 안전성 활동으로 시스템이 갖추어야 할 안전성이 만족되었는지 확인한다[1]. 안전성 분석은 기존의 개발 생명주기에서 요구사항 분석 이전에 수행하는 것으로 위험원을 식별하고, 식별된 위험원의 위험도를 분석하여 안전 요구사항을 도출한다.

명확한 요구사항은 성공적인 소프트웨어 개발을 이끈다. 즉 명확하지 못한 요구사항은 소프트웨어 개발의 실패를 야기한다[2]. 본 논문에서는 안전성 분석 지침을 작성하여 안전 요구사항의 식별을 돕고자 한다. 다음은 널리 사용되는 안전성 분석 기법이다[3].

- 예비 위험원 분석(PHA)
- 시스템 위험원 분석(SHA)
- 고장 유형 및 영향 분석(FMEA)
- 결함 트리 분석(FTA)
- 위험원 및 운용성 분석(HAZOP)

안전성 분석에 사용되는 기법은 목표 시스템의 성격과 개발 단계에 따라 선택적으로 적용해야 한다. 본 논문에서 다루고자 하는 도메인은 철도 분야로 국제 철도 표준 IEC 62278[4]을 기반으로 안전성 분석 개발 지침을 보이고, 사례로 선정한 철도 건널목 시스템 및 제어 소프트웨어 개발에 적용한다. 여기서 철도 건널목 시스템은 실제 건널목 시스템을 모형 철도 및 아두이노로 재현한 것으로, 철도 분야에서 가장 높은 안전 무결성 등급(Safety Integrity Level, SIL) 4를 요구한다[5,6].

본 논문의 구성은 다음과 같다. 2장에서는 안전성 분석을 지침 하고, 3장에서는 적용 사례로 선정한 건널목 시스템에 지침한 안전성 분석을 적용한다. 그리고 4장에서는 결론 및 향후 연구 방향을 기술한다.

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음"(IITP-2016-R0992-16-1014)

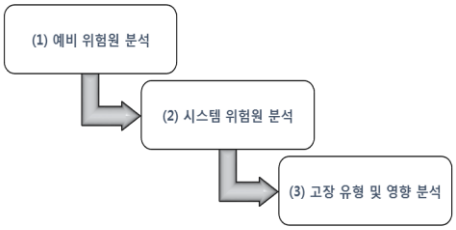
2. 안전성 분석

안전성을 만족시키는 소프트웨어를 개발하기 위해서는 안전성 분석이 요구된다. 여기서 안전성 분석이란 시스템이 갖추어야 할 안전성을 만족하는지 확인하는 활동을 의미한다[1]. 안전성 분석의 목적은 안전성을 확보하기 위하여 위험원을 식별하고, 식별된 위험원을 제거 또는 완화하는 것이다.

본 논문에서 다루고자 하는 도메인은 철도 분야로, IEC 62278을 기반으로 안전성 분석 지침을 <그림 1>과 같이 삼 단계로 제안한다. 각 단계마다의 설명은 다음과 같다.

2.1 예비 위험원 분석

예비 위험원 분석(Preliminary Hazard Analysis, PHA)



<그림 1> 안전성 분석 수행 절차

은 위험원과 원인, 영향 등을 식별할 수 있는 안전성 분석 기법으로 사용한다. 식별된 위험원을 발생시키는 원인을 제거해 제어하기 위한 목적으로 분석을 수행하며, 분석의 결과는 정성적으로 표현한다. PHA의 수행 범위는 시스템 전반으로, 시스템의 설계 정보를 바탕으로 시스템 개발 초기에 수행한다. 다음은 PHA의 수행 절차이다.

⑴ 작성 양식 결정
⑵ 기본 교육
⑶ 작성 양식 배포 및 작성
⑷ 작성 양식 취합 및 정리
⑸ 분석 활동 수행

PHA를 수행함에 있어서 우선되어야 할 것은 작성 양식을 결정하는 것으로, PHA를 수행하기 위해 가장 적합한 항목을 결정해야 한다. <표 1>은 결정되어야 할 항목을 나타낸다.

<표 1> PHA 결정 항목

결정 항목
1. 위험원 번호
2. 위험원
3. 위험원 설명
4. 원인
5. 영향
6. 초기 위험도(심각도, 발생 빈도, 허용 수준)
7. 저감 대책
8. 비고
9. 책임

작성 양식이 결정되면 시스템 설계 정보를 바탕으로 목표 시스템의 기능을 명세하고 예비 위험원을 식별해 작성 양식의 내용을 기입한다.

2.2 시스템 위험원 분석

시스템 위험원 분석(System Hazard Analysis, SHA)은 목표 시스템에 대한 보다 명확한 안전성 분석을 수행한다. 시스템을 구성하는 서브시스템과 인터페이스가 SHA의 수행 대상으로, PHA를 통해 식별한 목표 시스템의 위험원을 기반으로 서브시스템 및 인터페이스의 위험원과 원인, 저감 대책을 결정하기 위한 목적으로 수행한다. SHA의 수행 절차는 다음과 같다.

⑴ 서브시스템 및 인터페이스 기능 요구사항 분석
⑵ 서브시스템 및 인터페이스 기능 식별
⑶ 분석 활동 수행

기능 요구사항은 목표 시스템을 구성하는 서브시스템과 인터페이스가 요구하는 기능을 나타내는 것으로 PHA에서 명세한 기능을 이용한다. 기능 요구사항이 분석되면 기능을 세분화 해서 식별하고, 이로 인해 발생될 수 있는 위험원, 원인, 영향, 저감 대책을 분석한다.

2.3 고장 유형 및 영향 분석

고장 유형 및 영향 분석(Failure Mode and Effects Analysis, FMEA)은 시스템을 구성하는 컴포넌트의 잠재적인 고장 유형이 시스템에 미치는 영향을 분석하는 기법이다. 시스템의 특정 부분이 일으키는 고장 유형이 전체 시스템의 신뢰도나 안전성에 영향을 미칠 수 있는지 확인하는 방법으로, 원인으로부터 결과를 도출하는 상향식 접근 방법이다. FMEA의 수행 절차는 다음과 같다.

⑴ 컴포넌트 및 기능 식별
⑵ 고장 유형 및 원인, 영향 식별
⑶ 위험도 추정 및 저감대책 수립
⑷ 위험도 재 추정

컴포넌트 및 기능의 식별은 시스템을 구성하는 최하위 요소로 분류하고 그 기능을 식별하는 것이다. 고장 유형의 식별은 FMEA의 핵심 사항으로 이를 바탕으로 추적성을 확인하고 원인과 영향 식별 및 저감 대책을 수립하게 된다. 그 후에, 저감 대책 수립 전후의 위험도를 위험도 우선 순위(Risk Priority Number, RPN)으로 추정한다. RPN은 심각도, 발생 빈도, 허용 수준의 곱으로 계산되고, 심각도는 <표 2>, 발생 빈도는 <표 3>, 허용 수준은 <표 4>를 기준으로 한다.

<표 2> 심각도

심각도	등급
Catastrophic	A
Critical	B
Marginal	C
Insignificant	D

<표 3> 발생 빈도

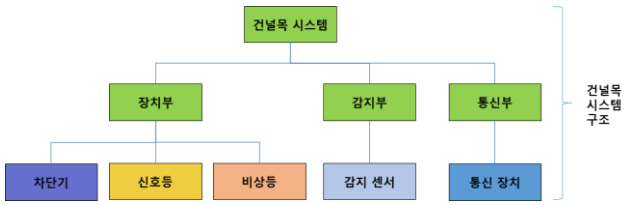
발생 빈도	등급
Frequent	1
Probable	2
Occasional	3
Remote	4
Improbable	5
Incredible	6

<표 4> 허용 수준

허용 수준		심각도			
		A	B	C	D
발생빈도	1	Intolerable	Intolerable	Intolerable	Undesirable
	2	Intolerable	Intolerable	Undesirable	Tolerable
	3	Intolerable	Undesirable	Undesirable	Tolerable
	4	Undesirable	Undesirable	Tolerable	Negligible
	5	Tolerable	Tolerable	Negligible	Negligible
	6	Negligible	Negligible	Negligible	Negligible

3. 안전성 분석 적용 사례

본 연구에서는 건널목 시스템의 제어 소프트웨어 개발을 위해서 안전성 분석을 수행하였다. 건널목 시스템은 실세계에서 SIL 4를 요구하기 때문에 안전성 분석을 철저히 수행해야 한다. 다음은 건널목 시스템의 설계 정보이고 이를 구조화 하면 <그림 2>와 같다.



<그림 2> 건널목 시스템의 구조도

건널목 시스템에 사용된 컴포넌트들은 다음과 같다.

- 차단기: 선로 진입을 차단한다.
- 신호등: 열차의 접근을 알려준다.
- 비상등: 비상상태를 알려준다.
- 감지 센서: 열차의 접근을 감지한다.
- 통신 장치: 중앙 서버와의 연결을 담당한다.

3.1 PHA 수행

2장에서 설명한 바와 같이 PHA를 수행하기 위해 먼저 기능을 명세 하였다. 명세한 기능은 이후 SHA, FMEA에서 활용한다. 기능이 명세 되면 예비 위험원 목록을 작성한다. 이 때, 기능 명세에서 명세한 각 세부 기능이 제대로 작동하지 않는 경우가 하나의 위험원이 된다. 위험원이 식별되면 원인과 영향, 저감 대책까지 기술한다. <표 5>는 PHA 수행 결과이다.

<표 5> PHA 수행 결과

예비 위험원	원인	영향	저감 대책
차단기 상승 실패	1. H/W 결함 2. S/W 결함	교통 혼란 야기	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
차단기 하강 실패	1. H/W 결함 2. S/W 결함	인명 피해	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
신호등 빨간등 점등 실패	1. H/W 결함 2. S/W 결함	인명 피해	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
신호등 빨간등 소등 실패	1. H/W 결함 2. S/W 결함	교통 혼란 야기	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
신호등 초록등 점등 실패	1. H/W 결함 2. S/W 결함	교통 혼란 야기	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
신호등 초록등 소등 실패	1. H/W 결함 2.S/W 결함	인명 피해	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
비상등 점등 실패	1. H/W 결함 2. S/W 결함	인명 피해	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
비상등 소등 실패	1. H/W 결함 2. S/W 결함	교통 혼란 야기	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
감지 센서 감지 실패	1. H/W 결함 2. S/W 결함	인명 피해	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험
통신 장치 연결 및 유지 실패	1. H/W 결함 2. S/W 결함	인명 피해	1. H/W 결함 허용 설계 2. S/W 건전성 확보 및 기능 시험

<표 6> SHA 수행 결과

시스템 위험원	원인	영향	저감 대책
장치부 제어의 수행 불가	1. 차단기의 결함 2. 신호등의 결함 3. 비상등의 결함 4. 차단기 제어 변수 오류 5. 신호등 제어 변수 오류 6. 비상등 제어 변수 오류	1. 인명 피해 2. 교통 혼란 야기	1. 장치 결함 여부 판별 2. 제어 변수 오류 여부 판별

3.2 SHA 수행

SHA는 건널목 시스템에 대한 PHA 결과를 보다 명확히 하기 위해 수행한다. 건널목 시스템의 서브시스템은 장치부, 감지부, 통신부의 3가지로, 각 서브시스템 및 인터페이스의 기능 요구사항과 요구되는 기능을 식별하고 그로 인해 발생할 수 있는 위험원, 원인, 영향, 저감 대책을 작성한다. <표 6>은 SHA 수행 결과의 일부이다.

3.3 FMEA 수행

FMEA는 컴포넌트의 잠재적인 고향 유형이 시스템에 미치는 영향을 분석하기 위해 수행한다. FMEA를 수행하기 위해 컴포넌트와 각 컴포넌트의 기능을 식별하였다. 기능이 식별되면 고장 유형과 원인, 영향을 식별하고 저감 대책을 수립한다. 그리고 저감 대책 수립 전후의 RPN을 추정한다. <표 7>은 FMEA의 수행 결과이다.

<표 7> FMEA 수행 결과

고장 유형	영향	원인	저감 대책 전 RPN			저감 대책	저감 대책 후 RPN		
			심각도	발생 빈도	허용 수준		심각도	발생 빈도	허용 수준
차단기 제어 오류	차단기 상승 불가능	1. 차단기의 결함 2. 차단기 제어 변수 오류	C	3	Undesirable	M1: 차단기 결함 여부 판별	C	5	Negligible
	차단기 하강 불가능		A	3	Intolerable	M2: 차단기 제어 변수 오류 여부 판별 M3: 비상 상태 돌입	A	5	Tolerable
신호등 제어 오류	빨간등 점등 및 소등 불가능	1. 신호등의 결함 2. 신호등 제어 변수 오류	A	3	Intolerable	M4: 신호등 결함 여부 판별	A	5	Tolerable
	초록등 점등 및 소등 불가능		A	3	Intolerable	M5: 신호등 제어 변수 오류 여부 판별 M6: 비상 상태 돌입	A	5	Tolerable
비상등 제어 오류	비상등 점등 및 소등 불가능	1. 비상등의 결함 2. 비상등 제어 변수 오류	A	3	Intolerable	M7: 비상등 결함 여부 판별 M8: 비상등 제어 변수 오류 여부 판별 M9: 비상 상태 돌입	A	5	Tolerable
감지 센서 제어 오류	감지 센서 거리 측정 불가능	1. 감지 센서의 결함 2. 감지 센서 제어 변수 오류	A	3	Intolerable	M10: 감지 센서 이중화 및 결함 여부 판별 M11: 감지 센서 제어 변수 오류 여부 판별 M12: 비상 상태 돌입	A	5	Tolerable
통신 장치 제어 오류	통신 장치 연결 및 유지 불가능	1. 통신 장치의 결함 2. 통신 장치 제어 변수 오류	A	3	Intolerable	M13: Heartbeat M14: 통신 장치 제어 변수 오류 여부 판별 M15: 비상 상태 돌입	A	5	Tolerable

4. 결론 및 향후 연구

본 연구에서는 철도 소프트웨어 개발에 적합한 안전성 분석 지침을 고안하였고, 건널목 제어 시스템에 이 분석 지침을 적용하였다. 그 결과, 아래와 같은 안전 요구사항을 도출하였다.

- SR1: 건널목 시스템은 장치 결함이 발생하면 비상등을 점등하고 중앙 제어부로 비상 상태를 송신해야 한다.
- SR2: 건널목 시스템은 감지 센서를 이중화 하여야 한다.
- SR3: 건널목 시스템은 감지 센서 비정상이 감지되면 비상등을 점등하고 차단기를 하강해야 한다.
- SR4: 건널목 시스템은 통신 장치 비정상이 감지되면 비상등을 점등하고 차단기를 하강해야 한다.

안전 요구사항은 <표 7>의 FMEA 수행 결과 중 저감 대책에서 도출한다. 각 안전 요구사항과 관련된 저감 대책을 나타내 보면, SR1은 M1~M9, SR2는 M10, SR3은 M11~M12, SR4는 M13~M15를 나타낸다.

도출된 안전 요구사항은 이후 설계, 구현, 테스트에 반영하였다. 안전성 분석을 거치지 않은 기존의 개발 시에는 감지 센서에 결함이 발생하면 차단기가 제어되지 않는 문제가 발생하였던 반면, 안전성 분석을 거쳐 안전 요구사항을 반영해 해당 문제를 제어할 수 있게 되었다. 이처럼 안전성 분석이 적용된 소프트웨어의 개발은 개발자의 경험과 판단에 의존하였던 기존과는 달리 체계적인 분석을 통해 소프트웨어가 가질 수 있는 위험원을 분석하여 제어할 수 있다.

본 연구가 기여하는 바는 두 가지로 본다. 첫째, 목표 시스템 및 제어 소프트웨어 개발을 위한 안전성 분석을 수행하는 방법을 알려준다. 이는 철도라는 분야에서 소프트웨어를 개발하기 위한 능력을 증강시켜준다. 둘째, 안전성 분석의 이해를 도와준다. 이들에 대한 이해는 기존에 제시되었던 방법의 문제점을 파악하고 더 나아가 새로운 것을 제안하도록 도와줄 것이다.

향후 연구 방향은 본 연구의 경험을 바탕으로 철도 분야뿐 만이 아니라 소프트웨어 전반에서 사용할 수 있도록 소프트웨어의 개발 주기를 따라 명확한 추적성을 가지는 안전성 분석 기법을 연구하고자 한다.

참고문헌

[1] 이장수, 이동아, “소프트웨어 기반의 안전 필수 시스템을 위한 안전성 분석 기법”, 정보과학회지 제33권 제7호, pp. 41-46, 2015

[2] 나관식, “요구사항 불확실성과 표준화가 소프트웨어 프로젝트 성과에 미치는 영향: 한국과 미국의 비교연구”, Journal of Information Technology Application & Management, 제11권 제2호, pp. 15-27, 2004

[3] Clifton A. Ericson, II, “Hazard Analysis Techniques for System Safety”, Wiley-Interscience, 2005

[4] IEC 62278:2002, Railway Applications - Specification and Demonstration of Reliabliity, Availablility, Maintainability and Safety (RAMS)

[5] IEC 62425:2007, Railway Applications - Communication, Signaling and Processing Systems- Safety Related Electronic Systems for Signaling

[6] Yellow Book, Railtrack on Behalf of UK Rail Industry, 2000

소프트웨어 요구사항 진화에서의 추적성 관리의 어려움 조사

박종열 ○, 류승희, 정세린, 이선아
항공우주 및 소프트웨어 공학전공, 경상대학교
{bjng237, tmdgml5400, rin0608 ,saleese}@gnu.ac.kr

Investigation on the difficulties of managing traceability in the evolution of software requirements

Jongyeol Park○, Seunghui Ryu, Serin Jeong, Seonah Lee
Department of Aerospace and Software Engineering, Gyeongsang National University

요 약

소프트웨어 요구사항은 소프트웨어 시스템에 대해서 사용자가 기대하는 기능, 품질 및 조건으로서, 초기에 완전하게 도출하기 어렵고, 개발 및 유지보수 중에 지속적으로 변화한다. 요구사항의 진화를 돕기 위한 주요 방법이 추적성 수립 및 관리이다. 그러나 기존의 실증적 연구는 추적성 관리가 실질적인 활용에 이르지 못한다고 보고한다. 본 연구에서는 추적성 관리를 포함한 소프트웨어 요구사항의 진화를 위한 자동화 연구를 문헌 조사한다. 또한 오픈 소스 프로젝트에서의 요구사항 변경이 발생할 때 소프트웨어 시스템에 어떤 변경이 발생하는지를 실증 조사한다. 문헌 조사와 실증 조사를 통해, 진화적 관점에서의 요구 사항 관리의 어려움의 원인을 분석하고, 도움을 줄 수 있는 자동화 연구 방향에 대해서 논의한다.

1. 서 론

소프트웨어 요구사항은 소프트웨어 시스템에 대해서 사용자가 기대하는 기능, 품질 및 조건이다. 소프트웨어 요구사항은 개발 단계 초기에 완전하게 도출하기 어렵고, 개발 및 유지보수 동안에 지속적으로 변화한다. 요구공학 프로세스는 소프트웨어 개발에서의 체계적인 요구사항의 개발을 위하여, 수집, 분석, 명세, 검증 및 관리의 단계를 제시한다. 이 중 관리의 단계에서는 주로 요구사항의 시스템 반영을 표현할 수 있는 요구사항 추적성을 수립하고 관리한다. 그러나, 지속적으로 변화하는 요구 사항을 위한 추적성 관리는 쉽지 않으며, 요구사항을 시스템에 반영하는 작업은 사람의 인지적인 능력과 결정에 대부분 의존하고 있다.

최근의 실증적 연구들은 추적성에 대한 상반된 결과를 내놓았다. 첫째는 추적성 관리가 실질적인 활용에 이르지 못한다는 보고이다. Rempel et al.은 실제 프로젝트에서 요구사항 추적성에 대한 표준 및 지침이 50%이상 적용되지 않음을 실증적으로 보였다 [1]. Cleland-Huang et al.은 추적성을 중요시하는 안전 중심의 소프트웨어 개발에서조차 바로 인증 전에 추적성을 수립하고 있음을 보고하였다 [2]. 반면 추적성의 유용성을 증명한 연구들이 있다. Asuncion et al.은 추적성 데이터를 웹을 통해 접근 가능하게 만들었을 때, 추적성 관련 작업이 2배 빨라졌음을 보고했다 [3]. Mader et al.은 실험환경에서 추적성을 사용할 때 24%의 효율성 향상과 50%의 정확도 향상을 보고했다. 이러한 연구들은 추적성의 활용이 개발자 생산성 향상에 도움을 줄 수 있으나, 실질적인 활용에 이르지 못하고 있음을 알려준다 [4].

본 논문은 상기 상반된 결과가 현 추적성 관리 방법이 소프트웨어 진화에서 발생하는 요구 사항의 변경에 대응하지 못하기 때문이라고 본다. 따라서 요구사항의 진화를 이해하고 추적성 관리를 포함한 자동화 연구의 수준을 분석하여 어떤 연구가 향후 필요한지 이해하고자 한다. 본 논문은 다음 두 가지 질문의 답을 찾기를 시도한다.

1) 요구사항의 변경에 따른 체계적인 소프트웨어 진화가 왜 어려운가?

2) 요구사항의 진화를 위한 기존 연구의 한계점과 향후 연구 기대 방향은 무엇인가?

본 논문은 이 두가지 질문에 답하기 위해서 문헌 조사와 실증 조사를 수행한다. 첫째, 문헌 조사에서는 소프트웨어 요구사항의 진화에 대한 이해 및 현 자동화 연구의 수준을 분석한다. 둘째, 실증 조사에서는 오픈 소스 프로젝트에서 요구사항 변경이 발생할 때 소프트웨어 시스템에 어떤 변경이 발생하는지를 조사한다. 이를 통해, 진화적 관점에서의 요구사항 관리의 어려움을 분석하고, 추적성 적용에 실질적으로 도움을 줄 수 있는 자동화 연구 방향에 대해서 논의한다.

본 논문의 구성은 다음과 같다. 2절에서는 소프트웨어 요구사항의 진화를 이해하기 위한 기본 개념들을 정리한다. 3절에서는 요구공학 프로세스와 자동화 연구의 문헌 조사 내용을 정리한다. 4절에서는 3절 문헌 조사를 통하여 이해한 요구사항의 체계적인 진화의 어려움을 정리한다. 4절에서는 실제 오픈 소스 프로젝트에서의 요구사항 진화를 조사한다. 6절에서는 3절부터 5절까지의 조사를 바탕으로 기존 자동화 연구의 한계점과 향후 연구 방향을 논의한다. 7절에서는 본 논문의 결론을 내린다.

2. 기본 개념

2절에서는 소프트웨어 요구 사항의 진화와 관련한 3가지 기본 개념을 정리한다.

2.1 소프트웨어 진화

소프트웨어 진화는 소프트웨어 시스템을 처음 설치한 시점부터 변경하는 모든 변경을 의미한다 [5]. Lehman은 1977년 미국의 소프트웨어 비용의 70%이상을 소프트웨어 유지보수에 사용하고 있음을 주목하고, 소프트웨어 진화의 법칙을 연구하였다[5]. 그의 연구에 따르면, 소프트웨어 시스템은 현실의 불명확한 문제를 해결하기 위해 개발되고 나면 현실의 일부가 되므로, 지속적인 변경에 노출될 수 밖에 없다. 따라서, 소프트웨어는 지속적으로 변화하며, 점증적으로 새로운 기능을 추가하면서 복잡도가 높아진다[5].

2.2 요구 공학 프로세스

요구 공학 프로세스는 관련자들로부터 소프트웨어에 대한 요구 사항을 수집하여 기대 수준에 부합하는 소프트웨어 시스템을 만들기 위한 요구 사항 관련 프로세스이다[6, 7]. 요구 공학 프로세스는 수집, 분석, 명세, 검증, 관리의 5가지 단계로 구성되며, 각 단계의 요약은 다음과 같다 [7]:

- 요구사항 수집: 고객, 기술, 환경에서의 요구사항을 수집한다.
- 요구사항 분석: 개발 단계별로 발전하는 요구사항 할당과 비즈니스, 사용자, 시스템, 소프트웨어 수준별로의 상세화를 진행한다.
- 요구사항 명세: 요구사항마다의 고유한 아이디 명시하고 사용사례와 품질 속성을 포함한 요구사항을 기술한다.
- 요구사항 검증: 시스템 수준의 요구 사항과 상세 수준의 요구 사항의 일치 등을 검토한다.
- 요구사항 관리: 추적성 수립을 통하여 소프트웨어 요구 사항의 생명 주기를 관리한다.

2.3 소프트웨어 추적성

소프트웨어 추적성은 “고유하게 식별 가능한 소프트웨어 공학 산출물을 다른 것과 연관시키는 기능”이다 [8]. 추적성을 수립하기 위하여 주로 요구사항 추적성 매트릭스(Requirement Traceability Matrix 혹은 추적성 테이블)를 사용하는데, 추적성 매트릭스란 개별 기능 요구사항과 다른 시스템 산출물 간의 논리적 맵핑을 설명하는 표로서, 시스템 요구사항, 소프트웨어 요구사항, 설계 항목, 코드 항목, 시험 항목으로의 맵핑을 순차적인 배열로 표현한다 [8]. 전형적인 추적성 매트릭스는 표 1과 같다.

[표 1] 전형적인 추적성 매트릭스 ([8,9])

요구사항	설계모듈	구현파일	테스트항목
RE001	DE003	IM002	TE002
…	…	…	…

이러한 추적성 맵핑 관계는 소프트웨어 시스템의 크기와 복잡성에 따라 기하 급수적으로 커지는 문제점이 있다.

3. 문헌 조사

3절에서는 소프트웨어 요구사항의 진화 및 자동화 연구를 문헌 조사한다. 표 2는 조사한 결과를 보여준다. 표 2에서는 연구를 개발과 진화로 나누어 요구사항 개발과 관련된 문헌은 일반 폰트로, 요구사항 진화와 관련된 문헌은 볼드체에 밑줄로 표시하였다. 세로축의 상세 분류는 기법 및 도구, 문헌 조사, 실현상 조사로 나눈다. 가로축으로는 먼저 요구사항 명세 전과 명세 후 활동으로 구분하고, 상세 분류는 수집, 분석, 검증, 관리의 활동으로 구분한다. 해당 표를 바탕으로 3.1와 3.2절에서는 기법 및 도구, 3.2절에서는 문헌 조사, 3.3절에서는 실현상 조사로 나누어 요약, 논의한다.

3.1 요구사항 수집, 분석, 검증의 자동화 기법 및 도구

요구사항 수집, 분석, 검증에 있어서 소프트웨어 진화와 관련된 연구는 요구사항 수집에서 있었다. 요구사항 분석 및 검증에서 소프트웨어 진화와 연동한 연구는 우리가 찾는 문헌 내에서는 없었다.

3.1.1 요구사항 수집의 자동화 연구

2013년도 논문 [10]은 소프트웨어 진화에 반영해야 하는 요구 사항의 자동 수집에 초점을 두고 있다. 이를 위해 사용자 코멘트를 분석해서 요구사항을 자동으로 도출하기를 제안하였다. 그리고 부정적인 내용에 대한 주제를 도출하기 위해 Aspect and sentiment unification Model(ASUM) 데이터 마이닝 기법을 사용하였다. 논문 [10]은 Manual 기법보다는 상대적으로 시간을 단축할 수 있다는 부분을 확인하였다. 단지, 일반적인 분석기법, K-median Clustering 기법과 비교하여 ASUM 적용 성능은 비슷하였다. 논문 [10]은 사용자 코멘트를 분석하는 토픽 모델링기법의 유용성을 처음으로 평가한 논문으로, 사용자들이 실제 요구하는 사항들을 분석함으로써 소프트웨어 진화에 반영되어야 하는 요구사항을 편리하게 도출할 수 있도록 돕는다.

3.1.2 요구사항 분석의 자동화 연구

2016년도 국내 논문 [11]은 자동화된 요구사항 우선 순위 분석을 제안하였다. 논문 [11]은 요구사항의 수가 많은 경우, 적용하기가 힘든 확장성 문제와 이해관계자의 가치판단에 노출되는 편향성 문제를 지적하였다. 그리고 정제된 요구사항과 정제되지 않은 요구사항 사이의 연관성을 계산하여 정제된 요구사항의 우선순위를 정하는 ToMSN 기법을 제시하였다. 또한 출입 제어 시스템 프로젝트의 정제되지 않은 요구사항과 정제된 요구사항에 대하여 ToMSN 기법을 적용한 사례를 제공하였다. 논문 [11]의 신규성은 중립적이고 풍부한 요구정보 데이터를 적극적으로 활용하는 기법을 선보인데 있다. 그러나, 요구사항의 진화에서 나타나는 우선순위 갱신 문제를 다루지는 않는다.

[표 2] 문헌 조사표

<table>
<tr><td rowspan="4">개발 및 진화</td><td>실현상 조사</td><td></td><td>Robinson et al. 2015
Schneider et al. 2016</td><td></td><td>Gotel et al. 1994
Asuncion et al. 2007
Rempel et al. 2014
Mader et al. 2015</td></tr>
<tr><td>문헌조사</td><td colspan="3">Ernst et al. 2014
Achimugu et al. 2014</td><td>Cleland-Huang et al. 2014</td></tr>
<tr><td>기법 및 도구</td><td>Carreno et al. 2013</td><td>Jang et al. 2016</td><td>Ko et al. 2016</td><td>Kim et al. 2010
Kim et al. 2012
Eom et al. 2015
Cleland-Huang et al. 2003
Klock et al. 2011
Cleland-Huang et al. 2014
Saito et al. 2016
Garcia et al. 2016</td></tr>
<tr><td colspan="2"></td><td>수집</td><td>분석</td><td>검증</td><td>관리</td></tr>
<tr><td colspan="2"></td><td colspan="2">명세 전</td><td colspan="2">명세 후</td></tr>
</table>

3.1.3 요구사항 검증의 자동화 연구

2016년도 국내 논문 [12]는 요구사항 패턴에 기반한 누락 요구 사항의 검증을 제안하였다. 논문 [12]에서는 요구사항 수집에 있어서 항목 누락이 될 수 있는 문제에 대하여, 요구사항 시나리오에 대해 기계학습을 적용하여 소프트웨어 요구사항 패턴을 추출하는 기법을 제시하였다. 평가를 위해서는 여덟 개의 프로젝트의 요구사항 시나리오에서 패턴을 시범적으로 도출하여 요구사항 패턴 추출의 자동화가 가능함을 보였다. 논문 [12]의 결과는 요구사항을 수집할 때 요구사항의 누락을 방지할 수 있도록 가이드라인을 생성하는데 활용할 수 있다. 논문 [12]는 요구사항 수집에 초점으로 두는 반면, 그 요구사항의 진화에서 발생하는 일관성 검증 문제를 다루지는 않는다.

3.2 요구사항 관리의 자동화 기법 및 도구

요구사항 관리는 2.3절에서 언급한 것과 같이 추적성 수립을 통한 방법을 주로 사용한다. 그러나 추적성 수립 이후의 소프트웨어 진화 방법에 대응하는 부분에 대해서는 자동화되어 있다고 보기 힘들다.

3.2.1 요구사항 관리의 기법 연구

2010년도 국내 논문 [9]는 요구사항 변경 시에 추적성 테이블의 요구사항 ID를 어떻게 관리할지를 제안하였다. 논문 [9]는 기존 추적테이블 연구는 변경관리 방법 및 추적의 효과를 구체적으로 제시하고 있지 않고 있음을 지적하였다. 또한 변경 영향 추정연구는 추정방법이 복잡하여 실용성에 한계가 있음을 주목하였다. 논문 [9]는 변경 요구사항에 대해서 새로운 ID를 부여하자고 제안하였다. ID의 변경요구사항에 대해서 변경의 원천 요구사항을 알 수 있도록 기재함으로써 변경율 및 변경영향도를 기존 연구에 비해 용이하게 추정할 수 있다고 주장하였다. 그러나, 이러한 제시 방법은 기존 추적성 방법에 변경 요구 사항을 어떻게 추가할 것인가의 개선 제안이라는 한계점을 가지고 있다. 즉, 요구사항 진화에서 발생하는 다양한 영향 요소 및 요구사항 변경에 영향을 받는 소프트웨어 산출물의 다양한 내부 관계(예: 요구사항 간의 계층 등)를 고려하지 않으며, 이에 따라 요구사항 진화를 통합 관리할 수 있는 방법이라고 보기 어렵다.

2012년도 국내 논문[13]은 기존의 추적 기법은 추적의 정도에 대한 정량화된 평가 방법과 판정 기준이 없다고 지적하였다. 따라서 정략적인 추적의 정도를 수치로 표현하기 위해서 요구사항 추적 모델에서의 선행항목이 후행항목과 관련되는 정도를 나타내는 요구사항 전개도 정도를 수식으로 일반화 하여 추적성을 정량적으로 나타내었다. 또한 산림청 정보화 사업 16개를 대상으로 사업 사례분석을 통해 요구사항 반영의 적정성 판정 사례를 보였다. 논문[13]은 저자들의 과거 연구보다 좀 더 다양한 추적 환경을 고려하여, 추적성의 정도를 정량화하였다. 그러나 소프트웨어 요구사항의 진화와 연관시키지는 않았다.

2014년도 국내 논문 [14]는 제품라인에서의 요구사항과 아키텍처 사이의 추적성 구축에 대한 연구를 진행하였다. 논문 [14]에서 주목한 문제로 소프트웨어 요구사항과 아키텍처 설계 사이의 추적성 구축은 요구사항과 아키텍처 구성 요소 사이의 다대다 관계와 아키텍처 계측으로 고려해야 할 요소가 많으며, 제품라인으로 가면 복잡도가 더 증가한다. 논문 [14]에 서는 이를 해결하기 위해 제품라인 및 아키텍처 계층을 고려한 추적성 수립을 위한 재귀적인 방법을 제시하였다. 사례 연구로는 전자레인지 제품라인 개발의 추적성 모델링을 진행하였다. 논문 [14]는 소프트웨어 제품 라인의 진화를 돕기 위하여 요구 사항과 소프트웨어 아키텍처를

맵핑하는 추적성 수립 방안을 제시하였다. 그러나, 추적성 수립일 경우로 국한되며, 이후의 요구사항의 진화에 대응하는 연구는 아니다.

3.2.2 요구사항 관리의 도구 연구

소프트웨어 추적성 수립을 지원하기 위한 다양한 요구사항 관리도구 들이 있다 [15]. 그 중 가장 유명한 상용화 도구는 DOORS이다 [16]. DOORS의 주요 기능은 요구사항 간의 혹은 요구사항과 다른 산출물 간의 추적성(Traceability) 명시와 히스토리, 베이스라인, 교환 등이다. DOORS의 문제점은 프로세스를 수정한 후 이를 반영할 때 자유롭게 변경 할 수 없는 제약이 있다. 많은 도구가 개발되었음에도 실질적인 요구사항 추적성의 적용과 활용이 성공적이지 않음은 오랫동안 인식되어 왔다 [1, 2, 17]. 이를 해결하기 위하여 진화적 측면의 요구사항에 대한 변경 전파를 위한 연구 및 제안들이 있었다 [8, 18, 19, 20, 21].

2003년도 논문 [18]은 요구사항의 변경 전파를 위하여, 이벤트 기반의 추적성 전파 시스템을 제안하였다. 제안 시스템에서는 요구사항의 변경을 변경 이벤트로 간주하고 이에 영향을 받는 산출물을 이벤트를 받아야 하는 구독자로 간주하였다. 그리고 요구사항의 변경이 발생할 때 이벤트 메시지에 수정해야 할 사항을 담아 넘기면 영향을 받는 산출물에 대해 해당 사항을 반영하였다. 사례 연구로 M-Net이라 는 웹 기반 학회 시스템을 대상으로 제안 시스템을 적용하여 활용 가능성을 보여 주었다.

2011년도 논문 [19]는 수동으로 산출물간에 추적성 수립을 하는 것은 최신 상태의 유지가 힘들다는 부분을 지적하였다. 해결 도구로 제안한 Traceclips는 자연어 분석과 정보 검색 기술을 활용하여 추적성 링크를 유추하여 사용자에게 보여주고 사용자 친화적인 인터페이스를 제공하여 개발자가 추적 가능성 링크를 관리 가능하게 한였다. 또한, 평가를 위해서 루신 프로젝트를 대상으로 75개의 추적성 링크를 자동, 유추하여 정확도가 약 40%, 재현도가 약 30%임을 보여주었다. 논문 [19]는 요구사항의 추적성의 갱신에 관하여 추적성을 자동으로 수립하는데 공헌하였다.

2014년도 논문 [20]은 추적성 데이터가 종종 불완전하고 부정확하며 중복되고 상충되고 더 이상 유효하지 않아 신뢰성이 낮음을 지적하였다. 이를 위해 추적성 도구에서 링크를 신뢰할 수 있는 추적 링크와 신뢰할 수 없는 링크로 구분하기를 제안하였다. 또한, 신뢰할 수 있는 링크들에서 더 이상 신뢰할 수 없는 링크를 끊임없는 제거하기를 제안하였다. 마지막으로, 추적 링크를 체계적으로 구축하거나 재구성하는 데 사용되는 신뢰할 수 없는 추적 링크 증거를 모으기를 제안하였다. 논의를 위해 두 개의 변경 시나리오를 제시하였으나 실제 사례를 보여주지는 못했다. 단지, 사용자가 추적 링크를 변경할 때, 링크들을 점검하는 활동을 언제할지 선택할 수 있다는 이점을 논의했다.

2016년도 논문 [21]는 요구사항 진화를 위한 시각화 도구를 제시하였다. 제시 도구는 정의된 일곱가지의 요구사항 진화 이벤트를 사용하여 요구사항 진화 차트를 시각화하였다. 도구 유용성을 평가하기 위해서는 문서 관리 및 승인 시스템의 개발자를 두 개의 그룹으로 나누어 활용하도록 하였다. 그 결과, 도구를 활용한 그룹이 도구를 활용하지 않은 그룹보다 변경된 요구 사항에 대해서 22프로 더 정확하고 15프로 더 신속하게 이해함을 확인하였다. 이 때의 추적성은 요구사항 간의 진화에 초점을 맞추고 있으며 시스템 반영 추적성은 시각화하지 않는다.

2016년도 논문 [8]은 요구사항의 추적성을 소프트 웨어 구현물에 직접 매핑하는 도구를 제시하였다. 논문 [8]에서는 산업계에서 기존 요구사항 관리 도구를 50% 밖에 사용하지 않음을 주목하였다. 그 이유로 기존의 추적성 도구가 산업계의 요구에 부적절하다고 보았다. 이에 따라 웹 애플리케이션의 요구사항과 구현 산출물간의 매핑할 수 있는 도구를 제안하였다. 제안 도구에서는 웹 페이지와 HTML 요소 및 컨트롤과 같은 구현 산출물을 클릭한 후, 기능 요구 항목을 드롭박스 에서 클릭하면, 해당 매핑 관계를 나타내는 추적성 링크를 데이터베이스에 저장한 후, 사용자가 원할 때 해당 추적성 매트릭스를 시각화하여 보여준다. 제안 도구은 추적성 링크를 생성하기 위해서는 인간 상호 작용을 지원하며, 요구사항과 실질적인 산출물인 웹 요소 사이의 직접적인 매핑을 하여 그 결과를 시각화하는 특성이 있다. 그러나 복잡한 소프트웨어 분석, 설계, 구현 항목들의 맵핑 관계를 간과한 측면이 있다. 또한 사람이 넣은 것 이상의 매핑 관계를 유추하여 보여주는 부분이 없다.

3.3 문헌 조사 연구

요구공학 분야에서의 문헌 조사 연구는 비교적 최근에 진행된 것으로 보인다. 문헌 조사는 주로 과거의 연구들을 정리함으로써 연구 방향을 수립하기 위해 진행되었으며, 우리가 찾은 3개의 문헌 조사는 요구분석 [22], 요구사항 진화 [23], 추적성 연구 [2] 각각에 초점을 맞추고 있다. 그러나 본 논문에서 추구하는 진화적 관점에서의 요구 사항 관리의 어려움에 대한 논의를 제공하지는 않는다.

2014년도 논문 [22]는 요구사항 분석의 자동화(3.1.2절 참조)와 관련된 요구 사항의 우선순위 결정 기법들을 모아 분석하였다. 분석 방법으로는 체계적인 문헌 검토 기법을 이용하여 기존의 우선순위 결정 기술에 대한 논문을 조사하고 분석하였다. 그 결과 분석 계층화 과정(Analytic hierarchy process), 과학적 연구 개발 방법(Quality functional deployment), 게임 계획(Planning game)등이 자주 연구, 활용됨을 밝혔다. 또한 기존 우선순위 결정 기술들은 확장성의 부족, 요구사항 진화 중 순위 업데이트 처리 방법, 이해 관계자 사이의 조정, 요구사항 종속성 등의 한계가 있음도 보였다. 논문 [22]에서 요구사항의 진화와 관련하여서는 우선순위 갱신이 어려움을 논의한 파트가 있었다. 단, 요구 분석의 우선 순위 기법에 대해 초점을 두며, 요구 진화에 따른 관리의 어려움을 이해하고자 하는 우리의 시도와는 초점이 다르다.

2014년도 논문 [23]은 요구사항 진화에 대한 문헌 조사를 진행하였다. 논문 [23]은 요구공학이 요구사항의 변경을

포함해야 함을 지적하였다. 이를 위해 요구사항의 변경이 소프트웨어 아키텍처와 구현에 미치는 영향을 분석하기 위한 프로세스 모델을 찾아내고자 하였다. 요구 사항의 변경을 이해하기 위해 문제를 요구사항 R, 요구사항 명세서 S, 그리고 도메인 지식 W의 관계로 정형화 하였다. 다음, 정형화한 문제를 바탕으로 각 기존 연구의 형태를 모델링하여 요구사항의 진화에 대한 프로세스 프레임워크를 제시하였다. 해당 논문은 요구 사항이 변경될 때의 처리 프로세스를 모델링하였으나, 요구사항 변경의 어려움 혹은 복잡성의 원인에 대한 분석은 하지 않았다.

2014년도 논문 [2]는 추적성과 관련한 과거 10년간의 연구를 분석함으로써 연구 방향을 찾고자 하였다. 논문 [2]도 추적성이 중요하다고 인식되고 있으나, 실제적인 적용이 힘든 점에 주목한다. 특히 안전 중심의 소프트웨어 개발에서조차도 추적성 수립이 인증 바로 전에 만들어지고 있어 실질적인 활용이 되지 않고 있는 문제를 지적하였다. 문헌 조사를 위해서는 세가지 관점, 목표, 프로세스, 내부구조를 정의하고, 이 세가지 관점에 기반하여 과거 10년간의 추적성 연구들을 분석하였다. 그 결과, 일반적인 추적성 연구가 50%가 넘고, 특정한 목표를 가정한 추적성 연구는 약 25%가 됨을 확인하였다. 또한 추적성 생성 연구에 40% 이상의 비중이 몰려 있음을 확인하였으며, 마지막으로 추적 알고리즘에 50%의 연구의 노력들이 몰려 있음을 확인하였다. 마지막으로 문헌 조사를 바탕으로 향후 연구 주제를 논의하였다. 논문 [2]는 도입부에서 지적한 추적성의 실제적인 적용이 왜 힘든지에 대한 논의는 하지 않았다.

3.4 실현상 조사 연구

요구공항 분야에서의 실현상 조사 연구는 주로 요구사항 진화의 현상에 초점을 맞추고 있다. 추가로 기존에 개발된 추적성 지침 및 도구의 실제적인 적용 및 효과에 대한 검증 연구가 있었다.

3.4.1 요구사항 진화의 실현상 연구

2015년도 논문[24]는 오픈 소스 프로젝트에서의 요구공학의 적용을 분석했다. 논문 [24]에서는 기 제안된 six-V 측정 모델을 바탕으로 오픈 소스 프로젝트를 수치적으로 분석하였다. 여기서 six-V는 Volume(양), Veracity(진실성), Volatility(휘발성), Vagueness(모호성), Variance(분산), Velocity(속도)를 의미한다. 논문 [24]는 sourceForge에 있는 31가지의 프로젝트에 대해서 각 V의 속성이 있는지 가설을 세워서 각 V의 요구사항 속성을 수치적으로 표현하고 통계적으로 가설을 만족하는지 검증하였다. 논문 [24]는 지속적으로 변화하는 요구사항에 대하여 수치적으로 분석하여 객관적인 자료를 도출시킬 수 있음을 보여주었다. 하지만, 소프트웨어 요구사항에서의 진화가 왜 어려운지, 어떻게 그 어려움을 극복할 수 있는지에 대한 논의를 제공하지는 않는다.

2016년도 논문 [25]는 요구사항의 진화의 매커니즘을 제시하기 위해, 한 회사의 소프트웨어 시스템의 생명주기 전반에 걸친 요구 사항의 진화를 분석하였다. 이를 위해 인터뷰 (2013 년 및 2014 년), 문서 수집 (internal slide decks, 스프레드 시트, 공개 문서), 비공식 임시 질문 (전화, 전자 메일, 비공식 현장 미팅 및 즉석 질문) 등을 수행하였다. 그 결과, 문맥적인 요소를 바탕으로 이벤트가 발생하면, 이러한 이벤트들이 요구 사항과 소프트웨어 시스템의 진화를 발생시키고, 이러한 진화된 결과물을 바탕으로 경험이 축적되어 또 다른 이벤트를 발생시키는 메커니즘을 밝혀 내었다. 또한, 구성을 설정하거나 유지 보수성을 강화시키는 요구사항들이 소프트웨어 생명 주기가 흐를 수록 좀 더 중요한 우선순위를 차지함도 밝혀 내었다. 그러나, 논문 [25]는 한 회사의 전사적 시스템 개발 및 사용의 경험 데이터를 관찰한 것으로, 겉으로 보이는 요구 사항에 대한 결과를 도출했을 뿐, 요구사항 진화의 어려움에 대한 요인 분석은 제공해 주지 않는다.

3.4.2 추적성 지침 및 도구 적용 연구

요구사항 추적성이 요구사항의 진화를 지원하지 못하는 문제점에 대해서는 오래 전부터 인식되어 왔다. 산업계에서 추적성을 활용하기에는 부적절한 이유를 파악하기 위해서 1994년 이미 실질적인 사용 조사를 진행한 바 있다. 논문 [17]은 그 당시 이미 100가지가 넘는 요구사항 추적성 도구의 강약점을 파악하고 100여명을 대상으로 설문조사와 인터뷰를 진행하여 추적성 활용의 문제점을 조사하였다. 논문 [17]의 결론은 첫째, 사전 RS 추적성에 대한 연구를 수행할 필요가 있으며, 둘째, 의사소통을 원활하게 하기 위해, 구체화한 그리고 정제한 요구 사항을 신속하게 찾아 접근할 수 있는 지속적인 능력이 필요하다는 것이었다. 25년이 지난 현 시점에서는 3.1.1절과 같이 사전 RS 추적성에 대한 자동화도 제시되고 있어, 해당 논문에서 주장한 문제점을 위한 솔루션이 나오고 있으나, 아직 상기 지속적인 능력에 대해서는 연구자들이 합의한 해결 방법을 찾지 못하고 있다.

2007년도 논문 [3]에서는 Doors와 같은 전문화 된 추적 도구가 채택 된 회사에서도 툴의 완성도, 좁은 초점 및 다른 도구와의 상호 운용성 부족으로 인해 추적성 문제가 존재함을 보고하였다. 따라서 이를 경제, 기술, 사회적 관점에서 보완한 실제 회사 제약 조건 내에서 최종 추적성(end-to-end tacebility) 소프트웨어 추적 도구를 개발하여 제공하였던 사례를 보고하였다. 개발한 도구는 웹을 통해 원격 사용자를 포함 조직의 모든 구성원이 즉시 엑세스 가능하게 하였다. 도구 사용 사례를 위해서 산업 소프트웨어를 전문적으로 개발하는 IT회사(직원 250명)의 실제 프로젝트 데이터로 저장소를 채워 개발한 도구를 사용하여 테스트를 진행하였다. 그 결과 추적성 관련 작업이 2배 빨라졌음을 보고하였다. 이는 추적성 데이타를 활용할 때 생산성이 높아질 수 있음을 보여주는 데이터이지만, 과거 데이터로 테스트를 진행한 점에 있어서 평가가 제약되는 점이 있다.

2014년 비교적 최근 논문 [1]에서도 여전히 실제 프로젝트 상황에서 요구사항 추적성에 대한 지침이 적용이 되지 않고 있음을 실증적으로 보여준다. 논문 [1]에서는 이러한 평가를

위하여, 요구사항 추적성 가이드라인으로부터 정형 모델을 만든 다음 프로젝트 데이터로부터 해당 정형 모델로의 맵핑을 분석한다. 이후 정형화된 추적성 문제 규칙을 적용하여 추적성 문제와 타입을 명시한다. 논문 [1]은 해당 평가 모델로 5가지 기술지침에 대해 7가지 안전성 소프트웨어 시스템의 추적성을 평가했으며, 대부분의 프로젝트가 50퍼센트 이상 추적성 가이드라인에 부합하지 않음을 보고한다. 논문 [1]에서는 가이드라인에 의해 규정된 것과 실제로 구현되는 것 사이에 차이가 있음을 객관적인 데이터로 보여준다.

2015년 논문 [4]에서는 요구사항 추적성의 유용성에 대한 평가가 발표된 적이 없음을 지적하였다. 이를 객관적으로 증명하기 위해 사용자 실험을 진행하였다. 실험 방법으로 2개의 제 3자 개발 프로젝트에서 실무자부터 다양한 경험이 많은 학생들까지 총 71명을 대상으로 추적성을 적용한 그룹과 적용하지 않은 그룹으로 나누어 실제 유지 보수 작업을 다시 수행하도록 하였다. 평가를 위해서는 2x2x4x4 계승 설계를 사용하였고, 두 프로젝트의 각각에 대해 4 개의 작업이 정의하고 각각 추적성을 사용하거나 사용하지 않고 작업을 수행하였다. 그 결과 추적성을 사용한 그룹이 사용하지 않은 그룹에 비해 평균적으로 24% 빠르게 작업을 하였고, 50% 더 정확한 해결책을 만들었음을 통계적으로 보여주었다. 이러한 연구는 추적성이 주는 생산성 향상을 객관적으로 보여주는 자료로 유용하다. 문제는 실험 연구이기 때문에 실질적인 추적성 수립 및 진화에서 발생하는 이슈는 배제한 실험이라는 것이다.

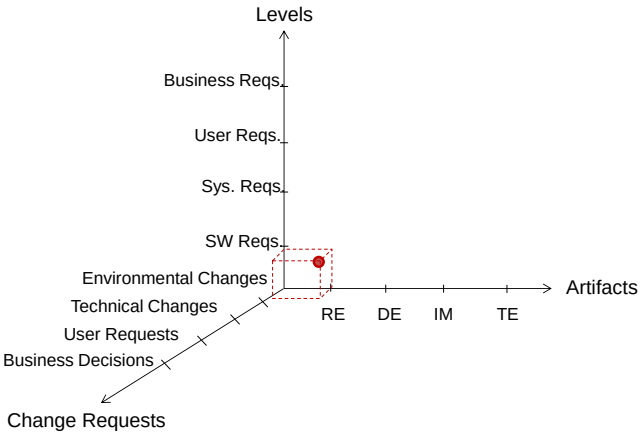
4. 요구사항의 진화에서의 어려움 요인 논의

4절은 문헌 조사를 통해 파악한 다음 질문의 논의를 정리하고자 한다: “요구사항의 변경에 따른 체계적인 소프트웨어 진화가 왜 어려운가?”

우리는 요구 사항의 변경과 연관된 다양한 측면이 존재하며, 요구 사항의 변경에서 발생하는 활동은 일반 코드의 변경과 유사하고, 마지막으로 소프트웨어 개발 산출물의 어느 항목에서 변경이 발생하든지 이를 반영할 수 있어야 한다는 사실을 발견하였다. 각각의 논의를 세부 절로 정리한다.

4.1 요구사항의 다차원적인 진화

논문 [6]의 요구공학 분석에서는 요구사항과 관련된 여러 다른 측면을 기술한다. 여기에서 도출할 수 있는 사실은 요구사항의 변경을 다루기 위한 여러 측면이 존재한다는 것이다. 그림 1은 그 중에 세 가지 측면을 차원으로서 표현해 본 것이다. 그림 1에서 변경요청의 측면에는 사용자의 요청, 기술적인 변화, 환경적인 변화 등이 있다. 요구사항 수준의 측면에서는 상위 수준의 요구 사항으로부터 하위 수준의 요구 사항이 존재한다. 예를 들어 사용자 수준의 요구 사항과 소프트웨어 시스템의 요구 사항으로 세분화될 수 있다. 마지막으로 변경영향 항목의 측면에서는 요구 항목의 변화로 영향을 받는 산출물로 개발, 구현, 시험까지의 각각의 항목이 될 수 있다 (문헌 [6]의 II.A.2절 참조).



[그림 1] 요구사항에서의 진화와 연관된 항목의 차원

논문 [14]에서도 비즈니스 결정사항을 포함한 문맥적 요소로 표현된 요구사항의 변경 요청을 어떻게 요구 사항 및 시스템에 반영하는지에 대한 메커니즘을 논의하고 있으나, 세분화된 요구사항과 산출물의 반영은 고려하고 있지 않다. 논문 [26]에서는 “multi-faceted” 추적성 문제를 언급하며, 추적성의 어려움에 많은 요인들이 연루되어 있음을 지적한다. 이를 통해 요구사항의 다차원적인 진화를 다루는 것이 어렵다고 본다.

4.2 요구사항의 변경 타입

4.1절의 다차원에서 발생하는 요구 사항의 진화에는 시간의 흐름이라는 이라는 추가 측면이 존재한다. 예를 들어 그림 1에서 환경적 변화 요청에 따라 요구명세서의 소프트웨어 요구 사항이 변경되었다면, 해당 변경이 무엇인지를 명시하는 것이 요구 사항의 진화를 이해하는데 도움을 줄 수 있다.

논문 [18, 21]은 요구사항의 변경 타입을 다음의 7가지 변경 타입으로 제시한다.

- 생성(Create): 새로운 요구사항을 생성한다.
- 비활성화(Inactivate): 요구사항을 비활성화한다.
- 수정(Modify): 요구사항의 속성이 일부 바뀐다.
- 세분화(Refine): 하나의 요구 항목이 항목의 의미가 변하지 않고 추가적인 요구사항으로 상세해진다.
- 분할(Decompose): 하나의 요구 사항이 여러 요구 사항으로 분할된다.
- 통합(Merge): 여러 요구 사항이 하나의 요구 사항으로 통합된다.
- 대체(Replacement): 하나의 요구 항목이 폐지되고 다른 요구 사항으로 바뀌는 것을 뜻한다.

상기 변경 타입을 검토하면 요구사항의 변경 또한 일반적인 변경과 같이 추가, 삭제, 수정이라는 부류가 있으며, 해당 부류들을 7가지로 세분화할 수 있음을 볼 수 있다. 이를 통해 요구사항의 진화에 있어 다차원에서 발생하는 변경에 대해서 변경 타입을 고려하여 소프트웨어 산출물에 반영하는 패턴을 볼 수 있다고 유추한다.

4.3 변경의 순방향, 역방향 전파

4.1절의 다차원적인 측면에서 변경은 여러 측면의 조합 중 어디에서나 발생할 수 있음을 파악하였다. 또한 4.2절의 변경 타입을 통해 요구사항의 변경이 일반적인 변경의 타입을 따름을 파악할 수 있었다. 이는 우리가 요구사항의 변경에 한정하지 않고, 어느 항목에서 변경이 일어나든 전파가 가능해야 한다는 것을 뜻한다.

반면 표 1의 추적성 매트릭스는 순방향 전파를 가정하고 있다고 볼 수 있다. 즉, 요구, 설계, 구현, 테스트 항목에 있어, 선행항목의 변경이 발생하면 어떤 후행항목이 영향을 미치는지를 분석할 수 있는 정보의 명시를 주 대상으로 한다. 예를 들어, 요구명세서의 요구 항목을 변경하면, 이를 구조 설계서의 모듈에 반영하고, 다음으로 상세 설계서 및 코드에 반영을 하는 방식이다. 우리는 이러한 가정이 틀렸다고 본다. 변경은 모든 소프트웨어 산출물 항목 어디서든 발생할 수 있다고 보는 것이 더 실제적이다. 따라서 순차적인 할당이 아니라 어디서든 발생하는 항목의 변경에 따른 전파를 지원하는 추적성 수립 형태가 있어야 한다고 본다. 따라서 선행 항목의 변경이 후행 항목에 미치는 영향을 분석하고 또한 후행 항목의 변경이 선행 항목에 미치는 영향을 분석할 수 있는 순방향, 역방향 전파의 영향 분석이 가능해야 한다.

5. 실 사례에서의 요구 사항의 진화 조사

4절에서는 실 사례에서의 요구 사항의 변경에 대하여 조사한다. 조사 대상은 오픈 소스로 운영되는 코드 편집기로써 GitHub가 개발한 ATOM 에디터이다[1]. ATOM 에디터는 2015년 6월 25일에 정식 버전이 출시되었으며 커뮤니티와 Git hub 사이트를 통해 사용자들과 소통이 활발하게 이루어지고 있다. 오픈 소스의 특성상 이슈 게시판에 사용자이 요구사항을 요청하기도 하고 토론도 진행되며, 개발자들은 요청 내용을 적극적으로 수용하여 패치를 배포한다.

본 논문의 저자 4명은 GitHub 싸이트의 ATOM 이슈 게시판에 올려진 사용자들의 요청사항을 조사하였다. 조사 방법은 각자 ATOM에디터에서 기능 하나를 임의로 선택하였고, 이슈 게시판의 검색과 필터링 기능을 사용하여 시스템 반영이 완료가 된 요청사항만을 조사 대상으로 하였다. 각 요청사항에 대한 이슈 보고서를 읽으면서 개발자 사이에서 어떠한 논의가 오갔는지와 결과적으로 어떠한 코드 변경이 발생했는지 조사하였다. 조사 결과를 요구사항 타입(Requitemnet Type) 대비 시스템반영 타입(Code Change Type)으로 나누어서 분류하였다

요구사항의 변경타입은 앞선 4.2절에서 언급한 7가지를 참조하였다. 시스템반영 타입(Code Change Type)은 직접 프로그램의 코드가 어떻게 변경되었는지를 말하며, Add, Delete, Invalidate, Reject, Change(Modify, Replace, Deprecate, Move, Refine, Merge) 7종류로 분류하였다. 각 타입의 의미는 다음과 같다. 추가(Add)와 삭제(Delete)는 변경된 요구에 따라 새로운 코드를 추가하거나 삭제한다. 무효화(Invalidate)는 요구를 반영하기로 하였지만 그럴 필요가 없어진 사항들이다. 거절(Reject)은 요구를 거부한다. 변경(Change)은 수정, 변경, 이동, 합병 등으로 세분화한다.

표 3은 우리가 조사한 결과이다. Random하게 조사한 결과 요구 사항의 변경에 있어 생성(Create), 수정(Modify), 세분화(Refine), 대체(Replace)의 결과를 찾아낼 수 있었다. 서로 다른 타입의 요구사항의 예를 제시하면 다음과 같다.

- 생성(Create): Issue#8939에서는 Top과 Bottom bar를 에디터에 추가해줄 것을 사용자가 제안하였다. 이에 대한 코드 반영은 추가(Add)와 수정(Modify)로 발생했다. 개발자들은 기본 에디터의 틀을 변경한 뒤, Top과 Bottom bar를 나타내는 요소를 추가해주었다. 요구사항 생성에 있어서는 대부분의 코드 반영에 추가(Add)가 존재하였다.
- 수정(Modify): Issue #11486에서는 CTRL+W로 작동하는 화면 닫기 기능을 CTRL+F4로 변경해 달라는 요구가 있었다. 코드 반영은 추가(Add)로 되었는데, 이유는 CTRL+W로 변경을 원하는 사용자들 또한 있어 두 단축키 모두를 화면 닫기 기능으로 사용할 수 있도록 추가하여 주었다. #13262에서는 Folding 메뉴에서 Fold All의 위치가 다른 항목들과 매치가 되지 않는다고 보고하였다. 코드 반영은 이동(Move)가 되었는데 개발자들은 Fold All의 항목을 Folding 메뉴 내부의 구분선 위쪽으로 이동시켜 주었다. 요구사항 수정에 있어서는 대부분의 코드 반영의 타입이 다양한 것을 볼 수 있다.
- 세분화(Refine): Issue #6922에서는 업데이트 시 활성화되는 아이콘이 업데이트가 끝나도 여전히 활성화되어 릴리즈 노트를 가르치는 문제를 지적하였다. 개발자들은 원래 릴리즈 노트를 가리킬 때 아이콘을 회색으로 수정하기로 합의하였었지만, 해당 아이콘 자체가 없어지면서 수정이 무효화 되었다. 따라서 시스템 반영에서 코드 변경 계획은 무효화(Invalidate)되었다. #1690에서는 파일 내용이 변경될 때마다 자동으로 저장이 되도록 기능을 변경해 달라고 요청하였다. 개발자들은 요구처럼 변경할 경우 메모리 과부하, 기존 사용자들의 불편을 일으킬 수 있다고 하였다. 토론 끝에 활성화, 비활성화가 가능하도록 기능을 대체하는 쪽으로 마무리되었다. 따라서 코드 변경은 대체(Replace)타입이 되었다.
- 대체(Replace) #6351에서는 Bootstrap CSS를 Hand-written CSS로 변경해달라는 제안이 있었다. 개발자들은 Bootstrap의 필요 없는 부분을 삭제하고, Hand-written CSS를 추가하였으며, Bootstrap 일부를 core로 이동시켰고 기본 스타일과 병합시켰다. 여러 번의 패치를 통해 다수의 변경이 있었다. 따라서 시스템 반영에는 Add, Delete, Move, Merge의 코드 변경이 혼합되었다.

[1] ATOM, https://github.com/atom/atom

[표 3] ATOM 에디터 프로젝트의 요구사항 변경 요청에 대한 시스템 반영 타입 분석표

요구사항변경			시스템 반영	
ID	타입	설명	타입	설명
3111	Create	모든 탭을 한 번에 닫기 위해 CTRL+SHIFT+W 추가 제안	Add	패치를 통해 메뉴바에 모든탭 한번에 닫기 버튼를 추가
7332	Create	화면 크기를 마우스가 아닌 키보드 단축키를 통해 조절할 수 있도록 추가	Add	패치를 통해 화면 크기 조절관련 단축키를 추가
8939	Create	Top 과 Bottom bar 를 에디터에 추가하기를 제안	Add, Modify	기본 에디터 틀 변경 후, top 과 bottom bar 를 나타내는 요소 추가
960	Create	스타일과 기능이 모든 OS 에 기본이 되는 Native OS 가 필요	Add, Modify, Merge	편집기 맨 위에 전역 메뉴를 추가하고 메뉴의 항목들을 수정, 병합
6134	Create	'디렉토리 내 검색'이 여러 개의 루트 폴더가 있을 경우에도 실행될 수 있게 제안	Replace	새로운 라이브러리로 대체하여 Scandal 에서 패턴이 파일인지 디렉토리인지를 판단 할 수 있도록 * 문자를 포함하지 않는 경로 패턴을 명시
3064	Create	리눅스 상에서 OPenFolder 기능이 단축키로 존재하지 않는데 추가 제안	Replace, Add	CTRL+SHIFT+O 를 OpenFolder 로 대체하고 원래의 devOpenFolder 단축키를 새로 추가
11486	Modify	현재 CTRL+W 로 작동하는 화면 닫기 기능을 CTRL+F4 로 변경	Add	현재 CTRL+W 기능을 사용하는 유저들이 있기 때문에 CTRL+F4 기능을 추가
12951	Modify	드보락 키보드를 사용할시 쿼티 자판처럼 인식하는 문제를 수정	Add	드보드락 사용자를 위한 키맵 패치를 추가
21	Modify	수정된 buffer 만을 저장하도록 변경을 제안	Modify	수정된 buffer 만을 저장하도록 수정
12451	Modify	들여쓰기 안내선을 표시하지 않는 기존 편집기(즉, 기존 프로젝트를 열 때 자동으로 열리는 편집기)에서 안내선을 표시할 수 있도록 수정	Modify	텍스트에디터를 deserialize 한 후에 모든 매개 변수로 디스플레이 레이어를 재설정하도록 수정
13262	Modify	Folding 메뉴에서 Fold All 의 항목의 위치를 다른 항목과 매치되게 변경 제안	Move	Fold All 의 항목을 Folding 메뉴 내부의 구분선 위쪽으로 이동
7145	Modify	Dialogue 를 저장할 때 필터 유형을 포함하는 것으로 변경 제안	Refine	이전 동작을 유지하고 필터 유형을 포함하는 개선된 기능을 추가
11504	Modify	Atom 관련 파일의 아이콘을 확장자에 따라 다르게 변경 제안	Refine	기본 파일의 아이콘은 그대로 유지하고 다른 확장자의 파일의 아이콘을 다르게 개선하여 추가
6917	Modify	다른 프로젝트 폴더 추가 시, 변경 사항을 무시하고 이전 프로젝트 폴더로 돌아가는 경우가 발생하지 않도록 수정	Refine	기존 창의 경로를 변경 후 창을 종료하였을 때, 변경 사항이 저장되도록 하는 코드를 추가
7178	Modify	팬 이동 커서의 일관성을 유지하도록 변경	Replace	모양이 다른 커서를 같은 모양으로 대체
6922	Refine	업데이트 시에 활성화되는 아이콘이 끝나도 여전히 활성화되어 릴리즈 노트를 가리킴	Invalidate	원래는 릴리즈 노트를 가리킬 때 회색으로 수정하기로 합의되었으나, 해당 아이콘 자체가 없어지면서 수정이 무효화
3174	Refine	자동 quick jump 기능을 개선하여 그것을 해제하는 옵션을 추가 요청	Refine	quick jump 기능을 그대로 두고 그것을 해제할 수 있는 옵션을 추가
1690	Refine	파일 내용이 변경될 때 마다 자동으로 저장이 되도록 기능을 변경 제안	Replace	활성화/비활성화가 가능하도록 기능을 대체
6351	Replace	Bootstrap CSS 를 hand-written CSS 로 변경 제안	Add, Delete, Move, Merge	Bootstrap 필요 없는 부분 삭제, hand-written CSS 추가, Bootstrap 의 일부 Core 로 이동, 기본 스타일과 병합
12961	Replace	Native menu 를 HTML5 버전으로 변경 제안	Reject	은 기본 취지가 웹 애플리케이션의 느낌을 없애자는 철학에서 시작되었으므로, 해당 제안이 기본 철학에 위배되어 거절

우리가 조사한 요구사항 요청이 이슈 게시판에 올려져 있는 사용자들의 요구사항이다 보니 대부분이 수정(Modify)이나 생성(Create)으로 분류되었다. 그에 대한 시스템 반영은 추가(Add), 수정(Modify), 세분화(Refine)가 가장 많았다.

6. 요구사항의 진화를 위한 기존 연구의 한계점과 향후 연구 방향

6절은 3절-5절의 문헌 조사와 실 사례 조사를 통해 파악한 다음 질문의 논의를 정리하고자 한다: "요구사항의 진화를 위한 기존 연구의 한계점과 향후 연구 방향은 무엇인가?"

6.1 기존 연구의 한계점

우리는 3.1절과 3.2절의 자동화 연구를 통해 연구자들이 요구공학과 관련한 여러 자동화 기법과 도구들을 제시하고 있음을 확인하였다. 이 중에서 소프트웨어 요구사항의 진화와 관련된 연구를 선별하면, 다음과 같다.

- 요구사항 수집: 요구사항 자동 도출 기법[10]
- 요구사항 관리: 요구 변경 ID 관리 기법[9]
- 요구사항 관리: 변경 전파 시스템 [18]
- 요구사항 관리: 추적성 링크 유추 도구 [19]
- 요구사항 관리: 신뢰할 수 있는 링크 구분 [20]
- 요구사항 관리: 진화를 위한 시각화 도구 [21]
- 요구사항 관리: 요구사항과 구현물 맵핑 도구[8]

우리는 상기 요구사항 관련 자동화 기술들로 4절에서 분석한 문제점을 해결하여 요구사항의 진화 작업을 체계화 할 수 있는지 검토하였다.

첫째, 요구사항 수집의 자동화 기법[10]은 그림 1을 기준으로 보았을 때, 변경 요청에 있어 사용자의 요청사항의 지속적인 피드백 부분에 초점을 두고 있다. 나머지 요구사항 수집을 어떻게 할 것인지 지원하지 않는다.

둘째, 요구사항 관리와 관련한 자동화 기법[9]은 표 1의 추적성 매트릭스의 처음 진입점으로서의 요구사항 변경에 대한 ID 변경 관리 기법을 제시하고 있다. 이는 그림 1의 요구 사항에서의 수준에 대한 축을 고려하고 있지 않다. 또한, 4.3절에 지적한 임의의 항목에서 변경이 발생할 때의 추적성 관리의 이슈를 다루지 못한다.

셋째, 요구사항 관리에 있어서 요구사항과 구현물의 맵핑 기법[8]은 표 1의 추적성 매트릭스 형식을 따르지 않고, 구현물을 클릭하여 요구 사항과 바로 맵핑가능한 지원 도구를 제공한다는 점에서 혁신적이다. 단, 요구사항이 단순한 웹 애플리케이션에 한정하여 적용된다는 점에 한계가 있다.

기타, 요구사항 관리에 있어서 연구 [18], [19], [20], [21]은 변경 전파, 추적성 유추, 진화의 가시화 등 흥미로운 아이디어를 제시하고 있다. 그러나 4절에서 논의한 소프트웨어 요구사항의 진화에 대한 다차원적인 변경의 영향에 대한 추적을 지원하는 기법이라고 보기는 어렵다.

6.2 향후 연구 기대 방향

우리는 4절에서 논의한 어려움을 해결하는 해법을 찾기 위한 연구 방향을 3절의 문헌 조사와 5절의 실 현상 조사를 바탕으로 다음과 같이 제안한다.

- 요구 변경의 다차원 관리 도구 제안: 4.1절에서 밝힌 바와 같이 요구 변경은 여러 측면과 관련되어 있다. 이러한 측면의 맵핑과 분석을 지원하는 도구가 있다면 요구사항의 진화에 대한 체계적인 분석 및 관리가 가능할 것으로 기대한다. 예를 들어, 논문 [8]의 요구사항과 구현물 맵핑 도구의 아이디어를 확장하여 다차원의 항목들을 맵핑하기 위한 도구를 제안, 개발할 수 있을 것으로 예견한다.
- 요구 변경의 발생 처리에 대한 메커니즘 제안: 요구사항의 변경이 일반 항목의 변경과 동일한 형태라는 이해 하에 변경 발생 처리에 대한 변경 관리 메커니즘을 기대할 수 있다. 특히 5절에서의 오픈 소스를 통해 사용자 요청 사항의 타입과 시스템 반영 타입을 링크를 시키는 실증적 연구가 이러한 방향에 도움을 줄 것으로 기대한다. 이러한 메커니즘은 요구 사항의 변경에 대한 이력 관리를 제공하며, 또한 추가로 코드의 변경과 연계를 제공해 주어야 할 것으로 기대한다.
- 발생한 변경의 전파 처리에 대한 제안: 요구사항의 변경은 단순하게 소프트웨어 명세서의 요구사항을 변경할 때 그 뒤의 후행 항목에 대한 변경 영향을 파악하는 것이 아니다. 요구사항의 변경에 영향을 미칠 항목의 변경과 요구사항 변경에 영향을 받은 항목의 변경에 대한 처리를 수행해야 한다. 이러한 방향으로 순방향 그리고 역방향 변경 전파 자동화 프로세스 제안할 때 요구사항 변경과 관련한 자동화의 기반을 만들 수 있으리라 예견한다.

7. 결론

우리는 문헌 조사의 결과를 통해 다음과 같은 요구사항의 추적성의 어려움의 이유를 도출하였다. 첫째, 요구사항의 변경과 관련하여 다양한 측면이 존재한다. 둘째, 요구사항의 변경은 일반 항목의 변경과 유사하게 생성, 변경(상세화, 분할, 통합, 대체 등), 삭제와 같은 타입과 생명주기를 가진다. 셋째, 변경은 소프트웨어 산출물 어디에서든 발생하므로, 변경의 순방향 전파와 역방향 전파를 할 수 있어야 한다. 또한 우리는 실증 조사로 오픈 소스 프로젝트에서의 사용자 요청 사항에 따른 시스템 반영에 대한 타입을 명시하고 맵핑하여 요구사항 변경에 대한 시스템 변경 대응이 어떻게 발생하는지 이해하였다. 본 논문의 결과로 요구사항 진화의 어려움의 요인을 규명하고 실질 이슈를 이해하는데 도움을 받아, 향후 요구사항 진화 관련 자동화 연구에 기여하기를 기대한다.

8. 참고 문헌

[1] Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Jane Cleland-Huang. "Mind the gap: assessing the conformance of software traceability to relevant guidelines." Proc. 36th Int. Conf. on Software Eng. (ICSE 2014). ACM, New York, NY, USA, 943-954. 2014.

[2] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. "Software traceability: trends and future directions" Proc. of the on Future of Software Eng. (FOSE 2014). ACM, New York, NY, USA, 55-69. 2014.

[3] Hazeline U. Asuncion, Frédéric François, and Richard N. Taylor. "An end-to-end industrial software traceability tool" Proc. of the the 6th joint meeting of the European Software Eng. Conf. and the ACM SIGSOFT symposium on The foundations of software eng. (ESEC-FSE '07). ACM, 115-124. 2007.

[4] Mäder, Patrick, and Alexander Egyed. "Do developers benefit from requirements traceability when evolving and maintaining a software system?" Empirical Software Eng. 20.2, 413–441. 2015.

[5] Lehman, Meir M. "Programs, life cycles, and laws of software evolution." Proc. of the IEEE. 68.9, 1060-1076. 1980.

[6] Pandey, Dhirendra, Ugrasen Suman, and A. K. Ramani. "An effective requirement engineering process model for software development and requirements management." Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 Int. Conf. on. IEEE. 2010.

[7] Sommerville, Ian. "Software Engineering. International computer science series." Ed.9 Addison Wesley. 2011.

[8] Garcia, Jorge Esparteiro, and Ana CR Paiva. "A Requirements-to-Implementation Mapping Tool for Requirements Traceability." J. of Software. 193-200. 2016

[9] 김주영, 류성열, 황만수. "추적테이블을 이용한 요구사항 변경관리 및 추적 효과 연구." 정보처리학회논문지. D 17.4, 271-282. 2010.

[10] Galvis Carreño, Laura V., and Kristina Winbladh. "Analysis of user comments: an approach for software requirements evolution." Proc. 2013 Int. Conf. on Software Eng. (ICSE '13). IEEE Press, Piscataway, NJ, USA, 582-591. 2013.

[11] 장종인, 백종문. "토픽 모델링과 이해관계자 요구 산출물을이용한 요구사항 자동 우선순위화." 정보과학회논문지. 43.2, 196-203. 2016.

[12] 고덕윤, 박수용, 김순태, 유희경, 황만수. "요구사항 시나리오 기계 학습을 이용한 자동 소프트웨어 요구사항 패턴 추출 기법." 한국인터넷방송통신학회 논문지. 16.1, 263-271. 2016.

[13] 김찬회, 김종배. "요구사항 추적모델 개선 연구." 한국디지털콘텐츠학회논문지. 13.2, 247-254. 2012.

[14] 엄석환, 강성원, 김진규, 이선아. "소프트웨어 공학: 소프트웨어 제품 라인의 요구사항과 아키텍처 간 추적성 모델링." 정보처리학회논문지. 소프트웨어 및 데이터 공학 4.11, 487-498. 2015.

[15] PSEG, 요구사항 관리 도구, http://pseg.or.kr/pseg/casereq, (2016 년 12 월 접근)

[16] Yianna Papadakis Kantos, "IBM Rational DOORS: Linking and traceability," https://www.youtube.com/watch?v=2tN_cVQP214&list=PLFB5C518530CFEC93, Apr 2012.

[17] Gotel, Orlena CZ, and C. W. Finkelstein. "An analysis of the requirements traceability problem." Requirements Eng., 1994., Proc. 1st Int. Conf. on. IEEE, 1994.

[18] Cleland-Huang, Jane, Carl K. Chang, and Mark Christensen. "Event-based traceability for managing evolutionary change." IEEE Trans. on Software Eng. 29.9, 796-810. 2003.

[19] Samuel Klock, Malcom Gethers, Bogdan Dit, and Denys Poshyvanyk. "Traceclipse: an eclipse plug-in for traceability link recovery and management." Proc. 6th Int. Workshop on Traceability in Emerging Forms of Software Eng. (TEFSE '11). ACM, 24-30. 2011.

[20] Cleland-Huang, Jane, Mona Rahimi, and Patrick Mäder. "Achieving lightweight trustworthy traceability." Proc. 22nd ACM SIGSOFT Int. Symposium on Foundations of Software Eng. (FSE 2014). ACM, 849-852. 2014.

[21] Shinobu Saito, Yukako Iimura, Hirokazu Tashiro, Aaron K. Massey, and Annie I. Antón. "Visualizing the effects of requirements evolution." Proc. 38th Int. Conf. on Software Eng. Companion (ICSE '16). ACM, 152-161. 2016.

[22] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz'ri Mahrin " A systematic literature review of software requirements prioritization research " Inform. and Software Technology. 56.6, 568–585. 2014.

[23] Neil Ernst, Alexander Borgida, Ivan J. Jureta, and John Mylopoulos. "An overview of requirements evolution." Evolving Software Systems. 3-32. 2014.

[24] Robinson, William, and Radu Vlas. "Requirements Evolution and Project Success: An Analysis of SourceForge Projects." SYSTEMS ANALYSIS AND DESIGN (SIGSAND), AMCIS. 2015.

[25] Stephan Schneider, Jan Wollersheim, Helmut Krcmar, and Ali Sunyaev. "How do requirements evolve over time? A case study investigating the role of context and experiences in the evolution of enterprise software requirements." J. of Inform. Technology. 1-19. 2016.

[26] Asuncion, Hazeline U. "Towards practical software traceability." Companion of the 30th int. Conf. on Software eng. (ICSE Companion '08). ACM, 1023-1026. 2008.

오디오 컨텍스트의 라이프 시맨틱 분석 프레임워크

정한터 [1,O], 김수동 [2], 라현정 [3,*]
숭실대학교 컴퓨터학과 [1], 숭실대학교 소프트웨어학부 [2], (주)스마트랩 [3]
{hanterkr, sdkim777, hjla80}@gmail.com

A Framework for Analyzing Life Semantics from Audio Contexts

Han Ter Jung[1,O], Soo Dong Kim[2], Hyun Jung La[3,*]
Department of Computer Science and Engineering, Soongsil University[1],
School of Software, Soongsil University[2], SmartyLab Corporation[3]

요 약

시맨틱 분석을 통해 개인 컨텍스트로부터 라이프 스타일, 삶의 질, 건강 등의 개인이 인지하지 못하는 다양한 측면을 추론해 낼 수 있다. 본 논문에서는, 개인 컨텍스트 중 오디오 컨텍스트에 대한 시맨틱 분석의 방법론을 제안한다. 특히, 시맨틱 분석의 종류 중 하나인 개인 선호도 분석에 대하여 상세한 알고리즘을 제안한다.

1. 서 론

사람들은 자신의 선호하는 것들이나 활동에 대한 패턴을 인지하고 있다. 하지만 인간의 삶은 다양하고, 복잡하며, 예상치 못하기 때문에, 무의식적으로 인지하지 못하는 부분이 있다. 반면에, 소프트웨어 기반의 분석은 보다 더 정확하고 정밀할 수 있다. 모든 세부 사항을 알아 낼 수 있으며, 경향, 상관관계 등까지도 추론해 낼 수 있다.

2. 관련 연구

Jose [1]의 연구에서는 사용자의 관심을 식별하는 시맨틱 그래프 기반의 분석 방법을 제안한다. Jin [2]의 연구에서는 트위터에서 감정 분석 방법들을 통합하는 특징 구축 방법을 제안한다. Makazhanov [3]의 논문은 SNS를 통해 사용자의 정치적 선호도를 예측한다. 이와 같이 대부분의 시맨틱 분석에 대한 연구는 특정 분야만 고려하지만, 본 논문에서는 오디오 컨텍스트에 대하여 분야에 관계 없이 시맨틱 분석을 한다.

3. 오디오 컨텍스트 시맨틱 분석의 기술적 이슈

오디오 컨텍스트의 시맨틱 분석 알고리즘을 설계하는 데에는 두가지 기술적인 어려움이 있다. 첫째로, 오디오 컨텍스트를 분석하여 사용자의 의도나 감정을 알아내는 것이 어렵다는 것이다. 둘째는 분석 결과가 높은 정확성과 신뢰성을 가져야 한다는 것이다.

4. 오디오 컨텍스트의 시맨틱 분석 프로세스와 기법

시맨틱 분석 프로세스는 그림 1과 같다. 먼저 오디오-텍스트 변환(Speech-To-Text; STT) 클라우드 서비스를 이용해 오디오 컨텍스트를 텍스트로 변환한다. 다음으로, 변환된 텍스트를 분석하기에 앞서 전처리 과정을 거친다. 마지막으로 전처리된 텍스트를 통해 시맨틱 분석을 진행한다.

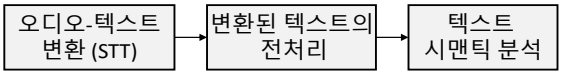


그림 1. 오디오 컨테스트의 시맨틱 분석 절차

4.1. 오디오-텍스트 변환

잘 알려진 STT 클라우드 서비스들은 오디오 파일을 텍스트로 변환할 때, 오디오 파일의 용량 또는 길이에 제한을 두고 있기 때문에 짧은 오디오 파일만 변환이 가능하다. 이러한 문제를 해결하기 위하여 사일런스 탐지 과정을 통하여 문장 사이를 발견하여 오디오 파일을 자른 후, 각각의 잘린 오디오 파일을 변환한다.

4.2. 변환된 텍스트의 전처리

변환된 텍스트를 분석하기 위해선 NLP를 사용하여 전처리를 해야 한다. 먼저, 텍스트 분할 과정을 통해 텍스트를 여러 문장으로 나눈다. 토큰화를 통해 문장을 여러 엔티티로 나누며, 품사(Part of Speech; POS) 태깅(Tagging)을 통해 각각의 엔티티에게 NLP를 통해 알아낸 품사를 태그로 나타낸다.

* 교신저자 (Corresponding Author)

4.3. 개인 컨텍스트의 선호 경향 분석

본 장에서는 개인 컨텍스트에 대한 시맨틱 분석 중 하나인 선호 경향 분석에 대한 절차 및 알고리즘을 제안한다. 선호 경향 분석은 개인 컨텍스트로부터 어떠한 사물(Thing)이나 활동(Activity)에 대하여 사용자가 얼마나 선호 하는 경향이 있는지를 분석하는 것이다. 그림 2는 선호 경향 분석의 절차를 나타낸다.

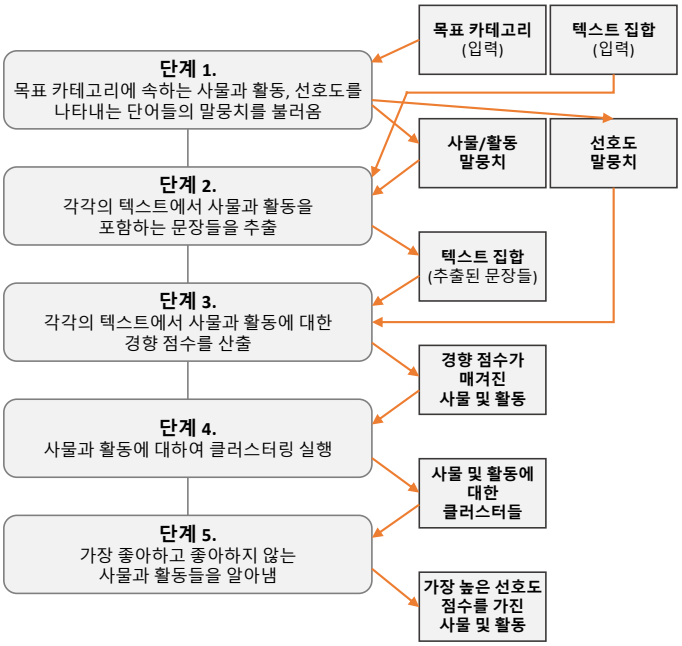


그림 2. 선호 경향 분석의 절차

5. 사례 연구 및 평가

5.1. 스마트 다이어리 시스템 (Smart Diary System)

일기는 사용자의 패턴, 경향 등을 분석하는데 유용한 정보를 포함하기 때문에, 스마트 다이어리 시스템을 통해 녹음된 오디오 일기를 텍스트로 변환하고, 이를 이용한 선호 경향 분석의 사례 연구를 보여준다.

5.2. 실험 결과

실험에서 사용할 분석 카테고리는 음식, 취미, 스포츠로 선정하였고, 3개의 일기 집합을 선정하였다. 각 일기 집합은 주기적으로 작성되었으며, 약 50개의 일기를 포함하고 있다.

선호 경향 분석의 최종 결과는 사용자가 가장 좋아하거나 좋아하지 않는 경향이 있는 사물과 활동을 알아낸 것이다. 그림 3은 A 일기 집합의 선호 경향 분석 결과를 보여준다. 결과를 통해, A 일기의 사용자는 음식에서는 삼겹살을 매우 좋아하였고, 취미로는 드라이빙을 꽤 좋아하는 편이며, 운동은 서핑을 별로 좋아하지 않았다는 경향을 유추해 낼 수 있었다.

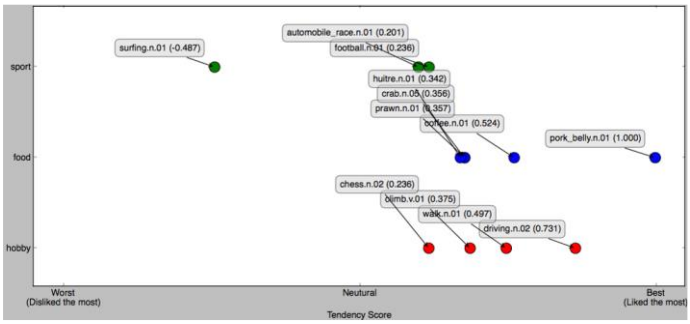


그림 3. A 일기 집합의 선호 경향 분석 결과

5.3. 알고리즘 평가 결과

제안하는 알고리즘은 두가지 방법을 통해서 평가하였다. 첫째로, 최종 분석 결과와 중간 결과, 분석에 사용된 일기 집합의 내용을 직접 비교 분석하였다. 그 결과, 모든 사물과 활동이 제대로 식별되었고, 선호 경향 분석의 결과는 타당하고 신뢰성이 있었다. 둘째로, 선호 경향 분석에 적용가능한 다른 두 개의 방법론들과 비교해 보았다. 이를 통해 본 논문의 접근법이 더욱 정확하고 신뢰할수 있는 결과를 도출한다는 것을 확인할 수 있었다.

6. 결 론

본 논문에서는 오디오 컨텍스트의 라이프 시맨틱 분석 알고리즘을 제안한다. 구현한 후 실험 및 평가를 통해 알고리즘의 타당성을 증명하였고, 높은 신뢰성의 결과를 도출 할 수 있다는 것을 보여주었다.

7. Acknowledgement

이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (2015R1A2A2A01004078).

8. 참고문헌

[1] L. Jose, et. al, “A semantic graph based approach on interest extraction from user generated texts in social media,” *International Conference on Data Mining and Advanced Computing (SAPIENCE)*, pp. 101-104, 2016.

[2] Z. Jin, et. al, “Combining user-based and global lexicon features for sentiment analysis in twitter,” *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 4525-4532, 2016.

[3] A. Makazhanov, et. al, “Predicting political preference of Twitter users,” *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, pp. 298-305, 2013.

JAVA 오픈 소스의 재사용성 및 확장성 향상을 위한 소프트웨어 설계 패턴 추출 기법

최용석 ○ 김두환 홍장의
충북대학교 컴퓨터과학과
{yschoi, dhkim}@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr

An Extraction Technique of Design Pattern for Enhancing Reusability and Extendibility of JAVA Open sources

YONGSEOK CHOI○, Doohwan Kim, Jang-Eui Hong
Department of Computer Science, Chungbuk National University

요 약

시간이 흐를수록 우리 생활에서 소프트웨어가 차지하는 비율은 높아질 것이고 그에 따른 소프트웨어의 수요 또한 폭발적으로 증가할 것이다. 또한 소프트웨어 개발자는 증가하는 사용자의 요구에 정확하고 신속하게 대응하기 위하여 다양한 방법을 모색할 것이다. 이러한 방법의 하나로써, 오픈 소스를 이용한 소프트웨어 개발 접근 방법이 있으며, 이는 적은 비용과 신속한 개발이 가능하다는 장점을 제공한다. 그러나 보통 커뮤니티 기반의 오픈 소스들은 코드에 대한 문서화가 되어 있지 않아 재사용하기 어려운 문제가 있다. 본 논문에서는 오픈 소스를 이용하여 소프트웨어 시스템을 개발한다고 할 때, 설계 과정에서 어떻게 오픈 소스에 대한 설계 정보를 활용할 것인가에 대하여 연구하였다. 특히 문서화가 되어 있지 않은 JAVA 소스로부터 설계정보를 추출하고, 이를 설계 패턴과 매핑하는 시도를 수행하였다. 제시하는 결과는 패턴 기반의 소프트웨어 개발에서 오픈 소스를 적극적으로 활용할 수 기회를 제공할 것이며, 이를 통해 오픈 소스의 재사용성과 확장성이 증대될 것이다.

키워드 : 오픈 소스, 재사용성, 확장성, 설계 패턴

1. 서 론

오늘날 우리는 생활 속에서 수많은 소프트웨어 시스템과 함께 살아가고 있다. 일상에서 매일 접하는 스마트폰이나 데스크 탑을 비롯해서 알게 모르게 주변에서 사용되는 임베디드 시스템까지 그 이용되는 분야가 우리가 생각하는 것 보다 훨씬 더 넓고 깊숙이 분포 되어있다. 또한 앞으로도 우리 생활에서 더 많은 응용 소프트웨어들이 지속적으로 사용될 것이며, 그 역할 측면에서 중요성도 높아질 것이다. 때문에 앞으로 응용 소프트웨어의 수요는 폭발적으로 증가할 것이며 그에 따른 개발의 효율성 증대 요구도 날로 높아질 것이다.

이런 폭증하는 소프트웨어에 대한 수요를 감당하는 하나의 방법으로 오픈 소스를 이용한 소프트웨어 개발 방법이 있다. 오픈 소스 제공 사이트나 신뢰할 수 있는 Community를 통해 오픈 소스를 쉽게 얻을 수 있다. 오픈 소스는 별도의 비용이 들지 않고 자유롭게 사용할 수 있는 장점이 있다. 하지만 오픈 소스 사용에 꼭 장점만 있는 것은 아니다. 기존의 오픈 소스들의 일부가 체계적인 문서화를 제공하고 있지만, 이들의 경우는 비용을 지불하거나 사용상 제약을 갖는 경우가 많다. 그러나 오픈 소스 활용에 대한 생각이 일반화되면서, 많은 개발자들이 자신이 개발한 소스를 공개하게 되었고, 다양한 Community를 통해 제공되고 있다. 이러한 형태의 공개된 소스들은 대부분 문서화가 되어있지 않기 때문에 오픈 소스를 그대로 가져다 쓸 경우, 코드에 대한 이해가 어렵거나, 개발하려는 응용 프로그램에 맞춰 많은 수정이 필요한 경우가 발생할 수 있다. 때문에 오픈 소스를 사용할 경우 원시코드의 역공학을 통해 문서화 하는 것을 고려할 필요가 있다.

본 논문에서는 오픈 소스를 이용하여 소프트웨어 시스템을 개발할 때, 설계 과정에서 어떻게 오픈 소스에 대한 설계 정보를 활용할 것인가에 대하여 연구하였다. 특히 문서화가 되어 있지 않은 JAVA 소스로부터 설계 정보를 추출하고, 이를 설계 패턴과 매핑하는 시도를 수행하였다. 특히 이러한 과정에서 JAVA 오픈 소스의 재사용성과 확장성을 위해 오픈 소스에서 설계 패턴을 추출하는 방법을 제시한다. Doxygen[1] 툴을 사용하여 원시코드를 분석하고 해당 오픈 소스에 맞는 설계 패턴을 제시하는 것이다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 설계 패턴 추출과 관련된 기존 연구를 조사하고, 3장에서는 Doxygen을 이용해 오픈 소스의 설계 정보 추출과 명세 그리고 그것을 이용한 설계 패턴의 식별 과정을 다루고, 4장에서는 실례를 들어 어떻게 오픈 소스의 설계 정보가 추출되고 명세화 되는지 또한 설계 패턴이 식별되는지 설명한다. 그리고 5장에서는 그 식별된 설계 패턴을 활용하는 방안을 설명한다.

마지막으로 6장에서는 본 논문의 결론 및 향후 연구 방향을 제시한다.

2. 관련연구

2.1 역공학을 통한 설계 패턴 추출

역공학은 물리적이고 한정적으로 표현된 정보들을 좀 더 일반적이고 추상화된 정보로 변화시켜 추후 그것들의 재사용을 위해 공유된 저장소에 일정한 형태로서 저장한다. 그러므로 역공학을 이용하면 시스템 전체의 일반적 설계 구조를 추출해 시스템 분석과 설계의 생산성을 높일 수 있는데 다시 말해서 품질을 향상시키고 컴포넌트간의 구조와 설계 용어의 표준화를 통해 재사용성을 높여주는 설계 패턴을 추출할 수 있는 것이다. 설계 패턴을 역공학적 관점으로 보면 기존 시스템의 설계 과정에 대해 공유될 수 있는 공통된 용어를 나타내고 특정한 문제의 해결 방안과 그 응용에 대한 상반 관계를 제시해 줌으로써 기존 설계의 이해성 및 재사용성을 향상시켜준다. 또한 객체지향 프레임워크의 문서화 도구로서 활용되어 비슷한 구조의 새로운 시스템을 구축 할 때 큰 도움을 준다. 즉 설계 패턴의 역공학을 통해 시스템을 보다 쉽게 이해할 수 있음은 물론이고 객체지향 시스템을 (재)개발할 때 소프트웨어의 재사용성의 효과를 향상시키는 방향으로 적용시킬 수 있다[2].

2.2 역공학을 이용한 설계 정보 추출 연구

현재 소프트웨어 산업은 새로운 응용 시스템 개발에 집중하고 품질 향상과 시장 출하 시간의 단축을 통한 사용자의 욕구 충족 및 새롭고 효율적인 개발 프로세스를 만드는데 치중하고 있다. 반면 개발된 소프트웨어에 대해서는 체계적인 유지보수가 미흡하여 유지보수 방법론의 필요성이 제기되고 있다. 기업들은 새로운 여러 요구 사항들을 시스템에 반영하기 바라며 이를 위하여 정확한 분석 작업을 필요해 한다. 특히 구조화 되지 못하고 문서화되지 못한 오래된 시스템을 교체 작업 및 수작업으로 유지 보수 한다는 것은 매우 어려운 일일 것이다[3, 4].

역공학 방법으로 이러한 소프트웨어의 유지 보수성을 향상 시킬 수 있는데, Cho[5]는 이러한 필요성을 충족하고 문제점을 해결하기 위한 방안으로 시스템의 원시코드 분석을 위한 역공학 4단계 방법을 제시하였다. 제시된 역공학 프로세스는 이미 개발된 원시코드의 입력물을 받아 우선 수작업을 통하여 기능 측면으로부터 모듈을 분할하고, CASE (Computer-Aided Software Engineering) 도구를 통한 여러가지 산출물을 재사용이 가능하도록 모듈로 생성하여 Repository에 저장함으로써 추후 개발할 때 관련된 업무에서 효율적으로 검색하고 사용할 수 있도록 하는 것에 목적이 있다.

단계별로 보면 1 단계는 원시코들 분석하는 단계로서, 고객으로부터 받은 많은 양의 소스 코드들이 어떤 기능과 어떤 의미가 있는지 McCabe 툴을 이용하여 분석하고, 다음으로 수작업을 통해 단계적으로 분할하는 것이다. 이때 원시파일들은 기능군으로 집단화 한다.

2 단계는 1 단계의 산출물을 분석하는 단계로서, 기능 군으로 집단화된 입력 원시코드를 다시 분석하여 사용자가 필요로 하는 분석 산출물을 출력한다. 또한 3 단계는 모듈의 인터페이스를 파악하는 단계로서 분석 산출물들을 기반으로 모듈 간의 데이터 입출력 관계와 제어흐름 인터페이스 관계를 분석, 파악하는 단계이다.

마지막으로 4 단계는 비즈니스 모델링을 하는 단계로서, 2, 3단계의 분석 산출물을 기반으로 BP/win을 이용하여 업무분석에 필요한 산출물들을 출력하는 단계이다. 앞 단계의 자동화된 산출물만으로는 시스템의 전체 기능들을 구체적으로 파악하기 어렵기 때문에 앞 단계의 출력물과 원시코드를 검토하여 수동으로 작성한다. 4단계까지 이루어진 후에 재사용 가능한 모듈의 추출 및 생성단계까지 거쳐 Repository에 저장할 수 있는 구조로 모듈을 변환시킨다.

Lee[6]는 구조적 설계 패턴 추출을 위하여 추출 프로세스를 제안하였다. 추출 프로세스는 Java 소스코드를 중간적 언어인 추상객체 언어(Abstract Object Language: AOL)로 변환하는 것과 AOL 기반 패턴 라이브러리를 이용하는 것, 그리고 AOL파서, 클래스 특성 추출기 및 설계 패턴 식별기로 나누어진 설계 패턴 인식기로 구성되어 있다. Java 원시코드를 AOL로 번역하여 설계 패턴 인식기를 통해 클래스와 그 관계 그리고 메소드의 정보를 추출하고 설계 패턴을 추출한다.

Kim[2]은 객체지향 시스템의 기존 원시코드를 대상으로 역공학을 적용하여 정규화하고 패키지화 하여 정량적으로 평가할 수 있는 설계 패턴을 추출할 수 있는 알고리즘을 정의 하였고 Java 시스템을 이용한 설계 패턴의 자동 추출을 위한 역공학 도구와 추출 패턴의 재사용 도구를 포함하는 재공학 도구를 설계하였다.

이 외에도 기존 여러 연구[7,8,9]에서 설계 패턴의 추출을 위한 도구(툴)와 특정언어의 사용 그리고 특정 알고리즘을 통한 역공학 프로세스의 개선을 제안하고 있지만 설계 패턴의 식별 및 분류 과정에서 만날 수 있는 문제는 고려하지 않았다. 본 논문에서는 이 부분을 고려하여 설계 패턴의 식별 및 분류 과정에서 만날 수 있는 세 경우를 생각해 보았는데, 기존 패턴과 일치하는 경우와 기존 패턴의 부분집합인 경우 그리고 기존 패턴과 일치하는 것이 없는 경우이다. 이 세 경우를 바탕으로 설계 패턴을 식별 및 분류함으로써 역공학 프로세스의 효율성 증대를 모색한다.

3. 설계 정보 추출 및 설계 패턴 명세화

본 절에서는 오픈 소스로부터 설계 정보를 추출하고, 이를 기반으로 패턴과 매칭시키는 절차를 Fig. 1과 같이 제시하고 각 단계별로 설명한다.

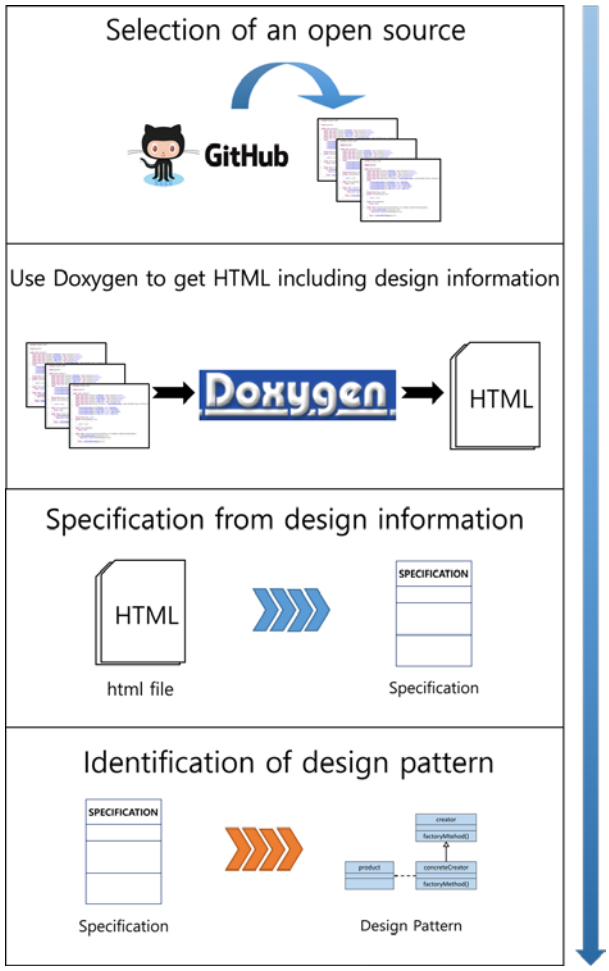


Fig. 1 Identification Strategy of Design Pattern

3.1 Selection of Open Source

기존에 존재하는 소스코드를 이용하여 소프트웨어를 개발한다면 재사용성 혹은 확장성을 향상시킬 수 있다. 이미 만들어진 소스코드를 볼 때 대표적으로 상용 프로그램이나 오픈 소스가 있는데 사용 프로그램은 비용적인 문제가 따른다. 후자의 오픈 소스의 경우 비용적인 문제에서 자유롭고 가져다 재사용하는데 어떠한 제약도 따르지 않는 편리함이 있다. 하지만 오픈 소스를 재사용하는 것에도 문제가 있는데 바로 문서화가 되지 않은 소스코드가 대부분이라는 것이다. 문서화가 되어있지 않다면 그 오픈 소스의 용도 등 소스코드를 이해하는데 어려움을 겪는다. 대표적인 오픈 소스 사이트가 ComponentSource[10], sourceforge [11] 그리고 github[12]가 있는데 많은 오픈 소스들이 문서화 되어있지만, 그렇지 않은 소스들도 상당수 차지하고 있다.

이런 경우 오픈 소스를 이해하기 위해 역공학을 통해 적절히 문서화 해줄 필요가 있다. 따라서 우선 이번 단계에서는 임의의 오픈 소스를 위에서 말했던 오픈 소스 사이트나 또 다른 신뢰할 수 있는 Community를 통해 얻는다.

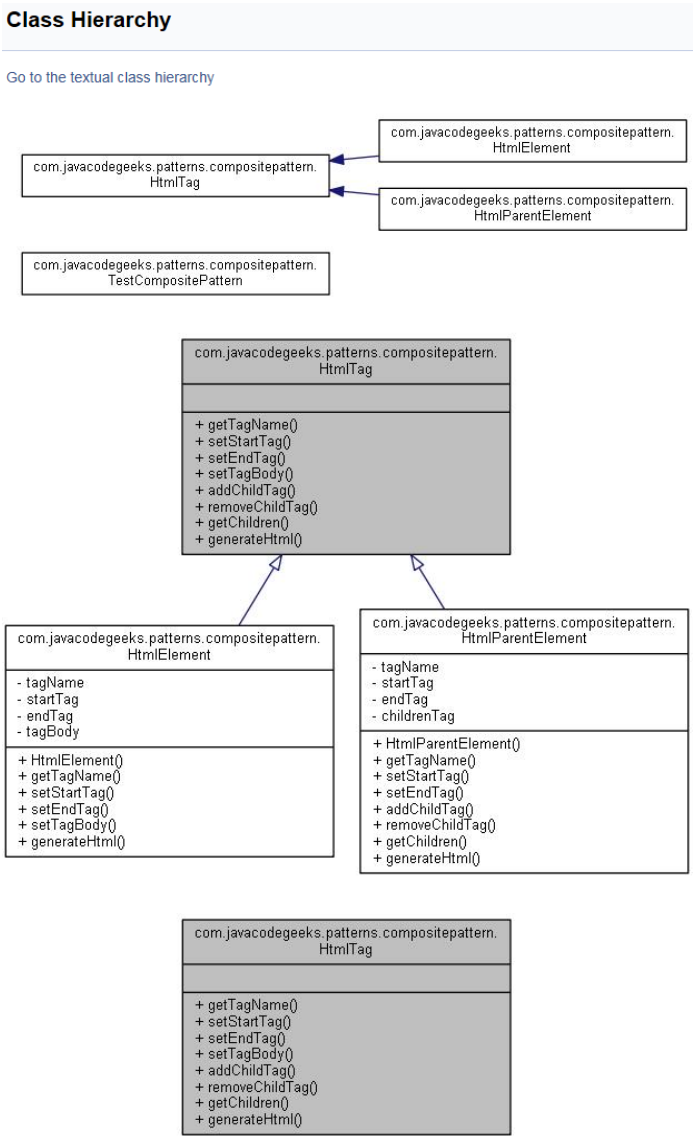


Fig. 2 Design Information from open source

3.2 Use Doxygen to get HTML including Design Information

선택된 오픈 소스를 문서화하기 위해 대상 오픈 소스로부터 설계 정보를 얻어야 하는데 그것을 위해 Doxygen을 사용한다. Doxygen은 주석이 달린 코드를 읽어내서 문서를 만들어 내는 소프트웨어 레퍼런스 문서 생성기로서 보통 HTML파일이나 설정에 따라 Work 파일로 결과를 추출할 수 있다. 코드 내의 주석으로부터 정보를 읽어오며 주석만 일정 규칙을 지키면 모든 언어

로부터 정보를 읽어 올 수 있다.

Doxygen을 사용해서 설계 정보를 얻기 위해 대상 소스파일을 선택한 후 Class Hierarchy, Function Call 등 보고자 하는 설계 정보를 설정 한 후 프로그램을 실행시키면 Fig. 2와 같은 결과를 얻을 수 있다.
하지만 Doxygen을 이용해서 얻는 설계 정보는 재사용하기에는 충분한 정보를 제공하지 못한다. Graphic 요소들이 주를 이루기 때문인데 추가적인 설계 정보는 오픈 소스의 사용자가 직접 코드의 주석이나 매뉴얼 등을 보면서 추출해야 한다.

3.3 Specification of Design Information

Doxygen을 사용해서 얻어진 HTML의 파일을 통해 Class Diagram과 Class Hierarchy 등과 같은 설계 정보를 확인 할 수 있고, 오픈 소스 사용자의 수작업을 통한 설계 정보 또한 확인 할 수 있다. 이제 확인된 설계 정보를 가지고 설계에 대한 정보를 요약하여 정리 한다.

설계 패턴을 명세화 하는데 어떤 특별한 표준 형식은 없다. 오히려 다양한 패턴 작성자가 다양한 형식을 사용하여 패턴을 정의한다. 그러나 Martin Fowler[13]에 따르면 특정 패턴 유형이 다른 패턴 유형보다 더 잘 알려졌고 그 결과, 새로운 패턴 표출의 공통 출발점이 됐다. 일반적으로 사용되는 문서형식의 일례로 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides ("Gang of Four"또는 줄여서 GoF)가 사용한 설계 패턴의 명세 형식이 있다[14].

본 논문에서는 이전 단계에서 얻어진 설계 정보를 정리하기 위해 GoF에서 사용한 명세 방법을 이용한다. 이의 결과는 Table 1과 같이 정리하였다.

3.4 Identification of Design Pattern

앞에서 정리한 오픈 소스의 설계 정보를 가지고 대상 오픈 소스가 어떤 설계 패턴을 가지고 있는지 식별 및 분류한다. 그 방법으로 GoF 설계 패턴을 가지고 비교하여 판단한다. 또한 오픈 소스가 가지고 있는 설계 패턴이 기존에 있던 패턴과 다르거나 기존 패턴의 부분집합일 수 있으므로 오픈 소스의 설계 패턴을 식별하고 분류하는데 아래와 같이 세가지 경우에 따라 다르게 한다.

- 기존의 설계 패턴과 일치하는 경우
- 기존 설계 패턴의 부분집합인 경우
- 기존 설계 패턴에 존재하지 않는 경우

우선 가장 간단한 식별 및 분류 방법으로 기존 GoF 설계 패턴과 일치하는 경우에 선택된 오픈 소스를 GoF의 일치된 패턴으로 식별 및 분류한다. GoF에서 정의하는 설계 정보와 오픈 소스를 통해 얻은 설계 정보를 단순 비교하여 일치하는 설계 패턴을 찾는다.

이러한 경우에 대해서는 해당 GoF 패턴이 개발 대상 소프트웨어의 설계 모델로 직접 활용이 가능하다. 따라서 해당 오픈 소스의 명세에 매핑된 GoF 패턴이 적용되었음을 명시하고 추후 코드 생성에서 해당 패턴에 대응되는 오픈 소스를 큰 변경 없이 재사용할 수 있다.

두 번째로 이전 단계에서 명세한 오픈 소스의 설계정보가 기존 GoF 설계 패턴의 부분집합인 경우이다. 이에 대한 처리 방안은 두 가지 경우로 나누어 생각할 수

Table 1. Design Information on Source File

SECTION	DESIGN INFROMATION
Pattern Name and Classification	• 패턴을 식별하고 설명하는데 도움이 되는 서술적이고 고유한 이름
Intent	• 패턴의 숨겨진 목표와 그것을 사용하는 이유를 설명
Also Known As	• 패턴의 다른 이름
Motivation (Forces)	• 이 패턴이 사용될 수 있는 상황으로 구성된 시나리오
Applicability	• 이 패턴이 사용 가능한 상황 또는 패턴의 문맥
Structure	• 패턴의 그래픽 표현 • 클래스 다이어그램과 상호 작용 다이어그램의 사용 될 수 있음
Participants	• 패턴에서 사용되는 클래스와 객체의 목록과 그 디자인의 역할
Consequences	• 패턴의 사용에 의한 결과, 부작용 그리고 장단점에 대한 설명
Implementation	• 패턴 구현의 설명 • 패턴의 해석 부분
Sample Code	• 프로그래밍 언어로 패턴이 어떻게 사용되는지에 대한 실례
Known Uses	• 패턴의 실제 용도의 예
Related Patterns	• 패턴과 어떤 관계를 가지는 다른 패턴 • 패턴과 유사한 패턴의 차이에 대한 논의

있다. 먼저 추출된 설계 정보의 부분 집합을 보완해서 완전하게 만든 후 GoF 설계 패턴 중 하나로 식별 및 분류하는 것이다. 일반적으로 설계 패턴의 경우는 보다 안정된 소프트웨어 시스템을 위해 권한의 분리, 클래스의 통합, 중간자의 추가 등과 같은 형태로 패턴화 되기 때문에 추출된 설계 정보에 대해서도 이러한 과정을 적용하여 온전한 패턴 형태로 수정한다.

다음의 처리 방법은 다른 오픈 소스와 통합해서 하나의 큰 오픈 소스를 생성하고, 이에 대한 설계 정보 추출을 통해 GoF 설계 패턴과 비교, 매핑하는 경우이다. 이 경우 다양한 조건을 고려할 수 있지만, 분명한 사실은 추출된 설계정보가 기존의 설계 패턴보다 더 많은 정보를 포함한다면 소프트웨어 개발에서 기존 패턴을 사용할 수 있다는 것이다.

마지막 세 번째로 오픈 소스의 설계 정보를 가지고 일치하는 설계 패턴을 찾을 수 없는 경우이다. 이 경우에는 단순하게 해당 오픈 소스는 설계 패턴 추출을 통한 재사용이 불가능하다고 판단하거나 아니면 전혀 새로운 설계 패턴으로 인식하고 그 설계 패턴을 새롭게 정의하여 분류 한다. 또한 해당 오픈 소스에 리팩토링을 적용시킴으로써 소스코드 자체를 재구조화 하여 GoF 설계 패턴 중 하나로 식별 및 분류하는 방법도 있다.

4. 사례연구

4절에서는 실제 오픈 소스 사이트로부터 확보한 예제 코드를 이용하여 설계 패턴을 추출하는 과정을 설명한다.

4.1 기존 설계 패턴과 일치하는 경우

단계 1) 오픈 소스 사이트를 통해 하나의 JAVA 오픈 소스 샘플 “Printer.java” 파일을 구한다.

단계 2) Doxygen을 이용해서 샘플 오픈 소스를 분석한다. 해당 샘플을 Doxygen을 통해 돌리면 HTML 파일을 얻을 수 있고 Fig. 3와 같은 Graphic 요소를 확인할 수 있다.
하지만 Graphic 요소만 가지고는 설계 정보를 충분히 얻을 수 없기 때문에 샘플 코드를 직접 보고 수작업으로 분석한다.

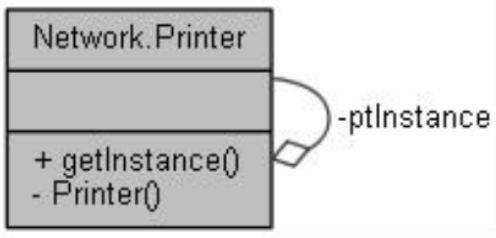


Fig. 3 Design Information: Collaboration diagram on “Printer.java”

단계 3) 앞의 단계 2)에서 얻은 HTML 파일의 Graphic 요소들과 수작업으로 분석한 샘플 코드의 설계 정보를 정리하여 Table 2와 같이 명세한다.

Table 2. Design Information on Source “Printer.java”

SECTION	DESIGN INFROMATION
Pattern Name and Classification	Singleton Pattern / Creation
Intent	하나의 객체만 존재하도록 보장한다
Also Known As	N/A
Motivation (Forces)	오직 하나의 객체만을 이용하여 작업을 한다
Applicability	하나의 객체만을 이용하는 분야 예) 프린터
Structure	하나의 Printer 클래스가 존재하며 그 자신이 전체의 부분이 되는 형태 (Fig. 3)
Participants	Printer 클래스: 최초 생성된 하나의 객체를 계속해서 사용
Consequences	N/A
Implementation	Private 생성자 Private static 필드: 자신의 생성자로 초기화 public static 메소드: 초기화한 필드 반환
Sample Code	N/A
Known Uses	N/A
Related Patterns	N/A

단계 4) 명세한 설계 정보를 가지고 GoF의 설계 패턴과 비교한다. 단계 3)에서 명세한 설계 정보 중 Structure와 Intent section을 보면 바로 알 수 있듯이 이 샘플은 Singleton Pattern이 적용되어 있다. 따라서 이 샘플 오픈 소스는 Singleton Pattern으로 구별한다.

4.2 기존 설계 패턴에 존재하지 않는 경우 (I)

단계 1) 오픈 소스 사이트를 통해 Fig. 4와 같은 JAVA 오픈 소스 샘플 “Shape.java” 파일을 구한다.

```
public Shape(int typecode, int startx, int starty, int endx, int endy){
  _typecode = typecode;
  _startx = startx;
  _endx = endx;
  _starty = starty;
  _endy = endy;
}
```

Fig. 4 A part of Source Code in “Shape.java”

단계 2에서 Doxygen을 이용한 산출물은 Fig. 5와 같

으며, 이를 기반으로 수작업 과정을 거쳐 확보한 단계 3의 설계 정보는 Table 3과 같다.

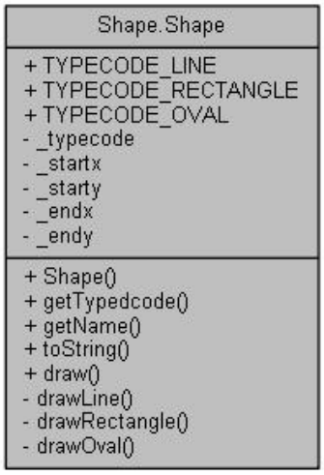


Fig. 5 Design Information: Collaboration diagram on "Shape.java"

Table 3. Design Information on Source "Shape.java"

SECTION	DESIGN INFROMATION
Pattern Name and Classification	N/A
Intent	N/A
Also Known As	N/A
Motivation (Forces)	N/A
Applicability	직선, 사각형, 원을 이용한 그래픽 분야
Structure	Shape 하나의 클래스가 존재한다 (Fig. 5)
Participants	Shape 클래스: 정해진 도형 표시
Consequences	N/A
Implementation	일반적인 클래스 형태
Sample Code	Fig. 4
Known Uses	N/A
Related Patterns	N/A

다음 단계에서는 명세된 설계 정보를 이용하여 GoF의 설계 패턴과 비교한다. 그러나 이 경우에 있어서는 기존 GoF 설계 패턴과 매핑되는 패턴을 찾을 수 없다. 그러나 해당 소스 코드를 재구조화(리팩토링)한다면 기존에 존재하는 GoF 설계 패턴으로 만들 수 있다. 리팩토링은 먼저 생성자를 호출하는 create라는 메소드를 만들고 이 create 메소드 안에서 기존의 생성자를 호출하도록 한다. 그리고 클라이언트에서 생성자를 호출 하는 곳을 방금 생성한 create 메소드의 호출로 수정한다.

리팩토링으로 재구조화를 완료한 후 적용된 샘플 JAVA 파일의 설계 정보를 통해 얻은 Fig. 6 – 7과 같은 모델과 또한 소스 코드를 직접 수작업으로 분석한 결과로 Table 4와 같은 설계정보를 얻을 수 있다.

이 설계 정보를 통해 Structure와 Implementation을 확인해보면 이 리팩토링이 적용된 새로운 소스 코드가 GoF의 Factory Method Pattern에 해당하는 것을 알 수 있다. 따라서 재구조화 된 이 샘플 오픈 소스를 Factory Method Pattern으로 구별한다.

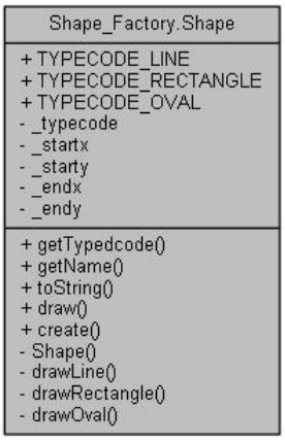


Fig. 6 Design Information: Collaboration diagram on "Shape_Factory.java"

Fig. 7 Design Information: Caller graph for create() function on "Shape_Factory.java"

Table 4. Design Information on Source "Shape_Factory.java"

SECTION	DESIGN INFROMATION
Pattern Name and Classification	Factory Method Pattern / Creation
Intent	특정한 메소드를 만들어 생성자를 대신 호출한다
Also Known As	N/A
Motivation (Forces)	여러 종류의 객체를 만드는데 특정한 메소드를 이용해서 만든다
Applicability	하나의 인터페이스로 여러 객체를 만들어 낼 때 예) 게임에서 한 건물을 통해 여러 유닛을 생성
Structure	하나의 Shape_Factory 클래스가 존재하고 클래스 안의 create() 메소드가 생성자를 직접 사용한다 (Fig. 6 – 7)
Participants	Shape_Factory 클래스: 내부의 메소드를 통해 생성자 역할을 대신 한다
Consequences	N/A
Implementation	Static 메소드: 객체를 대리 생산
Sample Code	N/A
Known Uses	N/A
Related Patterns	N/A

4.3 기존 설계 패턴에 존재하지 않는 경우 (II)

4.2절에서 설명한 것과 같이 기존에 설계 패턴이 존재하지 않는 경우 재구조화를 통화 설계 패턴과 매핑시킬 수 있지만 그렇지 않은 경우도 발생한다.

오픈 소스 사이트를 통해 하나의 JAVA 오픈 소스 샘플 “Book.java” 파일을 구하고, Doxygen을 이용해서 샘플 오픈 소스를 분석하면, Fig. 8와 같은 모델을 얻을 수 있다. 또한 앞서 설명한 바와 같이 수작업을 통해 소스 코드를 분석하여 좀 더 상세한 설계 정보를 Table 5와 같이 명세할 수 있다.

그러나 명세한 설계 정보를 가지고 GoF의 설계 패턴과 비교하여 특정 패턴을 식별하고자 할 때, 기존 GoF의 설계 패턴과 일치하는 것이 없으므로 이 샘플 오픈 소스를 특정한 설계 패턴으로 구별하지 않는다.

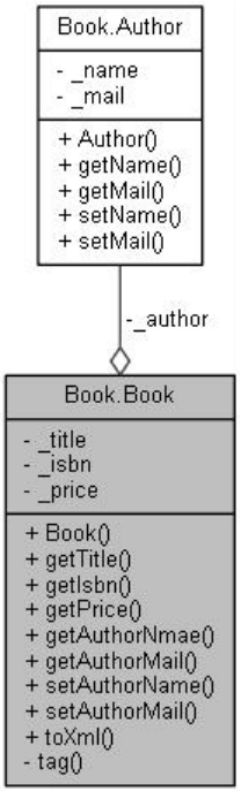


Fig. 8 Design Information: Collaboration diagram on “Book.java”

5. 설계 명세 활용 방안

Fig. 9는 일반적인 소프트웨어 개발 프로세스를 나타낸다. 소프트웨어를 개발 할 때는 보통 상세 설계 전 요구사항을 제대로 구현하기 위해 소프트웨어 아키텍처를 정의 하는데 소프트웨어 아키텍처는 시스템의 요약된 형태로서 향후 상세한 시스템 구성의 기본을 제공하고 시스템 분석, 설계 시의 초기 결정사항을 제공한다.

이에 본 장에서는 정의된 소프트웨어 아키텍처를 상세화 하는 상세 설계 단계에서, 3장에서 제시한 절차를 통해 얻은 추출된 설계 패턴의 활용 방안을 제시한다. 소프트웨어 아키텍처가 정해진 후에는 상세화를 진행할 것이고 그 과정에서 적절한 내부 기능들로 분할 될 것이다. 그리고 이 때 분할 된 기능들이 일정한 패턴을 보일 수 있는데 바로 이 패턴들과 일치하는, 3장에서 추출된 설계 패턴을 식별해서 상세 설계 과정에 직접 사용하는 것이다. 즉 상세 설계 과정에서 오픈 소스의 역공학을 통해 도출한 설계 패턴을 활용함으로써, 결국 구현과정에서 JAVA 오픈 소스를 그대로 사용 하는 것이다. 예를 들어 MVC 아키텍처를 갖는 시스템을 개발할 때 시스템의 상세 설계 과정에서 MVC의 Model 컴포넌트는 Strategy Pattern을 활용하고 View 컴포넌트는 Composite Pattern을 활용하는 것이다. 활용하는 설계 패턴에 맞춰 그 패턴을 가진 JAVA 오픈 소스를 사용해 시스템을 구성하는 것이다.

Table 5. Design Information on Source “Book.java”

SECTION	DESIGN INFROMATION
Pattern Name and Classification	N/A
Intent	N/A
Also Known As	N/A
Motivation (Forces)	N/A
Applicability	책의 정보를 이용하는 분야
Structure	Book 클래스와 Author 클래스가 집합 관계이다 (Fig. 8)
Participants	Book 클래스: 책의 정보 Author 클래스: 저자의 정보
Consequences	N/A
Implementation	일반적인 클래스 형태
Sample Code	N/A
Known Uses	N/A
Related Patterns	N/A

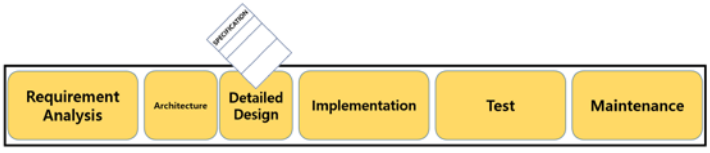


Fig. 9 Application of design information into detailed design

위와 같이 소프트웨어 개발의 상세 설계 부분에서 추출된 설계 패턴을 활용함으로써 오픈 소스의 재사용성을 높일 수 있으며 더불어 오픈 소스를 패턴화 시킴으로써 코드의 확장성도 높일 수가 있다.

6. 결론

본 논문에서는 범용성이 큰 JAVA 오픈 소스를 가지

고 재사용성과 확장성 향상을 위한 설계 패턴 추출 방법을 제시하였다. 네 단계로 이루어진 추출 방법으로서 신뢰할 수 있는 오픈 소스 사이트나 Community를 통해서 얻은 JAVA 오픈 소스를 Doxygen 이라는 툴과 사용자의 수작업으로 설계 정보를 얻은 후, 일정한 정보를 확보하기 위한 정해진 양식으로 이용하여 명세하였고, 이를 이용하여 GoF 설계 패턴과 비교함으로써, 해당 오픈 소스의 설계 패턴을 식별할 수 있었다. 또한 그 식별된 설계 패턴을 소프트웨어 개발과정에서 언제 어떻게 활용할 것인지에 대해서도 간략히 제시하였다.

향후에는 JAVA 오픈 소스의 재사용성 및 확장성을 보다 더 증진시키기 위한 방법으로 소프트웨어 개발의 전체 수명주기와 연결하는 것이다. 요구사항의 명세와 오픈 소스, 그리고 분석과 오픈 소스를 연계함으로써, 요구사항 기반의 자동 프로그램 생성이 가능할 수 있을 것이다.

Acknowledgments

This research was supported by the NRF funded by the MSIP (NRF-2014M3C4A7030505) and was also supported by the NRF funded by the MOE (NRF-2014R1A1A4A01005566).

7. 참고문헌

[1] Doxygen, http://www.doxygen.org, at Oct. 2016
[2] Guk-Boh Kim, “Reengineering Legacy systems into Design Patterns of Component Base Design (CBD)”, Journal of Internet Computing and Services, Vol. 5, No. 1, February 2004.
[3] Harald Gall and Rene Klsh, "Balancing in Reverse Engineering and in Object-Oriented Systems Engineering to Improve Reusability and Maintainability", Vienna University of Technology Institute of Information Systems, 1994.
[4] Yih-Farn Chen, Michael Y. Nishimoto, and C. V. Ramamoorthy, "The C Information Abstraction System", IEEE Trans. of Software Engineering, Vol.16, No. 3, March 1990.
[5] Hyun Hoon Cho, Yong Lak Choi and Sung Yul Rhew, “Case Study of Software Reverse Engineering using McCabe and BP / Win Tools”, Journal of KIISE : Computing Practices and Letters, Vol. 6, No. 5, October 2000.
[6] 이창목, 이정열, 김정옥, 유철중, 장옥배, “레거시 Java Code 로부터 디자인 패턴 추출을 위한 AOL 기반 프로세스”, 한국정보과학회 학술발표논문집, 29 권 2 호, 2002, 10.
[7] Kamran Sartipi, “Software architecture recovery based on pattern matching”, International Conference on Software Maintenance (ICSM), 2003.
[8] P. Tonella and G. Antoniol, “Inference of Object-Oriented Design Patterns,” J. Software Maintenance: Research and Practice, vol. 13, no. 5, pp.309-330, Sept.-Oct. 2000.
[9] Y. Cai, H. Wong, S. Wong, and L. Wang. "Leveraging design rules to improve software architecture recovery" in Proc. 9th International ACM Sigsoft Conference on the Quality of Software Architectures, pp.133-142, June 2013.
[10] The Software Superstore for Developers & IT Pors, https://www.componentsource.com at Dec. 2016
[11] Find, Create, and Publish Open Source Software for Free, https://sourceforge.net at Dec. 2016
[12] How people build software, https://github.com at Dec. 2016
[13] Martin Fowler, "Writing Software Patterns", e-pub (Aug., 2006), https://www.martinfowler.com/articles/writingPatterns.html
[14] E.Gamma, R.Helm, R.Johnson, and J.Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

디지털 건강 분석 솔루션을 위한 소프트웨어 플랫폼 설계

라현정 [1,O], 정한터 [2], 임성민 [2], 김수동 [3,*]

(주)스마티랩 [1], 숭실대학교 컴퓨터학과 [2], 숭실대학교 소프트웨어학부 [3]

{hjla80, hanterkr, hcooch2ch3, sdkim777}@gmail.com

Software Platform Design for Digital Health Analytics Solutions

Hyun Jung La[1, O], Han Ter Jung[2], Seong Min Yim[2], Soo Dong Kim[3,*]

SmartyLab Corporation[1], Department of Computer Science, Soongsil University[2], School of Software, Soongsil University[3]

요 약

디지털 헬스케어 서비스 분야는 사물인터넷 (Internet-of-Things, IoT) 컴퓨팅 개념을 적용하는 핵심 도메인으로 대두되고 있다. 다양한 의료 IoT 디바이스들의 이용 가능해짐에 따라, 이 디바이스로부터 수집된 건강 데이터를 활용하여 건강 상태를 알려주는 여러 종류의 건강 분석 애플리케이션이 개발되고 있다. 건강 분석 애플리케이션들이 공통으로 필요로 하는 기능성을 제공하는 소프트웨어 플랫폼이 개발된다면 여러 종류의 건강 분석 애플리케이션은 적은 개발 노력 및 비용으로 개발될 수 있게 된다. 그러나, 애플리케이션에서 수집될 수 있는 건강 데이터의 이질성 문제, 애플리케이션에서 필요로 하는 건강 분석 결과 종류의 다양성, 건강 분석의 정확도 보장 어려움의 기술적 문제로 이런 플랫폼을 개발하는 것이 어렵다. 그러므로 본 논문에서는 건강 분석 애플리케이션을 위한 디지털 건강 분석 소프트웨어 플랫폼을 제안한다. 제안된 디지털 건강 분석 플랫폼은 애플리케이션으로부터 전달되는 건강 측정치를 저장하고, 애플리케이션의 요청에 따라 최적의 건강 분석 결과를 제공한다. 제시된 플랫폼은 건강 분석 애플리케이션을 적은 개발 노력 및 비용으로 개발할 수 있으며, 광범위한 건강 데이터를 기반으로 계산되기 때문에 평가 결과의 정확도가 향상될 수 있을 것으로 기대된다.

1. 서 론

디지털 헬스케어 서비스 분야는 사물인터넷 (Internet-of-Things, IoT) 컴퓨팅 개념을 적용하는 핵심 도메인으로 대두되고 있다 [1]. 이 트렌드에 맞춰, 혈압, 맥박, 체지방량, EEG, ECG, EMG 등 사람의 여러 신체 정보를 측정하는 다양한 의료 IoT 디바이스들이 출시되고 있다.

다양한 의료 IoT 디바이스들의 이용 가능해짐에 따라 병원에 방문하지 않고 개인 건강 상태를 평가하는 애플리케이션도 개발되고 있다. 대표적인 예로 심장 상태 모니터, 혈당 관리 앱, 혈압 모니터 등이 있다.

건강 분석 애플리케이션은 공통적으로 건강 데이터 측정 및 저장, 건강 데이터를 이용한 개인 건강 상태 분석 기능을 제공한다. 건강 분석 애플리케이션들이 공통으로 필요로 하는 기능성을 제공하는 소프트웨어 플랫폼이 개발된다면 여러 종류의 건강 분석 애플리케이션은 적은 개발 노력 및 비용으로 개발될 수 있게 된다. 그리고, 플랫폼에서 제공되는 건강 분석 결과는 보다 광범위한 건강 데이터를 기반으로 계산되기 때문에, 정확한 건강 상태를 분석할 수 있다.

이런 장점에도 불구하고, 건강 분석 플랫폼을 개발하는 것은 어렵다. 첫째, 건강 분석 애플리케이션들로부터 수집되는 건강 데이터의 종류 및 형식이 다양하기 때문에, 이질적인 데이터를 관리하는

*교신저자 (Corresponding Author)

것이 어렵다. 좀 더 구체적으로, 동일한 종류의 건강 데이터를 측정하는 디바이스이라고 하더라고, 이들이 측정하는 건강 데이터의 형식이 다르다. 그리고, 앞으로 건강 기기 기술 발달로 현재에는 지원하지 않은 다양한 종류의 건강 데이터 측정을 허용할 수 있도록 플랫폼을 설계하는 것이 쉽지 않다.

둘째, 건강 분석 애플리케이션이 공통적으로 분석 기능을 필요로 하더라도, 건강 나이 예측, 질병 진단 등 건강 상태 분석 종류는 다양하다. 그리고, 그 구현 방법 역시 간단한 통계 처리 기법부터 복잡한 머신러닝 기반 추론 기법까지 다양하다. 이런 가변성으로, 소프트웨어 플랫폼이 모든 종류의 건강 분석 기법을 제공하기 어렵다. 이를 위해서는 플랫폼이 건강 분석에 필요한 핵심적인 기능을 제공하고, 애플리케이션에 특화된 건강 분석 기법을 구현한 컴포넌트를 허용할 수 있도록 확장성 있게 설계되는 것이 더 실용적이다. 그러나, 이렇게 다양한 종류의 건강 분석 컴포넌트를 허용하는 확장성 높은 플랫폼을 개발하는데 기술적 난이도가 높다.

그러므로 본 논문에서는 여러 종류의 건강 분석 애플리케이션을 위한 디지털 건강 분석 소프트웨어 플랫폼을 제안한다. 제안된 디지털 건강 분석 플랫폼은 여러 애플리케이션으로부터 전달되는 건강 측정치를 저장하고, 애플리케이션의 요청에 따라 최적의 건강 분석 결과를 제공할 수 있도록 개발된다. 그리고, 다양한 종류의 건강 측정치와 다양한 종류의 건강 분석 컴포넌트를 허용하는 등 앞에서 언급한 기술적 어려움을 해결하도록 설계된다.

본 논문은 구성은 다음과 같다. 2장에서는 센서로부터 수집된 건강 데이터를 이용한 건강 분석 프레임워크에 대한 관련 연구를 정리한다. 3장에서는 제시된 디지털 건강 분석 플랫폼의 기능적 요구사항과 비기능적 요구사항을 정리하고, 4장에서는 이를 만족시키기 위한 설계 모델을 제시한다. 마지막으로 5장에서는 제안된 플랫폼의 적용 가능성을 증명하기 위해 Health Index 애플리케이션 개발에 적용한 결과를 보여준다.

2. 관련연구

Forkan [2]의 연구는 연속적 모니터링을 통해 심장 질환을 조기 발견하는 확장 가능한 컨텍스트 인지 프레임워크를 제안한다. 이 프레임워크는 심장 질환 증상을 확인하기 위해 클라우드 저장소에 저장되어 있는 의사의 지식베이스를 활용한다. 이 연구는 ECG 측정치를 수집하고 심장 질병 발견에 한정되어 있다.

Kuo [3]의 논문에서는 실제 의료 데이터를 통한 빅 데이터 분석(Big Data Analytics; BDA)을 지원하기 위한 프레임워크를 제안한다. 이 프레임워크는 Hadoop을 통해 데이터를 마이그레이션하고 대량의 쿼리를 실행한다. 이 연구는 대용량의 의료 데이터를 저장하고 높은 성능으로 데이터 질의 및 조회에 한정되어 있다.

Sheriff [4]의 논문에서는 의료 기기로부터 나오는 대량의 데이터뿐만 아니라 BDA, IoT, Complex Event Processing (CEP)을 통합하는 레퍼런스 프레임워크를 제안한다. 하지만 이 논문에서는 단지 프레임워크의 구조, 프레임워크에서 제공될 수 있는 분석 기능 종류, 관련 구현 기술을 개략적으로 설명하고 있다.

Sakr [5]의 논문에서는 환자 모니터링 시스템으로부터 수집되는 데이터를 이용한 대규모 의료 데이터 분석 프레임워크인 SmartHealth를 제안한다. 제안된 프레임워크는 데이터 연결 계층, 데이터 저장 및 관리 계층, 분석 계층, 표현 계층으로 이루어진다. 그러나, 본 논문에서는 프레임워크의 개략적인 구조, 필요한 구현 기술, 활용 가능한 애플리케이션 시나리오만을 서술하고 있어, 이를 실제로 구현하는 데에는 한계가 있다.

Arjun [6]의 연구에서는 건강 상태를 모니터링 하는 헬스케어 디바이스를 사용하는 태블릿이나 휴대폰과 같은 스마트 디바이스와 연결할 수 있는 공통 소프트웨어 기반 분석 프레임워크를 제안한다. 이 프레임워크에서는 다양하고 이질적인 센서 데이터를 효과적으로 처리한다. 그러나, 프레임워크의 추상적인 설계 모델만 기술하였고, 제안된 프레임워크는 스마트 디바이스를 통해서만 수집되는 데이터를 관리하며, 분석 종류가 한정적이라는 점에서 한계가 있다.

Soudris [7]의 연구에서는 생체 데이터에 대한 고성능의 BDA를 가능하게 하는 접근법을 제안한다. 이 프레임워크는 생체 데이터 분석에 맞도록 이질적인 고성능 컴퓨팅 기술과 클라우드 컴퓨팅 기술을 적용하여 설계된다. 하지만 이 연구는 이런 특징을 제공하는 프레임워크와 관련된 주요 기술을 명세하고 사례 분석만 제시하고 있다.

이와 같이 대부분의 연구는 건강 분석 프레임워크 또는 플랫폼에 대하여 현재 진행 중이다. 완성된 프레임워크의 경우에도 분석이나 디바이스 연결 면에서

한정적이다. 반면에 본 논문에서는 다양한 디바이스와 분석 컴포넌트를 효과적으로 수용 가능한 플랫폼을 제안한다. 범용성과 확장성을 위한 세부 설계를 보여주고 사례 연구를 통해 실제 플랫폼이 구현되었고 실행됨을 증명 및 평가한다.

3. 플랫폼 요구사항 및 기술적 Challenges

3.1. 기능적 요구사항

제안된 플랫폼은 건강 분석 애플리케이션이 공통으로 필요로 하는 기능을 제공해야 한다. 현재 사용 가능한 애플리케이션 및 여러 관련 문헌에 언급된 시스템 종류를 분석하여, 총 7가지 종류의 기능을 도출하였다.

첫째, 사용자 프로파일 관리 기능을 제공한다. 이는 플랫폼과 상호작용하는 건강 분석 애플리케이션의 최종 사용자에 대한 프로파일을 등록, 수정, 검색, 삭제하는 기능을 의미한다. 건강 상태를 정확히 파악하기 위해서는 사용자의 건강관련 프로파일이 필요하기 때문에, 사용자 프로파일 관리 기능을 제공한다.

둘째, 건강 분석 애플리케이션 메타 정보를 등록, 검색, 수정, 제거하는 기능을 제공한다. 건강 분석 애플리케이션은 본 논문에서 제안하는 플랫폼을 기반으로 개발된 것으로, 건강 분석 애플리케이션 별 사용하는 건강 분석 컴포넌트를 보다 효과적으로 추적 및 관리하기 위해서 이 기능성을 제공한다.

셋째, 애플리케이션으로 수집된 의료 데이터를 저장하거나 요청에 맞게 검색하는 기능을 제공한다. 이 기능은 플랫폼에서 제공되는 주요 기능 중 하나로 사용자의 방대한 의료 데이터를 중앙에서 집중 관리할 수 있게 한다.

넷째, 내장 (Built-in) 분석 컴포넌트와 플러그인 (Plug-in) 분석 컴포넌트를 관리하는 기능을 제공한다. 내장 분석 컴포넌트는 여러 애플리케이션에서 필요로 하는 공통적인 분석 기능을 플랫폼에서 자체적으로 제공하는 것이며, 플러그인 분석 컴포넌트는 플랫폼 내에서 제공되지 않고 애플리케이션 개발자가 필요에 따라 추가한 것이다. 즉, 플랫폼은 설치 후에도 분석 컴포넌트를 새롭게 추가하여 실행시킬 수 있다. 이를 위해, 분석 컴포넌트 등록, 검색, 수정, 삭제 기능을 제공하고, 동적으로 추가된 분석 컴포넌트 로딩 (Loading) 및 언로딩 (Unloading) 기능도 제공한다.

다섯째, 플랫폼 내의 내장 분석 컴포넌트 또는 플러그인 분석 컴포넌트를 실행하여 사용자의 건강 상태를 분석, 평가하는 기능을 제공한다.

여섯째, 분석 컴포넌트에 의해 예측 및 계산된 건강 분석 평가 결과를 저장, 검색, 삭제 기능을 제공한다. 마지막으로, 애플리케이션의 요청에 따라 실행된 플랫폼 기능 수행에 대한 세션을 저장 및 검색하는 기능을 제공한다.

3.2. 비기능적 요구사항 및 기술적 Challenges

플랫폼의 품질은 비기능적 요구사항에 따라 좌우되며, 이는 기술적 Challenge와 연관이 있다. 제안된 플랫폼은 크게 2가지의 비기능적 요구사항을 고려하여 설계된다.

첫째, 건강 분석 애플리케이션들로부터 수집되는 건강 데이터의 종류 및 형식이 다양하기 때문에, 이질적인 데이터를 효과적으로 관리해야 한다. 그리고, 앞으로 추가될 건강 데이터도 수용하도록 설계되어야 한다. 현재에는 가정용 IoT 디바이스에서 수집될 수 있는 건강 측정치인 혈압, 체온, EEG, ECG, EMG 등의 데이터만을 관리하고 있지만, 추후에 다양한 종류의 데이터도 수용할 수 있어야 한다.

둘째, 플러그인 분석 컴포넌트와 플랫폼 간의 상호 운용성을 보장해야 한다. 건강 분석 애플리케이션이 공통적으로 분석 기능을 필요로 하더라도, 건강 나이 예측, 질병 진단 등 건강 상태 분석 종류는 다양하다. 그리고, 그 구현 방법 역시 간단한 통계 처리 기법부터 복잡한 머신러닝 기반 추론 기법까지 다양하다. 이런 가변성으로, 소프트웨어 플랫폼이 모든 종류의 건강 분석 기법을 제공하기 어렵기 때문에, 플랫폼이 확장성 있게 다양한 건강 분석 컴포넌트를 수용하여 실행할 수 있도록 해야 한다. 즉, 플랫폼이 건강 분석에 필요한 핵심적인 기능을 제공하고, 애플리케이션에 특화된 건강 분석 기법을 구현한 컴포넌트를 허용할 수 있도록 확장성 있게 설계되어야 한다.

4. 디지털 건강 분석 플랫폼 설계

4.1. 아키텍처 설계

소프트웨어 플랫폼은 여러 애플리케이션에게 고품질의 기능을 제공하는 것이 매우 중요하기 때문에, 소프트웨어 플랫폼 설계 시에 소프트웨어 아키텍처 설계는 매우 중요하다. 본 장에서는 기능적 요구사항과 비기능적 요구사항을 고려하여 아키텍처를 설계한다.

디지털 건강 분석 플랫폼은 MVC 아키텍처 스타일과

서비스 지향 아키텍처 스타일을 기반으로 설계된다. 그림 1은 두 스타일이 통합된 스켈레톤 아키텍처 구조를 보여준다.

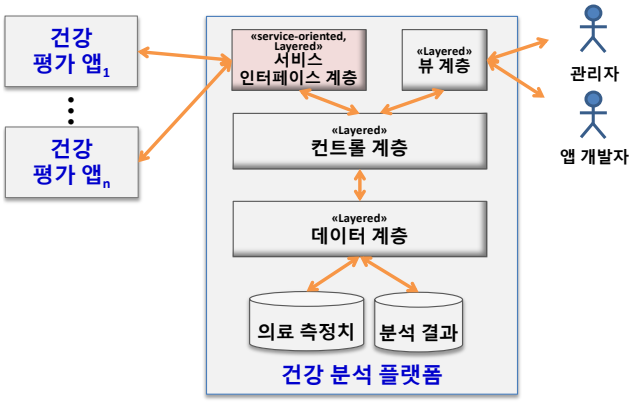


그림 1. 디지털 건강 분석 플랫폼의 스켈레톤 아키텍처

이 플랫폼은 데이터 접근하는 기능 그룹과 비즈니스 서비스를 제공하는 기능 그룹으로 분리되고 여러 애플리케이션에게 내부 구현을 숨기고 기능을 제공해야 하기 때문에, MVC 아키텍처 스타일을 선택하였다. 그리고, 여러 환경에서 운영되는 건강 분석 앱에게 기능을 제공해야 하므로, 서비스 지향 아키텍처 스타일을 채택하였다.

그림 2은 컴포넌트가 할당된 전체 아키텍처 구조를 보여준다. 각 컴포넌트는 플랫폼이 제공해야 하는 기능을 만족시키며, 컴포넌트 간 모듈성을 최대화할 수 있도록 정의하였다.

User Profile Manager는 건강 분석에 필요한 개인 건강 프로파일을 관리하며, Health Meas. Manager는 건강 분석 앱으로부터 수집된 건강 측정치를 관리한다.

H.A. App Manager는 건강 분석 플랫폼을 활용하는 건강 분석 앱에 대한 메타 정보를 관리하며, H.A. Comp. Manager는 건강 분석 플랫폼에서 실행할 수 있는 건강 분석 컴포넌트 메타 정보를 관리하고, H.A Result Manager는 건강 분석 결과를 저장소에 관리한다.

H.A. Comp Executor는 건강 분석 플랫폼에서 가장 중요한 역할을 한다. H.A. Comp. Executor는 런타임에 플러그인된 컴포넌트를 동적으로 로드시키고, 요청에 따라 특정 건강 분석 컴포넌트를 실행한다.

위의 컴포넌트의 기능은 서비스 인터페이스 (Service Interface) 계층의 REST 인터페이스로 노출되며, 앱의 운영 환경에 제약 없이 플랫폼의 기능을 호출할 수 있다.

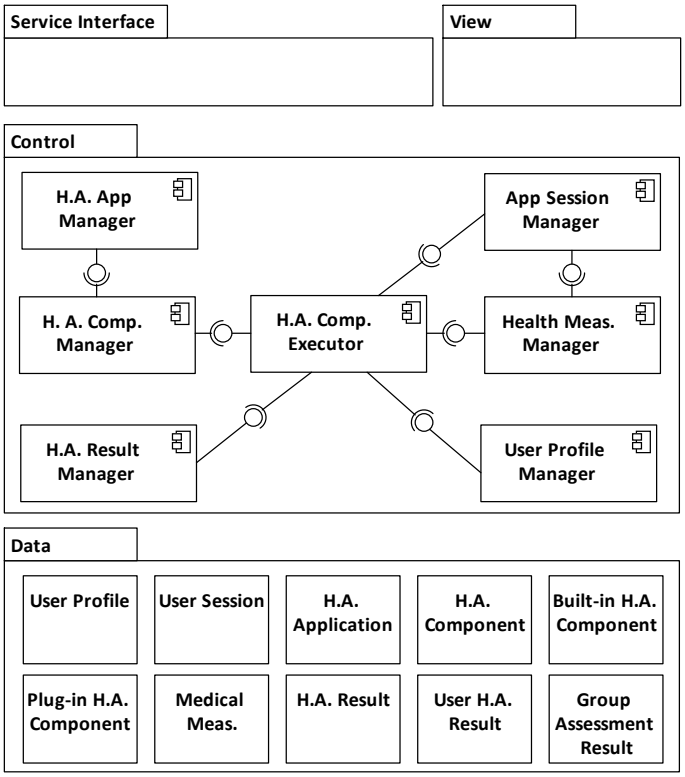


그림 2. 디지털 건강 분석 플랫폼의 기능 뷰 모델

데이터 계층에는 디지털 건강 분석 플랫폼에서 처리되는 핵심 데이터들이 관리되며, 이 계층에 할당된 엔티티 클래스에 대한 설명은 4.2장에서 다룬다.

표 1은 REST 형태로 정의된 H.A. Comp. Executor 컴포넌트 인터페이스의 부를 보여준다. * 로 표시된 입력 매개변수는 기능 호출을 위해 반드시 제공되어야 하는 항목을 의미한다.

표 1. H.A.Comp. Executor 컴포넌트의 인터페이스 일부

upload_health_assessment_component	
설명	새로운 건강 분석 컴포넌트의 메타 정보를 업로드하는 기능을 제공함.
입력 매개변수	Comp_namespace(*), comp_name(*), developer_id (*), input_types(*), output_type(*), comp_file, description
반환 타입	Boolean
retrieve_health_assessment_components	
설명	조건에 맞는 건강 분석 컴포넌트를 검색함.
입력 매개변수	Comp_id, comp_name, developer_id, input_types, output_type, keywords
반환 타입	List of H.A. Comps
…	

이렇게 설계된 디지털 건강 분석 플랫폼은 그림 3과 같이 배치된다.

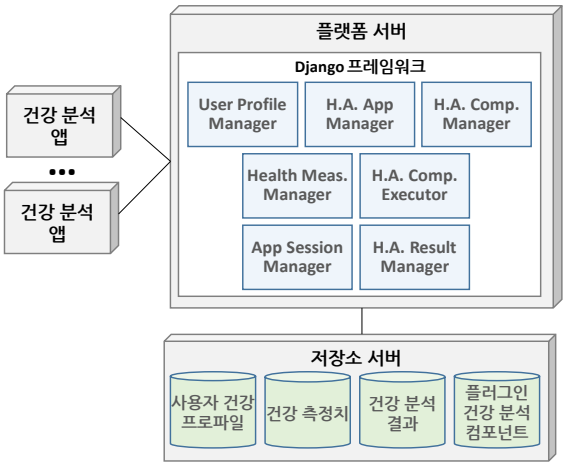


그림 3. 디지털 건강 분석 플랫폼의 배치 뷰 모델

건강 분석 플랫폼에서 건강 분석 결과뿐 아니라 보다 정확한 결과 제공을 위해 평가 시 활용된 건강 측정치도 모두 관리된다. 각 사용자 별로 수집될 수 있는 건강 측정치 데이터 양은 방대해질 수 있으므로, 별도의 디바이스에 위치시킨다.

4.2. 범용적인 건강 측정치 관리를 위한 저장소 설계

본 플랫폼에서는 다양한 타입의 애플리케이션, 사용자 프로파일, 의료 데이터, 건강 분석 컴포넌트, 건강 분석 결과를 지원할 수 있어야 한다. 이러한 데이터 컴포넌트의 타입이나 속성이 매우 다양하여 가변성이 높기 때문에, 이를 고려해서 저장소 설계해야 한다. 그림 4는 이러한 가변성을 처리하기 위한 데이터베이스 설계이다.

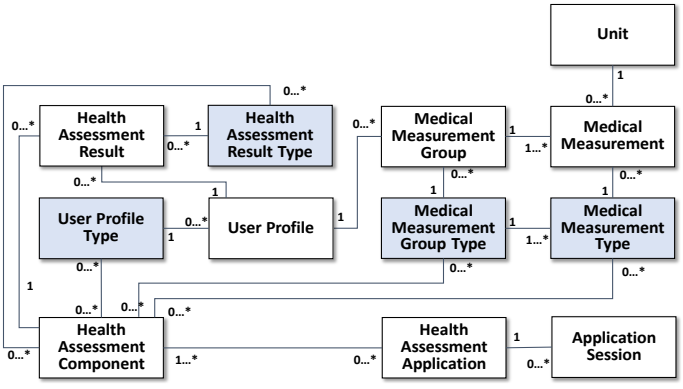


그림 4. 디지털 건강 분석 플랫폼의 데이터베이스 설계

의료 데이터의 경우 그 종류가 다양하고, 애플리케이션마다 필요한 데이터의 종류도 다르기 때문에 데이터 타입에 대한 엔티티를 따로 두어, Medical Measurement와 Medical Measurement Type을 도출한다. 또한 혈압과 같은 의료 데이터는 수축기 혈압과 이완기 혈압의 두 데이터로 이루어져 있고, ECG나 EEG와 같은 데이터는 그래프 형태의 연속적인 데이터의 집합이다. 따라서 이러한 특성도 고려하여 데이터 그룹에 대한 엔티티를 따로 두기 위하여, Medical Measurement Group과 Medical Measurement Group Type을 도출한다.

사용자 프로파일도 다양한 종류의 사용자 프로파일이 있으며, 그 값의 타입도 다양하다. 건강 분석 결과도 건강 분석 컴포넌트에 따라 다양한 타입이 나올 수 있기 때문에, 이를 고려하여 타입에 대한 엔티티를 따로 두어 User Profile과 User Profile Type, Health Assessment Result와, Health Assessment Result Type을 도출한다.

본 플랫폼에서는 애플리케이션을 관리하기 때문에 애플리케이션의 세션도 저장하고 있어야 한다. 이를 위해 Application Session 엔티티가 필요하다.

위와 같이 User Profile Type, Health Assessment Type, Medical Measurement Group Type, Medical Measurement Type 엔티티를 정의함으로써, 새롭게 추가된 데이터 종류를 타입 엔티티에 추가하여 관리한다. 그러므로, 여러 종류의 데이터를 동일한 이름으로 관리할 수 있게 된다.

4.3. 확장성 높은 건강 분석 기법 제공을 위한 설계

디지털 건강 분석 플랫폼에서 가장 기술적 난이도가 높고 중요한 요구사항은 개발자가 추가한 플러그인 건강 분석 컴포넌트도 실행시킴으로써, 건강 분석 기법의 확장성을 높이는 것이다.

이를 위해 H.A. Comp. Executor 컴포넌트는 먼저 전략 패턴을 이용하여 설계된다. 이 기법은 플랫폼에서 실행될 수 있는 건강 분석 컴포넌트의 인터페이스 계층 구조를 정의하는데 활용된다. 그리고, 건강 분석 인터페이스의 계층 구조가 잘 정의되어 있으면, 개발자는 목적에 맞게 인터페이스 계층 구조에 위치한 인터페이스를 구현하여 플랫폼에 플러그인되어 호출될 수 있다.

인터페이스 구조는 건강 분석 기능에 대한 공통성 및 가변성 분석 결과를 기반으로 정의하였다. 여러 건강 분석 애플리케이션은 공통적으로 건강 분석 기능 수행을 위해 제안된 플랫폼을 사용한다. 표 2은 건강 상태 분석 결과를 반환하는 공통 기능성을 나타내는 인터페이스이다. 이 인터페이스는 플랫폼에서 제공하는 모든 건강 분석 컴포넌트들이 반드시 제공해야 하는 오퍼레이션이다.

표 2. 공통 기능성을 나타내는 Required 인터페이스

assess_health	
설명	애플리케이션의 요청에 따라 건강 상태를 분석하고, 그 결과값을 반환함.
입력 매개변수	user_id (*), measurement_types (*), measurement_list, from_time, to_time, profile_types, weights
반환 타입	Assessment Result Type

그러나, 각 컴포넌트의 구현 알고리즘, 결과값, 컴포넌트의 입력 값이 다양하다. 그러므로, 건강 분석 플랫폼은 건강 분석 애플리케이션의 요청에 맞게 결과값을 제공할 수 있도록 설계가 되어야 한다. 그림 5는 전략 패턴을 이용한 H.A Comp. Executor 컴포넌트의 설계 구조를 보여준다.

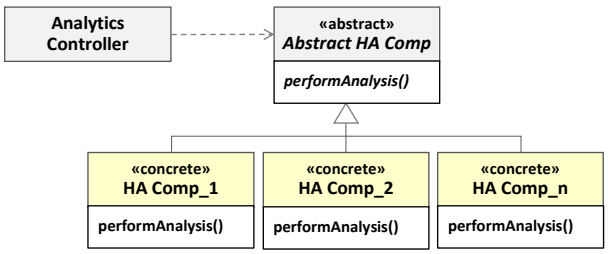


그림 5. 전략 패턴을 이용한 설계

그림의 오른쪽 부분에는 공통 인터페이스인 Abstract_HA_Comp를 각각 다른 알고리즘으로 구현한 건강 분석 클래스들이 위치한다. 이들은 모두 Abstract_HA_Comp의 오퍼레이션을 제공하고 있으므로, Analytics Controller에 의해 호출이 가능하다. 이는 건강 분석 기능을 확장할 수 있는 간단한 방법이다.

두 번째로 H.A. Comp. Executor에 적용된 설계 기법은 동적 로딩 기능을 지원하는 것이다. 이 기법은 런타임에 플러그인된 건강 분석 컴포넌트를 동적으로 로드하여 실행시키는데 필요하다. 이를 위해, 그림 5의 Analytics Controller 클래스는 리플렉션 기법을 이용하여 동적으로 클래스를 로드할 수 있도록 설계하였다. 리플렉션 기법을 이용하면, 런타임에 클래스 내부 구조를 파악하고, 이를 이용하여 오퍼레이션을 호출할 수 있다.

5. 애플리케이션 개발을 통한 플랫폼 평가

5.1. Health Index 애플리케이션 요구사항

Health Index 애플리케이션의 요구사항은 사용자 관리, 장치 관리, 측정치 관리, 건강 분석으로 구성된다.

사용자 관리는 사용자 계정을 관리하기 위한 기능으로서, 사용자 등록, 사용자 정보 수정, 사용자 검색, 사용자 탈퇴의 기능으로 구성된다.

장치 관리는 사용자가 건강을 측정하기 위해 사용할 IoT 장치 등록, 삭제, 연결, 연결 해제로 구성된다.

측정치 관리는 IoT 장치들로부터 건강 측정치를 획득하고, 이를 저장 및 검색하는 기능으로 구성된다.

건강 분석은 IoT 장치들로부터 얻어진 측정치들을 분석하여 Health Index를 계산하는 기능이다. 건강 분석 계산, 건강평가 결과 검색 및 출력으로 구성된다. Health Index는 사람의 건강한 정도를 0과 10사이의 수로 나타내는 단위이며 [9], 건강의 상태를 일반인들이 자신의 건강 상태를 직관적으로 이해하기 판단할 수 있는 유용한 척도로 사용된다.

5.2. 디지털 건강 분석 플랫폼 기반의 설계 모델

Health Index 애플리케이션의 대부분의 기능성은 본 플랫폼에서 제공한다. 애플리케이션이 직접 수행하는 기능성은 센서로부터 의료 데이터를 수집하는 것이며, 그 외의 기능은 모두 플랫폼 API를 호출하여 수행한다.

플랫폼에는 Abstract HA Comp 인터페이스가 정의되어 있고, 3가지의 구현 클래스가 정의되어 있었다. 그러나, Health Index 애플리케이션에서는 그 외에 Muscle Index Assessor와 Mental Index Assessor 컴포넌트가 필요하므로, 플랫폼의 인터페이스에 맞추어 그림 6과 같이 추가 건강 분석 컴포넌트를 설계하고, 플랫폼에 등록하였다.

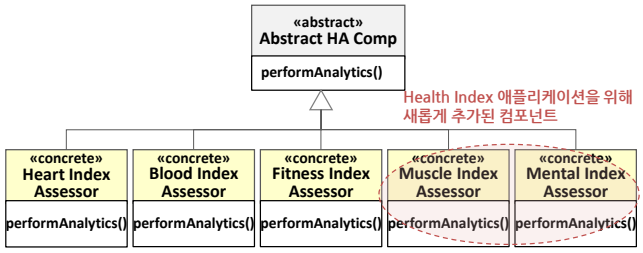


그림 6. Health Index 애플리케이션에서 필요로 하는 건강 분석 컴포넌트

Health Index 애플리케이션은 Web UI를 통해 사용자의 입력을 받고, 의료 데이터 측정 결과와 분석 결과를 보여준다. 따라서, 이 애플리케이션은 MVC 스타일 아키텍처를 사용하여 그림 7와 같이 설계된다.

뷰 계층에는 Web UI가 해당되며, 모델 계층 로컬 캐시와, 의료 데이터의 로컬 저장소가 해당된다. 컨트롤 계층에는 플랫폼을 통한 사용자 프로파일 관리를 위한 User Profile Manager, 플랫폼에 의료 데이터를 저장하고

필요 시 데이터를 검색하는 등의 기능을 제공하는 Health Measurement Manager, 플랫폼에게 분석 요청을 하고, 분석 결과를 받아오는 Health Index Manager가 있다. 이 컴포넌트들은 주로 플랫폼의 API 호출을 통해 기능성을 제공한다. 또한 Sensor Manager는 애플리케이션에 연결된 센서들을 관리하고, 센서를 통해 의료 데이터를 측정하는 기능성을 제공한다.

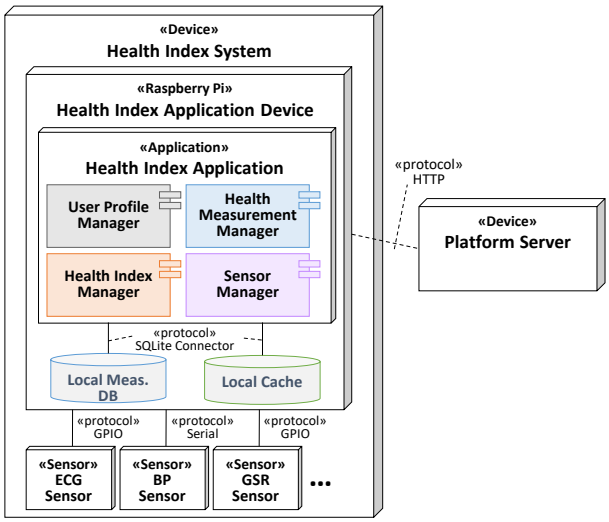


그림 7. Health Index 애플리케이션의 배치 구조

Health Index 애플리케이션은 Raspberry Pi에서 실행되고 있으며, 컨트롤 계층에는 네 가지 컴포넌트가 있다. 애플리케이션은 SQLite를 이용하여 로컬 의료 측정 데이터와 캐시를 관리한다. Raspberry Pi는 GPIO 핀과 Serial 포트를 이용하여 장착된 여러 센서들과 연결되어 입출력을 할 수 있다. 또한, HTTP 통신을 통해 플랫폼 서버의 기능성을 API를 호출하여 사용한다.

5.3. Health Index 애플리케이션 구현 결과

Health Index 애플리케이션은 Raspberry Pi 2 B 모델을 컴퓨팅 보드로 사용하고, 32GB Micro SD 카드를 장착하였으며, Raspberry Pi 보드에 최적화된 OS인 Raspbian을 사용한다. 언어는 Python 3.5, 웹 서버는 Tornado, DBMS는 SQLite3을 사용한다.

애플리케이션에서 사용될 장치의 측정 기능, 장치 관리 기능, 그리고 애플리케이션의 사용자 인터페이스 부분이 가변적이기 때문에 이 부분들을 중점적으로 구현하였고 이외의 공통 부분인 사용자관리, 측정치관리, 건강 분석의 기능은 플랫폼에 REST형태로 요청하여 얻은 결과물을 출력하는 방식으로 구현하였다.

표 3는 메인 화면에서 나타낼 Health Index 중 Mental Index를 플랫폼으로부터 받아오기 위하여 HTTP Get으로 요청하는 코드 부분이다.

표 3. 플랫폼으로부터 Mental Index를 받아오는 소스코드

```
1  #인자로 분석 컴포넌트 이름, 시작 날짜, 마지막 날짜.
2  def get_healthindex(ha_comp=None, datetime_from=None,
   datetime_to=None) :
3      params = {'ha_comp':ha_comp, 'datetime_from':
   datetime_from, 'datetime_to': datetime_to}
4      #전역변수 API_URL 과 인자를 넣고 플랫폼에 요청.
5      res = requests.get(API_URL, params=params)
6      if res.status_code == 200 :
7          data = res.json()
8          index = data['value']
9          return index
10     else :
11         return res.status_code, 'Fail'
12   ...
13 #최근 일주일 간의 시간을 설정.
14 datetime_today = datetime.date.today()
15 datetime_weekago = datetime_today -
16 datetime.timedelta(days=6)
17 #mental component 이름, 일주일 전의 Datetime,
   오늘의 Datetime 을 넣고 메소드 호출.
18 weekly_mental_index =
   gethealthindex('kr.ac.ssu.soft.rainbow.ha_comp.mentalhi',
   datetime_weekago , datetime_today)
```

2번째 줄부터 11번째 줄까지 플랫폼으로부터 Health Index를 요청하는 메소드이다. 5번째 줄은 HTTP Get으로 API의 URL과 분석 컴포넌트 이름, 시작날짜, 마지막 날짜를 인자로 넣고 요청하는 부분이다. 14번째 줄부터 16번째 줄까지 최근 일주일간의 시간을 설정하기 위한 부분이다. 18번째 줄은 메소드를 호출하여 플랫폼으로부터 Mental Index를 받아오는 부분이다. 인자로 Mental Index 분석 컴포넌트 이름, 일주일 전의 Datetime, 오늘의 Datetime을 인자로 메소드를 호출해서 일주일 동안의 Mental Index를 플랫폼으로부터 받는다.

이와 유사하게 모든 데이터들을 플랫폼으로부터 받아와서 그림 8과 같이 화면에 출력하게 된다. 그림 8은 메인 화면으로서 최근 일주일 동안의 사용자 Health Index들을 종류별로 한 눈에 보여주는 화면이다.

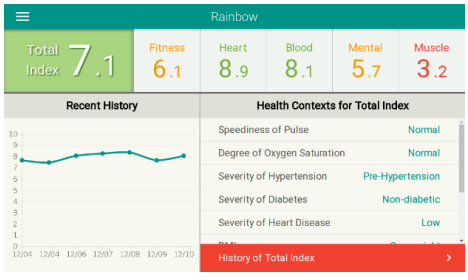


그림 8. 애플리케이션의 메인 화면

이처럼 애플리케이션에서 사용하는 대부분의 기능들을 플랫폼에서 제공하기 때문에 개발 부담을 줄이는 이점을 얻을 수 있다. 플랫폼에 내장되어 있는 분석 컴포넌트를 이용함으로써 건강 분석에 대한 구체적인 방법론의 연구와 이에 대한 개발을 생략할 수 있다는 것이 가장 큰 이점 중에 하나이다.

5.4. 플랫폼 평가 결과

본 장에서는 사례 연구를 결과를 바탕으로 플랫폼을 평가한다.

첫째로, 플랫폼의 범용성 관점에서의 평가한다. 제안된 플랫폼은 사용자 프로파일 타입, 건강 분석 결과 타입, 의료 데이터 타입을 따로 나누어 동적으로 추가할 수 있다. 그러므로, 본 플랫폼을 사용하여 개발할 때, 데이터베이스 설계 및 구축을 하지 않아도 되기 때문에 데이터 컴포넌트 부분에서 개발 비용을 줄일 수 있었다.

둘째로, 분석 컴포넌트의 상호운용성 관점에서 평가한다. 본 애플리케이션에 사용될 분석 컴포넌트는 플랫폼에서 제공하는 인터페이스에 맞추어 개발하였다. 따라서 분석 컴포넌트는 플랫폼에 플러그인되었고, 애플리케이션에서 API를 통해 사용할 수 있었다. Required 인터페이스와 플러그인 기법을 통해 분석 컴포넌트와 플랫폼 간의 상호운용성이 향상됨을 확인하였다.

6. 결 론

다수의 건강 분석 애플리케이션들이 공통으로 필요로 하는 기능성을 제공하는 소프트웨어 플랫폼이 개발된다면 여러 종류의 건강 분석 애플리케이션은 적은 개발 노력 및 비용으로 개발될 수 있게 된다.

그러나, 애플리케이션에서 수집될 수 있는 건강 데이터의 이질성 문제, 애플리케이션에서 필요로 하는 건강 분석 결과 종류의 다양성, 건강 분석의 정확도 보장 어려움의 문제로 플랫폼을 개발하는 것이 어렵다.

본 논문에서는 건강 분석 애플리케이션에게 기능을 제공하는 디지털 건강 분석 소프트웨어 플랫폼을 제안하였다. 이를 위해 먼저, 플랫폼의 기능적 요구사항과 비기능적 요구사항을 정리하였고, 기술적 어려움, 즉 비기능적 요구사항을 해결한 설계 기법을 제시하였다. 그리고, 제안된 플랫폼의 실용성 및 적용 가능성을 증명하기 위해 Health Index 애플리케이션 개발에 적용한 결과를 보여주었다.

제시된 플랫폼은 건강 분석 애플리케이션을 적은 개발 노력 및 비용으로 개발될 수 있게 하며, 플랫폼에서 제공되는 건강 분석 결과는 보다 광범위한 건강 데이터를 기반으로 계산되기 때문에 평가 결과의 정확도가 향상될 수 있을 것으로 기대된다.

7. Acknowledgement

이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (2015R1A2A2A01004078).

8. 참고문헌

[1] S. Hiremathc, et. al, “Wearable Internet of Things: Concept, Architectural Components, and Promises for Person-Centered Healthcare,” *In Proceedings of 2014 EAI 4th International Conference on Wireless Mobile Communication and Healthcare (Mobihealth 2014)*, pp. 304-307, 2014.

[2] A. Forkan, et. al, “Context aware Cardiac Monitoring for Early Detection of Heart Diseases,” *In Proceedings of 2013 Computing in Cardiology Conference (CinC 2013)*, pp. 277-280, 2013.

[3] M. Kuo, et. al, “Design and Construction of a Big Data Analytics Framework for Health Applications,” *In Proceedings of 2015 IEEE International Conference on SmartCity*, pp. 631-636, 2015

[4] C. Sheriff, et. al, “Healthcare Informatics and Analytics Framework,” *In Proceedings of 2015 International Conference on Computer Communication and Informatics (ICCCI 2015)*, pp. 1-6, 2015

[5] S. Sakr and A. Elgammal, “Towards a Comprehensive Data Analytics Framework for Smart Healthcare Services,” *Big Data Research*, Vol. 4, pp. 44-58, June 2016.

[6] R. Arjun and S. C. D’Souza, “Software Analytics Platform for Converged Healthcare Technologies,” *Procedia Technology*, Vol. 24, pp. 1431-1435, 2016.

[7] D. Soudris, et. al, “AEGLE: A big bio-data analytics framework for integrated health-care services,” *In Proceedings of 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2015)*, pp. 246-253, 2015.

[8] C.K. Reddy and C.C. Aggarwal, *Healthcare Data Analytics*, Chapman and Hall/CRC, June 2015.

[9] M.K. Kim, H.T. Jung, S.D. Kim, and H.J, La, “A Personal Health Index System with IoT Devices,” *In Proceedings of the 5th IEEE International Conference on Mobile Services (MS 2016)*, pp. 174-177, June 2016.

기능 분석과 아키텍처 패턴을 이용한 자율주행 자동차 소프트웨어 아키텍처 설계 방법

김순겸, 김두환, Nazakat Ali, 홍장의

충북대학교 컴퓨터과학과

{skkim, nazakatali, dhkim}@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr

Software Architecture Design of Autonomous Vehicle using Functional Analysis and Architectural Pattern

Soonkyeom Kim, Doohwan Kim, Nazakat Ali, Jang-Eui Hong

Department of Computer Science, Chungbuk National University

요 약

자율주행 자동차는 사회적 환경 변화는 물론 ICT 기술의 집약 등의 측면에서 많은 영향을 미치는 대상이며, 최근 많은 연구자들이 관심을 가지고 있는 핵심 기술분야이다. 특히 자동차가 매우 다양한 기능을 제공하고, 또 이들이 소프트웨어에 의한 구현으로 연결되면서 자동차에 탑재되는 소프트웨어 또한 매우 중요하게 인식되고 있다. 최근에는 많은 연구자 및 개발자들이 자율주행과 관련된 자동차 기술 및 소프트웨어를 개발하고 있다. 그러나 자동차의 형태와 기능이 매우 다양하게 제공되고 있음에도 불구하고 자율주행 자동차와 관련된 소프트웨어 개발에서는 자동차 소프트웨어의 전체적인 구조와 지속성 등을 고려하지 못하고 소프트웨어가 개발되고 있다. 따라서 본 연구에서는 자율주행 자동차를 구현하는 소프트웨어를 개발하고자 할 때, 기능 분석 및 소프트웨어 아키텍처를 고려한 소프트웨어 개발 접근 방법을 제시하고자 한다. 특히 자율 주행 기능에 대한 요구사항 분석을 기반으로 아키텍처 패턴을 선택하고, 이를 기반으로 자율주행 자동차가 만족해야 하는 속성을 고려한 소프트웨어 아키텍처 생성 방법을 제시한다. 제시하는 아키텍처의 기반 구조는 추후 다양한 자율 주행 기능과 관련된 소프트웨어 개발에 있어서 기본적인 프레임을 제공할 수 있을 것이며, 기능의 확장이나 개선에 있어서도 매우 유용한 설계 정보로 활용할 수 있을 것이다.

키워드: 자율주행자동차, 기능 분석, 아키텍처 패턴, 소프트웨어 아키텍처

1. 서 론

현대에는 자동차의 많은 기능들이 소프트웨어에 의해 조작되고 있다. 이러한 변화의 최종 목적은 최근 큰 이슈가 되고 있는 자율주행 기능이라고 할 수 있다. 사람이 조작을 하지 않아도 출발지에서 목적지까지 자동차 스스로 주행하는 자율주행 기능은 사람들로 하여금 많은 편의를 제공 하지만, 그만큼 소프트웨어의 안정적인 개발과 안전성의 확보가 가장 중요하다.

최근에 생산되고 있는 모든 자동차에 탑재되는 소프트웨어들은 자동차 기능 안전성 국제 표준 (ISO 26262)을 따라 개발이 되고 있다[1]. ISO 26262의 궁극적인 목표는 차량용 소프트웨어의 전반적인 개발 과정에서의 안전성을 확보하는 것이다. 또한, 코딩 표준으로 MISRA C를 적용함으로써 C언어로 작성된 임베디드 시스템의 코드 안전성, 호환성, 신뢰성을 보장한다[2,3]. 하지만 이러한 표준들을 이용해 자동차 소프트웨어에 대한 안전성을 강화한다고 하더라도, 소프트웨어 결함 또는 조작 실수에 의한 사고는 여전히 나타나고 있다. 또한 최근에 많은 관심을 가지고 개발되기 시작한 자율주행 지원 소프트웨어의 경우에 있어서는 단순히 자동차의 기능 안전과 관련된 부분이 아니라 영상처리, 센서, 무선 통신 등과 같은 다양한 IT 분야의 기술이 복합적으로 적용되는 분야이기 때문에 아직 ISO 26262나 MISAR C에 대한 적용이 충분하게 이루어지지 않고 있는 실정이다.

특히 자율주행 자동차는 운전자의 간섭이 기존의 자

동차보다 적어지는 특징을 가지고 있지만, 그만큼 보다 정교하게 소프트웨어가 자동차의 여러 기능들에 대한 제어를 담당해야 하는 문제가 생겨난다[4].

또한 자동차는 고가의 상품이며, 가격에 따라 동일 모델이라 하더라도 많은 기능적 차이를 보인다. 이는 가격 및 모델에 따른 소프트웨어의 동작이 상호 다른 제어 및 기능을 탑재함을 알 수 있다. 이러한 상황에서 기존 차량에 새로운 기능을 확장하거나 자동화 하는 것이 더욱 보편화될 것이다. 또한 운전자의 특성에 따라 자동차도 진열된 상품이라기 보다는 이제는 맞춤화된 개인 용품으로 변모할 것이라 예측할 수 있다.

이러한 자율주행 자동차에 대한 여건의 변화에서 소프트웨어는 매우 중요한 역할을 수행할 것이며, 특히 차량의 전체적인 소프트웨어 구성요소간 제어 및 연동 관계를 표현하는 차량의 소프트웨어 아키텍처는 무엇보다도 중요해질 것으로 판단된다.

본 논문에서는 이러한 기술 변화 및 다양성을 지원하기 위하여 자율주행 자동차의 핵심 소프트웨어에 대한 기능 분석을 수행하고, 해당 기능에 적합한 아키텍처 패턴을 이용하여 보다 안정적이고, 확장성 있는 자율주행 차량용 제어 소프트웨어 아키텍처를 설계하고자 한다. 이를 통해서 매우 다양한 자율주행 관련 기능들이 보다 쉽게 구현되고 실현될 수 있을 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 차량용 소프트웨어와 관련된 아키텍처 및 아키텍처 패턴 등과 관련된 기존의 연구들을 살펴본다. 3장에서는 자율주행 자동차 소프트웨어의 요구사항에 대하여 설명하고, 4장에서는 패턴 기반의 자율주행 자동차 소프트웨어 아키텍처 설계 방법을 제시한다. 5장에서는 제시한 방법에 근거하여 사례 적용 결과를 설명하며, 6장에서 결론 및 향후 연구 내용을 제시한다.

2. 관련 연구

2.1 차량용 소프트웨어 개발 플랫폼 AUTOSAR

AUTOSAR는 AUTOmotive Open System Architecture의 약자로, 자동차 전장용 임베디드 소프트웨어 개발의 생산성 향상을 위해 수 개의 자동차 관련 회사가 협력하여 개발되었다.

AUTOSAR는 표준화된 소프트웨어 구조를 정의하며, 모듈화 된 소프트웨어 구성을 지원한다. 이러한 구성을 통해 개발자 입장에서 기능들을 서로 다른 모듈로 구현하게 함으로써 복잡도를 감소시켜 생산성을 향상시킨다[5]. 모듈간 인터페이스만 정해지면 각 모듈 별로 다른 업체에서 개발이 가능하므로 분업이 가능하여 개발 일정을 단축시킬 수 있다. 또한 자주 사용되는 기능을 모듈화하여 모든 ECU에서 재사용이 가능하다[6].

AUTOSAR의 구조는 크게 SW-C (Software Component), RTE(Run-Time Environment), 그리고 BSW(Basic Software)의 3계층으로 나눠진다. RTE는 각 SW-C 사이 및 SW-C와 BSW 사이의 정보 교환을 위한 중추적인 역할을 하며, SW와 HW를 분리시키는 역할을 한다. RTE 하부에는 BSW 계층이 존재하며, BSW의 표준 계층으로 Service Layer, EAL(ECU Abstraction Layer), MCAL(Microcontroller Abstraction Layer), CDD(Complex Device Driver)로 구성된다[7].

AUTOSAR SW 개발은 시스템 설정 단계와 ECU 설정 단계로 나누어진다. 시스템 설정 단계에서는 SW-C의 데이터 타입, 인터페이스와 연결 상태 등을 기술하는 SW-C 명세서, 각 ECU의 HW 구성을 기술하는 ECU 자원 명세서, 버스 시그널, 토폴로지 등 시스템 제약 명세서를 작성한다. 각 SW-C 내부에는 응용 SW 구현을 위한 태스크 동작 정의 및 트리거 조건을 정의한다. 다음은 SW-C를 각 ECU에 매핑하고 네트워크 설계를 하여 시스템 설계 명세서를 기술한다. 작성된 파일은 XML 형식의 템플릿을 사용하며 XML을 사용함으로써 데이터의 공유 및 전달을 표준화 할 수 있다. 다음 단계는 시스템 설계명세서로부터 각 ECU 정보를 추출하여 ECU 설계를 하며, 태스크 정의 및 할당, RTE 생성, BSW 등을 설계하여 ECU 설계 명세서를 기술한다. 응용 SW와 함께 RTE, OS, Communication 등의 AUTOSAR SW 모듈 코드를 생성하고, 컴파일, 링크를 거쳐 실행 파일을 만들어 ECU 응용 서비스를 구현한다. 구현된 동작 가능한 결과물은 설계된 ECU에 올려 시험할 수 있다[8].

2.2 임베디드 소프트웨어 설계 패턴

2.2.1 Publisher-Subscriber 패턴

이 패턴은 하나의 Publisher가 다수의 Subscriber에게 상태 변화에 대한 정보를 단방향으로 통지하는 패턴이다. 이 패턴은 한 번의 호출로 다수의 컴포넌트의 상태를 변경해야 하는 경우에 사용하고 그 과정은 다음과 같다. 하나의 컴포넌트가 Publisher가 되고, Publisher로부터 상태 변경을 받을 컴포넌트가 Subscriber가 된다. Subscriber는 Publisher에서 제공되는 인터페이스를 통해 등록한다. Publisher의 상태가 변경되면, Subscriber에게 변경 상태를 전송한다. 여기서 Publisher에서 발생한 상태 변경에 대하여 하나 이상의 Subscriber가 수신해야 하고, Publisher와 Subscriber는 coupling이 높아서는 안 된다[9].

이 패턴의 장점으로는 하나의 패턴으로부터 다수의 컴포넌트로 동시에 변경에 대한 공지를 할 수 있다는 점과 GUI 인터페이스, GUI 빌더, 프레임워크 등을 쉽게 만들 수 있다는 점이 있고, 단점으로는 이벤트 핸들러와 프로세스 로직이 Tight하게 coupled 되어있다는 것과 Non-deterministic 한 문제가 있다.

2.2.2 Pipes and Filters 패턴

이 패턴은 데이터 스트림을 처리하는 패턴으로 데이터는 Pipe를 통해서 다른 Filter로 전달되어 진다. 전달된 데이터는 Filter에서 걸러지게 되고, 다시 Pipe를 통해 다음 Filter로 이동하는 패턴이다. 이 패턴은 입력된 데이터 스트림을 처리하거나 변환할 때 사용되는데, 입력된 데이터에 대하여 순차적으로 과정을 처리하고, 각 단계(Filter)에서 처리가 완료된 output 데이터는 다음 Filter의 input값으로 사용된다[9].

Pipes and Filters 패턴의 장점으로는 중간 결과 파일이 필요하지 않고, Filter의 교환이나 재조합이 쉬워 재사용이 용이하다는 점이 있고, 단점으로는 Filter 간의 전송 시간이나 Context Switching 시간과 같은 처리 시간이 성능에 악영향을 미칠 수 있다는 것과 에러 처리가 어렵다는 점이 있다.

2.3 아키텍처 개발 기법

ADD(Attribute-Driven Design) 방법은 소프트웨어 집약적 시스템의 소프트웨어 아키텍처를 설계하기 위한 체계적인 단계별 방법으로, 아키텍처의 품질 속성 요구사항에 기초하여 설계 프로세스를 기반으로 소프트웨어 아키텍처를 정의하는 방식이다. ADD 방법은 분해의 각 단계에서 품질 속성 시나리오를 충족시키기 위해 전술 및 아키텍처 패턴을 선택하는 재귀적 decomposition 프로세스를 따른다[10].

ADD에 필요한 입력은 기능 요구사항, 품질 속성 요구사항 및 제약사항이 있다. 기능 요구사항은 기능 목록 또는 사용 사례로 정의될 수 있고, 품질 속성 요구사항은 품질속성 시나리오를 사용하여 지정할 수 있다. 제약조건은 외부 요인에 의해 강제적으로 결정된다[11]. ADD 방법의 과정은 다음과 같다.

[Step 1] 분할 할 모듈의 선택.
[Step 2] 선택 할 모듈이 달성해야 하는 아키텍처의 요구사항을 나열하고 우선순위를 메김.
[Step 3] 아키텍처 패턴을 결정하여 하위 모듈과의 상호작용 방식 결정.
[Step 4] 하위 모듈을 실체화 하고 다양한 뷰를 통해 모듈 기능을 할당.
[Step 5] 하위 모듈의 인터페이스 정의.
[Step 6] usecase와 품질 속성을 검증하여 하위 모듈의 제약사항을 생성.
[Step 7] 모든 모듈을 분할 할 때까지 반복 수행.

3. 자율주행 자동차 기능 요구사항

자율주행 자동차가 제공하는 기능은 크게 인지기능, 판단기능, 그리고 제어 기능으로 크게 구분할 수 있다[12]. 그림 1과 같이 다양한 센서로부터 외부의 상황을 입력 받아, 이를 기반으로 주행을 위한 전략적 판단을 수행한다. 그리고 이를 기반으로 자동차의 관련 장치를 제어하여 차량이 안전하게 주행할 수 있도록 한다.

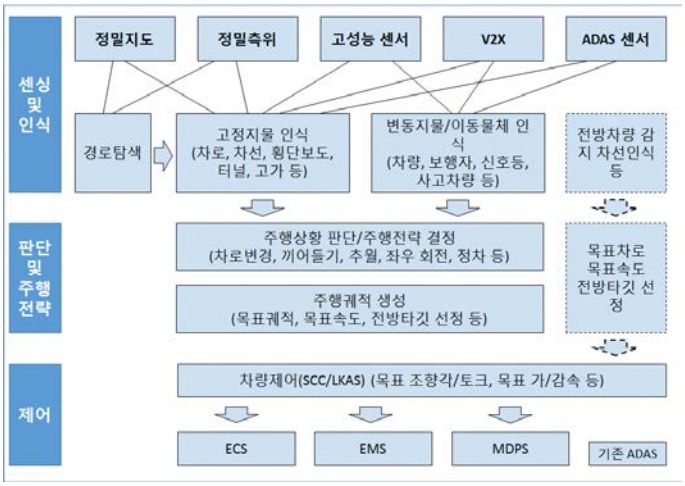


Fig. 1 자율주행자동차의 핵심 기능 구성

자율주행 자동차의 차선 탐지 기능을 구현 관점에서 살펴보면, 먼저 카메라를 통해 수집한 영상 데이터는 1) 카메라에서 전송 된 화면 중 시스템에서 도로의 환경을 분석해야 하는 영역을 지정하고, 2) 실제 시스템에서의 인식률을 높이기 위하여 gray 이미지로 변환하고, 3) edge를 검출하기 위해 canny edge detection 알고리즘[13]을 적용한다. 다음으로는 canny edge detection 알고리즘을 적용한 영상에서 환경 분석을 해야 하는 영역 내의 edge 값의 허용 범위를 결정하여 범위 내의 값들의 평균값을 차선으로 인식한다. 이렇게 차선으로 인식 한 두 평균 edge의 중앙값을 차선 중앙이라고 인식하도록 하여 주행 방향을 설정한다.

이와 같은 자율 주행을 위한 기능의 개발에 있어서 중요하게 고려되어야 하는 비기능적(품질) 요소들을 정리하면 표 1과 같다.

Table 1. 차선 탐지 시스템 비기능적 요구사항

품질요소	요구사항
성능	영상 데이터 입력으로부터 주행 방향을 정하는데 걸리는 시간이 15ms 이내여야 한다.
신뢰성	도로상의 변수가 차선 탐지에 영향을 미치면 안 된다.
	Edge 데이터의 수가 신뢰할 수 없을 정도로 적을 때는 전, 후 주행 방향의 데이터를 사용하여 주행 방향을 결정한다.
안전성	차량의 바퀴는 차선을 벗어나서는 안 된다.
	차량의 바퀴가 차선을 벗어나는 경우 사용자에게 알림을 줘야 한다.
보안성	자율주행 자동차에 부착된 카메라 외에 다른 영상정보를 입력 받지 않도록 한다.

자율 주행을 위한 차량용 소프트웨어 개발에 있어서 단순히 기능적인 측면뿐만 아니라 적합한 품질 요소를 고려하는 것도 매우 중요하다. 특히 이러한 품질 요소는 차량 및 운전자의 안전을 보장하는 중요한 속성이다. 따라서 차량 탑재용 소프트웨어 개발시에는 단지 기능적인 측면에 아닌 품질 측면의 요소도 반드시 고려되어야 한다[14].

소프트웨어 아키텍처는 이러한 품질적 속성을 관점(Perspective)이라는 차원에서 표현할 수 있는 방법을 제공한다. 해당 품질 요소를 만족할 수 있는 전술(Tactics)을 선정하고 선정된 방법을 소프트웨어 개발상에 적절하게 구현되어야 한다. 따라서 자율주행을 위한 소프트웨어 개발에서 아키텍처를 체계적으로 잘 적용하는 것은 무엇보다도 중요하다.

4. 자율주행 자동차 SW 아키텍처 설계 방법

이 절에서는 3절에서 설명한 자율주행 차량용 요구사항을 기반으로 자율주행 자동차의 소프트웨어 아키텍처를 설계하기 위한 절차를 제시한다.

제시하는 아키텍처 설계 절차는 그림 2에서 보는 바와 같이 먼저 주어진 요구사항을 기반으로 기능 분석(Functional Analysis)을 수행하고, 분석된 기능을 제공할 수 있는 적합한 아키텍처 패턴을 선택한다. 선택한 패턴을 활용하여 전반적인 아키텍처를 정의한 후에, 이 아키텍처가 소프트웨어의 품질 요소를 만족하고 있는지를 점검한다.

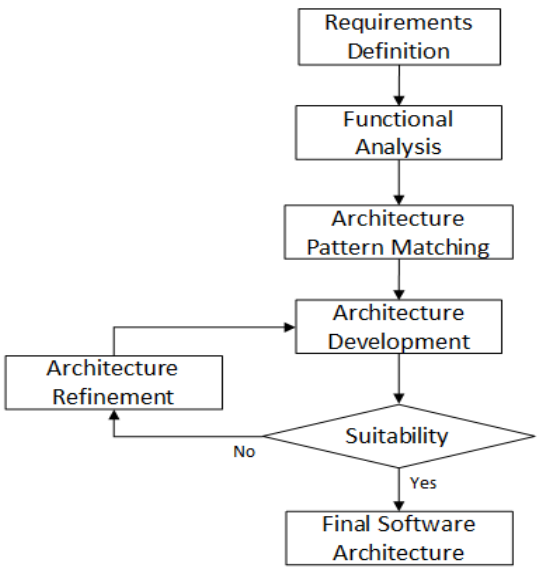


Fig. 2 기능분석 기반 소프트웨어 아키텍처 개발절차

그림 2의 아키텍처 개발절차는 차량 탑재형 소프트웨어에 대한 기능 분석과 이에 대응하는 패턴 선택을 통한 소프트웨어 아키텍처 설계 방법을 제시하고 있다. 이 과정에서 기능 분석은 특별히 ISO 26262에서 기능 안전성을 만족하는 측면을 고려해야 하며, 적합성(Suitability) 평가에서는 부가적으로 제시하는 자율주행 지원 소프트웨어에 대한 품질 요소가 충분히 만족되었는지를 점검하여 아키텍처를 보완하게 된다.

그림 2에 제시하는 절차의 각 단계별로 보다 구체적으로 설명하면 다음과 같다.

(1) Requirements Definition 단계

이 단계에서는 개발하고자 하는 자율주행 자동차의 요구기능을 정의하는 활동을 수행한다. 기존의 자율 주행과 관련된 요구사항들을 3장에서 간략히 설명되었지만, 만약 새로운 기능을 개발하고자 하는 경우(예를 들면, 차량간 무선 통신을 통한 위협 대처 기능)에는 이들 기능에 대한 정확한 요구사항을 비기능적 요구사항과 함께 식별되어 정의되어야 한다.

(2) Functional Analysis

기능 분석은 요구사항 분석을 통해 식별된 기능, 성능, 인터페이스 및 기타 요구사항을 설계 활동에 활용하기 위하여 시스템의 일관된 기능으로 변환하는 과정을 의미한다. 따라서 기능 분석은 시스템이 수행해야 하는 모든 기능을 지속적으로 분할하여 식별한다. 이러한 분할된 기능은 요구사항 관점에서 상호 매칭되어야 하며, 추후 시스템(혹은 소프트웨어) 아키텍처 개발의 기반 정보로 제공한다.

(3) Architecture Pattern Matching

기능 분석 과정 후, 이를 만족하는 아키텍처 패턴을 선정한다. 이를 위해서는 먼저 기능 분석을 통해 얻은 상세한 기능 분석 결과를 기반으로 아키텍처 개발을 위한 기반 드라이버(Drivers)가 무엇인지를 정의한다. 정의된 아키텍처 드라이버는 해당 기능을 제공하는 패턴을 식별하는 데 사용된다.

(4) Architecture Development

선택한 아키텍처 패턴을 이용하여 자율주행 자동차의 소프트웨어 기능을 개발하기 위한 아키텍처를 정의한다. 아키텍처 정의에 있어서는 패턴을 포함하는 전체 시스템 구성요소에 정확한 기능을 할당하고, 이들간의 인터페이스를 정의한다. 이러한 아키텍처를 개발하는 과정은 기존의 많은 연구들에서 사용하고 있는 ADD(Attribute Driven Development) 방법을 활용할 수 있다.

(5) Suitability Check

적합성 점검은 앞 단계의 결과물로 확보한 소프트웨어 아키텍처가 자율주행 자동차의 Architecture Drivers를 충족하고 있는지를 점검하는 것이다. 따라서 각 드라이버가 포함되는 아키텍처 구성요소가 무엇인지 정의하고, 이것이 요구사항관점에서 제시하는 비기능적 요구사항을 시나리오 기반으로 만족시킬 수 있는지 점검한다.

(6) Architecture Refinement

생성된 아키텍처 드라이버를 만족하는 소프트웨어 아키텍처가 생성되었다면, 이는 추후 자율주행 소프트웨어 개발에서의 기반 구조로 활용될 수 있다. 그러나 만약 아키텍처 드라이버를 만족시키지 못했다면, 생성된 아키텍처를 정제하는 과정을 수행해야 한다.

이러한 정제 과정에서는 기존에 식별되지 못한 새로운 아키텍처 구성요소가 생성되거나, 또는 기존의 요소를 합성하는 과정 등이 이루어질 수 있다.

5. 예제 시스템 적용

앞서 설명한 소프트웨어 아키텍처 설계 절차를 기반으로 자율주행 자동차에 탑재되는 차선 탐지 기능에 대한 아키텍처 개발 과정을 설명한다.

5.1 주행 차선 탐지 요구사항

자율주행 자동차의 차선 탐지 기능은 차량에 탑재된 카메라에서 입력 받는 정보를 이용하여 영상 처리를 통해 도로의 환경을 판단하는 것이다. 차선 탐지 절차를 따라 입력 받은 영상의 일정 구간에서의 차선을 인지하여 자율주행 자동차가 주행할 경로를 제시한다. 이에 대한 상위수준에서의 기능 및 비기능 요구사항을 정리하면 다음과 같다.

- 기능 요구사항
  - 신호 입력 기능: 카메라 센서 등으로부터, 차량 전방에 대한 환경 정보를 입력 받는다.
  - 신호처리 기능: 입력된 정보를 처리하여 전방에 대한 차선 인식을 포함하여, 고정지물, 변동지물, 사람 등을 식별한다.
  - 차량제어: 처리 정보를 기반으로 차량의 자율 주행을 위한 핸들 방향, 속도 등을 제어하기 위해 각각의 구동기(Actuator)에게 적합한 신호를 보낸다.

- 비기능(품질) 요구사항
  - 신호 입력에서부터 차량을 제어하기까지의 소요시간은 15ms를 넘을 수 없다.
  - 차량이 정해진 차원을 벗어나는 경우, 사용자에게 즉각 알림을 보내야 한다.
  - 주행 중에 위급 상황이 발생하는 경우, 차량은 반드시 정차되어야 하고, 자율주행 기능은 Off 된다.

5.2 자율 주행 기능 분석

자율 주행 기능의 요구사항은 크게 입력, 처리, 제어의 기능으로 구별되어 제시되었는데, 이들에 대한 기능 분석을 통해 보다 세부적인 기능을 도출하면 표 2와 같이 정리할 수 있다.

Table 2. 자율주행 기능의 분석

상위 기능	상세 기능	관련 품질사항
신호입력	카메라 영상 입력	연산 처리 속도
	센서 입력	
	지도 정보 입력	
신호처리	차선 검출	검출 오류 처리
	고정 지물 검출	
	변동 지물 검출	
	주행 전략 결정	
차량제어	바퀴 조향각 제어	사용자 경고
	속도 제어	급정지 제어

5.3 아키텍처 패턴 식별

차선 탐지를 통한 주행을 지원하기 위한 기능 개발에서 아키텍처에 가장 큰 영향을 끼치는 아키텍처 드라이버는 표 3과 같이 정리할 수 있다.

Table 3. 차선 탐지 기능의 아키텍처 드라이버

#	아키텍처 드라이버
1	카메라로부터 영상을 전송 받아 출력 한다.
2	영상 정보를 바탕으로 edge를 검출, 차선으로 인식한다.
3	차선으로 인식 된 edge의 중앙값을 이용해 주행경로를 도출한다.
4 5	영상 데이터 입력으로부터 주행 방향을 정하는데 걸리는 시간이 15ms 이내여야 한다.
6	Edge 데이터의 수가 신뢰할 수 없을 정도로 적을 때는 전, 후 주행 방향의 데이터를 사용하여 주행 방향을 결정한다.
7	차량의 바퀴는 차선을 벗어나서는 안 된다.
8	차량의 바퀴가 차선을 벗어나는 경우 사용자에게 알림을 줘야 한다.
9	자율주행 자동차에 부착된 카메라 외에 다른 영상정보를 입력 받지 않도록 한다.

이상에서 정의한 아키텍처 드라이버를 기반으로 고려할 수 있는 자율주행 자동차 소프트웨어의 특성은 concurrency가 중요한 사항이라는 것을 알 수 있다. 계속적으로 카메라로부터 영상을 전송 받고, 받은 영상 데이터를 각 단계에 맞는 프로세스를 실행하여야 하기 때문에 Synchronous와 Asynchronous 모두 고려한 아키텍처 패턴인 Half-Sync/Half-Async 패턴을 사용할 수 있다.

Half-Sync/Half-Async 패턴은 동기적 I/O 모델과 비동기적 I/O 모델을 모두 사용하는 패턴으로써 Synchronous task Layer와 Queueing Layer, 그리고 Asynchronous task layer로 나눠진다. 그림 3는 Half-

Sync/Half-Async 패턴의 구조를 나타낸 것이다.

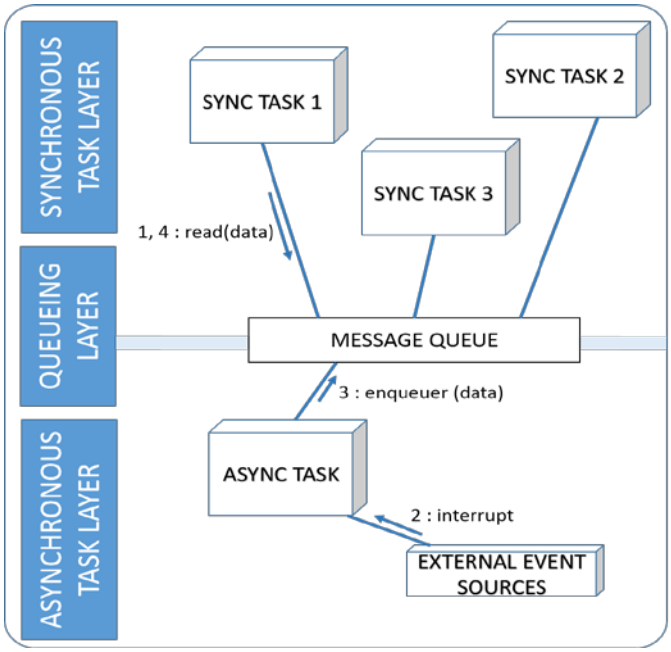


Fig. 3 Half-Sync/Half-Async 패턴

5.4 차선 탐지 기능의 아키텍처 생성

개발 대상의 아키텍처 패턴 선정된 이후에는 대상의 요구사항과 아키텍처 패턴을 이용하여 대상 시스템에 대한 소프트웨어 아키텍처를 개발한다.

그림 4은 차선 탐지 기능의 요구사항과 식별된 half-Sync/half-Async 패턴을 이용하여 개발한 차선 탐지 기능의 상위 수준 아키텍처 이다.

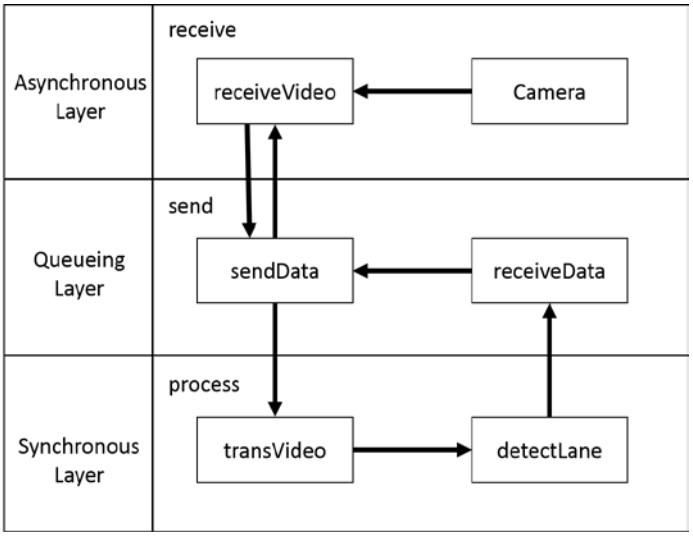


Fig. 4 차선 탐지 시스템 아키텍처

그림 4에 나타난 아키텍처 구성의 각 컴포넌트가 갖는 기능을 정의하면 다음과 같다.

- Camera : 도로의 영상 데이터를 입력받는다
- receiveVideo : Camera에서 입력 받은 영상 데이터 값과 영상 분석 process가 끝난 영상 데이터를 수신한다.
- sendData : 영상 데이터를 transVideo 또는 receiveVideo 컴포넌트로 보낸다.
- receiveData : 영상정보 분석이 끝난 영상 데이터를 수신한다.
- transVideo : 입력받은 영상 정보에 대하여 영상 color 변환 작업을 수행한다.
- detectLine : canny 영상을 이용하여 차선 인식을 위한 알고리즘을 적용하여 영상정보에서 차선을 찾아 표시한다.

5.5 적합성 점검

요구사항에 의한 아키텍처 개발 이후에는 이전 단계에서 만들어진 아키텍처에 대한 기능 만족도와 아키텍처가 차선 탐지 시스템의 요구사항을 만족하는지에 대한 적합성 검사가 필요하다. 이 절차에서는 ATAM 프로세스와 유사한 방법으로 아키텍처가 처음 의도한 방향대로 제대로 설계되었는지에 대해 검증한다.

이러한 점검을 위하여 다음과 같은 2가지 차원에서의 적합성 검증을 시도하였다. 첫 번째는 자율주행 자동차의 차선 검색을 위한 기본 속성 시나리오를 이용하여 아키텍처가 해당하는 부분의 기능을 지원하는 가를 판단하는 것이며, 두 번째는 안전한 주행을 위하여 요구사항에서 제시한 시간 제약 사항 - 즉, 영상 수신으로부터 차량의 방향 결정을 위한 평균 시간이 15ms - 의 적합성을 검증하는 것이다.

(1) 차선정보를 확보하지 못하는 경우 차선 식별

개요	정상적으로 차선을 식별하지 못한 경우에 대하여 주행 방향의 결정
시스템 상태	카메라로부터 외부의 정보를 지속적으로 수신하여 차선 검출 기능을 수행한다.
환경	정상 운영 상태
외부 자극	카메라 영상의 입력
시스템 반응	차량 주행 방향을 결정하는 바퀴의 목표 조향각을 결정한다.

위와 같은 시나리오에 대하여 그림 4의 아키텍처는 해당 기능에 대한 처리를 갖는 컴포넌트를 식별할 수 없다. 따라서 위의 시나리오를 만족할 수 있도록 아키텍처는 정제 또는 보완되어야 한다.

(2) 시간 제약 사항의 점검

카메라 영상으로부터 정상적인 주행 방향을 결정하기까지 소요되는 시간이 15ms 이내여야 한다는 요구사항

을 점검하기 위하여 그림 5와 같은 Sequence diagram을 생성할 수 있다.

그림 5에 나타난 Sequence Diagram [15]의 모든 메시지 전송에 대한 시간의 합이 15ms를 넘지 말아야 한다. 따라서 다이어그램에 나타난 모든 메시지에 대하여 다음과 같은 시간의 점검이 필요할 수 있다.

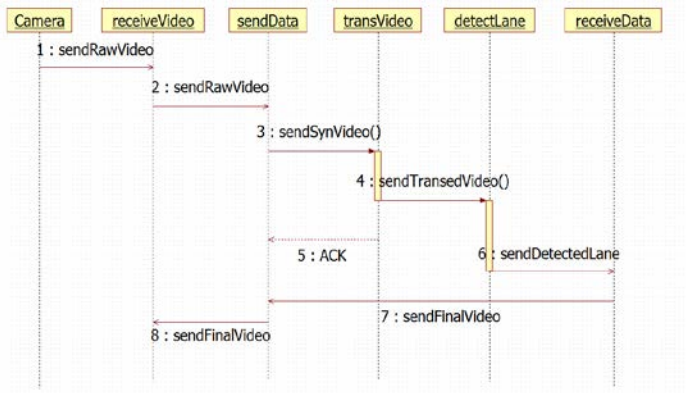


Fig. 5 차선 탐지 기능의 Sequence Diagram

- 메시지 1, “sendRawVideo“ : 1ms
- 메시지 2, “sendRawVideo” : 1ms
- 메시지 3, “sendSynVideo“ : 1ms
- 메시지 4, “sendTransedVideo“ : 4ms
- 메시지 5, “sendDetectedLane“ : 6ms
- 메시지 6, “sendFinalVideo“ : 2ms
- 메시지 7, “sendFinalVideo“ : 2ms

위와 같이 식별된 각 메시지에 대한 처리 시간의 합은 17ms가 됨을 알 수 있다. 따라서 시간 제약에 대한 요구사항이 만족되지 않기 때문에 이에 대한 보완이 요구된다.

5.6 아키텍처 보완 및 정제

5.4절에서 정의된 아키텍처가 기능 적합성을 만족하지 못하기 때문에 이에 대한 보완이 필요하다. 5.5절에서 제시한 두 가지 사항을 보완하기 위해서는 각각 다음과 같이 진행하였다.

- 충분한 차선 정보를 확보하지 못하는 경우: 이 경우에는 아키텍처를 구성하는 컴포넌트에 새로운 컴포넌트를 추가하여 해당 기능을 제공하는 기능을 할당해 주어야 한다.

- 시간제약 조건을 만족하지 못하는 경우: 이 경우는 시간제약 조건을 만족하기 위한 새로운 아키텍처 패턴이 필요하다. 즉, 카메라로부터 입력되는 영상에 대한 처리의 속도를 높이기 위해서 가장 많은 병목 현상을 나타내는 부분에 대한 해결책이 강구되어야 한다.

일반적인 영상처리의 과정에서 처리 속도의 향상을 위해서는 성능이 우수한 알고리즘을 사용하는 것이 일반적이라고 볼 수 있으나 본 연구에서는 알고리즘 차원이 아닌 영상 처리의 모듈의 다중화를 통해 해결하고자 한다. 따라서 카메라로부터 수신되는 영상 데이터의 변환 과정을 이중화하여 개선된 응답 시간을 얻을 수 있다.

위와 같은 두 가지 해결 방안을 기반으로 새롭게 정의 하는 자율주행 자동차의 차선 탐지 기능에 대한 소프트웨어 아키텍처는 그림 6과 같다.

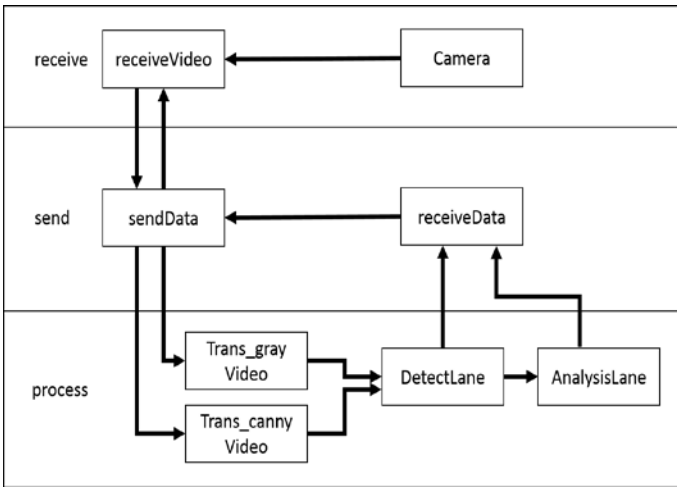


Fig. 6 최종 개발된 차선 탐지 기능의 아키텍처

6. 결론

본 논문에서는 요구사항 분석의 기능 분석을 통하여 자율주행 자동차의 기능 중 차선 탐지 시스템에 대하여 분석하고, 이를 바탕으로 아키텍처 패턴을 매칭시켜 초기 아키텍처 설계에 대한 가이드를 제공하고, 설계된 아키텍처에 대하여 적합성 테스트를 통하여 아키텍처 검증을 하는 방법을 제시하였다.

본 연구는 현재 활발하게 이루어지고 있는 자율주행 자동차 소프트웨어 개발에서의 아키텍처 패턴 선정에 대한 가이드를 제공함으로써, 안전성과 안정성이 중요한 자율주행 자동차 소프트웨어의 초기 아키텍처 선정 단계에서의 노력을 줄일 수 있을 것으로 기대된다.

향후 연구 내용으로는 차량용 소프트웨어의 기능적 특성들에 대하여 분류를 하고, 이에 따른 소프트웨어 아키텍처 패턴과의 매칭 방법에 대하여 연구하고자 한다.

Acknowledgement

This research was supported by the NRF funded by the MSIP (NRF-2014M3C4A7030505) and was also supported

by the NRF funded by the MOE (NRF-2014R1A1A4A-01005566).

참고문헌

[1] ISO 26262-6: Road vehicles-Functional safety-Part 6: Product development at the software level, 2011
[2] MISRA C_2012 Guidelines for the use of the C language in critical systems, 2012.
[3] MISRA Limited, Guidelines for safety analysis of vehicle based programmable systems, 2007
[4] Ashraf Armoush, "Design Patterns for Safety-Critical Embedded Systems", 2010
[5] R. Yerushalmi and R. A. Felice, "Implementing AUTOSAR Atomic Software Components Using UML/SYSML in C," SAE 2010-01-0265, 2010.
[6] R. Obermaisser, "A Multi-Core Platform for Integrated Modular Avionics Derived from a Cross-Domain Embedded System Architecture", SAE Technical Paper 2009-0-3262, 2009
[7] AUTOSAR Technical Overview 3.0, AUTOSAR, 2007.
[8] K. Lakshmanan, G. Bhatia and R. Rajkumar, "Integrated end-to-end Timing Analysis of Networked AUTOSAR-compliant Systems," in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp.331-334. 2010.
[9] F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "Pattern-Oriented Software Architecture: A System of Patterns", John Wiley & Sons, 1996.
[10] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, B. Wood, "Attribute-Driven Design (ADD)," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2006-TR-023, 2006.
[11] Robert L. Lord, William G. Wood, Paul C. Clements "Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design(ADD) Method", CMU SEI-2004-TN-017, 2004.
[12] Andr´e Batista de Oliveira, "Software Architecture for Autonomous Vehicles", 2009.
[13] J. Canny, "A Computational Approach To Edge Detection", IEEE Trans. Pattern Analysis and Machine Intelligence, (8) pp.679-714, 1986.
[14] Todd Litman, "Autonomous Vehicle Implementation Predictions", Victoria Transport Policy Institute, 2015.
[15] B. P. Douglass, Real-Time UML Workshop for Embedded Systems," Elsevier, 2007

행위 모델링 및 조합을 위한 객체지향성

김희언
세종대학교 컴퓨터공학과
hiun@divtag.sejong.edu

Object-orientation for Behavior Modeling and Composition

Hiun Kim
Computer Science Department, Sejong University

요 약

현대 소프트웨어가 가지는 기능의 수와 복잡성은 지속적으로 증가하고 있으며 이를 정확하게 모델링하고 높은 모듈성을 가지도록 구현하는 것이 중요한 요소이다. 에스펙트 지향 프로그래밍은 횡단적 관심사를 에스펙트 객체에 지역화시킴으로써 모듈성을 구현하였지만, 에스펙트 객체 자체의 유연성이 낮아 고차적인 재사용에는 제한적이었다. 본 연구에서는 횡단적 관심사들의 고차적인 재사용성을 확보하기 위해 객체지향성을 소프트웨어 행위의 모델링 및 조합에 적용하는 방법을 제시하며, 행위에 계층적 관계를 부여하는 접근을 통해 추상적인 부모 행위와 구체적인 자식 행위라는 개념을 제안하고, 공통된 횡단적 관심사를 부모 행위에 지역화시켜서 재사용성을 향상시켰다. 이를 유연하게 지원하기 위해 자가조합성이라는 개념을 도입, 다른 행위와의 조합과 함께, 행위 스스로 개선을 통해 고차적인 재사용성을 확보하는 방법을 제안하였다. 우리는 본 연구에서 AOP와 비교하여 웹서비스를 대상으로 본 기법의 효과성을 검증하였으며 행위가 객체와 분리된 존재로 자체적인 계층적 관계를 가짐으로써 더욱 정확하게 모델링되고 쉽게 조합되어 소프트웨어의 모듈화에 기여할수 있다는 것을 제시하였다. 본 연구의 궁극적인 목적은 객체지향프로그래밍의 동기를 따라서 실제 세상의 행위를 프로그래밍 레벨에서 모델링하는 것의 중요성을 자가조합프로그래밍이라는 방법을 통해 제시하며, 그 필요성을 연구자들에게 제시하는 데 있다.

1 서론

현대 소프트웨어는 지속적으로 그 기능(feature)[1]의 수와 기능의 수준이 증가하고 있으며, 결과적으로 하나의 소프트웨어 안에서도 기능간의 가변성(variability)이 증가하였다. 이러한 가변성을 관리하고 지속 가능하게 소프트웨어를 발전시킬 수 있는 요소들로 재사용성(reusability), 유연성(flexibility), 이해성(comprehension)등 이 있으며 [2],. 이러한 요소들을 충족시키기 위해서는 소프트웨어 기능의 모듈성(modularity)은 중요한 요소이다[3]. 프로그래밍 레벨에서 주요한 모듈화 방법인 에스펙트지향프로그래밍(Aspect-oriented Programming; AOP)[4]은 소프트웨어의 기능들과 뭉쳐있고(tangled) 따라서 전역적으로 산포되어(scattered) 있는 공통된 기능인 횡단적 관심사(cross-cutting concern) 모듈화시켜 재사용성과 이해성을 부여하였으나, 유연한 고차적 재사용성의 어려움으로 인해 횡단적 관심사가 일부분만 달라져도 이를 반영하기 위한 직접적 방법이 제한적이라는 것이다. 상속을 통해 이를 해결하고자 하는 방법은 존재하나[5, 6], 직접적인 문법적 지원의 필요성은 여전히 존재한다. 우리는 AOP을 비롯한 기존 방법들과 다르게, 객체를 중심의 사고가 아닌, 독립적인, 객체의 행위를 중심으로 한 사고를 통해 가변성의 문제를 바라보았으며, 소프트웨어의 내부적 행위들에게 관계를 부여하여 유연한 고차적 재사용을 통한 가변성의 관리를 하고자 한다.

1) 본 연구에서 기능(feature)은 Kang et al. [1]의 정의에 따라, '사용자적 관점에서 주요하고 구분될 수 있는 소프트웨어의 품질과 특성 그리고 성질'의 의미로 사용된다. 이와 대조적으로 행위(behavior)는 '소프트웨어의 관점에서 기능이 구현되기 위한 작업'으로 사용되며, 여러 행위가 하나의 기능을 만들 수도 있다.

1.1 현대소프트웨어의 행위적 유사성

우리는 높은 수준의 행위를 하는 현대 소프트웨어를 개발하면서 다양한 가변성을 마주하였다. 현대 소프트웨어를 모듈화하기 어려운 이유는 기능을 구현하는 행위들의 공통성(commonalities)이 적어지고 가변성이 늘어났기 때문이다. 우리는 가변적인 행위들이 공통적이지 않지만 유사(similar)하다는 점을 확인할 수 있었고 이를 행위적 유사성(behavioral similarity)이라 특정하였다. 대표적 예로는 네트워크 연관 소프트웨어가 있다. 시스템 소프트웨어(i.e. OS)와 다르게 이러한 소프트웨어들은 *상대적*으로 많은 종류의 그러나 적은 복잡성을 가진 모듈들의 집합으로 이루어져 있다(i.e. API 서버). 따라서 이러한 모듈들은 네트워크 연관 소프트웨어에서 큰 비중을 차지하며, 이들을 구성하는 횡단적 관심사인 인증이나 캐싱, 데이터 검증 등 하위 행위들이 모듈 간에 비슷한 패턴으로 실행되는 행위적 유사성을 보인다. 또한, 서비스 지향 아키텍처(Service-oriented Architecture; SOA)[7]등 네트워크를 기반으로 느슨하게 연결되어 동작하는 소프트웨어들은 기능의 더 많은 부분을 네트워크상의 다른 소프트웨어에게 의존하게 되므로[3] 이들의 정확한 작동을 위해 더 많은 - 행위적 유사성을 가진 횡단적 관심사들이 비슷한 패턴으로 사용될 것이다. 따라서 네트워크 연관성의 증가와 함께, 행위적 유사성은 가변성과 더불어 현대 소프트웨어의 주요한 성질로 자리 잡게 될 것이다.

1.2 애스펙트지향 접근법

현대 소프트웨어의 가변성을 관리하기 위해, AOP는 하나의 모듈을 핵심 관심사(core concern)와 횡단적 관심사로 분해(de-

compose)하고, 소프트웨어의 모듈 곳곳에 산포되어있는 횡단적 관심사를 함수와 해당 관심사의 연결 지점(pointcut)정보를 가진 에스펙트 객체로 모듈화시켜서, 이를 전후처리 등의 접합점(join point)에 조합하여 사용하는 방법을 제안하였다[8]. 웹서비스를 예로 들면, 핵심관심사는 *글쓰기*나 *댓글쓰기*와 같은 사용자가 사용하는 기능이 되며, 횡단적 관심사는 인증이나 검증과 같이 핵심 관심사의 정확하고 원활한 수행을 도와주는 내부적인 기능들이 된다. 이러한 모듈화는 프로그램의 자가 조작을 위한 방법인 메타프로그래밍(metaprogramming)을 통해서도 가능하나, AOP가 기여한것은 쉽고, 안전하고, 관리가능한 모듈화를 위한 직접적인 어의(direct semantical)를 제공한 것이다[9].

1.3 객체지향성의 본질

객체지향프로그래밍(Object-oriented Programming; OOP)[10]이 제안한 객체지향성(Object-orientation)은 컴퓨터가 실제 세계의 사물(thing)을 모델링하여 궁극적으로 그들 간의 교류(interaction)를 쉽게 표기하기 위한 프로그래밍 및 설계의 방법(paradigm)을 직접적인 어의를 통해 제공하여 준다. AOP는 이러한 객체들의 행위 사이에 횡단적으로 존재하는 관심사를 캡쳐하여 에스펙트 객체에 지역화(localisation)시킴으로써, 높은 모듈성을 부여해 주는 방법이다. 하지만 AOP는 여전히 객체 중심적 사고를 기반으로 횡단적 관심사를 모듈화 시킨 것이기 때문에 *객체의 재사용*은 가능하지만, 어드바이즈함수[11]와 같은 함수, *즉 행위의 재사용*은 어렵다. 1.1장에서 현대 소프트웨어의 행위는 복잡하기 때문에 하나의 행위가 여러 개의 하위 행위로도 구성될 수 있다고 하였는데, 단순히 행위를 재사용하는 것을 넘어서 유사한 패턴으로 사용되는 행위들의 집합을 재사용할 필요성이 대두되었다. 본 연구는 객체지향성의 핵심 아이디어인 객체의 원형인 클레스와 그 실제(realisation)인 인스턴스라는 개념들을 살펴보고, 이에 계층적 관계(hierarchical relationship)를 부여하여, 객체의 상속과 개선(refinement)을 통한 구체화(specification)가 소프트웨어의 복잡성을 감소시켜 준 것과 같이, 동일한 계층적 관계를 소프트웨어의 행위에도 적용시키고자 한다. 객체지향성을 행위에 적용시키기 위해, 우리는 행위가 가져야 할 속성을 자가조합성(Self-composability)이라고 명명하고 자가조합적 관점을 기반으로 프로그래밍 및 설계 그 방법으로 자가조합프로그래밍(Self-composable Programming)을 소개한다. 자가조합프로그래밍은 OOP의 클레스에 해당하는 *추상 행위*를 생성하고, 이를 상속 및 개선하여 구체화와 실제화를 할 수 있는 *구체행위*를 제안한다. 자가조합프로그래밍은 이러한 생성, 상속, 개선의 과정을 직접적인 어의로 제공하기 위한 2가지 기능인 자가조합성(self-composability)과 다단계 상속(multi-level inheritance)을 프로그래밍 레벨에서 제공한다.

1.4 자가조합적 접근법

자가조합프로그래밍은 AOP와는 다르게, 행위에 관계를 부여하여 횡단적 관심사의 모듈화를 시키는 방법을 취한다. 객체지향프로그래밍을 처음 선보인 언어들이 세상에 있는 객체들의 교류를 모델링 하기 위해 접근한 것처럼, 자가조합프로그래밍은 행위를 객체에 의존된 객체지향적인 관점으로 보지 않고, 독립적 존재로 취급한다. 따라서 객체를 상속하는 것처럼 행위를 상속할 수 있으며, 결과적으로 행위에 계층적 관계가 부여되어 추상적인 행위(부모 행위)를 상속하고, 개선하여 구체적인 행위(자식 행위)를 만드는 것이 가능해진다. 이때, 부모 행위에 횡단적 관심사가 모듈화되며, 자식 행위는 이를 자가조합하여 개선해나가면서 최종적으로 원하는 기능을 만들어간다. 즉, 부모 행위는 해당 작업에 대한 도메인 특정(domain-specific)한 재사용 가능한 패턴[12]을 만들어준다고 볼 수 있다. 본 논문의 나머지 부분은 자가조합성의 개념과 자가조합프로그래밍의 구현체인 Self-js[13]를 소개하며, 요구사항 분석 및 설계 단계에서 이러한 행위들의 관계 모델링을 위한 자가조합적 도메인 분석을 웹서비스를 예로 소개한다. 또한, 행위의 자가조합을 위한 5가지의 메서드를 이용한 직접적인 개선(explicit refinement)방법과 이를 고도화 시킬 Traits[14]등의 방안들을 소개한다. 마지막으로 웹서비스를 대상으로 자가조합 프로그래밍방법을 평가하고, 결과 및 개선점 그리고 한계점과 미래 연구 방향을 소개한다.

2 자가조합개념

2.1 조합의 의미

자가조합프로그래밍에서 조합(compose)은 행위 조합을 의미하며, UNIX 철학[15]과 비슷하게, 더욱 고수준의 문제를 해결하기 위해, 이미 조합된 행위가 더 높은 차원의 행위의 일부분으로 조합될 수 있는 고차적인 조합성도 포함한다. 메시징 서비스에서 메시지를 보내는 행위는, 메시지를 보내는 핵심 관심사와 데이터를 검증하고 저장하는 횡단적 관심사로 이루어져 있다. 만약 우리가 (a)파일을 공유하고 (b)자동으로 메시지를 보내는 기능을 만들고자 한다면, a행위와 와 b행위를 조합하여 (c)파일 공유하기라는 고차원의 행위로 바뀔 수 있다. 현대소프트웨어는 다양한 차원의 행위들을 필요로 하며, 이러한 고차원적 조합성을 직접적인 문법으로 지원하는 것은 AOP등 기존 방법들과 비교하여 효과적인 모듈성을 제공해준다.

2.2 행위의 자가조합성

행위의 자가추가 : 행위를 만들기 위해서 프로그래머는 저수준의 행위(하위 행위)를 조합하여 만들 수 있는 요소이다. 메시지 보내기는 인증, 로깅, 검증 등의 하위 행위로써 조합될 수 있으며, 이러한 조합을 통해 상속을 위한 클래스, 즉 추상 행위를 만들 수 있다.

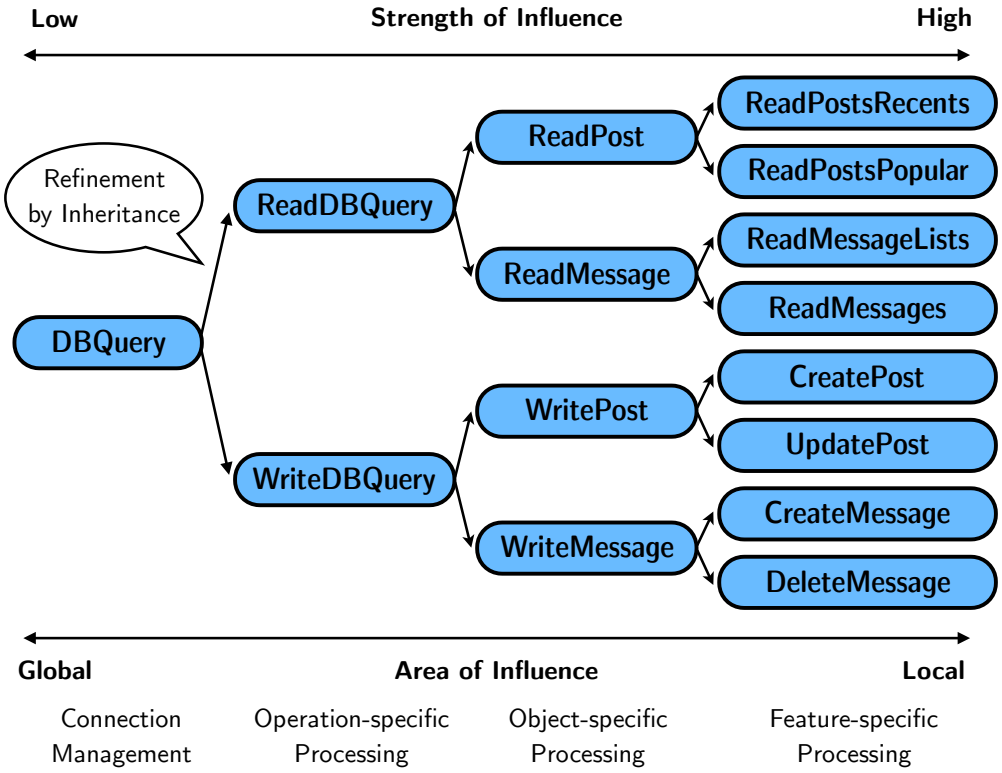


그림 1: 웹서비스 기능들을 구성하는 행위의 다단계 상속

행위의 자가업데이트 : 자가업데이트는 행위의 일부분을 업데이트하는 것으로, 각 행위가 하위 행위들로 구성되어진 점을 활용하여 일부의 하위 행위를 업데이트하는 것을 말한다. 물론 하위 행위 역시 또 다른 하-하위 행위들로 구현될 수 있으므로, 더욱 디테일한 업데이트도 가능하다. (e.g. 메시징 서비스에서 특정 모듈에 새로운 인증방법이 필요한 경우 인증에 관한 모듈을 업데이트하는 경우)

행위의 자가제거 : 특정한 하위 행위를 제거하는 것으로, 자가업데이트와 동일하게 적용 가능하다. (e.g. 공통적으로 적용되는 인증 기능을 없애서 퍼블릭한 API를 만드는 경우)

행위의 자가조작 : 행위의 반복 등 다양한 조작을 가할 수 있는 자유로운 조작 모드이다.

2.3 행위의 다단계 상속

자가조합된 행위는 조합과 개선 그리고 실행의 흐름을 가지고 소프트웨어 안에서 사용된다. 자가조합성이 개선을 위해 사용되는 것이라면, 다단계 상속은 개선할 대상을 만들어주는 역할을 한다. OOP에서 상속은 개선을 통한 객체의 구체화를 목적으로 하는 것처럼, 자가조합프로그래밍은 상속은 개선을 통한 행위의 구체화를 의미한다. 데이터베이스를 이용한 데이터 접근 모듈을 예로 들면, 그림 1이 나타내는 것처럼, 각 행위는 다른 행위와 관계를 가진다. 행위들은 같은 수준에서의 관계도 가지지만, 공통된 횡단적 관심사를 가진 상위의 추상 행위 및 자신을 상속받아 더욱 구체화된 하위의 구체행위와도 관계를 가진다. 본 예제에서는 웹서비스의 기능의 핵심 관심사인 데이터베이스 작업을 위한 4단계의 횡단적 관심사의 도메인을 정의하였다. 연결 관리(connection management)를 시작으로 작업형태에 관련 처리(operation-specific processing), 작업 대상인 객체관련 처리(object-specific processing), 최종 기능관련 처리(feature-specific processing)순으로 횡단적 관심사의 근본성(fundamentality)을 척도로 계층적으로 정의되었다. 가장 추상적인 부모 행위인 DBQuery는 모든 행위에 필요한 횡단적관심사를 모듈화하며, ReadDBQuery및 WriteDBQuery는 읽기와 쓰기라는 작업에 대한 모듈화를 진행한다. 그다음 ReadPost, ReadMessage등은 Post와 Message라는 객체에 대한 처리를 모듈화시키며 마지막 열의 ReadPostRecents 및 ReadPostPopular는 최종 기능의 횡당관심사에 대한 모듈화를 진행한다. 추상적인 행위일수록 글로벌한 영향력을 끼치지만, 그 강도는 낮고, 구체적인 행위일수록 강력하지만, 지역적인 영향력을 끼친다. 화살표가 의미하는 것은 각 행위가 보다 추상적이고 범용적인 행위를 상속받고, 해당 횡단적 관심사 도메인에 필요한 하위 행위를 넣어 개선되는 것을 의미한다(e.g. DBQuery는 일반적인 데이터베이스 모듈을 상속받아 작업 형태에 특정인 읽기 쓰기모듈로 개선시키는 이러한 작업).

앞서 언급한 행위의 계층성은 실제 조직의 의사결정 구조와 비

슷한 점이 많은데, 조직의 근본적인 책임을 지는 최고 책임자, 본 예제에서는 연결을 관리하는 (DBQuery)는 실제 일을 하는, 기능 관련 말단 조직원(ReadPostRecents)들에게 영향을 끼치기는 하지만 그들 개개인에 대한 영향력의 중요도 작다. 하지만, 그 말단 조직원의 직속 책임자(ReadPost)는 그 영향력이 지역적이지만 그 아래의 조직원들에게는 강력하다. 따라서 최종행위는 다양한 상위 행위들의 *어드바이스*의 영향을 받으면서 정의되며, 이러한 어드바이스가 최종 행위를 변경할 수 있다[11]. 행위의 계층적 관계를 아키텍처에 적용하거나 프로그래밍 레벨에서 이용하면 분산된 주체들이 협업하여 기능을 만드는 SOA 및 기타 대규모 시스템에서의 행위를 보다 정확하게 모델링 할 수 있다고 판단된다.

2.4 자가조합적 도메인 분석

자가조합 도메인 분석은 그림 1과 같이 요구사항 분석 단계에서 기능들의 수준을 기반으로 핵심적인 횡단 관심사의 도메인을 정의하고, 도메인을 기반으로 행위의 상속 단계를 설정한 후 소프트웨어를 자가 조합적으로 설계하는 것을 의미한다.

3 Self-js : 프로토타입 구현체

3.1 개요

Self-js[13]는 자가조합성의 구현체이며, 실제 소프트웨어 개발을 위해 행위의 자가조합성과, 다단계 상속을 지원하는 행위를 생성할 수 있는 구현체이다. 우리는 이 컨셉을 구현하기 위한 매개체(medium)로 OOP를 선택하였다[2]. OOP는 행위 스스로 자가조합을 진행하기에 적합한 메서드(method) 표기법을 지원하고, 상속을 지원한다. 우리는 구현체의 언어로 OOP와 함수형 프로그래밍을 지원하는 JavaScript를 선택하였다.

3.2 디자인 및 구현

Self-js는 JavaScript라이브러리로써 함수 생성자와 비슷하게 사용할 수 있다. Self-js를 사용하여 생성한 행위객체 안에는 하위 행위들이 직렬화되어 들어가 있는 배열과 그것을 조작하기 위한 표 1과 같은 메서드들을 가지고 있다. 각 메서드들은 공통적으로 1개의 원시적(primitive)함수나 하위행위로 포함될 또 다른 행위 객체를 인자로 받는다. 따라서 최종 행위적으로 개선된 행위의 실행은 한 최초 인자와 함께 최초 하위 행위를 실행시키고, 해당 하위 행위의 결과값을 다음 행위의 인자로 하여 실행을 반복한다는 형태로 진행된다. 각 하위행위의 파라미터의 경우, 기본적으로 범용성을 가지도록 만들 수도 있으나, 차 후 소개할 map메서드를 사용하여, 하위행위와 인터페이스를 분리할 수도 있다. 이러한 점에 기인하며, 각각의 하위행위가 독립적으로 사용될 수 있으므로, 행위들이 원자 적이고(atomic) 보편적인 빌딩 블록(universal building block)으로써, 여러 형태로 사용될 수 있다.

2) OOP가 대중화된 절차지향형 프로그래밍 위에 만들어진 것처럼, 우리는 자가조합프로그래밍을 OOP를 기반으로 만들었다.

표 1: Self-js의 메소드 목록

메소드 이름	설명
Behavior#add	지정된 행위에 입력된 함수 또는 행위를 덧붙인다(append).
Behavior#Behavior#before	지정된 하위 행위 앞에(before) 입력된 함수 또는 행위를 삽입한다.
Behavior#Behavior#after	지정된 하위 행위 다음에(after) 입력된 함수 또는 행위를 삽입한다.
Behavior#Behavior#update	지정된 하위 행위를 입력된 함수 또는 행위로 업데이트한다.
Behavior#Behavior#delete	지정된 하위 행위를 삭제한다.
Behavior#Behavior#map	지정된 하위 행위를 입력된 함수 또는 행위의 인자로 받아 조작한다(manipulate).

[a] 모든 메서드는 인자로 1개의 원시적(primitive) 함수나 Self-js에 의해서 생성된 행위 객체를 받는다.

4 Self-js의 사용

본 항목에서는 에서는 Self-js의 사용 개요를 코드레벨에서 다루며, 행위의 상호용운용, 개선, 상속에 대한 Self-js의 내부 수행 메커니즘에 대해 다룬다.

4.1 Self-js 한눈에 보기

Listing 1에서는 자가조합행위의 전체적인 라이프사이클을 보여준다.

4.2 행위의 생성

```
1  var Behavior = require('self');
2
3  var DBQuery = new Behavior();
4
5  DBQuery.add(auth);
6  DBQuery.add(validate);
7  DBQuery.add(monit);
```

Listing 2: 자가조합형 행위의 생성

Listing 2는 Self-js를 이용한 행위의 생성에 대해 다룬다. 먼저 Self-js는 범용 JavaScript 런타임인Node.js[16]의 require문을 통해 불러와 사용되어지며, behavior라는 변수에 할당하였다. behavior 객체는 생성자로써 데이터와 메서드 2개의 부분으로 이루어진 DBQuery 인스턴스를 생성한다. 데이터에는 네이티브 함수나 또 다른 Behavior 인스턴스들이 들어갈 수 있으며, 본 예제에서는 .add 메서드를 실행하여 횡단적관심사 인증(auth), 검증

```
/* CONSTRUCTION PART */

// define self-js
var Behavior = require('self');

// initialising behavior
var DBQuery = new Behavior();

// adds some sub-behaviors
DBQuery.add(auth); // authentication checker
DBQuery.add(validate); // data validation
DBQuery.add(monit); // monitoring


/* REFINEMENT PART */

// inherit DBQuery to operation-specific,
    WriteDBQuery
var WriteDBQuery = new DBQuery();

// add some sub-behaviors (refinements)
WriteDBQuery.add(writeBack);

// update specified sub-behavior to new sub-
    behavior
WriteDBQuery.monitoring.update(cacheMonit);

// add sub-behavior in specified location
WriteDBQuery.validate.before(beforeValidate);
WriteDBQuery.validate.after(afterValidate);

// manipulating sub-behavior
WriteDBQuery.validate.map(() => {
return (validate) => {
  validateWrapper(validate);
}
});
//delete sub-behavior
WriteDBQuery.beforeValidate.delete();


/* ADDITIONAL REFINEMENT */

// inherit WriteDBQuery to object-specific
    query
var CreatePost = new WriteDBQuery();
var CreateMessage = new WriteDBQuery();

CreatePost.add(createUserSQLExec);
CreateMessage(createMsgSQLExec);

//additional modification
CreatePost.auth.update(2factorAuth);
CreateMessage.auth.before(geographicalBlock);
```

Listing 1: 자가조합형행위의 라이프사이클

(validate), 모니터링(monit)과 같은 횡단적 관심사를 추가하였다.

4.3 행위의 상속

```
/* Operation-specific Processing */
var ReadDBQuery = new DBQuery();
var WriteDBQuery = new DBQuery();

// ...some refinement


/* Object-specific Processing */
var ReadPost = new ReadDBQuery();
var ReadMessage = new ReadDBQuery();
var WritePosts = new WriteDBQuery();
var WriteMessage = new WriteDBQuery();

// ...some refinement


/* Feature-specific Processing */
var ReadPostsRecents = new ReadPosts();
var ReadPostsPopular = new ReadPosts();
var ReadMessageLists = new ReadMessage();
var ReadMessages = new ReadMessage();
var CreatePost = new WritePost();
var UpdatePost = new WritePost();
var CreateMessage = new WriteMessage();
var DeleteMessage = new WriteMessage();
```

Listing 3: 자가조합형행위의 다단계 상속

Listing 3는 new 키워드를 사용한 다단계 상속에 대해서 다룬다. 본 Listing에서는 3단계 상속을 통해 각 행위가 여러 횡단적 관심사를 개선을 통해 지역화하며. 상속해나간다. 상속 시 내부적으로는 새로운 Behavior 인스턴스를 생성한 뒤, 데이터와 메서드들을 연결시키고(JavaScript에서는 Prototype 체인을 연결)하고 최종적으로 만들어진 인스턴스를 반환하는 방법으로 진행된다.

4.4 행위의 개선

```
var WriteDBQuery = new DBQuery();

WriteDBQuery.add(writeBack);
WriteDBQuery.monitoring.update(cacheMonit);
WriteDBQuery.validate.before(beforeValidate);
WriteDBQuery.validate.after(afterValidate);
WriteDBQuery.validate.map(() => {
return (validate) => {
  validateWrapper(validate);
```

```
10  }
11  });
12  WriteDBQuery.beforeValidate.delete();
13
14  var CreatePost = new WriteDBQuery();
15  var CreateMessage = new WriteDBQuery();
16
17  CreatePost.add(createUserSQLExec);
18  CreateMessage(createMsgSQLExec);
19  CreatePost.auth.update(2factorAuth)
20  CreateMessage.auth.before(geographicalBlock)
```

Listing 4: 자가조합형행위의 개선

Listing 4에서는 Listing 3에서 생성한 DBQuery 개선하여 WriteDBQuery를 만들고, 이를 다시 개선하여 CreatePost와 CreateMessage를 만든다. 개선은 자가조합프로그래밍이 지원하는 다양한 메서드를 통해 직접적으로 진행된다. 추가적인 하위행위들은 add메서드를 통해 덧붙여질 수 있으며, 특정 행위 전후에는before와 after메서드를 사용함으로써 상대위치에 추가할 수 있다. map 메서드는 하위행위들을 새로운 함수영역 안에서 조작할 수 있다. 이 경우, validateWrapper함수의 인자로 입력되었다. Self-js의 각 메서드는 개선의 편의를 위해 하위 행위를 객체의 속성으로 노출시키며, 노출된 행위안에 각각의 개선 메서드들이 작동된다. 하위행위의 이름을 키로 실제 해당 행위가 들어있는 배열에서의 위치를 특정할 수 있으므로, 이 후부터는 일반 배열에 조작을 가하는 것과 동일하게 작동된다.

4.5 기타개선방법

자가조합프로그래밍이 객체지향형 개선을 사용하기 때문에, traits와 같은 기타 OOP 기술을 사용하여 자가조합성을 지원할 수도 있다. 직접적인 개선은 강력하지만 큰 규모 개선의 경우 위험성과 이해성의 측면에서 좋지 않은 면도 존재한다. traits는 대상에 독립적인 행위의 집합으로써[14], 보다 고수준의 간접적인(implicit) 개선을 가능하게 한다.

4.6 행위의 실행

```
1  CreateMessage.exec(Handler);
```

Listing 5: 자가조합형행위의 실행

조합된 최종 행위는 exec메서드를 통해 실행되며, 최종결과물 역시 윈시적인 JavaScript 객체이기 때문에 다른 언어적 지원 없이 쉽게 독립적인 모듈로써 사용될 수 있다.

5 평가 및 분석

5.1 목적

본 섹션에서는 Self-js를 이용한 프로그래밍 및 설계의 모듈성의 평가 및 분석에 대해서 다룬다. 프로그래밍기법은 정량적인 효과만으로 유효성의 검증이 끝나는 것이 아니라 이해성이나 설계에도 영향을 끼치기 때문에 실증적(empirical) 자료가 부족한 새로운 기법의 경우 그 효과를 평가하기가 어렵다. 따라서 본 평가에서는 Self-js를 적용하여 개선된 점들을 기반으로, Self-js의 어떤 점이 모듈성의 측면에서 효율성을 가져다주는지 특정하는 것을 목표로 진행한다. 본 평가에서는 웹 서비스의 핵심 모듈들을 자가조합프로그래밍 및 AOP를 이용하여 고수준에서 구현함으로써[3] 모듈화 정도를 측정하고, 이를 통해 얻은 추세함수를 바탕으로 회귀분석을 통해 프로그램이 확장에 따른 코드량(SLOC)의 증가를 예측, 재사용성을 분석한다. 본 평가의 목적은 Self-js의 기본적인 모듈화 강점인, 재사용의 매계체인 애스펙트 자체를 얼마나 고차적인 재사용을 할 수 있는지를 측정하는 것이 목적이다.

5.2 첫번째 분석 : 새 기능당 요구되는 코드량

첫 번째 분석에서는 User와 Post에 대한 데이터베이스의 Read 작업을 하는 행위를 가진 모듈을 구현하였다. 총 4단계의 상속을 거치며 표 2에 있는 최종 기능 8개를 개선해나간다. 최종기능들의 횡단적 관심사는 인증 기능에 의해 가변성을 가지며, 이를 제외한 부분은 공통성을 가진다. 자가조합적 도메인 분석을 통해 본 기능들의 횡단적 관심사를 작업별, 객체별, 기능별, 형태별(Post에만 적용)의 4단계로 나누었으며 이 구조도는 그림 2 와 같다. 우리는 부록 AOP와 자가조합프로그래밍으로 각 기능을 구현하였고, 각 프로그램의 소스코드와 jsAspect[17]라이브러리 기반의 JavaScript AOP용 레퍼 함수의 소스코드는 부록 A, B, C에 있다. 각 기능별 소요된 SLOC를 산출하기 위해, 우리는 SLOC를 2가지 목적으로 분류하였다.

첫 번째로 최종 기능을 구현하기 위한, 황단적관심사들을 통합해주는 모듈에 사용된 SLOC를 측정하였다. AOP의 경우 에스펙트 객체를 생성하는 부분이 될 것이고, 자가조합프로그래밍의 경우 1 3차까지의 행위 생성 및 개선하는 부분이 될 것이다. 두 번째로 최종적으로 만들어진 횡단적 관심사들의 집합을 핵심 관심사와 조합하는 코드가 된다.

5.2.1 AOP 기반 구현체의 분석

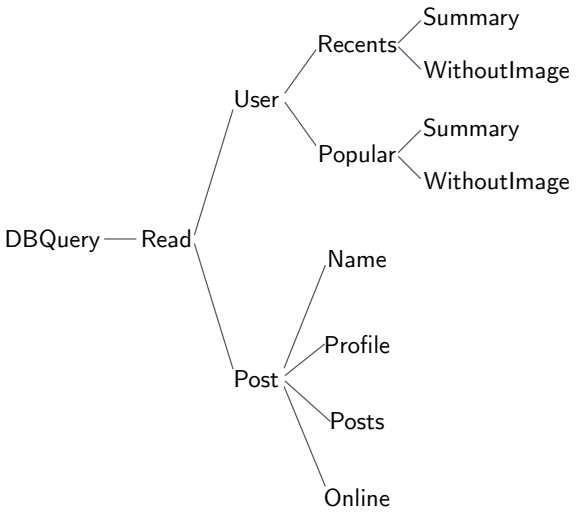
AOP에서 에스펙트 객체를 생성을 위한 내용이 되는 코드는 부록 A의 라인 1~5, 8~12, 21~27, 29~36 총 26줄(표 3의 a)이며, 그 중 실제 횡단적 관심사에 대한 코드는 2-5, 9-11, 22-26, 30-35

3) 본 평가에서 요구되는 기능이 간단한 관계로 AOP에서는 before 어드바이스만을 사용하였다. 또한 양 기법 모두 내부 구현을 생략한 고수준에서의 구현을 가정하였다.

표 2: 첫번째 분석용 소프트웨어의 기능 목록

기능명	설명
User.getName	인증 후 지정된 사용자 이름 받기
User.getProfile	인증 후 지정된 사용자 프로필 받기
User.getPosts	인증 후 사용자의 포스트들 받기
User.getOnline	인증 없이 사용자의 접속 여부 받기
Post.getRecentSummary	인증 후 최신 포스트 요약 받기
Post.getRecentsWithoutImage	인증 후 최신 포스트 글만 받기
Post.getPopularSummary	인증 없이 인기 포스트 요약 받기
Post.getPopularWithoutImage	인증 없이 인기 포스트 글만 받기

그림 2: 첫번째 분석용 소프트웨어의 행위 관계도



*4단계 상속을 거친 행위의 구조도는 위와 같다. 최종 기능의 이름은 왼편부터 끝 노드까지의 이름을 붙혀서 확인할 수 있다.

표 3: 하나의 기능을 구현하는데 필요한 평균 SLOC 비교

	AOP	자가조합
재사용 매계체 선언에 사용된 SLOC(a)	26	14
횡단적 관심사의 SLOC(b)	18	6
기능의 갯수	8	
하나의 기능 구현시 횡단적 관심사의 평균 SLOC (b/8)	2.25	0.75

18줄(표 3의 b)이다. 에스펙트 객체 생성을 위한 코드를 보면, 이들의 하위 행위인 logging, auth, cacheLookup, userIdValidation 등의 행위들이 ReadUser, ReadUserWithAuth, ReadPost, ReadPostWithoutAuth 모듈에 유사한 패턴으로 산포되어 있는 것을 알 수 있으며, auth에 의한 가변성 때문에 Aspect가 이를 효과적으로 재사용하지 못하여 이러한 하위 행위가 중복되게 호출되는 것을 확인할 수 있다.

5.2.2 자가조합 기반 구현체의 분석

자가조합프로그래밍은 프로그램의 이해성에 영향을 주지 않으면서도 중복호출에 의한 코드의 재사용성 하락을 피하기위해, 평평한(flatten) 조합을 하는 AOP와는 다르게, 계층적이고 보다 고수준의 조합을 보여준다. 횡단적 관심사를 통합해주는 User에 관한 1~2차 까지의 행위 생성 및 개선이 라인3, 5, 7, 14총 4줄이며, Post에 관한 행위 생성 및 개선이 라인15, 19 총 2줄로 총 6줄(표 3의 b)이다. 또한, 최종 단계의 개선에 사용되는 코드는 라인 9~12, 16~17, 20~21 총 8줄이며, 이 2개를 합하여 재사용 매계체 선언에 사용된 SLOC는 14줄(표 3의 a)이라 할 수 있다.

5.2.3 결과 분석

표 3에서 하나의 기능을 구현하는 데 필요한 횡단적 관심사의 평균 SLOC는 AOP의 경우 2.25으로 나타났고, 자가조합프로그래밍의 경우 0.75으로 나타났다. 즉 AOP를 사용하면 하나의 기능당 2.25개의 횡단적 관심사가 명시적으로 사용되는 반면, 자가조합프로그래밍의 경우 0.75개만 사용되는 것으로 상당히 고무적인 결과이다. 추가적으로 OOP가 데이터의 캡슐화(encapsulation)를 통해, 정보 감춤(information hiding)를 하는 것처럼 자가조합프로그래밍은 행위를 구성하는 하위 행위들의 캡슐화를 통해 정보 감춤을 하게 된다. 이의 장점과 단점 역시 이해성인데, 고수준의 사용이 가능한 점이 있지만, 이러한 개선 작업이 산포될 경우 프로그래머가 디테일한 동작을 알기 어렵다는 것이다. 또한 현재의 직접적인 개선은 행위의 정확성(correctness)를 깨트릴 수도 있으므로, IDE지원이나 문서화등의 외적인 방법과 함께, 간접적인 개선 방법과 이를 지역화시키는 방법에 대한 추가 연구가 필요하다. 마지막으로 OOP 역시 객체 간 협업(Object-oriented Collaboration)에서 협업의 내용과 단계가 늘어남에 따라 프로그래머가 소프트웨어의 동작을 직접적으로 정의하거나 확인할 방법이 없다는 점이 대두하였고, 이러한 협업의 과정을 보다 명료하게 만들기 위해, OOP 환경에서 데이터와 인터렉션을 분리하고, 둘을 연결하는 컨텍스트를 만드는 DCI architecture가 제안되었다[18]. 행위와 하위행위의 조합 역시 이러한 컨텍스트상에서 조합이 가능하다면 사용자가 행위의 내부를 파악하는데 도움이 될 것이라고 판단되며, 이러한 아키텍처의 필요성을 확인할 수 있었다.

표 4: 자가조합프로그래밍의 행위 상속 단계 별 SLOC 필요량

행위 상속 단계 (차수)	부모행위의 갯수	자식행위의 갯수	개선용 SLOC	전체 SLOC
1차	1	2	5	10
2차	2	5	5	50
3차	5	10	5	250
4차		*(추정치)*		1,250
5차		*(추정치)*		6,249

표 5: AOP의 행위 상속 단계 별 SLOC 필요량

행위 상속 단계 (차수)	부모행위의 갯수	자식행위의 갯수	개선용 SLOC	전체 SLOC
1차	1	2	10	20
2차	2	5	15	150
3차	5	10	20	1,000
4차		*(추정치)*		7,211
5차		*(추정치)*		50,988

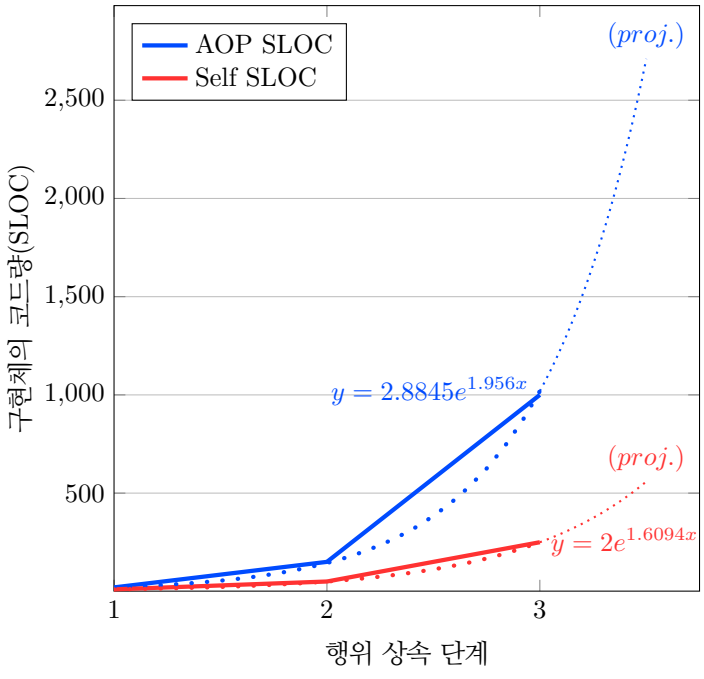


그림 3: 행위의 상속 수준에 따른 코드량의 증가 예측

5.3 두번째분석 : 새 기능당 재사용되는 코드량 추정

두 번째 분석은 AOP및 자가조합프로그래밍을 이용하여 가변성을 가진 기능이 고도화시키면서 어느 정도의 SLOC가 필요한지 측정하는 방법이다. 본 분석의 목적은 첫 번째 분석을 보다 대규모에서 시뮬레이션하여 행위의 개선 단계별로 어느정도의 SLOC 재사용되는지 측정하는 것이다 첫 번째 분석의 규모를 키워 가변성을 가진 소프트웨어의 기능의 수가 고도화되어감에 따라 추가적으로 필요한 소스코드의 양을 예측하여 재사용성을 측정하는 방법이다. AOP에서도 상속을 통해 에스펙트 객체를 재사용하는 방법이 있으나 직접적으로 고차적 재사용을 지원하지 않는다[5, 6]. 본 분석에서는 가상의 웹서비스의 행위를 3단계의 상속을 통해 구체화 시킨다, 따라서 각 단계에 사용되는 SLOC는 부모 행위의 수와 자식 행위의 수 그리고 행위별 사용된 소스코드량 3개 항목의 곱으로 산출된다. 우리는 본 실험에서, 상속을 통해 각각 2, 5, 10개의 자식 행위를 생성하고, 각 상속 시마다 5라인의 코드를 추가하여 요구사항 만족을 위한 개선을 진행한다. AOP의 경우 1차적 재사용만 가능하기 때문에 첫 번째 분석의 부록 A와 같이 기존 에스펙트와 행위적으로 유사한 함수의 실행패턴을 가진 새로운 에스펙트를 만든다. 결과적으로 표 5과 같은 SLOC의 증가량을 보이게 된다. 부록 C의 자가조합프로그래밍의 경우 행위자가 직접(explicit) 하게 실제 바뀐 부분만을 조작함으로써 프로그램 수정의 유연성을 높여 결과적으로 최소한의 비용으로 가변성을 달성하게 한다. 표 5상에서 개선에 필요한 소스코드가 5라인으로 유지되는 것이 이를 증명한다.

5.4 결과 및 개선점

표 4, 5를 기반으로 추세함수를 도출한 결과, 그림 3와 같이 소프트웨어의 행위 수준의 증가에 따른 코드 수의 증가를 효과적으로 억제(suppress)시켜주는것을 알 수 있다. 즉 자가조합프로그래밍은 소프트웨어의 가변성을 효과적으로 관리하여, AOP등의 기존 방법으로는 지수 배로 증가하는 SLOC을 실제 필요한 부분만 조작할 수 있도록 지원하는 것을 확인할 수 있었다.

6 관련 연구

AOP[4]를 비롯하여 GenVoca[19], subject-oriented programming[20], adaptive plug-and-play components[21], role components[22] 등 관심사 분리를 통해 소프트웨어의 가변성에 대응하고자 다양한 기술들이 존재한다. 자가조합프로그래밍이 모듈화에 있어서 5의 결과처럼 기존 방법들과 비교하여 나은 성능을 보여주었지만, 이는 극히 성능의 비교가 목적이 아닌, 효율성을 가지는 부분을 코드 레벨에서 확인하기 위함이었다. 자가조합프로그래밍이 기존 방법과 다른 부분은 코드 생성기나 IDE및 툴지원 없이 코드 레벨에서 OOP의 아이디어를 활용하여 행위들의 관계를 유연하게 모델링 하는 방법을 제공한 것이다.

자가조합프로그래밍은 소프트웨어의 행위를 객체에서 분리하여 생각하는 관점을 제공함으로써, 객체에 독립적으로 행위만의 계층구조를 만들 수 있는 생각의 틀(framework of thinking)을 제공하였다. 이를 프로그래머들에게 자신들이 개발할 소프트웨어의 기능을 자가조합적 도메인 분석을 통해 모델링 할 수 있는 방법을 제안하였다. 자가조합프로그래밍의 최종목적은 라이브러리 또는 프로그래밍언어의 내장 기능으로 동작하며, 가변성을 가진 모듈을 쉽게 만드는 것이며, 따라서 필요에 따라 OOP는 물론, AOP와도 상호보완적으로 사용 될 수 있다. AOP가 프로그래머에게 애스팩트적 사고(aspectual tinking)를 제안하여 AOP구현체의 사용과 별개로 소프트웨어를 설계하는데 하나의 방향을 제시한 것처럼[23], 자가조합프로그래밍 역시 프로그래머들이 소프트웨어의 행위를 바라보는 새로운 관점을 제시하고자 하는 것이 목적이다.

7 논의

7.1 구현형태

현재 자가조합프로그래밍은 라이브러리 형태로 제공되나, 문법 및 처리 측면에서 언어에 내장시킨다면 더욱 효율적인 처리가 가능하다.

7.2 한계점 및 미래 연구 방향

현재 자가조합프로그래밍이 가지는 주요한 한계점은 OOP의 한계점과도 흐름을 같이한다, 다단계 상속과 여러번의 개선을 통해 횡단적관심사를 지역화시킬수 있었지만, 그 개선과 상속과정이 산포될수 있다는 문제가 있다. 이에 의해 최종 사용자는 행위의 내부를 정확하게 파악하는데 어려움을 겪을수 있다. 따라서 개선과 상속과정 자체를 더 표현력있게 만드는 방법이 연구되어야 하며, 이를 위해 OOP의 traits나 mixin[24]등 간접적 조합 방법들을 행위에 적용시키는것이 필요하다. 또한 이러한 작업을 효과적으로 진행하기위한, MVC와 같은 설계 패턴(architectural pattern)이 만들어 질 필요가 있다. MVC는 프로그래미어의 생각 모델(mental model)과 컴퓨터의 데이터 모델을 분리하는 방법으로[25], 개선과 상속의 산포를 방지하기 위해 이러한 분리를 목적으로 DCI[18]와 같은 설계 패턴이 만들어질 필요가 있다. 앞의 실험결과가 보여주듯이, 자가조합프로그래밍은 대규모의 가변성을 가진 소프트웨어에 유용하나, 이러한 시스템을 자가조합하게 리펙토링(refactoring)하는것은 비용 문제가 크다. 따라서 기존 시스템을 자가조합하게 만들어줄 수 있는 프로그램 변환기법이나 런타임에서 동작하는 레퍼나 슬라이싱 방법들이 필요하다. 표기법의 측면에서는 도메인 특정한 행위 생성성 및 표기법(e.g. .add 대신, .addValidation)을 제공함으로써 범용 표기법에 대비해 처리 및 전달상의 용이성을 높일 수 있다. 또한, 자가조합프로그래밍을 패키지 관리 시스템과 연계시켜, 고수준의 기능을 빠른 프로토타이핑을 목적으로 하는 시스템 개발도 가능할 것이다. 이러한 다양한 응용은 자가조합프로그래밍이 기존에 통용되고 있는 OOP를 기반으로 만들어져서 다른 기법들과 다른 장점이지만 동시에 OO의 표현적 한계를 가지고 있다. 따라서 본 연구가 제시한 소프트웨어의 가변성 문제를 근본적으로 해결하기 위해서는 행위를 모델링 위한 전용 표기법 및 이의 정확성을 검증하기 위한 방법들이 개발되어야 한다. 또한, 행위 간의 조합을 원활하게 하려면 행위로부터 생성된 기능 교류 문제(feature interaction problem) 역시 해결되어야 한다[26]. 앞서 1장에서 언급한것처럼 본 연구의 최종 목적은 객체지향프로그래밍의 방식(fashion)과 방법(paradigm)을 행위의 모델링과 조합에 이용하는것이 아니다. 본 연구의 궁극적인 목적은 행위가 객체와 분리된 존재로써, 독립적인 관계를 가질수 있다는 사실을 바탕으로[14], 행위를 모델링하기위한 새로운 방법들의 필요성 및 방법을 제시하여 연구자(researcher)및 전문가(experienced practitioner)들에게 새로운 관점(new point of view)을 제공하는 것이다. 따라서 중장기적으로 우리는 행위의 유연한 모델링과 조합을 위해, 행위의 계층성을 지원하기 위해 본 연구에서 제안된 2가지 속성인 자가조합성과 다중상속성의 개념을 기반으로, 이를 지원하는 OOP-독립적인 새로운 언어적 표기법을 개발하고, 상속이나 형질과 같이 행위라는 순차성과 그 관계를, 실세 세상의 행위라는것의 본질(nature)을 다양한 관점에서 파악하여 프로그래밍 레벨에서 더욱 잘 모델링할수 있는 방법에 대한 연구를 진행할 예정이다.

8 결론

본 연구에서는 소프트웨어의 모듈화를 위해 행위의 계층성이라는 개념을 제시하고, 이를 모델링 및 조합을 하기위한 자가조합성과 다중상속성이라는 개념 제안하였다. 이를 프로그래밍에서 지원하기위해 객체지향성을 이용한 자가조합프로그래밍을 제안하였다. 우리는 현대 소프트웨어의 기능을 행위의 집합이라고 정의하고, 사용되는 행위들의 가변성의 증가에 따른 기존 모듈화 기법의 한계를 고차적인 재사용성 지원의 문제라고 규정하였으며, 고차적인 재사용성을 지원하기 위해 상속 및 자가조합이 가능한 행위를 생성하는 자가조합프로그래밍을 제안하고 자가조합형 도메인 분석을 통해 요구사항분석의 일환으로 자가조합적인 설계방법을 제시하였다. 우리는 자가조합프로그래밍의 웹서비스를 대상으로 AOP와 비교하여 평가 및 분석을 실시하여 평가 항목을 통해서 어떤 부분이 기존 방법인 AOP와 비교하여 효율성을 주는지 도출할 수 있었다. 본 연구는 자가조합프로그래밍이라는 실질적인 프로그래밍 방법을 제시함과 동시에, 객체에서 행위를 독립시키고, 행위들에게 자체적인 계층적인 관계를 부여하여 모듈화를 진행함으로써 행위모델링의 중요성을 강조하며, 이를 통한 실질적인 프로그래밍 기법을 제공함과 동시에 연구자와 전문가에게 새로운 관점을 제공해주는 것이 목표이다. 따라서 OOP 독립적인 추가

적인 새로운 표기법 및 행위라는 것이 가지는 본질을 프로그래밍 레벨에서 더욱 잘 표현하는 방법의 중요성을 강조하였다. 우리는 본 연구를 통해 객체지향프로그래밍이 50여 년 전에 실제 사물을 모델링 한 것처럼, 실제 세상의 행위를 정확하게 모델링하고, 쉽게 제어하는 것에 대한 중요성과 가치는 제시하였고, 자가조합프로그래밍이 행위를 모델링 하는 매계체로써 사용될 수 있음을 제시하였다.

참고 문헌

[1] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," tech. rep., Technical Report CMU/SEI-90- TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[2] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[3] M. Shaw, "Modularity for the modern world: summary of invited keynote," in *Proceedings of the 10th International Conference on Aspect-Oriented Software Development, AOSD*, pp. 1–6, 2011.

[4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, *Aspect-oriented programming*, pp. 220–242. Springer Berlin Heidelberg, 1997.

[5] S. Hanenberg and R. Unland, "Concerning aop and inheritance," in *Aspektorientierung-Workshop der GI-Fachgruppe*, vol. 2, pp. 3–4, 2001.

[6] S. Hanenberg and R. Unland, "Using and reusing aspects in aspectj," in *Workshop on Advanced Separation of Concerns, OOPSLA*, 2001.

[7] T. Erl, *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.

[8] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of aspectj," in *European Conference on Object-Oriented Programming*, pp. 327–354, Springer, 2001.

[9] G. Kiczales, "It's not metaprogramming," *Software Development Magazine*, 2004.

[10] O.-J. Dahl, B. Myhrhaug, and K. Nygaard, "Some features of the simula 67 language," in *Proceedings of the Second Conference on Applications of Simulations*, pp. 29–31, Winter Simulation Conference, 1968.

[11] W. Teitelman, "Pilot: a step toward man-computer symbiosis," tech. rep., PhD thesis, September 1966.

[12] E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *ECOOP'93 - Object-Oriented Programming, 7th European Conference*, pp. 406–431, 1993.

[13] "Self-js Webpage." https://github.com/hiun/self-js.

[14] N. Schärli, S. Ducasse, O. Nierstrasz, and A. P. Black, "Traits: Composable units of behaviour," in *European Conference on Object-Oriented Programming*, pp. 248–274, Springer, 2003.

[15] M. D. McIlroy, E. N. Pinson, and B. A. Tague, "Unix time-sharing system: Foreword," *The Bell System Technical Journal*, vol. 57, pp. 1899–1904, July 1978.

[16] "Node.js Webpage." https://nodejs.org.

[17] "jsAspect Webpage." https://git.io/vDmTh.

[18] T. Reenskaug and J. O. Coplien, "The DCI architecture: A new vision of object-oriented programming," *An article starting a new blog:(14pp) http://www. artima. com/articles/dci_vision. html*, 2009.

[19] D. Batory and S. O'malley, "The design and implementation of hierarchical software systems with reusable components," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 1, no. 4, pp. 355–398, 1992.

[20] W. Harrison and H. Ossher, *Subject-oriented programming: a critique of pure objects*, vol. 28. ACM, 1993.

[21] M. Mezini and K. Lieberherr, "Adaptive plug-and-play components for evolutionary software development," in *ACM Sigplan Notices*, vol. 33, pp. 97–116, ACM, 1998.

[22] M. VanHilst and D. Notkin, "Using role components in implement collaboration-based designs," *ACM SIGPLAN Notices*, vol. 31, no. 10, pp. 359–369, 1996.

[23] G. Kiczales, "Once more, from the top," *Software Development Magazine*, 2005.

[24] G. Bracha and W. Cook, "Mixin-based inheritance," *ACM Sigplan Notices*, vol. 25, no. 10, pp. 303–311, 1990.

[25] T. Reenskaug, "The model-view-controller (mvc) its past and present," *University of Oslo Draft*, 2003.

[26] S. Apel and C. Kästner, "An overview of feature-oriented software development.," *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

부록 A AOP 기반 구현체의 소스코드

```
var ReadUser = createAspect(function () {
  logging(data);
  auth(data);
  cacheLookup(data);
  userIdValidation(data);
});

var ReadUserWithoutAuth = createAspect(function
    () {
  logging(data);
  cacheLookup(data);
  userIdValidation(data);
});

var User = {
   getName: applyAspect(ReadUser,
       readUserNameQuery),
   getProfile: applyAspect(ReadUser,
       readUserProfileQuery),
   getPosts: applyAspect(ReadUser,
       readUserPosts),
   getOnline: applyAspect(ReadUserWithoutAuth,
       readUserOnline)
};

ReadPost = createAspect(function () {
  logging(data);
  cacheLookup(data);
  postNumberValidation(data);
  rangeValidation(data);
  ReadRecentsSummaryQuery(data);
});

ReadPostWithoutAuth = createAspect(function ()
    {
  logging(data);
  auth(data);
  cacheLookup(data);
  postNumberValidation(data);
  rangeValidation(data);
  ReadRecentsSummaryQuery(data);
});

var Post = {
  getRecentSummary: applyAspect(ReadPost,
      readPostRecentsSummary)
  getRecentsWithoutImage: applyAspect(ReadPost,
      readPostRecentsWithoutImage)
  getPopularSummary: applyAspect(
      ReadPostWithoutAuth,
      readPostPopularSummary)
  getPopularWithoutImage: applyAspect(
      ReadPostWithoutAuth,
      readPostPopularSummary)
}
```

부록 B AOP Helper 의 소스코드

```
function createAspect (beforeFunc, afterFunc) {
  return new jsAspect.Aspect(new jsAspect.
      Advice.Before(beforeFunc, afterFunc);
}

function applyAspect (aspect, func) {
  //wrapper for applying aspect to function,
      instead of object.
  var obj = {
    method: func
  };
  aspect.applyTo(obj);
  return obj[method];
}
```

부록 C 자가조합프로그래밍 기반 구현체의 소스코드

```
var DBQuery = new Behavior().add(logging);

var DBQueryRead = new DBQuery().add(auth).add(
    cacheLookup);

var DBQueryReadUser = new DBQueryRead().add(
    userIdValidation);

```

```
7  var User = {
8  getName: new DBQueryReadUser().add(
       readUserNameQuery),
9  getProfile: new DBQueryReadUser().add(
       readUserProfileQuery),
10 getPosts: new DBQueryReadUser().add(
       readUserNameQuery),
11 getOnline: new DBQueryReadUser().add(
       readUserOnline).auth.delete()
12 };
13
14 var DBQueryReadPost = new DBQueryRead().add(
       postNumberValidation).add(rangeValidation);
15 var DBQueryReadPostRecents = new
       DBQueryReadPost().add(ReadRecentsQuery);
16 var DBQueryReadPostPopular = new
       DBQueryReadPost().add(ReadPopularQuery);
17
18 var Post = {
19 getRecentSummary: new DBQueryReadPostRecents.
       ReadRecentsQuery.update(
       ReadRecentsSummaryQuery),
20 getRecentsWithoutImage: new
       DBQueryReadPostRecents.ReadRecentsQuery.
       update(ReadRecentsSummaryWithoutImageQuery)
       ,
21 getPopularSummary: new DBQueryReadPostPopular.
       ReadPopularQuery.update(
       ReadPopularSummaryQuery).auth.delete(),
22 getPopularWithoutImage: new
       DBQueryReadPostPopular.ReadPopularQuery.
       update(ReadPopularWithoutImageQuery).auth.
       delete()
23 };
```

Cascade K-Medoids 기반 자바 메소드 식별과 패턴 자동 식별 기법

김태영, 김순태, 이정휴
전북대학교 소프트웨어공학과
rlaxodud1200@jbnu.ac.kr, stkim@jbnu.ac.kr, jhlee25@jbnu.ac.kr
전라북도 전주시 덕진구 백제대로 567

An Automatic Approach to Identifying Java Method Identifier Patterns Using Cascade K-Medoids

Taeyoung Kim, Suntae Kim, Jeong-Hyu Lee
Dept. of Software Engineering, Chonbuk National University
567 Baekje-daero, deokjin-gu, Jeonju-si, Jeollabuk-do 54896, Republic of Korea

요 약

많은 소프트웨어 오픈소스 및 프로젝트에서 메소드는 시스템의 특정 기능을 구현하는 단위이며, 메소드 식별자에 따른 일반적인 구현 패턴이 존재한다. 이러한 구현 패턴에서 벗어난 메소드는, 코드의 독자가 메소드에 기대했던 구현 내용과 코드를 이해하는데, 시간이 걸려 이에 따른 유지보수비가 상승한다. 본 연구에서는 메소드 식별자에 따라 대표되는 여러 패턴을 자동적으로 식별하기 위해 메소드의 식별자와 구현 부 내용을 고려하여 메소드의 특징 벡터를 정의하였으며 기존의 군집화 알고리즘들을 개선하여 Cascade K-medoids를 적용하였다. 실험을 통해 개선된 알고리즘을 군집화에 적용하여 메소드 naming에 따른 메소드들의 대표되는 다양한 패턴을 추출하였으며 메소드 식별자마다 평균적으로 2~4개 정도로 패턴을 식별하였다.

1. 서 론

많은 소프트웨어 오픈 소스 및 프로젝트에서 메소드는 시스템의 특정 기능을 구현하는 단위이며 메소드의 식별자와 구현 부 사이에는 의미적 관계가 존재한다. 또한 식별자에 따라 코드의 내용의 표현이 달라진다. 즉 메소드 식별자에 따른 일반적인 구현 패턴이 존재한다. 이러한 구현 패턴에서 벗어난 메소드는, 코드의 독자가 메소드에 기대했던 구현 내용과 코드를 이해하는데, 시간이 걸려 이에 따른 유지보수비가 상승한다[1]. 따라서 본 연구에서는 자바 메소드 식별자를 이용하여 식별자에 따른 코드의 구현 패턴들을 추출하기 위해 메소드의 식별자와 구현 부 내용을 기반으로 한 특징 벡터를 정의하였다. 또한 K-Medoids을 개선하여 새로운 Cascade K-medoids로 만들고 적용하여 각 메소드의 식별자에 따른 코드의 패턴들을 추출하였다.

2. 자바 메소드 이름에 따른 패턴 자동 식별

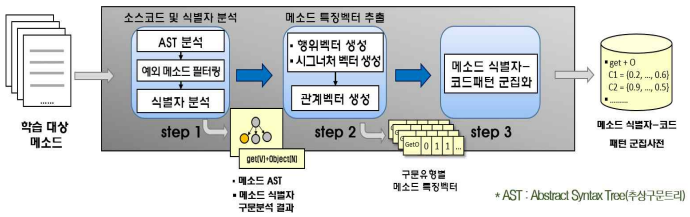


그림 1. 메소드 식별자 코드패턴 군집 과정

2.1 AST를 통한 메소드 식별자 추출

클래스 내에서 해당 메소드를 추출하고 그 메소드 내의 행위와 시그니처를 기반으로 특징 벡터를 구성하기 위해 AST(Abstract Syntax Tree)를 사용하였다. 또한, 메소드 내용에서 특별한 특징을 찾기 어려운 경우 연구결과의 정확도를 현저히 떨어뜨리는 경향이 있다. 따라서 이러한 경우에 해당하는 메소드를 사전 필터링을 통하여 학습 및 실험 대상 메소드에서 제외 시켜 주기 위해 기존에 메소드 식별자 연구를 참조하여 해당 내용들을 제외 하였다[2]. 메소드의 식별자 분석을 위해 'Java Language Specification(JLS)'[3]에서 제안하는 컨벤션 규칙들을 기준으로 하여 메소드 식별자를 구성하는 형태소와 단어의 POS(Part of Speech)를 NLP Parser를 사용하여 도출하였다.

2.2 메소드 특징벡터 추출

일반적으로 메소드는 메소드의 식별자가 포함된 메소드 선언부와, 메소드의 행동과 로직이 구현되는 구현 부로 구성된다. 또한 메소드의 선언부와 메소드의 구현 부 사이에는 서로 연관성이 존재한다. 따라서 우리의 기존 연구[4]에서 제시한 특징 벡터를 확장하여 메소드의 관계와 구성을 기반으로 특징 벡터를 행위벡터, 시그니처 벡터, 관계 벡터로 하여 크게 3가지로 구성하여 정의했다.

2.3 Cascade K-Medoids

데이터를 군집하기 위한 기존의 여러 알고리즘이 존재한다. 그런데 이런 기존의 군집화 기법들은 본 논문에서 제안하는 방법에 적용하는 경우 2가지 문제점이 발생한다.

첫째는 대부분 클러스터 개수 K값을 입력 파라미터로 지정해주어야 한다[5]. 하지만 우리의 경우 K값을 지정해줄 수 없다. 두 번째는 기존의 몇몇 군집화 기법들은 임의의 값을 만들어 중심점을 잡는다. 하지만 패턴확인을 위해서는 실제 값이 필요하다. 따라서 그런 문제를 해결하기 위해 클러스터 내 분산, 클러스터 간 분산을 이용해 best K를 선정해주는 Calinski and Harabasz을 적용하였으며[6] 실제 값들 중 하나를 중심점으로 잡아서 계산을 수행하는 K-medoids 클러스터 기법을 적용하였다[7]. 즉 기존에 검증된 2가지 방법을 착안하여 Cascade K-medoids로 알고리즘을 개선하여 본 연구에 적용하였다. 알고리즘은 다음과 같은 단계로 진행된다.

1. 가능한 K값들에 대해 군집화를 반복하고 각 K에 대한 Calinski-harabasz score를 기록한다.
2. 가장 높은 Calinski-harabasz scroe를 가진 K를 선정한다.
3. 선정된 best k값과 Euclidian distance를 이용해 k-medoid 군집을 실시한다.

3. 실험 및 결과

각 메소드 식별자에 따른 패턴을 확인하기 위해 25개 이상의 Java 기반 오픈소스 프로젝트를 기반으로 사용하였다.

표1은 해당 메소드 식별자에 따른 대표 구현패턴의 개수와 해당 식별자의 패턴을 가진 메소드의 개수를 보여주고 있다. 또한, 메소드 식별자에 따라 평균적으로 2~4 정도의 패턴이 도출 되었음을 알 수 있다. 그리고 다음의 **그림2**는 **표1**의 식별자중 convert 식별자에 해당하는 메소드 식별자의 대표 구현패턴의 유형을 보여주고 있다. 각 패턴은 convert의 식별자 메소드 군집들의 중심점이 되는 대표 패턴들이다. 또한, 각 중심점의 군집 내 분포 개수는 1번 군집은 632개 2번 군집은 387개 3번 군집은 449개로 구성되어 있다.

4. 결론

본 연구에서는 메소드 식별자에 따른 다양한 코드의 대표 구현 패턴을 식별하기 위해 메소드들의 식별자를 분석하고 메소드의 구조를 분석하여 메소드의 특징 벡터를 추출했다. 또한 이 과정에서 분석된 식별자와 특징 벡터를 기반으로 기존의 K-Medoids 알고리즘을 개선한 cascade k-medoids를 적용하여 자동적으로 최적의 군집 개수를 선정하고 그에 따른 실제 값을 확인하여 각 메소드 식별자 마다 평균적으로 2~4 사이의 패턴의 유형을 얻어냈으며 각 패턴에 따른 메소드의 분포 또한 결과로 얻어내었다.

본 논문에서의 메소드 식별자 별 다양한 패턴의 추출과 군집기법을 이용해 다음 연구로는 메소드 식별자에 따른 대표 구현 패턴을 기반으로 부적절한 코드의 형태를 검출할 예정이다.

메소드 식별자	총 메소드 수	구현 패턴	메소드수
get_NP	101,721	**#1**	40,351
		#2	27,591
		#3	24,753
		#4	9,026
set_NP	45,762	**#1**	23,451
		#2	22,311
is_NP	9,076	**#1**	5,263
		#2	3,813
convert	1,468	**#1**	632
		#2	387
		#3	449

표1. 식별자에 따른 구현 패턴과 엔티티의 수

```
1. public Object convert(final Object object) {
       String str = (String ) object;
       return (new Character((str.length() == 0) ? 0 : str.charAt(0)));
   }

2. public Object convert(final Object object) {
       return object;
   }

3. @Override
   public AccessType convert(Object value) {
       if ( value == null ) {
           return null;
       }

       if ( CacheConcurrencyStrategy.class.isInstance( value ) ) {
           return ( (CacheConcurrencyStrategy) value ).toAccessType();
       }

       if ( AccessType.class.isInstance( value ) ) {
           return (AccessType) value;
       }

       return AccessType.fromExternalName( value.toString() );
   }
```

그림2. convert 식별자의 패턴

참고문헌

[1] K. Beck, (2007). Implementation patterns. Pearson Education.

[2] S. Kim., & D. Kim. (2016). Automatic identifier inconsistency detection using code dictionary. Empirical Software Engineering, 21(2), 565-604.

[3] Java Language Specification(JLS), https://docs.oracle.com/javase/specs/, 2015.

[4] I. Lee, S. Kim., S. Park., & Y. Cho. (2016). Attributes for Characterizing Java Methods. In Advanced Multimedia and Ubiquitous Engineering (pp. 185-191). Springer Berlin Heidelberg.

[5] C. M. Bishop. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 12). New York: Springer.

[6] M. Kozak. (2012). "A Dendrite Method for Cluster Analysis" by Caliński and Harabasz: A Classical Work that is Far Too Often Incorrectly Cited. Communications in Statistics-Theory and Methods, 41(12), 2279-2280.

[7] H. Park , C. Jun, A simple and fast algorithm for K-medoids clustering, Expert Systems with Applications, 36, (2) (2009), 3336–3341.

딥러닝을 활용한 주행 패턴 특징 추출 연구

이주영 O1 이홍석 2 정희진 2 장기태 1

1 한국과학기술원 조천식녹색교통대학원

2 한국과학기술정보연구원 슈퍼컴퓨팅본부

Mose585@kaist.ac.kr, hsyi@kisti.re.kr, hjjung1974@gmail.com, kitae.jang@kaist.ac.kr

Deep Learning for Feature Extraction of Driving Pattern Computing

Jooyoung Lee O1, Hongsuk Yi2, Heejin Jung2, Kitae Jang1

1Korea Advanced Institute of Science and Technology,

2Korea Institute of Science and Technology Information

Mose585@kaist.ac.kr, hsyi@kisti.re.kr, hjjung1974@gmail.com, kitae.jang@kaist.ac.kr

요 약

최근 대용량의 데이터 수집이 가능해지고, 딥러닝 기술의 등장으로 인해 교통 안전 향상을 위한 연구 수행이 용이해졌다. 본 연구에서는 운전자 위험도 도출을 위하여 비지도학습 기반 오토인코더 알고리즘을 활용한 라지-스케일 주행 기록 특징 추출 기법을 제시하였다. 분석에 사용된 데이터는 서울, 대구, 부산 지역 택시운전자 39명의 일주일 주행기록이다. 은닉층에 속한 각각의 뉴런들은 특정 패턴의 특징을 추상화하였으며, 오토인코더로 운전 패턴의 특징을 추출하였다.

1. 서 론

빅데이터 기반 딥러닝 기술의 도약적인 발전에 따라, 이를 활용할 수 있는 여러 가지 분야에서 관련 연구가 활발히 수행되고 있다. 교통 분야에서도 지능형 교통 시스템의 구축으로 인하여 다양한 종류의 교통 데이터가 수집되고 있으며, 이러한 흐름에 맞추어 교통류 패턴 예측 및 수요 예측 등 해당 기술을 이용한 연구가 진행되고 있다[1, 2]. 그러나 교통안전과 관련된 연구분야에서는 데이터 확보의 어려움 및 사고의 돌발성·불확실성 등으로 인해 관련 연구개발에 어려움을 겪어 왔다.

교통사고의 원인 3요소는 차량(vehicle), 물리적 및 사회적 환경(physical and social environment), 인적(human) 요소이며, 이중 인적 요소가 90% 이상 작용하고 있다[3], 이에 교통안전 증대를 위해서는 인적 요소에 대한 연구가 필요하다. 특별히 우리나라에서는 비사업용 자동차와 비교 시 교통사고 위험도가 더 높아 고위험군으로 분류되는 사업용자동차 운전자들의 안전관리를 위해 디지털 운행기록계(Digital Tachograph, DTG)의 장착이 의무화 되었다. 실제로 2013년 까지 모든 사업용 차량에 장착되어 매일 대량의 주행 기록이 수집되고 있다[4]. 기존의 연구에서는 제한된 조건에서의 실험 또는 시뮬레이션을 통한 주행 기록 분석을 통해 운전자의 위험 성향을 도출하거나 위험도 등을 도출하였으나, 실제 상황이 반영된 라지-스케일 운행 기록 데이터를 활용한 연구는 부족한 상황이다.

주행 기록을 통한 운전자 성향 분석을 위해서는 주행 기록으로부터 위험운전패턴을 검지하여 점수화가 필요한데, 이를 위해 주행 패턴의 특징 추출이 선행되어야 한다. 시계열 자료의 특징 추출 기법은 기존에 많이 연구되어 있다[5, 6]. 라지-스케일 주행 기록 데이터의 경우 많은 양의 계산이 필요하며 다차원 시계열 자료이기 때문에 패턴 분류의 정확도 및 계산 효율을 높이기 위한 딥러닝 기반의 적합한 특징 추출 방법이 요구된다.

본 연구에서는 라지-스케일 주행 기록 데이터를 활용하여 비지도학습 알고리즘의 하나인 오토인코더 기반의 주행 패턴 특징을 추출하는 기법을 제시하고자 한다.

2. 연구내용 및 방법

2.1 디지털 운행 기록 데이터

디지털 운행 기록 데이터는 초단위로 장착 차량의 운행정보를 전자식 기억장치에 자동적으로 기록하고 있으며, 우리나라에서는 모든 사업용 차량(화물, 버스, 택시)에 의무 장착되어 데이터가 수집되고 있다. 교통안전법에서 명시하고 있는 필수 기록 데이터로는 차량속도, 분당 엔진회전수(RPM), 브레이크 신호, GPS 위치, 가속도 센서 기반 충격 감지, 기기 및 통신 상태의 오류 검출이 명시되어 있으며 제조사에 따라 연비, 주행 도로 정보 등에 대해 추가로 기록이 가능하다. 현재는 운수업체에서 데이터를 개별적으로 관리하며 주기적으로 교통안전공단 서버로 데이터를 업로드하고 있는데, 이동통신망을 활용하여 실시간으로 데이터를 전송하는 시스템도 운영 중이다. 교통안전공단에서는 제출된 사업용 자동차의 운행 기록에 대하여 11개 위험 운전 행동으로 구분하여 유형별 차종, 지역 등으로 구분하고 수치화하여 안전운행 지도 자료로 활용 중이다.

본 연구에서는 라지-스케일 도로 차량운행기록 데이터로 운전자 식별이 가능한 택시회사 4개의 디지털 운행기록계자료를 확보하여 분석에 이용하였다. 전체 운전자 수는 39명이며, 2012년 1월 2일부터 2012년 1월 8일동안 총 7일간의 운행기록계 데이터를 수집하였다. 39명의 택시 운전자들의 주 운전 지역은 국내 대도시 3곳으로, 서울지역 20명, 대구지역 10명, 부산지역 9명이다. 운행기록계로부터 기록되는 데이터는 운행 일자, 전체 운행거리, 주행속도, 가속도, 조향각, 도로명, 차량 및 운전자 정보 등이 기록되며 연구에 이용된 데이터에 대한 개요를 요약하여 아래 표로 정리하였다.

표 1 운행기록계 데이터 개요

분류	개요		
기간	7 일 (2012.01.02. ~ 2012.01.08.)		
데이터 개요	운전자 수	운행 지역	
	39	서울, 대구, 부산	
기록 데이터	분류	단위	비고
	주행 날짜	16 digit	
	주행속도	Km/h	
	가속도	Km/h/sec	
	조향각	Degree	0~359
	제한속도	Km/h	GPS 기반
	브레이크 센서	0,1	Binary
	전체 주행거리	Km/day	
	위치	도로명	
	차량번호	4 digit	

2.2 사전 이벤트 검지

주행 기록계 데이터는 각 차량 별로 하룻동안 주행한 전체 기록이 매초마다 저장되기 때문에 매우 방대한 양을 가지고 있으나, 대부분의 주행 기록은 특이한 사항이 없는 정속주행 데이터가 대부분이다. 안전도 분석을 위해서 필요한 인적 요인이 반영되어 있는 데이터는 운전자가 차량의 주행 속도 및 방향을 조작할 때에 나타난다. 이에 본 연구에서는 운전자가 차량을 급격하게 조작하는 모든 행동을 사전 이벤트로 정의하고, 급격한 조작 행동들을 검출하여 주행 패턴 특징 추출에 이용하였다.

급변점검출은 시계열 자료에서 급격히 변하는 지점을 찾는 통계적인 방법론으로, 현재 데이터마이닝 분야에서 많은 연구가 수행되고 있는데, 이중 정확한 변환점 검출을 목적으로 하는 방법론 중 [7]에서 효과가 입증된 Relative unconstrained least-squares importance fitting(RuLSIF)를 통해 사전 이벤트 검지를 수행하였다. RuLSIF는 운행기록계 자료를 작은 시간창으로 나누어 밀도비 기반의 비유사성을 계산하여 변환점 점수를 계산하였다. 해당 점수가 높은 점들을 변환점으로 검출하는데, 밀도비 기반의 비유사성(D)은 운행기록계 데이터를 따라 시간창을 앞뒤로 모두 옮겨가며 각 점의 차이성을 점수화 하며, 아래 식으로 계산된다.

$$\mathrm{D}(\mathrm{P}||\mathrm{P}') \coloneqq \int p'(Y) f\left(\frac{p(Y)}{p'(Y)}\right) dY$$

여기서 $\mathrm{p}(\mathrm{Y})$ 는 입력데이터의 확률 밀도를 나타내며, 15초의 시간창을 이용하여 변환점 점수가 500점을 초과하는 점들을 사전 이벤트로 검지하여 39명의 택시 운전자 일주일 운행 기록으로부터 총 170,238개의 사전 이벤트를 검지 하였다.

2.3 오토인코더 알고리즘

운전자들의 성향 분석에 이용하기 위해 추출된 사전 이벤트들은 운행 패턴 별로 특징 생성 과정이 필요하다. 일반적으로 시계열 패턴의 특징추출에는 최대값, 최소값, 평균, 분산 등의 통계 값들이 주로 활용되어 왔으나, 본 연구에서는 다차원 시계열 자료에서 더 일반화된 특징을 생성해 낼 수 있는 인공신경망 기반의 오토인코더를 활용하여 추출된 사전 이벤트들의 특징 생성을 수행하였다. 오토인코더는 기본적인 단층 인공신경망 구조에서 입력값과 출력값을 동일하게 하여 학습시킴으로 은닉층의 값이 입력된 시계열 패턴에 대한 특징 값으로 생성되게 하는 비지도학습 방법이다.

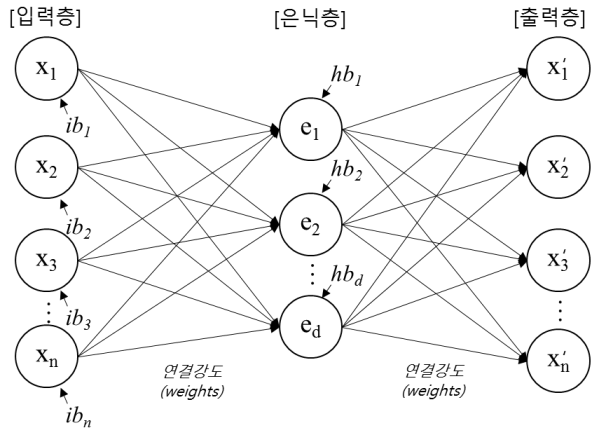


그림 1 비지도학습 오토인코더 신경망

오토인코더를 통해 사전 이벤트의 특징을 추출하는 기법은 위험 운전 패턴 분류, 위험도 산정 등을 위한 깊은 신경망의 초기 층으로, 딥러닝 특징 추출 기법으로 활용할 수 있다. 아래 그림에 기본적인 오토인코더의 구조를 나타내었으며, 일반적인 인공신경망과 같이 각 층의 뉴런 연결 강도(weight)를 stochastic gradient descent 방법을 이용하여 변경하며 학습을 진행한다.

3. 오토인코더 기반 운전 패턴 특징 추출

오토인코더의 입.출력값은 사전 이벤트 도출에 사용된 속도, 가속도, 조향각 각각의 15초간의 센서 값을 벡터 형태로 치환하여 이벤트당 45개의 뉴런을 사용했으며, 특징 생성을 위한 은닉층의 뉴런은 30개로 설정했다. 또한 생성된 특징들에 대해 대표성을 강화하기 위해 sparsity를 반영할 수 있는 추가 학습 변수들을 추가하였는데, L2 weight regulaizer: 0.001, Sparsity regularizer: 4, Sparsity proportion: 0.05로 설정하였다. 이와 같은 학습 변수들을 통해 은닉층의 뉴런 별로 특정 패턴을 대표하는 값을 학습하도록 유도할 수 있다. 오토인코더를 통한 특징 추출을 통해 170,238개의 사전 이벤트로부터 학습된 은닉층과 연결 강도를 구축할 수 있었다. 학습된 은닉층과 특징 추출 결과에 대한 예를 아래 그림과 같이 나타낼 수 있다.

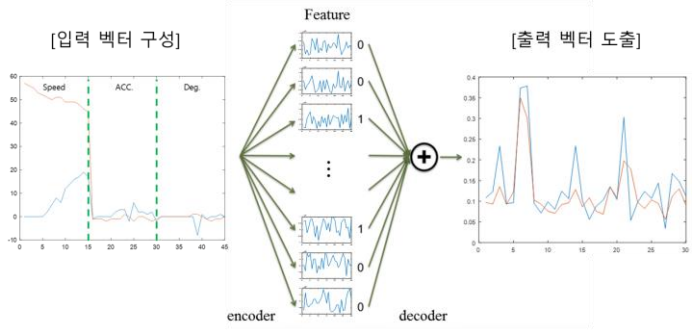


그림 2 운전패턴 특징 추출 예시

은닉층의 특징 값들이 생성되어, 이들간의 연결 강도 조합을 통해 출력 벡터를 생성할 수 있다. 생성된 특징 벡터는 각각 특정한 뉴런에서 값이 큰 경향을 보이는데, 이는 sparsity 변수들에 의한 영향으로, 은닉층에 속한 각각의 뉴런들이 특정 패턴의 특징을 추상화 하였음을 의미한다.

4. 결론 및 향후 연구과제

본 연구에서는 디지털 운행 기록계 데이터로부터 운전자의 성향을 내포하고 있는 운전 패턴의 특징 추출 기법을 제시하였다. 디지털 운행 기록계는 현재 우리나라 모든 사업용 차량에 장착되어 매초 운행 차량의 주행 정보를 기록한다. 수집되는 데이터의 양이 방대하여 현재는 위험운전행동 기준을 정해 일부 자료만을 활용하고 있다. 운전자의 위험도 도출을 위해 운행 기록 데이터로부터 급변점 검지 알고리즘을 적용하여 사전 이벤트를 추출하고, 이들을 효과적으로 분류하거나 계산하기 위한 오토인코더 기반의 특징 추출 기법을 개발하였다.

오토인코더를 통한 주행 패턴 특징 추출은 비지도 학습으로 딥러닝에서 사전 학습의 과정이다. 오토인코더는 기계학습에서 많이 사용되고 있는 주성분석(PCA)와 비슷한 특성을 갖고 있으며, 높은 차원의 데이터를 적당한 저 차원 데이터로 분류가 가능하다. 또한 그림 1에서 보듯이 오토인코더 신경망 은닉층의 활성함수를 시그모이드(Sigmoid)에서 초월함수(tanh) 혹은 ReLU(Retified Linear Unit) 적용으로 딥러닝 신경망의 정확도를 높일 수 있다. 따라서 딥러닝 기반 오토인코더 신경망 시뮬레이션 기법은 디지털 운행기록계 데이터와 같이 많은 특성을 포함한 높은 차원의 비지도학습 데이터를 분류하는데 매우 효율적인 방법이 된다. 특히, 자율주행 자동차를 위한 운전자 위험 행동 분류 및 위험도 산출이 가능할 것으로 기대된다.

본 연구에서는 제한된 데이터를 다루기 위하여 Matlab 기반의 오토인코더 신경망을 이용하였는데, 이를 확장하여 텐서플로우 혹은 Theano 등 딥러닝 공개 어플리케이션 구조로 개발하고 있다. 특히 멀티 GPU 기반의 고성능 컴퓨터를 이용해 라지-스케일 주행 기록 데이터에 대한 병렬성능 확장성 및 GPU 기반의 CUDA 아키텍처에서도 성능최적화 연구를 진행하고 있다.

5. 참고문헌

[1] 이홍석, and 정희진. "슈퍼컴퓨터 기반 교통 혼잡 예측을 위한 딥러닝 성능최적화." *한국정보과학회*

학술발표논문집 (2016): 915-917.
[2] Ma, Xiaolei, et al. "Large-scale transportation network congestion evolution prediction using deep learning theory." *PloS one* 10.3 (2015): e0119044.
[3] Lum, Harry, and Jerry A. Reagan. "Interactive highway safety design model: accident predictive module." *Public Roads* 58.3 (1995).
[4] 박기웅. "TPM 기반 위변조 방지형 디지털 운행기록 장치 설계 및 구현." *한국차세대컴퓨팅학회 논문지* 9.4 (2013): 6-13.
[5] Kuge, Nobuyuki, et al. *A driver behavior recognition method based on a driver model framework*. No. 2000-01-0349. SAE Technical Paper, 2000.
[6] 남광우, 이창우, and 송충원. "3 축 가속도 센서와 방향센서를 이용한 운전 패턴 인식. "*한국컴퓨터정보학회 학술발표논문집* 20.1 (2012): 7-10.
[7] Liu, Song, et al. "Change-point detection in time-series data by relative density-ratio estimation." *Neural Networks* 43 (2013): 72-83.

추상구문트리의 연어 관계 분석을 통한 Java API 패턴 추출 및 추천 시스템 개발

권찬우 ○ 황상원 남영광

연세대학교

arknell@yonsei.ac.kr, arsenal@yonsei.ac.kr, yknam@yonsei.ac.kr

Development of a System for Extracting and Recommending Java API Pattern by Collocation Analysis of AST

Chan-Woo Kwon○ Sang-Won Hwang Young-Kwang Nam

Yonsei University.

요 약

본 연구에서는 추상구문트리의 연어 관계를 이용하여 API 패턴을 추출 및 추천하는 JACE(Java AST Collocation-pattern Extractor) 시스템을 개발하였다. JACE는 추상구문트리를 분석하여 API 호출 노드를 추출하고 노드 간 연어 관계 분석 후 연어 관계 사전을 구축한다. 해당 사전으로 연어 관계 리스트를 형성하여 패턴을 정의하고, 이클립스 플러그인으로 제작된 프로그램을 통해 해당 패턴이 추천된다. 약 800개의 오픈소스 프로젝트 분석 및 약 1천 4백만개의 API 호출 노드를 추출하여 실험하였다.

1. 연구배경

오픈 소스의 보급으로 여러 개발자가 작성한 실무 코드 및 API를 활용하는 빈도가 높아졌다. 이를 사용 하기 위해 API를 제공하는 사이트 및 검색 엔진으로 예제를 검색한다. 그러나 API 제공 사이트가 모든 사용 방법을 제시하고 있지는 않으며, 검색 엔진을 활용한 검색은 많은 시간 비용이 소모된다. 이를 개선하기 위해 본 연구에서는 API 사용 패턴을 추출하고 이 패턴을 예제와 함께 추천하는 시스템을 개발하였다. 본 연구에서 사용한 연어 관계란 두 개 이상의 단어가 결합하여 의미적으로 하나의 단위를 이루는 경우를 말한다[1]. 패턴은 API 클래스의 사용 방법을 의미하며 연어 관계를 이용하여 분석한다. 분석된 패턴은 이클립스에서 사용자에게 추천된다. 본 연구에서는 사용자가 웹 검색 엔진으로 패턴을 검색한 결과와 본 연구의 JACE 시스템의 결과를 비교한다.

2. 시스템 구현

본 연구는 추상구문트리를 이용하여 자바 프로젝트를 분석해 패턴을 추출한다. 그림 1은 시스템의 전체 구조를 나타낸다. 파일 분류기(File Identifier)는 자바 프로젝트에서 java 소스 파일과 자바 아카이브 파일(.jar)을 구분하여 추출하는 모듈이다. Jar 파서는 jar파일에서 클래스 리스트와 메소드 정보를 분석한다. API 추출기는 jar 파일로부터 분석된 클래스 리스트와 메소드 정보 및 java파일의 코드를 이용하여 API 호출 노드를 추출한다. 이 노드는 역할, 라인 넘버, 스코프 정보 등과 함께 데이터베이스에 저장된다.

연어 검색기(Collocation Finder)는 추출된 API 호출 노드에서 연어 관계 노드들을 찾아 연어 관계 사전을 구성한다. 패턴 추출기(Pattern Extractor)는 연어 관계 사전을 기반으로 연어 관계 리스트를 형성하며, 이 리스트는 하나의 패턴으로 정의되어 패턴 저장소에 저장된다.

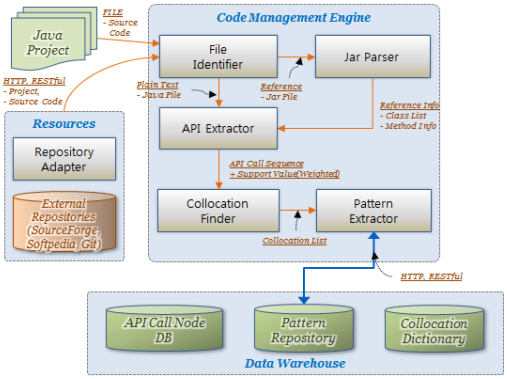


그림 1. 시스템 구조도

본 시스템은 다음과 같은 세 단계로 패턴을 추출한다. 첫째, 자바 프로젝트를 분석하여 API 호출 노드를 추출한다. 둘째, 추출된 API 호출 노드의 연어 관계를 분석하여 연어 관계 사전을 구축한다. 셋째, 구축된 연어 관계 사전으로 패턴을 추출하여 저장소를 구축한다. API 호출 노드는 자바의 추상구문트리를 분석하여 추출하고 이를 이용하여 해당

* 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (No.NRF-2014M3C4A7030505)

코드가 전체 자바 코드에서 담당하고 있는 역할을 파악한다. 역할을 모두 파악한 후, API 호출과 관련된 역할을 하는 코드만을 이용하여 API 호출 노드를 구성한다. 자바 추상구문트리로부터 추가로 얻을 수 있는 정보는 코드 라인 번호, 패키지 정보 등 다양하다. 표 1은 분석된 API 호출 노드의 역할과 종류이다.

표 1. API 호출 노드의 역할과 종류

태그 이름	역할	종류
PD (Package Declaration)	패키지 정의	비분석 노드
ID (Import Declaration)	임포트 정의	비분석 노드
FD (field Declaration)	필드 정의	비분석 노드
RS (Result Statement)	메소드 반환	비분석 노드
VDE (Variable Declaration Expression)	로컬 변수 선언	분석 노드
MCE (Method Call Expression)	메소드 호출	분석 노드
OCE (Object Creative Expression)	객체 생성	분석 노드
LE (Lambda Expression)	람다식	분석 노드

저장된 API 호출 노드는 단어로 취급하여 연어 관계를 분석한다[2]. 이후 단어들의 연어 관계를 연계하여 리스트를 형성하고 확장 분석하였다. 우도비 λ는 연어 관계 분석에 사용되는 두 가지 우도비 가설을 활용하여 계산하였고, −2logλ를 취해 카이 제곱 분포와 유사하도록 변형하였다. 그 후, 자유도 1, a=0.005의 임계점 7.88 보다 큰 결과의 두 단어를 연어 관계로 정의하였다. 표 2는 분석된 연어 관계 중 −2logλ 값이 큰 관계를 보여준다.

표 2. 우도비를 이용한 연어 관계 분석

코드 1	코드 2	−2logλ
`Class.forName(className);`	`DriverManager.getConnection (conInfo, ID, PWD);`	30.89
`Class.forName(className);`	`e.printStackTrace();`	30.89
`Connection instanceCon = null;`	`DBData data = new DBData();`	26.39

추출된 연어는 1 대 1의 관계를 가지고 있지만 패턴은 일반적으로2개 이상의 코드로 구성되기 때문에 연어 관계 리스트로 확장이 필요하다. 연어 관계 리스트는 다음과 같이 추출할 수 있다.

$$\{\alpha \mid \alpha \in A, A \text{ is collocations of API Call Node } \omega_1\}$$
$$\{\beta \mid \beta \in B, B \text{ is collocations of API Call Node } \omega_2\}$$

위와 같은 두 집합을 정의하였을 때, $\omega_2 \in A$ 를 만족하고 A, B가 한 스코프에서 사용된 API 호출 노드라면, ω_1, ω_2, B 는 연어 관계 트리를 형성한다. 이로부터 $\{\omega_1, \omega_2, \beta_n\}$의 연어 관계 리스트를 추출할 수 있다.

추출된 패턴은 이클립스 플러그인으로 개발된 시스템에서 추천 받을 수 있다. 사용자가 코드를 입력하고 코드의 일부분을 선택하여 마우스 우측 버튼을 클릭한 후 find pattern 메뉴를 선택하면 추천 예제 결과가 나타난다

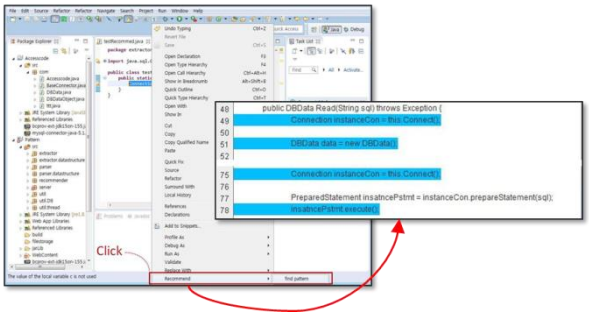


그림 2. JACE를 이용한 패턴 추천 예제

3. 실험 및 결과

실험에서는 794개의 자바 프로젝트 중 컴파일이 가능한 271,664개의 파일을 분석하였다. 추출된 API 호출 노드는 20,144,830개, 분석 노드는 14,947,247개, 스코프는 1,873,032개이다. 추출된 패턴의 정확도 검증을 위해 API 호출 노드에서 출현 빈도가 높은 java.util.List, java.lang.StringBuilder 등의 클래스와 웹 검색 엔진의 결과와 비교하였다. 본 시스템은 실험 대상 API 클래스에 대해 모두 예제 코드를 제공하였으나 참조사이트에서는 대부분 예시 자료를 제공하지 않았으며, 몇몇 클래스에 대해서 본 시스템이 더 다양한 패턴 및 예시를 추천하였다.

4. 결론

본 논문에서는 자바 추상구문트리를 이용하여 API 호출 노드 추출 및 노드 간 연어 관계 기반 패턴 분석 시스템 JACE를 개발, 제안하였다. API 호출 노드는 분석/비분석 노드로 분류되며 우도비 검정을 통해 추출된 노드의 연어 관계를 분석하였다. 분석된 연어 관계는 하나의 패턴을 구성하나, 연어 관계는 1대 1 관계이기 때문에 하나의 패턴으로 정의하기 어렵기 때문에 연어 관계 리스트로 관계를 확장하였다. 정확도 검증을 위해 실제 API가 사용된 예제 코드와 코드 검색 엔진을 비교하였고, 본 시스템이 더 많은 예제를 제공하였다. 본 시스템은 개발자로 하여금 적절한 예제코드를 활용하여 신뢰도 있는 코드를 작성할 수 있도록 도와주며 코드 재사용 도구 제작에도 기여할 수 있다. 향후 더 많은 프로젝트를 수집하여 패턴의 정확도를 높인다면 더욱 활용도가 높은 시스템이 될 것이라 판단된다.

참고문헌

[1] 고려대학교민족문화연구원. “고려대 한국어 대사전” (1964)

[2] C.Manning, Hinrich Schütze. “Foundations of statistical natural language processing”(1998)

경북대학교 SW중심대학사업단의 SW가치확산 활동 사례 연구

정원일 O

경북대학교 SW 교육센터

wonil@knu.ac.kr

영문제목

A Case Study of the Diffusion of SW Value in the SW Education Center of Kyungpook National University.

Chung, Wonil O

The SW Education Center of Kyungpook National University

요 약

본 연구는 2016년도에 경북대학교 SW교육센터에서 SW가치확산을 실시한 방법과 과정을 소개하고자 한다. 먼저 2명의 인력이 투입되어 6천 명 이상의 유아,초,중,고등, 대학, 학부모, 및 일반을 대상으로 실시한 교육의 기획, 실시, 결과에 대해서 관찰한 다음, SW가치 확산이 사회에 미치는 긍정적인 결과들을 공유해서 가치확산사업을 실시하는 타 기관에게 best practice를 제공하자 한다. 본 사례 연구는 지역의 협력기관과 체계적이고 지속적인 네트워킹이 보다 효과적인 가치확산 활동을 전개할 수 있음을 보여준다.

소프트웨어중심대학사업 수행사례 - 성균관대학교 사례

이은석
성균관대학교 소프트웨어대학
leees@skku.edu

A Case Study of the Program of Software Centered University -Sungkyunkwan University Case

Lee Eunseok
College of Software, Sungkyunkwan University

요 약

본 발표는 지난 2년간 성균관대학교에서 소프트웨어 중심대학사업을 수행한 경험을 소개하고 진행방식이나 수행시 겪게 된 여러가지 개별적 문제점들에 대해 정리하고 공유하는 것을 목적으로 한다. 이를 통해 동일 사업을 수행하고 있는 타 대학과 방법론이나 보완사항의 공유는 물론 본 사업의 성공적인 수행을 위한 개선책을 더불어 모색해 나가는 것을 목표로 한다. 또한 본 사업에 참여하고자 하는 대학들에 대해서 필요한 정보를 제공하는 것을 또 다른 목적으로 한다.

비전공자를 위한 컴퓨팅사고력 교육과정 -고려대학교 운영사례[1]-

김현철 ○
고려대학교 정보대학 컴퓨터학과
harrykim@korea.ac.kr

Computational Thinking Curriculum for Non-major students -Cases at Korea University-

Hyeoncheol Kim ○
Korea University, Computer Science and Eng.

요 약

본 발표에서는 대학 교양교육에서의 중요한 요소로 최근 이야기 되고 있는 Computational Thinking, 즉 컴퓨팅사고력 교육을 위한 교육과정 설계와 운영 이슈들을 고려대학교에서의 사례로 제시하고자 한다.

먼저, 교육과정설계는 컴퓨팅사고력의 정의와 배경, 목표에 바탕을 두고 목표 역량의 공통성과 다양성을 균형 있게 제시하는 것에 초점을 두었다. 따라서 모든 3,800여명의 입학생이 반드시 들어야 하는 공통교양 '정보적사고' 수업을 개발하여 기본적인 탐색을 할 수 있는 기회를 갖도록 하며, 대신 1학점 Pass/Fail, 그리고 동영상 위주의 수업으로 진행하여 경쟁과 성적에 대한 부담을 감소시키고자 하였다. 그 다음 단계로는, 핵심교양 영역에 10여가지의 다양한 과목을 개발하여 제공함으로써 학생들 스스로의 관심사와 전공에 따라서 선택하여 들을 수 있도록 하였다. 컴퓨팅사고력을 바탕으로 한 다양한 주제와 관점을 가진 이들 13과목에 대하여 2016년 기준으로는 1200여명의 비전공자들이 선택수강을 하였다. 또한 이러한 과목을 수강한 후에 좀 더 심화된 컴퓨터과학 수업을 듣고자 하는 비전공 학생들을 위하여 컴퓨터학과에서 개설되는 기본 전공과목은 비전공자용 수업을 별도로 개설하여 제공하고 있다. 또한 정규 수업 외에도 방학 중 별도의 특강을 제공하여, 정규 수업으로 제공하기에는 적절하지 않지만 많은 관심을 가지는 주제(iOS 와 안드로이드 프로그래밍)에 대하여 강의를 제공하고 있다. 이러한 교육과정을 통하여, 전공에 관계없이 모든 학생들이 컴퓨팅사고력 주제에 반드시 하지만 쉽게 접근할 수 있도록 다양한 접근 경로를 제공해주고, 그 다음에는 각자의 관심과 전공에 따라 심화된 과정을 통하여 자신의 분야에서 그러한 역량을 습득하고 발휘될 수 있도록 하고 있다.

위의 교육과정을 실제로 적용하고 운영하는 과정에 발생하는 대부분의 문제는 대형강의 상황에서 발생한다. 수업을 수강하는 학생들의 수에 비하여 교수나 조교의 수가 현저히 부족한 상황에서는 대규모 강의는 자연스럽게 등장을 하게 된다. 그리고 프로그래밍 관련 수업을 대규모로 해본 경험이 부족한 상황에서 기존과 같은 방식의 수업을 대규모로 할 경우에는 학생들의 수업 만족도와 성취도가 눈에 띄게 떨어지게 될 가능성이 매우 높다. 따라서, 강의 방식, 실습 방식, 실습 환경, 숙제 및 시험에 대한 피드백, 질의 응답 등에 기존과는 다른 새로운 교수학습방법이 연구되고 적용될 필요가 있다. 또한 이러한 종류의 수업에 대한 학생들의 거부감을 없애고 동기부여를 줄 수 있는 다양한 방안도 필요하다. 본 발표에서는 이와 관련되어 지난 2년간 운영했던 경험을 사례로 논의하고자 한다.

[1] 이 연구는 2015년 대한민국 교육부와 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2015S1A5A2A01011911)

QoS 요구사항 만족을 위한 온톨로지 기반 태스크 매치메이킹 기술

김 민 협　　고 인 영

한국과학기술원 전산학부

{minhyeop.kim, iko}@kaist.ac.kr

Ontology Based Task Matchmaking for Satisfying QoS Requirement

MinHyeop Kim　　In-Young Ko

KAIST, School of Computing

요　약

IoT 환경에서는 다양한 종류의 IoT 기기들이 공존한다. 이러한 IoT 기기는 센서와 액츄에이터로 이루어져 있으며, 각 기기의 기능을 서비스로 제공함으로써 사용자의 요구에 맞게 조합된 서비스를 제공할 수 있다. 특히 많은 수의 센서로부터 수집된 센서 데이터에 기반하여 해당 공간에서 일어나는 비상 상황을 인지하고, 해당 비상 상황에 알맞은 서비스 조합을 통해 태스크를 수행할 수 있다. 그러나 IoT 기기의 수가 많아짐에 따라, 해당 상황에서 사용할 수 있는 서비스의 수가 늘어나고 각 기기들의 사용 여부에 따라 수행할 수 있는 태스크가 동적으로 변화하기 때문에 알맞은 태스크를 찾아 제공하는 태스크 매치메이킹이 어려워진다. 본 연구에서는 이러한 문제를 해결하기 위해서 비상 상황 및 태스크에 대한 정보를 저장하기 위한 온톨로지를 구성하고, 태스크에 필요한 IoT 기기의 사용 가능 여부를 동적으로 확인하여 태스크 매치메이킹에 필요한 탐색 공간을 줄이는 태스크 매치메이킹 방법을 제안하였다. 매치메이킹 환경 및 이벤트, 태스크의 특성에 따라 태스크 필터링 순서를 조절하고 이에 따른 성능 변화를 측정하여 분석하였다.

1. 서　론

IoT 환경에서는 다양한 종류의 IoT 기기들이 공존하며, 이러한 기기들에 대하여 투명하고 seamless한 접근을 허용한다 [1]. 이러한 IoT 기기는 센서(sensor)와 액츄에이터(actuator)로 이루어져 있는데, 각 IoT 기기의 기능을 서비스로 제공함으로써 사용자의 요구에 맞게 조합된 서비스를 제공할 수 있다. 미래의 IoT 환경에서는 각 공간에 배치된 IoT 기기의 수가 늘어나면서 더욱 복잡한 서비스를 제공할 수 있다. 특히 많은 수의 센서로부터 수집된 센서 데이터에 기반하여 해당 공간에서 일어나는 상황을 인지하고, 해당 상황에 알맞은 서비스 조합을 통해 태스크(task)를 수행할 수 있다 [2]. 태스크는 사용자의 요구를 충족시키기 위하여 다수의 서비스로 구성된 서비스의 조합으로, 각 상황에 따라 다른 태스크가 존재할 수 있다.

이러한 상황 인지 기술이 발전하면 IoT 공간에서 일어나는 비상 상황 또한 인지할 수 있다. 비상 상황은 공간에 설치된 센서로부터 평소와는 다른 이상 센서 데이터가 수집되거나, 비상 상황에 정의된 특정 센서 값이 수집될 경우에 인지될 수 있다. 비상 상황이 인지되었을 경우, 해당 비상 상황에 알맞은 태스크를 선택하여 수행하는 것은 비상 상황에서도 사용자의 요구를 만족시킨다는 점에 있어서 매우 중요하다. 이러한 비상 상황 해결을 위한 태스크를 이벤트 핸들링 태스크(Event Handling Task)라 정의하고, 비상 상황 인지시 알맞은 이벤트 핸들링 태스크를 매치메이킹(matchmaking) 하는 것을 목적으로 한다. 이 때, 비상 상황에 알맞은 태스크는 비상 상황을 해결하는 데에 도움이 되는 태스크들 중 가장 효과적이고, 해당 시점에 수행 가능한 태스크를 말한다.

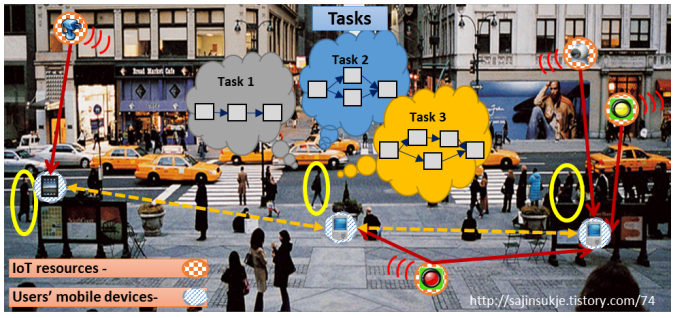


그림 1. 다양한 서비스가 존재하는 IoT 환경

그러나 해당 비상 상황을 해결하기 위해 이벤트

핸들링 태스크를 매치메이킹 하는 것은 해당 공간의 IoT 기기들의 상태를 파악하여 어떤 태스크들이 수행 가능한지를 알아낼 필요가 있다. 또한 해당 비상 상황의 Quality of Service(QoS) 요구사항과 각 태스크들의 QoS를 매치하여 가장 알맞은 태스크를 찾아내야 한다.

그러나 IoT 기기의 수가 많아지면서, 해당 공간에서 감지해낼 수 있는 비상 상황의 수가 많아질 뿐만 아니라, IoT 기기가 제공하는 서비스의 수가 많아짐에 따라 해당 공간에서 수행할 수 있는 태스크의 수가 많아진다. 이 때문에 특정 비상 상황에 가장 알맞은 태스크를 찾는 태스크 매치메이킹의 복잡도는 점점 증가한다. 그러므로 해당 상황이 벌어지는 환경의 특성과 해당 환경에서 제공 가능한 태스크의 특성, 그리고 해당 상황에 특성에 따라 태스크 매치메이킹 방식을 다르게 적용하여 탐색 공간을 효과적으로 줄여 나가는 것이 중요하다.

이를 위해서 본 연구에서는 온톨로지를 정의하여 태스크 매치메이킹에 필요한 각 비상 상황의 정보와 해상 비상 상황을 해결할 수 있는 이벤트 핸들링 태스크의 정보를 저장하였다. 또한 태스크 매치메이킹 과정을 태스크의 타입에 따른 필터링, 이벤트의 QoS 요구사항에 따른 필터링, IoT 기기 사용 가능 여부에 따른 수행 가능한 태스크 필터링 등, 총 세 가지의 태스크 필터링 과정으로 나누어 수행하였다. 또한 이는 상황에 따라 탐색 공간을 효과적으로 줄이기 위하여 이 세 가지의 태스크 필터링 과정의 순서를 이벤트, 이벤트 핸들링 태스크, IoT 기기 상태의 특성에 따라서 바꾸면서 상황에 맞춘 효율적인 필터링 순서를 찾아내었다.

본 논문의 구성은 다음과 같다. 2장에서 태스크 매치메이킹 기법에 관한 관련 연구를 소개하며, 3장에서는 이 논문에서 대상으로 하는 문제 정의, 4장에서 문제 해결을 위한 접근 방법을 상세히 설명한다. 그리고 5장에서는 해당 접근 방법의 실험 결과에 대해서 설명하며 6장에서는 결론 및 향후 연구에 대해서 설명한다.

2. 관련 연구

Paolucci [3]는 웹 서비스의 capability를 매치메이킹 하기 위하여 DAML-S(DARPA agent markup language for services)[4]를 이용하여 서비스 프로파일(service profile)을 기술하고, 서비스 광고(service advertisement)와 서비스 요청(service request)을 매칭하기 위한 매칭 엔진을 제안하였다. 이를 위해서 서비스의 capability에 대한 의미적인 정보와 서비스 광고와 서비스 요청을 매칭하기 위한 서비스 매칭 알고리즘의 명세를 표현할 수 있는 DAML-S를 사용하였다. 이 연구의 매칭 엔진은 서비스 광고와 서비스 요청의 입력값과 출력값의 키워드 매칭을 사용하여 각 입출력간의 포함관계를 분석하는 방식으로 서비스 요청에서 요구되는 서비스의 캐퍼빌리티(capability)를 매칭하였다. 그러나 이 연구에서는 서비스의 현재 사용 가능 여부를 기본 가정으로 삼고, 사용 가능 여부를 확인하지 않는다는 한계가 있다. 이러한 한계는 서비스의 사용 가능 여부가 실시간으로 바뀌는 IoT 환경에는 적용하기 어렵다.

Cassar [5]는 Latent Semantic Analysis(LSA)를 이용한 통계적 서비스 매치메이킹 알고리즘과 서비스의 IO 파라미터간 링크 가중치에 기반한 로지컬 시그니처 매치메이킹(logical signature matchmaking)을 조합한 하이브리드 시맨틱 매치메이킹 방법을 제안하였다. 이러한 하이브리드 방법은 논리 기반 매치메이킹(logic based matchmaking)의 strict matching보다는 유연한 매칭을 제공하고, 비 논리 기반 매치메이킹(non-logic based matchmaking)에 비해서는 보다 세분화된 매칭을 할 수 있다는 장점이 있다. 또한 시맨틱 매치메이킹에서의 의미적 동의성 문제를 해결할 수 있다. 그러나 이 방법 역시 서비스의 실시간 사용 여부를 확인하지 않으며, 다수의 서비스로 이루어진 태스크 매치메이킹에 사용하기 위해서는 태스크의 description이 존재해야만 LSA를 이용한 매칭이 가능하다는 한계가 있다.

Fenza [6]는 에이전트 기반의 패러다임과 퍼지 모델링(fuzzy modelling)을 결합한 하이브리드 시맨틱 매치메이킹 방식을 제안하였다. 이는 에이전트 기반의 패러다임을 이용하여 서비스 제공자간 상호운용성을 높여 서비스 제공자간 중개 및 매치메이킹을 제공하는 방식이다. 이 연구에서는 서비스 캐퍼빌리티에 대한 기술이 가능한 OWL-S[7]를 사용하여 웹 서비스를 정의하고, OWL-S에 기술된 다양한 레벨의 정보를 유연하게 나타낼 수 있는 퍼지 멀티셋을 이용하여 서비스 매치메이킹을 한다. 이들의 연구는 OWL-S에 저장된 서비스의 description 정보를 이용하기 때문에, 다수의 서비스로 이루어진 태스크에 적용하기에는 무리가 있다.

Kritikos [8]는 서비스의 수가 증가함에 따라 같이 증가할 수 밖에 없는 매치메이킹 시간을 최적화를 통해서 줄이기 위한 방법을 제시하였고, 그를 위해 기능상으로는 같은 여러 개의 서비스를 선택하는데 있어서 비기능적(non-functional)인 요소를 고려하였다. 이 연구에서는 서비스간 “subsumes” 및 “subsumedBy”의 관계를 이용하여 탐색해야 하는 request-to-advertisement 쌍의 수를 줄임으로써 서비스 요청의 매치메이킹을 하는 데 걸리는 시간을 줄이는 방법을 제시하였다. 또한 서비스의 비기능적 요소를 고려함으로서 탐색해야 하는 서비스 제안의 공간을 줄여, 걸리는 시간을 줄이는 방법을 제안하였다. 이러한 방법들은 태스크 매치메이킹 하는 데에 부수적으로 사용될 수 있을 것으로 보인다.

Alnahdi [9]는 QoS 온톨로지를 이용하여 웹 서비스

명세에 QoS attribute를 추가하고, 서비스의 QoS 명세를 이용해 서비스를 필터링 하는 매치메이킹 알고리즘을 제안하였다. 이 연구의 매치메이킹 알고리즘은 서비스 요구에서의 QoS 요구값과 매치메이킹 대상 서비스 중 요구된 서비스와 기능적으로 같은 서비스의 QoS 값의 유사도를 비교하여 유사도가 높은 서비스를 매치메이킹 한다. 만일 유사도가 낮은 경우에는 파라미터 조정 알고리즘을 사용해서 서비스 요청자로 하여금 요청하는 QoS 값을 조정하게 한 후, 매치메이킹 알고리즘을 다시 사용하도록 하였다. 그러나 이 연구에서는 서비스 수행시 서비스의 사용 가능 여부를 확인하지 않고, 다수의 서비스로 이루어진 태스크에는 적용하기 어렵다.

3. 문제 정의

IoT 기기의 수가 늘어남에 따라 각 IoT 기기가 제공하는 서비스의 수가 늘어나는 상황에서는 다음과 같은 문제점을 고려해야 한다.

첫째, 인지 가능한 비상 상황 및 이벤트 핸들링 태스크 수의 증가에 의한 문제가 있다. IoT 환경에 존재하는 IoT 기기의 수가 늘어나면서 각 공간에 설치된 센서의 수도 늘어나게 되는데, 이를 통해서 인지 가능한 비상 상황의 수가 늘어날 수 있다. 또한 새로운 비상 상황들은 기존의 비상 상황의 QoS 요구사항과는 다른 QoS 요구사항이 있을 수 있다. 새로 늘어난 QoS 요구사항 타입의 수는 태스크 매치메이킹의 복잡도를 증가시키며, 태스크 매치메이킹의 속도에 영향을 끼친다.

또한 인지 가능한 비상 상황이 늘어나면서 이에 대처하기 위하여 환경에 액츄에이터를 추가하게 된다. 이 때 추가된 액츄에이터가 제공할 수 있는 서비스들이 새로 생기게 되면서 이벤트 핸들링 태스크의 수가 증가할 수 있으며, 이미 존재하던 이벤트 핸들링 태스크 역시 구성하는 서비스가 추가되거나, 서비스에 참여하는 IoT 기기의 수가 늘어나 더욱 복잡해질 수 있다. 이렇듯 늘어나는 IoT 기기의 수는 문제의 복잡도를 급격하게 증가시킨다.

둘째, IoT 기기의 사용 가능 여부(availability)에 따른 태스크 수행 가능 여부를 파악하는 것이 어렵다. 이벤트 핸들링 태스크를 수행하기 위해서는 해당 태스크를 구성하는 서비스를 제공하는 IoT 기기가 사용 가능해야만 한다. 이러한 IoT 기기의 availability 정보는 기본적으로 태스크 매치메이킹 알고리즘에 제공되어야 한다. 태스크 수행에 필요한 IoT 기기가 사용가능하지 않을 경우 해당 태스크는 성공적으로 수행될 수 없으며, 비상 상황 해결을 위해선 다른 태스크를 선택해야 한다. 그러나 태스크 수행에 참여하는 IoT 기기의 수가 증가하면서, 각 태스크의 수행 가능 상태 파악의 복잡도가 상승한다. 이는 이벤트 핸들링 태스크 수의 증가와 함께 태스크 매치메이킹의 소모 시간을 증가시키며 비상 상황 해결에 어려움으로 작용한다.

3.1. 문제 요구 사항

비상 상황 해결을 위한 이벤트 핸들링 태스크 매치메이킹에는 기본적으로 다음과 같은 가정이 필요하다. 첫째, 비상 상황 인지가 제공되어야 한다. 이는 비상 상황 인지 기능이 어떠한 방식으로든 행해져서 현재의 비상 상황을 파악해야 한다는 것이다. 이렇게 파악된 비상 상황은 매치메이킹 알고리즘의 입력으로 주어져 태스크 매치메이킹의 목표로 설정되며, 매치메이킹 알고리즘은 비상 상황 인지의 입증을 하지 않는다. 둘째, 현재 목표 공간 내에 위치한 IoT 기기의 상태를 파악해야 한다. 태스크를 구성하는 서비스의 실행 가능 여부 파악을 위해서는 해당 서비스 실행에 필요한 IoT 기기들의 상태 파악이 필수적이다. 이는 비상 상황 해결을 위한 태스크 선택에 매우 중요한 요인이 된다. 그러므로 태스크 매치메이킹을 하는 시점에 태스크에 필요한 모든 IoT 기기의 사용 가능 여부를 파악하여 태스크 매치메이킹 알고리즘에 입력으로 제공해야 한다.

위와 같은 기본 가정에 기반하였을 때, 비상 상황을 해결하기 위한 이벤트 핸들링 태스크 매치메이킹은 다음과 같은 요구사항이 있다. 첫째, 비상 상황이 요구하는 QoS 요구사항을 모두 만족시키는 태스크가 존재할 때, 이러한 태스크를 찾아내 제공하여야 한다. 이는 비상 상황 해결을 위해 특정 QoS가 필수적인 경우가 있으며, 비상 상황의 QoS 요구사항에 맞는 QoS를 갖춘 태스크가 있을 경우 해당 태스크를 제공해야 한다는 말이다. QoS 요구사항을 만족시키는 태스크가 존재함에도 불구하고 태스크 매치메이킹 과정에서 이러한 태스크를 찾지 못한다면 태스크 매치메이킹의 기본 목적에 부합할 수 없다.

둘째, 태스크 제공시 태스크의 수행 가능 여부는 사용자의 요구를 만족시키는 데 가장 중요하기 때문에 태스크에 필요한 IoT 기기의 수행 상태를 파악하여 태스크가 제공 가능한지 확인하여야 한다. QoS 요구사항을 만족시키는 태스크가 존재하더라도 태스크 수행 시점에 IoT 기기의 상태에 따라 수행 가능하지 못할 수 있기 때문에, 이러한 확인은 필수적이다.

셋째, 요구사항을 모두 만족시키는 태스크가 여러 개 있을 경우, 이러한 후보 태스크 중에서 QoS가 높아 사용자를 만족시킬 수 있는 태스크를 선택하여야 한다. 그러나 본 연구에서는 각 이벤트의 QoS 요구사항이 충분히 세부적으로 정의되어 있다고 가정하고, 이에 따라 매치메이킹 과정에서 가장 높은 QoS를 갖춘 태스크를 찾는 것이 아닌 QoS 요구사항을 모두 만족시키는 태스크들을 찾아내는 것을 목적으로 한다.

3.2. 요구사항 분석

3.2.1. 태스크 매치메이킹을 위한 온톨로지 설계

태스크 매치메이킹을 위한 온톨로지에는 태스크

매치메이킹을 위해 필요한 정보로 비상 상황과 각 비상상황을 해결하기 위한 이벤트 핸들링 태스크를 정의해야 한다. 비상 상황에 대한 정보로는 해당 비상 상황의 타입(Type)과, 비상 상황의 심각도, 비상 상황 해결에 필요한 QoS 요구사항에 대한 정보가 있다. 이러한 정보는 태스크 매치메이킹 과정에서 이벤트 핸들링 태스크의 정보와 비교하여 가장 알맞은 태스크를 매칭하는데 필수적이다.

이벤트 핸들링 태스크의 정보로는 태스크의 이름과 태스크의 타입이 기본적으로 필요하다. 태스크의 타입은 특정 비상 상황에 필요한 태스크들의 타입을 매칭하기 위해서 필요하다. 또한 해당 태스크가 제공 가능한 QoS의 타입과, 각 QoS의 레벨을 저장해야 한다. 이벤트 핸들링 태스크의 QoS 타입은 각 비상 상황에 필요한 QoS 요구사항과 매칭하여 해당 요구사항을 만족시키는 태스크들을 걸러내는데 사용한다. 또한 이러한 QoS 타입마다 QoS 레벨을 제공함으로써 비상 상황의 심각도에 따라 QoS 레벨을 맞추고, 높은 QoS를 가진 태스크를 제공할 수 있게 한다. 또 각 이벤트 핸들링 태스크를 구성하는 서비스 조합에 관한 정보를 저장하여야 한다. 서비스 조합에 관한 정보는 해당 서비스 조합을 구성하는 서비스들의 정보와 태스크를 수행하는데 필요한 IoT 기기의 타입 정보로 구성되어 있으며, 이러한 정보는 태스크 매치메이킹 과정에서 각 태스크가 수행 가능한지를 판단하기 위해 사용된다.

3.2.2. 이벤트 핸들링 태스크 매치메이킹 알고리즘 설계

이벤트 핸들링 태스크 매치메이킹 알고리즘은 다음과 같은 요구사항이 필요하다. 비상 상황이 요구하는 QoS 요구사항을 모두 만족시키는 태스크가 존재할 때, 이러한 태스크를 찾아내 제공하여야 한다. 이는 비상 상황 해결을 위해 특정 QoS가 필수적인 경우가 있으며, 비상 상황의 QoS 요구사항에 맞는 QoS를 갖춘 태스크가 있을 경우 해당 태스크를 제공해야 한다는 말이다. 둘째, 태스크 제공시 태스크의 수행 가능 여부는 사용자의 요구를 만족시키는 데 가장 중요하기 때문에 태스크에 필요한 IoT 기기의 수행 상태를 파악하여 태스크가 제공 가능한지 확인하여야 한다. QoS 요구사항을 만족시키는 태스크가 존재하더라도 태스크 수행 시점에 IoT 기기의 상태에 따라 수행 가능하지 못할 수 있기 때문에, 이러한 확인은 필수적이다.

4. 접근 방법

4.1. 태스크 매치메이킹을 위한 온톨로지 설계

비상 상황 해결을 위한 태스크 매치메이킹을 구현하기 위해 다음과 같은 온톨로지를 설계하였다. 이 온톨로지는 태스크 매치메이킹에 필요한 각 비상 상황의 정보와 해당 비상 상황을 해결할 수 있는 이벤트 핸들링 태스크의 정보의 관계를 정의하고 이러한 정보를 저장하는 데에 쓰인다. 이 온톨로지로 정의하는 각 개념은 그림 2에 표현되어 있으며 자세한 정의는 다음과 같다.

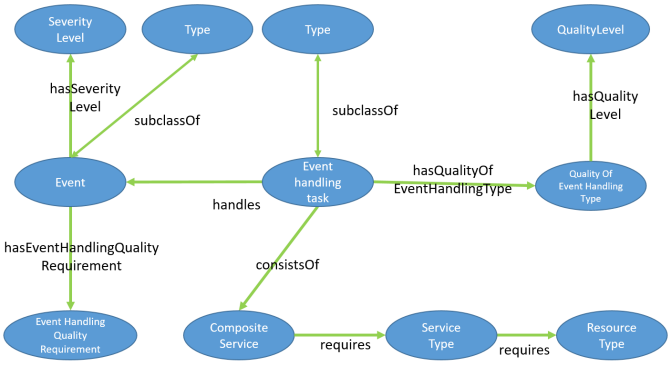


그림 3. 태스크 매치메이킹을 위한 온톨로지

비상 상황의 정보는 각 비상 상황의 이름과 해당 비상 상황의 타입(Type), 심각도(Severity level), 이벤트 핸들링 태스크의 QoS 요구사항(Event handling quality requirement)으로 이루어져 있다. 이러한 정보는 태스크 매치메이킹 과정에서 이벤트 핸들링 태스크의 정보와 비교하여 가장 알맞은 태스크를 매칭하는데 필수적이다.

일단 비상 상황의 타입은 해당 상황을 해결할 수 있는 태스크의 타입이 결정하는데 쓰인다. 각 이벤트는 하나의 타입을 가질 수 있으며, 해당 이벤트를 해결할 수 있는 이벤트 핸들링 태스크의 타입이 미리 결정되어 있다. 위의 타입 매칭을 통해 결정된 타입의 태스크 중에서 QoS 요구사항을 만족시키는 QoS를 보유한 태스크들이 해당 비상상황을 해결하기 위한 태스크 후보가 된다. 각 이벤트는 하나 이상의 QoS 요구사항을 가질 수 있기 때문에, 이러한 비상 상황을 해결하기 위해서는 해당 이벤트의 모든 QoS 요구사항을 모두 만족시킬 수 있는 태스크를 골라야만 한다. 이렇게 결정된 태스크 후보 중에서도 비상 상황의 심각도를 해결할 수 있는 알맞은 QoS 레벨을 갖춘 태스크를 골라야 한다. 비상 상황의 심각도는 이벤트마다 하나의 값으로 결정되며, 비상 상황이 심각할수록 이 값이 높아진다, 이러한 심각도 값은 범위로 정해져 있으며 설정에 따라 최소 1에서 최대 n까지 가능하다.

이벤트 핸들링 태스크의 정보는 태스크의 이름, 타입, 제공하는 QoS의 타입(Quality of event handling type), 해당 QoS의 레벨(Quality level), 태스크를 구성하는 조합 서비스(Composite service), 해당 조합 서비스에 필요한 서비스 타입(Service Type)과 해당 서비스를 수행하는데 필요한 IoT 자원 타입(Resource type)으로 이루어져 있다. 태스크의 타입은 위에 설명 했듯이 특정 비상 상황에 필요한 태스크들의 타입을 매칭하기 위해서 필요하다. 또한 해당 태스크가 제공 가능한 QoS의 타입과, 각 QoS의 레벨이 저장되어 태스크 매치메이킹에 쓰인다. 이벤트 핸들링 태스크의 QoS 타입은 각 비상 상황에 필요한 QoS 요구사항과

매칭하여 해당 요구사항을 만족시키는 태스크들을 추려낼 수 있다. 또한 이러한 QoS 타입마다 QoS 레벨을 제공함으로써 비상 상황의 심각도에 따라 QoS 레벨을 맞추어, 충분한 QoS 레벨을 가진 태스크를 제공할 수 있게 한다.

또한 각 이벤트 핸들링 태스크를 구성하는 서비스 조합에 대한 정보를 저장한다. 태스크마다 하나의 서비스 조합이 있으며, 서비스 조합은 하나 이상의 서비스 타입들의 조합으로 이루어진다. 각 서비스 타입은 하나 이상의 IoT 기기가 필요하여, 해당 타입 정보들로 구성된다. 이 IoT 기기 타입의 정보는 각 태스크를 수행하는데 필요한 자원이 어떤 것인지 파악하는데 쓰인다. 이는 매치메이킹 과정에서 입력 값으로 주어지는 IoT 기기 사용 가능 여부와 비교하여 각 태스크가 수행 가능한지를 판단하기 위해 사용된다.

이처럼 위에 설명한 온톨로지는 태스크 매치메이킹을 위해서 필요한 정보와 해당 정보간의 관계를 정의하고, 이러한 정보를 알맞은 형태로 저장하는 데에 쓰이며, 저장된 정보는 4.2절에서 설명할 이벤트 핸들링 태스크 매치메이킹 알고리즘에서 사용된다.

4.2. 이벤트 핸들링 태스크 매치메이킹 알고리즘 설계

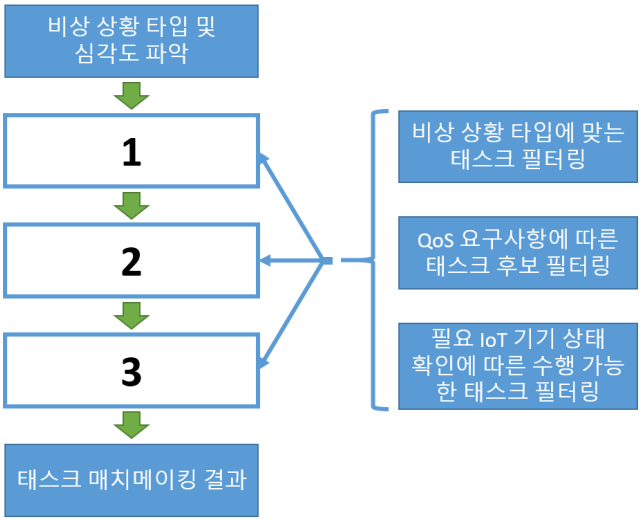


그림 4. 태스크 매치메이킹 알고리즘의 개요

3절에서 분석한 요구 사항을 정리하면 다음과 같다. 첫째, 비상 상황이 요구하는 QoS 요구사항을 모두 만족시키는 태스크가 존재할 때, 이러한 태스크를 찾아내 제공하여야 한다. 둘째, 태스크 제공시 태스크의 수행 가능 여부는 사용자의 요구를 만족시키는 데 가장 중요하기 때문에 태스크에 필요한 IoT 기기의 수행 상태를 파악하여 태스크가 제공 가능한지 확인하여야 한다.

위의 요구사항을 만족시키기 위해서 매치메이킹 알고리즘은 다음과 같은 단계로 이루어져 있다. 그림 5은 위의 매치메이킹 알고리즘의 첫째, 현재 제시된 비상 상황을 파악하고 이로부터 이벤트 핸들링 태스크 선택을 위한 쿼리를 생성한다. 이는 제시된 비상 상황의 타입을 파악하고, 해당 비상 상황이 요구하는 QoS 타입이 어떤 것인지 파악한다. 또한 이 주어진 비상 상황의 심각도에 따라서 각 QoS 요구사항의 레벨을 정한다.

둘째, 해당 쿼리로부터 비상 상황이 요구하는 태스크의 타입에 맞지 않는 태스크를 필터링 한다. 이는 비상 상황에 따라 그 비상 상황을 해결할 수 있는 태스크의 타입이 정해져 있기 때문이다. 이 과정에서는 해당 공간에서 가능한 모든 태스크 중에서 비상 상황이 요구하는 태스크 타입에 맞는 태스크만 남기고 나머지 태스크들은 필터링 한다.

셋째, QoS 요구사항에 따라 태스크를 필터링 한다. 이 단계에서는 비상 상황의 심각도에 따라 현재 검색된 태스크 후보 중에서 태스크의 QoS의 레벨의 제한에 미달하는 태스크들을 필터링 한다. 이는 비상 상황이 심각할 경우 QoS 레벨이 낮은 태스크들은 해당 상황을 해결할 수 없기 때문이다. 또한 해당 비상 상황에 여러 개의 QoS 요구사항이 존재하는 경우 이벤트 핸들링 태스크가 해당 비상 상황의 모든 QoS 요구사항과 QoS 레벨을 만족시켜야만 해당 비상 상황을 해결할 수 있다.

넷째, 현재 태스크 후보들이 수행 가능 가능 여부를 IoT 기기 상태 확인을 통해 확인하고, 수행 불가능한 태스크를 필터링한다. 세 번째 단계에서 QoS에 기반한 필터링을 하고 남은 태스크들은 모두 해당 상황을 해결할 수 있는 태스크들이다. 그러나 IoT 환경에서는 이러한 태스크들이 모두 수행 가능할 것이라는 보장이 없다. 이 때문에 남아있는 태스크들은 해당 상황에 수행 가능한지 확인을 위해 태스크 수행에 필요한 자원들의 상태를 모두 확인하여야 한다. 만약 최종적으로 남은 태스크가 없을 경우, QoS 요구사항 조정을 통해 QoS 레벨이 낮은 태스크를 선택할 수 있으나, 비상 상황임을 감안했을 때 이러한 조절이 상황 해결에 도움이 되지 않을 가능성이 있다.

위의 과정들을 모두 거치면 매치메이킹 된 태스크의 목록을 얻을 수 있다. 그러나 위에 설명된 둘째 단계에서 넷째 단계까지의 순서는 상황에 따라 변경될 수 있다. 이는 태스크 매치메이킹이 일어나는 환경과 해당 환경에서 일어날 수 있는 비상 상황, 그리고 해당 환경에서 수행 가능한 태스크의 특성에 따라 위의 세가지 필터링을 위 특성에 맞는 방식으로 적용하여 검색 공간(search space)을 효과적으로 줄이기 위함이다. 본 연구에서는 각 환경 및 비상 상황, 태스크의 특성을 여러 케이스로 나눠서 정의하고, 각 케이스마다 가장 효과적인 필터링 순서를 실험을 통해 알아보았다.

4.3. 태스크 매치메이킹의 환경 특성

4.2절에서 설명했듯이, 태스크 매치메이킹이

일어나는 상황은 태스크 매치메이킹이 일어나는 환경, 해당 환경에서 일어날 수 있는 비상 상황, 그리고 해당 환경에서 수행 가능한 이벤트 핸들링 태스크의 특성에 따라 달라질 수 있다. 위의 특성에 의해 태스크 매치메이킹 알고리즘 내부의 필터링 순서의 효과가 달라질 수 있다. 각 요소의 달라질 수 있는 특성을 정리하면 다음과 같다.

태스크 매치메이킹이 일어나는 환경의 특성은 해당 환경에 설치된 IoT 기기의 수와 해당 IoT 기기의 사용 가능 여부에 따라 결정된다. IoT 기기의 수는 태스크 매치메이킹 과정에서 IoT 기기 상태 확인 필터링 과정의 검색 공간의 크기와 직접적인 연관이 있다. 이 단계에서는 태스크 매치메이킹의 후보 리스트에 남아있는 모든 태스크마다 해당 태스크에 필요한 IoT 기기가 사용 가능한지, 즉 태스크가 수행 가능한지를 확인한다. 그 과정에서 확인해야 하는 IoT 기기의 수가 해당 환경에 설치된 IoT 기기의 수와 비례할 가능성이 높다. 또한 해당 환경에 설치된 IoT 기기의 사용 가능 여부 추세에 따라 태스크의 수행 가능 확률이 크게 좌우되기 때문에, IoT 기기 상태 확인 필터링의 순서에 따라 필터링 순서를 바꿀 경우 영향이 클 수 있다.

해당 상황에서 일어날 수 있는 비상 상황의 특성은 비상 상황의 타입의 수, 비상 상황당 QoS 요구사항 수, 비상 상황의 심각도에 따라 결정된다. 비상 상황의 타입의 수는 태스크 매치메이킹에 있어서 이벤트 핸들링 태스크의 타입의 수와 직접적인 연관이 있으며 타입의 수가 많아질수록 타입 기반 필터링에서 필터링 되는 태스크의 수가 많아질 수 있다. 또한 비상 상황당 QoS 요구사항 수와 비상 상황의 심각도는 증가할수록 QoS 기반 필터링에서 후보 태스크들이 필터링을 통과하기 어렵게 만든다.

이밴트 핸들링 태스크의 특성은 태스크의 수, 타입의 수, 해당 태스크의 QoS 종류와 QoS 레벨, 해당 태스크에서 필요한 IoT 기기의 종류 및 수가 결정한다. 태스크의 수는 태스크 매치메이킹 자체의 탐색 공간을 증가시키고, 태스크의 타입의 수는 위에서 설명했듯이 타입 기반 필터링에 큰 영향을 준다. 또한 태스크의 QoS 종류 및 레벨은 QoS 기반 필터링, 태스크에 필요한 IoT 기기의 종류와 수는 IoT 기기 상태 확인 필터링에 영향을 준다.

5. 실험 및 평가

이 절에서는 4장에서 제시한 태스크 매치메이킹 알고리즘 내부의 필터링 순서에 따라 변화하는 알고리즘의 성능을 다양한 태스크 매치메이킹 상황을 구성하여 시뮬레이션을 통해 평가한다. 태스크 매치메이킹 상황은 4.3절에서 논의된 바와 같이 태스크 매치메이킹이 일어나는 환경, 해당 환경에서 일어날 수 있는 비상 상황, 그리고 해당 환경에서 수행 가능한 이벤트 핸들링 태스크의 특성의 변수들을 조합하여 정한다.

표 1. 필터링 순서에 따른 패턴 조합

패턴 ID	필터링 순서
1	typeCheck, resourceCheck, QoSCheck
2	typeCheck, QoSCheck, resourceCheck
3	resourceCheck, typeCheck, QoSCheck
4	resourceCheck, QoSCheck, typeCheck
5	QoSCheck, resourceCheck, typeCheck
6	QoSCheck, typeCheck, resourceCheck

표 1은 필터링 순서에 따른 패턴 조합에 대한 표이다. 이는 세 가지 필터링 방식으로 이루어진 모든 필터링 조합 패턴을 나타낸 것이며, 위의 패턴의 순서에 따라 실험을 진행하였다. 위의 패턴 조합을 설정한 태스크 매치메이킹 상황에서 모두 실행하고, 어떤 조합이 해당 상황에서 가장 우수한 조건을 보이는지를 평가하였다.

이 시뮬레이션의 입력 값은 4.2절에서 제시한 상황 변수들에 기반하여 무작위적인 방법으로 생성하였다. 총 3가지 입력 값을 생성하였는데, 첫째로 IoT 기기의 상태는 IoT 기기의 수를 정하고 해당 IoT 기기가 사용 가능한지 확률적으로 정하였다. 이 확률은 상황 설정에 따라 다르게 부여하였다. 둘째, 비상 상황은 비상 상황의 타입 종류, 비상 상황당 최대 QoS 요구사항 수, 최대 및 최저 심각도를 매개변수로 하여 생성하였다. 셋째, 이벤트 핸들링 태스크는 태스크의 타입 종류, 태스크가 제공하는 QoS의 수 및 레벨의 최대/최저치를 설정하고 해당 태스크에 필요한 자원의 최대 수를 설정하여 생성하였다.

이 시뮬레이션은 위의 입력값에 기반하여 Intel(R) Core(TM) i7-3537U CPU @ 2.00GHz (4 CPUs) 64bit와 8GB 메모리를 사용하는 기기에서 수행되었다.

5.1. 기준 환경 상의 태스크 매치메이킹

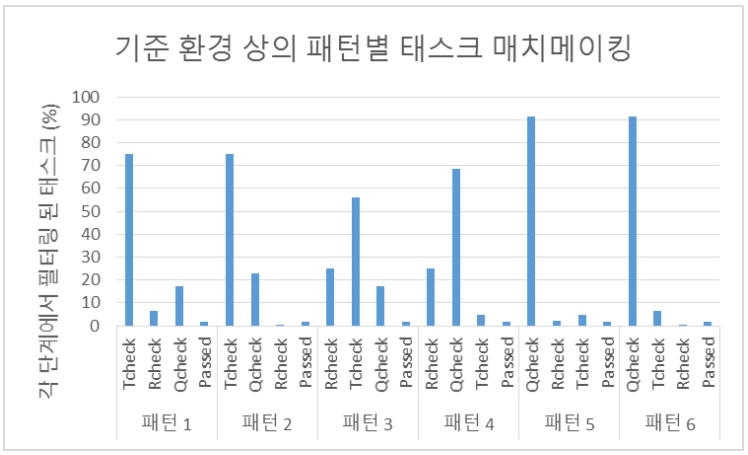


그림 5. 기준 환경 상의 패턴 별 태스크 매치메이킹

위 그림은 기준 환경상의 패턴별 태스크 필터링 결과를 보여준다. 기준 환경은 4.2절에서 논의된 상황

변수 중 모든 변수를 극단적이지 않은 범위에서 설정하였다. 예를 들어, 이벤트의 최저 심각도를 1, 최대 심각도를 10으로, 이벤트의 QoS 요구사항을 최대 3개로 설정하였고, 각 IoT 기기가 90% 확률로 사용 가능하도록 하였으며, 태스크에 필요한 최대 자원의 수를 4개, 최저 QoS 레벨을 3으로 설정하였다. 위의 결과는 3가지의 IoT 기기 상태를 설정하고, 100개의 이벤트에 대응하여 총 500개의 태스크를 위의 6개의 패턴을 대응하여 실험한 결과이다.

그러나 모든 입력 값들이 위에 설정한 범위 내에서 무작위적으로 생성되었기 때문에 이렇게 비교적 심각하지 않은 상황에서도 실제 태스크 매치메이킹이 되는 경우는 2%도 되지 않았다 (그림 5에서 Passed). 하지만 이는 본 논문에서 제시한 태스크 매치메이킹 알고리즘이 매치메이킹 할 수 있는 태스크를 찾지 못한 것이 아니라 실제 입력된 태스크 중에서 매칭 가능한 태스크가 적은 것으로, 본 논문에서 제시한 알고리즘은 모두 결정적(deterministic)으로 작동하여 매칭 가능한 모든 태스크를 찾는다.

위 그림을 보면, 대개 가장 먼저 수행된 필터링이 가장 많은 수의 태스크를 필터링한다. 특히 가장 많은 태스크를 필터링 하는 것은 QoS 요구사항 기반 필터링으로 이와 같은 결과를 패턴 5,6에서 볼 수 있다. QoS 요구사항 기반 필터링의 경우 여러 개의 QoS 요구사항이 있고, 각 요구사항마다 충족시켜야 하는 QoS 레벨이 정해져 있기 때문에 필터링을 통과하는 태스크가 상대적으로 매우 적다. 타입 기반 필터링이 가장 먼저 수행된 패턴 1,2의 경우 약 75%의 태스크가 필터링 되었다. 이는 이 실험에 사용된 입력 값을 총 4개의 타입 중에 비상 상황이 요구하는 1가지 타입의 태스크를 통과하도록 설정하였고, 입력 값이 무작위적으로 생성되었기 때문이다. 패턴 3,4의 경우는 현재 환경에서 각 IoT 기기가 90% 확률로 사용 가능하기 때문에 각 태스크가 필요로 하는 IoT 기기의 수가 여러 개이더라도, 상대적으로 많은 수의 태스크가 필터링을 통과한 것을 알 수 있다. 해당 필터링을 통과한 이후에는 타입 기반 필터링과 QoS 요구사항 기반 필터링이 대다수의 태스크를 필터링 하였다.

5.2. IoT 기기 작동 확률이 낮은 환경에서의 태스크 매치메이킹

IoT 기기의 작동 확률이 낮은 환경에서의 태스크 매치메이킹 실험은 5.1절의 환경 설정 상태에서 각 자원의 작동 확률을 50%로 낮추어 실험을 하였으며, 총 자원의 수는 30, 각 태스크에서 필요로 하는 자원의 수의 최대값은 4로 5.1절의 실험 환경과 같은 값으로 설정하여 실험하였다.

그 결과 패턴 3,4, 즉 IoT 기기 상태 확인 기반 필터링을 1순위로 하는 패턴에서 추가적인 태스크의 수가 필터링 된 것을 알 수 있었다. 이러한 결과는 그림 6에서 확인할 수 있다.

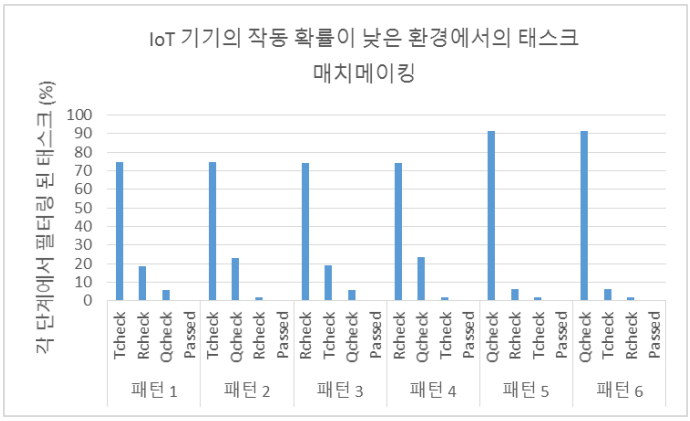


그림 6. IoT 기기 작동 확률이 낮은 환경에서의 태스크 매치메이킹

이는 5.1절의 실험에서 IoT 기기 상태 확인 기반 필터링이 25%의 이벤트 핸들링 태스크를 필터링한 것과 비교하였을 때, 75%로 약 3배수의 태스크를 필터링 한 것을 보여주었다. 이 경향은 각 태스크에서 필요로 하는 자원의 수가 더 높을 경우에 더 커질 것으로 예상된다. 즉, IoT 기기의 작동이 보장되기 힘든 경우 IoT 기기 상태 확인 필터링을 먼저 수행하면 탐색 공간을 효과적으로 줄일 수 있는 것으로 보인다. 이 때, 다른 제한 조건이 없는 경우 IoT 기기 상태 확인 필터링을 수행 한 후 타입 기반 필터링과 QoS 요구사항 기반 필터링의 차이는 크지 않음을 알 수 있다.

5.3. QoS 요구사항이 강화된 환경에서의 태스크 매치메이킹

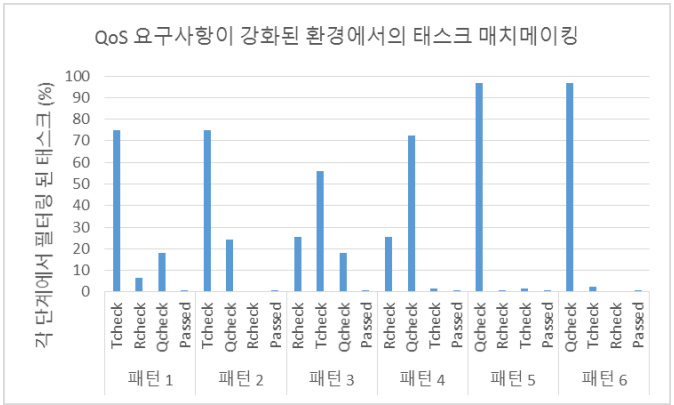


그림 7. QoS 요구사항이 강화된 환경에서의 태스크 매치메이킹

QoS 요구사항이 강화된 환경에서의 태스크 매치메이킹 실험은 5.1절의 환경 설정 상태에서 각 비상 상황의 심각도의 범위를 7~10으로 하고, QoS 요구사항의 최대 값을 4로 증가시켰다. 또한 이에 따라 태스크의 QoS 수도 최대 5로 증가시켜 실험하였다. 그 결과 패턴 5, 6에서의 QoS 요구사항 기반 필터링은 96% 이상의 필터링 성능을 보였다. 이는 태스크의

QoS 수를 증가시켰음에도 불구하고, 추가된 QoS 요구사항에 의하여 각 비상 상황을 해결할 수 있는 태스크의 수가 더욱 줄어들었기 때문이다. 이렇게 강화된 조건에 의해서 태스크 매치메이킹의 결과는 약 0.5%의 태스크만이 나올 수 있었다. 또한 패턴 3, 4에서도 QoS 요구사항 기반의 필터링에서 소폭의 증가가 있었다. 이를 보면 심각도의 범위가 더욱 증가하였을 경우, QoS 요구사항 기반 필터링이 대부분의 검색 공간을 줄일 것이라는 것을 알 수 있다.

6. 결론

IoT 환경에 설치된 IoT 기기의 수가 증가하면서 IoT 기기들이 제공하는 기능에 기반한 서비스의 숫자도 증가하게 된다. 이러한 서비스들은 사용자의 요구에 맞게 조합되어 태스크를 제공할 수 있다. 또한 IoT 기기가 수집하는 다양한 센서 데이터에 기반하여 해당 공간에서 일어나는 비상 상황을 감지하고, 이러한 비상 상황을 해결하기 위해 알맞은 태스크를 골라 수행하는 환경이 될 것이라고 예상할 수 있다.

본 논문에서는 이러한 비상 상황 해결을 위한 태스크를 이벤트 핸들링 태스크라 정의하고, 비상 상황 인지시 알맞은 이벤트 핸들링 태스크를 매치메이킹 하는 방안을 제시하였다. 이러한 태스크 매치메이킹 방안은 기존의 태스크 매치메이킹 방안과는 다르게, 실시간으로 필요한 IoT 기기의 상태를 확인함으로써 각 태스크가 수행 가능한지를 확인한다.

이를 위해서 비상 상황과 이벤트 핸들링 태스크의 속성과, 해당 속성들의 관계를 정의하는 태스크 매치메이킹을 위한 온톨로지를 정의하였다. 이 온톨로지는 태스크 매치메이킹에 필수적인 정보들을 정의하여 이를 효과적으로 저장하고 사용하는 데에 효과적이다.

또한 태스크 매치메이킹 알고리즘을 타입 기반 필터링, QoS 요구사항 기반 필터링, IoT 기기 상태 확인 기반 필터링의 세 가지의 필터링 방식으로 나누었고, 태스크 매치메이킹 과정에서 다양한 환경 요소들에 따라 효과적으로 탐색 공간을 줄이는 필터링 순서를 실험적으로 확인하였다. 그 결과, IoT 기기의 작동 확률이 낮은 경우 IoT 기기 상태 확인 기반 필터링을 우선적으로 하고 QoS 요구사항이 강화된 경우에는 QoS 요구사항 기반 필터링을 우선적으로 하였을 때 해당 필터링 단계가 탐색 공간을 효과적으로 줄이는 것을 시뮬레이션을 통하여 실험적으로 확인하였다.

향후 연구로는 탐색 공간을 줄이기 위하여 확률적 접근법을 이용하여 탐색 공간을 일차적으로 줄이고, 그 후 위의 세 가지의 필터링 방식을 순차적으로 적용하는 방안을 연구할 계획이다.

Acknowledgement

본 연구는 IoT 기반 스마트 홈 커뮤니티에서 안전하고 행복한 삶을 위한 소셜 매칭 및 소통 서비스 기술 개발(B0101-16-0334)의 결과로 수행되었음.

참고 문헌

[1] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54.15 (2010): 2787-2805.

[2] Jimenez-Molina, Angel, et al. "A task-oriented approach to support spontaneous interactions among users in urban computing environments." Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on. IEEE, 2010.

[3] Paolucci, Massimo, et al. "Semantic matching of web services capabilities." International Semantic Web Conference. Springer Berlin Heidelberg, 2002.

[4] Ankolekar, Anupriya, et al. "DAML-S: Web service description for the semantic web." International Semantic Web Conference. Springer Berlin Heidelberg, 2002.

[5] Cassar, Gilbert, et al. "A hybrid semantic matchmaker for IoT services." Green Computing and Communications (GreenCom), 2012 IEEE International Conference on. IEEE, 2012.

[6] Fenza, Giuseppe, Vincenzo Loia, and Sabrina Senatore. "A hybrid approach to semantic web services matchmaking." International Journal of Approximate Reasoning 48.3 (2008): 808-828.

[7] Martin, David, et al. "OWL-S: Semantic markup for web services." W3C member submission 22 (2004): 2007-04.

[8] Kritikos, Kyriakos, and Dimitris Plexousakis. "Novel optimal and scalable nonfunctional service matchmaking techniques." IEEE Transactions on Services Computing 7.4 (2014): 614-627.

[9] Alnahdi, Amany, Shih-Hsi Liu, and Austin Melton. "Enhanced Web Service Matchmaking: A Quality of Service Approach." 2015 IEEE World Congress on Services. IEEE, 2015.

포그 컴퓨팅 환경에서 서비스 QoS를 고려한 자동화된 서비스 인스턴스 선택 방법

권 정 현　　고 인 영

한국과학기술원

{junghyun.kwon, iko}@kaist.ac.kr

Automated Service-instance Selection by Considering QoS in Fog Computing Environments

Jung-Hyun Kwon　　In-Young Ko

Korea Advanced Institute of Science and Technology

요　약

본 연구에서는 포그 컴퓨팅 환경에서 동일한 서비스 인스턴스가 여러 곳에 분산되어 있을 때, 사용할 서비스 인스턴스를 결정하는 문제에 대해 다룬다. 문제 해결방안으로 각 서비스 인스턴스의 과거 Quality of Service(QoS) 데이터를 이용하여 회귀 모델을 만들고 그 모델을 이용하여 사용자 서비스 QoS 요구사항을 만족하는 서비스 인스턴스를 자동으로 선택하는 방법을 제시한다. 검증 실험에서는 제안 기법과 두 개의 베이스라인 기법을 사용하였을 때 서비스 응답 시간의 차이를 비교하였으며 제안 기법이 베이스라인 기법보다 평균 149~163 밀리초, 누적 14~16초 빠른 응답시간을 보였다.

1. 서　론

일상의 환경에 배치되는 사물인터넷(Internet of Things, IoT) 기기들이 늘어남에 따라 IoT 기기들로부터 생성되는 데이터의 양이 늘어나고, 그에 따라 IoT 데이터의 저장, 처리, 분석에 사용되는 데이터의 양도 늘어난다. Gartner는 2020년까지 97억개의 IoT 기기가 사용될 것으로 예상하고 있으며[1], IDC는 IoT 기기로부터 생성되는 데이터 중 40%가 저장, 처리, 분석에 사용될 것이라고 예상하고 있다[2].

아마존, 구글, 마이크로소프트 Azure 등의 클라우드 서비스는 IoT 데이터를 저장할 수 있는 많은 공간을 제공하고, 그 데이터를 분석하고 시각화할 수 있는 서비스도 제공한다. 하지만 현재의 클라우드 서비스는 향후 예상되는 IoT 기기들의 수에 비해 매우 적은 수의 데이터 센터에서 운영되고 있고 IoT 기기들은 지리적으로 분산되어 있다. 그래서 사용자가 IoT 데이터를 데이터 센터에 보내고 서비스에 대한 응답을 받기까지 많은 네트워크 지연시간이 발생할 수 있다[3, 4, 5]. 그리고 처리해야할 IoT 데이터가 늘어나면 클라우드 서비스에 많은 요청들이 동시에 들어오게 되고, 동시 요청이 많으면 요청 처리에 필요한 계산 자원이 부족해져서 서비스의 응답이 지연될 수 있다.

포그 컴퓨팅(Fog Computing)[3]은 앞서 제기한 문제를 해결하기 위해 등장한 개념으로서 IoT 기기와 지리적으로 인접한 컴퓨팅 노드(포그 노드)들을 두고 이 노드들이 해당 IoT 기기에서 발생하는 데이터를 처리하도록 하는 것이다. 이러한 포그 노드를 통해 IoT 기반 서비스 제공에 있어서의 네트워크 지연시간을 감소시킬 수 있고, 클라우드 서비스에 대한 요청이 포그 노드로 분산되기 때문에 클라우드 서비스 주변 네트워크의 부하를 감소시킬 수 있다.

포그 컴퓨팅과 관련된 연구로는 IoT 환경에서 IoT 기기와 가까운 컴퓨팅 노드를 활용하여 클라우드와 함께 분산처리하는 것이 있다. ANGELS는 사용자의 스마트폰과 같이 컴퓨팅 파워가 있는 엣지(Edge) 기기를 활용하였다[4]. 이 연구에서는 복잡한 계산을 요구하는 일은 클라우드에, 간단한 계산은 엣지 기기에 할당하여 클라우드의 부하를 줄이는 작업 파티셔닝 방법을 제안하였다. 다른 관련 연구로 Aura는 사용자가 IoT 기기와 다른 컴퓨팅 기기를 사용하여 애드혹(Ad-hoc) 클라우드를 형성하고, 사용자 태스크를 애드혹 클라우드의 노드들에 분산하여 수행하는 방법을 사용하였다[5]. 기존 연구에서 개선이 필요한 점으로 (1) 어떤 기준으로 작업 파티셔닝을 할지에 대한 자동화된 방법 제공과 (2) 사용자 태스크 수행을 위한 서비스 조합 차원에서 엣지 기기(포그 노드) 활용방안을 제시하는 것이 있다.

본 연구에서는 IoT 기반 서비스의 인스턴스(Instance)가 포그 노드와 클라우드에 동시에 배치되어 있을 때, 서비스 인스턴스 위치 선택을 자동으로 결정하여 사용자의 서비스 QoS 요구사항을

만족시켜주는 방법을 제안한다. 이를 위해 과거 서비스 수행에 대한 서비스 QoS 히스토리 정보를 이용하여 회귀 모델을 학습하고, 요청할 서비스 정보를 회귀 모델에 입력하여 클라우드와 포그 노드 중 어느 곳에 배치된 서비스 인스턴스를 사용할지 결정한다.

제안하는 방법에 대한 검증 실험에서 마이크로소프트 Azure 클라우드 서비스와 저사양 PC인 포그 노드를 사용하여 행렬 계산 서비스에 대한 응답시간을 비교하였다. 제안하는 기법을 사용하였을 때, 클라우드 또는 포그 노드만 사용하였을 때보다 평균적으로 약 149~163 밀리초 (누적 14~16초) 빠른 응답시간을 제공함을 확인하였다.

본 연구의 기여점으로 첫번째는 포그 컴퓨팅 환경에서 사용자 태스크 수행을 위한 서비스 조합 시 사용할 서비스 인스턴스의 선택문제를 제시한 점이다. 두번째로 어떤 비율로 클라우드와 포그 노드에 분산 처리를 할지 자동으로 결정하는 방법을 제안하였다는 점이다.

논문의 구성은 다음과 같다. 2장에서는 본 연구가 다루는 문제와 그 문제를 해결하기 위해 본 논문에서 제안하는 방법에 대해 설명한다. 3장에서는 검증 실험 및 그 결과에 대한 설명을 하고 4장에서 결론을 맺는다.

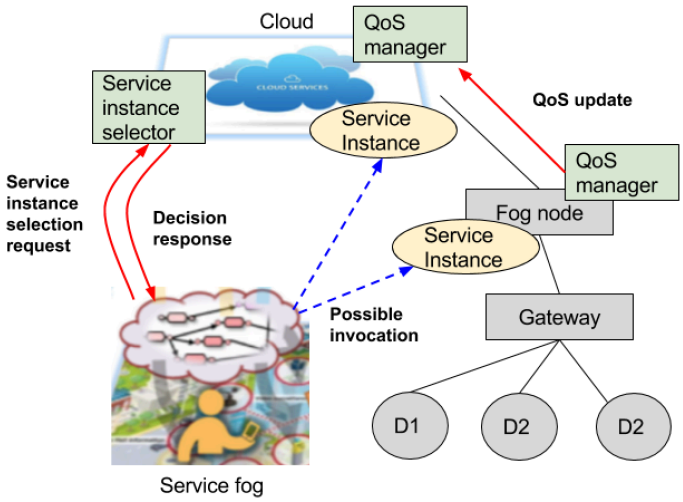


그림 1 포그 기반 서비스 제공 환경 개요

2. 서비스 QoS를 고려한 자동화된 서비스 인스턴스 선택 방법

그림 1은 본 연구가 다루는 환경을 보여준다. 사용자는 자신의 모바일 기기를 활용하여 필요한 태스크를 수행한다. 태스크는 여러 서비스들의 조합으로 표현되고, 그 서비스 조합을 실행함으로써 수행된다. 사용자가 태스크를 처음 구성할 때, 서비스 조합을 이루는 각 추상 서비스가 서비스 인스턴스에 바인딩된다. 바인딩이 완료된 서비스 조합을 우리는 *서비스 포그(Service Fog)*라고 부른다. 그림에서 사용자 태스크 수행을 위해 필요한 서비스 인스턴스는 IoT 기기들(D1, D2, D3)을 활용한다. 서비스 인스턴스는 IoT 기기에 인접한 포그 노드에 배치될 수 있지만 상대적으로 먼 클라우드에도 동시에 배치될 수 있다. 태스크의 추상서비스에 바인딩 될 서비스 인스턴스는 포그 노드 또는 클라우드에 배치된 서비스 인스턴스가 된다.

사용자는 포그 컴퓨팅환경에서 포그 노드와 클라우드 중 어떤 곳의 서비스 인스턴스를 사용해야 사용자의 서비스 QoS 요구사항을 만족할지 판단하기가 어렵다. 왜냐하면 사용자가 어떤 서비스를 사용하는지 또는 서비스의 어떤 기능을 사용하는지에 따라 QoS 요구사항을 만족하는 서비스 인스턴스 바인딩이 달라질 수 있기 때문이다. 예를 들어 사용자의 QoS 요구사항이 응답 시간의 최소화라고 하자. 서비스가 복잡한 계산을 수행해야 한다면 컴퓨팅 능력이 큰 클라우드의 서비스 인스턴스에 바인딩 하는 것이 효과적일 것이고, 간단한 계산을 수행한다면 컴퓨팅 능력은 작지만 사용자와 가까운 포그 노드의 서비스 인스턴스에 바인딩하는 것이 효과적일 것이다. 그리고 클라우드와 포그 노드의 컴퓨팅 성능에 따라서도 바인딩 결과가 달라질 수 있다. 따라서 사용자 서비스 QoS 요구사항을 만족하는 서비스 인스턴스를 자동으로 선택 해주는 방법이 필요하다.

사용할 서비스 인스턴스를 자동으로 선택하기 위하여 본 연구에서는 각 서비스 인스턴스의 과거 서비스 요청 정보와 QoS 데이터를 이용하여 회귀 모델을 학습시킨다. 그리고 학습된 회귀 모델은 사용자의 태스크 수행을 위해 어떤 서비스, 어떤 기능을 사용할지에 대한 정보를 바탕으로 가장 적은 응답시간이 예상되는 서비스 인스턴스를 추천해준다. 본 연구에서는 서비스 QoS 요구사항을 서비스 응답시간 최소화로 정하였다. 왜냐하면 포그 노드를 사용하는 중요한 이유 중 하나가 빠른 응답을 얻기 위해서 이기 때문이다. 하지만 본 모델은 서비스 수행 성공률, 서비스 이용 가격 등 다른 QoS 요구사항을 고려한 모델로도 확장 가능하며, 여러 QoS 요구사항을 고려할 경우에는 하나의 QoS 값 대신 효용함수(utility function) 값을 모델 구축에 사용할 수 있다[6].

그림 1은 본 논문에서 제안하는 방법과 관련된 요소들과 그들간의 상호작용을 보여준다. 사용자는 클라우드에 자신의 태스크 수행에 필요한 서비스 정보 및 QoS 요구사항을 보내 어떤 서비스 인스턴스를 사용할지 추천을 받는다. 클라우드에서는 자신의 서비스 인스턴스와 포그 노드의 서비스 인스턴스에 대한 과거 QoS 데이터를 QoS 매니저가 관리한다. 포그 노드에도 QoS 매니저가 있어서 자신의 서비스 인스턴스에 대한 일정 기간 동안의 QoS 데이터를 축적하고 클라우드에 전송한다. 클라우드에서는 축적된 QoS 데이터를 바탕으로 회귀 모델을 학습시킨다. 이후, 학습된 회귀

모델을 사용하여 사용자의 서비스 인스턴스 선택 요청을 분석하고 가장 응답시간이 낮게 예상되는 서비스 인스턴스를 추천한다.

회귀 모델을 구축하기 위해 우리는 회기 알고리즘으로 많이 사용되는 랜덤 포레스트(Random Forest)[7]를 사용하였다. 모델 학습을 위한 훈련 데이터로는 과거 서비스 QoS 데이터를 사용하였다. QoS 데이터의 각 레코드는 특징 값(feature) 벡터와 라벨(label)로 구성되어 있다. 특징 값 벡터는 서비스 요청 메시지에서 알 수 있는 정보로서 서비스 이름, 서비스 기능 명칭, 기능의 파라미터 이름, 기능의 파라미터 값, 서비스 요청 메시지의 길이 등으로 구성된다. 라벨은 서비스 응답 결과로 알 수 있는 QoS 값으로 본 연구에서는 응답시간을 사용하였다. 이러한 특징 값 벡터와 라벨을 사용해서 랜덤 포레스트 회귀 모델을 학습시킨다. 학습된 회귀 모델은 임의의 특징 값 벡터에 대한 라벨, 즉 응답시간을 예측할 수 있다. 이러한 회귀 모델은 기본적으로 각 서비스 인스턴스마다 구축된다. 각 회귀 모델로 예측한 응답시간 중 가장 낮은 응답시간으로 예측된 서비스 인스턴스의 위치를 사용자에게 알려준다.

3. 실험 및 결과

본 연구에서 우리는 제안한 서비스 인스턴스 선택 방법의 효과를 확인하기 위해 실험을 진행하였다. 우리는 본 기법이 베이스라인(Baseline) 기법보다 얼만큼의 응답 시간 감소 효과를 보였는지에 대해 실험하였다. 이 실험에서는 세가지 방법(본 연구에서 제안하는 방법, 포그 노드의 서비스 인스턴스만 선택하는 방법, 클라우드의 서비스 인스턴스만 선택하는 방법)에 대한 응답시간을 측정한다.

실험 환경을 구성하기 위해 우리는 행렬 곱셈 연산을 제공하는 서비스를 구현하였다. 행렬 곱셈 연산을 서비스로 구현한 이유는 행렬 곱셈이 IoT 센서 값 처리 및 분석에 많이 사용되기 때문이다. 구현된 서비스를 마이크로소프트 클라우드인 Azure[1]에 배치하여 서비스 인스턴스를 생성하였다. 포그 노드로는 저사양 PC를 이용하였다. 저사양 PC의 CPU는 AMD E-350이며 RAM은 4GB이다. 포그 노드에도 마찬가지로 서비스 코드를 실행하여 서비스 인스턴스를 생성하였다.

회귀 모델에 학습시킬 서비스 QoS 데이터는 각 서비스 인스턴스에 대해 100개의 서비스 요청 메시지와 응답 시간을 수집하여 생성하였다. 각 서비스 요청 메시지는 3에서 250 사이의 무작위 크기의 행렬 정보를 포함한다. 그리고 네트워크 지연을 포함한 응답시간을 측정하기 위해 각 서비스 요청 메시지마다 응답이 오기까지의 시간을 측정하였다. QoS 특징 값 벡터는 행렬 크기, 행렬 값으로 구성하였고, 라벨은 응답시간으로 구성하였다. 회귀 모델에 대한 성능을 측정하기 위한 테스트 QoS 데이터도 같은 방법으로 생성하였으며, 각 서비스 인스턴스 당 100개씩 생성하였다.

실험 과정은 다음과 같다. 각 인스턴스의 회귀 모델에 테스트 QoS 데이터를 입력으로 주었을 때, 출력되는 예상 응답시간 결과값을 서로 비교해서 작은 응답시간을 제공하는 서비스 인스턴스를 선택한다. 같은 방법으로 나머지 테스트 QoS 데이터에 대해서도 서비스 인스턴스를 선택한다. 본 접근방법이 선택한 100개의 서비스 인스턴스에 대한 실제 응답시간을 알기 위해 테스트 QoS 데이터로부터 각 서비스 인스턴스에 대한 응답시간을 추출한다. 두 개의 베이스라인 기법의 응답시간도 마찬가지로 테스트 QoS 데이터로부터 추출한다. 그리고 각 기법에 대한 응답 시간의 평균과 누적값을 비교한다.

표 1 응답시간 통계 (단위: 밀리초)

	cloud only	*fog only*	*proposed approach*
mean	1,062.33	1,048.61	899.23
std	717.22	1,030.12	834.88
min	102.71	14.34	14.34
25%	441.78	100.82	100.82
50%	974.68	726.29	726.29
75%	1,602.50	1,936.77	1,569.16
max	3,525.59	3,345.39	3,525.59

표 1은 제안한 방법과 베이스라인 방법의 응답시간 결과에 대한 통계를 보여준다. 제안한 방법의 결과는 "proposed approach"열에, 클라우드만 사용한 베이스라인 방법의 결과는 "cloud only"열에, 포그 노드만 사용한 베이스라인 방법의 결과는 "fog only"열에 표시하였다. 통계 값으로 평균, 표준편차, 최소값, 최대값, 25%, 50%, 75%에 위치하는 값을 표시하였다. 표에서 볼 수 있듯이, 제안한 방법이 다른 베이스라인 방법보다 평균적으로 149~163 밀리초 빠른 응답시간을 보임을 알 수 있다.

서비스가 반복적으로 수행될수록 본 연구에서 제안한 방법의 효과는 누적되어 많은 응답시간 단축의 효과를 얻을 수 있다. 그림 2는 제안한 방법과 베이스라인 방법을 사용했을 때의 누적 응답 시간을 보여주는 그래프이다. X축은 각 테스트 QoS 데이터를 의미한다. 즉, 클라우드 또는 포그 노드에 100번의 서비스 요청을 하여 수집한 결과이다. Y축은 각 요청에 대한 응답 시간의 누적치를 나타낸다. 제안한 방법은 빨간색

[1] https://azure.microsoft.com/ko-kr/

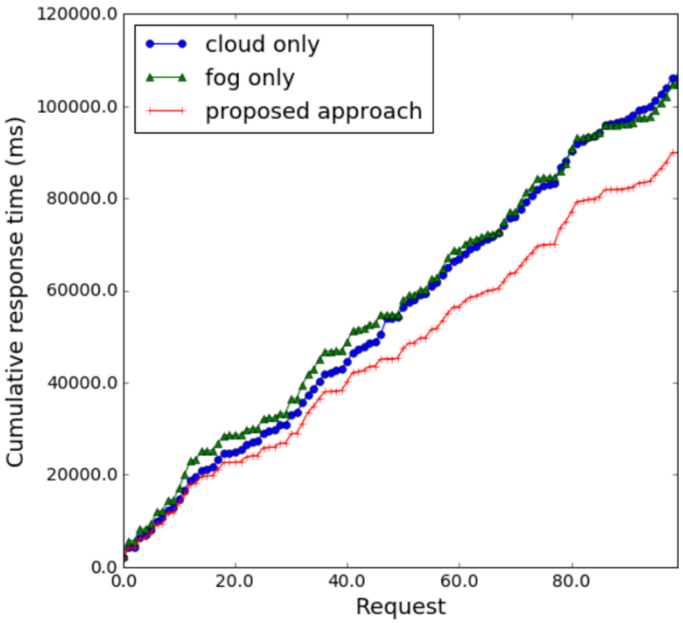


그림 2 누적 응답 시간 비교

더하기로 표시되어 있으며, 클라우드만 사용한 베이스라인 방법은 파란색 동그라미로, 포그 노드만 사용한 베이스라인 방법은 녹색 세모로 표시하였다. 서비스 요청이 누적될수록, 제안한 방법을 사용하였을 때의 누적 응답 시간과 베이스라인 방법을 사용하였을 때의 누적 응답 시간과의 차이가 점점 벌어짐을 알 수 있다. 최대 누적 응답 시간 차이는 14~16 초이다.

본 연구를 통해 제안한 방법이 빠른 응답시간을 보이는 이유는 과거 QoS 데이터로 학습한 회귀 모델이 낮은 응답시간을 제공하는 서비스 인스턴스에 대한 결정을 정확하게 하였기 때문이다. 우리는 원인을 좀 더 자세히 파악하기 위해 본 회귀 모델이 클라우드와 포그를 어떤 비율로 선택하였는지 분석하였다. 본 회귀 모델은 100개의 테스트 요청 중 37개의 요청의 경우에는 클라우드를 선택하였고, 나머지 63개의 요청에 대해서는 포그 노드를 선택하였다. 그리고 이러한 선택은 92%의 정확도를 보였다. 즉, 본 접근 방법이 선택한 100개의 서비스 인스턴스 중 92개의 서비스 인스턴스가 선택하지 않은 서비스 인스턴스보다 실제로 더 낮은 응답시간을 제공했음을 의미한다.

4. 결 론

본 논문에서는 포그 컴퓨팅 환경에서 서비스 인스턴스 선택 문제에 대해 논의하였다. 문제 해결방안으로 각 서비스 인스턴스의 과거 QoS 데이터를 이용하여 회귀 모델을 만들고 그 모델을 이용하여 사용자 서비스 QoS 요구사항을 만족하는 서비스 인스턴스를 자동으로 선택하는 방법을 제시하였다. 실험에서는 제안한 방법과 두 개의 베이스라인 방법을 사용하였을 때 서비스 응답시간 차이를 비교하였으며, 제안한 회기 모델 기반의 방법이 베이스라인 방법보다 평균 149 ~ 163 밀리초, 누적 14 ~ 16초 빠르게 응답함을 보였다.

향후 연구 계획은 다음과 같다. 첫번째로 실험에 사용할 서비스를 더욱 다양화하고, 서비스 인스턴스를 다수의 포그 노드에 배치하여 실험을 하는 것이다. 두번째로 포그 컴퓨팅 환경에서 응답시간 이외에 다른 중요한 서비스 QoS 요소가 있는지 파악하고 그 요소를 본 접근방법이 고려할 수 있도록 확장하는 것이다.

사 사

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2016R1A2B4007585)

참고문헌

[1] Gartner, Inc., “Gartner Says Smart Cities Will Use 1.1 Billion Connected Things in 2015”, March 18, 2015 http://www.gartner.com/newsroom/id/3008917

[2] IDC, “IDC FutureScape: Worldwide Internet of Things 2015 Predictions”, Dec, 2014

[3] Bonomi, Flavio, et al. "Fog computing and its role in the internet of things." Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, 2012.

[4] Mukherjee, Arijit, et al. "Angels for distributed analytics in iot.", IEEE World Forum on Internet of Things (WF-IoT), p. 565-570, 2014.

[5] Hasan, Ragib, Md Mahmud Hossain, and Rasib Khan. "Aura: an IoT based cloud infrastructure for localized mobile computation outsourcing.", IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, p. 183-188, 2015.

[6] Zeng, Liangzhao, et al. "Qos-aware middleware for web services composition." IEEE Transactions on software engineering 30.5, p. 311-327, 2004.

[7] Random forest. (2016, December 5). In Wikipedia, The Free Encyclopedia. Retrieved 16:31, December 5, 2016, from https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=753170837

IoT 기반의 실시간 가축 건강 및 번식 관리를 위한 모바일 어플리케이션 개발

김희진 ○*, 오세은*, 안세혁**, 최병주*
*이화여자대학교 컴퓨터공학과 **주식회사 유라이크코리아
master@ulikekorea.com, osen@ewhain.net, anse@ulikekorea.com, bjchoi@ewha.ac.kr

Development of IoT-based Mobile Application for Livestock Healthcare and Breeding Management in real time

Heejin Kim○*, Seeun Oh*, Sehyeok Ahn**, Byoungju Choi*
*Dept. of Computer Science and Engineering, Ewha Womans University
**uLikeKorea Co., Inc.

요 약

본 연구에서는 IoT 센서로부터 수집된 체온 데이터를 바탕으로 가축의 건강과 번식 상태를 모니터링 하기 위한 모바일 어플리케이션을 개발한다. 가축 건강에 이상이 발생한 경우 모바일 어플리케이션 경고 메시지를 통해 사용자에게 실시간으로 알리고 또한 가축의 번식 예정일을 제공해 사용자가 가축의 번식 시기를 놓치지 않고 대응할 수 있도록 한다.

1. 서론

구제역, AI 등의 질병은 단 한 번의 발생으로 농가에 막대한 피해를 입히며 생산 및 공급 불안정으로 인한 경제 손실은 시장 전체에 영향을 준다. 이에 가축 질병 예찰과 건강 모니터링에 관한 연구가 꾸준히 진행되어 왔다[1][2]. 또한 국내 100두 이상의 대규모 축우 사육 농가의 증가로 IT 기술을 활용한 효율적이고 정확도 높은 가축 건강 모니터링이 요구되는 상황이다[3].

하지만 아직까지 가축 생체 데이터 수집 및 이를 활용한 연구가 부족하고, 가축 건강 및 번식 관리를 위한 관련 연구는 매우 미흡한 상황이다.

따라서 본 연구에서는 질병 예찰과 번식 성공률 향상을 위해 IoT 기반의 경구투여용 센서를 이용하여 가축의 체온 데이터를 실시간으로 수집하고, 이를 바탕으로 가축의 건강 상태와 번식 예정일을 판단하는 가축 모니터링 모바일 어플리케이션을 개발하고자 한다.

2. 관련연구

가축 경구투여용 센서는 귀, 목, 발목 등에 장착되는 부착형 센서에 비해 외부 환경의 간섭을 받지 않아 정확한 생체 데이터를 얻을 수 있다. 또한, 가축 체내에 안정적으로 안착되어 미세한 체온 변화를 감지하고 지속적인 데이터 수집이 가능하다[3][4]. 가축의 체온은 질병 발생 혹은 발정, 분만 등 생체 변화에 따라 달라지므로, 가축의 체온을 지속적으로 모니터링 함으로써 건강 및 번식 상태를 판단할 수 있다.

이와 같은 IoT 기반의 경구투여용 센서를 이용하여 본 연구에서는 실시간 가축 건강과 번식 관리를 위한 모바일 어플리케이션을 개발한다.

3. IoT 기반의 가축 관리 모바일 어플리케이션

3.1 어플리케이션 설계

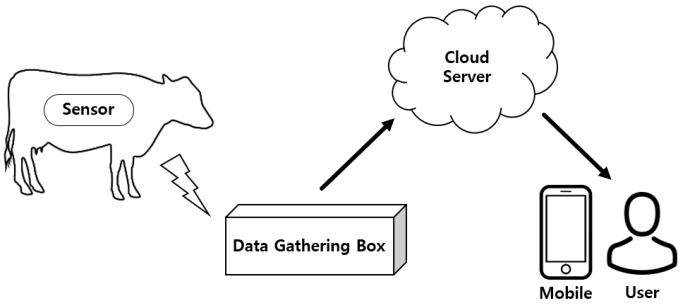


그림 1 모바일 어플리케이션 동작 개념도

그림 1은 개발한 모바일 어플리케이션 동작 개념도를 나타낸 것이다. 가축에 경구 투여된 IoT 센서는 가축 체내에서 실시간으로 체온을 측정한다. 측정된 체온 데이터는 데이터 취합 장비에 의해 수집되고, 클라우드 서버를 통해 사용자에게 전달된다.

IoT 센서를 적용한 가축 정보를 어플리케이션에 등록하면, 클라우드 서버에 저장된 가축 체온 데이터를 실시간으로 모니터링 가능하다. 또한 해당 가축의 건강 상태 변화가 감지되거나 번식 예정일이 되었을 때 어플리케이션을 통해 알람을 받아 볼 수 있다. 알람을

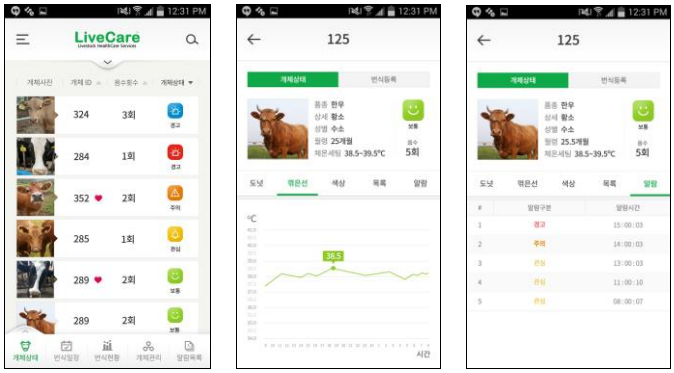
받은 사용자는 가축의 체온 기록을 확인하여 가축 상태를 판단하고 이에 따라 적절한 조치를 취한다.

3.2 어플리케이션 구현

본 논문의 모바일 어플리케이션은 개체의 건강 상태를 실시간으로 확인할 수 있는 개체상태와 번식 내용을 등록하고 예정일을 확인할 수 있는 번식일정, 번식현황이 주 기능으로 구성되며, 안드로이드 기반으로 개발되었다.

개체상태는 경구투여용 IoT 센서에 의해 수집된 체온 데이터와 가축의 건강 상태로 표시되며, 어플리케이션에 등록된 전체 가축의 건강 상태를 체온 상승과 체온 하락에 대한 경고, 주의, 관심, 보통 항목으로 나타낸다.

그림 2는 가축의 건강 관리를 위한 개체상태를 구현한 UI 화면들이다. 그림 2(a)는 전체 개체들의 상태를 보여주는 UI 화면으로 현재 임신상태, 음수 횟수, 체온 상태를 보여주며, 사용자가 전체 가축 상태에 대해 직관적으로 이해할 수 있도록 건강 상태 항목을 서로 다른 색상의 아이콘으로 표시했다. 그림 2(b)는 상세 개체상태를 보여주는 UI 화면으로 실시간 수집된 체온 데이터를 그래프 형식으로 나타내 가축의 체온 변화 추이를 한눈에 파악할 수 있다. 그림 2(c)는 개체상태 알람 목록을 보여주는 UI 화면으로 체온이 일정 범위 이상 상승 또는 하락하는 경우 경고 메시지 목록을 표시한다.

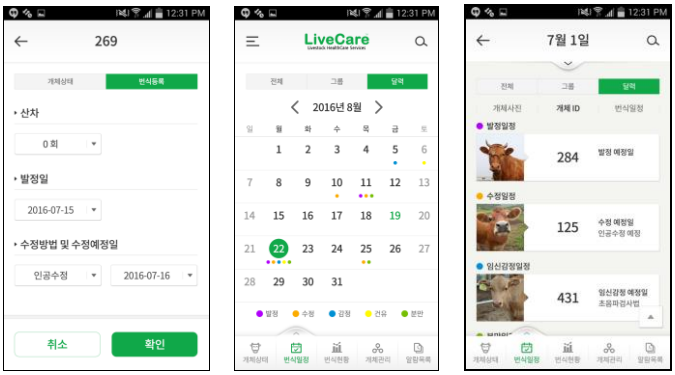


(a) (b) (c)

그림 2 어플리케이션 건강관리 UI

번식일정은 사용자가 발정, 수정, 임신, 분만과 같은 가축의 번식 내용과 계산된 예정일을 등록하면, 이를 달력 형식과 날짜 별 리스트 형식으로 표시한다. 번식현황은 번식에 사용된 정액번호나 분만 결과와 같은 상세 번식 내용을 나타낸다.

그림 3은 가축의 번식 관리를 위한 번식일정, 번식현황을 구현한 UI 화면들이다. 그림 3(a)는 번식등록 UI 화면으로 가축의 번식 내용과 계산된 예정일을 입력할 수 있다. 그림 3(b)는 번식일정 UI 화면으로 번식 예정일을 쉽게 파악할 수 있도록 달력 형식으로 구현했다. 번식이 예정된 날짜에는 발정, 수정, 감정, 건유, 분만과 같은 번식 내용에 따라 서로 다른 색상의 아이콘이 표시된다. 이 화면에서 특정 날짜를 선택하면 그림 3(c)와 같이, 날짜 별 번식일정 UI 화면이 나타나고, 해당 일에 속하는 가축의 번식 예정 일정을 리스트 형식으로 표시한다.



(a) (b) (c)

그림 3 어플리케이션 번식관리 UI

4. 결론

본 연구에서는 IoT 기반의 실시간 가축 건강 및 번식 관리를 위한 모바일 어플리케이션을 개발하였다. 개발한 모바일 어플리케이션을 통해 농장주 및 가축 관리자는 시간과 장소에 구애받지 않고 가축의 건강 상태를 간편하게 모니터링할 수 있다. 뿐만 아니라, 알람을 통해 건강에 이상이 생긴 가축에 시기적절한 대응이 가능하여 경제적 피해를 절감할 수 있으며, 가축의 번식 일정을 미리 파악해 발정, 수정, 분만에 대한 준비를 할 수 있다.

현재 개발한 모바일 어플리케이션은 경구투여용 IoT 센서와 함께 국내 축산 농가에 보급되어 실제 사용되고 있으며, 가축 질병에 대한 빠른 대처와 가축 번식률 향상에 기여하고 있다.

Acknowledgement

“본 논문은 ㈜유라이크코리아의 지원을 받아 수행된 연구임.”

참고문헌

[1] 박명철, 정현철, 하옥균. "가축 질병 예방을 위한 가축 생체정보 모니터링 장치의 개발." 한국컴퓨터정보학회논문지, 21(10), 91-98, 2016

[2] 김양범, 김정국, 최동운. "IoT 기반의 한우 생체정보 수집용 단말기 개발." 한국엔터테인먼트 산업학회논문지, 10(3), 319-327, 2016

[3] 김희진, 안세혁. "경구투여용 센서를 통한 축우 발정탐지 방법 및 사례분석." 한국멀티미디어학회 추계학술발표대회 논문집, 19(2), 1324-1325, 2016

[4] LiveCare 라이브케어. http://www.livecare.kr

다중 조명 제어 기기간의 최적화를 위한 자가-적응 의사 결정 방법

김현우 ○, 이의종, 백두권†
고려대학교 정보대학 컴퓨터학과
{khw0809 , kongjjagae, baikdk}@korea.ac.kr

Self-adaptive Decision-making Method for Optimization among Multi-light Control Device

Hyunwoo Kim○, Euijong Lee, Doo-kwon Baik†
Dept. of Computer Science and Engineering, College of Informatics, Korea University

요 약

본 연구에서는 자가-적응 소프트웨어를 기반으로 한 조명 제어 시스템을 바탕으로 자연 광원과 인공 광원을 포함한 다중 조명 제어 환경을 제안한다. 이는 다중 조명 환경에서 제어 기기간의 최적화를 위한 자가-적응 의사 결정 방법을 제안하여 다중 조명 환경에서 에너지 소비를 효율적으로 할 수 있게 한다. 더불어 사용자의 편의성을 우선시해 사용자가 원하는 조명 환경으로 실시간으로 변경 하게 하는 것을 목적으로 한다. 본 연구를 통해 기존 자가-적응 조명 제어 시스템의 한계를 극복했으며 제안하는 기법을 응용해 추후 요구사항이 상충되는 사물 인터넷 환경에서의 최적화를 위해 응용이 가능할 것으로 판단된다

1. 서 론

현재 전세계에는 20억이 넘는 사람들이 인터넷을 사용하여 의사소통을 하고 정보를 주고 받으며 살고 있다. 네트워크 기술의 발전으로 인해 전 세계의 사람들은 네트워크 통신망인 인터넷을 사용해 수 많은 정보들을 서로 주고 받을 수 있게 되었다. 또한 최근 소형 네트워크 기기의 발전으로 인해 소형 전자기기에서도 통신이 가능하도록 하드웨어 기술이 발전하게 되었고, 이는 모든 것들이 인터넷에 연결되는 사물인터넷(Internet of Things)이라는 분야를 탄생시켰다. 사물인터넷을 활용해 사용자의 주변 환경의 객체들로부터 정보를 주고 받으며 능동적으로 주변환경에 반응하는 것을 목표로 하고 있다.

사물인터넷에 관한 연구는 센서, 보안, 네트워크, 인식, 등 전분야에 걸쳐서 계속 진행 중이다 [1]. 사물인터넷 연구 분야 중 센서 시스템을 기반으로 한 분야는 제한된 범위 안에 단순한 감지 및 적용을 하는데 국한되고 있다 [1].

†: 교신저자 (백두권)

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No.2012M3C4A7033346).

자가-적응 소프트웨어(Self-Adaptive Software)는 주변 환경을 탐지하고 소프트웨어의 요구사항에 따라 스스로 소프트웨어 자신을 변화시키는 소프트웨어이다 [2]. 환경에 따라 스스로 적응하는 자가-적응 소프트웨어의 특징은 실생활과 가장 밀접하기 때문에 예상치 못한 환경의 변화가 잦은 사물인터넷 분야에 접목 될 수 있다. 본 연구는 센서를 기반으로 하는 사물인터넷 환경에서 센서에 탐지된 결과에 따라 시스템이 자가-적응을 수행 할 수 있도록 하는 다중 조명제어 환경을 제안하고자 한다. 앞서 말한 자가-적응 소프트웨어는 동적으로 변하는 환경에 요구사항에 맞춰 사용자의 조작 없이 시스템 스스로 변화 하는 시스템이다[2]. 그렇기 때문에 동적으로 변하는 환경에 맞춰 시스템이 변경되는 조명제어 시스템에 자가-적응 소프트웨어를 접목시킨다면 사용자의 선호도에 따라 조명이 제어가 되고 에너지도 효율적으로 관리 할 수 있게 된다. 하지만 기존에 제안된 자가-적응을 활용한 조명 제어 시스템은 단일 인공 광원에 국한되어 있어 자동 제어 커튼을 활용한 자연 광원이 포함 되어 있는 환경에서는 적합하지 않다. 그렇기 때문에 자연 광원을 포함한 시나리오에서는 다중 조명 제어 기기간의 의사 결정 방법이 필요하다.

본 논문에서는 다중 광원 상태의 자가-적응

시나리오를 제안하고 다중 조명 제어 기기간의 최적화를 위해 자가-적응 의사 결정 방법을 제안한다.

2. 관련 연구

2.1 자가-적응 소프트웨어(Self-Adaptive Software)

자가-적응 소프트웨어(Self-Adaptive Software)는 소프트웨어 스스로 자신의 상태와 주변환경을 파악하고 만약 요구사항의 위반이 생겼을 때 스스로 처리 할 수 있는 소프트웨어를 말한다 [2]. 자가-적응 소프트웨어는 자가-적응 소프트웨어의 컨트롤 루프 중 하나인 MAPE-K Loop로 구성되며 이는 크게 4단계가 있다.

첫번째 단계는 감시(Monitoring)단계 이다. 감시 단계에는 시스템이 시스템의 상태를 외부 센서(Sensor)를 통해 시스템 및 외부요인들을 감시하는 과정이다. 이 과정에 나온 정보들을 시스템 내부의 지식 저장소(Knowledge Base)에 저장해 놓는다. 두번째 단계는 분석(Analysis)단계 이다. 분석 단계는 감시 단계에서 나온 정보들의 패턴을 분석하는 단계이다. 지식 저장소로부터 나온 정보들의 패턴을 분석하여 이상 증상을 진단하고 향후 참조를 위해 증상을 지식 저장소에 저장하게 된다. 세번째 단계는 계획(Planning) 단계이다. 계획 단계에는 분석된 증상들을 해석하고 해석한 결과들을 바탕으로 이펙터(Effector)를 통해 문제 해결을 위한 과정들을 계획하는 역할을 한다. 마지막으로 네번째 단계는 실행(Execution) 단계이다. 계획 단계에서 나온 계획을 바탕으로 이펙터에 변경 명령을 내리게 된다. 센서와 이펙터는 시스템 외적으로 다수의 집합으로 구성되어 있을 수 있다. 적용이 된 후에 다시 센서를 바탕으로 모니터링 단계를 수행하며 유지하고 4단계를 반복한다[3].

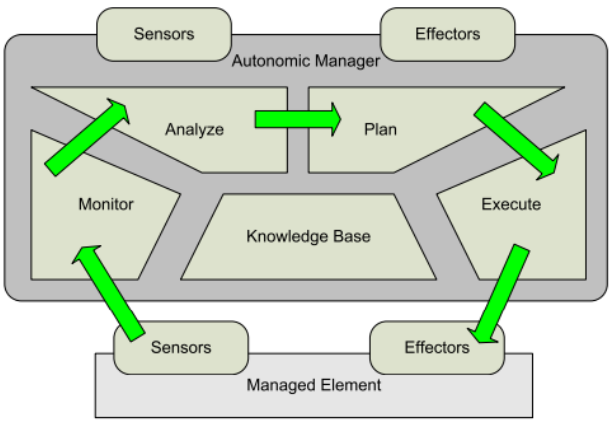


Figure 1 MAPE-K Loop [3]

자가-적응 소프트웨어가 적용 된 연구로는 [4,8-10] 등 이 있다. [4]는 산불 감지 센서 시스템에 자가-적응 소프트웨어를 접목시켜 감지된 데이터를 바탕으로 사전에 감지 주기를 조절하여 기존의 규칙 기반 센서 시스템보다 효율적 임을 밝혔다. [8]은 자동 교통 라우팅 판단 문제를 자가-적응을 활용하여 다수의 사용자가 생성하는 지식데이터를 하나의 시스템으로 관리하여 효율적인 길 배분을 해 기존 최단거리를 활용한 교통 라우팅 시스템에 비해 성능 향상을 시뮬레이션을 통해 보여주었다. [9]는 실시간으로 동작하는 자가-적응 소프트웨어를 기반으로 한 네트워크 시스템 프레임워크를 제안하였고 [9]의 프레임워크를 [10]에서는 Znn.com이라는 가상의 사이트를 통해 구현하여 제안한 결과의 효과를 기존의 시스템과 비교해 효과적임을 증명하였다.

2.2 센서 기반 조명 제어 시스템

기존의 센서 기반의 조명 제어 시스템의 연구는 [5,6] 등이 있다. [5]에 나오는 지능형 조명 제어 시스템은 사용자의 편의와 운영 비용의 최적의 절충점을 찾아 효용 기반의 전략을 도출해내는 방법을 제안한다. 해당 연구는 정의되어 있는 규칙에 따라 시스템이 변화하고 시스템의 상태를 지속적으로 예측하고 계산하기 때문에 유지비용이 높다는 한계점이 있다.

[6]은 퍼지 신경 네트워크 기반으로 한 블라인드 조절 시스템을 제안하였다. 이 시스템은 광량에 따라 블라인드의 각도를 퍼지 신경 네트워크를 바탕으로 가장 효율적인 각도를 계산하게 되고 이를 토대로 블라인드의 각도를 조절해 사용자의 시인성과 에너지의 효율성을 최적화시키는 연구를 진행했다. 해당 연구는 블라인드 조절 시스템이기 때문에 자연 광원의 조절에 국한되어 실내의 내부 인공조명과의 연계가 힘든 한계점이 있다.

상기 언급된 연구들은 시스템이 주변 환경을 파악하지만 기존에 정의되어 있는 규칙과 사용자 선호도를 사용해 광량을 제어하는 방법으로 사용자의 선호도 변화에 따른 적응이 어렵다는 단점이 존재한다. 이러한 문제를 해결하고자 본 논문에서는 기존 연구 중 조명제어 시스템과 자가-적응 소프트웨어를 적용한 광원 제어 환경을 제안했다 [7,11]. 하지만 이 연구는 단일 인공 조명만이 적용된 사례로 사물인터넷환경과 같이 다중 광원이 존재하는 경우에는 적용이 어려운 한계점이 존재한다

본 논문에서는 상기 언급된 연구들의 문제를 해결하고자 적응 형 조명 제어 소프트웨어에 자가-적응 형 소프트웨어의 MAPE-K를 적용 시켰다. 또한 단일 광원이 아닌 다중 광원에 대한 최적화를 위한 방법을 제안함으로 사물인터넷 환경으로의 확장을 도모하고자 한다.

Figure 2 시나리오 흐름도 및 MAPE-K 루프

3. 다중 조명 제어 최적화 의사결정 기법

3.1 다중 광원 시나리오

자가-적응을 활용한 조명 제어 소프트웨어는 기존에 연구된 바가 있다 [7,11]. 하지만 이 연구는 단일 인공 광원이기 때문에 실제 자연 광원을 포함하고 있는 경우에는 적용이 어렵다는 단점이 있다. 따라서 본 논문의 다중 광원 시나리오는 자연 광원이 유입되는 창문의 광량을 조절하는 자동 제어 커튼을 포함한 자연 광원과 기존의 실내 조명인 인공 광원을 포함한다. 따라서 인공 광원과 자연 광원을 가지고 있는 공간이 있다고 가정 한다. 그 공간에 사용자가 들어오게 되면 사용자의 스마트폰의 애플리케이션과 연동하여 현재 스마트폰의 조도 설정 값을 시스템이 읽어오게 된다. 사용자의 선호도가 애플리케이션에 저장되어 있는 상태라면 사용자의 선호도를 바탕으로 인공 광원과 자연 광원을 조절해 선호도에 따른 조명 최적화를 하게한다. 다른 사용자가 추가로 들어올 경우 기존에 있던 사용자와 추가로 들어온 사용자간의 선호도 평균치를 찾아 조절하게 된다.

Figure 2는 자가-적응 소프트웨어 기반의 조명 제어 시스템을 다중 광원 시나리오에 접목시켰을 때 발생하는 시나리오 흐름을 MAPE 순서로 적용 시킨 결과이다. 적용이 끝나면 다시 모니터링 상태로 돌아가 시스템은 반복하게 된다.

3.2 사용자 선호도 기반 조명 제어 방법

기존의 사용자 선호도를 바탕으로 조명을 제어하는 연구[5]를 참고하면 사용자 i 는 선호도를 나타내는 효용 함수 $\Phi_i(a, x_i)$를 가지며, x_i 는 사용자의 위치, a는 사용자 위치의 현재 조도를 나타낸다. 상기 값을 사용해 효용 함수의 최대값을 계산하는 수식은 (1)로 표현할 수 있다.

$$argmax_a \sum_{i=1}^{n} \Phi_i(a, x_i) \qquad (1)$$

수식(2)는 수식 (1)에 관리자 효용 함수 Ψ를 추가해 사용자의 선호도가 없을 경우에 조명의 상태를 조절 할 수 있게 한다. 관리자 효용 함수는 사용자들의 선호도보다 낮게 설정해 놓아 사용자들이 존재하지 않을 경우에 조명 시스템의 조도 양을 낮춰주는 역할을 한다.

$$\mathrm{U(a,x)} = \sum_{i=1}^{n} \Phi_i(a, x_i) + \gamma\Psi(a) \qquad (2)$$

3.3 다중 광원 최적화 방법

인공 조명 al과 자연 광원 nl이 있다고 가정할 때 자연 광원은 인공 광원에 비해 에너지 소비가 거의 없다고 볼 수 있다. 그렇기 때문에 조도를 조절해야 할 경우

에너지 효율 측면에 따라 우선 순위를 줄 수 있게 된다.

수식 (3)에서 총 조도 값은 자연 광원의 최대 조도 nl, 인공 광원의 최대 조도 al 앞에 $0 \le c_n \le 1$, $0 \le l_n \le 1$ 인 각 자연 광원 조도 계수 c_n, 인공 광원 조도 계수 l_n 를 두어 계수 값만 실시간으로 조절해 전체 조도를 조절 할 수 있도록 한다. 자연 광원의 최대 조도는 주기적으로 외부 센서를 통해서 체크를 하게 되고 인공 광원의 최대 조도는 시스템을 실행 시키기 전 인공 광원을 설치하고 측정 한 뒤 시스템에 저장해놓고 활용한다.

$$illumination(lx) = c_n(nl) + l_n(al) \qquad (3)$$

예를 들어, 현재 조도가 사용자 선호도보다 낮아 조명을 켜야 할 경우 자연 광원 계수를 우선순위로 두어 인공 광원보다 에너지 효율이 좋은 자연 광원을 우선적으로 사용하게 한다. 반대의 경우인 현재 조도를 낮춰야 하는 경우에는 에너지 소비 효율이 낮은 인공 광원을 먼저 낮춰서 광원의 에너지 효율을 최적화 한다.

4. 결론

본 논문에서는 자가-적응 소프트웨어의 MAPE-K를 기반으로 한 조명 제어 시스템에 인공 광원과 자연 광원이 동반하는 다중 광원 시나리오를 제안하였다. 또한 다중 광원 시나리오에서 에너지 효율 최적화를 위한 적응 의사 결정 방법을 제안하였다.

본 논문에서는 기존 자가-적응 조명제어 시스템의 한계를 극복하고자 했다. 기존 단일 조명제어 기기 만을 고려하던 기존 연구와는 달리, 다중 조명 기기가 존재할 수 있는 사물인터넷 환경을 고려했다. 더불어 제안하는 기법을 응용해 추후 요구사항이 상충되는 사물인터넷 환경에서의 최적화를 위해서 응용이 가능할 것으로 판단된다.

향후 연구로는 제안된 시스템을 아두이노 등의 기기를 사용해 구현하고 시뮬레이션 할 예정이다. 더불어 다중 인공 광원 상태에서 사용자의 위치를 고려하여 더 가까운 곳에 있는 인공 광원 일수록 에너지 소비 효율이 좋다고 볼 수 있기 때문에 인공 광원의 계수를 다르게 하여 실험을 해 볼 예정이다. 마지막으로 다중 광원 상태에서 최적화 알고리즘을 예외 상황이 생길 경우를 대비하여 보완해야 한다.

5. 참고문헌

[1] Miorandi, Daniele, et al. "Internet of things: Vision, applications and research challenges." Ad Hoc Networks 10.7 (2012): 1497-1516.

[2] Salehie, Mazeiar, and Ladan Tahvildari. "Self-adaptive software: Landscape and research challenges." ACM Transactions on Autonomous and Adaptive Systems (TAAS) 4.2 (2009): 14.

[3] Brun, Yuriy, et al. "Engineering self-adaptive systems through feedback loops." *Software engineering for self-adaptive systems*. Springer Berlin Heidelberg, 2009. 48-70.

[4] Anaya, Ivan Dario Paez, et al. "A prediction-driven adaptation approach for self-adaptive sensor networks." Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. ACM, 2014.

[5] Singhvi, Vipul, et al. "Intelligent light control using sensor networks." Proceedings of the 3rd international conference on Embedded networked sensor systems. ACM, 2005.

[6] Pan, Meng-Shiuan, et al. "A WSN-based intelligent light control system considering user activities and profiles." IEEE Sensors Journal 8.10 (2008): 1710-1721.

[7] 이준희, 이의종, and 백두권. "자가-적응 조명 제어 소프트웨어의 시뮬레이션 및 성능 평가." 한국시뮬레이션학회논문지 25.2 (2016): 63-74.

[8] Wuttke, Jochen, et al. "Traffic routing for evaluating self-adaptation." *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2012.

[9] Garlan, David, et al. "Rainbow: Architecture-based self-adaptation with reusable infrastructure." *Computer* 37.10 (2004): 46-54.

[10] Cheng, Shang-Wen, David Garlan, and Bradley Schmerl. "Evaluating the effectiveness of the rainbow self-adaptive system." *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2009.

[11] 이준희, 이의종, 백두권. (2015.12). 스마트 강의실 : 자가-적응 소프트웨어를 위한 사례 연구. 한국정보과학회 학술발표논문집,490-492.

스마트 냉장고 상호작용 시 사용자 개입을 최소화하기 위한 스마트 스토리지 시스템 설계 및 프로토타입 개발

박준희 O, 손희석, 김재헌, 이동만

전산학부, 한국과학기술원

{wns3645,heesuk.son, rlawogjs618, dlee}@kaist.ac.kr

A Smart Storage System to Minimize User Interventions in a Smart Fridge Interaction

Junhui Park O , Heesuk Son, Jaeheon Kim, Dongman Lee

School of Computing, KAIST

요 약

최근 높아지는 스마트 홈에 대한 관심에 따라 스마트 냉장고에 관련된 연구가 진행되고, 여러 제품이 출시되고 있다. 스마트 냉장고로써의 조건을 만족하기 위해서는 냉장고가 얼마나 향상된 식품 보관 기능을 사용자에게 제공하는지가 중요하다. 하지만 최근에 출시되고 있는 냉장고 제품들은 식품의 보관 기능보다는 사용자에게 엔터테인먼트 미디어를 제공하는 기능에 초점을 두어, 식품의 보관 및 관리 기능을 제공함에 있어 많은 사용자 개입을 요구하여 사용성을 저해하고 있다. 본 논문에서는 스마트 냉장고의 식품 보관 기능에 초점을 맞추어, 사용자가 냉장고를 사용할 때 냉장고가 식품을 자동으로 판별하고 이에 대한 정보를 제공할 수 있는 스마트 스토리지 시스템을 제안한다.

1. 서 론

최근 스마트 홈 환경 구축에 대한 사람들의 관심이 높아지면서 스마트 가전에 대한 연구가 활발하게 진행되고 있다. 그 중에서도 일상에서 가장 많이 사용되는 스마트 냉장고와 관련해서 많은 상용 제품과 연구 결과가 발표되고 있다. 예를 들어, 삼성전자에서는 '삼성 패밀리 허브' 냉장고를 출시하여 사용자에게 다양한 엔터테인먼트 에 관한 기능을 제공하고 있고, LG전자에서는 냉장고에서 Windows 10 운영체제를 사용할 수 있는 'LG InstaView' 냉장고를 출시하였다.

스마트 냉장고에 대한 연구는 기반 시스템 및 스마트 서비스에 걸쳐 다양하게 진행되고 있다. Rouillard[1]는 냉장고에서 사용되지 못하고 버려지는 식품으로 인한 식품 낭비 문제를 해결해주기 위해 스마트폰의 바코드 스캔 기능을 이용해 식품을 판별하고 정보를 저장하는 Pervasive Fridge System을 제안한다. Luo[2]는 마찬가지로 바코드 스캔을 활용하여 식품을 판별하는 기능을 사용하여 사용자의 영양 섭취 및 건강 관리 기능에 초점을 맞춘 스마트 냉장고 시스템을 소개한다. Gu[3] 와 Xie[4]는 식품에 부착되는 RFID 태그를 이용하여 냉장고 안의 식품을 판별할 수 있는 시스템을 제안하고, Sandholm[5]는 웹 캠과 IR 센서를 이용하여 식품을 판별하고 위치를 알아내는 기능을 가진 스마트 냉장고 시스템을 제안한다.

하지만 기존의 상용 제품 및 관련 연구의 경우, 냉장고 본연의 기능인 식품 보관 기능을 제공함에 있어 사용자의 개입을 전제로 한다는 한계점을 가진다. Sandholm의 경우 이러한 제약으로부터 비교적 자유롭긴 하지만, 냉장고 선반을 여덟 구역의 격자 형태로 설계되었기 때문에 사용자가 식품을 놓을 때 구분 된 각 구역에 정확하게 맞추어서 놓아야지만 해당 식품에 대한 위치의 인지 및 추적이 가능하다는 불편함이 존재한다. 이렇듯 스마트 냉장고가 제공하는 기능을 사용할 때마다 사용자가 직접 정보를 입력하거나 물건을 놓는 위치의 정확도에 인식을 지속적으로 해야 한다면, 이는 결국 사용자 경험을 저해할 뿐 아니라 사용자로 하여금 스마트 기능의 사용을 멈추는 계기로 작용할 수 있다. 이러한 한계점을 극복하고 Mark Weiser[6]가 제안한 Seamlessness를 갖춘 스마트 식품 보관 기능을 제공하려면, 사용자 개입 없이 보관하고 있는 식품 리스트 및 각 식품에 대한 상세 정보를 자동으로 인식 및 관리하고 그를 기반으로 응용 서비스가 제공되어야 한다.

본 논문에서는 추가적인 관련 센서를 활용하여 식품 인지 및 추적에 있어 사용자 개입을 최소화하고 그를 기반으로 응용 서비스에 식품 재고 정보를 제공하는 스마트 스토리지 프레임워크를 제안한다. 제안하는 프레임워크는 식품 재고 상황을 인지하고 그를 다른

모듈과 공유할 수 있는 정보로 가공해주는 Food Context Widget, 생성 된 정보를 유기적으로 관리해주는 In-Fridge Food DB, 그리고 정보를 활용하여 사용자에게 스마트 기능을 제공하는 Application로 구성이 된다. 스마트 스토리지의 프로토타입 구현을 위해서는 Food Context Widget의 인스턴스인 식품 유형 분류 모듈 (Food Type Classifier), 식품 위치 인지 및 추적 모듈 (Location Tracker), 그리고 냉장고 문 상태 분석기 (Door Open/Close Detector)를 실제 냉장고에 설치 및 구현하였다. 그리고 그에 기반한 냉장고 재고 관리 프로그램을 웹 기반으로 개발하여 그 사용성을 실험해 보았다.

2. 관련 연구

최근 삼성에서 출시한 '삼성 패밀리 허브' 냉장고는 사용자가 전면에 부착된 큰 화면을 통해 음악 감상, 앨범 공유, TV 시청 등의 다양한 엔터테인먼트 기능을 제공한다. 하지만, 식품 보관에 대해서는 단지 냉장고 내부 사진을 찍어 사용자가 스마트폰으로 확인할 수 있게 할 뿐, 각 식품의 이름, 보관 기간 등을 사용자가 직접 입력해야 하는 번거로움이 존재한다. LG에서 출시한 'LG InstaView' 냉장고는 Windows 10을 탑재하여 냉장고에서도 사용자가 원하는 응용 프로그램을 마치 개인용 컴퓨터와 같이 자유롭게 사용할 수 있으며, 사용자가 냉장고 문을 두드리면 내부에 불이 켜지면서 투명한 창을 통해 냉장고 내부를 바로 확인할 수 있다. 하지만 자동화된 식품의 인식이나 식품의 리스트 작성 및 제공은 지원하지 않는다.

Rouillard[1]는 냉장고에서 사용되지 못하고 버려지는 식품으로 인한 식품 낭비 문제를 해결 하기 위해 Pervasive Fridge system을 제안하여 식품 보관을 효율적으로 할 수 있는 방법을 제시한다. 냉장고에 식품을 보관하고자 할 때, 스마트폰 애플리케이션을 이용하여 식품의 바코드를 찍으면 애플리케이션은 이를 자동으로 인식하여 식품의 리스트를 만들고 정보를 제공해주는 기능을 제공한다. 하지만, 식품을 냉장고에 넣을 때마다 매번 스마트폰으로 바코드를 찍는 것은 사용자에게 굉장히 번거로울 수 있으며, 바코드만으로 식품의 상세 정보를 제공하는 것에는 공개 된 데이터의 한계가 있다. Luo[2]는 스마트 냉장고가 가져야할 여러 기능 중, 사용자의 영양 섭취 및 건강 관리 기능에 초점을 맞춘 스마트 냉장고를 제안한다. 냉장고 안에 보관 중인 식품의 리스트를 제공하며, 이 리스트를 기반으로 만들 수 있는 요리의 레시피를 추천해주는 기능을 포함한다. 위 연구와 마찬가지로 사용자는 바코드 스캐너를 이용하여 식품을 판별하고 식품의 리스트를 관리하기 때문에 사용자가 편하게 사용하기에는 부족한 점이 있다.

Gu[3]과 Xie[4]는 식품에 부착되는 RFID 태그를 이용하여 식품을 판별하고 식품의 위치를 확인할 수 있는 스마트 냉장고를 제안한다. 하지만 매번 RFID 태그를 찍는 행동이 번거로우며, RFID 태그가 모든 식품에 부착되어 있어야 한다는 비현실적인 가정으로 인하여 기능의 실현 가능성이 부족하다. Sandholm[5]의 Cloud Fridge에서는 식품의 판별을 위해 Google Search By Image를 사용하고, 식품의 위치를 IR proximity sensor를 이용하여 인식한다. 또한, 수집되는 식품의 정보를 바탕으로 식품의 영양 정보를 제공하고, 사용자가 냉장고에서 식품을 찾을 때 도움을 주는 애플리케이션을 제공한다. 하지만 Google Search By Image는 많이 검색될 수 있는 식품에 한해서만 알맞은 결과를 보여주는 문제점이 있으며, IR proximity sensor를 이용할 경우 식품의 위치 인식을 위해 정해진 공간에 정확히 식품을 두어야 한다는 한계가 있다.

3. 요구사항 분석

스마트 냉장고 사용 시 사용자의 개입을 최소화 하면서 식품 보관 기능을 향상시키기 위해서 가장 중요한 것은 냉장고가 자동으로 식품을 판별하고 식품에 대한 정보를 수집하여 사용자에게 제공하는 것이다. 이를 만족시키기 위한 첫번째 요구사항으로, 사용자는 단순히 냉장고에 식품을 넣는 동작 이외에는 특별한 동작을 취하지 않아도 냉장고가 식품을 자동으로 판별할 수 있어야한다. 식품 판별의 자동화를 통해서 현재 냉장고에 보관중인 식품의 리스트를 작성하여 사용자에게 제공할 수 있다. 사용자는 냉장고 문을 열지 않더라도 현재 보관중인 식품의 리스트를 보고 냉장고에 어떤 식품이 있는지 쉽게 확인할 수 있어야 한다.

두번째로는 냉장고 내부의 식품의 위치를 판별하여 알려주는 기능을 갖추어야 한다. 단순히 식품의 리스트만 제공하는 것이 아니라 식품의 위치를 함께 제공할 때, 사용자는 냉장고에서 원하는 식품을 찾는데 들이는 시간을 절약할 수 있다. 마지막으로, 이러한 식품 보관 기능을 통해 얻어지는 정보들을 사용자가 쉽게 이용할 수 있는 웹 서비스나 기타 애플리케이션이 쉽게 제공받을 수 있어야 한다. 사용자는 이 애플리케이션의 사용을 통해서 언제, 어디에서나 냉장고 내부의 식품을 쉽고 빠르게 확인하며 냉장고를 사용할 수 있다. 뿐만 아니라 이를 기반으로 레시피 추천이나 쇼핑 리스트 자동 생성 등의 응용 서비스들을 편리하게 제공받을 수 있다.

4. 시스템 설계

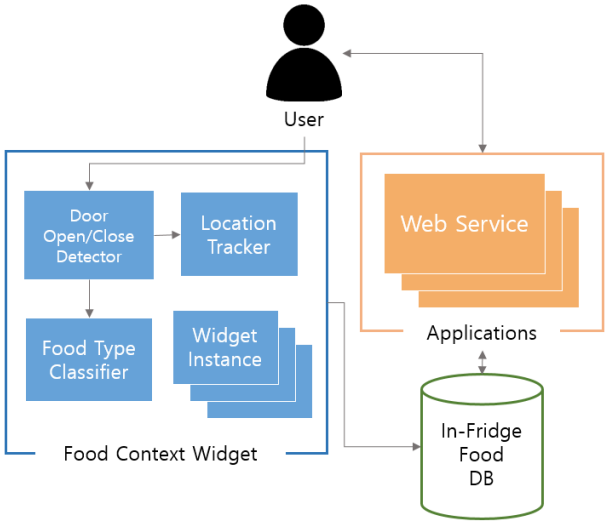


그림 1. 스마트 냉장고 시스템 설계 개요

그림 1은 본 논문에서 제안하는 스마트 냉장고 시스템의 설계 개요로, 크게 세가지 컴포넌트로 구성된다. 첫 번째 컴포넌트인 Food Context Widget 는 사용자 상호작용의 대상이 되는 식품과 관련된 컨텍스트 정보를 인지 및 생성해주는 모듈이다. 사용자가 냉장고 문을 열고 식품을 넣고자 할 때, Food Context Widget의 인스턴스인 Food Type Classifier는 식품이 어떤 타입인지 (예: 계란, 우유, 주스 등) 식별한다. 식별된 식품에는 고유한 ID가 할당되어 두 번째 인스턴스인 Location Tracker로 해당 정보가 전송이 되는데, Location Tracker는 식품의 초기 위치 및 향후 위치의 변화를 자동으로 계산 및 추적해준다. 이렇게 얻어진 식품 관련 컨텍스트 정보는 두 번째 컴포넌트인 In-Fridge Food DB에 저장되어 다른 모듈에 의해 활용될 수 있는데, 사용자에게 스마트 서비스를 제공하는 세 번째 컴포넌트인 애플리케이션 (예: 웹 서비스 등) 이 사용자에게 해당 정보를 쿼리 및 가공할 수 있다.

4.1 Food Context Widget

스마트 스토리지 기능을 위해 요구되는 상황/식품 정보로는 냉장고 문의 개폐여부, 식품 유형, 그리고 식품의 위치를 들 수 있다. 본 연구에서는 이러한 대표적인 세 가지 상황 정보의 인지를 위해 Food Context Widget의 인스턴스인 Door Open/Close Detector, Food Type Classifier, 그리고 Location Tracker를 설계한다.

Door Open/Close Detector는 냉장고 문이 열렸는지 닫혔는지에 대한 정보를 제공하며, 사용자가 식품을 냉장고에 넣고 냉장고의 문을 닫으면 Food Type Classifier와 Location Tracker 인스턴스로 하여금 식품의 유형 및 위치를 인지하도록 신호를 보낸다.

Food Type Classifier는 저장 된 식품이 무엇인지 파악하기 위한 모듈로, 식품 인지 정확도를 높이기 위해 가장 높은 정확도를 보이는 이미지 기반 객체 식별 기술을 활용한다. 냉장고 내부에 설치된 Web Cam을 통해 냉장고 내부로 들어온 식품의 이미지가 Server로 전송되는데, 전송된 이미지로부터 추출되는 특징 정보를 (예: 식품의 Label 및 Logo) 활용하여 해당 식품이 무엇인지를 인지한다. 인지 된 식품 정보는 In-Fridge Food DB에 저장되어 다른 응용 서비스들에 의해 소비된다.

Location Tracker는 냉장고에 저장 된 식품의 내부 위치를 인지 및 추적하기 위한 모듈이다. Sandholm의 연구에서는 사용자가 지정 된 위치에만 식품을 놓아야 한다는 제약조건이 있는데, 본 연구에서는 이러한 한계를 극복하기 위해 8개의 압력 센서로 구성 된 Weight Detection Board를 설계하여 냉장고 선반의 어떤 위치에 놓여진 식품이든 감지할 수 있도록 한다. 그림 2는 압력 감지 센서의 측정값을 이용해 식품의 위치를 판별하기 위해 설계 된 알고리즘으로, 사용자가 냉장고에 식품을 넣으면 각 구역에 설치된 압력 감지 센서의 측정값을 비교하여 어느 위치에 식품이 놓여졌는지 또는 옮겨졌는지 판별한다. 식품 위치의 이동을 판별하기 위해서는 압력 값이 가장 많이 감소한 센서와 가장 많이 증가한 센서를 찾아, 전자의 위치를 식품의 이전 위치, 후자의 위치를 식품의 새로운 위치로 판별한다. 이렇게 식품의 위치 판별을 위해 압력 감지 센서를 사용하면 식품을 놓을 수 있는 위치가 훨씬 자유로워지므로 사용자의 불편을 초래하지 않으면서 동시에 냉장고의 공간을 더 효율적으로 활용할 수 있다.

Algorithm 1 Location Tracking

$deltaValue \leftarrow currValue - prevValue$ for each position
$fromSection \leftarrow 0$
$toSection \leftarrow 0$
for i = 0 to 8 **do**
  if $deltaValue[fromSection] \geq deltaValue[i]$ **then**
    $fromSection \leftarrow i$
  end if
  if $deltaValue[toSection] \leq deltaValue[i]$ **then**
    $toSection \leftarrow i$
  end if
end for
if $deltaValue[fromSection] \geq 0$ **then**
  $fromSection \leftarrow -1$
end if
if $deltaValue[toSection] \leq 0$ **then**
  $toSection \leftarrow -1$
end if
return $fromSection, toSection$

그림 2. Location Tracking을 위한 알고리즘

인지 된 식품의 위치 정보는 In-Fridge Food DB에 저장되어 활용된다.

4.2 In-Fridge Food DB

In-Fridge Food DB에는 Food Context Widget을 통해서 얻어지는 식품에 대한 모든 정보들을 저장한다. 고유의 ID를 부여받은 각 식품에 대해서 Food Type Classifier로부터 얻어지는 Label과 Logo 분석의 결과와 Location Tracker로부터 얻어지는 위치 정보를 저장한다. 이 때, 식품을 보관할 때의 시간 정보를 함께 저장하는데, 이 정보는 추후에 보관 기간을 활용한 다양한 응용 서비스 (예: 쇼핑 리스트 자동 완성 및 식품 부패 시간 계산기 등) 에 의해 사용될 수 있다. 이 In-Fridge Food DB는 Food Context Widget이 접근하여 내용을 수정할 수 있으며, 사용자 또한 웹 서비스를 통해 저장 되어있는 식품에 대한 정보를 언제 어디에서나 확인할 수 있다.

4.3 Application

사용자가 현재 냉장고 안에 보관되고 있는 식품에 대한 정보를 확인하기를 원할 때, 사용자는 어디서든 웹 서비스에 접속하기만 하면 현재 냉장고 속에 어떤 식품이 어느 위치에 있는지를 쉽게 확인할 수 있다. 사용자가 웹 서비스에 접속하면 가장 최근 상태의 냉장고 내부 사진 또한 확인할 수 있다. 이러한 응용 프로그램은 스마트 폰 애플리케이션의 형태로도 확장될 수 있으며, 단순한 냉장고 재고 확인을 넘어 재고에 기반한 스마트 레시피 추천 서비스 등으로 응용될 수 있다.

5. 시스템 구현

본 논문에서 제안하는 스마트 스토리지 시스템의 사용성을 확인해보기 위해 스마트 스토리지 프로토타입 및 그에 기반 한 재고 관리 웹 서비스를 개발하였다. 그림 3은 프로토타입을 구성하는 물리적인 컴포넌트들의 동작 과정을 보여준다. Food Context

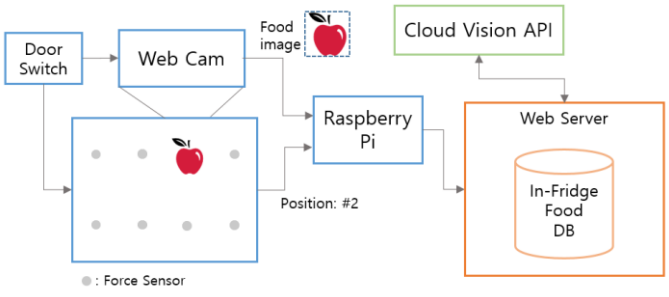


그림 3. 각 하드웨어 컴포넌트의 동작 과정

그림 4. 냉장고에 설치된 스위치

그림 5. 냉장고에 설치된 카메라

그림 6. Web Cam이 냉장고 내부를 촬영한 사진

Widget을 통한 식품 정보 생성 및 처리를 위한 상용 미니 PC로써 Raspbian Jessie 운영체제 기반의 Raspberry Pi 2(Model B)를 사용하였다. Door Open/Close Detector의 구현을 위해서는 그림 4와같이 냉장고의 문이 닫히는 위치에 스위치를 부착하여 사용자가 냉장고를 열고 닫는 동작을 식별할 수 있도록 하였다.

Food Type Classifier의 구현을 위해서는 그림 5와 같이 Raspberry Pi에 Odroid HD Web Camera를 연결하였다. 냉장고 문이 닫히면 Odroid HD Web Camera는 그림 6과 같은 냉장고 내부 이미지를 촬영하여 Web Server로 전송하는데, Web Server는 전송 받은 사진을 Google Cloud Vision API를 이용하여 Labeling한다. Location Tracker의 구현을 위해서는 그림 6과 같이 냉장고 선반 크기의 아크릴판 위에 8개의 Phidgets Flexi Force Sensor(1lb)를 부착하고, 그 위를 아크릴판으로 덮어서 압력 센서 보드를

```
[
  {
    "_id": "5875d33d3d60b8459a5fc809",
    "file_name": "00000001.png",
    "__v": 0,
    "user": "Junhui",
    "date": "2017-01-11T06:39:57.427Z",
    "position": 5,
    "logo": [
      "Coca-Cola"
    ],
    "label": [
      "soft drink",
      "drink",
      "carbonated soft drinks",
      "freezing"
    ]
  }
]
```


그림 7. 콜라 캔을 넣었을 때 얻어진 JSON 결과값

제작하였다. 이를 통해 얻어진 위치 정보는 식품 Labeling 정보와 함께 JSON 형태로 In-Fridge Food DB에 저장된다. 그림 7은 콜라캔을 냉장고에 보관했을 때 얻어진 결과값을 보여주는 JSON 파일 예제이다.

그림 8은 구현 된 스마트 스토리지 프로토타입과의 데이터 교환을 통해 냉장고 안에 저장되어 있는 식품 정보를 확인할 수 있는 웹 서비스의 메인 페이지이다. 그림과 같이 사용자는 메인 페이지에서 가장 최근의 냉장고 내부 사진을 바로 확인할 수 있을 뿐 아니라 왼쪽의 메뉴를 통해서 카테고리별로 분류된 식품의 리스트를 확인할 수 있다. Drink, Fruit, Vegetable 과 같이 식품의 Label Detection로 얻어진 결과를 이용해 식품의 카테고리를 나누어 사용자가 쉽게 원하는 식품을 확인할 수 있도록 하였다. 예를 들어, 사용자는 Drink 탭에 접근하여 현재 냉장고에 보관중인 식품 중에서 Drink Label을 가지는 모든 식품에 대한 정보를 확인할 수 있다.

사용자가 직접 데이터를 확인할 수 있도록 하기 위한 Web Service의 구현에는 Node.js HTTP Server를 사용하고, In-Fridge Food DB의 관리를 위하여 MongoDB를 사용한다. DB 접근을 위해서는 HTTP Request 기반의 RESTful API를 구현하였다. 따라서 사용자가 데이터를 조회 및 수정하거나 Raspberry Pi에서 데이터를 생성 및 수정하여 DB에 저장하고자 할 때 RESTful API를 이용하여 In-Fridge Food DB에

그림 8. 냉장고에 저장 된 식품 정보를 확인하기 위한 웹 서비스

접근할 수 있다.

6. 결론

본 논문에서는 사용자의 개입 없이 냉장고가 자동으로 식품을 판별하여 웹 서비스를 통해 사용자에게 식품에 대한 정보를 제공할 수 있는 스마트 스토리지 시스템을 제안하였다. 사용자가 수동으로 정보를 입력 하는 등의 특별한 행동을 하지 않아도 Food Context Widget의 인스턴스인 Door Open/Close Detector, Food Type Classifier, 그리고 Location Tracker가 자동으로 식품의 유형 및 위치를 인식하여 데이터베이스에 저장해준다. 스마트 스토리지 프로토타입을 기반으로 구현 한 샘플 웹서비스를 통해서는 사용자가 냉장고 안에 보관중인 식품의 현재 상태를 직접 사진으로 볼 수 있으며, 각 식품에 대한 정보를 카테고리별로 모아서 체크할 수 있다.

본 연구의 연장선으로는, 제안하는 스마트 스토리지의 사용성 및 활용성을 검증하기 위한 보다 다양한 스마트 서비스를 설계 및 개발해보고자 한다. 사용자가 직접 웹 서비스에 접속하지 않아도 사용자에게 자동으로 유통기한 알림을 주거나 재고 확인을 통해 쇼핑 리스트를 자동으로 작성 및 제공해주는 애플리케이션이 그 예가 될 수 있다. 이 외에도 스마트 냉장고를 통해서 얻을 수 있는 보다 다양한 상황정보를 확보하는 것 또한 중요하다. 본 연구에서 인지한 정보의 경우 식품의 종류 및 위치에 불과하지만 앞으로 식품에 대한 영양 정보, 유통 기한, 보관 기간 등을 함께 제공하여 사용자들로 하여금 스마트 냉장고 서비스를 더 유용하게 사용할 수 있도록 하고자 한다.

6. 사사문구

본 연구는 미래창조과학부 및 정보통신기술연구진흥센터의 방송통신·산업기술 개발사업의 일환으로 수행하였음. [B0101-16-0334 , IoT 기반 스마트 홈 커뮤니티에서 안전하고 행복한 삶을 위한 소셜 매칭 및 소통 서비스 기술 개발]

참고문헌

[1] Rouillard, J., The Pervasive Fridge. A Smart computer System Against Uneaten Fool Loss, Seventh International Conference on Systems (ICONS2012), Feb 2012, 135-140, 2012

[2] Luo, S., Xi, H., Gao, Y., Jin, S., and Athauda, R., Smart Fridges with and ability to enhance health and enable better nutrition, Int. J. Multimedia Ubiquitous Eng, 4.2, 66-80, 2009

[3] Gu, H, and Wang, D., A content-aware fridge based on RFID in smart home for home-healthcare, Advanced Communication Technology, ICACT 2—0, 11th international Conference, Vol 2, 987-990, 2009
[4] Xie, L., Sheng, B., Yin, Y., Lu, S., Lu, X., iFridge: An Intelligent Fridge for Food Management based on RFID Technology, Ubicomp'13, September 8-12, 291-294, 2013
[5] Sandholm, T., Lee, D., Tegelund, B., Han, S., Shin, B., and Kim, B., Cloud Fridge: A Testbed for Smart Fridge Interactions, arXive preprint arXive:1401.0585, 2014
[6] Weiser, M. The computer for the 21st century., Scientific American 265.3, 94-104, 1991

시험 자동화도구를 적용한 소프트웨어 통합시험 커버리지 달성방안

장정훈 O 이기영 이원택 강유선 류인수

㈜모아소프트

jhjang@moasoftware.co.kr, gylee@moasoftware.co.kr, acct3832@moasoftware.co.kr, yskang@moasoftware.co.kr, isryu@moasoftware.co.kr

A Method to achieve Test Coverage for the Software Integration Test using LDRA Testing Tool

Jeong-Hoon Jang O, Ki-Young Lee, Won-Taek Lee, Yu-Sun Kang, In-Soo Ryu

MOASOFT

요 약

본 연구에서는 항공기, 자동차 소프트웨어 개발에 자동화 시험 도구를 활용하여 항공 및 자동차 산업 규격에서 요구하는 소프트웨어 통합시험 커버리지 달성 사례 분석을 통해 해당 규격요건을 충족하기 위한 소프트웨어 통합 시험 절차 및 방안을 제시하였으며, 소프트웨어 구현, 시험 및 유지보수 단계에서 시험 자동화 도구를 활용하여 소프트웨어 개발 비용 및 시간을 절약할 수 있다.

1. 서론

최근 항공기, 자동차, 철도, 선박, 원자력 등 산업전반에서 소프트웨어의 안전성이 강화됨에 따라 소프트웨어 품질을 향상시키기 위한 소프트웨어 시험 및 검증이 주요 이슈로 떠오르고 있다. 소프트웨어 시험은 구현된 소프트웨어의 결함을 조기에 발견함으로써 품질 향상 및 개발비용 절감의 효과가 있어, 실제 소프트웨어 개발 비용의 절반 이상이 소프트웨어 시험단계에 할애되고 있다.

DO-178C(항공기 SW), ISO 26262-6(자동차 SW) 등 모든 국제 산업규격에서는 소프트웨어 단위 시험(Unit Test), 소프트웨어 통합 시험(Software Integration Test), 하드웨어/소프트웨어 통합시험(HW/SW Integration Test) 등의 소프트웨어 시험 수행을 권고하고 있다.

DO-178C는 소프트웨어 안전무결성(SIL, Software Integrity Level)을 충족하기 위한 입증자료로 소프트웨어 통합 시험단계의 제어 결합도(Control Coupling)에 대한 시험 커버리지(Test Coverage)를 달성하도록 요구하고 있으며, ISO 26262-6에서도 자동차 안전무결성(ASIL, Automotive SIL)을 충족하기 위해 통합 시험단계의 시험 커버리지로 Function Coverage와 Call Coverage를 달성하도록 요구하고 있다.

본 연구에서는 항공기 및 자동차 소프트웨어 개발에서의 소프트웨어 통합 시험사례를 통해 소프트웨어 안전규격이 요구하는 시험 커버리지 목표 달성을 위해 자동화 시험 도구를 활용한 절차 및 방안을 제시하며, 아울러 소프트웨어 시험 커버리지 미달성 요인에 대한 해결방안도 제안한다.

2. 소프트웨어 시험 프로세스

2.1. DO-178C 시험절차

DO-178C는 구현된 소프트웨어에 대한 전반적인 시험 프로세스 및 활동을 그림 1과 같이 규정하고 있으며 그에 따른 시험절차는 아래와 같다.

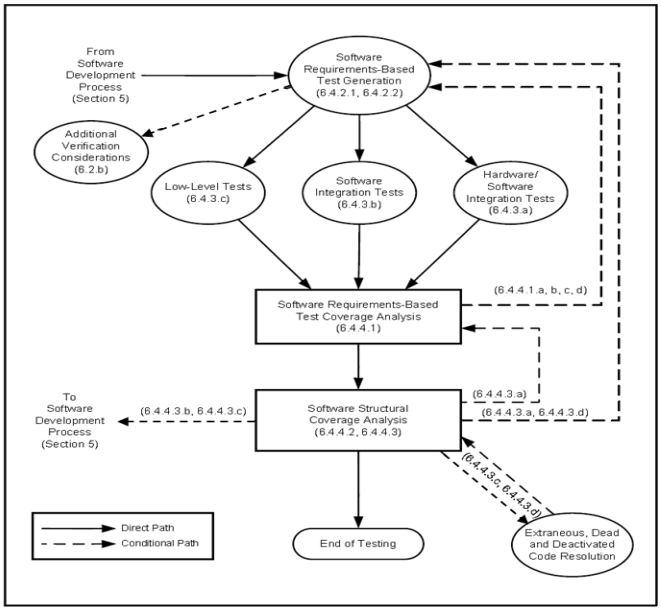


그림 1 DO-178C 소프트웨어 시험 프로세스

Step 1. 소프트웨어 상위 요구사항(high-level requirements) 및 소프트웨어 하위 요구사항(low-level requirements)를 바탕으로 소프트웨어 요구사항 기반의 테스트 케이스를 생성한다.

Step 2. 소프트웨어 단위시험, 소프트웨어 통합시험, 하드웨어/소프트웨어 통합시험을 위해 Step 1에서 생성한 테스트 케이스를 실제 소스코드 실행환경(실제 타겟) 위에서 수행한다.

Step 3. Step 2의 결과가 목표값(100%)에 도달하지 못했다면 상위/하위 요구사항과 테스트 케이스를 검토하여 목표값에 도달할 때까지 Step 1~2를 재 수행한다.

Step 4. Step 3이 완료된 코드를 바탕으로 시험 커버리지 분석을 수행하며 목표값에 도달할 때까지 Step 1~3을 반복한다.

2.2. ISO 26262-6 시험절차

ISO 26262-6의 10장 소프트웨어 통합 및 시험에서는

10.4.1. 소프트웨어 시험 계획 수립

10.4.2. 소프트웨어 통합시험 계획 및 수행

10.4.3. 소프트웨어 통합시험 방법 선정

10.4.4. 테스트 케이스 도출기법

10.4.5. 테스트 완료 평가

10.4.6. 테스트 완료 평가를 위한 측정 메트릭 제시

10.4.7. 내장형 소프트웨어 시험 기준

10.4.8. 시험 환경

을 요건으로 제시하고 있다. 그 중에서 10.4.6.의 테스트 완료 평가를 위한 측정 메트릭으로 그림 2와 같이 Function Coverage와 Call Coverage를 요구하고 있다.

Methods		ASIL			
		A	B	C	D
1a	Function coverage[a]	+	+	++	++
1b	Call coverage[b]	+	+	++	++

[a] Method 1a refers to the percentage of executed software functions. This evidence can be achieved by an appropriate software integration strategy.

[b] Method 1b refers to the percentage of executed software function calls.

그림 2 ISO 26262-6 Table 15 Structural coverage metrics at the software architectural level

2.3. 소프트웨어 통합 시험 수행 절차

소프트웨어 통합 시험은 단위 시험을 통과한 소프트웨어 모듈에 대해 소프트웨어 모듈 간의 인터페이스 및 상호 연동하는 동작을 시험하기 위한 활동으로 모듈을 결합하여 시험을 진행한다.

모듈 결합 순서에 따라 그림 3과 같이 하향식(top-down) 시험과 상향식(bottom-up) 시험으로 나눌 수 있다. 하향식 시험은 상위 모듈에서 하위 모듈까지 순서대로 통합하여 시험하는 방식으로 하위 모듈이 완성되어 있지 않은 경우에도 사용될 수 있다. 상향식 시험은 하위 모듈에서부터 상위 모듈로 순서대로 통합하여 시험하는 방식이다. 소프트웨어 개발 프로젝트 특성을 고려하여 시험방식을 선택적으로 적용한다.

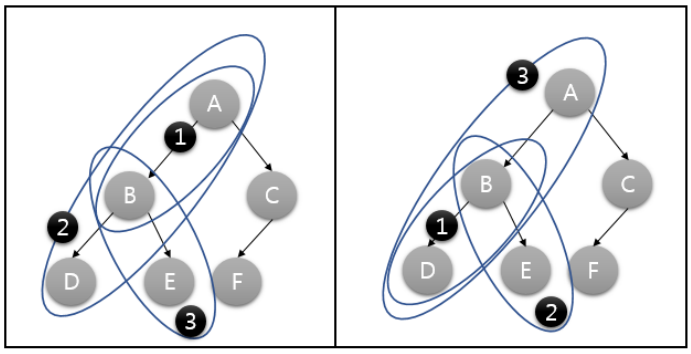


그림 3 하향식 시험(왼쪽)과 상향식 시험(오른쪽)

시험은 개발자가 아닌 별도의 시험담당자가 소프트웨어 통합시험 계획서 및 절차서에 기술된 테스트 케이스로 수행한다.

소프트웨어 통합 시험은 모듈간의 호출관계를 통해 확인할 수 있지만 소스 코드와 문서만으로는 확인하기가 어렵다. 따라서 통합시험을 지원해주는 자동화 도구를 이용하여 정적분석을 수행하여 그림 4와 같이 호출관계에 대해 그래프(Call Graph)를 추출한 후 테스트 케이스를 수행하여 올바른 함수호출관계를 그래프로 확인하거나 Function/Call Coverage Report에서 시험 커버리지(Test Coverage)를 확인한다.

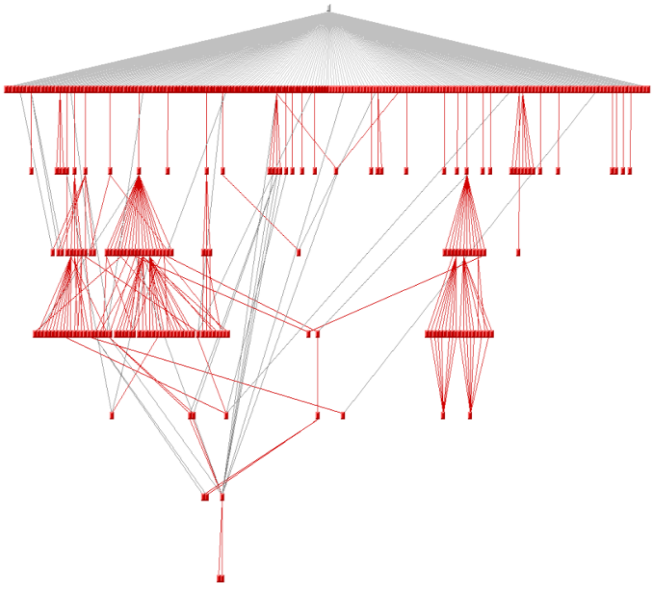


그림 4 자동화도구로 추출한 함수 호출 그래프

3. 소프트웨어 통합시험 사례 분석

본 연구에서는 항공기 및 자동차 소프트웨어 개발 프로젝트에서 자동화 시험 도구를 활용하여 시험단계에서 적용되었다.

적용한 자동화 시험 도구인 LDRA는 DO-178C, ISO 26262 등 여러 산업 분야 시험도구로서의 자격 인증(Tool Certification)을 받았으며, 소스코드에 대한 정적 분석, 단위시험, 하드웨어/소프트웨어 통합시험을 지원한다. 특히, DO-178C에서 SW Level A(안전성 최고 등급)에 대하여 요구하고 있는 2 가지 Objective인 1) Modified Condition/Decision Coverage 달성, 2) Data Coupling/Control Coupling 달성을 지원한다.

다음 절에 기술되어 있는 통합시험 수행절차와 자동화 시험 도구를 활용하여 그림 5와 같이 항공기 소프트웨어 규격에서 요구하는 제어 결합도(Control Coupling)에 대한 시험 커버리지를 산출하였으며, 자동차 소프트웨어 규격에서 요구하는 Function Coverage와 Call Coverage(그림 2)에 대한 리포트를 도구로 생성하여 규격에서 요구하는 항목에 대항 충족여부를 확인할 수 있었다.

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.d	6.4.4.2.c 6.4.4.2.d 6.4.4.3	●	●	O		Software Verification Results	11.14	②	②	②	

그림 5 DO-178C A-7 Verification of Verification Process Result

,

3.1. 통합시험 적용 수행절차

소프트웨어 소스코드가 구현된 후 통합시험을 진행하였기 때문에 상향식으로 진행하였으며, 아래와 같은 절차로 시험을 수행하였다.

1. 자동화 시험도구(LDRA)와 소프트웨어 개발 환경을 연동한다.
2. 자동화 시험도구를 이용하여 정적분석(static analysis)을 수행하여 함수 호출 그래프(Function Call Graph)를 추출한 후 함수간 시험대상을 식별한다.
3. 테스트 케이스에 나와 있는 입력값을 자동화 시험도구에 입력하여 실행하여 예상되는 함수가 호출되었는지 함수 호출 그래프를 통해 확인한다. 함수가 호출되었다면 그림 6과 같이 파랑색에서 빨간색으로 변경된다. 시험 결과는 함수 호출 그래프와 함수 호출 커버리지 리포트(그림 7)를 통해서 확인한다.
4. 함수 호출 그래프가 모두 빨강색으로 변경되었거나 함수 호출 커버리지 리포트에서 'Call/Return Coverage' 항목이 100%을 달성할 때까지 시험을 반복하여 수행한다.

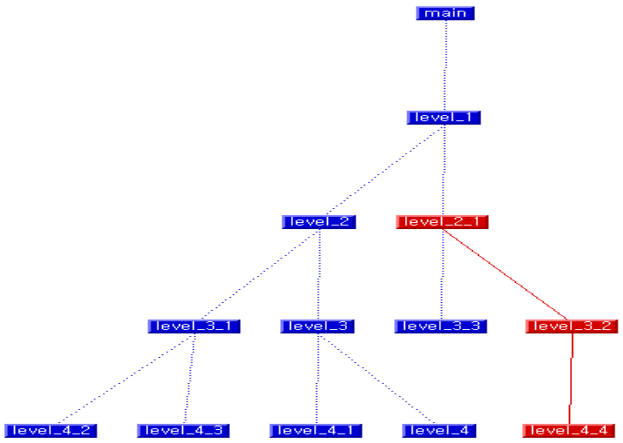


그림 6 함수 호출 후 그래프 변화

Procedure/Function Call/Return Coverage (Pass)

FROM LINE REF. (SOURCE)	PROCEDURE	TO LINE REF. (SOURCE)	PROCEDURE	PREVIOUS RUNS	CURRENT RUN	COMBINED
24 (7)	level_2	52 (19)	level_3	37	1	38
30 (9)	level_2	71 (25)	level_3_1	12	1	13
32 (10)	level_2	143 (45)	level_1	32	1	33
43 (14)	level_2_1	89 (31)	level_3_2	20	1	21
48 (16)	level_2_1	102 (35)	level_3_3	28	2	30
50 (17)	level_2_1	149 (48)	level_1	36	2	38
62 (21)	level_3	107 (37)	level_4	39	1	40
67 (23)	level_3	112 (38)	level_4_1	7	1	8
69 (24)	level_3	25 (7)	level_2	37	1	38
80 (27)	level_3_1	117 (39)	level_4_2	12	1	13
85 (29)	level_3_1	122 (40)	level_4_3	7	1	8
87 (30)	level_3_1	31 (9)	level_2	12	1	13
98 (33)	level_3_2	127 (41)	level_4_4	15	1	16
100 (34)	level_3_2	44 (14)	level_2_1	20	1	21
105 (35)	level_3_3	49 (16)	level_2_1	28	2	30
110 (37)	level_4	63 (21)	level_3	39	1	40
115 (38)	level_4_1	68 (23)	level_3	7	1	8
120 (39)	level_4_2	81 (27)	level_3_1	12	1	13
125 (40)	level_4_3	86 (29)	level_3_1	7	1	8
130 (41)	level_4_4	99 (33)	level_3_2	15	1	16
142 (45)	level_1	14 (5)	level_2	32	1	33
148 (47)	level_1	34 (12)	level_2_1	36	2	38
150 (49)	level_1	156 (52)	main	53	1	54
155 (52)	main	132 (43)	level_1	53	1	54
157 (54)	main	-1		14	1	15

Summary	Prev. Runs	Current run	Combined
Number of Calls/Returns	25		
Number Executed	25	25	25
Number not Executed	0	0	0
Call/Return coverage (%)	100	100	100

그림 7 Function Call Coverage Report

3.2. 통합 시험 커버리지 목표값 달성방안

통합시험 완료 후 시험 커버리지 목표값(100%)을 달성할 수 없다면 그 이유를 분석하여야 하며, 잘못된 요구사항, 불필요한 코드(dead code), 실행할 수 없는 코드(deactivated code), 의도하지 않은 기능의 구현 등의

원인이 있을 수 있다.

본 연구에서 수행했던 소프트웨어 개발 프로젝트 사례에서는 테스트 케이스의 부족과 잘못된 소스코드 구현으로 인하여 테스트 커버리지 목표값을 달성하지 못하였다. 따라서 시험단계의 문제발생을 방지하기 위하여 다음과 같은 검토 활동들이 수행되어야 한다.

3.2.1. 설계 및 구현 검토

그림 8과 같이 Call Depth가 10 이상인 함수 호출 구조인 경우, 또는 그림 9와 같이 순환호출(recursive call)이 있는 경우에는 소스코드 분석 및 시험 수행에 어려움을 겪는다. 따라서 구현단계에서 자동화 시험 도구의 정적분석 기능을 사용으로 함수 호출 그래프를 추출하여 소스 코드의 구조를 검토하며 개선한다면 복잡한 구조를 사전에 예방할 수 있어 소프트웨어의 유지보수성 뿐만 아니라 개발 비용 및 시간을 줄일 수 있다.

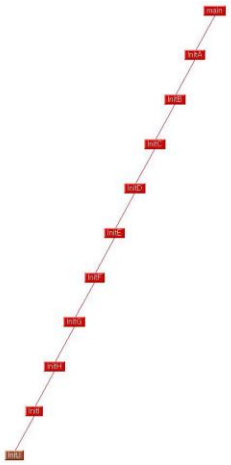


그림 8 Call Depth가 큰 소프트웨어 구조

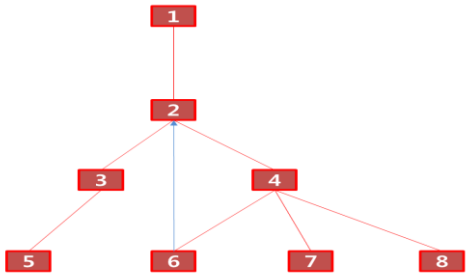


그림 9 Recursion Call이 있는 구조 (2→4→6→2)

소스코드 정적분석을 통해 표 1과 같은 소프트웨어 구조와 관련된 코딩규칙을 검사하는 것도 중요하다. 또한 소스코드의 전체 품질 수준을 나타내주는 표 2와 같은 품질메트릭에 대한 산출과 분석도 시험하기 전에 설계와 소스코드 구현 결과를 검토하는 가능한 방법이다.

LDRA 코딩규칙 ID	코딩규칙 설명	비고
3 D	전역변수를 파라미터로 사용함	Data Coupling
5 D 69 D	초기값이 부여되지 않은 변수 사용	
7 D 70 D	변수에 값이 정의되었으나 사용되지 않음	
8 D	변수에 값이 두 번 정의되는 동안 변수가 사용되지 않음	
57 D	전역변수 선언 시 초기화하지 않음	
6 D 1 U	순환 호출 검사	Control Coupling
7 C	프로시저에서 1개 이상의 Exit Point 존재	
1 S 7 X 25 X 26 X 29 X 31 X 33 X	프로시저 이름 재사용 또는 변수와 중복	
34 D	다른 파일에서 프로시저 이름 중복	
76 D 99 S	호출되지 않은 프로시저	

표 1 코딩 규칙(예시)

구분	품질메트릭 설명	기준값 (목표값)
Cyclomatic Complexity	조건 분기의 수	20 이하
Number of Call Levels	조건 분기의 깊이 정도	6 이하
Number of Calling Functions	함수호출 횟수	8 이하
Number of Logical Operator	함수 내 논리연산자 개수	20 이하
Number of Called Functions	함수 피호출 횟수	10 이하
Number of Executable Code Lines	함수 내 코드 라인 수	200 이하
Estimated Static Program Paths	조건문으로 발생하는 실행경로 수	200 이하

표 2 소스코드 품질메트릭(예시)

또한 함수 호출 그래프(Call Graph)로 시험대상 및 범위를 한눈에 파악할 수 있어 테스트 명세서 작성에 도움이 되며 테스트 명세서에 해당 그래프 이미지를 삽입하여 시험 담당자의 시험업무에 도움을 줄 수 있다.

3.2.2. 테스트 케이스 추가 및 보완

테스트 명세서의 데이터들을 이용하여 올바른 테스트 케이스를 입력하였을 때 해당경로가 실행되지 않는다면 구현 오류뿐만 아니라 잘못된 테스트 케이스인지 검토 및 분석을 해야 한다. 테스트 케이스를 하나씩 실행하여 함수호출 그래프를 확인하면서 요구사항에 맞지 않는 잘못된 테스트 케이스이거나 테스트 케이스가 부족한 경우를 검출한다. 그 후 그래프를 활용하여 해당경로를 수행하기 위한 올바른 테스트 케이스를 도출할 수 있다.

4. 결론

항공기, 자동차 프로젝트에 시험자동화 도구를 활용하여 산업규격에서 요구하는 소프트웨어 통합시험 목표값을 달성한 사례를 바탕으로 해당 규격을 만족하기 위한 소프트웨어 통합시험 절차 및 방안을 제시하였다.

DO-178C에서 요구하고 있는 데이터 커플링(Data Coupling) 관련 소프트웨어 구조적 커버리지 달성을 위해서는 함수 호출간에 전달되는 매개변수(parameter) 및 전역변수(global variable)에 대한 흐름 및 제어관계를 소프트웨어 통합시험을 통해 검증하고 그 결과를 분석하여야 한다. LDRA 도구는 이러한 데이터 커플링에 대여 소프트웨어 설계 및 구현 시 데이터 흐름도(Data Flow Diagram) 또는 데이터 흐름 분석(Data Flow Analysis)를 지원하는 자동화 시험도구이다.

LDRA 자동화 시험 도구는 소프트웨어 시험단계에서만 활용하는 것이 아니라, 정적분석 기능을 이용하여 구현단계의 소스코드에 대한 코딩규칙 검사, 소스코드에 대한 품질 메트릭(예: 복잡도, 테스트가능성 등) 산출, 리팩토링 등에도 활용하고 하드웨어 통합시험 및 유지보수 단계에서도 활용함으로써 소프트웨어 개발 비용 및 시간을 절약할 수 있다.

5. 참고문헌

[1] RTCA Inc, DO-178C (Software Considerations in Airborne Systems and Equipment Certification), 2011.

[2] ISO, ISO 26262-6(Road vehicles - Functional safety - Product development at the software level), 2011.

[3] LDRA Inc, M. A. Hennell, Data Coupling and Control Coupling (Technical Note), 2014.

[4] LDRA Inc, LDRA TBrun User Guide, 2016

[5] LDRA Inc, LDRA Testbed User Guide, 2016

SILS 및 런타임 모니터링을 활용한 레거시 S/W 시나리오 테스팅 및 속성검증 사례연구

박민규, 서보영, 조현승, 이호정
LG전자
H&A제어연구소
{mingyu88.park,by.seo,blve.cho,sophiahj.lee}@lge.com

장훈†
현대자동차
샤시기술센터
fctermer@google.com

요 약

내장형 S/W(Embedded Software)는 H/W와 밀접한 연관성을 갖고 있기 때문에 H/W의 행위 또한 반드시 고려되어야 한다. 이러한 이유로 내장형 S/W를 검증을 위해서 보통 제품 세트 위에서 시스템 단계 테스팅을 중심적으로 검증을 수행하고 있다. 하지만 이러한 제품기반 테스트로는 결함의 검출시기가 늦어지고, 시간속성, 항상성을 가지는 시스템의 요구속성을 검증하기 힘들다는 단점이 있다. 따라서 본 연구에서는 레거시 S/W를 PC위에서 구동하기 위한 SILS 환경을 구축해 조기에 시스템 단계 테스트를 수행할 수 있도록 하고, 속성을 런타임 모니터로 작성해 검증하는 방법을 제안한다. 또한 제안한 방법을 현재 개발중인 세탁기 제어 S/W에 적용해보고, 결함을 검출한 사례를 제시한다.

1. 서론

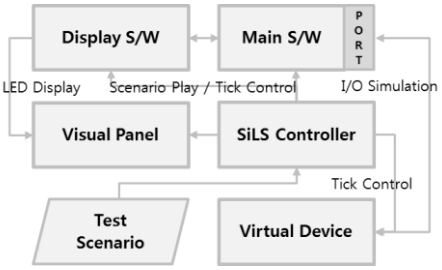
내장형(Embedded) S/W 오작동은 사용자에게 물질적 또는 금전적인 위협을 가할 수 있을 뿐만 아니라, 일반적으로 제품 출하 이후 S/W 수정이 어려운 특징으로 인해 제품 출시 전 개발단계에서의 양산 품질 확보가 중요하다. 테스팅 관점에서 내장형 S/W는 H/W와 밀접한 연관성을 갖고 있기 때문에 보통 제품 기반의 시스템 단계 테스팅을 통해 검증을 수행한다.

제품 기반 시스템 단계 테스팅은 탑재할 대상 H/W가 개발되는 개발 후기단계에 수행되기 때문에 결함을 조기에 도출하기 힘들고, 결함 재현이 힘들며, 결함의 주체를 파악하는데 상당한 기간을 소요하게 되는 문제점이 있다. 또한 H/W의 구동에 의해 시험에 걸리는 시간이 길기 때문에 테스트 기간이 길어질 수 밖에 없는 문제점이 있다. 또한 시스템 단계 테스팅은 제품의 기능에 목적을 둔 검증에는 적합하지만, 시스템이 항시 만족해야 할 속성을 만족하고 있는지 검증하기에는 적합하지 않다. 이는 테스트 케이스가 입력과 출력의 쌍으로 구성되기 때문인데, 시스템의 속성을 검증하기 위한 테스트 조건은 입력과 출력의 쌍 만으로는 도출하기 힘들기 때문이다.

따라서 본 연구에서는 개발 초기에서부터 가속 시험을 수행할 수 있도록 SILS기법을 활용해 테스트 환경을 구축하고, 조기에 시스템 단계 테스팅을 수행할 수 있도록 지원하고자 한다. 또한 속성 검증을 위해 시스템 속성을 런타임 모니터로 작성해 이를 검증하는 방법을 제안한다. 제안한 방법에 대한 사례 연구로 현재 개발중인 수십만 LOC를 가진 세탁기 제어 S/W에 실제 적용하여 결함을 도출한 사례를 제시한다.

2. SILS(Software-in-the-Loop Simulator)

SILS는 S/W 하위 구성요소에 대한 피드백 행위등을 추상화해 PC에서 내장형 S/W를 구동할 수 있도록 하는 시뮬레이션 방법이다. SILS는 제품 H/W가 개발되기 이전부터 제품 S/W의 구동이 가능하기 때문에 조기에 검증활동을 수행할 수 있고, 시험자에 대한 물리적 위험이 없기 때문에 차량 등의 내장형 S/W 도메인 검증 분야에서 특히 주목을 받고 있다[1-2]. 본 연구에서는 이러한 SILS를 조기 검증활동 및 가속 시험을 통한 S/W의 기능성 테스트를 위한 도구로서 활용하고자 한다.



(그림 1) SILS 구성도

본 연구에서는 레거시 S/W에 SILS를 적용하기 위해 환경을 (그림 1)과 같이 크게 5가지 요소로 구성했다.

우선 Display와 Main S/W는 제품에 탑재되는 S/W를 그대로 활용했다. 다만 PC로 코드를 빌드하기 위해 마이크로 프로세서를 위한 특수 문법은 SILS로 빌드할 시에는 제외하도록 매크로로 처리했고, I/O 교환을 위한 포트는 가상 포트로 대체하였다. 다음으로 Visual Panel은 사용자가 제품을 제어하는 키 입력을 처리하고, 상태를 확인을 위한 LED를 출력하는 UI 구성요소이다. 다음으로 Virtual Device는 부하 전장들의 행위들을 모델링 해, Main S/W로의 I/O 피드백을 제공한다. 가상 부하들은 각각 정의된 상태에 의해 독립적으로 동작하고, Main S/W에서의 포트를 통한 요청뿐만 아니라, 외부에서의 요청을 수용할 수 있도록 설계 되었다. 이를 통해 개발자들이 테스트 수행 도중 위험한 상황에 빠질 수 있는 비정상 조건 테스트를 SILS에서도 수행할 수 있다. 모델링 도구로는 Simulink[3]를 활용하였다. 마지막으로 SILS Controller는 SILS 구성요소의 제어를 담당하고 있다. Display와 Main S/W 스레드 제어, 각 구성 요소간의 틱 동기화, 인터럽트 발생 등을 수행한다.

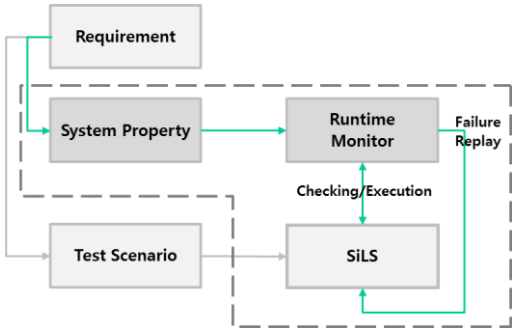
3. 시스템 속성 검증을 위한 Runtime Monitor

런타임 모니터링 기법은 시스템과 함께 병렬로 동작하는 런타임 모니터를 작성하고, 시스템을 항상 모니터링 하는 기법이다. 보통 런타임 모니터링 기법은 시스템의 실패시의 안전(Fail Safe)를 확보하기 위한 시스템 복구(Fail Tolerance)나 성능 분석, 동작 상태 추적 등에 활용되지만 본 연구에서는 이를 속성 위배를 검출하기 위한 오라클로서 사용하고자 한다.

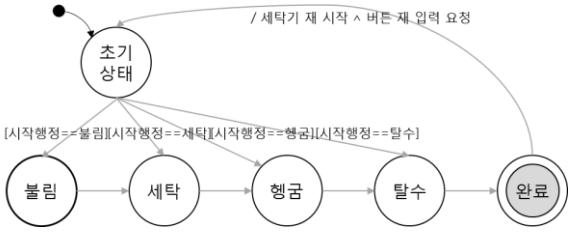
(†본 연구는 저자가 LG전자 재직 당시 수행한 연구임)

본 연구에서 검증하고자 하는 속성으로는 아래와 같은 P_1을 선택하였다.

P_1. 단위행정은 불림-세탁-헹굼-탈수 순으로 동작해야 한다.



(그림 2) 런타임 모니터링 환경 구성도



(그림 3) P_1 만족 여부 판단을 위한 상태 기계

본 연구에서는 검증 속성 P_1과 대응되는 (그림 3)의 상태 기계를 SILS에 삽입하였다. 이렇게 작성된 런타임 모니터는 (그림 2)와 같이 SILS와 함께 동작하며 시스템이 속성을 위배하는지를 항시 확인한다. 만약 런타임 모니터 내부 상태 기계(State Machine)가 오류 상태에 진입하게 되면 SILS는 이를 파악해 개발자에게 경고 및 현재까지의 수행 로그를 제공한다.

4. 사례연구

이번 장에서는 SILS와 속성 검증 모니터의 유용성을 입증하기 위해 현재 개발중인 수십만 LOC의 세탁기 제품에 본 방법을 적용한 사례를 설명한다. 실험은 Windows7, i5-4310M, 12GB RAM을 가진 일반적인 PC에서 수행되었다.

4.1 시나리오 테스트

본 장에서는 중복되지 않는 임의의 테스트 시나리오를 정의하고 시스템 단계 테스트를 수행한다. 아래는 시험 횟수에 따른 분기 커버리지 및 소요된 시험수행기간을 측정한 결과이다.

<표 1> 시나리오 테스트 수행 결과

시험횟수	커버리지	시험시간	실제 수행시간*
10	32%	3분	1일
50	41%	15분	9일
100	42%	33분	18일
200	47%	72분	37일
400	47%	143분	75일

* 실제 수행시간 ➔ 1H 30M 표준 코스, 1일 업무 8시간, 제품 1대.

실험 결과 코드 분기 커버리지는 200회를 기준으로 증가 없이 거의 수렴하는 것을 알 수 있다. 커버되지 않은 코드를 분석한 결과 비정상 조건에 대응하기 위한 보호 코드, H/W와 연관되어 추상화되어 더 이상 호출되지 않는 코드였다. 시험시간은 200회 수행시 약 1시간 12분이 소요되었다. 이는 개발자가 식사나 휴식 시간 중 테스트를 충분히 수행할 수 있는 시간이고, 실제 수행시 약 37일*이 소요되는 테스트이다.

4.2 시스템 속성 검증

3장의 검증 속성 P_1이 만족되는지를 확인하기 위해 코스별 단위 행정 조합의 변경을 유발하는 사용자 버튼 입력 및 H/W 입력 시그널을 무작위로 SILS에 인가한 이후, 동작을 반복적으로 수행하며 (그림 3)의 상태기계를 만족하는지 판별하였다. 상태기계를 만족할 경우에는 위의 과정을 재 수행하고 위배할 경우에는 현재까지 수행한 입력 조건들을 반례로 출력하였다.

SILS의 가속시험(240~300배)에서 약 1시간 동안 200여 회에 걸쳐 검증을 수행한 결과, S/W 내부 우선 순위 큐(Heap)의 잘못된 연산에 의한 헹굼 행정 순서 역전 현상을 검출하였다. 검출된 결함은 Heap의 삭제 연산 도중 이진 트리의 우선순위가 역전되는 문제로 복잡한 복합 조건에서 발생하였다. Heap은 삽입 및 삭제 연산의 조합에 따라 이진 트리의 노드와 내용이 다양하게 결정되기 때문에 일반적인 단위 및 통합 검증을 통해 검출이 힘든 결함이다.

5. 결론

본 연구에서는 제품 세트 위에서 시스템 단계 테스팅을 중심적으로 검증을 수행할 수 밖에 없던 현황을 개선하기 위해 레거시 S/W를 PC위에서 구동하기 위한 SILS 환경을 구축했다. SILS를 통해 조기에 시스템 단계 테스트를 수행할 수 있고, 시스템 요구 속성을 런타임 모니터로 작성해 이를 검증할 수 있음을 보였다. 제안한 방법은 현재 개발중인 세탁기 제어 S/W에 적용, 기존의 시나리오 테스트를 가속 시험을 통해 최대 300배 개선한 효과를 보였고, 속성 검증 결과 1건의 행정 역전 현상을 검출할 수 있었다. 이를 통해 SILS의 가속 검증이 제품을 이용한 테스팅에 드는 시간적 비용과 물리적 한계를 효과적으로 극복할 수 있음을 확인했다.

본 연구에서는 런타임 모니터를 직접 작성해 검증을 수행하였다. 이는 모니터를 추가 작성해야 하는 노력이 필요하기 때문에 향후 모니터의 생성 문법 혹은 패턴을 도출해 모니터를 손쉽게 명세/생성하는 연구를 진행하고자 한다. 또한 시나리오 테스트에서 분기 커버리지가 47%에서 수렴됨을 확인할 수 있었다. 커버되지 않은 코드는 비정상 조건에 대응하기 위한 보호 코드였으며, 이를 커버하기 위한 방법이 필요하다. 향후 이를 커버하기 위한 입력 조건 도출 연구를 진행하고자 한다.

참고문헌

[1] Sooyong Jeong, Yongsub Kwak and Woo Jin Lee, "Software-in-the-Loop simulation for early-stage testing of AUTOSAR software component," International Conference on Ubiquitous and Future Networks (ICUFN), Vienna, 2016, pp. 59-63.

[2] Brückmann, H., et al. "Model-based development of a dual-clutch transmission using rapid prototyping and SiL." International VDI Congress Transmissions in Vehicles. 2009.

[3] MathWorks, SIMULINK http://mathworks.com/products/simulink.html

차량용 SW의 HiL 테스팅을 위한 빅데이터 압축 수집 기법

신종환 [1 O], 최기용 [1], 이병정 [2], 이정원 [1]
아주대학교 전자공학과 [1], 서울시립대학교 컴퓨터과학과 [2]
sjh1334@ajou.ac.kr; ki815kaisian@ajou.ac.kr; bjlee@uos.ac.kr; jungwony@ajou.ac.kr;

A Method of Big Data Compression and Acquisition for HiL Testing of Automotive Software

Jong-Hwan Shin[1 O], Ki-Yong Choi[1], Byeongjeong Lee[2], Jung-Won Lee[1]
Department of Electrical and Computer Engineering, Ajou University[1]
Department of Computer Science and Engineering, The University of Seoul[2]

요 약

차량에 탑재되는 소프트웨어 의존적인 기능이 늘어남에 따라, 차량 업계는 차량 소프트웨어의 품질에 대한 중요성을 인식하고 개념화 단계에서부터 여러 단계의 테스트를 통해 개발 전주기적인 안전성 보증 및 품질 관리를 수행한다. 이 중 통합 시험 단계에서 이루어지는 Hardware-in-the-loop (HiL) 테스트는 통상적인 방법을 이용한 결함 지역화에 한계가 존재하며, 이것을 도와주기 위한 여러 방법들이 있다. 그 중 메모리 업데이트 빈도를 이용한 방법은 대용량의 메모리 정보를 주기적으로 수집하여야 한다. 본 논문에서는 차량용 소프트웨어의 HiL 테스트에서 발생한 결함의 위치추정을 위한 메모리 데이터 수집 기법을 개선한다. 기존 기법에 델타 압축을 적용하여 프로세서와 통신 채널 부하를 최소화하고 빅데이터를 수집한다. 이 방법은 프로세서 부하를 고려한 가변적인 데이터 캐스케이딩 알고리즘을 적용함으로서 프로세서 부하를 안전 한계 아래로 동작하게 하였고, 델타 압축을 통해 통신 채널의 부하를 최소화하였다. 개선된 방법을 검증하기 위해 차량 전장 제어기인 Body Control Module(BCM)의 기능을 구현한 임베디드 시스템에서 실험을 수행하였다. 검증을 통해 제안된 방법을 통해 결함 위치추정을 위한 대용량 메모리 수집이 가능함을 확인하였다.

1. 연구 배경 및 필요성

차량 전자 장치와 제어 소프트웨어는 오늘날의 자동차 산업의 기술 혁신을 이끄는 원동력이다. 차량 제조사는 고객 만족, 엄격한 환경 규제 및 능동 안전을 포함한 다양한 목표를 달성하기 위해 소프트웨어에 의존적인 기능들을 적극적으로 적용하고 있다[1]. 소프트웨어의 오작동은 자동차 리콜의 주요 원인이 되고 있다. 지난 5년간 소프트웨어 결함으로 인한 차량 리콜은 2011년 5퍼센트에서 2015년에는 15퍼센트로 증가하였으며, 그 수는 연간 수 백만 대에 이르고 있다[2]. 차량 소프트웨어의 오작동은 막대한 리콜의 원인일 뿐만 아니라 치명적인 사고의 원인이 된다. 따라서 자동차 소프트웨어의 품질을 보장하는 것은 필수적이며, 이를 위해서는 개념화 단계에서부터 테스트 단계까지 개발 전 주기에 대한 안전성 보증 및 품질 관리가 필요하다.

이에 따라 자동차 산업계는 자동차 소프트웨어에 대한 품질 보증의 중요성을 인식하고 승용차용 전자 장치의 개발 방법론에 대한 국제 표준인 ISO 26262를 제정하여 따르고 있다. 해당 표준은 차량용 전자 장비의 개발 주기 동안 포괄적이고 정확한 테스트를 필요로 하며, 테스트 과정은 통합 시험(Integration Test)을 포함한 여러 단계로 나뉘어진다[3]. 통합 시험에서 테스터는 다양한 소프트웨어 및 하드웨어 시스템을 통합하고 시스템 동작이 시스템 요구 사항에 적합한지 시험한다. 이 단계에서 HiL 테스트는 차량용 소프트웨어를 테스트 하는데 자원 상 여러 가지 이점을 가지고 있기 때문에 널리 선택되는 방법이다.

그러나, HiL 테스트는 테스트 입력과 테스트 출력만을 관찰할 수 있는 블랙박스 테스트로서 결함 발생시 결함 지역화에 한계가 있다. 통상적으로 결함 지역화를 하기

"이 논문은 2017년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (No. NRF-2014M3C4A7030504)."

위해서는, 변수 값에 따라 영향을 받는 지점을 기준으로 코드를 나누는 Code Slicing 기법이 있다. 이 기법은 코드를 나누고 프로그램을 동작시켜 결함의 발생 여부를 확인한다. 그렇지만 자동차 산업의 OEM 구조는 개발자와 테스트 엔지니어가 분리되어 있어 테스트 엔지니어는 현실적으로 코드에 접근하기 어렵다[4]. 또한 차량 제어기와 같은 임베디드 시스템은 내장되어 있는 디버깅 인터페이스를 통해 중단점에 기반한 디버깅을 수행한다. 이 방법은 중단점에 시스템의 동작을 멈추고 레지스터를 관찰하고, 한 구문씩 동작시켜 결함의 발생 여부와 위치를 확인하는 방법이다. 그렇지만 차량을 위한 HiL 테스트에서는 디버깅 인터페이스는 노출되어야 하지 않기 때문에 이용이 어렵다. 또한 다수의 시뮬레이터는 기존 디버깅 인터페이스의 동작 방식인 이벤트 기반 제어가 불가능하다. 마지막으로 통합 테스트 중에는 테스트 대상 하드웨어를 제외한 추가 하드웨어의 부착이 제한된다. 따라서 별도의 데이터 수집 모듈을 이용하는 것은 불가능하다.

따라서 HiL 테스트 중 테스트 대상 시스템(System Under Test, SUT)의 특징인 블랙 박스 시스템에서의 결함 지역화를 위해서 여러 가지 연구가 진행되고 있다. [5]에서는 차량 ECU의 센서 데이터를 이용하여 결함의 발견, 차원 감소, 결함 분류 등 일련의 기계학습 기법을 통해 위험도 추정을 수행한다. [6]에서는 정상 상태의 신호와 비정상 상태의 신호를 수집하고 최소 자승 – 서포트 벡터 머신(LS-SVM)을 이용하여 정상 상태와 비정상 상태를 나눌 수 있는 결함 검출기를 제안하였다. [7]에서는 시스템 결함 발생을 재현하여, 동적 분석을 통해 결함 발생의 근본 원인을 찾는다.

그러나 제안된 연구들의 경우 위에 기술된 HiL 테스트 에서의 제약 사항을 충분히 고려하지 않았기 때문에 그대로 적용할 수 없다. 또한 센서 데이터나 입출력 데이터와 같이 시뮬레이터의 출력을 이용한 방법은 결함이 일어나는 조건을 지역화 하는데 효과적이지만, 프로그램 코드의 어떤 부분에서 결함이 발생했는지에 대한 정보가 없기 때문에 프로그램 코드의 결함 지역화에 어려움을 겪는다.

우리는 이 문제를 해결하기 위해 메모리 업데이트 빈도를 활용하는 방법을 제안하였다[8]. 이 기법은 SUT의 테스트 입력에 따라 업데이트 되는 메모리의 빈도를 계산하고, 결함 상황에서 입력이 반영되는 구간의 업데이트 빈도가 증가하는 신호를 결함의 후보군으로 제시한다. 해당 기법에서는 대용량의 메모리 정보가 획득되어야 한다. 그러나 일반적으로 차량 전장 네트워크로 사용되는 CAN (Controller Area Network)의 대역폭은 실시간 메모리 정보를 전송하기에 대역폭이 충분하지 않다. 그러므로, CAN의 제한된 대역폭을 극복하여 메모리 정보를 획득할 수 있는 데이터 캐스케이딩 방법을 제안하였다[9]. 그림 1은 해당 방법의 전체적인 구조를 표현하고 있다. 해당 방법에는 시뮬레이터, SUT, 모니터링 시스템으로 구성된다.

시뮬레이터는 SUT에 테스트 입력을 넣고, 테스트 출력을 받는다. SUT에서 생성되는 메모리 정보는 SUT 내에 있는 전송 에이전트가 수집하게 된다. 데이터는 대역폭 극복을 위해 여러 세그먼트로 분할되어 한 세그먼트씩 모니터링 시스템에 전달된다. 그 뒤, 시뮬레이터는 동일한 테스트를 반복하여 전송 에이전트가 전체 세그먼트를 수집할 수 있게 한다. 마지막으로 모니터링 시스템에서는 전송된 세그먼트를

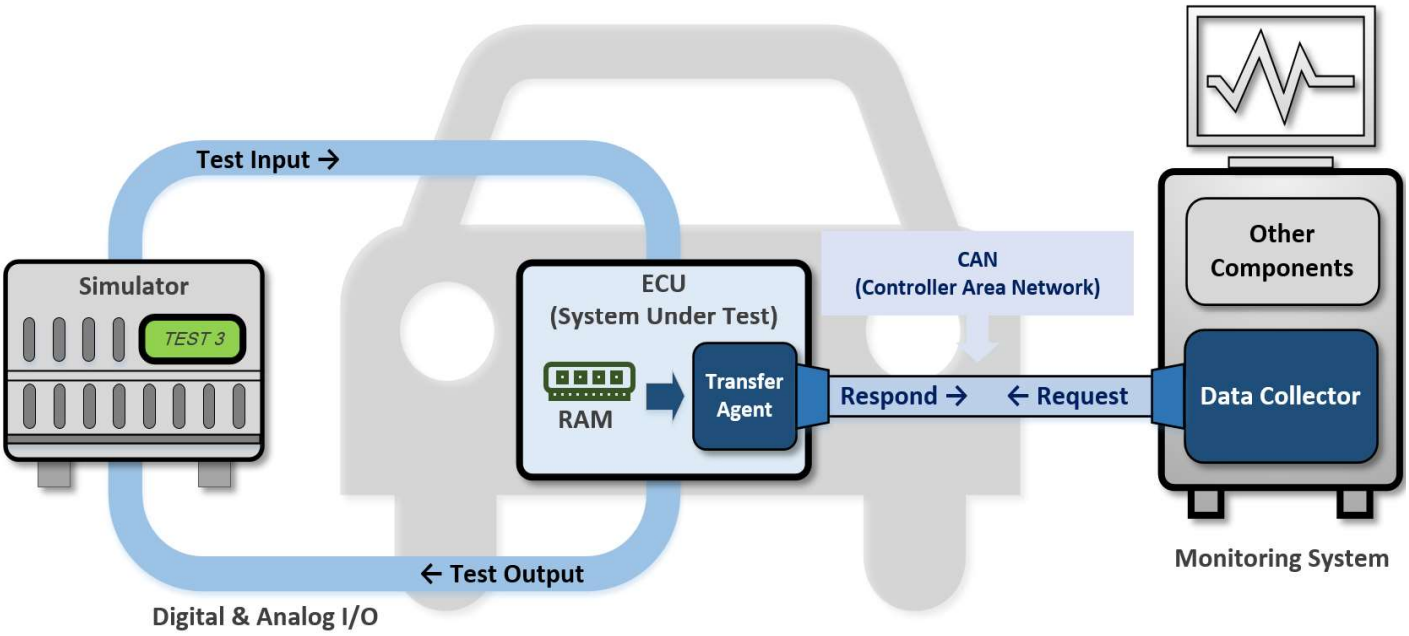


그림 1. 데이터 캐스케이딩 기법의 전체 구조

조합하여 시간에 따른 메모리 정보를 완성한다.

한편, 기존 연구에서는 압축에 대한 가능성을 일축하였다 [9], [10]. 기존 제안된 방법으로 압축을 할 경우 실시간 데이터 전송이 가능한 압축률을 달성하지 못하였기 때문이다. 또한 압축 알고리즘으로 인한 시스템 부하가 고려되지 않았었다. 그렇지만 전체 데이터 대신 각 세그먼트를 압축하게 되면, 원하는 대역폭을 확보할 수 있고 실시간 데이터를 송신할 수 있게 된다. 그리고 수집되는 데이터를 관찰한 결과 각 반복간 데이터가 대부분 중복된다는 것을 확인하였으며, 실험을 통해 수천 개의 연산보다 한 번의 데이터 전송이 테스트 반복시 더 높은 프로세서 점유를 발생시키는 것을 확인하였다. 따라서 시간에 따라 발생하는 패킷의 차이만 전송하는 기법인 델타 압축을 이용하면 적은 시스템 부하로도 전송량은 감소하고, 전송되는 정보의 양은 늘어나 차량용 빅데이터 수집의 효율 개선 가능성을 확인하였다.

따라서 본 논문에서는 개선된 빅데이터 획득 기법을 제안한다. 이 방법은 블랙박스 테스트 중 SUT의 데이터 획득을 위해 데이터를 분할 전송하는 데이터 캐스케이딩 방법에서 전송되는 각 세그먼트의 패킷을 압축시켜 전송량을 감소시키고 통신 채널 및 프로세서의 부하를 줄인다. 이 방법은 해당 세그먼트의 첫 번째 반복의 데이터는 그대로 전송하고, 나머지 반복은 첫 번째 반복의 데이터와의 차이만을 전송하는 델타 압축을 통해 달성될 수 있다. 또한 제안된 방법을 다양한 부하 상황에서 데이터 무결성, 성능 비교, 신뢰성 검증 과정을 거쳐서 제안하는 알고리즘의 강건성과 사용 가능성을 입증한다.

이 연구를 통해 블랙박스 테스트 중 임베디드 시스템에서의 동작을 설명하기 위한 메모리 동작을 얻는데 도움이 되며, 이것은 블랙박스 테스트에서의 메모리 정보를 이용한 결함 지역화의 기초가 될 수 있다.

논문의 나머지 장은 다음과 같이 이루어져 있다. 2장에서는 CAN 프로토콜의 변형과 데이터 압축을 통한 대역폭 극복에 대한 관련 연구에 대해 설명 한다. 3장에서는 기존 데이터 캐스케이딩 방법을 간략하게 소개한다. 4장에서는 데이터 캐스케이딩 방법을 위한 압축 방법을 제안한다. 5장에서는 제안하는 방법의 검증을 위해 실험을 수행한다. 마지막으로 6장에서는 전체 내용을 요약하고 향후 연구를 간단하게 서술한다.

2. 관련 연구

같은 통신 채널에서 데이터 전송 속도를 개선하기 위한 관련 연구로는 크게 두 가지가 있다. 하나는 기존 CAN 프로토콜을 변형한 대역폭 극복 방법이다. 다른 하나는 데이터 압축을 통해 데이터를 변형하여 전송량을 감소시키는 방법이다. 대역폭 극복 방법으로는 [11], [12]가 있었고, CAN을 위한 압축 알고리즘은 [13]-[16]가 있었다.

2.1 CAN 프로토콜 변형

차량 전장 네트워크인 CAN은 버스형 구조의 네트워크이다. 내고장성 향상과 전자기에 의한 영향을 최소화 하기 위해 하나의 CAN 메시지는 실제 데이터가 전송되는 Data Field를 제외하고도 Identifier를 나타내는 Arbitration Field, Data Length를 나타내는 Control Field, 에러 검출을 위한 CRC Field와 메시지 손실을 최소화 하기 위한 ACK Field가 존재한다[17].

하나의 통신 채널에 모든 ECU가 연결되어 있어, 최근 차량 내 탑재되는 ECU의 개수가 늘어남에 따라 CAN을 통해 전송되어야 하는 정보가 늘어났다. 따라서, 최대 1Mbps로 제한되는 기존 CAN 프로토콜을 수정하여, 같은 통신 채널에서의 전송 속도를 개선하려 하려는 연구가 있다[11], [12].

[11]에서는 대역폭 이상의 큰 데이터를 세그먼트로 분할하여 CAN의 제한된 대역폭 내에서 전달하고 세그먼트 시퀀스를 나타내기 위해 Control Field 내부의 사용되지 않는 영역인 Reserved bit을 재정의하여 데이터의 순서를 나타내도록 하였다. [12]에서는 기존의 데이터 프레임 중 사용되지 않은 회색 영역을 통해서 정보를 전송함으로써 CAN의 대역폭을 늘리는 방법을 제안하였다.

하지만, 두 연구 모두 기존 프로토콜의 데이터 영역 이상을 사용하여 기존 프로토콜의 수정이 필요하기 때문에 CAN 드라이버의 수정이 불가피하였다. 기존 프로토콜을 수정하는 것은 이미 개발된 시스템을 변경하여야 하기 때문에 우리의 목적인 시스템 디버깅 정보 획득의 수단으로 적합하지 않았다. 또한 프로토콜 변경 후 CAN Message 전송에 사용되는 프로세서 및 통신채널 자원에 대한 고려가 부족하여 기존 시스템 작업의 주는 영향을 최소화하여야 하는 우리의 제약조건에 적합하지 않았다.

2.2 압축을 통한 데이터 변형

차량 전장 네트워크인 CAN을 통해서는 주로 차량의 여러 센서 값, 차량 속도 등 차량 동작을 위한 매개변수들이 전송되며, 이는 실제 차량 동작과 운용 환경의 반영이기 때문에 연속적이거나 고정된 값이 높은 확률로 전송된다. 이런 점에서 착안하여 데이터 압축을 통해 제한된 대역폭을 극복하거나 버스 점유율을 줄이기 위한 연구가 존재한다[13]-[16].

CAN을 사용하는 시스템이 주로 임베디드 시스템인 특성상 압축을 위한 계산에 필요한 프로세서 자원이나 저장 공간이 큰 경우 구현에 어려움이 있다. 따라서 해당 제약 사항 안에서 압축을 수행할 수 있도록

부하가 작은 알고리즘을 주로 사용한다. CAN을 위한 압축 알고리즘은 주로 델타 압축을 사용하며, 이 방법은 첫 번째 데이터는 실제 값을 그대로 보내고, 나머지 내용들을 첫 번째 데이터의 변경된 값만을 보내어 원래 데이터보다 작은 양의 데이터를 보낸다.

데이터의 변경된 값인 델타 값의 데이터 인코딩에 따라 압축 방법이 변화한다. 델타 압축을 위해서는 해당 매개변수가 변경되었음을 알리는 알림 비트과 변화량이 전송되어야 하기 때문에 너무 작은 값은 압축되지 않는다. [13],[16]에서는 5비트 이상의 변수의 경우 변화 값을 보내어 델타 압축을 수행한다. [14]에서는 6비트 이상의 매개 변수를 전송할 때 변화하는 범위가 ±15이하의 값들을 변화 값만 보내는 Boundary of fifteen 알고리즘을 제안하였다. [15]에서는 데이터를 특정 숫자로 나누어 몫의 변화값과 나머지만을 전송하는 Quotient Remainer 알고리즘을 제안하였다.

그렇지만 위에서 서술한 압축 알고리즘의 경우 데이터의 변화량에 따라 압축률이 변하고, 압축률이 가장 높을 때에도 실시간 메모리 데이터 전송이 가능할 만큼 압축률을 달성하지 못했기 때문에 실시간 데이터 전송에는 적합하지 않았다. 그렇지만, 이 방법들을 이용하여 각 분할된 데이터들을 압축하는데 도움이 되었다. 위의 관련 연구를 기반하여 우리는 전송 에이전트가 버스 점유율을 줄이는 방법을 고안할 수 있었다. CAN을 위한 압축 알고리즘을 사용하는 방법으로서 4장에 소개 한다.

3. 기존의 데이터 캐스케이딩 기법

데이터 캐스케이딩 기법은 차량 전장 네트워크인 CAN의 좁은 대역폭을 극복하여 대용량 데이터를 얻기 위한 방법이다. 이 방법은 크게 데이터를 전송하는 SUT와 데이터를 수신하는 모니터링 시스템으로 나뉜다. 이 방법은 통합 테스트 중 결함 발생시 결함이 발생한 테스트를 반복하며 원하는 메모리 영역을 얻는다. 해당 알고리즘은 대역폭을 포함한 다양한 변수를 고려하여 데이터 분할을 수행하고, 분할된 데이터들은 새로 정의된 메시지 프로토콜을 이용하여 버퍼에 들어간다. 그리고 프로세서 부하를 고려한 가변적인 데이터 캐스케이딩 알고리즘을 통해 데이터를 전송한다.

3.1 데이터 세그멘테이션

데이터 세그멘테이션은 전송 대상 데이터를 나누는 작업으로서, 데이터 범위, SUT의 주 작업 주기 동안 전송 가능한 CAN 메시지 수, 한 패킷에 담길 수 있는 데이터의 양, 타 작업들의 통신 채널 부하를 고려하여 결정된다. 데이터가 분할되는 수는 동작하는 시스템의 안정성과 테스트 반복 횟수에 영향을 미치기 때문에 적절한 세그먼트 숫자를 정하는 것이 중요하다. 식 (1)은 세그먼트 숫자를 결정하는 수식을 표현하고 있다.

$$\#\ \text{of Segments} = \frac{D_{monitor}}{C_m \times D_m \times (1-OCC)} \quad (1)$$

- $D_{monitor}$: 전송하려는 데이터 범위
- C_m: SUT의 주 작업 주기동안 전송 가능한 메시지 수
- D_m: 한 패킷 동안 전송 가능한 바이트 수
- OCC: 타 작업이 점유하는 통신 채널 부하

3.2 메시지 프로토콜

데이터를 분할하여 전송하기 위해서 메시지 프로토콜을 정의하였다. 우리는 CAN 메시지 프레임의 다른 영역을 사용하지 않고 데이터 프레임만을 사용하여 프로토콜을 존재한다. 그림 2는 데이터 캐스케이딩을 위한 프로토콜을 나타내고 있다.

Protocol Type	BYTES								
	0		1	2	3	4	5	6	7
	0	1-7							
DR	Info	RSVD		# of Iterations		Start Address		End Address	
CI	Info	Offset		Iteration Index		Reserved			
DT	Info	Base Addr		Data of Interest					

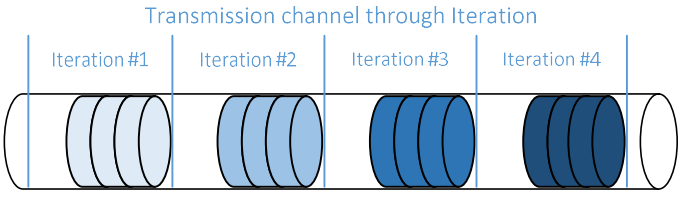
그림 2. 데이터 캐스케이딩을 위한 메시지 프로토콜

- Data Request Protocol(DR): 데이터를 요청하는 역할을 하며, 전송 에이전트에게 관심 데이터 범위와 반복 횟수를 알려주는 프로토콜이다.
- Cascading Information Protocol(CI): 현재 전송되는 패킷의 데이터 범위와 반복 횟수를 알려주는데 사용되는 프로토콜이다.
- Data Transfer Protocol(DT): 실제 데이터를 전송할 때 사용되는 프로토콜이며, 데이터의 시작 주소와 데이터를 포함하고 있다.

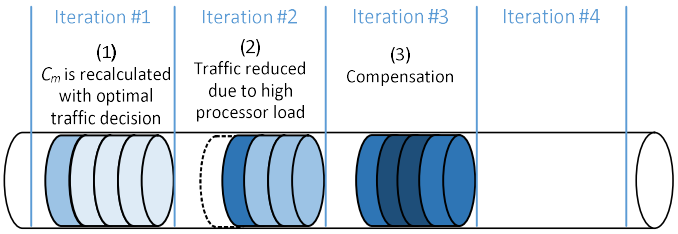
3.3 가변적인 데이터 캐스케이딩 기법

[9]에 제안된 기법은 데이터를 고정된 전송량으로 송신한다. 시스템 동작에 영향을 미치지 않도록 전송량을 결정하나 시스템의 비정상적인 동작으로 인해 프로세서 부하가 높아졌을 경우 전송 에이전트의 동작으로 인한 시스템 오류가 발생하였다. 따라서 [10]에서는 사전 테스트 세션을 통한 최적 전송량 결정, 가변적인 전송 에이전트 및 보상 알고리즘을 통해 개선한다.

그림 3은 각각 기존 기법과 가변적인 데이터 케스케이딩 기법의 데이터 전송을 나타낸다. (a)에 나타난 기존 기법은 일정한 전송량으로 전송 작업을 수행한다. 반



(a) [9]에서 제안된 기존 기법의 데이터 전송



(b) [10]에서 제안된 가변적인 기법의 데이터 전송

그림 3. 기존 기법과 가변적인 데이터 케스케이딩 기법의 데이터 전송

면에, (b)에 나타난 가변적인 기법은 사전 테스트 세션 시 수행되는 최적 전송량 결정 과정을 통해 한 주기 동안 전송되는 패킷의 수(C_m)를 최적화한다(1). 또한 프로세서 고 부하 상황에서 전송 작업을 조절하고 미전송된 패킷을 저장한다(2). 그리고 시스템의 유휴 자원이 발생할 때마다 미전송된 패킷을 전송하여 보상한다(3).

4. 델타 압축을 통한 전송 에이전트의 채널 부하 개선

데이터 캐스케이딩 알고리즘을 이용하여 메모리 정보를 전송한 결과 시간 변화에도 불구하고 같은 영역의 메모리 정보는 대부분 중복되었다. 따라서 기존의 차량의 매개변수 송신시 압축을 통해 점유율을 줄이는 CAN을 위한 방법을 이용하여 전송시의 버스 점유율을 낮출 수 있다는 가능성을 확인하였다.

다만 HiL 테스트 중에 대용량 데이터를 얻기 위해 시스템의 유휴 자원을 이용하여 데이터를 전송할 경우 압축 알고리즘을 적용하기 위해서는 몇 가지 고려사항이 있다. 먼저, 데이터 획득 작업의 시스템 프로세서의 부하이다. 시스템의 유휴 자원을 이용하여 실시간으로 데이터를 전송하기 때문에, 통신 채널의 부하뿐만 아니라 프로세서 부하, 그리고 통신 채널의 버퍼로 사용되는 공간도 같이 고려해주어야 한다. 따라서 해당 제약사항을 만족하지 못하는 알고리즘은 제외되어야 한다. 두 번째, 기존 프로토콜의 변경이 불가능하다. 전송 에이전트는 SUT의 주 작업에 삽입되어 작업을 수행하기 때문에 CAN 프로토콜 등 기존 시스템의 설정을 변경할 수 없다. 따라서 이러한 고려 사항을 반영하여 다음과 같은 델타 압축 알고리즘을 제안한다.

4.1. 델타 압축을 위한 데이터 캐스케이딩의 순서

그림 4은 델타 압축을 통해 개선된 데이터 캐스케이딩 방법의 전체 과정을 나타내고 있다. 해당 방법은 송신과 수신 작업으로 나뉠 수 있다. 송신 작업에서 전송 에이전트는 메모리 데이터 중 관심영역을 분할하여 첫 번째 영역부터 수집한다(1). 그 뒤 델타 압축을 통해서 데이터 압축을 수행한다(2). 그 뒤 시스템 프로세서 부하에 따라 가변적으로 전송량을 조절하며 압축된 영역을 전송한다(3). 테스트가 반복될 때마다 다른 분할된 영역을 차례대로 전송한다. 전송 중 높은 프로세서 부하가 발생하면 전송량을 조절한다(4). 이 때 전송되지 못한 경우 보상 버퍼에 저장하였다 시스템이 다시 낮은 부하를 가질 때 전송하여 보상하게 된다(5).

데이터 수집기는 송신된 패킷을 수집한다(6). 수집된 패킷들은 세 가지 단계를 통해 모니터링 시스템 내 데이터베이스로 삽입된다. 먼저, 보상된 패킷의 제자리를 찾는 작업이 수행된다(7). 그리고 압축된 데이터를 복원한다(8). 마지막으로 분할된 영역으로 보내어진 패킷들을 재조립하여 시간에 따른 메모리 데이터로 완성한다

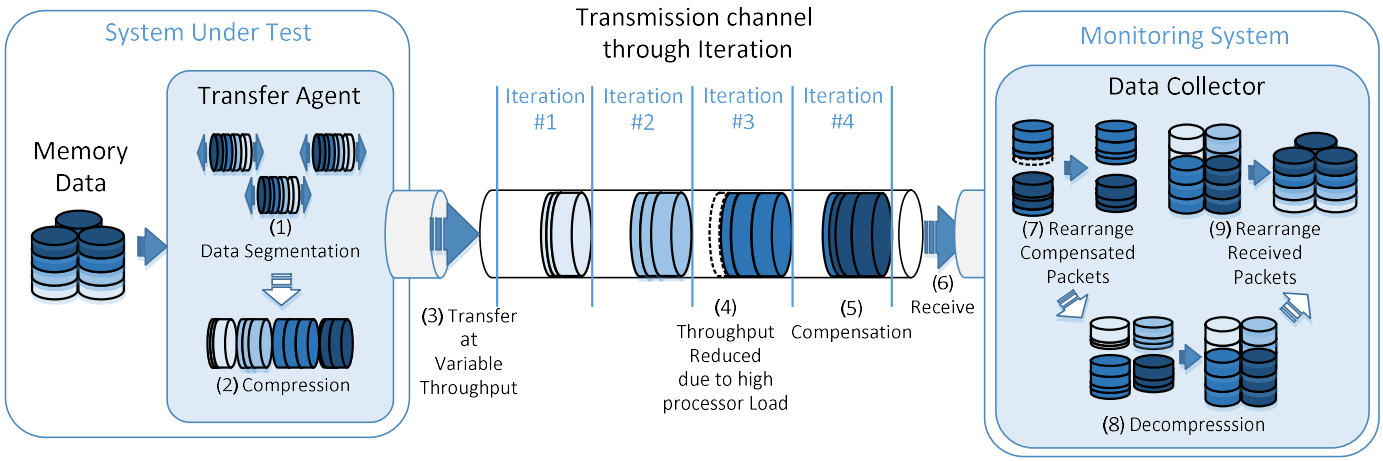


그림 4. 델타 압축을 통해 개선된 데이터 캐스케이딩 방법

(9). 기존 기법을 개선하기 위해 Data Transfer Protocol을 델타 압축을 위해서 재정의 해야 하며, 우리의 방법에 맞는 델타 압축 방법이 필요하다. 4.2절은 데이터 캐스케이딩 방법을 위한 델타 압축 방법을, 4.3절은 델타 압축 방법에 맞는 새로운 Data Transfer Protocol을 제안한다.

4.2. 델타 압축 방법

델타 압축을 위한 알고리즘은 SUT 내 전송 에이전트가 담당하는 송신부와 모니터링 시스템 내 데이터 수집기가 담당하는 수신부로 나뉜다. 그림 5는 송신부의 델타 압축 알고리즘을 나타내고 있다. 분석 결과 같은 데이터 영역에서는 반복 횟수가 변하여도 많은 부분이 중복되기 때문에 첫 번째 반복의 데이터와의 변화값만을 전송하도록 하였다. 이 알고리즘은 먼저 첫 번째 반복인지를 확인한다. 첫 번째 반복이라면, 필요한 매개변수를 초기화하고 현재 수집한 내용을 CompressionBuffer로 삽입한다. 나머지 반복에서는 CompressionBuffer와의 비교가 수행된다. 첫번째 반복의 데이터와 동일하지 않으면, CompressionTxBuffer에 변경된 내용을 삽입한다. 그리고 CompressionIndicator에 변경되었음을 표시함으로서 압축 과정이 수행된다.

Algorithm 1 Compression by Transfer Agent

```
Input:
    TxBuf - Current acquired Information from transfer agent
    FrameNum - Current iteration sequence
    C_m - The number of message can be transfered on SUT's single period
Output:
    CompressionIndicator - Indicator bytes for indicator messages
    CompressionTxBuffer - Values of Difference Vector Messages
 1: procedure CompressDataSegment
 2: begin
 3:     for currentSeq = 0 to C_m do
 4:         for i = 0 to size(TxBuf) do
 5:             if FrameNum = 1 then
 6:                 Initialize countDiff, countIndicator to zero
 7:                 FlushCompressionTxBuffer, CompressionIndicator
 8:                 CompressionBuffer[currentSeq][i] ← TxBuf[i]
 9:             else
10:                 if CompressionBuffer[i] != TxBuf[i] then
11:                     countDiff ← countDiff+1
12:                     CompressionTxBuffer[countDiff][i] ← TxBuf[i]
13:                     Mark on CompressionIndicator
14: end
```

그림 5. 송신부의 델타 압축

그리고 모니터링 시스템에서 수행되는 압축 해제 알고리즘은 다음과 같이 수행될 수 있다. 그림 5는 수신부에서의 압축 해제 알고리즘을 나타내고 있다. 먼저 알고리즘은 Difference Vector Message의 현재 위치를 나타내는 diffSeq를 0으로 초기화 한다. 그 뒤 모든 CI Packet 마다 현재 패킷뒤부터 다음 CI 패킷 전까지의 Message를 CompMes에 저장한다. 만약 해당 CI 패킷이 첫 번째 반복의 패킷이면, CompMes를 첫 번째 패킷들의 집합인 FirstMesBuff로 저장하고, 이것을 DecompBuff로 보낸다. 그렇지 않다면, CompMes에서 Indicator Message와 Difference Vector Message를 분리한다. 그리고 Indicator Message의 각 Bit에 따라서 DiffVector로부터 변경된 값을 DecompBuff로 보낸다.

Algorithm 2 Decompression by Data Collector

```
Input:
    AcquiredData[] - Acquired data from transmission
    D_m - The length of data in normal data transfer packet
    C_m - The number of message can be transfered on SUT's single period
Output:
    DecompBuff[] - Decompressed data
 1: procedure DecompressDataSegment
 2: begin
 3:     for each CI packet do
 4:         initalize diffSeq to zero
 5:         CompMes ← Data from the first DT packet to the next CI Packet
 6:         if This is the first iteration then
 7:             Copy(CompMes,FirstMesBuff[offset],sizeof(CompMes))
 8:             Copy(CompMes,DecompBuff,sizeof(CompMes))
 9:         else
10:             Copy(FirstMesBuff[offset],DecompBuff,sizeof(CompMes))
11:             indicatorSize ← D_m × C_m ÷ 8
12:             compMesSize ← lengthof(CompMes)
13:             Copy (CompMes, IndicatorMes, indicatorSize)
14:             Copy (CompMes, DiffVector, compMesSize - indicatorSize)
15:             for i = 0 to C_m do
16:                 indicator ← IndicatorMes[i]
17:                 for j=0 to D_m do
18:                     if indicator indicates change on following bit then
19:                         DecompBuff[i][j] ← DiffVector[diffSeq]
20:                         diffSeq++
21: end
```

그림 6. 수신부의 압축 해제 과정

4.3. 델타 압축을 위한 Data Transfer Protocol 재정의

이 장에서는 델타 압축 방법을 위해 DT 프로토콜을 재정의한다. 두 가지 경우에 따라 DT Protocol이 정의된다. 압축시는 제안된 프로토콜을 따르며, 미 압축시에는 기존 프로토콜을 따른다.

Protocol Type	BYTES								
	0		1	2	3	4	5	6	7
	0	1-7							
DT	Info	Base Addr	Data of Interest						
DI (C_m=15)	IB [0]		IB [1]	IB [2]	IB [3]	IB [4]	IB [5]	IB [6]	IB [7]
	IB [8]		IB [9]	IB [10]	IB [11]	IB [12]	IB [13]	IB [14]	IB [15]
DV (Diff=6)	DV [0]		DV [1]	DV [2]	DV [3]	DV [4]	DV [5]		

그림 7. 기존 Data Transfer Protocol과 압축을 위해 제안된 프로토콜

그림 7은 기존 프로토콜과 재정의된 프로토콜을 보여주고 있다. 재정의된 프로토콜은 두 부분으로 구성되며, Difference Indicator(DI)와 Difference Vector(DV)로 나뉜다. DI는 SUT의 1 주기 전송횟수인 개의 Indicator Byte(IB)로 구성된다. 각 IB는 하나의 패킷을 의미하며,

1인 경우 해당 패킷의 해당 바이트가 변경을 의미한다. Difference Vector(DV)는 변화된 값을 보내기 위한 프로토콜이다. 다만 메모리 정보는 변화 위치와 변화량이 불분명하기 때문에 변경된 부분이 그대로 전송된다. DV Message의 전체 길이는 변경된 부분의 길이이다.

그림 8은 전송 대상 원본 메시지와 압축 후 전송되는 메시지를 표현하고 있다. (a)에서는 전송 대상 영역의 첫번째 획득의 데이터를 나타내고 있다. (b)에서는 전송 대상 데이터를 나타내고 있다. 이것은 (a)와 같은 영역이지만 첫 번째 획득과 비교하였을 때 'DF','04','0A' 부분이 다르다. (c), (d)는 (b)에 나타난 전송 대상 데이터를 압축한 결과이다. 이 경우, 메시지는 크게 두 부분으로 나뉜다. (c)는 각 패킷의 각 바이트가 변경 되었음을 알려주는 Indicator Message 부분이며 패킷의 순서에 따라 차례대로 구성되어 있다. '08'은 이진수로 '001000'으로서 두 번째 패킷의 주소 바이트를 제외한 세 번째 위치가 변경 되었음을 나타낸다. '10'및 '04'도 마찬가지로 세 번째, 네 번째 패킷의 변화위치를 나타낸다. (d)는 변경된 데이터 부분인 Difference Vector Message이다.

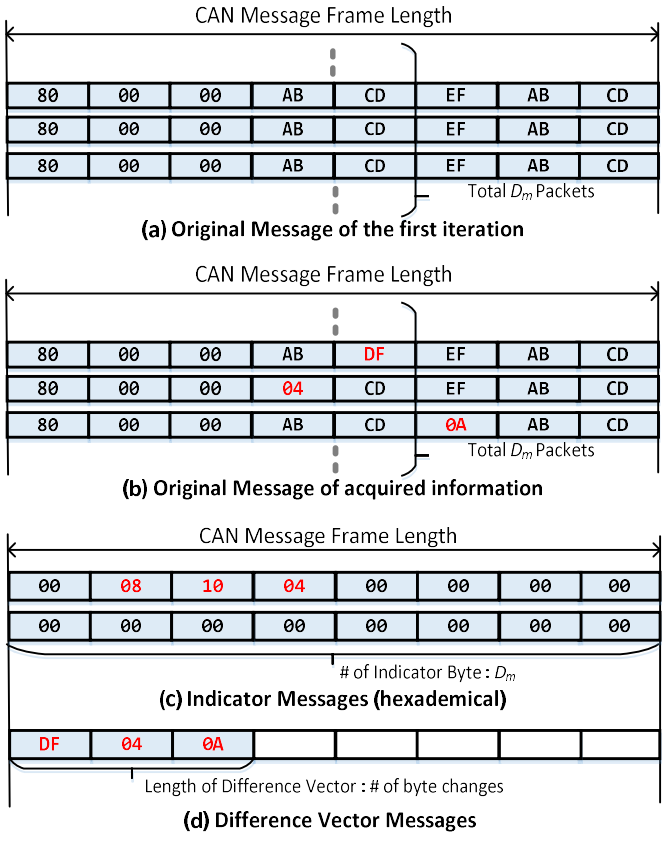


그림 8. 전송 대상 원본 메시지와 재정의된 Data Transfer Protocol을 따라 압축된 메시지

5. 실험 및 검증

우리는 제안된 방법의 검증 및 기존 방법과의 비교를 위하여 실험을 수행하였다. 실험은 데이터 검증 실험, 기존 알고리즘과의 성능을 비교하기 위한 성능 비교 실험, 마지막으로 프로세서 부하와 채널 부하를 측정하는 신뢰성 검증 실험이다.

5.1. 실험 환경 및 조건

먼저 실험 환경은 표 1과 같이 설정하였다. SUT와 모니터링 시스템의 환경은 다음과 같다. SUT는 자동차 제어기로 널리 사용되는 Freescale S12XF 임베디드 보드와 운영체제로는 OSEK-OS 표준을 준수하는 Qplus-Auto를 이용하였으며, 차량 전장 제어기인 Body Control Module(BCM)을 시뮬레이션 하도록 하였다. 모니터링 시스템을 위해서는 PC를 이용하였고, 제안된 data cascading 알고리즘을 통해 전송된 데이터의 수신을 위한 MATLAB 어플리케이션을 만들어 실험을 수행하였다. 아래 표는 실험 환경을 나타내고 있다.

표 1 실험 환경 구성

실험 환경	구성	
SUT	H/W	Freescale S12XF 기반 임베디드 보드
	S/W	Qplus-auto(OSEK OS 표준 준수)
모니터링 시스템	H/W	PC, USB-CAN Analyzer, NI DAQ
	S/W	MATLAB

각 실험 동안 프로그램의 데이터 영역을 얻는 실험이 수행하였다. 실험 조건은 시스템이 유휴 상태일 때(유휴), 그리고 부하 상태일 때를 비교하였으며, 부하 상태일 때에는 각 20,40,60%(부하1-3)의 시스템 부하가 10초 간격으로 발생하며 10초 간격으로 유지 되도록 하였다.

5.2. 데이터 검증 실험

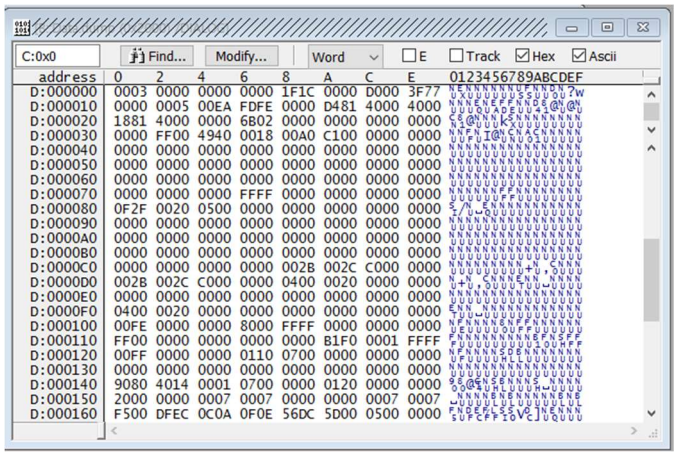
데이터 검증 실험은 제안한 방법을 이용하여 획득된 데이터가 실제 데이터와 비교시 일치 여부를 확인하는 실험이다. 우리의 제안된 방법을 통해 얻은 데이터를 임베디드 시스템의 Debugger인 Trace 32를 통해 얻은 데이터와 비교하게 된다. 실험 결과 다음과 같은 결과를 확인할 수 있었다.

표 2 데이터 검증 실험(회수율, 단위: 패킷 수)

	유휴	부하-1	부하-2	부하-3
[9]	274922/ 274922 (100%)	274922/ 274922 (100%)	274922/ 274922 (100%)	274922/ 274922 (100%)
[10]	274922/ 274922 (100%)	342972/ 342972 (100%)	457296/ 457296 (100%)	685944/ 685944 (100%)
제안 기법	55901/ 55901 (100%)	55901/ 55901 (100%)	55901/ 55901 (100%)	55901/ 55901 (100%)

표 2은 데이터 검증 실험의 회수율을 나타내고 있다. 표 안의 숫자는 수신한 패킷 수를 나타내고 있다. 패킷 수는 프로그램 영역의 크기, 획득을 원하는 시간 범위, 1회당 메시지 전송 횟수 및 획득 횟수에 따라 달라지게 되며 식(1)을 이용하여 획득 횟수를 결정하게 된다. 실험에서는 프로그램 영역의 크기는 16.33KB으로 동일하였으며, SUT 주기당 메시지 전송횟수에 따라 획득 횟수가 달라지게 되었다. 시간에 따른 메모리 변화를 획득 시 1회당 전송 횟수에 따라 유휴시에 3.3MB 가량의 데이터를 획득하였으며, [10]의 부하-3 실험의 경우 8.2MB 가량 획득하게 된다. 대조군 1,2와 실험군은 4가지 상황에 따라 30번 수행되었으며, 전체 1.43GB의 데이터를 획득하였다.

회수율은 전체 데이터 영역 중 실제로 회수한 영역의 비율을 의미한다. 유휴 상태에서는 거의 모든 기법이 100%를 달성한 것을 볼 수 있다. 그러나 부하-2,3 상황에서 [9]의 기법은 회수는 완전하나, 프로세서 부하로 인한 지연이 생기기 때문에 이 때 수신된 패킷은 무효하다.



(a) Debugger를 이용한 메모리 획득 결과

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	03	0F	00	00	00	00	00	1F	1C	00	00	D0	00	3F	77
00	10	00	00	00	05	00	00	FD	FE	00	00	D4	81	40	00	40	00
00	20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	30	00	00	FF	00	49	40	00	78	00	A0	C1	00	00	00	00	00
00	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	70	00	00	00	00	00	00	FF	FF	00	00	00	00	00	00	00	00
00	80	0F	2F	00	20	05	00	00	00	00	00	00	00	00	00	00	00
00	90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	C0	00	00	00	00	00	00	00	00	00	2B	00	2C	C0	00	00	00
00	D0	00	04	00	00	C0	00	00	00	04	00	00	20	00	00	00	00
00	E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	F0	04	00	00	20	00	00	00	00	00	00	00	00	00	00	00	00

(b) 제안 기법을 이용한 메모리 획득 결과

그림 9. Debugger와 제안 기법을 이용한 메모리 획득 결과

그림 9는 메모리의 특정 영역을 Debugger를 이용한 방법과 제안 기법을 이용하여 얻은 메모리 영역을 보여주고 있다. 제안된 기법을 이용하여 획득한 영역을 Debugger를 이용한 메모리 획득 결과와 비교한 결과 두 결과는 그림 9와 같이 모두 일치하는 것을 볼 수 있어 제안된 기법이 실제 메모리 정보를 획득할 수 있다는 것을 보여주었다.

5.3. 성능 비교 실험

성능 비교 실험은 각 기법에 따른 전송 속도를 비교하는 실험이다. 전체 데이터를 전송하는데 드는 시간으로 비교한다. 표 3은 성능 비교 실험의 결과를 나타내고 있다. 각 소요 시간은 초로 나타내어졌다. 실험 결과 [9]는 부하 상황에서도 크게 시간이 증가하지 않았지만, 프로세서에 미치는 영향이 고려되지 않았다. [10]은 최적 전송량 결정 알고리즘을 통해 부하 상황에서는 표준 전송량을 줄이도록 설계되어 부하가 높아질수록 데이터 획득에 필요한 시간이 크게 늘어나는 것을 확인할 수 있었다. 그리고 제안한 기법은 프로세서의 부하에 따라 전송량을 결정하지만, 1회 전송량이 크지 않기 때문에 시간 변화가 크지 않은 것을 확인할 수 있었다.

표 3 성능 비교 실험(시간, 단위: 초)

	유휴	부하-1	부하-2	부하-3
[9]	183.8s	183.8s	183.8s	183.8s
[10]	185.3s	185.3s	386.1s	1139.6s
제안 기법	183.6s	184.3s	183.8s	183.8s

5.4. 신뢰성 검증 실험

신뢰성 검증 실험은 제안된 방법이 반복되는 테스트 동안 프로세서 부하나 채널 부하가 안전 한계를 초과하지 않았는가를 확인하는 실험이다. 따라서 데이터 획득시의 프로세서 부하와 채널 부하를 측정하였다. 표 4는 최대 프로세서 부하를 측정한 결과를 보여주고 있다. 또한 [10] 및 제안된 기법에서는 프로세서의 부하에 따라 전송량을 조절하여 안전 한계인 95%를 넘지 않는 것을 볼 수 있었다.

표 4 최대 프로세서 부하

	유휴	부하-1	부하-2	부하-3
[9]	85.1%	95.1%	100%	100%
[10]	84.8%	95.2%	94.8%	94.5%
제안 기법	85.1%	94.1%	94.7%	95.0%

[9]의 방법에서는 고정된 전송량을 가지고 있기 때문에 높은 부하 상황에서 안전 한계를 넘는 것을 관찰하였다. 제안된 기법이 델타 압축을 수행함에도 불구하고 프로세서 부하가 낮은 이유는 SUT가 CAN 전송 작업을 수행할 때 프로세서가 CAN 의 전송 완료를 기다리기 때문에 프로세서 부하를 차지한다. 따라서 실제 전송 횟수가 낮은 제안된 기법이 전체적인 프로세서 부하를 적게 발생시키기 때문이다.

표 5에서는 전송 작업시의 채널 부하를 나타내고 있다. 제안된 기법의 평균 채널 부하는 기존 기법들에 비해 훨씬 우수한 결과를 보여주고 있다. 특히 압축을 통한 적은 채널 부하로 인해 높은 부하상황에서도 높은 데이터 전송율을 보여준다. 그에 비해 기존 기법은 높은 전송 횟수를 필요로 하기 때문에 채널 부하가 높은 것을 확인할 수 있다. 이를 통해 제안된 기법이 실시간 전송 기법인 데이터 캐스케이딩 기법이 프로세서 및 채널 부하가 높은 상황에서도 제안된 기법을 통해 최소한의 부하로 데이터를 획득할 수 있다는 것을 보여주었다.

표 5 평균 채널 부하

	유휴	부하-1	부하-2	부하-3
[9]	28.60%	28.60%	28.60%	28.60%
[10]	28.60%	24.00%	19.40%	14.70%
제안 기법	7.20%	7.20%	7.24%	7.27%

6. 결 론

우리는 차량 소프트웨어의 HiL 테스트의 결함 지역화를 돕기 위해 제한된 차량 전장 네트워크의 대역폭에서 대용량 메모리 정보를 전송하는 데이터 캐스케이딩 기법의 개선 방법을 제안하였다. 이 기법은 기존에 제안된 프로세서 부하에 가변적인 전송 에이전트와 반복적으로 전송되는 메모리 정보를 압축하는 델타 압축으로 구성하였다. 제안한 기법은 일련의 검증을 통해 기존 방법과 비교시 부하 상황에서 낮은 프로세서 부하와 평균 채널 부하를 달성하였으며, 이를 통해 시스템에 결함 발생시 높은 부하 상황에서의 결함 지역화에 이용할 수 있었다. 우리는 제안된 기법을 이용하여 차량용 소프트웨어의 HiL 테스트 중 메모리 업데이트 정보를 이용하여 결함 위치를 추정하는 연구에 사용될 예정이다.

참고문헌

[1] J. Mössinger, “Software in Automotive Systems,” *IEEE Softw.*, vol. 27, no. 2, p. 92, Jan. 2010.

[2] B. Halvorson, “Software Now To Blame For 15 Percent Of Car Recalls,” *Popular Science*, 2016.

[3] ISO 26262-6:2011, *Road vehicles -- Functional safety -- Part 6: Product development at the software level*. 2011.

[4] J. Schroeder, C. Berger, and T. Herpel, “Challenges from Integration Testing using Interconnected Hardware-in-the-Loop Test Rigs at an Automotive OEM: An Industrial Experience Report,” *Proc. First Int. Work. Automot. Softw. Archit. - WASA ’15*, Jan. 2015.

[5] K. Choi, J. Luo, K. Pattipati, S. Namburu, L. Qiao, and S. Chigusa, “Data Reduction Techniques for Intelligent Fault Diagnosis in Automotive Systems,” *2006 IEEE Autotestcon*, Jan. 2006.

[6] T. S. Khawaja, G. Georgoulas, and G. Vachtsevanos, “An efficient novelty detector for online fault diagnosis based on least squares support vector machines,” *IEEE Autotestcon 2008, Sept. 8, 2008 - Sept. 11, 2008*, no. September, pp. 202–207, 2008.

[7] H. Eichelberger, T. Kropf, J. Ruf, T. Greiner, and W. Rosenstiel, “Efficient Fault Localization During Replay of Embedded Software,” *2015 41st Euromicro Conf. Softw. Eng. Adv. Appl.*, pp. 43–52, 2015.

[8] K.-Y. Choi, J. Seo, S. Y. Jang, and J.-W. Lee, “HiL Test based Fault Localization Method using Memory Update Frequency,” in *Advances in Computer Science and Ubiquotous Computing*, vol. 373, Springer Singapore, 2015, pp. 765–772.

[9] J.-W. Lee, K.-Y. Choi, S. Y. Jang, and J.-W. Lee, “Data Cascading Method for the Large Automotive Data Acquisition Beyond the CAN Bandwidth in HiL Testing,” in *Advances in Computer Science and Ubiquotous Computing*, Springer Singapore, 2015, pp. 773–780.

[10] J.-H. Shin, K.-Y. Choi, J.-W. Lee, and J.-W. Lee, “Improvement of Large Data Acquisition Method without the Interference on the CPU Load for Automotive Software Testing,” in *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2016, pp. 223–230.

[11] C. Shin, “A framework for fragmenting /reconstituting data frame in Controller Area Network (CAN),” in *International Conference on Advanced Communication Technology, ICACT*, 2014, pp. 1261–1264.

[12] T. Ziermann, S. Wildermann, and J. Teich,

"CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates.," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, 2009.

[13] R. Miucic, S. M. Mahmud, and Z. Popovic, "An enhanced data-reduction algorithm for event-triggered networks," in *IEEE Transactions on Vehicular Technology*, 2009, vol. 58, no. 6, pp. 2663–2678.

[14] S. Kelkar and R. Kamal, "Boundary of Fifteen Compression Algorithm for Controller Area Network Based Automotive Applications," in *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA)*, 2014, pp. 162–167.

[15] S. Kelkar and Rajkamal, "Control area network based quotient remainder compression-algorithm for automotive applications," *IECON Proc. (Industrial Electron. Conf.*, pp. 3030–3036, 2012.

[16] Y. Wu, J.-G. Chung, and M. H. Sunwoo, "Design and implementation of CAN data compression algorithm," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 2014.

[17] Bosch, "CAN Specification Version 2.0," p. 72, 1991.

SW 테스트 수준 측정 기반의 테스트 중심 품질관리 프레임워크의 실무 활용 효과에 대한 연구

Study on practical use effect of test-centered Quality Management Framework based on SW test level measurement

요 약

많은 지역 IT기업이 정부가 발주하는 지역SW융합 과제에 참여하여 SW제품을 개발하고 있으나 대부분 테스트 수준이 높지 않다. 최근 정보통신산업진흥원은 표준을 기반으로 하는 SW 테스트 중심의 품질관리 프레임워크를 개발하고, 이를 정부 발주 과제를 수행하는 기업에 적용하여 과제를 수행하는 기업의 테스트 수준이 대폭 향상되는 효과를 보고 있다. 즉, TMMi 모델의 레벨 2를 기준으로 품질관리 프레임워크 적용 전보다 적용 후의 테스트 수준이 약 27점이 향상되었다. 본 연구에서는 TMMi 모델을 사용하여 품질관리프레임워크로 품질 지원을 받고 과제를 수행한 지역 IT기업의 테스트 수준이 얼마만큼 향상되었는가를 분석하였다. 또한, 수도권을 중심으로 한 대기업 중심의 일반 IT 기업과도 비교해 보니 품질관리 프레임워크로 품질 지원을 받은 지역 IT 기업의 테스트 수준이 일반 IT 기업의 테스트 수준보다 12점이 더 높아서 이를 통해 품질관리프레임워크의 실무 활용 효과성을 확인하였다.

1. 서 론

국내 많은 기업들이 TMMi, CMMi, SPICE 등의 국제 표준 모델을 활용하여 기업의 품질관리 또는 테스트 역량 향상을 위한 내부 노력 및 외부 컨설팅을 받고 있으나, 대개 1억 이상의 높은 컨설팅 비용이 들고 역량 개선을 위한 매우 많은 노력이 필요하며, 실제적인 내재화에 중점을 두기 보다는 인증 취득 자체에 치중하는 경향이 커서 실제 품질관리 수준 향상에 제대로 효과를 본 기업이 많지 않은 실정이다. 특히 재정 및 인력 구조가 취약한 소규모의 지역 IT 기업들은 높은 수준의 품질 컨설팅을 받기에 비용과 역량 측면에서 더욱 적합하지 못하다.

정보통신산업진흥원(이하 NIPA)은 2012년부터 지역 IT 기업에 적합한 품질관리 프레임워크를 개발하고 이를 NIPA 발주 과제인 지역SW융합과제에 적용하여 품질 지원을 진행하는 과제를 수행해 왔다. 품질관리 프레임워크는 2012년에 개발된 후 현재까지 지역SW융합 과제에 적용되면서 지속적으로 개선되고 있다. 지역SW융합 과제의 품질 지원은 해당 과제 사업비의 일부 예산으로 진행되고 있는데, 이는 TMMi 또는 CMMi 컨설팅(인증 포함) 비용의 약 10% 수준에 해당하는 적은 비용이다.

본 연구에서는 조직의 테스트 수준 측정 표준인 TMMi 모델을 사용하여 품질관리 프레임워크에 의해 품질 품질 지원을 받은 28개 기업들의 테스트 수준을 품질 지원 이전과 이후로 구분하여 측정하고 비교 분석하였다. 또한, 수도권에 위치한 대기업 중심의 39개 일반 IT 기업에 대해서도 동일한 TMMi 모델을 사용하여 테스트 수준을 측정한 결과와 비교하였다.

2. 본 론

2.1 품질관리 프레임워크 소개

NIPA가 개발한 품질관리 프레임워크는 지역 IT 기업이 SW 테스트를 중심으로 SW 제품 개발 전반의 SW 품질 관리를 체계적으로 수행하도록 품질 지원 기관이 지역 IT 기업을 대상으로 효과적으로 품질 지원을 할 수 있도록 개발한 경량화 버전의 품질 지원 프레임워크[1]로서 ISTQB 지식체계[2], TMMi 프레임워크[3], PMBOK 가이드[4] 등을 참조하였다.

그림 1은 품질관리 프레임워크의 구조도로 SW 개발 단계를 요구수집 및 분석 단계, 제품 설계 단계, 제품 구현 단계, 품질 검증 단계, 제품 인증 단계로 구분하여 체계적인 품질 관리를 위해 과제 주관 및 참여 기업, 지역 진흥원, NIPA, 품질 지원 기관의 관계와 각 단계별 수행해야 하는 품질 활동을 나타내고 있다.

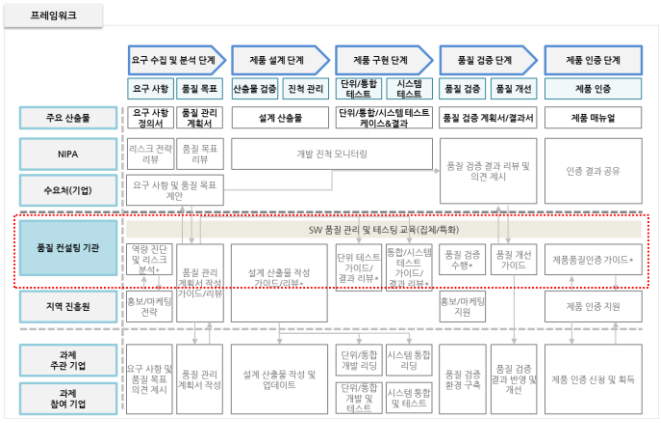


그림 1. 품질관리 프레임워크 구조도

그림 2는 품질관리 프레임워크의 세부 내용의 일부로서 요구수집 및 분석 단계의 세부 품질 활동을 나타낸다. 즉, 각 단계의 세부 활동에 대해 수행 주체, 수행 형태, 수행 목적, 세부 수행 업무, 세부 수행 내용, 관련 조직, 입력물, 출력물, 수행 방법 등을 상세히 제시하고 있다.

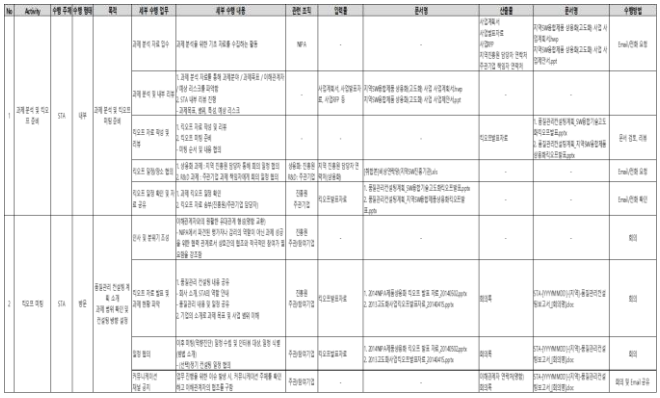


그림 2. 단계 별 세부 품질 활동

결국 품질관리 프레임워크는 별도의 품질 조직도 없고, 품질 관리 방법론도 미흡한 중소 규모의 지역 IT 기업에 테스트 중심의 SW 품질관리를 품질 지원 기관이 체계적으로 가이드할 수 있도록 개발되었고, 2012년부터 지역SW융합과제를 수행하는 지역 IT 기업 대상의 품질 지원에 적용하면서 이후 지속적으로 개선되어 왔다.

2.2 테스트 수준 측정 모델 소개

SW 테스트를 중심으로 SW 제품 개발 전반의 SW 품질 관리를 체계적으로 수행할 수 있도록 개발된 품질관리 프레임워크의 실무 활용 효과성을 분석하기 위해 조직의 테스트 수준 측정 모델인 TMMi(Test Maturity Model integration)를 사용하였다. 테스트 수준을 측정할 수 있는 모델로 TMMi 외에도 ISO/IEC 33063, TPI-NEXT 등이 있지만, 본 연구에서 측정 모델로 TMMi를 사용한 이유는 TMMi가 진단 체계를 가장 잘 갖추고 있고 테스트 수준 측정 모델로 산업계에서 널리 사용되고 있는 모델이기 때문이다.

TMMi는 그림 3과 같이 조직의 테스트 수준을 5개 레벨로 구분해 측정한다.

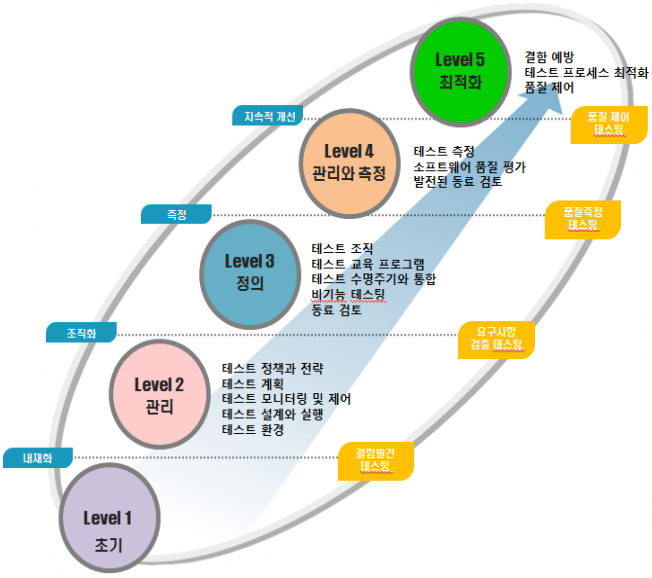


그림 3. TMMi 구조

이번 연구에서는 TMMi의 5개 레벨 중 '조직이 테스트를 주요 업무로 여기고 관리하는 수준'인 레벨 2를 사용하여 기업의 테스트 수준을 측정하였다. TMMi 레벨 2를 만족한다면 그림 4와 같이 기업에서 필요한 테스트 활동을 기본적으로 수행하고 있다고 볼 수 있다.

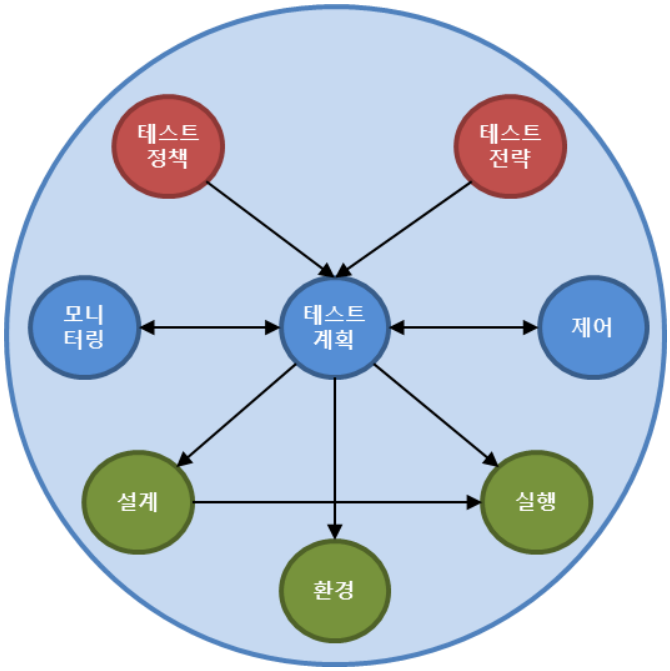


그림 4. 기업에서 필요한 테스트 활동

표 1은 TMMi 레벨 별로 결정되는 등급을 나타낸다. [5] 이 중 FA에 해당하는 85점 이상을 받아야 해당 레벨을 인정받을 수 있다.

TMMi 레벨 별 등급	설명
NA(Not Achieved)	0 ≤ 측정 결과 ≤ 15
PA(Partially Achieved)	15 < 측정 결과 ≤ 50
LA(Largely Achieved)	50 < 측정 결과 ≤ 85
FA(Fully Achieved)	85 < 측정 결과 ≤ 100

표 1. TMMi 레벨 별 등급 설명

2.3 일반 IT 기업 테스트 수준 측정 결과

39개 국내 중소기업 및 대기업인 일반 IT 기업의 통계적 특성을 분석하여 각 기업의 테스트 수준을 측정하였다.[6] 측정에 포함된 39개 기업 중 대기업이 23개(59%), 중소기업이 16개(41%)이다. 그리고, 수도권 기업이 34개(82%), 지역 기업이 5개(18%)로 수도권 기업이 다수를 차지한다.

진단 결과 85점 이상에 해당하는 TMMi 레벨 2를 만족한 기업은 없었고 전체 평균 34.8점, 대기업 평균 35.6점, 중소기업 31.3점으로 나타났다.

표 2와 같이 50점 이하를 받은 기업은 36개(92%)이고, 50점 이상을 받은 기업은 3개(8%)에 불과했다.

TMMi 레벨 2 등급	전체	비율	비고
NA	15	38%	TMMi 레벨 2 미달성
PA	21	54%	
LA	3	8%	
FA	0	0%	TMMi 레벨 2 달성
전체	39 개	100%	

표 2. 일반 IT 기업의 테스트 수준 측정 결과

2.4 지역 IT 기업 테스트 수준 측정 결과

품질관리 프레임워크를 적용한 2014~15 NIPA 지역SW융합 제품상용화 사업 28과제에 대해 과제 시작 전과 종료 후 TMMi 모델에 의해 각 테스트 수준을 측정하여 비교하였다.

표 3과 같이 품질 지원 전 각 기업 별로 측정한 결과, TMMi 레벨 2를 만족한 기업은 없었고 평균이 19.5점이었으며 28개 전체가 50점 이하인 것으로 나타났다. 이 중 15점 이하 기업이 10개(36%)를 차지했다.

TMMi 레벨 2 등급	전체	비율	비고
NA	10	36%	TMMi 레벨 2 미달성
PA	18	64%	
LA	0	0%	
FA	0	0%	TMMi 레벨 2 달성
전체	28 개	100%	

표 3. 품질 지원 전 지역 IT 기업 테스트 수준 측정 결과

표 4와 같이 품질 지원 후 다시 각 동일 기업 별로 측정한 결과, 여전히 TMMi 레벨 2를 만족한 기업은 없었고 평균이 46.8점이었으며 50점 이하 기업은 15개(54%), 50점 이상 기업은 13개(46%)인 것으로 나타났다.

TMMi 레벨 2 등급	전체	비율	비고
NA	0	0%	TMMi 레벨 2 미달성
PA	15	54%	
LA	13	46%	
FA	0	0%	TMMi 레벨 2 달성
전체	28 개	100%	

표 4. 품질 지원 후 지역 IT 기업 테스트 수준 측정 결과

2.5 품질관리 프레임워크 효과 분석

품질관리 프레임워크에 의해 품질 지원을 받은 지역 IT 기업들이 품질 지원 전의 테스트 수준 평균이 19.5점이고 품질 지원 후의 테스트 수준 평균이 46.8점으로 테스트 수준이 평균 27.3점만큼 높아졌다.

이 중 TMMi 레벨 2의 주요 테스트 활동 중 테스트 목표 수립, 리스크 분석, 테스트 실행이 평균 30점 이상 향상되어 전체 주요 테스트 활동 중 테스트 수준이 가장 향상된 테스트 활동인 것으로 분석되었다. 상승폭이 가장 낮은 주요 테스트 활동은 17점이 향상된 테스트 환경 관리로 이는 지역 IT 기업이 품질 지원 전부터 다른 주요 테스트 활동에 비해서 높은 수준의 활동을 진행하고 있었기에 상대적으로 상승폭이 낮은 것으로 분석되었다.

또한, 품질관리 프레임워크에 의해 품질 지원을 받은 지역 IT 기업의 테스트 수준이 수도권을 중심으로 한 대기업 중심의 일반 IT 기업보다 평균 12점이 더 높은 것으로 분석되어 품질관리 프레임워크의 실무 활용

효과성을 확인하였다.

3. 결 론

본 연구를 통해 품질관리 프레임워크가 지역 IT 기업의 테스트 수준 향상에 있어 실무적으로 매우 효과적임을 확인하였다.

이를 통해 지역 IT 기업과 환경이 유사한 중소규모 수준의 일반 IT기업에 대해서도 품질관리 프레임워크를 적용할 경우에도 동일한 효과를 얻을 수 있음을 예상할 수 있다.

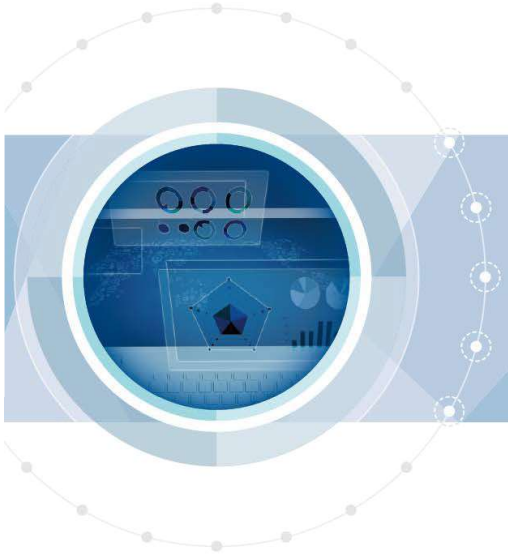
본 연구 주제와 관련하여 향후 연구할 사항은 품질관리 프레임워크의 실무 활용 효과성 상세 분석과, 자동차, 조선, 모바일, IoT 등 도메인 별 또는 기술 별로 특화된 품질관리프레임워크의 추가 개발과 그 활용 사례 연구 등이다.

[참고 문헌]

[1] NIPA, STA테스팅컨설팅, 소프트웨어 테스트 실무 가이드, 79-295, 2013년
[2] ISTQB Certified Tester Foundation Level Syllabus, 2011년
[3] Erik van Veenendaal, Test Maturity Model integration, 9-12, 23-70, 2012년
[4] A Guide to the Project Management of Knowledge(PMBOK Guide) - Fourth Edition Korea
[5] Andrew Goslin, Brian Wells, TMMi Assessment Method Application Requirements, 14-15, 2014년
[6] 윤진우, 테스트 프로세스 진단을 통한 국내 소프트웨어 테스트 성숙도 현황 연구, 52-54, 강원대학교 석사학위논문, 2014년

무기체계 SW 신뢰성 관리를 위한 프로세스 연구

A Research on the Process for reliability of weapon system software



서달미, 손근태, 서형오, 오정섭*
김태현, 박삼준**

* ㈜엔에스이 융합솔루션개발팀
** 국방과학연구소 SW신뢰성기술실

Content

Ⅰ. 연구개요

1. 연구배경

무기체계 요구 기능 변화

- 무기체계의 복잡화, 첨단화, 무인화, 자율화, 네트워크화 추세
- 무기체계 제반 기능의 상당 부분을 SW로 구현함에 따라 SW 비중/중요성 증가

SW 결함에 의한 사고

- 패트리어트 미사일 방어시스템(1991)
  - 걸프전에서 운영되던 미사일 방어시스템의 SW 버그로 미군 28명 사망
  - 수치를 누적하는 시스템 시계에서 작은 시간 오류발생으로 추적시스템 오동작 발생
- European Space Agency의 Ariane 5(1996)
  - 발사 37초 후에 자체 폭발
  - 경로 변경과정에서 numerical overflow로 인하여 on-board computer가 잘못된 고도 전달
- Some Recent Software Failures Caused by Software Bugs !!!
  - http://www.sereferences.com/software-failure-list.php

무기체계 소프트웨어의 결함 발생 시,

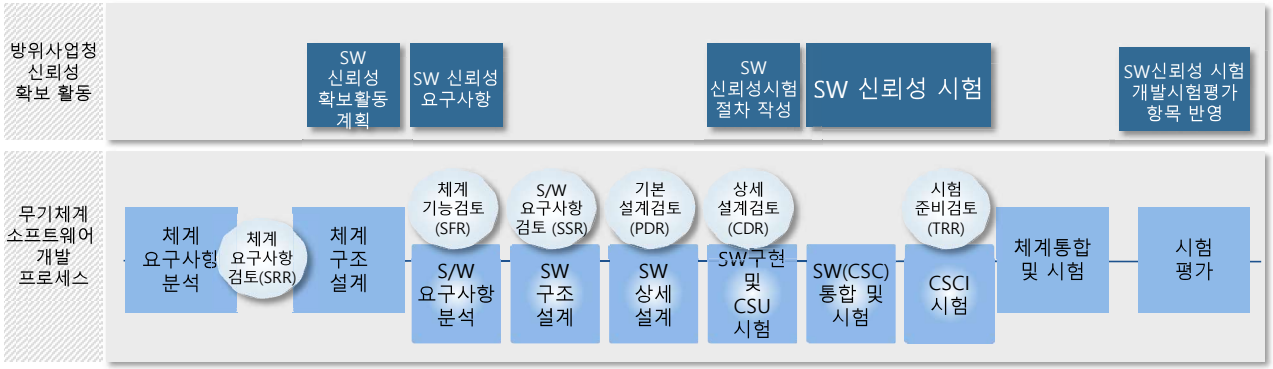
- 무기체계 파손
- 인명 및 비용 손실 등 치명적인 결과 초래

→ 소프트웨어 신뢰성 확보로 결함의 최소화

2. SW 신뢰성 확보 활동

SW 신뢰성 확보 활동 관련 제도

- 명시 규정/지침서 : 방위사업관리규정, 무기체계 SW 개발 및 관리 매뉴얼
- SW 신뢰성 확보 활동 : 무기체계 소프트웨어가 정확하고 일관성 있게 수행될 수 있도록 소스코드 내의 잠재적 결함을 최소화시키는 활동



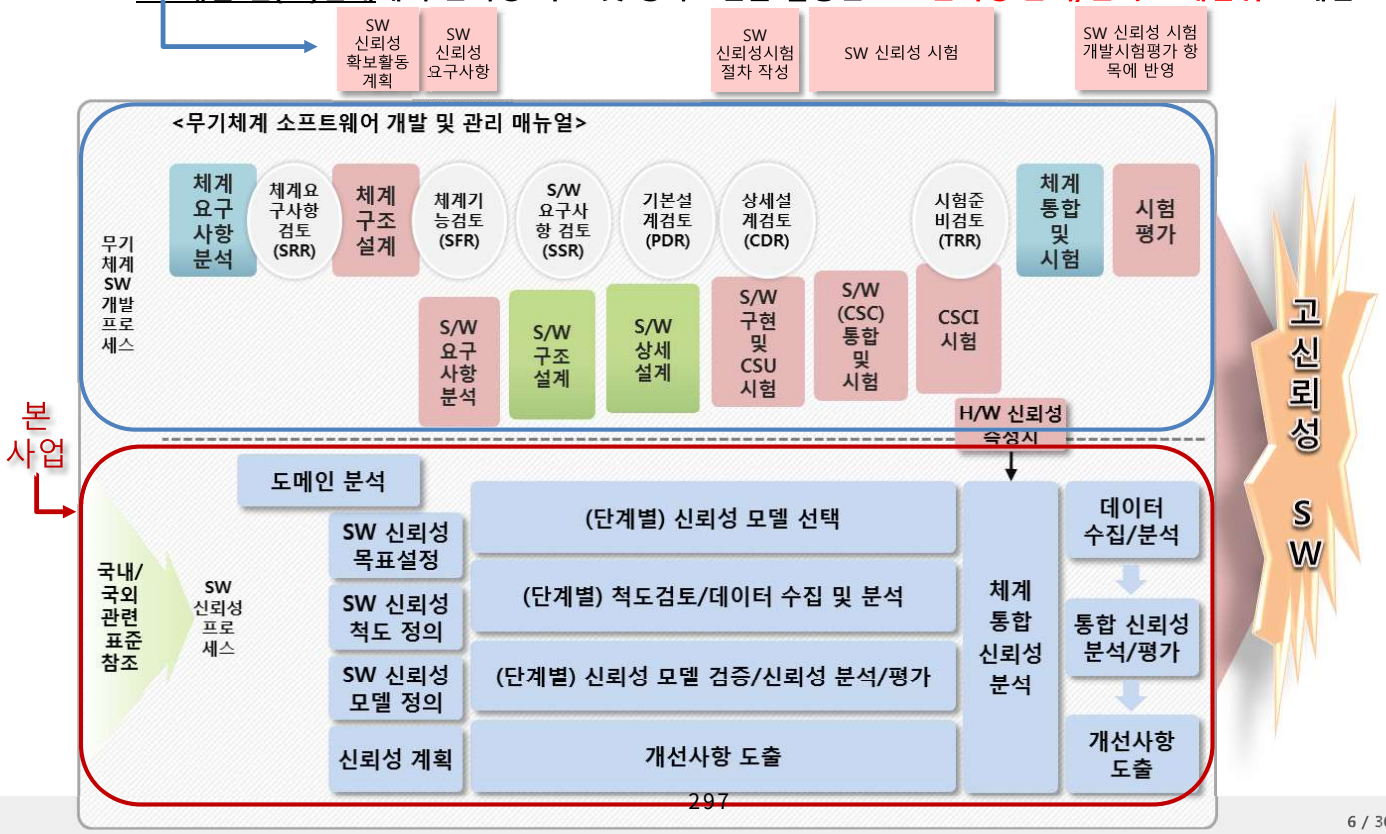
- SW 신뢰성 시험 : 소스코드에 대한 시험, 시험대상/기준 등 IPT 협의 결정
  - 코딩규칙
  - 실행시간 오류
  - 코드실행률

구 분	내 용
정적시험 (Static)	SW의 소스코드 결함을 검출하는 코딩규칙 검증, 실행시간오류(Runtime Error) 검출 시험 등
동적시험 (Dynamic)	SW의 코드실행률(Code Coverage) 확인 시험

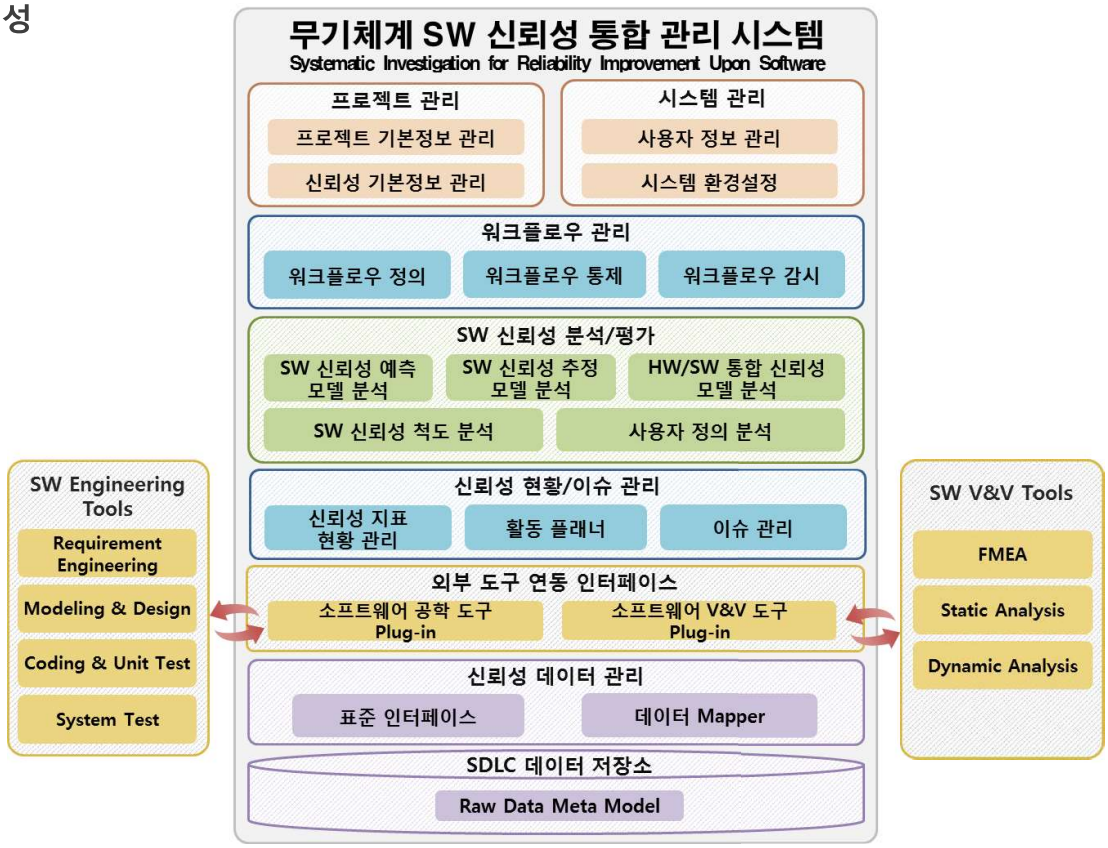
3. 사업개요

사업개념

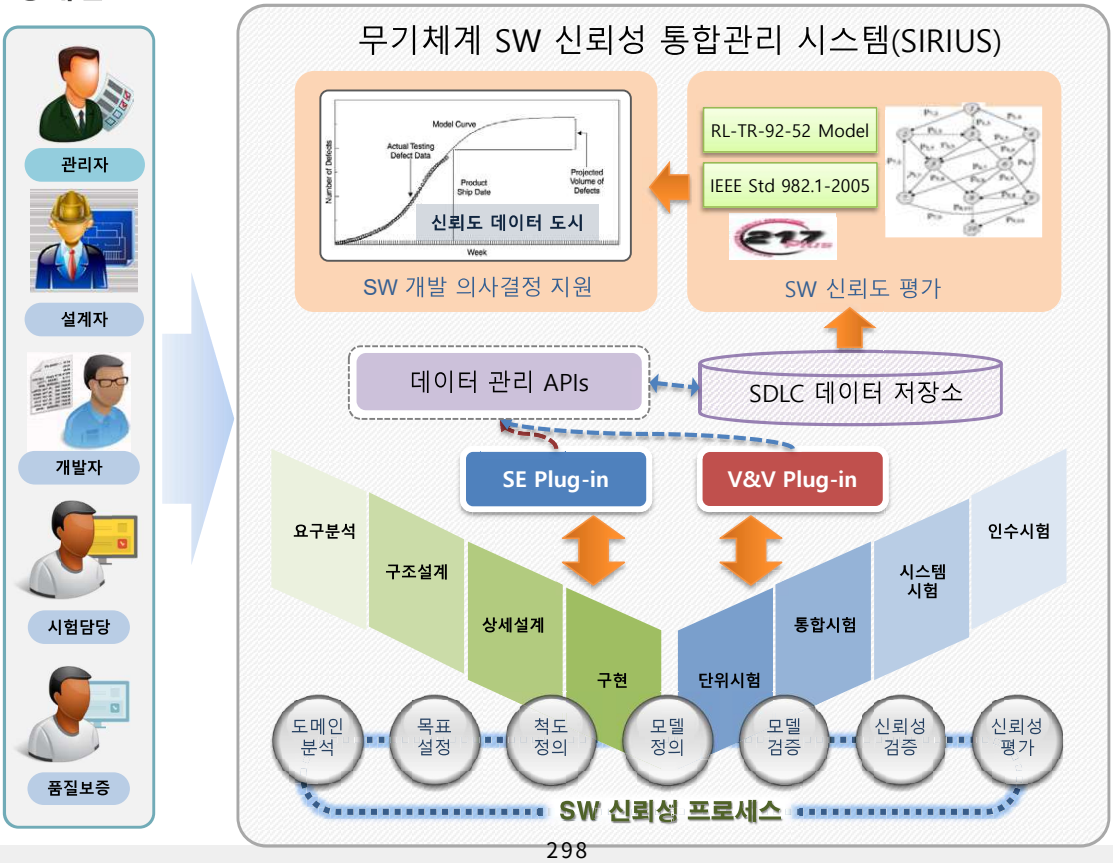
- SW개발 전(全)단계에서 신뢰성 척도 및 평가모델을 활용한 SW 신뢰성 분석/관리 프레임워크 개발



구성


무기체계 SW 신뢰성 통합 관리 시스템
Systematic Investigation for Reliability Improvement Upon Software
프로젝트 관리
프로젝트 기본정보 관리
신뢰성 기본정보 관리
시스템 관리
사용자 정보 관리
시스템 환경설정
워크플로우 관리
워크플로우 정의
워크플로우 통제
워크플로우 감시
SW 신뢰성 분석/평가
SW 신뢰성 예측 모델 분석
SW 신뢰성 추정 모델 분석
HW/SW 통합 신뢰성 모델 분석
SW 신뢰성 척도 분석
사용자 정의 분석
신뢰성 현황/이슈 관리
신뢰성 지표 현황 관리
활동 플래너
이슈 관리
외부 도구 연동 인터페이스
소프트웨어 공학 도구 Plug-in
소프트웨어 V&V 도구 Plug-in
신뢰성 데이터 관리
표준 인터페이스
데이터 Mapper
SDLC 데이터 저장소
Raw Data Meta Model
SW Engineering Tools
Requirement Engineering
Modeling & Design
Coding & Unit Test
System Test
SW V&V Tools
FMEA
Static Analysis
Dynamic Analysis

운영개념


관리자
설계자
개발자
시험담당
품질보증
무기체계 SW 신뢰성 통합관리 시스템(SIRIUS)
신뢰도 데이터 도시
SW 개발 의사결정 지원
RL-TR-92-52 Model
IEEE Std 982.1-2005
SW 신뢰도 평가
데이터 관리 APIs
SDLC 데이터 저장소
SE Plug-in
V&V Plug-in
요구분석
구조설계
상세설계
구현
단위시험
통합시험
시스템 시험
인수시험
도메인 분석
목표 설정
척도 정의
모델 정의
모델 검증
신뢰성 검증
신뢰성 평가
SW 신뢰성 프로세스

Ⅱ. IEEE Std. 1633-2008의 신뢰성 프로세스

1. 신뢰성 프로세스

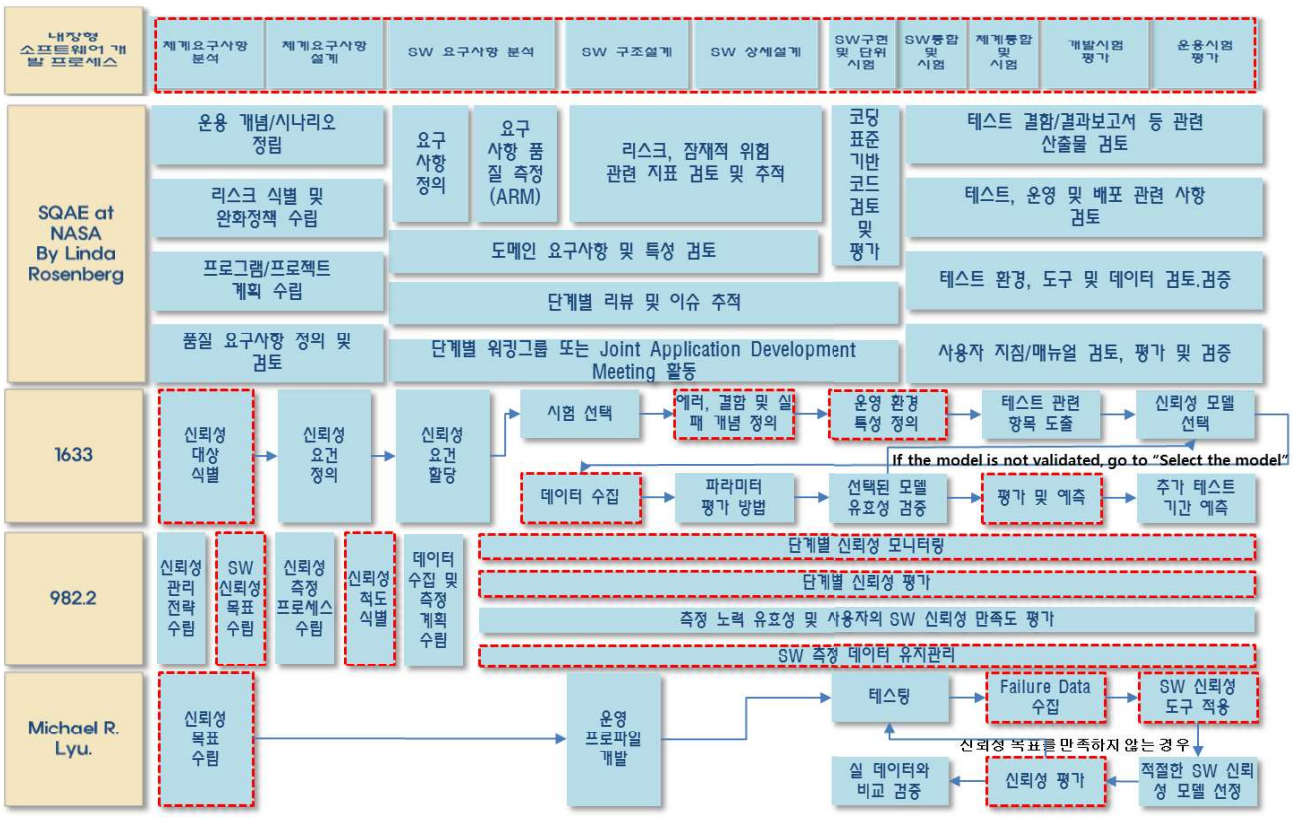
2. SIRIUS Process

1. IEEE Recommended Practice on Software Reliability

No.	Practice	Description
1	Identify the Application	▪ 소프트웨어 신뢰성 측정 대상 소프트웨어 식별
2	Specify the Reliability Requirement	▪ 소프트웨어 신뢰성에 대한 명세서/목표를 기술
3	Allocate the Reliability Requirement	▪ 소프트웨어에 대한 신뢰성 요구사항 할당
4	Make a Reliability Risk Assessment	▪ 요구사항과 요구사항 변경에 따른 SW 결함이나 에러에 대한 위험 기반으로 수행 ▪ 요구사항의 수 혹은 변경에 의한 영향도를 기반의 위험 규정 방법이 정확하지 않으나, 큰 규모 시스템의 설계 분석 시 반드시 필요 ▪ 위험 분석 방법은 남아있는 결함의 위험과 다음 결함이 일어날 시간에 대한 위험을 계산하는 방식으로 수행
5	Define Errors, Faults, and Failures	▪ 에러(Error), 결함(Fault), 실패(Failure)에 대한 정의를 프로젝트의 특성에 맞게 정의 ▪ 프로젝트 전반에 걸쳐 일관성 있게 정의
6	Characterize the Operational Environment	▪ 3가지 측면에 기반한 운영환경 특성 기술(시스템 형상, 시스템 업그레이드, 시스템 운영 프로파일)
7	Select Tests	▪ 실 시스템에 적용할 테스트와 test case를 선택

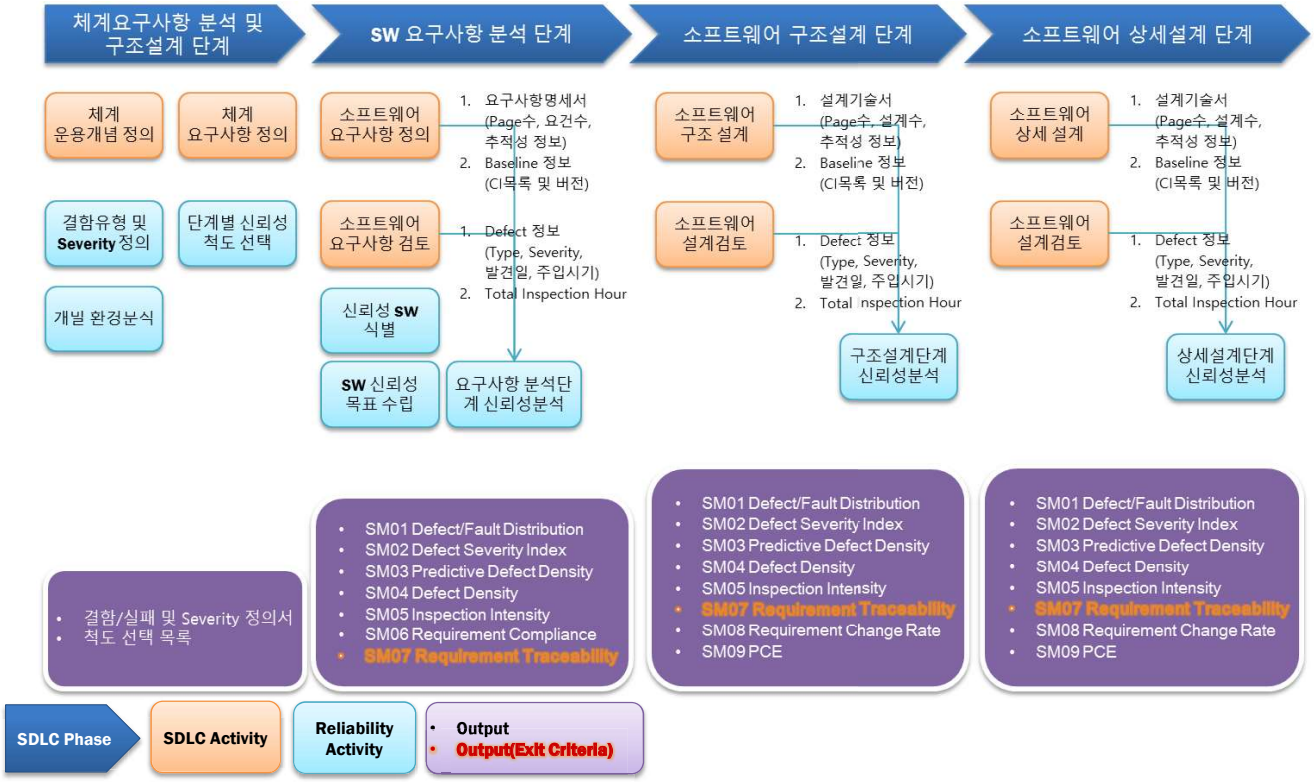
No.	Practice	Description
8	Select Models	▪ 한 개 이상의 신뢰성 모델 선택 및 적용
9	Collect Data	▪ 데이터 수집 목적을 명확하게 정의 ▪ 적합한 데이터 수집
10	Estimate Parameters	▪ 데이터 평가/분석 방법 선택 ▪ Maximum likelihood estimation (MLE), least squares estimation 등 방법 사용
11	Validate the Model	▪ 선택된 신뢰성 모델의 유효성 검증, 수집 데이터와 파라미터 평가 방법에 대한 검증
12	Perform Assessment and Prediction Analysis	▪ 소프트웨어 신뢰성, 남아 있는 코드 결함 수 등 분석 수행
13	Forecast Additional Test Duration	▪ 목표된 초기 결함과 모델의 모수가 정해지면 추가적인 테스트 기간 예측

관련 연구



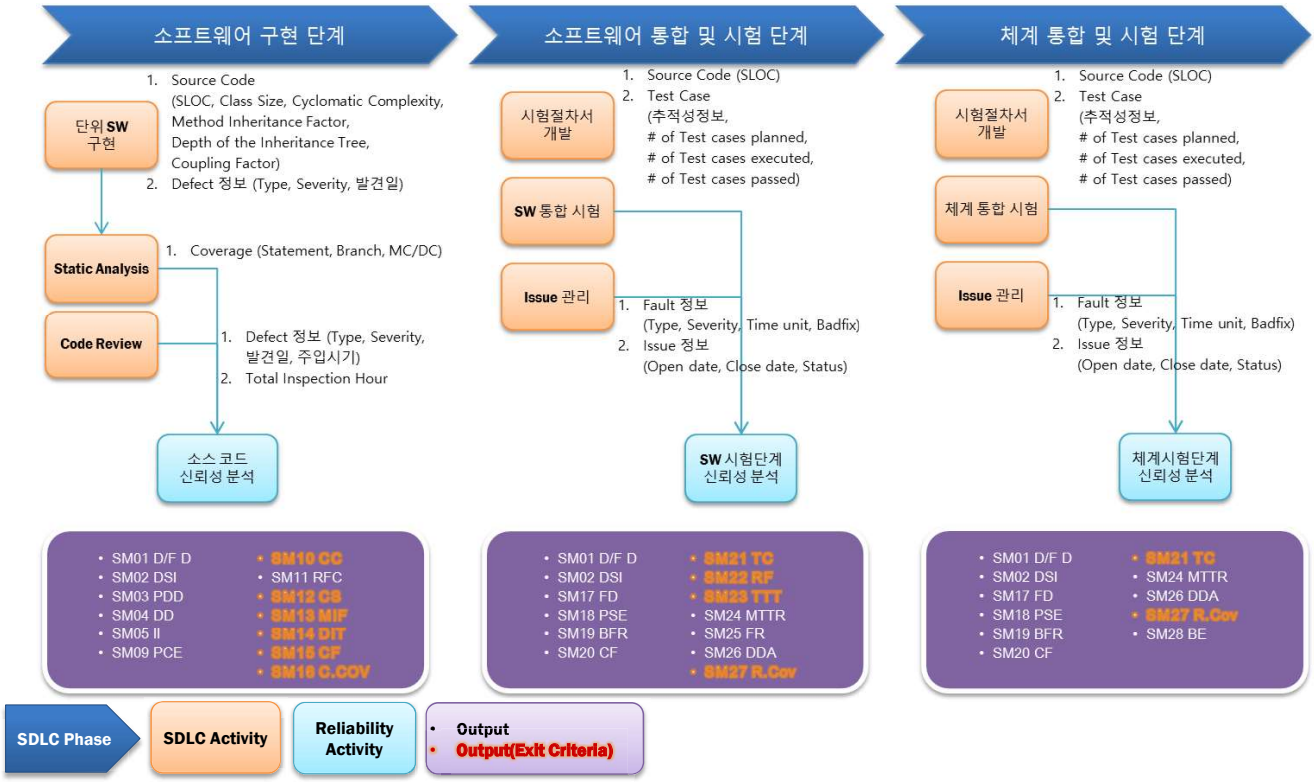
SIRIUS Process

CDR 이전 프로세스



SIRIUS Process

CDR 이후 프로세스



Ⅲ. IEEE Std. 1633-2016 신뢰성 프로세스

1. 신뢰성 프로세스

1. IEEE Approved Draft Recommended Practice on Software Reliability

Activity	세부 Activity			Type	Input	Output
Planning for software reliability	Characterize The Software System	Identify the SW LRUs in the system		E	List of SW and FW LRUs; Customer, user, function mode and functional profile	List of SW and FW LRUS that are applicable for SRE; Bill of Material, operational profile; Impact on system design by software
		Construct a SW BOM				
		Characterize the Operational Environment				
		Identify the impact of the SW design on the SW and system design				
	Define Failures And Criticality			E	System requirements, Past history concerning Effects of software failures	A Failure Definitions and Scoring Criteria(FDSC)
	Perform A Reliability Risk Assessment	Assess safety risks		T	Known or suspected risks regarding the system or software	A software reliability risk assessment including risks related to safety, security, product maturity, risk of underestimating size and overestimating reliability growth, potential for defect pileup into future release
		Assess security and vulnerability risks				
		Assess product maturity				
		Assess project and schedule risks				
		Assess whether there are too many risks for on SW release				
	Assessment The Data Collection System			T	Data requirements and capability to collect that data	A list of modifications to current systems or processes
	Review Available Tools Needed For Software Reliability			T	A list of SRE related tools	
	Develop A Software Reliability Plan (SRPP)			E	5.1.1~5.1.5 Output	

Activity	세부 Activity			Type	Input	Output
Develop failure modes model	Perform Software Defect Root Cause Analysis(RCA)			P	Defects from a prior release(internal or external) or defects from this release If operational data is available, the results of section 5.6.3	Most common root causes of defects from prior software release
	Perform Software Failure Modes Effects Analysis (SFMEA)	Prepare the SFMEA		P	5.2.1. output Software requirements, architectural design, detailed design, use cases, installation scripts, users manuals, vulnerabilities If available, the results of tasks 5.2.1 If available, a preliminary hazards analysis or the results of task 5.2.3 5.2.3 output	SFMEA Report List of software related failure modes Critical Items List(CIL)
		Analyze Failure Modes and Root Causes				
		Identify Consequences				
		Mitigate				
		Generate a Critical Items List (CIL)				
		Understand the Differences Between a Hardware and Software FMEA				
	Include Software In The System Fault Tree Analysis(FTA)			P	System fault tree analysis documentation If available, the results of task 5.2.2 5.2.2 output	System fault tree analysis with software included A list of software related hazards

1. IEEE Approved Draft Recommended Practice on Software Reliability
Ⅲ. IEEE Std. 1633-2016 신뢰성 프로세스

Activity	세부 Activity			Type	Input	Output
Apply software reliability during development	Identify/Obtain The Initial System Reliability Objective			E	List of known risks from section 5.1.3 System reqruiements Recent similar system reliability measurements experienced during operation. Estimated change in functionality between new system and a predecessor If avilable, operational data from a previous release as per section 5.6 5.3.6 Plan the reliability growth needed to achieve system prediction and objective	SR objectives 5.3.5 Determine the required total software reliability to meet the system objective
	Perform A Software Reliability Assessment And Prediction	Collect data about Product and Project		E	Characteristics about the development process, the software, the Software Development Plan, size, organization, capabilities, personnel experience, known risks from section 5.1.3. If available results from production SFMEA section 5.2.2 5.3.3 Sanity check the Early prediction Viability of prediction	SR predictions for each LRU including COTs, GFS, FOSS
		Select a model to predict software reliability early in development				
		Use the selected software reliability prediction model	Predict total defects			
			Predict the defect distribution			
			Predict failure rate and MTBF			
			Predict Reliability			
			Predict Availability			
		Apply software reliability models with incremental development				
		Use the assessment to qualify a subcontractor, cots, or FOSS vendor				

Activity	세부 Activity			Type	Input	Output
Apply software reliability during development	Sanity Check The Prediction			T	5.3.2 Perform software reliability assessment and prediction	Viability of prediction 5.3.2 Perform software reliability assessment and prediction
	Merge The Predictions Into The Overall System Predictions			E	HW Predictions, System RBD	
	Determine An Appropriate Overall Software Reliability Requirement	Establish software MTBF/MTBCF objective		E	5.3.4 Merge the predictions into the overall system predictions	Top level SW prediction 5.3.8 Allocate the required software reliability to all software LRUs 5.3.6 Plan the reliability growth needed to achieve system prediction and objective
		Establish a software reliability objective				
		Establish an availability objective				
		Verify that the software objective is feasible				
	Plan The Reliability Growth			E	Avilable schedule/resources 5.3.2 Perform software reliability assessment and prediction	Reliability growth needed to reach objective
	Perform A Sensitivity Analysis	Perform a sensitivity analysis of each software LRU		T	5.3.8 Allocate the required software reliability to all software LRU 5.3.6 Plan the reliability growth needed to achieve system prediction and objective 5.3.2 Perform software reliability assessment and prediction	List of practices that improve prediction with least risk/cost
		Perform a sensitivity analysis of the software LRUs effect on the system reliability				
	Allocate The Required Reliability To The Software LRUs	top down allocation		T	5.3.5 Determine the required total software reliability to meet the system objective	SW LRU Allocations, predictions, top level SW prediction
		bottom up allocation				
	Employ Software Reliability Metrics For Transition To Testing	requirements traceability		E		Test coverage, Requirements coverage
		structural coverage				

Activity	세부 Activity			Type	Input	Output
Apply software reliability during testing	Develop A Reliability Test Suite	operational profile testing		T	Operational Profile from step 5.1.1. Effects on system design as per step 5.1.1.4. Results of failure modes analyses section 5.2	5.4.3 measure test coverage
		requirements testing				
		model based testing				
		model based testing				
		timing and performance testing				
		failure mode testing				
	Increase Test Effectiveness Via Software Fault Insertion			E	Software Failure modes from 5.2.2 White box test results from step 5.3.9.2	
	Measure Test Coverage			P	Requirements traceability results 5.3.9.1 SFMEA report from 5.2.2 Ranked list of most common failure modes 5.4.8 Revisit the defect RCA	Percentage of lines of code, branches, requirements covered
	Collect Fault And Failure Data			E	Date/time of observed Software failures during testing, testing and usage time per day	Failure rate trends, estimated remaining defects. Failures scored via FDSC from 5.1.2
	Select Reliability Growth Models	fault rate		T	Available SRE tools from 5.1.5 5.4.4 collect failure and defect data	Current and future estimated MTBF, availability, reliability
		linear versus non linear decreasing				
		inherent defect content				
		effort required to use the model				
		model data formats				
		apply software reliability growth models with an incremental or evolutionary life cycle model				
	Apply Software Reliability Metrics			E		Percentage of defects that require multiple corrective actions, fix rates, fault days number, backlog, testing defect density

Activity	세부 Activity			Type	Input	Output
Apply software reliability during testing	Determine The Accuracy Of The Prediction And Reliability Growth Models	determine the accuracy of the prediction models used early in development		E	SR predictions, from 5.3.2, 5.3.4 Current and future estimated MTBF, availability, reliability 5.4.5 5.4.4 Required software reliability objective from step 5.3.1	Relative accuracy of estimations and predictions
		determine the accuracy of the software reliability growth model				
	Revisit The Defect Root Cause Analysis			P	Results of 5.2.1 Defect RCA from prior similar systems 5.4.4 Collect failure and defect data	Ranked list of most common failure modes Available SRE tools from 5.1.5

Activity	세부 Activity			Type	Input	Output
Support release decision	Determine Release Stability			E	Software reliability growth model results from section 5.4.5 Required software reliability objective from step 5.3.1	Whether or not reliability objective is met
	Forecast Additional Test Duration			T	Software reliability growth model results from section 5.4.5	Number test hours needed to reach objective
	Forecast Remaining Defects And Effort Required To Correct Them			E	Software reliability growth model results from section 5.4.5	Whether there is sufficient maintenance staff to correct the remaining defects if deployed now
	Perform A Reliability Demonstration Test			T	Observed SW Failures during the RDT	Accept/reject of reliability objective based on a statistical sample
Apply software reliability in operation	Employ Software Reliability Metrics To Monitor Operational Software Reliability			E	Failure discovered in operation(date of discovery and severity) Number of installed sites Estimated duty cycle of all installed sites	Actual operational defect density Immediate customer found defects Number backlogged defects Incoming defect rate Fix rate(for operational defects) SW Adoption rate SWDPMH Actual failure rate, MTBF, MTBCF, availability, reliability

Activity	세부 Activity			Type	Input	Output
Apply software reliability in operation	Compare Operational Reliability To Predicted Reliability			E	Predictions from step 5.3.2 SWR growth estimations from step 5.4.5 5.6.1	Relative accuracy of models
	Assess Changes To Previous Characterizations Or Analyses			T	Relative accuracy of models 5.6.2	5.1.2 Define failures and criticality 5.1.3 Risk Assessment 5.1.6 Updated the SRE plan 5.2.1 Perform Defect RCS 5.2.3 Put software on system FTA 5.3.1 Determine/obtain system reliability objective 5.3.3 Sanity check the early predictions 5.4.1 Develop a Reliability Test Suite 5.4.7 Determine the accuracy of the predictive and estimation models 5.5.4 Forecast defect pileup
	Archive Operational Data			T	5.6.3	

Ⅳ. IEEE Std. 1633 (2008 vs 2016)

1. 버전별 비교

1. 버전별 비교

IEEE Std. 1633 - 2008	IEEE Std. 1633 - 2016	
1. Identify the Application 6. Characterize the Operational Environment	Planning for software reliability	Characterize The Software System
5. Define Errors, Faults, and Failures		Define Failures And Criticality
4. Make a Reliability Risk Assessment		Perform A Reliability Risk Assessment
		Assessment The Data Collection System
		Review Available Tools Needed For Software Reliability
		Develop A Software Reliability Plan (SRPP)
	Develop failure modes model	Perform Software Defect Root Cause Analysis(RCA)
		Perform Software Failure Modes Effects Analysis (SFMEA)
		Include Software In The System Fault Tree Analysis(FTA)
	Apply software reliability during development	Identify/Obtain The Initial System Reliability Objective
		Perform A Software Reliability Assessment And Prediction
		Sanity Check The Prediction
		Merge The Predictions Into The Overall System Predictions
2. Specify the Reliability Requirement		Determine An Appropriate Overall Software Reliability Requirement
		Plan The Reliability Growth
		Perform A Sensitivity Analysis
3. Allocate the Reliability Requirement		Allocate The Required Reliability To The Software LRUs
		Employ Software Reliability Metrics For Transition To Testing

* 붉은 색 글씨 : 2016 버전에서 추가된 항목
* [ ] : SIRIUS Process에 반영된 항목

1. 버전별 비교

IV. IEEE Std. 1633 (2008 vs 2016)

IEEE Std. 1633 - 2008	IEEE Std. 1633 - 2016	
7. Select Tests	Apply software reliability during testing	Develop A Reliability Test Suite
		Increase Test Effectiveness Via Software Fault Insertion
		Measure Test Coverage
9. Collect Data		Collect Fault And Failure Data
8. Select Models		Select Reliability Growth Models
10. Estimate Parameters		Apply Software Reliability Metrics
11. Validate the Model		Determine The Accuracy Of The Prediction And Reliability Growth Models
		Revisit The Defect Root Cause Analysis
13. Forecast Additional Test Duration	Support release decision	Forecast Additional Test Duration
		Forecast Remaining Defects And Effort Required To Correct Them
		Perform A Reliability Demonstration Test
	Apply software reliability in operation	Employ Software Reliability Metrics To Monitor Operational Software Reliability
		Compare Operational Reliability To Predicted Reliability
		Assess Changes To Previous Characterizations Or Analyses
		Archive Operational Data

* 붉은 색 글씨 : 2016 버전에서 추가된 항목
* [ ] : SIRIUS Process에 반영된 항목

1. 버전별 비교

IEEE Std. 1633 – 2008 대비 2016 주요 특징

- 단계별 최대 하위 4 레벨까지의 상세 Activity 기술
- Activity 의 Type – Essential, Typical and Project Specific
- Role – RE, SQA, SM and ACQ
- Activity 별 Checklist
- Activity 별 Input 및 Output
- SRE Related Tools
- Reliability Models
- Failure modes model 및 Operation 단계의 신뢰성

IEEE Std. 1633 – 2008 버전의 한계

- 테스팅에 의한 신뢰성 평가에 중점을 두고 있음
- 각 단계별 프랙티스, 산출물 등 미제시

V. SIRIUS Process Revised

1. SIRIUS Process vs IEEE Std. 1633 – 2016
2. SIRISU Process Revised

1. SIRIUS Process vs IEEE Std. 1633 - 2016

SIRIUS Process 내 개발단계

IEEE Std. 1633 -2016 단계

개발단계	단계
체계 요구사항 분석 및 구조 설계	Planning for software reliability
SW 요구사항 분석	Develop failure modes model
SW 구조설계	Apply software reliability during development
SW 상세설계	Apply software reliability during testing
SW 구현	Support release decision
SW 통합 및 시험	Apply software reliability in operation
체계 통합 및 시험	

2. SIRIUS Process Revised

SIRIUS Process Revised

개발단계	신뢰성 활동
체계 요구사항 분석 및 구조 설계	개발 환경 분석
	결함유형 및 Severity 정의
	단계별 신뢰성 척도 선택
	SFMEA 수행
SW 요구사항 분석	신뢰성 SW 식별
	SW 신뢰성 목표 수립
	요구사항 분석 단계 신뢰성 분석
SW 구조설계	구조설계 단계 신뢰성 분석
SW 상세설계	상세설계 단계 신뢰성 분석
SW 구현	구현 단계 신뢰성 분석
SW 통합 및 시험	통합 및 시험 단계 신뢰성 분석
체계 통합 및 시험	체계 통합 및 시험 단계 신뢰성 분석

IEEE Std. 1633 - 2016

단계	신뢰성 활동
Planning for software reliability	Characterize The Software System
	Define Failures And Criticality
	Perform A Reliability Risk Assessment
	Assessment The Data Collection System
	Review Available Tools Needed For Software Reliability
	Develop A Software Reliability Plan (SRPP)
Develop failure modes model	Perform Software Defect Root Cause Analysis(RCA)
	Perform Software Failure Modes Effects Analysis (SFMEA)
	Include Software In The System Fault Tree Analysis(FTA)

* 붉은 색 글씨 : 2016 버전에서 추가된 항목
* : SIRIUS Process에 반영된 항목

SIRIUS Process Revised

개발단계	신뢰성 활동
체계 요구사항 분석 및 구조 설계	개발 환경 분석
	결함유형 및 Severity 정의
	단계별 신뢰성 척도 선택
SW 요구사항 분석	신뢰성 SW 식별
	SW 신뢰성 목표 수립
	요구사항 분석 단계 신뢰성 분석
SW 구조설계	구조설계 단계 신뢰성 분석
SW 상세설계	상세설계 단계 신뢰성 분석
SW 구현	구현 단계 신뢰성 분석
SW 통합 및 시험	통합 및 시험 단계 신뢰성 분석
체계 통합 및 시험	체계 통합 및 시험 단계 신뢰성 분석

IEEE Std. 1633 - 2016

단계	신뢰성 활동
Apply software reliability during development	Identify/Obtain The Initial System Reliability Objective
	Perform A Software Reliability Assessment And Prediction
	Sanity Check The Prediction
	Merge The Predictions Into The Overall System Predictions
	Determine An Appropriate Overall Software Reliability Requirement
	Plan The Reliability Growth
	Perform A Sensitivity Analysis
	Allocate The Required Reliability To The Software LRUs
	Employ Software Reliability Metrics For Transition To Testing

* 붉은 색 글씨 : 2016 버전에서 추가된 항목
* [ ] : SIRIUS Process에 반영된 항목

2. SIRIUS Process Revised
V. SIRIUS Process Revised

SIRIUS Process Revised

개발단계	신뢰성 활동
체계 요구사항 분석 및 구조 설계	개발 환경 분석
	결함유형 및 Severity 정의
	단계별 신뢰성 척도 선택
SW 요구사항 분석	신뢰성 SW 식별
	SW 신뢰성 목표 수립
	요구사항 분석 단계 신뢰성 분석
SW 구조설계	구조설계 단계 신뢰성 분석
SW 상세설계	상세설계 단계 신뢰성 분석
SW 구현	구현 단계 신뢰성 분석
SW 통합 및 시험	통합 및 시험 단계 신뢰성 분석
체계 통합 및 시험	체계 통합 및 시험 단계 신뢰성 분석

IEEE Std. 1633 - 2016

단계	신뢰성 활동
Apply software reliability during testing	Develop A Reliability Test Suite
	Increase Test Effectiveness Via Software Fault Insertion
	Measure Test Coverage
	Collect Fault And Failure Data
	Select Reliability Growth Models
	Apply Software Reliability Metrics
	Determine The Accuracy Of The Prediction And Reliability Growth Models
	Revisit The Defect Root Cause Analysis
Support release decision	Forecast Additional Test Duration
	Forecast Remaining Defects And Effort Required To Correct Them
	Perform A Reliability Demonstration Test
Apply software reliability in operation	Employ Software Reliability Metrics To Monitor Operational Software Reliability
	Compare Operational Reliability To Predicted Reliability
	Assess Changes To Previous Characterizations Or Analyses
	Archive Operational Data

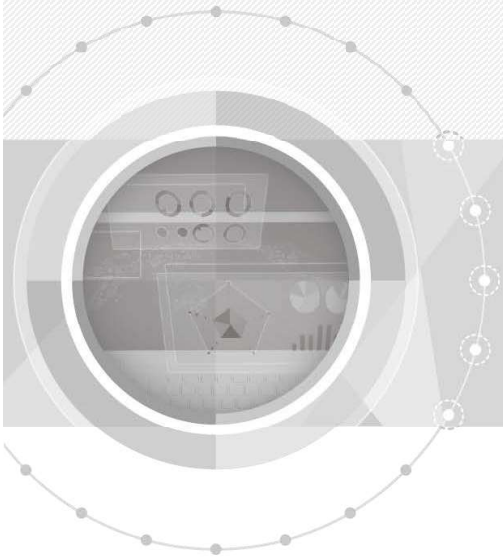
* 붉은 색 글씨 : 2016 버전에서 추가된 항목
* [ ] : SIRIUS Process에 반영된 항목

VI. 결론

무기체계 SW 신뢰성 관리를 위한 Process 고찰

- SW 신뢰성과 관련된 국제 표준에 부합하는 Process를 구축함으로써 무기체계 SW의 체계적인 개발 및 관리 가능
- SW 신뢰성 예측/추정 등 무기체계 SW 개발 특성에 적합한 SW 개발 전(全) 단계에서 지속적인 분석 및 관리가 가능한 신뢰성 Process 제시
- Software Failure Mode Model 중 일부(Software FMEA)를 반영
- Release 시점까지 신뢰성 평가를 위한 Process 제안

감사합니다



NSE TECHNOLOGY

한국형 테스트 성숙도 모델을 위한 경량화 연구

박보경*, 장우성*, 김기두**, 이근상***, 김영철*O, C. R. Carlson****

*홍익대학교 소프트웨어공학연구실

**한국정보통신기술협회, ***전북테크노파크, ****Dept. of ITM, IIT

{park*, jang*, yi***, bob*O}@selab.hongik.ac.kr, **kdkim@tta.or.kr, ****carlson@iit.edu

A Study on Lightweighting for Korean Test Maturity Model

Bo Kyung Park*, Woo Sung Jang*, Kidu Kim**, Keunsang Yi***, R. Young Chul Kim* O,

C. R. Carlson****

*SE Lab., Hongik University, **TTA, ***Jeonbuk TP, ****Dept. of ITM, IIT, USA

요 약

국내의 많은 기업들은 기존의 다양한 소프트웨어 품질 평가 모델을 도입하고 있다. 하지만, 이 모델들은 광범위하며, 국내 벤처/중소기업에 적용하려면, 컨설팅을 통한 교육, 훈련 및 풍부한 실무 경험이 뒷받침되어야 한다. 비용, 인력 면도 현실적인 제약이 크며, 준비 할 부산물의 양이 많다. 이 문제를 위해, 국내 SW 개발 환경과 조직을 고려해 테스트 성숙도 모델의 경량화 방법을 제안한다. 제안한 방법의 절차는 KTMM의 구조를 정의하고 핵심요소를 도출한다. 도출된 핵심요소 중 유사항목을 통합하여 최종 모델을 개발한다. 이 모델로 중소기업 내 테스트 조직의 성숙도를 개선하여, 소프트웨어 품질 개선할 시간과 비용 절약을 기대 한다.

1. 서 론

소프트웨어 비중이 증가함에 따라, 소프트웨어의 품질 중요도에 대한 관심이 커지고 있다. 현재 국내의 많은 기업들은 고품질의 소프트웨어를 개발하기 위해 CMMI나 TMMi 같은 소프트웨어 품질 평가 모델을 도입하고 있다. 하지만 기존의 성숙도 모델들은 적용 분야가 매우 광범위하며. 이러한 프로세스를 적용하기 위해서는 컨설팅을 통한 교육, 훈련 및 풍부한 실무 경험이 뒷받침되어야 한다. 또한 국내 기업에서 도입하기에는 비용, 인력 면에서 현실적인 제약이 큰 실정이며, 인증 평가 시, 준비해야 할 문서가 많다. 따라서 국내 중소기업이 기존에 개발된 외국의 테스트 성숙도 모델을 도입하는데 한계가 있다.

이 문제를 위해, 본 연구실에서는 국내 SW개발 환경과 조직을 고려한 경량화된 테스트 성숙도 모델(Korean Test Maturity Model, KTMM)을 제안한다[1]. 2015년 1차 TTA 한국형 테스트 성숙도 모델 프로젝트의 결과물을 TMM의 레벨 2와 3의 형식을 기반으로 국내 중소기업에 적합하게 반영하였다[2]. 전체적인 구조는 TMM을 기반으로 TMM과 TPI Next의 실무 활동 중심으로 개선되었다. 2016년도 2차 TTA 사업으로 2,3레벨 개선과 4,5 레벨 고도화 개발을 수행하였다[3]. 최종 모델의 개발 범위는 레벨 2부터 5까지이다. 한국형 모델을 개발함으로써, 국내의 중소기업 내 테스트 조직의 성숙도를 개선하여 소프트웨어 품질 개선할 시간과 비용 절약을 기대한다. 또한 KTMM은 기존의 외국에서 개발된 모델에 비해 실무활동의 경량화로, 국내 중소기업에서 보다 쉬운 활용을 기대한다.

[1] 이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548)

2. 본 론

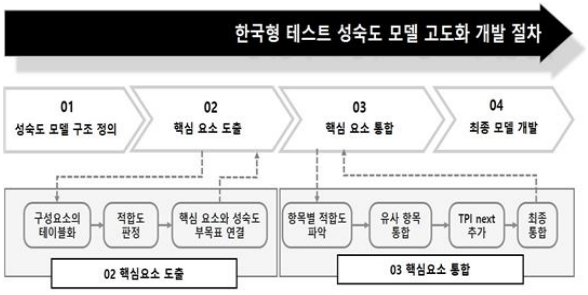


그림 1 개발 절차

본 연구는 2015/2016년 1/2차 TTA 한국형 테스트 성숙도 모델 프로젝트의 결과물을 기반으로 경량화된 테스트 성숙도 모델을 개발한다. 개발 과정은 2~5레벨의 구조 정의, 핵심요소 도출 및 최종 모델 개발이다. 그림 1은 KTMM의 개발 절차이다.

2.1 테스트 성숙도 모델 구조 정의

KTMM은 TMM을 국내 중소기업들의 실무 현황에 맞춰 개선한 테스트 성숙도 모델이다. 5레벨의 성숙도 수준으로 표현하고, 각 성숙도 레벨마다 달성해야 하는

목표가 있다. 그림 1과 같이, 성숙도 목표는 성숙도 부목표들에 의해 지원되며, 부목표들은 활동/작업/책임(핵심요소)을 수행함으로써 달성된다.

2.2 핵심요소 도출

KTMM의 테스트 성숙도 요소 추출 과정은 1)구성요소 테이블화, 2)적합도 판정, 3)핵심요소와 성숙도 부목표 연결로 구성된다. 구성요소 테이블화 단계에서는 서술형으로 정의된 핵심요소를 테이블 형태로 변환한다. 변환 과정은 TMM 레벨의 테이블화, 각 레벨에 대한 성숙도 목표 추가, 각각의 성숙도 목표에 대한 성숙도 부목표 추가, 성숙도 부목표에 대한 핵심요소 추가로 진행된다. 핵심요소의 적합도 판정 단계에서는 테스트를 위해 필요한 요소를 최소화 한다. TMM의 세 가지 관점(관리자, 개발자/테스터, 사용자/ 클라이언트)의 모든 요소들을 식별하여, KTMM에 적합한지 판정한다. 적합도 판정 기준은 “적합”, “부적합”, “부분적합”으로 구성하였다. 적합은 KTMM에 적용 가능한 요소를 말하여, 부적합은 적용 불가능한 요소이다. 부분적합은 부분적으로 적용 가능한 경우이다. 기존 TMM 성숙도 부목표와 핵심요소 연결에서는 성숙도 부목표에 관리자, 개발자/테스터의 핵심 관점 활동을 연결한다. 그리고 각 관점을 하나의 관점으로 통합한다.

2.3 핵심관점 통합

TMM 평가 항목들은 관리자, 개발자/테스터, 유저/클라이언트 관점에서 수행해야 할 활동들을 제시하였다. 이는 관점의 차이만 있을 뿐, 수행활동이 비슷한 경우가 많다. 따라서 유사 항목들을 분석하여 하나의 항목으로 통합한다. 핵심 관점의 통합 절차는 유사항목 통합, TPI Next 추가, 최종 통합이다. 유사항목 통합 단계에서는 핵심 관점 항목들 중 유사한 항목을 분석한다. 그림 2는 성숙도 부목표 4.1에 대한 유사항목 통합 결과이다.

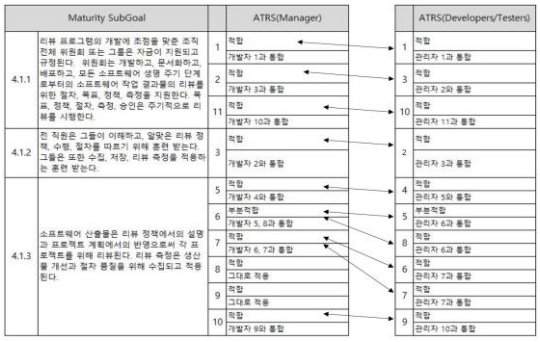


그림 2 성숙도 목표 4.1의 유사항목 통합 결과

TPI Next 추가에서는 성숙도 부목표 별로 부족한 테스트 활동이 없는지 분석한다. TMM의 실무활동들이 테스트와 관련된 모든 것을 다루지는 못하기 때문에, TPI Next의 활동들로 보완하였다. 예를 들어, 성숙도 부목표 4.2.3은 테스트 측정 프로그램 지원을 위한 훈련/도구/자원의 제공에 관한 것이다. 이 항목에서 측정에 대한 입력 방법과 훈련/도구/자원의 수집 및 게시에 대한 구체적인 내용이 부족하기 때문에, TPI Next의 Metrics와 Test Tools 영역의 항목을 추가하였다. 이러한 과정을 통해, 레벨 2부터 5까지 필요한 TPI Next 항목들을 적용한다.

2.4 최종 결과

표 1은 제안한 KTMM의 최종 결과이다. 기존 TMM의 목표, 부목표, 실무활동들을 제안한 모델과 비교한 것이다. 두 모델의 성숙도 목표는 동일하였다. 성숙도 부목표는 KTMM이 기존의 TMM보다 2개 더 많았다. KTMM에서는 테스트 범위에 대한 구체적인 활동이 부족하였기 때문에, TPI Next의 Degree of Involvement 성숙도 부목표를 추가하였다. 또한 TMM에서는 테스트 프로세스 제어 및 모니터링을 위한 테스트 환경 활동이 부족하기 때문에 성숙도 부목표 3.4.4에 추가하였다. 핵심요소들은 TMM에 비해 KTMM의 개수가 현저히 줄어들었다. TMM의 전체 항목에서 40.77%의 항목이 감소되었다.

구분	TMM					한국형 테스트 성숙도 모델					결과
	레벨2	레벨3	레벨4	레벨5	총합	레벨2	레벨3	레벨4	레벨5	총합	
성숙도 목표	3	4	3	3	13	3	4	3	3	13	-
성숙도 부목표	13	11	9	10	43	14	12	9	10	45	+2
ATRs	91	118	79	97	385	61	79	38	50	228	157

표 1 최종 결과

3. 결 론

국내 SW개발 환경과 조직을 고려한 개선된 한국형 테스트 성숙도 모델의 경량화 방법을 제안하였다. KTMM은 기존의 인증 모델에 비해 평가 절차 및 항목별 산출물을 국내 실정에 맞게 개선하였다. 따라서 국내 중소기업의 테스트 조직의 성숙도 개선 및 소프트웨어 품질 개선에 도움이 될 것으로 기대한다. 하지만 개발된 모델이 국내 SW 개발 조직에 적합한지에 대한 검증이 필요하다. 향후 연구로는 개발된 모델을 실제 기업에 시범 적용하여 개선사항을 도출하고, 반영하는 연구가 필요하다.

참고문헌

[1] Bo Kyung Park, So Young Moon, Ki Du Kim, Woo Sung Jang, R. Young Chul Kim, C. R. Carlson, “Refining an Assessing Model for Simplified TMM”, 2016 PlatCon, 2016.02

[2] 한국정보통신기술협회, “한국형 테스트 성숙도 모델 개발 최종보고서(1 차)”, 2015. 11

[3] 한국정보통신기술협회, “한국형 테스트 성숙도 모델 개발 최종보고서(2 차)”, 2016. 11

이슈 관리 프로세스의 정량적 평가 프레임워크

오승원 ○ 전은진 ○ 한혁수 ○
상명대학교 ○
ohsing0906@gmail.com○, dmswls3030@gmail.com○, hshan@smu.ac.kr○

A Quantitative Evaluation Framework for Issue Management Process

Seungwon Oh○ Eunjin Chun○ Hyuksoo Han○
Sangmyung University○

요 약

SW 프로젝트의 모니터링 및 통제 프로세스에서 이슈 관리는 프로젝트의 성공에 중요한 역할을 한다. PM은 프로젝트 상태를 파악하기 위해 주기적으로 이슈 발생 여부와 기존 이슈들의 상태를 점검해야 한다. 이슈 관리 프로세스는 이러한 프로젝트 상태 점검의 기반을 이룬다고 할 수 있기 때문에 QA는 주기적으로 이슈 관리 프로세스 준수 여부를 점검할 필요가 있다. 그런데 QA의 점검은 일반적으로 체크리스트로 이루어지기 때문에 QA의 주관에 의존하는 경향이 크고, 일관성과 정확성이 떨어질 수 있다. 본 논문에서는 이슈 관리 프로세스의 정량적 평가 프레임워크를 제시하여, 이러한 문제를 보완하고자 한다. GQM 방식으로 이슈 관리 지표들을 정의하고, 각 지표들의 산출 방식에 객관성을 제공하기 위해 이슈 관리 도구에 축적되어 있는 자료들을 활용하는 방안을 제시한다. 또한 하나의 지표로 현 상태를 파악하기 위해 이슈 관리 지표들을 입력으로 받아 이슈 관리 프로세스 준수율을 계산할 수 있는 퍼지 시스템의 구축 지침을 제공한다.

1. 서 론

SW 개발 프로젝트가 시작되면, 프로젝트 모니터링 및 통제 프로세스(Project Monitoring & Control Process)에 따라 프로젝트를 관리한다. 주간 회의, 월간 회의, 마일스톤 회의 등을 통해 계획서에 정의된 활동들이 예정대로 수행되고, 성과를 내고 있는지를 점검하고 통제한다. 이 때 계획과 비교해서 크게 벗어나는 요소들은 이슈(Issue)로 처리하여, 적절한 시정조치를 취하게 된다.

이슈는 프로젝트 진행 상태의 이상 여부를 파악할 수 있는 핵심 항목으로, PM(Project Manager)은 점검회의 때, 이슈 발생 여부와 기존 이슈들의 상태에 대해 점검한다.

이슈의 등록, 진행, 해결로 이루어지는 이슈 관리 프로세스는 이러한 프로젝트 점검 활동의 기반이 된다. 따라서 조직들은 이슈 관리 프로세스를 수립하고, 이에 기반하여 정확하고 체계적으로 이슈 관리를 진행하는 것이 중요하다. 그러므로, QA(Quality Assurance)들은 이슈 관리 프로세스가 제대로 수행되고 있는지를 평가하기 위한 방안이 필요하다[1].

일반적으로 QA는 체크리스트를 기반으로 프로세스 준수 여부에 관한 평가를 수행한다. 하지만, 체크리스트 기반 평가는 평가자의 전문적 지식과 주관적인 판단에 의존하기 때문에 일관성과 정확성이 떨어질 수 있다. 그러므로, 체크리스트를 보완하여 객관성을 확보하기 위한 방안이 필요하다.

본 논문에서는 이슈 관리 프로세스의 준수 수준을 파악하기 위한 지표(Indicator)로 “이슈 관리 프로세스 준수율” 이라는 지표를 정의하고, 이 수치를 도출하기 위해 정량적 평가 프레임워크(Framework)를 제안한다.

1단계로 GQM(Goal Question Metric) 방식에 따라 이슈 관리 평가 Metric들을 도출한다. “이슈 관리 프로세스의 준수 여부 파악”을 Goal로 삼고, 이 Goal을 만족시킬 수 있는 Question들을 통해 체크리스트를 작성한다. 그리고 Question들에 대응하는 Metric들은 평가자의 주관적 판단과 이를 보완하기 위한 객관적 Data로 구성된다. 이 Metric들을 정의하는 과정에서 객관적 Data는 이슈 관리 도구(Issue Management System)에 축적되어있는 자료들을 활용한다.

2단계에서는 하나의 지표로 현재의 프로젝트의 상태를 파악할 수 있도록, 1단계에서 도출한 이슈 관리 평가 Metric들을 종합하여, 이슈 관리 프로세스 준수율을 도출한다. 이를 위해 퍼지 시스템(Fuzzy System)을 구축하고, 전문가들의 지식을 기반으로 퍼지 규칙(Fuzzy Rule)들을 작성한다.

본 연구는 형상관리 도구로 SVN 또는 Git를 사용하고, 이슈 관리 도구로 Redmine을 사용하는 조직을 기반으로 이루어졌다.

2. 관련 연구

2.1 프로젝트 모니터링 및 통제 프로세스 (Project Monitoring & Control Process)

프로젝트 모니터링 및 통제는 계획과 비교하여 프로젝트 성과에 대해 수집, 분석, 보고하는 활동이다. 이는 프로젝트 진행 상황이 계획에서 벗어났을 때 효과적인 시정조치를 취하기 위한 검토 활동과 이슈 관리 활동으로 이루어진다[2][3].

(1) 검토 활동

프로젝트의 진행 정도는 주로 WBS(Work Break Down Structure) 범위 내의 프로젝트 작업 산출물, 소요 비용과 투입된 공수, 자원 등을 계획과 비교한다.

(2) 이슈 관리 활동

검토 활동 결과, 계획과의 편차가 식별되면 이슈로 처리하여 시정조치를 취한다. 프로젝트 관리 측면에서의 편차 이외에도 테스트나 검토 과정에서 발견한 결함, 새로 요청 받은 기능, 기능의 변경 등 관심을 가지고 지켜보며 해결해야 할 대상들도 이슈로 다루는 것이 일반적이다.

2.2 이슈 관리 프로세스(Issue Management Process)

이슈 관리는 [표 1]과 같이 이슈를 발견하고 등록하는 단계와 이슈를 분석하여 적절한 개발 담당자에게 할당하는 단계, 이슈를 처리하는 진행 단계, 개발 담당자가 처리를 끝낸 후 해결하고 완료하는 단계, PM이 해결된 이슈를 검증하고 완료하는 단계들로 구성된다.

[표 1] 이슈 관리 프로세스의 과정 및 담당자

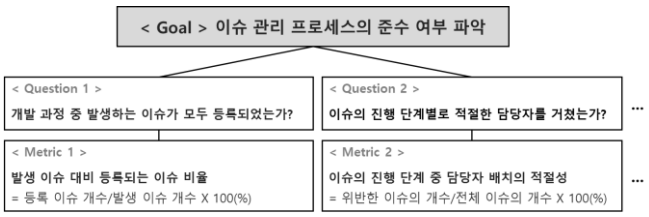
NO.	단계	담당자		
		새기능	결함	이슈
1	이슈 발견 및 등록	PM	결함을 발견한 사람	PM
2	이슈 분석 및 할당	PM 혹은 개발 담당자	PM 혹은 개발 담당자	PM 혹은 개발 담당자
3	이슈 진행	PM 혹은 개발 담당자	PM 혹은 개발 담당자	PM 혹은 개발 담당자
4	이슈 해결	담당 개발자	담당 개발자	담당 개발자
5	검증 및 완료	PM	결함을 발견한 사람	PM

본 논문에서 이슈 관리 프로세스는 오픈 소스 도구인 Redmine을 통해 수행된다. [그림 1]은 Redmine으로 이슈 관리를 하고 있는 예를 보여주고 있다.

[그림 1] Redmine을 통한 이슈 관리

2.3 GQM(Goal Question Metric)

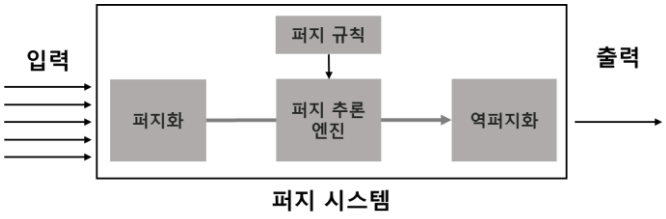
GQM은 목표 지향 측정 지표를 선정하기 위한 방식으로, Goal 달성 여부에 집중할 수 있도록 Question들과 Metric들을 정의한다. Goal을 달성하기 위해 필요한 요건들을 Question들으로 제시하며, 그 Question들에 대한 수치적 해답을 얻기 위해 Metric들을 정의한다[4]. [그림 2]는 일반적인 GQM 과정을 나타내고 있다.



[그림 2] 일반적인 GQM 과정

2.4 퍼지 시스템(Fuzzy System)

퍼지 시스템은 측정된 자료를 입력으로 받아, 주어진 규칙에 따라 추론하여 논리적으로 타당한 수치를 얻기 위한 시스템으로, 퍼지화, 퍼지 규칙, 퍼지 추론 엔진, 역퍼지화로 구성되어 있다[5]. 퍼지 시스템은 전문가의 지식을 자동화하기 위해 채택되어 왔다[6]. [그림 3]은 퍼지 시스템을 도식화한 것이다.



[그림 3] 퍼지 시스템

(1) 퍼지화(Fuzzification)

측정된 자료를 입력 값으로 받아, 그 입력 값들을 소속 함수로 바꾸는 작업을 한다.

(2) 퍼지 규칙(Fuzzy Rule)

전문가의 지식을 기반으로 생성된 추론 규칙을 정의한다.

(3) 퍼지 추론 엔진(Fuzzy Inference)

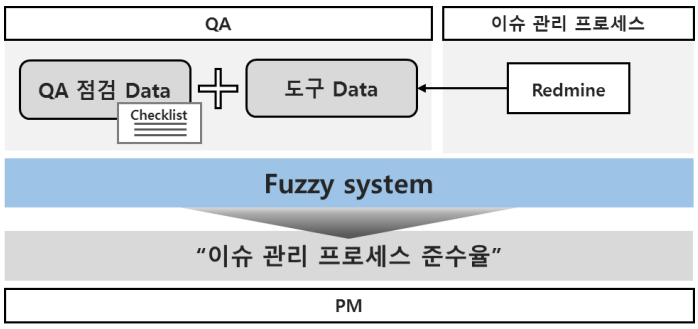
퍼지 함수와 퍼지 규칙을 이용하여 전문가의 판단과 유사한 퍼지 추론 값을 구한다.

(4) 역퍼지화(Defuzzification)

퍼지 추론 엔진의 결과 값을 변환하여 0과 1사이의 수치로 변환한다.

3. 이슈 관리 평가 프레임워크

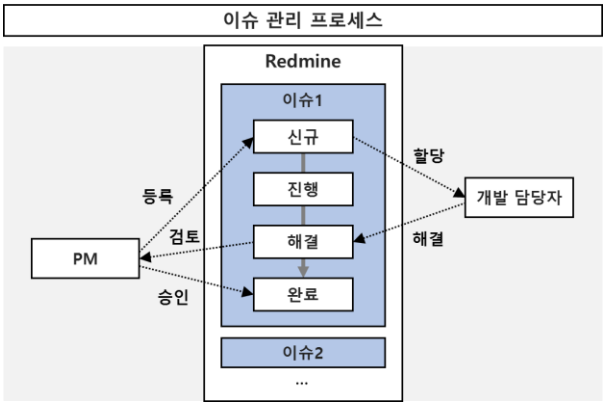
본 논문에서는 이슈 관리 프로세스의 정량적인 평가를 위해 QA의 Checklist와 Redmine으로 수집한 데이터를 기반으로 프로세스 이행 점검 결과를 추출하여 퍼지 시스템에 적용하는 이슈 관리 평가 프레임워크를 제안한다. [그림 4]는 이슈 관리 평가 프레임워크를 도식화하여 나타낸 것이다.



[그림 4] 이슈 관리 평가 프레임워크

3.1 이슈 관리 프로세스 구축

이슈 관리 프로세스는 [그림 5]와 같이 Redmine을 통해 이루어 진다. 이슈가 발생하면 PM이 이슈를 등록하고, 개발 담당자에게 할당한다. Redmine 내부의 이슈 상태는 개발 담당자가 진행을 하고, 해결한 후 PM의 승인을 받으면 종료하는 과정으로 이루어 진다.



[그림 5] 이슈 관리 프로세스

3.2 GQM 기반의 이슈 관리 평가 지표 도출

이슈 관리 프로세스를 평가하기 위해 GQM을 기반으로 평가 지표를 도출한다.

[표 2]와 같이 "이슈 관리 프로세스의 준수 여부"를 Goal로 정의하고, 이 수치를 도출하기 위한 Question들과 Metric들을 정의한다.

[표 2] Goal에 따른 Question과 Metric

지표		이슈 관리 프로세스 준수율	
Goal		**Question**	**Metric**
이슈 관리 프로세스의 준수 여부 파악	1	개발 과정 중 발생하는 이슈가 모두 등록되었는가?	발생 이슈 대비 등록되는 이슈 비율
	2	이슈의 진행 단계별로 적절한 담당자를 적절하게 할당했는가?	이슈의 진행 단계 중 담당자 배치의 적절성
	3	이슈가 예상 완료 기한 대비 적절하게 끝났는가?	예상 기간 대비 소요 기간 비율
	4	실제 진행 상태와 동일하도록 적절하게 진행하고 있는가?	실제 진행 상태 대비 이슈 상태 일치 여부
	5	소스코드 변경의 경우, 관련 Commit Message가 연동이 되고 있는가?	이슈 별 저장소 연동 유무

3.3 Metric별 Data 수집 및 측정 방법 도출

각 Metric의 측정은 QA의 체크리스트 기반 점검 Data와 Redmine에 수집된 도구 Data를 활용하여 이루어진다.

- QA 점검 Data: GQM 기반의 Question을 적용하여 작성된 체크리스트 Data
- 도구 Data: Redmine의 수집된 data

[표 3]은 Metric을 수치화 하기 위한 data의 수집 방법과 그것의 측정 방법을 나타낸다.

[표 3] 각 Metric에 대한 data 수집 및 측정 방법

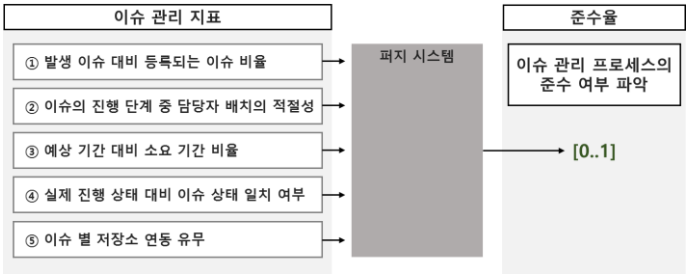
Metric	수집 방법	측정 방법
발생 이슈 대비 등록되는 이슈 비율	발생 이슈 개수 - QA 점검 **등록 이슈 개수 - Redmine**	**P = A/B X 100(%)** (A = 등록 이슈 개수, B = 발생 이슈 개수)
이슈의 진행 단계 중 담당자 배치의 적절성	위반한 이슈 개수 - QA 점검	**P = A/B X 100(%)** (A = 위반한 이슈 개수, B = 전체 이슈 개수)
예상 기간 대비 소요 기간 비율	**예상기간 - Redmine** **소요기간 – Redmine**	**P = A/A+A' X 100(%)** 정상 이슈: A (예상 기간 >= 소요 기간) 비정상 이슈: A' (예상 기간 < 소요 기간)
실제 진행 상태 대비 이슈 상태 일치 여부	위반한 이슈의 개수 - QA 점검	**P = A/B X 100(%)** (A = 위반한 이슈 개수, B = 전체 이슈 개수)
이슈 별 저장소 연동 유무	**저장소와 연동된 이슈 개수 - Redmine** **전체 이슈의 개수 – Redmine**	**P = A/B X 100(%)** (A = 저장소와 연동된 이슈 개수, B = 전체 이슈 개수)

예를 들어, "개발 과정 중 발생하는 이슈가 모두 등록되었는가?"에 대한 Metric은 발생 이슈 대비 등록된 이슈의 비율이다. 이 Metric을 계산하기 위한 발생 이슈는 QA의 점검으로부터 추출하고 등록 된 이슈는 Redmine으로부터 얻어진다.

3.4 퍼지 시스템 적용

각각의 Metric별로 도출된 수치 결과를 종합하여 하나의 평가 지표 값인 이슈 관리 프로세스 준수율을 구하기 위해 퍼지 시스템을 구축한다.

본 논문에서는 맘다니형 추론(Mamdani method)을 사용하여 값을 도출한다. [그림 6]은 이슈 관리 준수율을 구하기 위한 퍼지 시스템을 나타낸다.



[그림 6] 이슈 관리 준수율을 구하기 위한 퍼지 시스템

(1) 퍼지화

앞의 [표 3]에서 제공한 5개의 Metric으로부터 산출된 값들을 퍼지화하여 퍼지 시스템에 입력한다.

(2) 퍼지 규칙

프로세스 평가 전문가와 관련 표준으로부터 도출된 이슈 관리 프로세스에 관한 퍼지 규칙을 퍼지 시스템에 입력한다.

(3) 퍼지 추론

(1)을 통해 퍼지화된 퍼지 집합과 (2)를 통해 도출된 퍼지 규칙 사이의 연산으로부터 통합된 출력 퍼지 집합을 생산한다.

(4) 역퍼지화

(3)을 통해 도출된 출력 퍼지 집합을 역퍼지화하여, 최종 평가 지표인 이슈 관리 프로세스 준수율에 대한 수치를 도출한다. 역퍼지화에는 COG(Centre Of Gravity) 방식을 채택하였다.

퍼지 시스템 적용 결과로 나온 정량적 수치는 이슈 관리 이행 시, 이슈 관리 프로세스를 얼마나 준수하는지에 대해 나타낸다. 따라서 PM과 QA는 이 수치를 활용하여 현재의 이슈 관리 준수 상태를 보다 정확하게 파악할 수 있다.

4. 결론

프로젝트의 성공을 위해 주기적으로 이슈들의 상태를 점검하는 것이 필요하다. 이슈 관리 프로세스는 프로젝트 상태 점검의 기반을 이룬다고 할 수 있기 때문에, 이슈 관리 프로세스의 상태를 점검하는 것이 요구된다.

본 논문에서는 이슈 관리 프로세스의 준수 수준을 파악하기 위한 지표로 “이슈 관리 프로세스 준수율”이라는 지표를 정의하고, 이 수치를 도출하기 위해 정량적 평가 프레임워크를 제안했다.

1단계로 GQM 방식에 따라 이슈 관리 평가 Metric들을 도출하고, 2단계에서는 “이슈 관리 프로세스 준수율”이라는 하나의 지표로 현재의 상태를 파악할 수 있도록 하였다. 이를 위해 퍼지 시스템을 구축하기 위한 지침을 제공했다.

제안된 프레임워크를 활용하여 기존 방식 보다 객관적이고 일관된 평가를 수행할 수 있을 것이다.

하지만, 본 논문에서 제시한 GQM은 각 조직의 성격에 맞도록 Goal, Question, Metric을 수정한 후에 적용해야 함을 유의해야 한다.

논문 사사(Ack)

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음. (IITP-2016-R0992-16-1014)

참고 문헌

[1] Ansoff, H. Igor. "Strategic issue management." Strategic management journal 1.2 (1980): 131-148.

[2] Chrissis, M. B., Konrad, M., & Shrum, S, “CMMI: Guidelines or Process Integration and Product Improvement, Addison-Wesley Longman Publishing Co,. Inc., 2011

[3] 오승원 외, “소규모 소프트웨어 프로젝트의 통제와 제어 프로세스 구축”, 한국정보과학회 2016년 한국컴퓨터종합학술대회 논문집, 678-680 , 2016

[4] R. Van Solingen and E. Berghout, “the Goal/Question/Metric Method- A Practical Guide for Quality Improvement of Software development”, McGRAW-Hill Companiew, London 1999.

[5] Michael Negnevitsky, “Artificial Intelligence: A Guide to Intelligent Systems”, Addison Wesley Publishing Company ,2001

[6] 김준홍, “퍼지 서비스 FMEA를 이용한 서비스 설계”, 한국산업경영시스템학회 2009 춘계학술대회 논문집, 162-167, 2008

ACMI 데이터 링크 설계 및 M&S 수행 결과

2017.
심인보, 오지현, 김천영, 지철규
국방과학연구소

목차

1/19

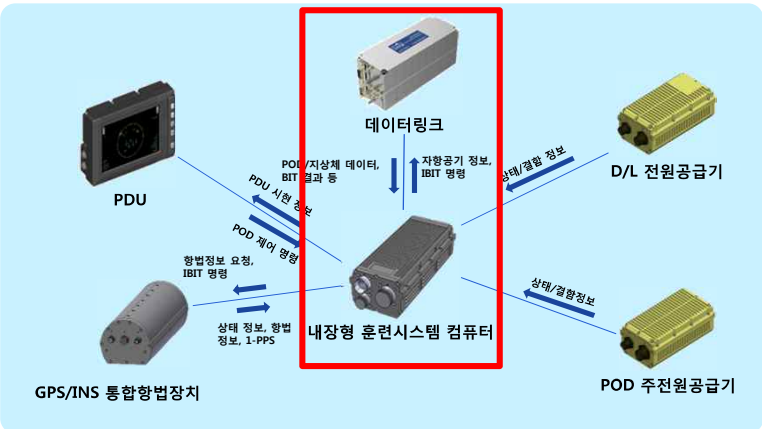
국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

I. 개 요

- 연구목표
- ETC 실행개념
- 연구방안

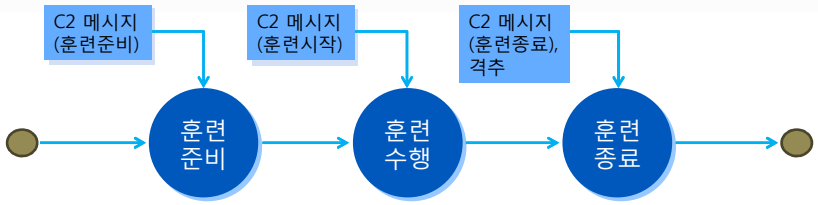
1.1 연구목표

- 내장형 훈련시스템 컴퓨터(데이터링크)
  - CTP/AIM-9형 ACMI POD 구성품인 내장형 훈련시스템 내 데이터링크 관리 프로그램 개발



1.2 ETC 실행개념

- 운용 시나리오 및 모드



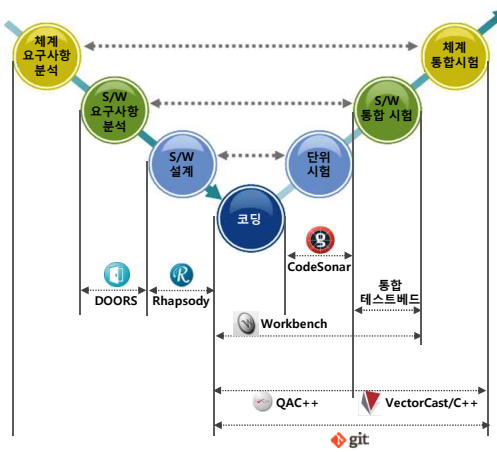
훈련 준비		훈련 수행	훈련 종료	
▪ RDS에 시나리오 장입 ▪ 이륙 및 훈련 구역으로 이동	▪ 시나리오 로딩 ▪ 항전/무장 모델 초기화	▪ 항전/무장 모델 모의 시작 ▪ 훈련안전 경고 활성화 ▪ 훈련데이터 저장 시작	▪ 항전/무장 모델 모의 중지 ▪ 제한적 훈련안전 경고 ▪ 훈련데이터 저장 중지	▪ RTB

4/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

1.3 연구방안

- 개발 프로세스
  - 객체지향 설계 및 UML을 이용한 모델링 기반 설계
  - 방위사업청 매뉴얼 2014-1호 '무기체계 소프트웨어 개발 및 관리 매뉴얼' 준수



5/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

II. 데이터링크 관리 및 설계

6/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. 데이터링크 관리 및 설계

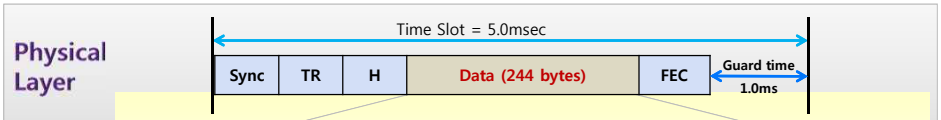
- 데이터링크 관리 P/G 구성
  - 네트워크 접근 제어
    - 우선순위 테이블 생성 및 관리
    - 네트워크 맵 관리 및 갱신
    - GPS Time 수신 및 Time Slot 관리
    - 네트워크 접근 관리
  - 데이터 전송 관리
    - Occasional Minor 스택 관리
    - 타 객체 Navigation Minor 스택 관리
    - C2 메시지 및 C2 응답 메시지 스택 관리
    - Wind Minor 관리
  - 데이터링크 장비 인터페이스 관리
    - Transmitter 제어
    - Receiver 제어
    - BIT 제어

7/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. 데이터링크 관리 및 설계

- 데이터링크 관리 CSU
  - Frame Packing 구조
    - ACMI의 MAC은 Dynamic TDMA를 이용하여 설계
    - Time Slot은 5.0msec로, 200Hz로 데이터 전송 가능
    - 데이터 충돌방지를 위해 Propagation을 고려하여 1.0ms Guard Time 지정
    - 하나의 Time Slot은 총 244bytes의 크기로 구성
    - 하나의 Time Slot은 네트워크 헤더 / 자항공기 항법정보 / 3개의 비 주기 메시지 / 5개의 relay 또는 wind 정보 / 2개의 CRC로 구성



D/L Layer

크기	24	30	12	12	12	30	30	30	30	30	2	2
필드	Net Minor	Self Nav.	Occa-sional	Occa-sional	Occa-sional	Relay / Wind	Relay / Wind	Relay / Wind	Relay / Wind	Relay / Wind	CRC	CRC

8/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. 데이터링크 관리 및 설계

- 데이터링크 관리 CSU (계속)
  - Static vs Dynamic TDMA
    - Static TDMA 방식
      - Static 방식으로 200 Slot을 100개의 노드가 사용할 경우 업데이트 주기는 2Hz
      - 참여 중인 항공기(노드)가 적은 경우 낭비되는 Slot 발생
    - Dynamic TDMA 방식
      - 자신이 속해있는 네트워크 내에서 200 Slot을 분할하여 사용

      * 네트워크에 속한 노드가 20개인 경우, 업데이트 주기는 10Hz가 되므로 전송률을 높일 수 있음
  - Dynamic TDMA 알고리즘
    - 모든 네트워크 노드는 200Hz 주기 중 자신이 송신해야 할 시점을 알고 있음
    - 한 Time Slot 시점에는 해당되는 노드 이외에는 송신을 금지
    - 각 노드는 자신의 Time Slot마다 자신의 B-list와 Silent Time 전송
    - 이를 수신한 다른 노드들은 B-list와 C-list를 갱신
    - 만약 다른 멤버로부터 B-list를 받고 B-list에 있는 멤버로부터 직접 B-list를 수신하지 못했을 경우, 해당 멤버는 C-list로 갱신

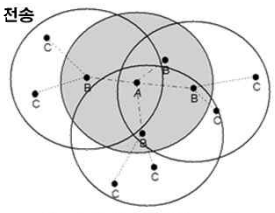


Figure 2 : The Network Neighbours Of A

9/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. 데이터링크 관리 및 설계

- 데이터링크 관리 CSU (계속)
  - Silent Time
    - RF 신호의 충돌(Collision)을 방지하기 위해 각각의 노드들은 정해진 시간에만 송신 가능
    - 각각의 노드들이 송신할 수 있는 시간을 지정하기 위해 노드 간 Silent Time 공유
    - 공유를 위해 모든 항공기는 자신의 Time Slot 에 Silent Time 정보 송신
    - Silent Time에 속한 노드는 해당 주기 동안 데이터 송신 불가
    - B-list에 참가중인 노드가 20개인 경우 하단 수식과 같이 최대 전송률 10㎐ 유지 가능

$T_{Silent} = T_{MAX}[\ 20_{Slot},\ (A_{Slot}+B_{Slot}+C_{Slot})\ ]$

$TX_{MAX(최대전송률)} = 200㎐ / T_{Silent}$

    - 지정된 Silent Time 만료 후, 3번의 재전송 요청에도 자신의 Time Slot이 돌아오지 않으면 Stress 상태로 간주하며 이 상태에서는 C-list를 무시
      * 해당 Slot에서 자신보다 우선순위가 높은 B 멤버가 없으면 자신보다 우선순위가 높은 C 멤버가 있더라도 무시하고 자신이 Time Slot 차지
        - 이 순간에는 A와 C 사이에 잠재적인 Collision 발생 가능
        - Stress 상태에서 Time Slot 획득 시, 다음 Silent Time은 실제 전송한 Time Slot부터 재 계산
        - Stress 상태가 되었다는 것은 해당 Time Slot을 차지하기 위한 경쟁이 순간적으로 치열하다는 의미이며, 상기의 알고리즘을 통해 Congestion 현상 해소 가능

2. 데이터링크 관리 및 설계

- 데이터링크 관리 CSU (계속)
  - 시간 동기화 방법
    - GPS 시간을 이용하여 모든 POD 시각 동기화 수행
    - 매초마다 GPS로부터 수신한 1-PPS를 이용하여 내부 타이머 동기화
    - 5ms 마다 스케줄러를 동작시킴으로써 데이터링크 CSC 실행 동기화
  - Time Tag
    - System Time Tag
      - TDMA 시각 동기에 사용
      - GPS에서 수신한 시간을 이용하여 TOD(Time Of Day)로 변환
        - 매주 토요일 00시를 기준으로 산출
        - 일주일을 1ms 단위로 변환하여 관리
        - 시간 해상도 : 1ms
    - Communication Time Tag
      - 데이터링크 메시지 생성 시간을 식별하는데 사용하며 메시지 헤더에 저장하여 전송
      - 수신 측에서 메시지를 수신한 시점과 메시지가 생성된 시점의 오차 보정을 위해 사용
      - 시간 해상도 : 100ms

2. 데이터링크 관리 및 설계

- 데이터링크 관리 CSU (계속)
  - Relay(중계) 알고리즘
    - LOS / 전송거리 등의 통신 제약을 해소하기 위한 중계기능 제공
    - 타 항공기로부터 획득한 정보를 다른 항공기로 중계 가능
    - 중계용으로 하나의 Minor Frame에 5개의 중계 필드 제공

크기	24	30	12	12	12	30	30	30	30	30	2	2
필드	Net Minor	Self Nav.	Occasi-onal	Occasi-onal	Occasi-onal	Relay / Wind	Relay / Wind	Relay / Wind	Relay / Wind	Relay / Wind	CRC	CRC

    - 중계 기능을 통해 자신의 정보 또는 타 항공기로부터 획득한 정보를 통달거리 밖의 노드에 전송 가능
    - 어떤 정보를 우선 전송할 것인지는 메시지 우선순위 기반하여 결정

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. 데이터링크 관리 및 설계

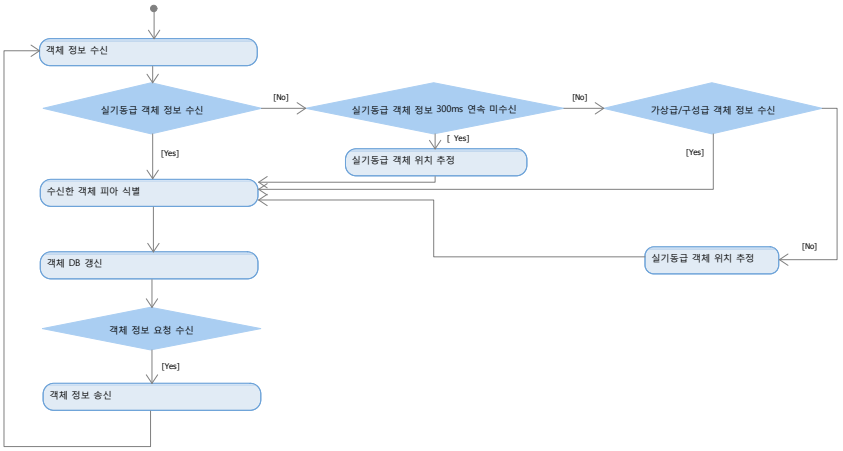
- 데이터링크 관리 CSU (계속)
  - 메시지 우선순위 관리
    - Minor 메시지 종류에 따라 전송 우선순위 정보 관리
    - 메시지 송신 시 하단 표를 기반으로 우선순위가 높은 메시지 우선 송신

우선순위	메시지 명	설 명
0	A2A Collision	공중 충돌 경고
1	Ground Collision	지상 충돌 경고
2	End of A2A Missile Simulation	공대공 미사일 모의 완료
3	Safety Hazard	비행 안전 경고 (기동 제한 초과)
4	C2 Message	C2 메시지
5	Training Status	훈련 상태 메시지
6	A2A Missile Fire	공대공 미사일 발사
7	Gun Fire	기총 발사
8	End of Gun Simulation	기총 모의 종료
9	A2G Weapon Fire.	공대지 미사일 발사
10	End of A2G Weapon Simulation	공대지 미사일 모의 완료
11	Heat Seeker Status	Seeker 상태 정보
12	Radar Status	Radar 상태 정보
13	CMDS Fire	CMDS(Chaff/Flare) 발사 정보
14	Weapon Status	무장 상태 정보
15	Call Sign	Call sign
16	General Information	항공기 정보

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. 데이터링크 관리 및 설계

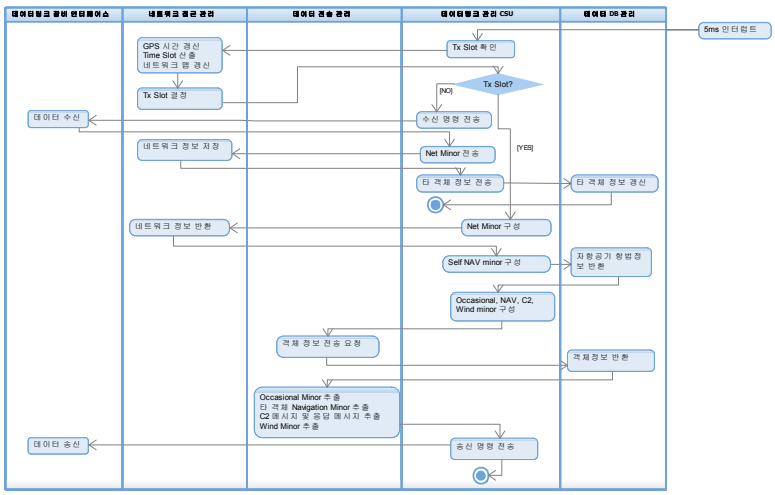
- **데이터링크 객체 관리 CSU 다이어그램**
  - 활동 다이어그램



국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. 데이터링크 관리 및 설계

- **Datalink 연동 CSU 다이어그램**
  - 활동 다이어그램



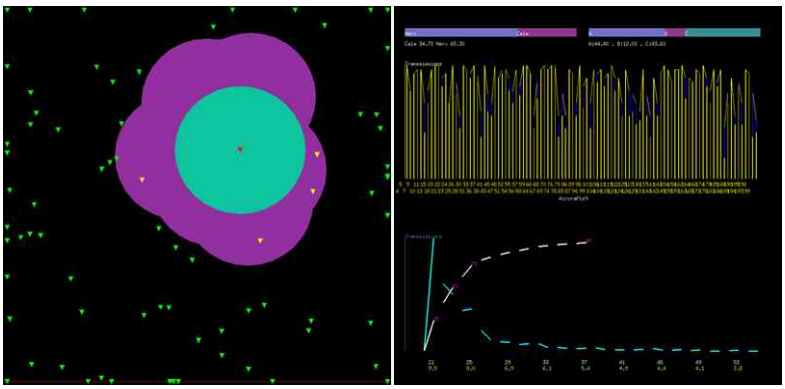
국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

III. M&S 결과 및 발전방안

16/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

3. M&S 결과 및 발전방안



<시뮬레이션 수행 화면>

17/19

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

3. M&S 결과 및 발전방안

- 시뮬레이션 시나리오
  - 데이터링크 MAC 설계 시, 시뮬레이션을 수행하여 최적의 파라미터 값 도출
  - 시뮬레이션에 사용된 파라미터는 하단 표와 같음

파라미터	설정 값
기체 수	100대
기체 이동속도	마하 0.3 ~ 1.0
기체 이동방향	랜덤
RF 송신 세기	40 W
RF 통달거리	70 NM
Bit Rate	1 Mbps
MAC	TDMA
Time Slot	200 slots / sec
1 slot 사이즈	2000 bits

<데이터링크 장비 특성>

파라미터	설정 값
RemoveBtoC	2000 slots
SlotsToDeleteC	1000 slots
MaxSilentSlots	180 slots
MinSilentSlots	20 slots
MinStress	25 slots
MaxStress	75 slots
MaxSlotsForNotToTX	280 slots
Network Mode	Dynamic

<도출된 파라미터>

- 시뮬레이션 결과
  - B-list 멤버 간 평균 지연 시간 : 25.17ms
  - 노드의 평균 전송 주기 : 약 8Hz

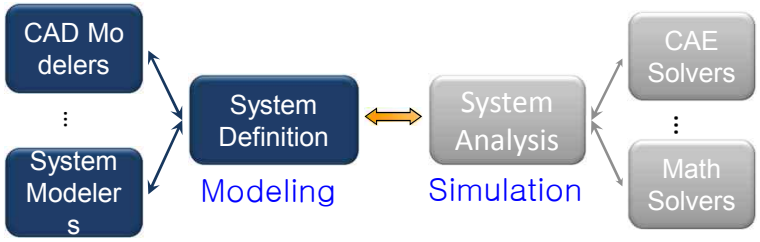
국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

3. M&S 결과 및 발전방안

모델 기반 시뮬레이션(Model Based Simulation)

MCAD
(CATIA, UGS-NX, Pro/E,…)
ECAD
(Zuken CR500, MGC BoardStation,…)

FEA Solvers (ABAQUS, ANSYS,…)
CFD Solvers (Fluent,…)
DEVS Solvers (DESIDE, ExtendSim, ARENA,…), …



SysML/UML
(Rhapsody, Telelogic UML, MagicDraw, Artisan Studio, E+, …)

Matlab
Mathematica
OpenModelica, …

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

소프트웨어 개발 조직의 특성에 관한 연구 : 현상과 특성에 관한 조사

김윤수 ○, 김재욱, 이원영, 김태호, 민상윤

KAIST 소프트웨어 대학원

서울시 강남구 논현로 28 길 25

ystoto@kaist.ac.kr, kjwook@kaist.ac.kr, i.am.peter.kr@kaist.ac.kr,

evan.kim@kaist.ac.kr, symin@kaist.ac.kr

A Study on the Characteristics of Software Development Organization : Investigation on phenomenon and characteristics

Yun-Su Kim○, Jae-Wook Kim, Won-Young Lee, Tae-Ho Kim, Sang-Yoon Min

KAIST Software Graduate Program

요 약

시대의 변화에 따라 조직은 점점 거대해지고 복잡해지며 지속적으로 발전해왔다. 그에 따라 조직의 내/외적으로 일어나는 많은 현상들도 조직의 여러 가지 특성에 따라 매우 복잡하게 일어나고 있다. 소프트웨어 산업은 제조업 등 기존의 노동집약적 산업들에 비해 지식산업적 특성을 가지고 있어 더욱 복잡한 조직 현상이 일어나며 이를 예측하기도 매우 어렵다. 이로 인해 소프트웨어 개발 조직을 운영하는 경영자들은 조직 개발에 많은 노력을 기울임에도 불구하고 예상치 못했던 새로운 조직의 문제점에 봉착하며 조직 개발의 어려움을 겪고 있다. 본 연구에서는 소프트웨어 개발 조직에서 구성원들이 느끼는 문제점과 그 수준을 확인하고, 소프트웨어 개발 조직의 특성을 People, Process, Technology 세 가지 관점에서 해석하였다. 이를 통해 소프트웨어 개발 조직에서 나타나는 현상들의 복잡성을 이해하고 소프트웨어 개발 조직에 특화된 조직 연구의 필요성을 제시하였다.

1. 서 론

1.1. 배경

1.1.1. 시대 흐름에 따른 조직이론의 변화

조직을 체계적으로 관찰하고 이해하는 조직학(Study of organization)은 산업혁명 이후 대규모의 복잡한 조직들이 형성되고 발전하면서 활발히 연구되기 시작하였다.

조직학을 구성하는 조직이론은 능률주의, 공식적 구조와 과정을 중시한 *고전조직이론*으로 출발한다. *고전조직이론*은 조직학을 독자적 연구 영역으로 만드는 기틀이 되었으나 능률에만 집중하며 조직현상에 영향을 미치는 비공식적 요인을 간과하고, 조직 구성원을 조종/통제 가능한 단순한 모형으로 다루는 등 여러 한계점을 보였다.[1]

시대적 변화에 따라 근로자의 위상이 높아지고 조직의 규모가 더욱 팽창하여 환경의 복잡성이 증대되면서 *고전조직이론*의 한계점을 극복하자는 *신고전조직이론*이 등장하게 된다. *신고전조직이론*은 조직구성원의 만족도, 사회적 욕구의 충족도를 강조하며 비공식적 요인을 중시하고, 조직구성원을 단순한 하나의 구성요소가 아닌 사회적 인간으로서 바로보기 시작한다. 하지만 이 이론 역시 한정적인 관점에서 조직현상을 바라보았다는 점에서 *고전조직이론*과 유사하게 편협한 접근을 하였고, 인간의 욕구체계를 단순화/획일화하는 등 한계점을 보였다.[1]

이 후 고도산업화와 정보화, 그리고 세계화 등 시대적 변화가 급격하게 일어나면서 *현대조직이론*이 등장하게 된다. 이 이론은 더욱 복잡해진 조직들을 이해하기 위해 다양한 접근방법으로 조직을 바라보며, 여러 학문분야의 이론과 기법을 동원하여 조직현상의 본질을 파악하려는 학제적 연구를 심화시켰다.[1]

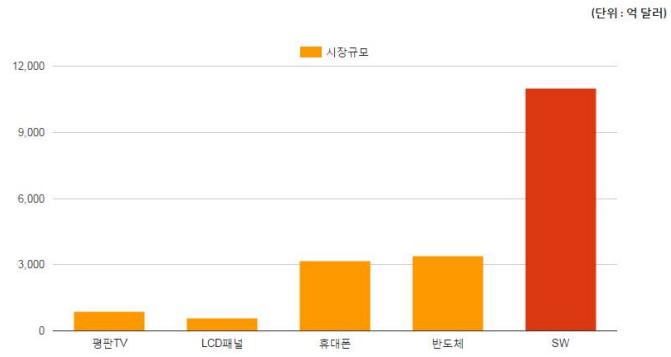
이러한 조직이론의 변화는 시대적 흐름에 따라 점점 다양해지고 복잡해진 조직의 변화와 사회적 요구에 맞추어 지속적으로 발전해 온 것이다.

조직현상은 한 조직의 목표, 구성원, 구조, 환경, 업종, 문화, 시간 등에 따라서 매우 다르게 나타나기에 계속

해서 변화하는 복잡성을 띈다.[2] 따라서 조직이론을 현실에 적용할 때는 어느 정도의 괴리가 나타날 수 밖에 없는데 이는 조직에 대한 지속적이고 깊이 있는 연구를 통해 좁혀 나가야 할 것이다.

1.1.2. 소프트웨어 개발 조직에 대한 연구의 필요성

많은 전문가들이 현대 사회에서 지식산업의 영향력과 중요도를 논하고 있다.[3][4] 그 중에서도 소프트웨어 산업은 대표적인 지식산업으로 평가 받고 있다.[5] 인적 자원의 지식을 활용한 창조 활동, 상호 간의 지식 교류 등을 통해 여러 산업 분야와 융합되어 다양한 제품/서비스를 생산하는 소프트웨어 개발 조직들은 지속적으로 성장하고 있으며 그에 따라 소프트웨어 산업의 규모도 폭발적으로 증가하고 있다. 소프트웨어 정책 연구소(SPRi)의 발표에 의하면 세계 소프트웨어 시장은 10,966억 달러로 반도체 시장의 약 3.3배, 휴대폰 시장의 약 3.5배 규모에 달한다고 한다.(2016년 기준)[6]



[그림 1] 세계 소프트웨어 시장의 규모(2016)

[표 1] 세계 소프트웨어 시장의 규모

구분	평판TV	LCD패널	휴대폰	반도체	SW
시장규모	865	552	3,160	3,372	10,966

이와 같이 인적자원의 활동과 상호관계가 조직의 성공에 큰 영향을 미치는 소프트웨어 개발 조직은 그 규모가 커질수록 인적자원을 고려한 조직관리가 더욱 중요해지며 많은 소프트웨어 개발 조직들이 이를 위해 다양한 조직설계 활동을 하고 있다.

하지만 소프트웨어 산업은 타 산업분야에 비해 상대적으로 늦게 형성되고 발전하여 조직학자들의 집중적인 연구대상이 되지 못하였다. 반면에 소프트웨어 산업 종사자들 또는 소프트웨어 공학자들에 의한 소프트웨어 개발 조직에 대한 실증적 연구는 상대적으로 활발히 진행되고 있어 조직학적 연구에 대한 산업 현장의 요구가 매우 절실함을 확인할 수 있다.

1.2. 문제점

위와 같이 소프트웨어 산업의 급격한 성장에 따라 소프트웨어 개발 조직은 목표 달성을 위해, 그리고 생산성/효율성 향상과 조직의 영속성을 유지하기 위해 주기적 또는 비주기적인 조직설계를 시행한다.

하지만 앞서 설명한 조직현상의 복잡성에 소프트웨어 개발 특유의 복잡성[20]까지 가중되어 조직이론의 적용을 통한 조직설계가 쉽지 않다. 이로 인해 조직 구성원들로부터 조직설계의 효과성에 대한 불만이 나타나는 경우가 종종 발생한다. 그 예로 업계에서 많이 언급되는 문제점은 상위 관리자의 모호성, 상충되는 업무지시, 외부 압력에 취약한 관리자, 비효율적인 의사결정 체계, 낮은 업무만족도 등이 있다.

이는 인적자원의 영향력을 많이 받는 소프트웨어 개발 조직의 특성상 조직의 운용에 있어 큰 위험요소가 되므로 지속적이고 체계적인 연구가 필요하다.

따라서 본 논문에서는 소프트웨어 개발 조직에서 구성원들이 느끼는 문제점과 그 수준을 조사하고자 한다. 또한 소프트웨어 개발 조직의 능률을 향상시키기 위해 중점적으로 관리해야 할 **3대 요소(People, Process, Technology)** 관점에서 소프트웨어 개발 조직의 특성을 분석하여 조직현상의 복잡성을 확인하고자 한다. [7]

2. 관련 연구

2.1. 보편적 조직이론

현대조직이론에서는 조직현상의 복잡성을 연구하기 위해 조직구조, 조직문화, 리더십, 의사소통, 조직관리 등 조직에서 파생되어 나오는 다양한 개념들간의 상관관계, 그리고 그에 따른 특성들을 확인하는 연구가 있다.

2.1.1. 조직구조에 대한 연구

조직구조는 크게 *공식화, 집권화, 복잡성*에 따라 분류를 하는데 이러한 각 특성이 조직효과성에 미치는 영향을 분석한 연구[10]가 있다. 또한 유기적인 조직구조의 유형으로 기능조직, 프로젝트 전담조직, 매트릭스 조직, 사업부 조직 등이 있는데 특히 매트릭스 조직의 경우 기능조직과 프로젝트 전담조직의 장점을 고루 취할 수 있지만 실질적 운영에 많은 어려움이 있어 이에 대한 대책 연구들이 있다.[11][12]

2.1.2. 조직문화에 대한 연구

조직문화는 개인과 조직의 태도와 행동에 영향을 주는 공유된 가치와 규범을 의미하는데 그 범위와 영향력, 그리고 조직의 다른 특성들과 매우 밀접하게 연관되어 있어 다른 개념들과 복합적으로 연구되는 사례가 많다. 리더십과 함께 조직문화가 업무만족도에 상당한 영향을 미친다는 연구[13]도 있으며 성과지향적 조직문화가 조직효과성에 부정적 영향을 미친다는

연구도 있다.[14]

2.1.3. 의사소통에 대한 연구

조직에서는 개인과 개인, 개인과 집단, 그리고 집단과 집단 사이에 많은 의사소통이 일어나는데 이러한 의사소통이 원활히 이루어질 때 조직의 성과를 향상시키는 효과가 있다고 한다.[15]

이러한 연구들은 모든 조직에 보편적으로 적용 가능한 경우도 있으나 특정 산업분야에 한정된 연구들도 상당하다. 앞서 언급된 바와 같이 조직현상에 영향을 미치는 요인들은 매우 다양하여 특정 산업 분야에서 나타난 현상이 다른 산업 분야에서는 다르게 나타날 가능성도 배제할 수는 없을 것이다.

2.2. 소프트웨어 개발 조직에 관한 연구

소프트웨어 산업은 지식산업이라는 특징 때문에 조직구조, 조직문화, 의사소통 등 외에도 인적 자원에 대한 운용, 양성방안, 지식관리 등의 연구가 두드러진다.

2.2.1. 인적자원 운용에 관한 연구

한 팀의 규모, 즉 몇 명의 인원이 팀을 구성했을 때 가장 탁월한 성과를 나타내는가를 분석한 연구들에서는 대체적으로 소수 인원으로 구성되었을 때 생산성도 높게 나타나며 팀원간 상호작용도 잘 이루어진다고 한다. [16][17][18] 또한 개발단계에 따라 인력을 효율적으로 배분하는 방안에 대한 연구도 있다. [19]

2.2.2. 양성 / 지식관리에 관한 연구

소프트웨어 개발 과정을 하나의 복잡계로 인식하여 소프트웨어 개발 조직 구성원이 창발적으로 발전해 나가도록 접근방법을 달리해야 한다는 연구[20]가 있으며 소프트웨어 개발 조직이 가진 지적 자산의 중요성을 부각시키며 어떻게 관리해야 하는지에 대한 연구도 있다. [21][22]

이 외에도 소프트웨어 개발조직에 매트릭스 조직을 적용하였을 때의 조직운영 방안에 관한 연구[23], 소프트웨어 개발 조직의 복잡도를 계량하여 소프트웨어 품질을 확인하는 연구[8], 그리고 조직 내 의사소통에 미치는 요인들에 대한 연구[9] 등이 있다. 하지만 이러한 연구들은 조직 전체를 기준으로 한 거시적 관점에서 다루어져 조직구성원들이 체감하게 되는 문제점들, 그리고 그것에 따른 조직의 특성들을 잘 보여주지 못한다.

2.3. 용어 설명

조직 이론: 조직의 구조를 비롯한 조직의 요소를 연구하여 조직을 보다 바람직하게 유지·발전시키는데 도움을 주고자 체계화한 이론

조직 효과성: 체제로서 조직의 능력을 활용하여 조직의 합의된 목표(혹은 기능)를 달성한 정도

조직 개발: 조직 효과성 향상을 목적으로 조직의 전략, 구조 및 과정을 계획적으로 개발하고 강화하는 데에 있어 시스템 전반적으로 행동과학지식을 적용하는 것을 의미

조직의 영속성: 조직이 계속 유지되는 정도

조직의 공식화: 조직 내의 직무가 표준화되어 있는 수준

조직의 집권화: 조직 내에서 의사결정이 어느 위치에서 이루어지는가에 따른 수준

조직의 복잡성: 조직의 분화 정도에 따른 수준

3. 연구 방향

3.1. 연구 목적

소프트웨어 개발 조직에서 구성원들이 느끼는 문제점과 그 수준을 확인하고, 소프트웨어 개발 조직의 특성을 People, Process, Technology 세 가지 관점에서 해석한다. 이를 통해 소프트웨어 개발 조직에서 나타나는 조직현상의 복잡성을 이해한다.

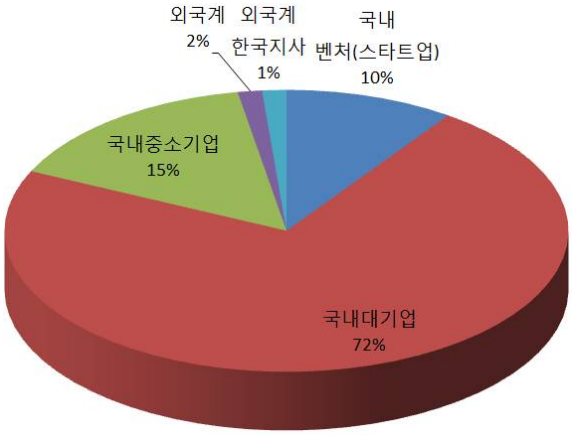
3.2. 연구 방법

다양한 소프트웨어 개발 조직의 구성원들로부터 정보를 수집하기 위하여 온라인 설문조사를 진행하였다.

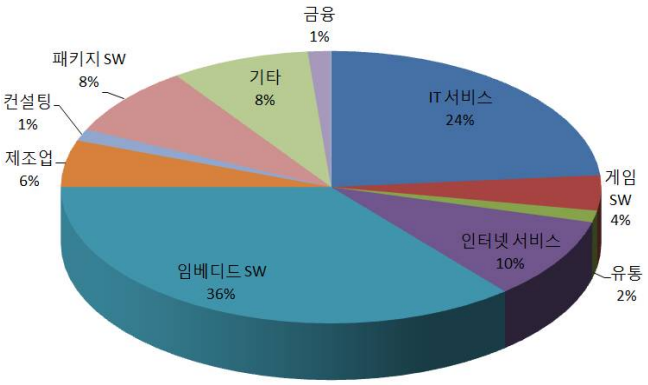
설문의 신뢰성을 높이기 위해 응답자에게는 추첨을 통해 인센티브를 제공하였으며 개발자들로 구성된 커뮤니티를 통해 설문을 실시하였다.

3.3. 설문 대상

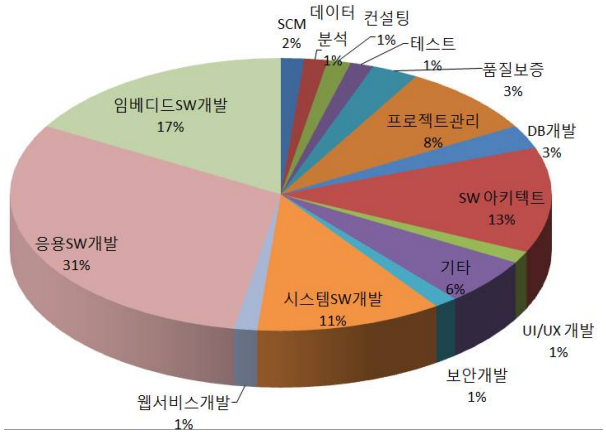
설문조사에 응답한 자는 총 72명이며 국내/외 총 25개 기업의 종사자들로 구성되었다. 기업유형별로 보면 국내 대기업(72%)이 가장 많았으며 국내 중소기업(15%)이 뒤를 이었다. 업종별 응답자 비율은 임베디드 소프트웨어(36%)가 가장 많았으며, IT서비스(24%), 인터넷서비스(10%)가 그 다음으로 많았다. 직무별 응답자 비율은 응용 소프트웨어 개발(31%)이 가장 많았으며 임베디드 소프트웨어 개발(17%), 시스템 소프트웨어 개발(11%)이 그 다음으로 많았다.



[그림 2] 기업유형별 설문대상



[그림 3] 업종별 설문대상



[그림 4] 직무 유형별 설문대상

3.4. 설문 항목

본 연구에서는 소프트웨어 개발 조직에 대해 조직 전체가 아닌 구성원 개인의 관점에서 답변할 수 있도록 설문항목을 설계하였다. 설문지는 '조직문화', '조직구조', '의사소통', '조직효율성', '조직역량', '직무환경' 총 6가지 분야에 대해 총 32개 설문 항목으로 구성되었으며 People, Process, Technology에 해당하는 항목을 조직설계 관점에서 포함시켜 이에 연관되는 조직의 특성들을 관찰할 수 있도록 하였다.

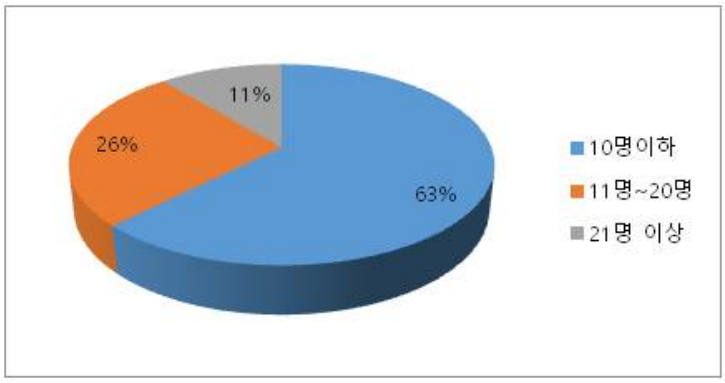
4. 연구 결과

설문조사 결과, 조직의 여러 가지 특성들을 확인할 수 있었으며, 이를 Capability Maturity Model Integration (CMMI)에서 설명하는 소프트웨어 개발의 3가지 요소인 'People', 'Process', 'Technology' 의 관점으로 분석하였다.[7]

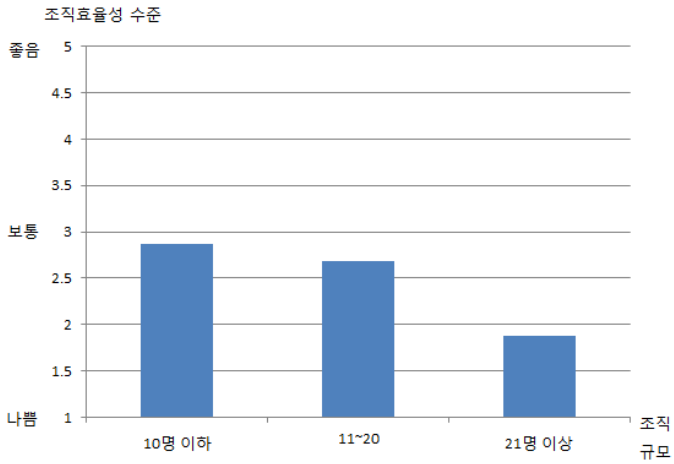
분석된 결과는 크게 두 단락으로 나뉘는데 첫 번째 단락에서는 **위 3요소에 대해 각각 독립적으로 분석**하였으며, 두 번째 단락에서는 **위 3요소의 상호 관계를 파악하기 위해 복합적으로 분석**하였다.

4.1. People

4.1.1. 최소 단위조직 규모



[그림 5] 최소단위 조직 규모에 따른 응답자 비율



[그림 6] 조직규모에 따른 조직 효율성 수준

설문조사 결과 응답자 중 63%는 자신이 속한 최소의 팀 인력이 10명 이하라고 응답하여 소규모로 팀이 구성되는 경우가 많음이 확인되었다. 또한 10명 이하인 조직에서 조직효율성 수준에 대한 점수가 가장 높게 나왔다.

관리자는 조직을 구성할 때 그 조직에 주어지는 역할에 따라 몇 명의 인력이 최적인지 고민을 하게 된다. 많은 전문가들은 상대적으로 작은 팀이 큰 팀에 비해 역할을 수행하는데 효과적이라고 말하고 있는데 대체적으로 3~8명이 소프트웨어 개발에 있어 적절한 인원이라고 주장하였다.[16][17] 이러한 주장은 그림

6의 설문 결과를 통해 다시 한 번 증명되었다. 하지만 그림 6의 전체적인 점수를 보면 보통(3점) 이하로 낮음을 알 수 있어 조직규모에 따라 상대적으로 향상됨을 확인했을 뿐, 절대적인 조직효율성의 수준이 높음을 의미하지는 않는다.

4.1.2. 조직구조

"A Guide to the Project Management Body of Knowledge"에 따르면 대표적인 조직 구조로서 기능조직, 프로젝트 전담조직, 그리고 매트릭스 조직 이렇게 3가지 유형을 설명하고 있다.[24]

하지만 현대적인 조직에서는 조직의 목표, 문화, 환경, 규모 등 다양한 요인에 의해 복합적인 형태를 갖추고 있어 조직 전체에 대해서는 어느 한가지 유형으로 설명하기가 어렵다. 이는 소프트웨어 개발 조직에도 동일하게 적용 된다.

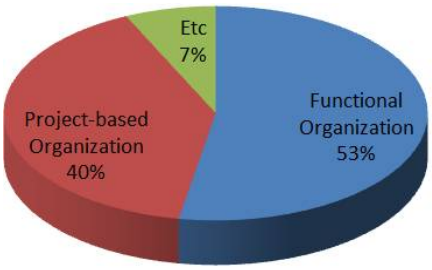
우리는 서론에서 소프트웨어 개발 조직은 인적자원의 활동과 상호관계가 조직의 성공에 큰 영향을 미친다고 했다. 따라서 **개인의 업무 수행에 직접적인 영향을 미치는 단계의 조직, 개인에게 가장 인접한 조직의 단위인 '최소단위 조직'을 집중적으로 연구**하였다.

최소단위 조직을 기준으로 하면 다음과 같은 장점이 있다.

첫째, 조직 전체를 설명하는 것 대비 앞서 언급된 3가지 유형(기능조직, 프로젝트 전담조직, 매트릭스 조직)을 단순화하여 설명하기가 용이하다.

둘째, 최소단위 조직의 유형을 기능조직과 프로젝트 전담조직으로 분류 할 수 있다.

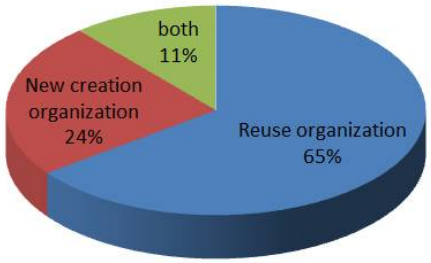
셋째, 조직간의 서로 미치는 영향과 특성을 추가로 확인 할 수 있다.



[그림 7] 조직구조 유형

최소단위 조직의 구성형태는 그림 7과 같이 기능조직(Functional Organization)이 53%. 프로젝트 전담조직(Project-based Organization)이 40%로 기능 조직이 대체적으로 많음을 확인하였다. 여기서 매트릭스 조직 유무는 나타내지 않고 있는데 일부 설문자들이 공식적 매트릭스 조직의 의미에 대해 혼동을 하고 있어 그보다 더 직관적인 의미로 본인의 업무수행에 영향을 미치는 상위관리자가 2명 이상인지 여부를 확인하였다. 이와 관련된 내용은 4.1.3에서 다루도록 한다.

다음 그림 8은 신규 프로젝트를 진행 시 조직을 신규로 구성하여 프로젝트를 진행하는지, 아니면 기존 조직을 활용하여 진행 하는지를 조사한 결과이다.



[그림 8] 신규 프로젝트에 따른 조직재구성 여부

신규 프로젝트 수행 시 새로운 조직을 구성하지 않고 기존 조직을 이용하여 신규 프로젝트를 진행하는 비율이 전체 응답자 대비 65%로 도출 되었다. 이는 각 조직이 해당 분야에 대해 기존에 잘 훈련되어있고 기술이 많이 축적되어 있는 전문가를 활용하여 신규 프로젝트를 진행하려는 것으로 이해된다. 최종적으로 조직은 자신들의 환경에 적합한 가장 효율적인 조직 구성 방식을 택하여 정립한 것으로 해석된다.

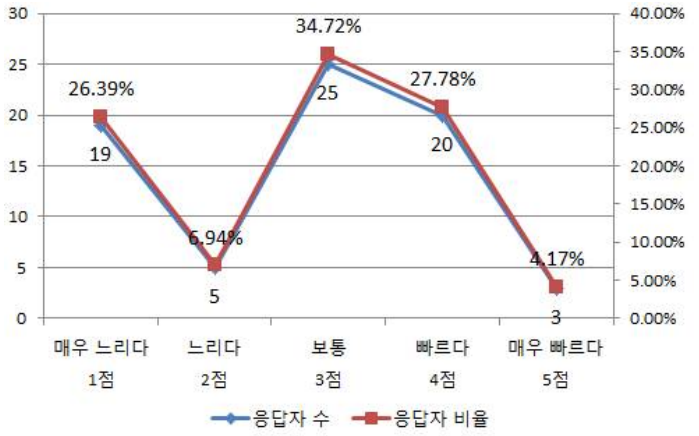
4.1.3. 상위관리자 지배 체계의 모호성

[그림 9] 업무에 영향을 미치는 관리자의 수

최소단위 조직에서는 한 명의 관리자에 의해 그 조직이 관리되는 것이 일반적인 조직의 형태이다. 하지만 조직의 목적, 환경에 따라 여러 명의 관리자에 의해 공식적 또는 비공식적으로 관리될 수도 있다. 프로젝트의 대형화 및 모듈화가 점점 늘어나고 있는 소프트웨어 개발 조직에서는 하나의 최소 단위 조직이 여러 다른 조직들과 긴밀하게 연결되어 협업을 해야 하는 경우가 종종 발생한다. 이러한 상황에서 각 조직 구성원은 본인의 직속 관리자 이외에 다른 조직의 관리자들로부터 업무에 영향을 받기도 한다. 이 경우 매트릭스 조직의 단점인 업무지시의 충돌로 인한 업무 우선순위 결정의 어려움, 보고체계의 중복과 혼선 등의 문제점이 나타날 수 있다.

그림 9는 본인의 직속 관리자 외에 또 다른 관리자로부터 업무에 영향을 받는 정도를 조사한 것으로 약 61%의 응답자가 여러 관리자에 의해 업무에 영향을 받는 것으로 나타나 위에 언급된 문제점들이 잠재되어 있을 가능성이 있는 것으로 해석되었다.

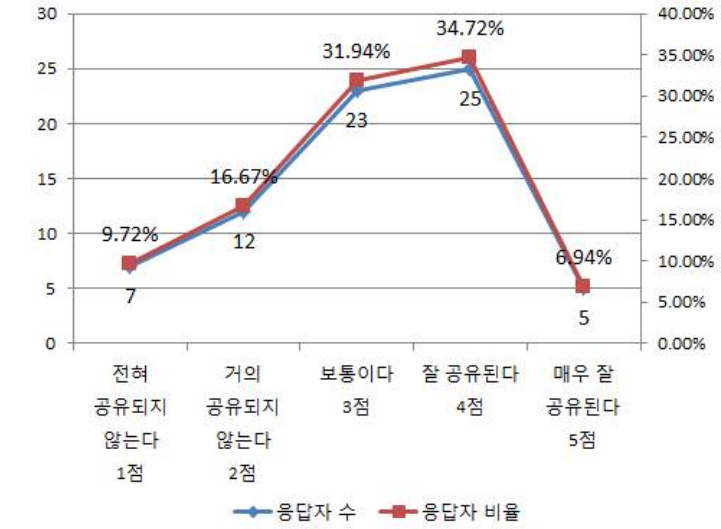
4.1.4. 의사결정속도



[그림 10] 의사결정 속도

그림 10에는 각 조직의 의사결정이 얼마나 빠른가에 대한 응답결과가 나타나있다. 평균점수는 보통(3점)보다 약간 낮은 2.76점으로 나타나 전반적으로 약간 느린 수준으로 확인되었으며 이는 매우 느리다(1점)라고 응답한 수가 상당히 많아 평균 점수를 낮추는 결과를 가져왔다. 몇몇 조직에서 극단적으로 느리다는 평가가 나왔으나 본 설문에서는 원인으로 추정할 만한 특성은 발견할 수 없었다.

4.1.5. 프로젝트 목표 공유 수준

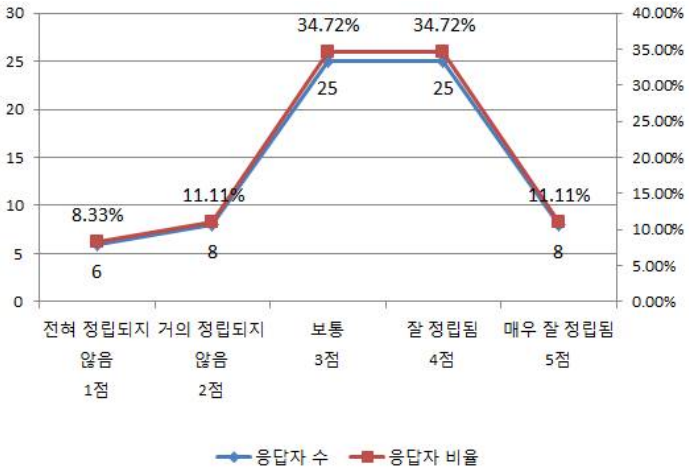


[그림 11] 프로젝트 목표 공유 수준

그림 11에서, 프로젝트 목표에 대한 공유는 전반적으로 잘 이루어지는 것으로 나타났다. 그 외 특별한 특성은 없는 것으로 조사되었다. 프로젝트의 공유 수준을 다른 특성을 가진 집단, 예를 들면 프로세스 정립 수준과 연관 지어 해석하는 것이 특성을 이해하는데 도움이 될 것이다.

4.2. Process

4.2.1. 프로세스 정립 수준



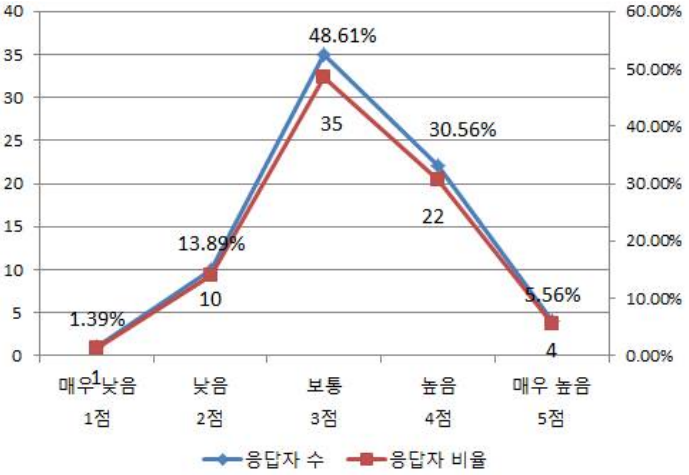
[그림 12] 프로세스 정립 수준

그림 12결과를 보면, 응답자 중 46%(그렇다. 34.72%, 매우 그렇다 11.11%)가 조직 내에 프로세스 정립이 잘 되어 있다고 답하였다. 이는 '매우 아니다(8.33%)'와, '아니다(11.11%)'라고 응답한 그룹 대비 상당히 높은 수준을 보여주고 있다.

소프트웨어가 점점 대형화되고 복잡해짐에 따라 프로세스의 중요성에 대한 인식이 커져서 나타난 결과로 해석된다.

4.3. Technology

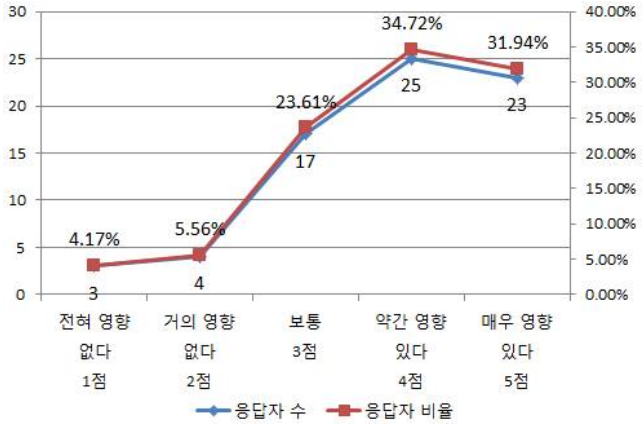
4.3.1. 조직원들의 평균 기술수준



[그림 13] 조직구성원의 평균적 기술수준

그림 13에서 소프트웨어 조직 내 평균 기술 수준이 높다라고 응답한 비율은 약 36%(높다 30.56%, 매우 높다 5.56%)로 기술 수준이 낮다고 응답한 그룹보다 상대적으로 높게 나타나 긍정적인 평가를 보여주고 있다.

4.3.2. 소수 구성원의 기술력이 프로젝트에 미치는 영향도

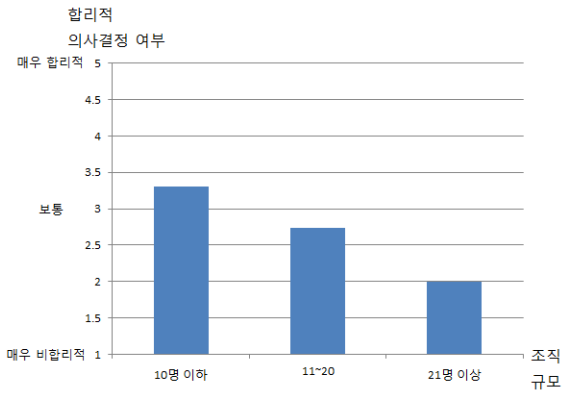


[그림 14] 소수 구성원의 기술 영향력

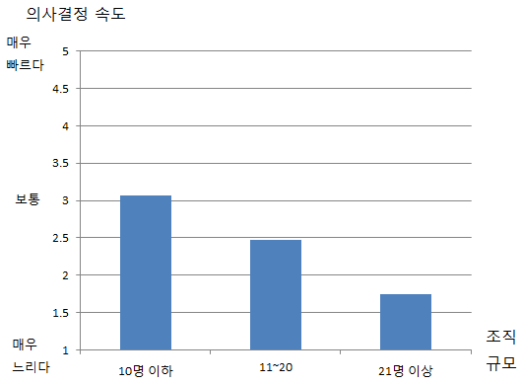
그림 14를 보면 소수의 뛰어난 인력이 프로젝트에 매우 큰 영향을 준다고 답변한 그룹이 약 67%로 매우 높다. 이는 소프트웨어 개발 조직이 기술 집약적이며 인력 자원의 능력, 지식 수준에 크게 의존하고 있다는 점을 보여주고 있어 지식산업의 특징을 잘 나타낸다고 해석할 수 있다.

4.4. People과 Process의 복합적 특성

4.4.1. 최소단위 조직의 규모에 따른 의사결정 효율성



[그림 15] 조직규모에 따른 합리적 의사결정 여부



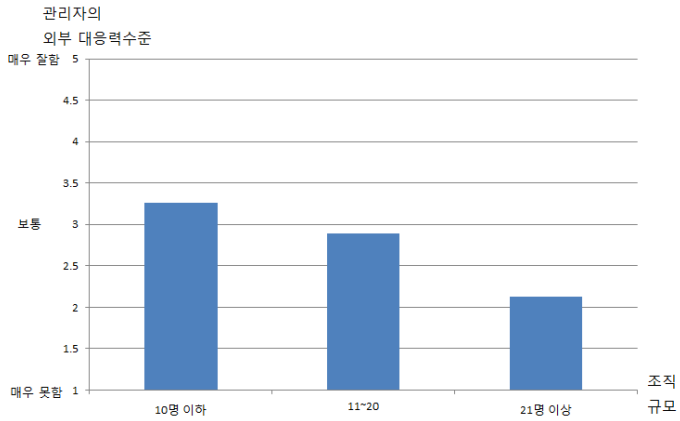
[그림 16] 조직규모에 따른 의사결정 속도

위 결과를 보면 소규모 조직일수록 대규모 조직에 비해 상대적으로 의사 결정 속도가 빠르며, 의사 결정시 조직 구성원들의 의견도 합리적으로 반영되고 있음을 알 수 있다.

이는 조직구성원의 수가 적을수록 의견 교환 및 합의에 걸리는 절대적인 시간이 적기 때문이라 판단할 수 있다. 동일 시간 내에서는 구성원의 수가 많은 조직에 비해 상대적으로 의사결정에 필요한 협력 활동을 더 많이 수행할 수 있기 때문으로 해석된다.

하지만 그림 16에서 나타난 의사결정 속도의 경우 절대적인 수치가 전반적으로 '보통 이하(1.7~3점)' 영역에 분포하기 때문에 의사결정 속도에 영향을 주는 또 다른 요인들이 존재할 것이라고 해석할 수 있다.

4.4.2. 최소단위 조직의 규모에 따른 관리자의 외부 대응 능력 수준

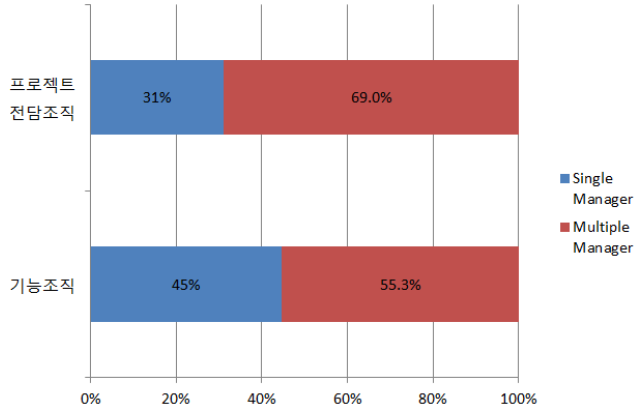


[그림 17] 조직규모에 따른 관리자의 외부 대응 능력 수준

위 그래프는 소규모 조직이 대규모 조직에 비해 상대적으로 관리자의 외부 대응 능력이 높게 평가됨을 보여준다. 따라서 최소단위 조직의 규모가 관리자의 외부 대응력에도 유의미한 영향을 주고 있다고 해석할 수 있다. 이는 조직구성원의 수가 적을수록 관리자가 수행해야 할 조직 내부에서의 활동량이 적어, 상대적으로 외부 대응에 투입할 수 있는 시간적 여유가 많기 때문이라고 해석할 수 있다.

관리자의 외부 대응 능력에 대한 절대적인 수치가 전반적으로 낮은 것으로 보아 조직의 규모 외에도 관리자의 의지, 협상력, 그리고 전문성 등 다양한 요소가 복합적으로 영향을 주었을 것으로 해석된다.

4.4.3. 조직 유형과 상위 관리자 지배체계

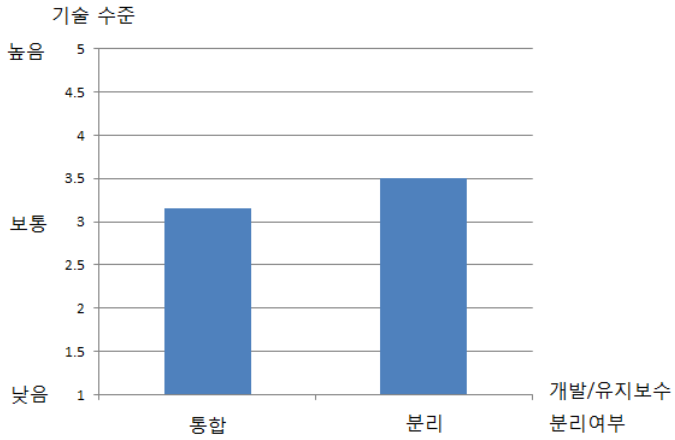


[그림 18] 조직유형별 상위 관리자 지배체계

일반적으로 기능조직이나 프로젝트 전담조직 유형의 최소단위 조직은 다수의 실무자와 한 명의 관리자로 구성된다.[24] 하지만 많은 조직에서 실무자들은 두 명 이상의 관리자로부터 업무에 영향을 받는 경우가 있다.

그림 18은 프로젝트 전담조직과 기능조직에 속한 조직구성원들이 업무 수행 시 직속 관리자 외 다른 관리자로부터 영향을 받는지 여부를 보여준다. 두 조직유형 모두 50% 이상의 응답자가 직속 관리자 외 또 다른 관리자로부터 업무에 영향을 받는다고 답변하였다. 프로젝트 전담조직의 경우 약 69%로 기능조직에 비해 여러 관리자의 영향을 받는 경우가 더 많았다.

4.4.4. 개발/유지보수 분리여부와 조직의 기술수준



[그림 19] 개발/유지보수 팀 분리여부에 따른 조직의 평균 기술수준

소프트웨어 시스템의 규모가 커지면서 나타나는 몇 가지 특징이 있다. 표2. Lehman's Law에 서술된 'Continuing Change', 'Increasing complexity' 등의 특징은 신규개발과 더불어 '유지보수'라는 업무의 필요성을 잘 설명해준다.[25] 신규개발과 유지보수라는 업무는 그 목적이 서로 다르다. 신규개발은 미래에 릴리즈되어야 할 기능이 주 대상이며, 유지보수는 과거에 릴리즈된 기능을 주 대상으로 한다.

그림 19는 신규개발 팀과 유지보수 팀 구성원의 평균적 기술 수준을 조사한 것이다. 개발과 유지보수 팀이 분리되어있는 조직이 분리되지 않은 조직보다 평균 기술수준이 높게 나타났다.

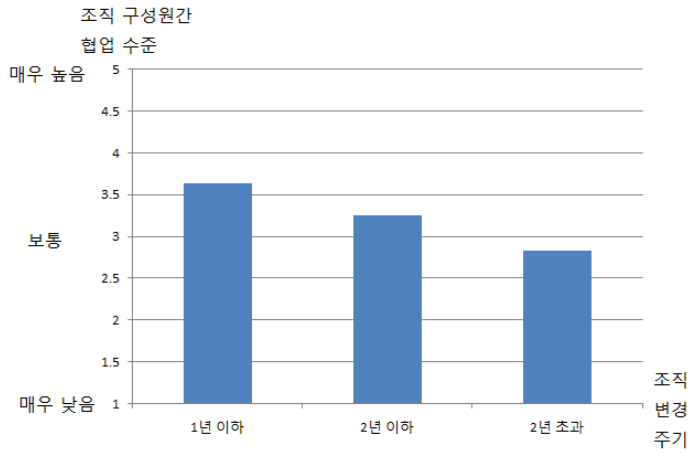
이는 신규개발 팀과 유지보수 팀을 분리했을 때 한 가지 일에 몰입할 수 있는 여건이 갖추어져 구성원 개개인이 기술력을 쌓는데 좀 더 집중할 수 있는 환경을 제공하였기 때문으로 해석된다.

하지만, 평가 결과의 편차가 0.5점차 이내이며, 모두 평균 3점 이상으로 높은 수치를 나타내고 있어 분리 여부의 영향력이 절대적이라고 단정지을 수는 없다.

[표 2] Lehman's Law [25]

1. Continuing Change: 지속적으로 변화하지 않으면 소프트웨어의 유용성이 저하된다. 2. Increasing complexity: 변경이 일어날수록 소프트웨어는 복잡해진다. 3. Large program evolution: 프로그램 사이즈, 릴리즈 주기, 보고되는 에러의 개수는 매 시스템 릴리즈 마다 거의 변하지 않는다. 4. Declining Quality: 운영 환경에 걸맞게 시스템을 변화시키지 않으면 그 시스템은 쇠퇴하게 된다.

4.4.5. 조직의 변경주기와 협업수준



[그림 20] 조직변경주기에 따른 협업수준

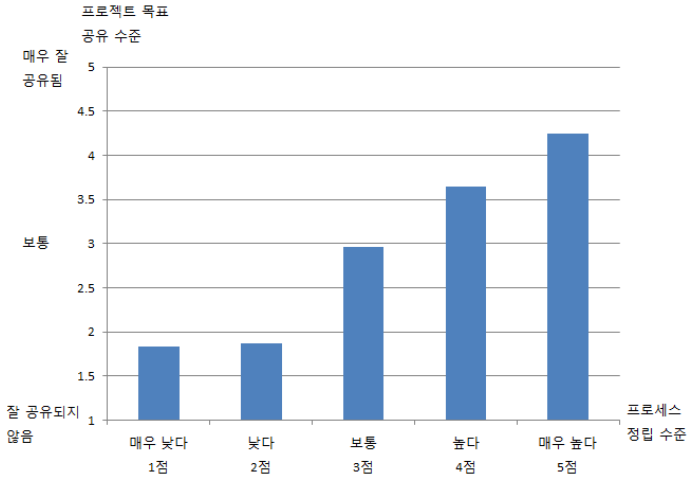
소프트웨어 개발 조직의 산출물은 소프트웨어이며, 소프트웨어는 지속적으로 변화한다는 특징을 가지고 있다. 이런 소프트웨어의 특징으로 인해 소프트웨어 개발 조직은 타 산업의 조직에 비해 상대적으로 짧은 조직변경 주기를 가진다.

그림 20은 조직변경 주기에 따른 조직 구성원간 협업 수준을 조사한 것이다. 조직의 변경 주기에 대한 응답은 최소 1개월부터 최대 7년까지 다양하였다. 조직 변경 주기에 따른 협업 수준 변화도 추이를 살펴보기 위하여, 조직 변경 주기를 1년이하, 1~2년, 2년초과 이렇게 세 가지로 분류하였다.

그 결과 변경 주기가 길어질수록 상대적으로 조직 구성원들과의 협업 수준이 감소하고 있음을 확인할 수 있었다. 이는 변화하는 환경에 적절하게 적응하지 못하면 협업 등과 같은 조직활동이 저하되는 것으로 해석된다.

표2. Lehman's Law에 의하면 소프트웨어는 계속 진화하며 요구사항에 의해 지속적으로 변하는 특징을 가지고 있다. 지속적으로 변화하는 소프트웨어 요구사항에 효율적으로 대응하기 위해 조직도 그에 맞게 지속적으로 변경될 필요성이 있다.

4.4.6. 프로세스 정립 수준과 프로젝트 목표 공유 수준

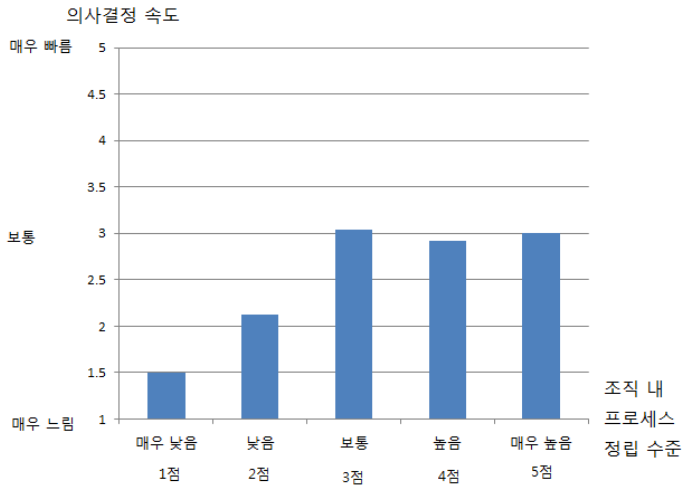


[그림 21] 프로세스 정립 수준과 프로젝트 목표 공유 수준간 상관관계

그림 21은 프로세스의 정립 수준별로 조직 내에서 프로젝트의 목표가 얼마나 잘 공유되는지 조사한 것으로 두 가지 특성이 거의 비례하게 나타나고 있다.

이는 프로세스 정립이 잘 된 조직일수록 체계적인 정보와 표준화된 용어를 이용하기 때문에 프로젝트 목표 공유도 용이하였다고 해석할 수 있다.

4.4.7. 프로세스 정립 수준과 의사결정 속도

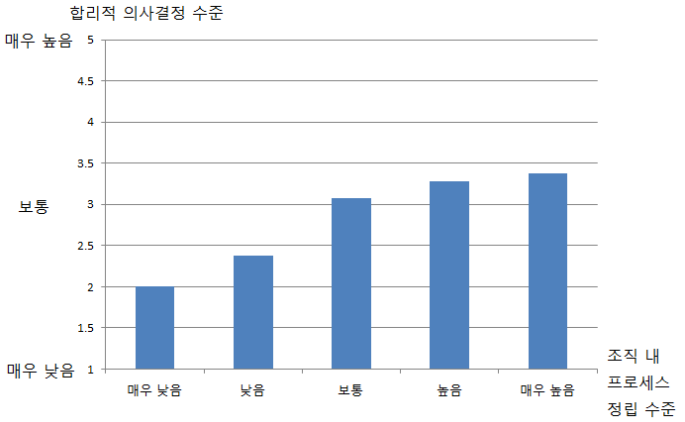


[그림 22] 프로세스 정립 수준과 의사결정 속도간 상관관계

그림 22는 프로세스의 정립 수준별로 조직 내 의사결정 속도가 얼마나 빠른가를 조사한 것이다.

프로세스 정립 수준이 낮은 그룹에서는 조직의 의사결정 속도 역시 비례하게 낮게 나왔다. 하지만 조직의 프로세스 정립 수준이 높은 그룹에서는 의사결정 속도가 모두 보통(3점) 수준으로 나타나 큰 상관관계가 없는 것으로 나타났다. 이는 프로세스 정립수준이 높은 조직의 경우 수행하는 프로젝트의 난이도, 복잡도도 증가하기 때문에 의사결정에 고려해야 할 사항들이 매우 복잡하여 나타나는 것이라고 해석된다.

4.4.8. 프로세스 정립 수준과 합리적 의사결정 수준

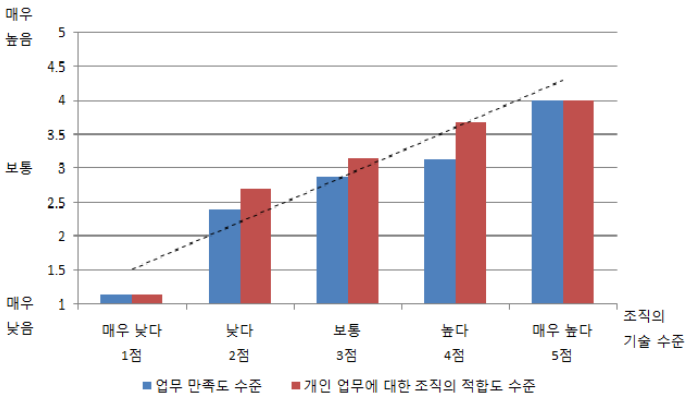


[그림 23] 프로세스 정립 수준과 합리적 의사결정 수준간 상관관계

그림 23는 프로세스의 정립 수준별로 조직 내에서 의사결정이 얼마나 합리적으로 이루어지는가를 조사한 것이다. 프로세스 정립 수준이 높아질수록 의사결정의 합리성도 대체적으로 향상되는 것으로 나타나지만 그 향상 수준은 '낮음'(2점)에서 '약간 높음'(3.3점)으로 완만하게 올라가고 있어 영향력이 큰 편은 아니라고 해석할 수 있다.

4.5. People 과 Technology 의 복합적 특성

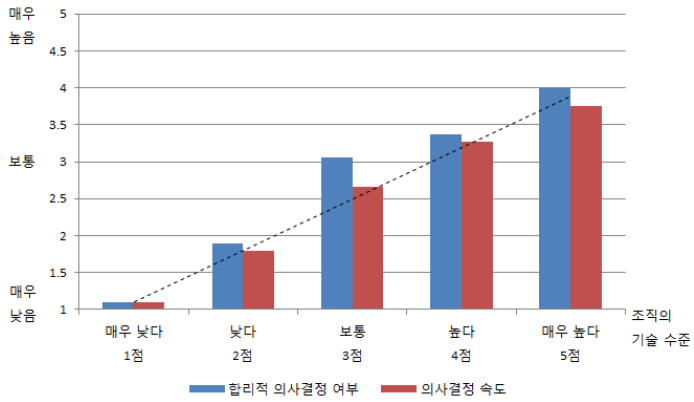
4.5.1. 조직의 평균 기술수준과 업무 만족도



[그림 24] 기술수준과 업무만족도간 상관관계

그림 24은 조직원들의 평균 기술수준에 따른 업무 만족도를 조사한 것이다. 조직원들의 평균 기술수준이 높아질수록 업무 만족도도 높아진다는 것을 볼 수 있다. 또한 업무에 대한 조직의 적합성 여부도 기술수준에 비례하게 증가함을 볼 수 있다.

4.5.2. 조직의 평균 기술수준과 효율적 의사결정

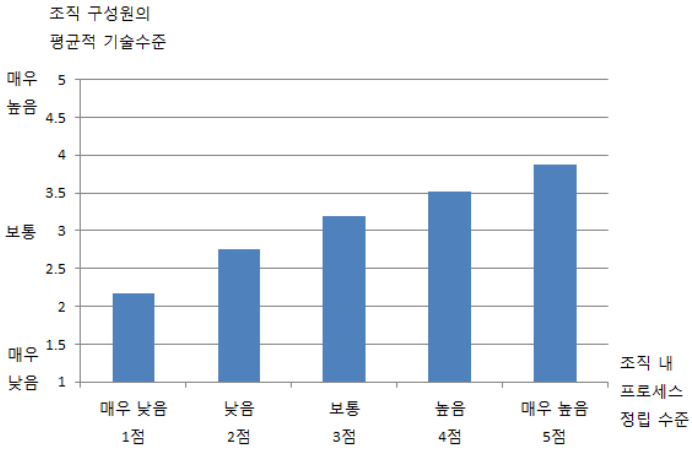


[그림 25] 기술수준과 의사결정간 상관관계

그림 25는 조직원들의 평균 기술수준에 따른 의사결정의 효율성 수준을 나타낸 것이다. 이 역시 그림 24의 업무만족도와 같이 조직의 기술 수준에 비례하여 의사결정의 속도도 빨라지고 보다 합리적인 의사결정이 이루어지는 것으로 나타났다.

4.6. Process 와 Technology 의 복합적 특성

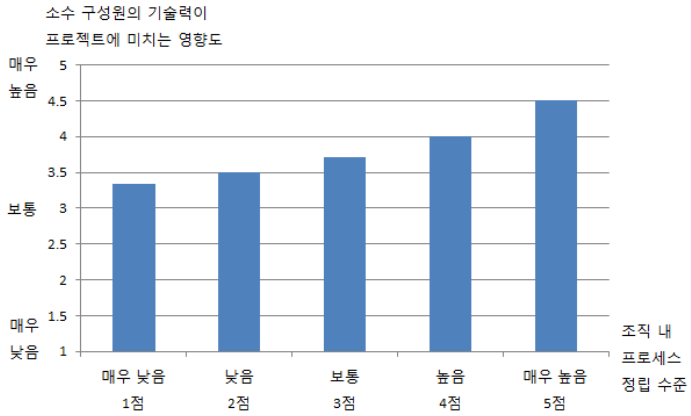
4.6.1. 프로세스 정립 수준과 조직의 평균적 기술 수준



[그림 26] 프로세스 정립 수준과 기술 수준간 상관관계

그림 26은 프로세스의 정립 수준별로 조직 구성원들의 평균적인 기술 수준이 어느 정도인가를 조사한 것이다. 프로세스 정립 수준이 높아짐에 따라 기술 수준도 비례적으로 높아짐을 볼 수 있다.

4.6.2. 프로세스 정립 수준과 소수 구성원의 기술 영향력



[그림 27] 프로세스 정립수준과 소수 구성원의 기술 영향력간 상관관계

그림 27은 프로세스의 정립 수준별로 소수 구성원들의 기술력이 프로젝트에 미치는 영향력을 조사한 것이다. 4.3.2 항의 분석내용에서 볼 수 있듯이 소프트웨어 개발 조직에서는 소수 구성원의 기술력이 미치는 영향력이 매우 크다. 이러한 영향력은 조직의 프로세스 정립 수준이 높아질수록 더욱 강화됨을 확인할 수 있었다.

5. 결론

본 연구를 통해 소프트웨어 개발 조직 구성원들이 조직으로부터 느끼는 다양한 긍정/부정적 특성들을 확인할 수 있었다. 다음 표는 그 특성들을 요약한 것이다.

[표 3] 조직의 긍정/부정적 특성들

분 류		결과 (1~5점 척도 기준)
부정적 평가	조직효율성	평균 2.7점
	의사결정 속도	평균 2.7점
	업무만족도	평균 2.9점
긍정적 평가	소수구성원의 기술 영향력	평균 3.8점
	조직 내 협업 수준	평균 3.5점
	조직구성원의 기술수준	평균 3.3점
	프로세스 정립 수준	평균 3.3점

또한 소프트웨어 개발 조직의 여러 특성들간에 나타나는 다양한 상관관계가 확인되어 실제 소프트웨어 산업 분야가 얼마나 복잡한 조직현상을 나타내고 있는지를 보여주었다.

다음은 특징적으로 나타나는 상관관계들을 요약한 것이다.

- 조직규모가 작을수록 의사결정의 효율성은 높아진다.
- 조직규모가 작을수록 관리자의 외부 대응 능력은 높아진다.
- 조직의 변경주기가 짧을수록 조직의 협업수준은 높아진다.
- 프로세스가 잘 정립되어 있을수록 조직 내에서 프로젝트의 목표가 잘 공유된다.
- 조직구성원의 기술수준이 높을수록 업무만족도도 높아진다.
- 조직구성원의 기술수준이 높을수록 의사결정의 효율성은 높아진다.
- 프로세스가 잘 정립되어 있을수록 조직구성원의 기술수준도 높아진다.
- 프로세스가 잘 정립되어 있을수록 소수구성원의 기술 영향력이 높아진다.

이러한 특성들은 서로 명확하게 인과관계가 규명되지는 않지만 일정한 패턴의 상관관계를 보여주고 있다. People, Process, Technology 의 세 가지 분야에서 각 특성들이 서로 얽혀있는 관계로 나타남으로써 소프트웨어 개발 조직의 성공을 위해서는 이 세 가지 분야가 동시적으로 향상되어야 함이 입증되었다.

이는 소프트웨어 개발 조직에 보편적 조직이론을 적용한다는 것이 상당히 어렵다고 해석될 수 있으며, 그만큼 소프트웨어 개발 조직에 대한 더욱 체계적이고 깊이 있는 조직학적 연구가 필요함을 말해주는 결과이기도 하다.

본 연구에서 확인된 소프트웨어 개발 조직의 특성들간에 나타난 상관관계들을 각각 심층 분석하여 어떠한 특성이 원인이고 어떠한 특성이 결과인지 그 인과관계를 해석해낸다면 여러 소프트웨어 기업들이 각각의 목표에 맞게 조직을 개선하고자 할 때 매우 활용도가 높은 가이드라인이 될 수 있을 것이다.

6. 향후 연구방향

본 논문에서는 1차로 소프트웨어 개발 조직의 특성과 관련된 요소를 항목화하고 이에 대해 상관 관계가 있는 요소를 설문 조사 결과를 기반으로 확인하였다.

2차 연구 과제에서는 각 조직 별 특성과 상관 관계를 분석하여 그 인과 관계를 규명해 보고자 한다. 또한 소프트웨어 개발 구성원 자신이 속한 조직이 어떤 위치에 있고 어떤 특성이 있는지 확인할 수 있는 가이드라인을 제시하고자 한다.

3차 연구과제로는 특정 소프트웨어 조직에 맞춤형 대안 조직을 적용하여 그 효과성을 확인하는 연구를 할 예정이다.

7. 참고 문헌

[1] 오석홍, “조직이론(제8판)”, ISBN 9791130300108 2014.

[2] HODGE, Billy J. “Organization theory: an environmental approach Instructor's manual”... 1979.

[3] POWELL, Walter W.; SNELLMAN, Kaisa. “The knowledge economy. Annual review of sociology”, 199-220, 2004.

[4] SMITH, K. H. “What is the Knowledge Economy'? Knowledge intensity and distributed knowledge bases”, 2002.

[5] 김익환. “글로벌 소프트웨어를 말하다”, ISBN 9788968481031, 2014.

[6] SPRi 최무이, “세계SW시장의 위상”, (https://spri.kr/post/4535), 2016.

[7] CURTIS, Bill; HEFLEY, William E.; MILLER, Sally. “Overview of the People Capability Maturity Model”. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1995.

[8] NAGAPPAN, Nachiappan; MURPHY, Brendan; BASILI, Victor. “The influence of organizational structure on software quality: an empirical case study. In: Proceedings of the 30th international conference on Software engineering”. ACM, p. 521-530, 2008.

[9] SEAMAN, Carolyn B.; BASILI, Victor R. “Communication and organization in software development: an empirical study”. IBM Systems Journal, 36.4: 550-563, 1997.

[10] 김태룡. "일반논문: 조직구조와 조직효과성: 관계와 재해석." 행정논총 54.1: 1-29, 2016.
[11] GALBRAITH, Jay R. “Matrix organization designs How to combine functional and project forms”. Business horizons, 14.1: 29-40, 1971
[12] 장성근, “신제품개발에 매트릭스 조직의 강점 살리려면”, LG Business Insight, 2014.
[13] CHANG, Su-Chao; LEE, Ming-Shing. “A study on relationship among leadership, organizational culture, the operation of learning organization and employees' job satisfaction. The learning organization”, 14.2: 155-185, 2007.
[14] 박순애; 오현주. “성과지향적 조직문화와 조직효과성”. 한국행정학보, 40.4: 225-252, 2006.
[15] 조경식; 이양수. “조직간 의사소통이 조직성과에 미치는 영향에 관한 실증적 연구”. 한국행정학보, 42.1: 229-252, 2008.
[16] 이상운; 박중양; 박재흥. “소프트웨어 공학: 개발과 유지보수 프로젝트의 이상적인 팀 규모”.
정보처리학회논문지 D, 10.1: 77-84, 2003.
[17] 박석규; 이상운. “기능점수 기반 소프트웨어 개발팀 규모와 개발기간 예측모델”, 정보처리학회, 제10D권, 제7호, 1127-1136, 2003
[18] RODRIGUEZ, D., et al. “Empirical findings on team size and productivity in software development. Journal of Systems and Software”, 85.3: 562-570, 2012.
[19] 박석규. “소프트웨어 개발인력 배분 모델”. 인터넷정보학회논문지, 8.2: 23-31, 2007.
[20] 오상인. “창발적 소프트웨어 개발 조직과 사람 양성 방법”. 정보과학회지, 30.3: 35-46, 2012.
[21] RUS, Ioana; LINDVALL, Mikael. “Knowledge management in software engineering”. IEEE software, 19.3: 26, 2002.
[22] ACTON, Thomas; GOLDEN, Willie. “Training the knowledge worker: a descriptive study of training practices in Irish software companies. Journal of European Industrial Training”, 27.2/3/4: 137-146, 2003.
[23] 양미나; 이건호. “매트릭스조직의 소프트웨어 개발 스케쥴링”. 한국경영과학회 2006년 추계학술대회 논문집, 67-70, 2006.
[24] Project Management Institute. “A Guide to the Project Management Body of Knowledge”,
ISBN 9781935589679, 2013.
[25] LEHMAN, Meir M. On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software, 1: 213-221, 1979

소비되지 않은 출력값(UTXO) 추출을 통한 블록체인 사이즈 축소 방안

이 동 영 O, 박 수 용
서강대학교
Freesam02@naver.com, Sypark@sogang.ac.kr

Reduction of BlockChain Size By UTXO Extraction

Dong Young Lee O, Soo Yong Park
Sogang University

요 약

비트코인 블록체인이 출시된 후 7년 동안 많은 발전을 이루어 내었다. 비트코인의 활성화와 함께 거래량이 증가하고, 이에 따라 전체 블록체인 사이즈 또한 현재 약 90GB로 증가하였다. 전체적인 관점에서 블록체인 사이즈 증가를 통한 데이터 저장 공간 과사용은 합의를 위한 에너지 낭비와 함께 자원 낭비의 하나로 뽑히고 있다. 본 연구에서는 데이터 저장 공간 낭비를 해결하기 위하여 UTXO(Unspent transaction Output) 추출 기법을 통한 블록체인 사이즈 축소 방안을 소개한다.

1. 서 론

비트코인은 2009년 사토시 나카모토에 의해 개발 되었다. 최초의 비잔틴 장군 문제를 컴퓨터로 해결한 기법인 PoW(Proof of Work) Consensus를 적용하여 분산된 사용자 간의 신뢰도 확보를 이루어내 각광 받게 되었다. 분산된 사용자간의 신뢰 구축이라는 특징은 특히 금융 분야에서 각광 받게 되었고, 최근 R3CEV, ChinaLedger등 세계 각국의 금융기업들이 연합하여 컨서시움을 구성할 만큼 블록체인에 대한 관심이 커지고 있는 상황이다. 세계 기업과 사용자들의 관심과 함께 비트코인 관련 사업들이 출시 되었다. 현재 많은 기업(Amazon, steam 등)에서 비트코인을 통한 결제 시스템을 제공 하고 있으며, 세계 여행시 환전 대신 비트코인을 통해 다른 국가에서 결제 및 환전을 진행하는 추세이다.

아래 그림1 과 같이 비트코인 총 거래 건수는 2013년 이후로 급격하게 증가 하고 있으며, 현재 약 1억 8천만개의 거래가 이루어졌고 최근 1년동안 거래량이 약 8천만개로 80%정도를 차지하고 있으며 이러한 증가 속도는 해가 지날수록 사용자의 증가와 함께 더 빨라질 꺼라 예상 되어 지고 있다.

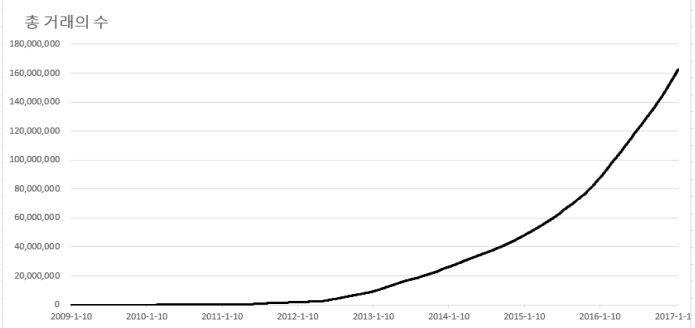


그림1. 비트코인 총 거래 건수

모든 거래는 블록체인에 담겨야 한다는 비트코인 블록체인의 개념에 따라 거래 건수의 기하급수적인 증가는 블록체인 크기의 증가와 비례하게 된다. 그림 2 에서 보이는 것처럼 따라서 2013 년 중순 10GB 에 불가 했던 블록체인 사이즈는 현재 100GB 에 가까워지고 있는 상황이다.

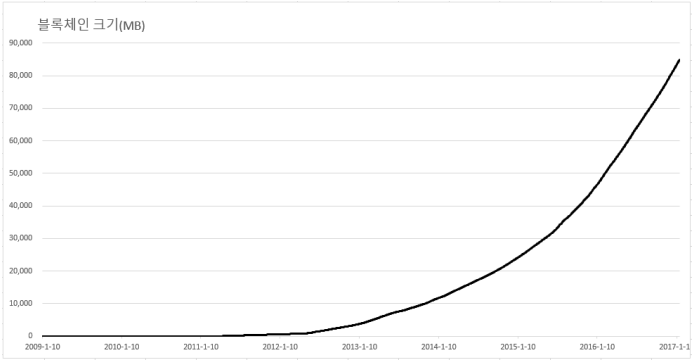


그림2. 블록체인 크기

위에서 말했듯이 그림 1 과 같이 비트코인의 거래 건수가 급격히 증가 함에 따라 하나의 블록(1MB)에 들어갈 수 있는 거래의 수(대략 2000 개)로 한정 되어 10 분안(Confirmation 속도)에 처리 되지 못하고 다음 블록으로 넘어가 2~3 배의 속도로 처리되고 있다. 그리고 처리 속도를 빠르게 하기 위해서 사용자는 더 비싼 수수료를 지불하고 거래를 진행하게 되었고 이에 따라 최근 평균 수수료가 150 원 이상이 되어 비트코인 결제 시스템의 장점인 마이크로 페이먼트가 불가능하게 되었다. 이에 대한 해결책으로 'Bitcoin-Xt'가 개발 및 출시되었다. 'Bitcoin-Xt'란 비트코인 거래 평균 처리량을 증가시키기 위하여 기존 1MB 인 블록 사이즈를 매년 두 배씩 2MB, 4MB, …로 증가시키는

구조로 되어있다. 하지만 위에서 말했던 것처럼 거래량의 증가와 블록체인 크기는 비례 함으로 블록체인 크기의 증가 속도 또한 2 배, 4 배, … 로 증가할 수 밖에 없다.

결국 장기적인 관점에서 블록체인의 크기를 축소시키지 못한다면 Full Node 를 사용하는 마이너, 사용자의 하드웨어 저장소 구축을 위한 추가적인 비용이 발생하게 되고, 마이닝 비용을 통한 수익을 넘는 구축비용이 발생하게 될 것이다. 마이닝이 없는 비트코인은 합의가 이루어질 수 없는 것이고 결국 무너지게 될 것이다.

따라서 본 논문은 비트코인 블록체인의 특성인 불변성을 유지하면서 블록체인 사이즈를 축소 시킬 방안을 제안한다.

본 논문은 아래와 같이 구성된다. 2 장에서는 비트코인 블록체인의 원리에 대해 기술하고 기존의 블록체인 사이즈 축소 방안 연구에 대해 소개한다. 3 장에서는 본 논문에서 제안하는 블록체인 크기 축소방안을 소개한다. 4 장은 결론을 기술한다.

2. 배경 지식 및 기존 연구

2.1 비트코인 블록체인의 원리

비트코인 블록체인은 공공분산장부 개념을 이용해 거래가 이루어진다. 실제로 송금이 이루어질 때, 이에 대한 코인이 실제로 전송되는 것이 아니라 거래 내역을 통해서 코인이 전송됨을 확인하여 거래가 이루어진다. 그래서 거래 내역에 대한 관리가 매우 중요하여 PoW(Proof of Work) 라는 합의 기능이 추가 되어있다. 마이너라고 불리는 사용자들은 'PoW' 라는 합의 알고리즘을 가지고 마이닝을 하여 자신에게 전송된 거래 내역들이 올바른 거래인지 확인하고 이에 대한 보상을 받게 된다. 그리고 확인된 거래들을 하나의 블록에 넣어 모든 노드에게 공표한다. 다른 노드들은 전송 받은 노드가 올바른 블록인지 확인 한 뒤, 기존에 가지고 있던 블록체인에 연결 추가하게 된다. 또한 마이닝을 하기 위해서는 모든 거래의 연결을 확인하여 이 거래가 유효한지 확인하게 되는데 따라서 마이너들은 약 90GB 의 블록체인 전체를 가지고 있어야 한다.

2.2 Merkle Root

Merkle Root는 블록 내에 존재하는 모든 거래의 해쉬값들을 또다시 SHA-256 해쉬 함수를 통해서 해쉬화 시킨다. 최종적으로 Root에 존재하는 해쉬 값은 블록 내에 포함된 모든 거래 값들에 대한 기록이며 Proof of Work를 통해 블록을 생성할 때 이 값을 포함하여 계산을 하기 때문에 위변조가 불가능하다고 하는 것이 비트코인 프로토콜의 목적이다. 따라서 위변조가 발생하게 되면 최종적으로 Merkle Root의 값이 달라지게 되고, 계산 결과 또한 동일하게 나올 수 없는 구조이다.

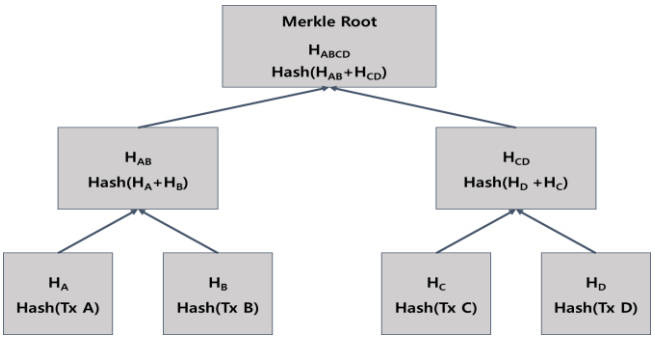


그림3 Merkle Root

2.3 기존연구

비트코인 사용자는 2014 년경부터 블록체인 사이즈에 대한 논의를 시작하였다. 그를 통해 많은 BIP(Bitcoin Improvement Proposal)이 나오게 되었고, 그 중 사용자들간의 합의를 통해서 실제로 동작 중인 방안은 2 가지 이다.

먼저 SPV(Simple Payment Verification)는 마이닝을 수행하지 않고 거래만 사용하는 사용자를 위한 Client 이다. 현재 비트코인의 사용자들의 대부분은 SPV 를 이용한 지갑을 사용한다. SPV 는 Full Node 에 연결되어 단일 거래의 전송 및 검증 기능을 제공한다. 단일 거래의 전송 및 검증이란 사용자가 비트코인의 실제 적합한 주소인지, 잔액 등 거래의 적합성을 판단하는 것이다. 단일 거래의 전송 및 검증에는 블록체인 전체가 필요하지 않음으로 모바일, 웹 지갑용으로 많이 사용되고 있다.

다음으로는 Bitcoin Pruned Mode 는 비트코인 full node 를 위한 데이터 축소 방안이다. 사용자가 원하는 만큼의 블록체인을 가져와 거래 및 마이닝에 이용하는 것이다. 현재 기준 2 일 동안 생성된 블록(약 550MB)을 최소한 가져야 하고 사용자가 원하는 데이터 크기만큼 가져 온 뒤 자료구조 큐의 형태로 새로운 블록이 들어오게 되면 가지고 있는 가장 오래된 블록을 삭제한다. 따라서 사용자가 원하는 블록체인 크기만을 가질 수 있는 장점이 있지만 약 2 주에 한번씩 있는 블록체인 전체에 대한 확인 및 검증이 이루어질 수 없어 모든 블록체인 노드가 Pruned Mode 를 이용할 수 없게 된다.

마지막으로 실제로 적용 중이지는 않지만 제안된 Unused Output Tree in the Blockchain(UOT)은 위에서 말했듯이 SPV 는 거래의 적합성을 판단 할 수 있지만, 정당성을 판단 할 수 없다는 단점을 해결하기 위해 제안된 내용이다. 돈이 전송되기 위해서는 보내는 돈은 다른 곳으로 전송되지 않은 UTXO 이어야 한다. 하지만 블록체인 곳곳에 퍼져있는 UTXO 를 확인 할 수 없는 SPV 는 이에 대한 확인 없이 거래가 이루어지고 full node 에 의해 거래에 대한 정당성 파악이 이루어지게 기다려야 한다. 위와 같은 문제를 해결하기 위해서 SPV client 에 트리 형태로 UTXO 정보를 보관하여 이에 대한

파악을 하는 것이 목적이다. 이를 통해 SPV 를 통한 거래 시 신뢰도가 향상 되게 된다.

3. 분산 저장된 UTXO 에 대한 신규 블록 생성 방안

관련 연구에서 제시한 방안들은 기존 블록체인의 불변성을 깨뜨리지 않기 위하여 추가하는 부분을 통해 데이터 축소하였지만, 이는 근본적인 비트코인 Scalability(확장성)를 해결할 수 없다는 것을 알 수 있다. 비트코인의 Scalability 를 해결하기 위해서는 결국 비트코인 블록체인 자체에 대한 수정이 필요하다는 점을 생각해 볼 수 있다. 이에 대한 시도가 없었던 것은 블록체인 기술의 장점인 Immutable(불변성) 무너뜨리지 않고 블록체인 크기 축소는 불가능 하기 때문이다. 하지만 비트코인 블록체인 기술의 immutable 이 가능한 것은 Merkle Tree Hash 를 통해 과거의 기록에 대한 수정이 불가능하게 만들었기 때문이다. 결국 Merkle Tree Hash 의 값이 기존과 동일하게 유지시키면서 다음 블록을 생성시킬 수 있다면 무결성을 무너뜨리지 않고 블록체인 크기 축소가 가능하게 되는 것이다.

조사에 따르면 현재 비트코인 거래의 약 23%가 UTXO 이며 투자를 위해 비트코인을 구매한 경우가 많은 현재의 특성상 2009 년에 생성된 블록에서부터 지금 생성된 블록까지 널리 퍼져있다. 또한 비트코인의 특성상 과거의 기록을 삭제 할 수 없는데 가장 큰 이유는 바로 UTXO 가 모든 블록에 널리 퍼져 있기 때문이다. 이 기록이 모든 블록체인 상에서 사라지게 된다면 그 돈의 출처 여부를 확인 할 수 없게 되고 검증 또한 이루어질 수 없게 되는 것이다.

본 논문에서는 위와 같이 과거의 블록에 저장되어 있는 UTXO 를 추출하여 새로운 신규 블록을 생성하고 이를 통해 블록체인 사이즈를 줄이는 방안을 소개할 것이다.

먼저 Block 마다 퍼져 있는 UTXO 들을 새로운 BLOCK 으로 생성 한다. 이를 위해서 다음과 같은 프로세스로 생성이 진행된다.

1. 각 Full Node 는 첫 번째 Block(Genesis Block) 에서 가장 최근까지 생성된 Block 에 포함된 UTXO 들을 확인하여 Sender 주소를 Receiver 주소로 변경 후 축소용 DB(mempool)에 추가한다.
2. Full Node 는 축소용 DB 에 저장된 UTXO 를 통하여 마이닝을 진행한다. 하나의 Block 에 들어갈 UTXO 의 개수는 1500 개로 한정한다. 또한 UTXO 의 Timestamp 에 따른 순서로 1500 개씩 구분 짓는다.
3. 마이닝에 성공한 Node 는 새로운 Block 을 공표하고 다른 Node 에 전송한다. 다른 Node 는 새로운 Block 을 확인 후 그 이후의 1500 개에 대한 Mining 을 실행한다.
4. 모든 UTXO 가 처리되고 나면처리 과정 중 전송 받은 기존 거래 시스템의 거래 내역들에 대해서 마이닝 작업을 시작하면 된다.

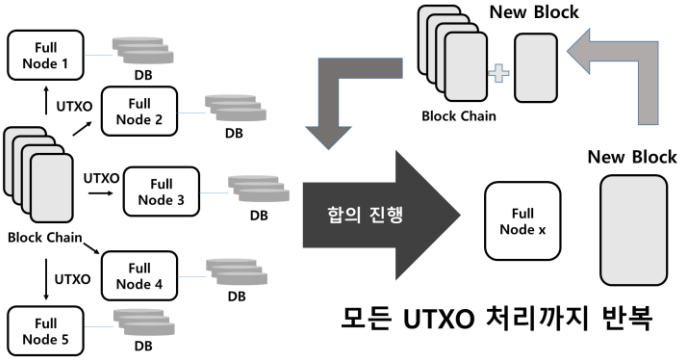


그림4 UTXO 추출 및 신규블록 생성 프로세스

위의 프로세스 진행 후 그림 4 과 같은 형태로 블록체인은 변경된다.

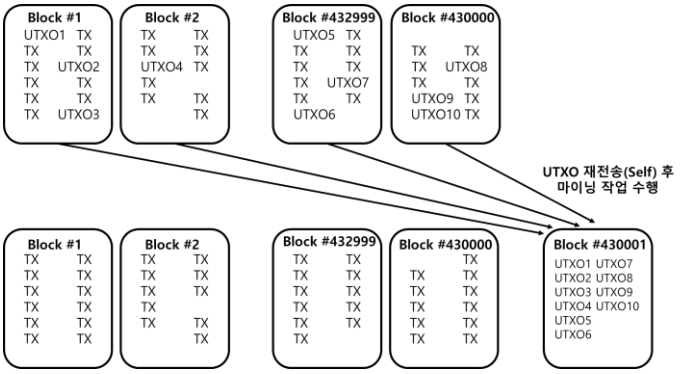


그림5 UTXO 추출 전후 비교

위와 같은 진행을 통해 진행하게 되면 기존에 있던 거래들의 변화가 없음으로 거래들의 Hash 인 Merkle Root 값의 변화가 없게 되어 기존 거래들에 대한 신뢰도를 유지 할 수 있게 된다. UTXO 추출 기법을 통해 추출된 거래들에 대해서는 금액 손실 방지를 위해서 전송 수수료를 0 으로 전송하게 되는데, 기존 DB(mempool)과 다른 DB 를 통해 진행되어 비트코인 채굴 시 존재하는 거래 우선순위 전략을 피해 진행 할 수 있게 된다.

UTXO 를 통한 새로운 블록체인이 구성되고 나면 이미 검증이 된 블록체인 (전체 블록체인의 약 70%)는 기록확인용 일뿐 사용자가 저장할 필요가 없게 된다. 따라서 확인이 된 블록에 대해서는 사용자가 제거를 하여도 비트코인 이용(거래 및 마이닝)에 문제가 없지만, 사용자의 선택 및 기록을 통한 서비스를 제공하는 기업(blockchain.info, blockexplorer.com)의 경우에는 삭제 하지 않고 자료를 보관하는 것이 필요하다.

4. 결론

본 논문에서 제안한 UTXO 추출 기법을 적용한 기대효과를 예상한다면 먼저, 현재 약 4300 만개의 UTXO 가 블록들에 존재하고, 총 블록의 개수는 약 445000 개 정도이다. 따라서 4300 만개 UTXO 를 위에서 언급한대로 1500 개씩 묶어 위의 프로세스를

진행하게 되면 43,000,000 / 1500 = 28667 개의 블록이 신규 생성 되게 된다. 28667 개의 블록의 평균 크기는 대략 0.8MB 임으로 22.39GB 정도의 블록체인만 저장해도 비트코인 전체 시스템이 동작하는데 문제가 없게 되는 것이다. 따라서 현재 95GB 의 블록체인 크기에서 22.39GB 로 23% 축소가 가능하게 된다.

하지만 이런 축소를 위해서는 주기적인 기존 거래 시스템의 정지가 필요하다. Full Node 들이 거래 처리와 동시에 UTXO 추출 시 혼선이 발생할 가능성이 생기게 되고 그로 인해 리플라이 어택과 같은 공격 성공 가능성이 증가할 수 있기 때문이다. 따라서 은행 점검시간과 같은 시간을 두어 데이터 축소 시간을 마련해야 한다. 그렇지만 은행 공동점검 시간과의 차이점으로는 은행 점검시간의 경우 전송 및 수신 자체가 불가능 하지만 제안한 방안의 경우 전송 및 수신을 통하여 외부 노드에서 보내는 정보는 기존 mempool 에 저장하는 시스템은 그대로 이용할 수 있음으로 Confirmation 의 시간이 늘어나는 것일 뿐 거래 시스템의 완전한 정지는 아니다. 따라서 사용자는 거래 시스템의 정지 보다는 블록체인의 Confirmation 지연이라고 느끼게 할 수 있다.

이 방안을 바로 적용을 할 경우 모든 노드들이 동일하게 사용해야 함으로, 소프트포크가 필요하게 된다. 현재 진행중인 비트코인 소프트포크의 경우 모든 노드 중 70% 가 이 제안을 받아들여 새로운 버전의 Bitcoin-core 프로그램을 업데이트 해야 성립이 이루어지게 된다. 그리고 초기 28667 개의 새로운 블록을 구성하는데도 많은 시간이 필요하게 됨으로 사용자들의 동의를 얻는 설득이 필수적이다.

이러한 방향이 초기에는 설득력을 가지기 힘들지만 추후로 보았을 때 하루 생성되는 UTXO 의 개수는 일정하게 증가하게 되는 현재 추세를 볼 때 짧은 시간(한시간 미만)으로 가능할 것으로 예상되며, 또한 블록 반감기로 인해 마이닝 보상값(현재 12.5BTC+ 거래 수수료)이 지속적으로 감소하게 되고 이로 인한 마이너들의 수익 감소 및 마이닝에 대한 의욕 상실에 대한 해결책으로 이용 가능 할 것이라고 본다.

향후 지속적인 연구 발전을 통하여 보다 효율적인 UTXO 추출 및 관리 기법을 적용하고 또한 UTXO 가 추출되고 남은 77%의 블록에 대해서 압축, 파일 형태 변형들 유지 관리 할 수 있는 기법에 대해 연구하고자 한다.

참고문헌

[1] Nakamoto, Satoshi, “Bitcoin: A peer-to-peer electronic cash system”, Consulted 1, 2012(2008):28.

[2] O’Reilly Chimera, “Mastering Bitcoin”,111~117,164~172,188 , 2014

[3] Karl J. O’Dwyer, “Bitcoin Mining and its Energy Footprint”, ISSC, 2014

[4] Simon Barber, Xavier Boyer, “Bitter to Better – How to Make Bitcoin a Better Currency”, International Conference on Financial Cryptography and Data Security, 2012

[5] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gun Sirer, Dawn Song, and Roger Wattenhofer, “On Scaling Decentralized Blockchains”, International Conference on Financial Cryptography and Data Security, 2016

[6] Arthur Gervals, Srdjan Capjun, “Is Bitcoin a Decentralized Currency?”, IEEE Security & Privacy, 2014

[7] Cyrus Malekpour, “Pruning”, http://bitcoin-class.org/projects/pruning.pdf

[8] Alberto Torres, "Merkle tree of unspent transactiosn(MTUT), for serverless thin clients and self-verifiable pruned blockchain” , https://en.bitcoin.it/wiki/User:DiThi/MTUT

블록체인 기반 전자투표 시스템

이우승°, 고동휘, 박수용
서강대학교 컴퓨터공학과
whotname@gmail.com

고덕윤
피노텍 글로벌 프로젝트
maniara.k@gmail.com

Electronic Voting System Based on Block Chain

Woo Seung Lee°, Donghwi Go, Sooyong Park
Sogang University

Deokyoon Ko
Finotek Global Project

요 약

투표는 많은 비용을 필요로 할 뿐만 아니라, 조작의 가능성과 투표율도 높아야 한다는 문제점도 존재한다. 현재 투표시스템에서는 이러한 것들을 개선해 나가기 어렵다. 본 논문에서는 선거 4대원칙과 전자투표 요구사항에 대하여 논의한 뒤 블록체인의 무결성과 보안성을 이용하여 투표 결과를 조작할 수 없게 만들고 비대칭키와 믹스 기법을 이용하여 익명성을 보장해주는 전자투표 시스템을 제안한다.

1. 연구배경

선거는 조직의 대표를 선정하는 절차로서 많은 나라에서 지도자를 선택하는 수단으로 사용할 뿐 아니라, 많은 조직에서 다양한 방법으로 사용되고 있다. 이러한 수단으로써 선거는 조작의 위험성으로부터 보호받을 수 있어야 하고, 구성원들의 많은 참여를 최대한 유도할 수 있어야 한다. 뿐만 아니라 선거는 막대한 비용이 필요하다. 이러한 문제들을 가장 발전된 형태인 전자선거를 채택함으로 해결 할 수 있다.

많은 나라에서 나라의 수장을 선출하는데 전자선거를 이용하고 있다. 전자선거를 사용한다면 비용절감을 가져올 뿐 아니라 개표 오류의 최소화와 신뢰성 있는 개표 결과를 제공해주는 장점이 있다. 그러나, 중앙화된 서버에서 서비스를 제공하는 전자선거는 서버 관리 주체에 의해 조작될 수 있는 가능성을 항상 내포한다. 그리고 이러한 사실은 많은 나라에서 전자선거를 채택하는데 큰 걸림돌이 되고 있다.

한국을 살펴보면 1993년부터 국내 전자투표에 대한 연구들이 진행되어 왔다. 하지만 2001년에 나온 터치스크린 방식의 투표시스템 같은 경우, 시스템에 대한 불신, 과다한 비용, 국민적 미합의 등이 겹쳐 도입이 유보되었다[1]

세계적으로는 전체 200여개의 나라 중 이탈리아, 미국, 영국, 독일 등 35여개국이 전자투표를 추진하였는데 각국의 다른 환경으로 성공과 실패요인 또한 다르게 나타나고 있다.[2]

표 1 전자투표 추진국가[2]

국가	주요제도	주관	목적	쟁점
아르헨티나	대통령 중심제, 양원제, 연방제	선거 위원회	투표개선 비용절약	제도개선,, 홍보부족,, 합의 미흡,
호주	입헌군주제 연방제 의무투표제	선거 위원회	투표개선	보안문제 인증문제
벨기에	입헌군주제 내각책임제 연방제 의무투표제	내무부 지방정부	환경개선 빠른 개표	코드문제, 기기이상, 회사문제
캐나다	입헌군주제 연방제	선거 위원회	투표율상승	보안문제, 인증문제
핀란드	대통령중심제	법무부	비용절약 투표율제고	강제투표가능, 보안문제
프랑스	대통령중심제	내무부 지방정부	환경개선 비용절약	신뢰도미흡 보안문제
독일	대통령중심제 연방제	내무부	투표율제고 환경개선	도입신중
영국	입헌군주제 의원내각제	선거 위원회	환경개선 투표율제고	이용률 저조 초기비용
미국	대통령중심제 양원제	연방선거 위원회 지방선거 위원회	환경개선 투표율제고	법 적 소송 안정성 미흡

한국의 선거제도의 경우, 기본적으로 네 가지 원칙을 전제로 한다. 선거의 4대 원칙은 아래와 같이 요약할 수 있다.

표 2 선거 4대 원칙

원 칙	설 명
보통선거	성별, 학벌, 재산 등의 관계없이 일정한 나이가 된 성인들에게 선거권, 피선거권을 부여함을 보장하는 원칙
평등선거	모든 투표자의 표는 동일한 가치와 효력을 갖는다는 원칙
직접선거	선거권자가 직접 투표에 참여해야 한다는 원칙
비밀선거	선거권자가 투표에 참여하여 누구에게 투표했는지 본인만 알 수 있게 해야한다는 원칙

.

전자선거 기술은 선거의 4대 원칙을 지킬 수 있어야 할 뿐 아니라 임의의 공격자로부터 선거 결과가 조작될 수 없어야 한다.

블록체인은 분권화된 한 방향 데이터베이스로서 한번 기록이 되면 조작이나 삭제가 불가능하다. 비트코인은 이러한 블록체인 기술을 이용하여 중앙화된 기관이 없이 분권화되어 전자 화폐가 관리되고 있다.

본 논문에서는 선거의 4대 원칙을 지킬 뿐 아니라 조작이 불가능한 블록체인 기반의 전자 선거 시스템 기법을 제안한다.

본 논문은 아래와 같이 구성된다. 2장에서는 배경지식으로 블록체인에 대해 기술하고, 기존의 전자 선거 시스템을 제안한 연구를 살펴본다. 3장에서는 본 논문에서 제안하는 블록체인을 이용한 전자선거 기법을 소개한다. 4장은 전자 선거의 요구사항을 토대로 본 논문의 제안 기법을 논의하고 5장에서 결론을 기술한다.

2. 배경 지식 및 기존 연구

2.1 블록체인 기술

블록체인 기술(BlockChain Technology)이란 2009년 나온 비트코인(BitCoin)[3]에서 사용된 핵심기술이다. 블록체인은 거래내역들이 블록에 들어가있고 블록들이 체인처럼 길게 일자로 연결된 형태로서 블록체인이라고 불린다.

블록체인은 무결성과 보안성을 가지고 중앙기관이 없는 시스템을 만들 수 있다는데 큰 강점을 가지고 있어서 많은 곳에서 응용되고 있다.

블록체인은 해시함수를 이용한 단 방향 데이터 저장 기술로서, 모든 참여자가 전체 장부를 소유하고 검증하여 위/변조가 불가능하게 만드는 기술이다. 해시함수를 이용하기 때문에 출력값으로 입력값을 찾을 수 없다는 특징을 가지고 있다.

현재 블록체인은 다양한 곳에서 활용되고 있는데 개인정보, 인증서 증명에 사용되고 있고 몇몇 국가에서는 부동산 거래를 블록체인을 이용하여 하겠다고 발표하고 있다.

2.2 해시함수

해시 함수[4]는 임의의 길이의 입력을 고정된 길이의 출력값으로 압축시키는 함수이다. 데이타의 무결성 검증, 메세지 인증 등에 사용 가능하다. 해시 함수는 일방향성과 충돌 회피성이라는 2가지 성질을 만족해야 한다. 먼저, 일방향성은 어떤 출력값의 입력값이 무엇인지 찾는 것이 불가능 해야함을 가리킨다. 그리고 강한 충돌 회피성이란 주어진 입력 값에 대해 출력값이 같은 두 입력값을 임의로 찾는것이 불가능 하다는 것을 가리킨다.

2.3 믹스넷

믹스넷은 1981년 David Chaum[5]에 의해 소개된 것으로 입력값과 출력값 사이의 연관관계를 알 수 없도록 섞어주는 기술이다. 믹스넷은 여러 개의 믹스 서버로 구성되고 초기 입력값 또는 이전 믹스 서버의 출력값을 받아 섞는 과정을 반복한다. 이때 믹스서버가 올바르게 수행되었는지 확인하기 위하여 영지식 증명, RPC(Randomization Parity Check)을 많이 사용한다. 믹스넷을 사용한다면 입력값에 대한 출력값이 무작위로 섞여서 어떤 입력값이 어떤 출력값을 갖는지 확인할 수 없게 된다.

2.4 기존연구

첫번째로 전자투표 프로토콜[6]을 만들었다. 그 프로토콜은 신뢰하는 선거관리센터를 가정하여 투표자의 투표용지 내용을 알 수 없도록 하여 프라이버시를 보호하였으며 Chaum의 부정할 수 없는 도전 및 응답기법을 이용하여 투표자와 집계자 사이의 부정을 증명할 수 없도록 하였다. 또한 투표자와 집계자 사이의 부정이 있더라도 투표과정에서 부정을 증명할 수 있게 하였다. 전체 투표과정은 준비, 등록, 투표, 개표단계로 이루어지고 선거관리센터와 집계자는 게시판을 이용한다. 하지만 신뢰된 선거기관을 가져야하고 영수증을 출력할 수 있는 별도의 기기가 필요하다, 또한 개별 검증은 가능하나 투표 내용 전체가 조작될 경우는 개개인이 확인할 수 없는 한계점이 있다.

두번째로 영수증 발급을 통해서 전자투표의 신뢰성과 편의성을 증대시키는 기법[7]이다. 이 기법은 ElGamal 암호 방식을 이용하여 임의의 난수들을 생성(e'~e'') 시키고 거기서 속하는 난수(e*⊆{e',e''})를 선택하여 검증값으로 사용한다. 투표기는 투표자가 선택한 n개의 e와 함께 e를 생성할 때 사용한 난수w⊆{w',w''}를 영수증

에 출력해준다. 투표자는 출력용지의 e가 투표기 화면에 표시된 것과 동일한지 검증한다. 그런 뒤 투표를 한다. 투표기는 e'와 e'' 가운데 2단계에서 검증값(e*)으로 선택되지 않은 값을 투표값으로 하여 영수증에 출력한다 투표자는 출력된 값이 투표기 화면과 같은지 표시하여 확인, 검증하고 완료되면 게시판에 등록한다. 그러나 임의의 난수값을 생성하여야하고 그 난수값이 사용자 수보다 많아야 한다는 한계점을 가지고 있다.

3. 요구사항 논의

전자투표에는 여러가지 요구사항이 존재한다. 3장에서는 전자투표에 필요한 요구사항들에 대해 알아보겠다.[8]

표 3 전자투표 요구사항[8]

구 분	내 용
완전성	모든 투표자의 유효 투표가 정확하게 집계되어야 하며, 최종 집계에서 제외되는 일이 없어야 한다.
건전성	악의적인 투표자에 의해 선거가 방해되어서는 안되고 최종집계에서 선거에 영향을 주면 안 된다.
비밀성	모든 투표가 비밀이 되어야 한다. 개인의 투표내용으로부터 개인을 추측할 수 없어야 한다
단일성	이중투표가 불가능 해야 한다
적임성	권한이 있는 자 만이 투표할 수 있다.
공정성	투표 중 일부분 결과를 알게 되어서 투표에 영향을 미치는 상황이 없어야 한다
검증성	선거결과를 속일 수 없도록 누구라도 투표 결과를 확인하여 검증해 볼 수 있어야 한다
매표방지	표 거래를 방지해야 한다.

본 논문에서 제안하는 블록체인 기반의 투표 아키텍처는 위의 요구사항을 다음과 같이 만족한다.

완전성의 경우 모든 투표가 공개되기 때문에 유효투표가 최종 집계에서 제외되는 일이 없다.

블록체인의 특징이 무결점과 보안성에 있기 때문에 악의적인 투표자가 선거를 방해하여 최종집계에 영향을 줄 수 없다.

블록체인은 PKI 기반의 암호이기 때문에 공개키만 가지고 누가 누군지 추적하는 것은 불가능하다. 그렇기 때문에 비밀성이 보장된다.

이중적 투표를 검증하기 때문에 단일성도 보장된다.

적임성은 생체정보를 통해 개인이 맞는지 아닌지를 판단하기 때문에 본인이 아니라면 투표에 참여할 수 없다.

모든 투표결과가 공개됨으로써 선거결과를 조작하거나 속일 수 없기 때문에 공정성을 만족시킨다.

매표방지에 대해선 자신의 공개키가 자신인지 확인을 시킬 수 없기 때문에 매표방지가 된다.

하지만 공정성면 에서는 모든 투표결과가 실시간으로 확인되기 때문에 투표에 영향을 미칠 수 있다. 우리는 투표 데이터를 확인 가능하고 전체 데이터에서는 누구에게 투표한지 알지 못하게 하는 비대칭 암호화 알고리즘이나 특정시간에 도달 할 때까지 투표정보를 볼 수 없게 하는 스마트 계약에 대하여 연구하고 추후에 보완을 할 예정이다.

4. 블록체인 기반 전자투표 아키텍처

본 논문에서는 블록체인을 기반의 전자 투표 아키텍처를 제안한다. 기존의 전자 투표 시스템은 해커가 중앙 서버에 침투하여 선거 데이터를 조작할 경우 투표 데이터가 조작될 수 있고 이는 재투표로 이어질 수 있기 때문에 큰 비용이 들 수 있다. 본 논문에서 제안하는 블록체인 기반 전자 투표 아키텍처는 여러 노드 들이 투표 정보를 복제해서 가지고 있기 때문에 충분한 노드가 확보가 된다면 사실상 조작이 불가능하다.

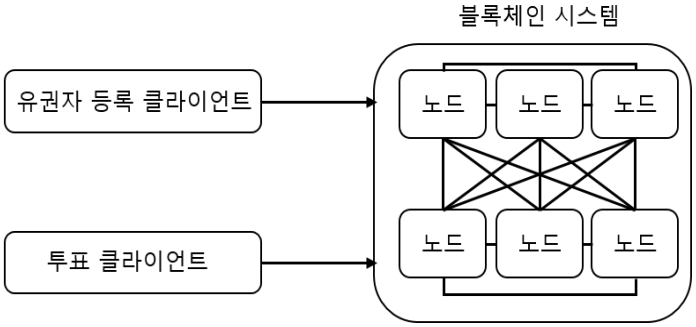


그림 1 블록체인 기반 전자투표 구조

그림 1은 본 연구에서 제안하는 투표시스템의 아키텍처이다. 노드들은 투표결과가 오면 올바른 유권자인지 검증하여 투표 데이터가 들어간 블록을 생성한다. 유권자 등록 클라이언트는 공적인 기관에서 운영하는 유저 등록권한을 가진 클라이언트이며, 자기 자신만의 암호화키와 복호화키를 가지고 있다. 모든 노드 들은 유권자 등록 클라이언트의 복호화키를 가지고 있으며 유권자 등록 클라이언트에서 유권자를 등록하면 복호화키를 이용해 유권자 등록 클라이언트인지 확인한 뒤 유권자 정보를 등록하게 된다. 투표 클라이언트는 사용자가 투표를 하는 클라이언트이고 투표를 하게 되면 자신의 개인키로 암호화 하여 블록체인 시스템에 투표결과를 전송하게 된다.

4.1 유권자 등록

그림 2은 유저를 등록하는 프로세스이다. 유권자들의 생체 정보나 개인정보를 이용하여 공개키를 생성한다. 이때 유권자 등록 클라이언트는 두 데이터를 분리해서

index없이 저장하며, 데이터를 섞는다. 이러한 데이터가 충분히 많은 숫자가 모이게 되면 개인정보를 이용한 공개키 데이터를 암호화하여 블록체인 네트워크에 보낸다. 이때 모든 노드들은 올바른 클라이언트가 보냈는지 투표 개인정보가 겹치지 않는지 확인하여 유권자를 등록한다.

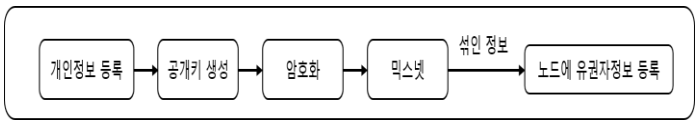


그림 2 유권자 등록 프로세스

4.2 투표

유저는 개인의 핸드폰, PC와 같은 투표 클라이언트를 이용하여 생체정보 인증이나 개인인증을 통해 블록체인 네트워크에 투표를 할 수 있다. 그림 3은 투표 프로세스이다.

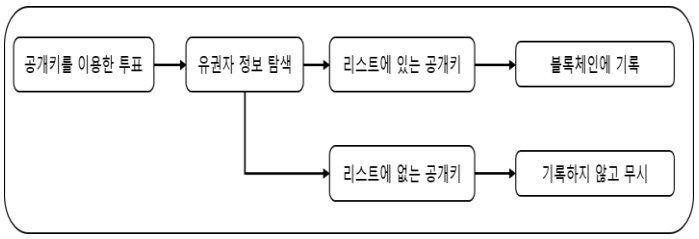


그림 3 투표 프로세스

투표권을 가진 유저는 누구에게 투표할지 정한 이후 자신의 생체정보나 개인정보를 통한 공개키를 이용해 투표 데이터를 암호화한다. 투표 데이터를 전송하고 블록체인 네트워크는 데이터 검증을 하여 올바른 데이터라면 데이터를 저장한다.

5. 결론

본 논문에서는 기존 전자 선거 프로토콜의 조작위험성을 방지하기 위하여 전체 검증과 개인검증이 가능하고 악의적인 단체에서 가상의 투표권자를 만드는 것을 방지하기 위하여 투표권을 가지지 못한 자가 투표가 불가능한 아키텍처를 제안하였다. 블록체인 기반의 아키텍처를 통하여서 유권자 등록과 투표과정을 다수의 노드들이 검증하여 조작이 불가능하게 구성하였다.

본 연구에서 제안하는 아키텍처는 신뢰가 있는 개인등록 클라이언트를 통해 개인과 투표권자를 모르게 하며 가상의 투표권자가 등록하지 못하도록 하였다. 따라서 신뢰도가 있는 기관이 없을 경우 가상의 투표권자가 생겨 투표하는 경우가 생길 수 있다. 또한 투표 기간중에 투표 내용 확인이 가능하기 때문에 유권자들이 상황에 따라 투표를 바꿀 수 있으며 블록체인 기반 아키텍처이기 때문에 노드가 확보되지 않는다면 조작위험성이 증가한다.

우리는 추후 연구를 통해 소수의 노드로 무결성을 가져갈 수 있는 분산합의 알고리즘에 대하여 연구할 것이며, 투표 시간중에는 투표결과를 알 수 없고 일정시간 후에 결과가 공개되는 스마트 컨트렉트에 대해 연구하고, 공개성에 비해 어떤 장점을 갖는지 확인하여 사용자들이 투표 내용을 투표 중에 확인하는 것을 방지할 계획이 있다.

참고문헌

[1] D.Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms,“

[2] 조희정, “해외의 전자투표 추진 현황 연구”, 사회연구 통권, pp. 45 ~ 72, 2007

[3] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." Consulted 1.2012 (2008): 28.

[4] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. 《Handbook of Applied Cryptography》(2001)

[5] D.Chaum, "Untraceable Electronic Mail, Return ddresses, and Digital Pseudonyms,“ Communications of the ACM, vol.24, no.2, pp. 84-88, Feb 1981

[6] 김순석, 이재신, 김성권, “실용적이고 안전한 전자투표 프로토콜에 관한 연구”, 한국정보보호학회, 정보보호학회논문지 10(4), 2000.12, pp. 21-32

[7] 이윤호, 이광우, 박상준, 김승주, 원동호, “향상된 사용자 편의성을 갖는 안전한 전자투표 영수증 발급 방식 한국정보보호학회, 정보보호학회 논문지 17(4), 2007.8, pp. 75-88

[8] 허원근, 김희선, 김광조, “전자선거 프로토콜의 요구사항 연구”, 한국정보보호학회, 정보보호학회지 10(1), 2000.3, pp. 63-69

스마트 계약 기반 탈 중앙 전자 투표 시스템

Decentralized E-Voting System based on Smart Contract

노승학, 인호

고려대학교 컴퓨터학과
서울 성북구 안암로 145
blue619@korea.ac.kr, hoh_in@korea.ac.kr

요약: 본 논문은 기존의 전자투표 시스템에서 문제가 되는 익명성, 투명성의 문제를 해결하기 위한 것이다. 우리는 블록 체인의 진보된 거래 스크립트 기술인 스마트 계약(Smart Contract)을 이용하여 모든 이의 투표가 공정하고 안전하게 처리되는 투표 시스템을 제안한다. 우리는 제안하는 투표 시스템의 투명성을 보장하기 위해 공용 블록체인의 일종인 이더리움(Ethereum)의 스마트 계약 기술을 이용하였으며 신뢰할 수 있는 제 3 의 서비스(Trusted Third Party Service) 개입 없이 RSA 비대칭 키 암호화를 통해 투표의 비밀성을 보장한다.

핵심어: 전자 투표, 모바일 투표, 탈 중앙화 시스템, P2P, 블록 체인, 스마트 계약

1. 연구 배경

대한민국은 현재 전자정부(Electronic Government)의 실현을 통하여 초고속 IT 인프라와 함께 국제적인 영향력과 명성을 얻고 있다. 또한 국민들의 정치참여 수단인 투표 부분에서도 이를 활용한 전자투표(Electronic Voting)의 부분적이고 제도적인 도입이 이루어지고 있다. 전자투표는 90 년대 중반부터 세계 주요 국가들이 도입을 하고 있으며, 현재는 약 50 여 개국이 공직선거에 전자투표를 도입하여 활용하고 있다. 공직선거에 전자투표제도를 도입하는 것은 국가별로 상이한 정치 문화 제도와 시민사회 및 정치권의 이해관계에 따라 도입 수준의 차이를 보이고 있으며, 현재 대한민국에서는 전자투표 시스템에 대한 신뢰부족 등으로 정치권, 예산관계부처, 일부 시민단체의 반대에 부딪혀 원활히 진행되지 못하고 있는 실정이다. 이는 투표 제도가 디지털로 변화되는 과정에서 겪게 되는 정치 사회적 갈등 과정이라 볼 수 있을 것이다[7].

공직 선거의 전자투표 도입은 기존의 종이투표에서 디지털 방식으로 전환되는 단순한 선거관리방식의 변화라기보다 국가 정책의 파급효과가 큰 정치적인 성격이 강한 공공정책으로 평가되어야 한다[8].

따라서 이러한 전자투표 시스템이 유권자의 신뢰를 얻기 위해서는 기술적인 보안성과 안정성이 완벽하게 담보되었을 때 여론의 동의를 얻을 수 있을 것이다.

때문에 본 논문은 공격자, 혹은 시스템 관리자가 개표 결과를 조작하지 못하도록 보안성과 투명성을 보장하는, 또한 투표자의 익명성이 보장되고 이중투표를 막을 수 있는 이더리움의 스마트 계약과 RSA 비대칭 키 암호화 기반 탈 중앙 전자 투표 시스템을 제안한다.

2. 배경 기술

2.1 전자 투표(E-Voting)

전자 투표 방식은 정해진 투표소에서 전자투표를 할 수 있는 방식인 '투표소 전자투표(PSEV, Poll Site E-Voting)'와 투표소에 가지 않고 통신기기를 이용하는 곳이면 어디서든 사용 할 수 있는 '원격 인터넷 투표(REV, Remote Internet E-Voting)' 방식으로 구분된다[9].

본 논문에서는 REV 방식의 전자 투표 시스템을 제안한다. 서버-클라이언트 구조를 채용한다면 알려진 많은 취약점들에 노출되고 클라이언트 변조 등과 같은 악의적인 공격행위에 안전을 보장할 수 없으므로 네트워크 구성에 있어서 P2P(Peer-to-Peer) 구조를 채택한다. 그 중에서도 비트코인(Bitcoin) 네트워크의 투명성과 안정성을 위해 사용되고있는 블록 체인(Blockchain) 기술을 개량, 확장한 이더리움(Ethereum) 플랫폼을 활용함으로써 투표 결과에 대한 조작을 방지하고 클라이언트 위변조를 통한 공격의 가능성을 차단한다. 또한 RSA 비대칭 키 쌍을 이용한 투표내역 암호화를 수행함으로써 비밀 투표를 보장한다.

2.2 블록 체인(Blockchain)

비트코인은 2009 년 나카모토 사토시(Nakamoto Satoshi)가 발표한 디지털 통화의 한 종류이며, 통화를 발행하고 관리하는 중앙 장치가 존재하지 않는 탈 중앙화된 분산 시스템 구조를 가지고 있다. 대신 비트 코인의 거래는 P2P 기반 분산 데이터베이스, 블록 체인에 의해 이루어지며, RSA 기반의 공개키 암호화 방식 기반으로 거래를 수행한다. 비트코인은 지갑 파일 형태로 저장되며 이 지갑에는 각각의 고유 주소가 부여되어 그 주소를 기반으로 비트 코인의 거래가 이루어진다[3].

또한, 서버가 없이 운영되는 블록 체인 데이터는 모든 참여자가 나누어서 가지고 있기 때문에 공격자가 변조된 거래내역으로 다수결 합의에서 승인을 얻기위해서는 전체 컴퓨팅 파워의 51%이상을 점유하고 있지 않는 한 불가능 하다[3]. 결론적으로 하나의 노드가 데이터 변조를 시도 할 지라도 네트워크의 다수 노드와의 경쟁에서 승리 할 수 없기 때문에 클라이언트 변조를 통한 부정투표 결과는 성립할 수 없다. 이를 통해 REV 방식의 단점이었던 클라이언트 취약점이 해결된다.

그러나 비트코인에서 제안한 블록체인 시스템은 오직 금융 목적으로 이용되는 것을 전제로 하는 태생의 특성상 미리 지정되어 있는 표준 거래 스크립트 형식 외의 스크립트의 실행을 보장할 수 없으며 저장공간의 제약 등의 한계가 존재한다.

때문에 이더리움(Ethereum) 과 같은 기존 블록체인의 한계를 극복하기 위한 새로운 블록체인 프레임워크를 제안하는 움직임이 있어왔으며 대기업의 경우 IBM 의 Hyperledger, 삼성과 IBM 이 공동으로 제안한 IoT(Internet Of Things) 환경에 최적화된 블록체인 프레임워크 ADEPT 등의 다양한 시도가 이루어지고 있다 [4, 5].

2.3 스마트 계약(Smart Contract)

Vitalik Buterin 이 제안하고 Gavin Wood 가 구체화시킨 새로운 개념의 블록체인 플랫폼, 이더리움(Ethereum)에서는 기존 블록체인의 단순히 검증과 연산에 그쳤던 거래 스크립트를 상태와 함수를 정의하고 상태 변이와 데이터 저장이 가능한 튜링 완전한 (Turing Complete) 프로그래밍 언어로 기능하도록 재설계하였다 [4, 5].

이렇게 블록체인 상에 등록되고 실행되는 컴파일된 코드 조각을 더 이상 거래 스크립트가 아닌 스마트 계약(Smart Contract)이라고 한다. 스마트 계약이란 블록체인상의 모든 노드에서 일련의 규약에 따라 작성, 업로드되고 실행되는 코드를 지칭한다. 비트코인 거래 스크립트 또한 작은 의미의 스마트 계약이라고 할 수 있다.

$$APPLY(S, TX) \rightarrow S' \quad (1)$$

스마트 계약을 지원하는 플랫폼의 거래는 계약을 거치는 서명된 데이터 패키지로써 표현되며 계약은 거래를 입력으로 받는 수식(1)과 같은 형태의 상태 변이 함수(State Transition Function)로써 표현 할 수 있다. 이때 S 는 이더리움 계정(Account)들의 상태 집합으로 표현된다.

계정은 사용자에 의해 소유되는 사용자 계정(EOA, Externally Owned Account)과 계약 계정(Contract Account) 두가지가 존재한다. 계약 계정은 사용자 계정과는 달리 스마트 계약 코드를 가지고 있고 스마트 계약 코드는 스스로 함수를 실행할 수 없으며 다른 계약 계정이나 사용자 계정에서 호출되어 실행된다.

계약 코드는 외부 계정을 통해 2 가지의 방법으로 실행될 수 있다. 한가지는 거래를 통한 실행(Message Transaction)이다. 거래의 목적지로 계약코드를 포함하는 계정을 할당하고 Data Payload 에 실행 함수와 매개변수를 실어 거래 메시지를 전파하면 이더리움의 각 노드가 거래를 검증하는 과정에서 해당 계약 코드의 함수가 실행되고 상태 변이가 일어난다. 다른 하나는 호출(Call)이다. 이는 상태변이를 일으키지 않는 함수를 호출 하는 행위로 각 노드가 거래의 발생 없이 함수를 단독적으로 실행하는 방식이다. 이는 블록체인 상에 거래를 전파하지 않으며 이에 따라 상태변이도 일어나지 않는다.

3. 스마트 계약 기반 탈 중앙 전자 투표

본 장에서는 스마트 계약에 기반한 탈 중앙 전자투표 시스템의 개요와 동작 시나리오를 설명한다.

3.1 투표 시스템 개요

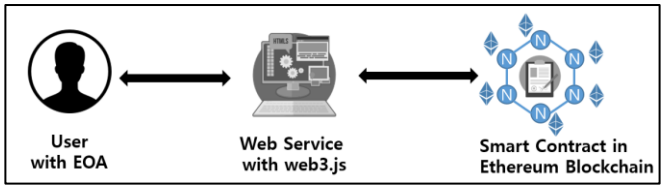


Figure 1. 탈 중앙 시스템 개요

본 시스템은 Figure 1 과 같이 이더리움 프레임워크에 올라가는 SCV(Smart Contract for Vote)와 이를 연결하고 사용자와의 상호작용에 필요한 웹 어플리케이션으로 이루어진 탈 중앙화된 어플리케이션(Decentralized Application, DApp) 이다. SCV 는 다음의 기능을 하는 함수들로 이루어져 있다.

- RSA 비대칭 키 쌍 발급(get_rsa_key_pair)
- 암호화(encrypt)
- 투표 생성(create_vote)
- 투표지 발급(get_ballot_id)
- 투표권 행사(vote)

- 개표(count)
- 개표 결과 조회(get_result)

투표 참여자는 선거관리위원회(이하 선관위), 유권자, 후보자 세가지 역할군으로 나누어지며 각자의 이더리움 사용자 계정을 가지고 있다고 가정하고 선관위는 유권자와 후보자의 이더리움 사용자 계정 리스트를 보유하고 있다고 가정한다.

3.2 동작 시나리오

3.2.1 투표 생성

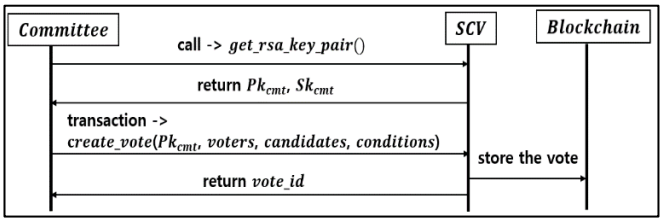


Figure 2. 투표 생성 Sequence

선관위(Committee)는 자신이 보유하고있는 사용자 계정에서 SCV 의 RSA 비대칭 키 쌍 발급 함수(get_rsa_key_pair)를 호출하여 공개키(Pk_{cmt})와 비밀키(Sk_{cmt})를 발급받는다. 호출은 2.3 장에서 기술한 바와 같이 이더리움 블록체인의 상태 변이를 일으키지않고 자신의 이더리움 노드에서만 함수를 실행한다. 키를 발급받는 행위가 이더리움 블록체인 네트워크상에 기록된다면 후술할 투표지를 발급하는 과정에 있어서 투표자의 신원이 노출될 가능성이 높으므로 앞으로의 키발급은 모두 호출로 이루어짐을 명시한다.

선관위는 투표 생성 함수(create_vote)를 호출하는 거래를 생성하며 자신의 공개키(Pk_{cmt}), 유권자들의 계정 주소들($voters$), 후보자의 계정 주소들(candidates), 투표 시작시간, 종료시간 등의 제약조건(conditions)를 매개변수로 입력한다. Pk_{cmt}는 유권자들의 투표내역 암호화에 사용된다. 투표가 생성되면 해당 투표의 ID($vote\_id$)가 발급된다. 이때 유권자, 후보자의 사용자 계정들은 선관위에서 공인한 인증절차에 따라 1인 1계정으로 인증 되어있다고 가정한다.

3.2.2 투표지 발급

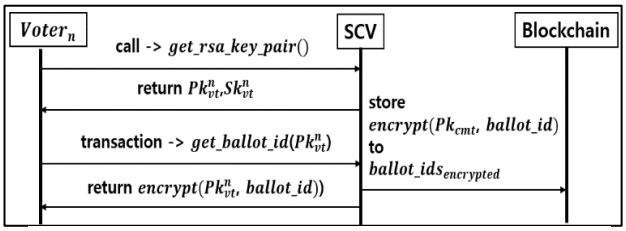


Figure 3. 투표지 발급 Sequence

설정된 투표 시작시간 이후에 유권자($Voter_n$)는 자신의 사용자 계정으로 get_rsa_key_pair 함수를 호출해 키 쌍(Pk^n_{vt},Sk^n_{vt})을 발급받는다.

유권자는 Pk^n_{vt}를 매개변수로 하여 투표지 발급 함수(get_ballot_id)를 호출하는 거래를 생성한다. 해당 거래가 검증되면 투표 계약의 저장소 내에는 선관위의 공개키로 암호화된 무작위의 투표용지 ID($ballot\_id$)가 저장된다. $ballot\_id$ 값은 RFC4122 의 UUID(Universal Unique Identifier)[11] 형태를 취한다. 한번 $ballot_id$를 발급받은 유권자는 다시 발급받지못하도록 기록한다.

3.2.3 투표 시행

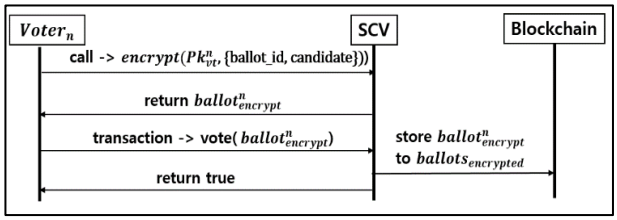


Figure 4. 투표 시행 Sequence

유권자는 자신의 $ballot_id$와 자신이 선택한 후보자의 사용자 계정 주소(candidate)를 기입한 투표용지를 Pk_{cmt} 로 암호화한다. 암호화 한 값($ballot^n_{encrypt}$)을 매개변수로 입력하고 투표 시행 함수(vote)를 호출한다. 해당 함수는 입력 받은 값을 검증하지 않고 바로 암호화된 투표 내역($ballots_{encrypted}$)에 저장한다. 투표지 발급과 마찬가지로 한번 투표한 유권자 계정은 다시 투표할 수 없도록 기록한다.

3.2.4 개표 시행

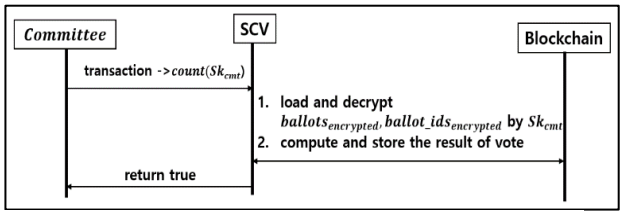


Figure 5. 개표 Sequence

선관위는 투표 종료 시간 이전에 개표를 시행할 수 없다. 투표 종료 시간 이후에는 Sk_{cmt}를 매개변수로 개표 함수(count)를 호출하는 거래를 생성한다. count는 $ballots_{encrypted}$와 $ballot_ids_{encrypted}$를 Sk_{cmt}로 복호화 하여 투표지의 $ballot\_id$ 와 등록된 $ballot_id$를 비교해 투표용지를 유효화 혹은 무효화 한다. 이후 유효한 개표 결과를 계산한 뒤 해당 투표의 저장소 상에 기록한다.

3.2.5 개표결과 조회

모든 유권자는 개표가 종료된 뒤 개표결과 조회 함수(get_result)를 호출함으로써 개표 결과를 조회할 수 있다.

4. 관련 연구

전자 투표 시스템의 오래된 기존 연구들은 서버-클라이언트 방식의 시스템 구조에 서버 보안 및 클라이언트 보안 체계를 강화하는 방식으로 진행되어 왔다[1,2]. 최근 소수의 논문에서는 블록 체인을 사용한 투표시스템이 제안되었다. 그러나 해당 연구들은 명확한 한계점을 지니고 있다. Kibin Lee가 제안한 블록체인을 사용한 투표시스템 [6]의 경우 총 투표내역중 유효한 투표내역을 걸러내기 위해서는 신뢰할 수 있는 제 3의 서비스를 두어야 한다는 한계가 존재한다. 유헌우가 제안한 블록체인에 기초한 투표 시스템[9]은 투표 거래 내역 추적을 통해 어떤 유권자가 어떤 후보에게 투표하였는지 거래 내역 추적을 통해 알 수 있다는 점에 있어서 비밀성을 보장할 수 없다는 한계를 가지고 있다.

본 연구에서 제안하는 투표 시스템은 기존 연구와 같이 블록체인을 활용하여 투명성을 보장한다는 점은 같으나 신뢰할 수 있는 제 3의 서비스가 필요 없을 뿐만 아니라 이더리움의 스마트 계약과 RSA 비대칭 키 암호화를 이용하여 투표 내역을 암호화하는 비밀 투표 시스템 디자인을 제안하여 기존 연구의 한계를 극복하였다.

5. 결론 및 향후 연구

본 논문에서는 전자투표 수용의 신뢰성을 보장하고 기술적인 안정성을 담보할 수 있는 원격 인터넷 투표를 위한 스마트 계약 기반의 탈 중앙화된 전자 투표 시스템을 제안하였다.

블록체인 기술을 적용한 전자 투표 시스템은 기존의 전자투표 시스템의 장점과 종이 투표 시스템의 장점을 모두 포함하고 있으며, 자유롭고 자발적인 노드의 참여를 통한 투표 내역 검증을 수행하므로 개표의 신뢰성을 높일 수 있다. 동시에 투표 내역을 RSA 비대칭 키로 암호화하여 선관위와 투표자 본인만 열람할 수 있도록 설계해 익명성과 투명성을 모두 만족하였다.

그러나 투표 내역을 암호화 혹은 복호화하는 기능을 수행하는 데 있어 유권자 혹은 선관위가 부담해야하는 실행 비용이 매우 커질 것으로 예상된다는 한계점이 존재한다. 해당 내용은 현재 이더리움 커뮤니티 내에서도 논의가 되고 있는 주제이며 가까운 시일 내에 이더리움 자체적으로 관련 OPCODE를 지원할 것으로 보인다[10].

이와는 별개로 우리는 특정 요구사항을 충족하는 스마트 계약을 구현할 때의 정적, 동적 비용을 최적화하는 개발 방법론에 대한 연구를 수행할 예정이다.

6. Acknowledgement

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(2012M3C4A7033345).
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [2016(R2215-16-1007), 빅데이터 분석을 이용한 패션 시장 흐름 분석 및 소비자 의견 추출 기법 연구]

참고문헌

[1] Nikos Chondros, Bingsheng Zhang, “D-DEMOS: A distributed, end-to-end verifiable, internet voting system, arXiv, 2015
[2] Chris Culnane et al, “vVote: A Verifiable Voting System”, ACM TISSEC, 2015
[3] Andreas M. Antonopoulos, “Mastering Bitcoin: Unlocking Digital Crypto-Currencies, O'Reilly Media, 2014
[4] Vitalik Buterin, “The Ethereum White Paper”, https://github.com/ethereum/wiki/wiki/White-Paper
[5] Gavin Wood, “Ethereum: A Secure Decentralised Generalised Transaction”, https://github.com/ethereum/ yellowpaper
[6] Kibin Lee, Joshua I.James, Tekachew Gobena Ejeta, Hyoung Joong Kim, “Electronic Voting Service Using Blockchain”, ADFSL, 2016
[7] 박해영, “전자 투표를 통한 국민 주권 실현 방안 연구”, 창원대학교, 2007
[8] 조희정, “미국의 전자투표와 기술 수용 정치: 브라질, 에스토니아와 비교를 중심으로”, 서강대학교, 2007
[9] 유헌우, “블록체인 방식의 전자투표 시스템 구현 및 성능 개선 방안 연구”, 아주대학교 학위논문, 2016
[10] Vitalik Buterin, “Privacy on the Blockchain”, https://blog.ethereum.org/2016 /01/15/privacy-on-the-blockchain/, 2016
[11] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, 2005

모바일 디바이스 배터리 소모 분석 기법 비교

송지영 ○ 조치우 지은경 배두환
KAIST
{jysong, cwcho, ekjee, bae}@se.kaist.ac.kr

Comparison on mobile device battery consumption analysis techniques

Jiyoung Song○ Chiwoo Cho Eunkyoung Jee Doo-Hwan Bae
KAIST

요 약

제한된 자원인 모바일 디바이스 배터리의 소모는 회로 설계자들이 회로를 분석 및 평가할 때 중요한 척도가 된다. 이에 연구자들은 모바일 디바이스 배터리 소모 분석을 위해 여러 배터리 소모 모델 생성 연구를 수행하였다. 서로 다른 특징을 갖는 배터리 소모 모델 생성 기법들 간의 비교 분석을 수행한 연구는 많이 이루어지지 않았으며, 모델의 검증 및 테스팅 목적으로의 비교를 다루지 않았다. 본 연구에서는 모바일 디바이스 회로 설계자들에게 도움을 주기 위하여 기존 기법들에 대하여 종합적인 비교를 수행하고, 향후 모바일 디바이스 배터리 소모 분석 연구의 방향성을 제시한다.

1. 서 론

모바일 디바이스 사용의 증가와 모바일 디바이스를 구성하는 하드웨어 컴포넌트들의 복잡도가 높아짐에 따라 제한된 자원인 배터리를 효율적으로 사용하는 것이 중요해지고 있다. 회로 설계자들이 모바일 디바이스의 배터리를 효율적으로 사용하는 회로를 설계하기 위해서 배터리 소모를 정확하게 측정할 수 있어야 한다. 이에 연구자들은 모바일 디바이스 기종별 배터리 소모 모델 생성에 대한 연구를 진행해왔다[1-6].

모바일 디바이스 회로 설계자들의 주어진 상황에 따라 적절한 기법 선택을 돕거나 기존 기법들의 단점을 보완하는 기법을 제안하기 위해서는 기존 배터리 소모 분석 기법들 간의 비교 분석 연구가 필요하다. 본 논문에서는 기존의 배터리 소모 모델을 생성하는 기법들 간의 비교 분석을 수행하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 배터리 소모 모델에 대한 배경지식에 대하여 서술한다. 3장에서는 선정된 논문들을 비교 분석하며, 향후 모바일 디바이스 배터리 소모 분석 연구의 방향성을 제시한다. 마지막으로 4장에서 결론 및 향후 연구를 기술한다.

2. 배경 지식

선형회귀기법이란[7] 통계적 예측 기법 중 하나로, 회귀 분석에서 독립 변수에 따라 종속 변수의 값이 일정한 패턴으로 변해 갈 때, 변수간의 관계를 나타내는 회귀선이 직선에 가깝게 나타나는 것을 선형 회귀 분석이라 정의한다. 하드웨어 컴포넌트들은 시간에 따라 소비 전력이 일정하게 증가하므로 많은 논문에서 선형회귀분석기법을 모델링하는데 사용하였다.

유한 상태 기계 기법은[8] 유한한 상태를 가지는 시스템을 모델링한 기법으로, 유한 상태 기계는 하나의 현재 상태를 가지며 입력에 따라 상태 전이가 발생한다. 모바일 디바이스 컴포넌트들은 Active, Idle, Sleep, Doze 와 같은 상태를 가지며, 모바일 디바이스에 주어지는 입력이나 환경에 따른 상태 전이가 유한 상태 기계로 모델링 되어왔다.

* 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 **(No. R0126-16-1101, (SW 스타랩) 모델 기반의 초대형 복잡 시브템 분석 및 검증 SW 개발)**

3. 모바일 디바이스 배터리 소모 분석 기법 평가

3.1 모바일 디바이스 배터리 소모 분석 기법 및 선정

모바일 디바이스 배터리 소모 분석 기법을 비교하기 위해서 논문 조사를 수행하였다. 논문 선정 기준은 1) 최근 5년 이내에 한국정보과학회에서 작성한 소프트웨어 분야 우수 학술대회[9]에서 발표된 논문 중 2) 모바일 디바이스 배터리 소모 분석에 관련되어야 한다는 것이다. 관련 유무는 논문[10]에서 서술한 배터리 소모 분석 용어들의 조합: mobile device (power / battery) (measurement / model / analysis / meter / profile)이 포함되었는지로 판단하였다. 위 키워드를 포함하는 논문을 우선 선정 후, 초록을 읽고 모델을 사용하지 않거나 배터리 소모 예측 관련 논문을 조사 대상에서 제외하였다.

3.2 모바일 디바이스 배터리 소모 분석 기법 비교 분석

최종적으로 선정된 6개의 논문은 정원우 외(2012)가 CODES + ISSS에서 발표한 DevScope[1], 윤찬민 외(2012)가 USENIX ATC에서 발표한 AppScope[2], Fengyuan Xu 외(2013)가 NSDI에서 발표한 V-edge[3], Abhinav Pathak 외(2012)가 EuroSys에서 발표한 Eprof[4], Abhijeet Banerjee(2014)가 FSE에서 발표한 논문[5], 김기태 외(2014)가 DATE에서 발표한 FEPMA[6]이다.

DevScope [1]. 런타임 시에 배터리 소모 모델을 생성하는 DevScope는 기존의 오프라인 상태에서의 모바일 디바이스 배터리 소모 모델링 기법과 비교하여 배터리 소모 모델을 동적으로 생성할 수 있다는 장점을 가지고 있다. DevScope는

내장된 배터리 모니터링 유닛을 사용하여 배터리 소모 모델을 생성하며, 정확한 배터리 소모 모델 생성을 위해 컴포넌트 별 배터리 소모 모델을 생성하고 전체 배터리 소모를 분석하였다.

AppScope[2]. AppScope는 DevScope의 단점을 보완하기 위해 확장한 기법이다. DevScope는 하드웨어와 시스템 소프트웨어에 대한 정보만 접근이 가능했던 한계 때문에 어플리케이션이 소모하는 배터리 모델의 정확도가 낮은 단점이 있었다. AppScope는 어플리케이션의 배터리 소모를 정확하게 측정하기위해 어플리케이션의 시스템 호출과 binder IPC 데이터 정보를 이용하여 동적으로 모델을 생성한다.

V-edge[3]. 배터리를 측정할 수 있는 방법으로 전류 측정 센서를 이용하는 방법과 전압 측정 센서를 이용하는 방법이 있다. 그러나 대부분의 모바일 디바이스는 전류 측정 센서가 없어 전류 측정 센서를 이용하는 기법들은 다수의 모바일 디바이스에 적용될 수 없다. V-edge는 이러한 단점을 극복하기 위해 전압이 동적으로 변하는 순간(voltage-edge)을 사용하여 배터리 소모 모델을 동적으로 생성하였다.

Eprof[4]. Pathak 외가 제안한 Eprof는 안드로이드와 Windows기반의 오프라인 상태에서 배터리 소모 모델을 생성하는 프레임워크이다. Eprof는 NDK를 사용하여 시스템 호출이 발생한 시간, 파라미터, 콜 스택 등을 기록한다. 시스템 호출은 다른 하드웨어 컴포넌트를 사용할 때 발생하므로 컴포넌트별 모델을 생성하기에 용이하다. 따라서 시스템 호출을 추적한 정보를 이용하여 배터리 소모 모델을 생성한다.

Banerjee 외[5]. Banerjee 외(2014)가 제안한 기법은 다른 기법들과 다르게 배터리 소모 모델을 테스트 생성을 목적으로 사용한다. 저자들은 에너지 핫스팟과 에너지 버그를 찾아서 오프라인에서 자동으로 테스트를 생성해주는 기법을 제시하였다. 제안된 기법은 이벤트 트레이스를 생성하면서 컴포넌트의 사용 확인을 위해 시스템 호출을 기록하며 모델을 분석한다.

FEPMA[6]. FEPMA는 센서 없이 런타임 시에 배터리 소모를 분석하는 프레임워크이다. FEPMA는 센서를 이용하지 않는 간접적 배터리 측정 방법인 상태 기반 배터리 측정을 사용하며, 안드로이드 운영체제를 수정함으로써 커널의 최하위 레벨에서 전원 정보를 제공받아 시간 지연 문제와 정확한 하드웨어 컴포넌트들의 상태 식별 문제를 해결하였다.

표 1. 배터리 소모 모델 분석 기법 표

기법 이름 / 저자 이름	모델링 레벨	모델 타입	런타임 모델 생성 (온라인 / 오프라인)	모델 생성 목적
DevScope	시스템, 컴포넌트	선형회귀모델, FSM	온라인	배터리 소모 추정
AppScope	어플리케이션	DevScope모델	온라인	배터리 소모 추정
V-edge	시스템, 컴포넌트	선형회귀모델	온라인	배터리 소모 추정
Eprof	시스템, 컴포넌트, 어플리케이션	시스템 호출 트레이스, FSM	오프라인	배터리 소모 추정
Banerjee 외.	시스템, 컴포넌트, 어플리케이션	시스템 호출 트레이스, 선형회귀모델	오프라인	테스트 시나리오 생성
FEPMA	시스템, 컴포넌트, 어플리케이션	시스템 호출 트레이스, 선형회귀모델	온라인	배터리 소모 추정

표 1에서는 6개 논문에 대해서 모델링 레벨, 모델의 타입, 런타임 모델 생성 여부, 모델 생성 목적에 따라 비교한 것을 나타내었다. 위 기법들은 시스템 커널에서 어플리케이션 레벨에 이르기까지 다양한 레벨로 모델을 생성하였고, 배터리 소모 모델은 선형 회귀, 유한 상태 기계, 시스템 호출 트레이스 타입으로 나뉜다. 런타임 시에 모델이 생성되는지 여부에 따라 모델의 동적 생성 여부를 판단하였다. 대부분의 배터리 소모 모델 생성의 목적은 배터리 소모 추정이었고, Banerjee 외의 논문은 모델을 테스트 생성 목적으로 활용하였다.

3.3 모바일 디바이스 배터리 소모 분석 연구 향후 과제

모바일 디바이스 회로 설계자들에게 정확한 배터리 소모 모델을 제공하는 것도 도움이 될 수 있으나, 하드웨어의 오류를 잡거나 최적화 해야하는 부분을 찾기 위해서 모델을 테스팅 및 검증 목적으로 활용하는 연구가 진행되어야 할 것이다. 또한, 모바일 디바이스 사용 대상자의 배터리 소비패턴을 모델에 반영하는 연구가 진행되어야 한다. 통계적 검증을 사용하여 대상 모바일 디바이스 하드웨어 모델에 실제 사용자들의 다양한 사용 패턴 샘플들을 분석할 수 있다면, 모바일 디바이스 회로 설계자들이 대상 연령층이나 사용자 특성에 맞는 설계를 수행할 수 있을 것이다.

4. 결론 및 향후 계획

본 논문에서는 모바일 디바이스 배터리 소모 분석 기법들을 비교 및 분석하였다. 분석 결과 모바일 디바이스 배터리 소모 모델에 대한 검증 연구가 미성숙하다는 것과 실제 사용자들의 배터리 소모 데이터 분석 결과가 모델에 반영돼야 함을 확인할 수 있었다. 평가 결과를 바탕으로 실제 사용자 배터리 소모 데이터를 활용한 배터리 검증 연구를 진행할 예정이다.

참고 문헌

[1] Jung, Wonwoo, et al. "DevScope: a nonintrusive and online power analysis tool for smartphone hardware components." Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. ACM, 2012.

[2] Yoon, Chanmin, et al. "AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring." USENIX Annual Technical Conference. Vol. 12. 2012.

[3] Xu, Fengyuan, et al. "V-edge: Fast Self-constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics." NSDI. Vol. 13. 2013.

[4] Pathak, Abhinav, Y. Charlie Hu, and Ming Zhang. "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof." Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012.

[5] Banerjee, Abhijeet, et al. "Detecting energy bugs and hotspots in mobile apps." Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014.

[6] Kim, Kitae, et al. "FEPMA: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring." Proceedings of the conference on Design, Automation & Test in Europe. European Design and Automation Association, 2014.

[7] Montgomery, Douglas C., Elizabeth A. Peck, and G. Geoffrey Vining. Introduction to linear regression analysis. John Wiley & Sons, 2015.

[8] https://en.wikipedia.org/wiki/Finite-state_machine

[9] https://an.kaist.ac.kr/~sbmoon/misc/SWconf.pdf

[10] Hoque, Mohammad Ashraful, et al. "Modeling, profiling, and debugging the energy consumption of mobile devic-es." ACM Computing Surveys (CSUR) 48.3 (2016): 39..

소프트웨어 정의 네트워크 환경에서 애플리케이션 QoS 보장을 위한 모니터링 아키텍처

황제승 O*, 김웅수*, 박준석**, 염근혁 ∮ 교신저자
*부산대학교 전기전자컴퓨터공학과, **부산대학교 물류혁신네트워킹연구소,
∮부산대학교 전기컴퓨터공학부
{hjs0790, kus9010, pjs50, yeom}@pusan.ac.kr

Monitoring Architecture for Ensuring application QoS in Software Defined Networking Environment

Jeseung Hwang O*, Ungsoo Kim*, Joonseok Park**, Keunhyuk Yeom ∮ corresponding author
Department of Electrical and Computer Engineering, Pusan National University*∮
Research Institute of Logistics Innovation and Networking, Pusan National University**

요 약

최근 고품질의 IT 서비스를 제공하게 됨으로써 대규모 트래픽이 발생함에 따라 네트워크 분야에서는 기존의 네트워크 체계를 변경하기 위해 나온 개념이 소프트웨어 정의 네트워크(Software Defined Networking, SDN)이며, 이는 소프트웨어로 네트워크를 제어할 수 있는 환경을 지칭한다. SDN이 등장함에 따라 네트워크를 제어하거나 모니터링하는 관련 연구가 활발하게 진행되고 있다. 그러나 기존 연구들은 네트워크에 연결되어 있는 Switch, Host의 장비들 즉, 자원들을 효율적으로 제어 하는 것에 초점을 맞추어 진행되어 왔다. 본 논문에서는 네트워크에 운영중인 애플리케이션 관점에서 애플리케이션이 원활한 서비스 제공을 위해 필요로 하는 요구사항을 기반으로 모니터링을 하고 애플리케이션의 안전성을 보장하여 지속적인 서비스를 제공하기 위한 모니터링 아키텍처를 제안한다.

1. 서 론

최근 IT 기술의 급격한 발전으로 인하여 다양한 기능의 고품질 서비스 제공이 가능해졌다. 고품질의 서비스를 제공하면서 대규모의 트래픽을 실시간으로 관리하는 방법과 새로운 기능에 대한 지속적인 업데이트가 요구되었다. 이러한 문제는 기존 네트워크 구조의 변화를 요구하였고, 이를 지원하기 위한 방법으로써 소프트웨어 정의 네트워크(SDN)가 대두되었다. SDN은 네트워크의 데이터 평면(data plane)과 제어 평면(control plane)을 분리하는 것으로, 네트워크 장비들을 소프트웨어로 제어하는 개념이다. SDN의 핵심 특징은 프로그래밍이 가능(program-ability)하여 동적으로 네트워크를 구성하거나 관리할 수 있다[1]. 또한, 데이터 평면(data plane)과 제어 평면(control plane)을 분리하여 중앙 집중화된 컨트롤러로 구성하여 네트워크 트래픽을 제어한다. 네트워크 애플리케이션을 통하여 SDN 컨트롤러를 제어하기 위하여 애플리케이션 평면 (application plane)과 제어 평면(forwarding plane) 사이에는 Northbound API가 제공된다. 해당 API를 이용하면 실제로 물리적인 네트워크 인프라 제어가 가능하므로 동적으로 변화하는 네트워크 환경을 실시간 모니터링 하여 감지하고 원하는 경로에 네트워크 트래픽을 전달하도록 구현할 수 있다. 미래의 네트워크에는 SDN 기반의 네트워크가 형성될 것이라는 전망과 함께 SDN의 필요성이 강조되고 있다. 하지만 아직 시장은 초기 단계이며 관련된 기술 개발 및 연구가 한창 진행 중이다. 그 중에서도 SDN의 데이터 평면(data plane)을 모니터링을 하는 방법들에 대해 많은 연구가 진행되었다[2]. 특히 FlowCover, FlowSense 및 PayLess는 데이터 평면 모니터링 연구들로, 데이터 평면의 사용량(Utilization)을 정확하고 낮은 비용으로 모니터링을 하는 것을 목적으로 한다[3]. 기존 연구는 네트워크를 중심으로 트래픽을 제어 및 관리하며 모니터링하는 방법을 위주로 진행되었다. 또한 애플리케이션 관점의 모니터링 관련 연구는 부족한 상태이다.

따라서 본 논문에서는 SDN 환경에서 애플리케이션별 요구하는 QoS를 보장하기 위해 네트워크 경로를 관리하며 모니터링을 지원하는 아키텍처를 제시한다.

"이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2014R1A1A2007061)"

2. 관련 연구

2.1 OpenFlow 기반의 모니터링

OpenFlow는 SDN을 실현하기에 가장 적합한 기술로 평가되고 있으며 SDN 구조의 제어 평면(control plane)과 데이터 평면(data plane) 사이에 인터페이스 규격으로 사용되고 있는 기술이다. 기존 SDN 모니터링 연구들은 OpenFlow를 이용하여 SDN 환경을 모니터링을 하는데 집중되어 있다. OpenFlow를 이용하면 Switch, Host와 같은 장비들의 트래픽 정보를 획득하거나 제어가 가능하다. 하지만 실시간 네트워크 환경에서는 대규모의 트래픽이 송수신되므로 중앙집중식 구조로 되어 있는 SDN 에서는 많은 트래픽 오버헤드를 발생시키게 된다. 따라서 트래픽 오버헤드를 줄이기 위하여 낮은 비용으로 모니터링을 하는 연구가 주를 이루었다[4]. PayLess[5], SUMA[6] 연구에서는 제어 평면(control plane)에 추가적인 Middlebox 형태의 구조를 제시하였다. Middlebox를 추가하게 됨으로써 모니터링을 하는 컨트롤러의 부하를 줄이고 독립적으로 관리하게 됨으로써 보안성, 확장 가능성을 갖춘 SDN 아키텍처를 제시하였다.

2.2 Flow 샘플링 기반의 모니터링

Flow는 OpenFlow에서 네트워크 트래픽을 제어하기 위한 방법이다. 패킷이 들어오면 Flow Table을 확인하여 정해진 규칙대로 패킷을 전송하는 방법이다. MopSamp[7] 연구는 SDN에서의 트래픽 모니터링에 적합한 플로우 샘플링 방법을 제시하였다. 확장할 수있고 효율적인 모니터링 아키텍처를 제시하였으며 트래픽을 샘플링하기 위한 SDN 애플리케이션을 설계하였다.

3. 애플리케이션 QoS 보장을 위한 모니터링 아키텍처

애플리케이션 QoS 보장을 위한 모니터링의 목표는 SDN 환경에서 애플리케이션의 요구사항을 기반으로 이를 보장하기 위해 네트워크 트래픽을 모니터링하고 컨트롤러를 제어하여 원활한 네트워크 환경을 유지하는 것이다.

본 장에서는 SDN 환경에서 애플리케이션 QoS 에 대한 요구사항을 분석하고 애플리케이션 유형별로 SDN 환경을 모니터링 하기 위한 아키텍처를 제시한다.

3.1 아키텍처 요구사항

SDN 환경에서 애플리케이션에 대해 모니터링을 하기 위해 아래의 [표 1]과 같이 총 6가지의 모니터링 요구사항을 도출하였다. [표 1]의 6가지 요구사항은 SDN 환경에서 애플리케이션이 원활한 운영을 위해 자원들을 파악하고 네트워크를 제어하기 위한 모니터링 요구사항으로써 정리하였다.

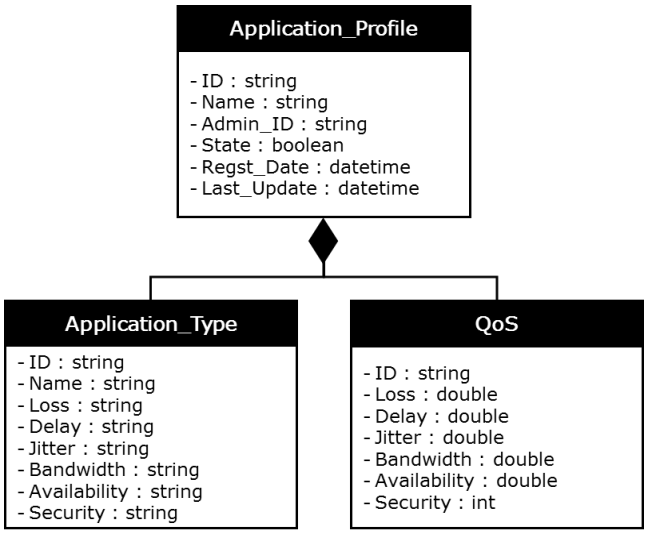
[표 1] 모니터링 아키텍처 요구사항

No	Requirements	Description
Req1	애플리케이션 프로파일 관리	애플리케이션에 대한 프로파일 정보를 수집 하고 관리한다.
Req2	Topology 모니터링	네트워크를 구성하는 Topology에 대한 정보를 보여주며 연결 구조를 트리 형태로 가시화한다
Req3	Switch, Host 모니터링	네트워크상에 연결되어 있는 Switch, Host에 대한 상세 정보(MAC Address, IP, Port 등)를 볼 수 있다.
Req4	Routing 관리	네트워크에 구성된 모든 Routing 경로를 관리 하며, 애플리케이션이 사 용 하고 있는 Routing 경로에 대해 정보를 볼 수 있다.
Req5	Flow Table 관리	네트워크 트래픽 제어를 위해 Flow Table 전체 현황 및 Flow Rule의 추가, 수정, 삭제를 할 수 있다.
Req6	이벤트 로깅	모니터링을 통해 모든 트래픽의 현황 및 송수신, 네트워크 상태 변경에 대한 모든 이벤트를 기록하고 확인할 수 있다.

애플리케이션의 요구사항을 파악하기 위하여 프로파일 정보를 수집한다. 프로파일 정보에는 애플리케이션의 이름, 애플리케이션 유형, 관리자, 애플리케이션이 요구하는 QoS 등이 있다. 네트워크 모니터링을 위하여 현재 네트워크를 구성하고 있는 Topology에 대한 정보와 연결된 Switch, Host에 대해 IP, Port, MAC Address, QoS 등의 정보를 수집한다. 또한 네트워크의 모든 Routing 정보를 수집하여 실제로 어떠한 경로로 트래픽이 전송되고 있는지 확인하며 Flow Table을 이용하여 트래픽에 대해 경로를 Flow Rule을 통해 제어한다.

3.2 애플리케이션 프로파일 명세

애플리케이션의 프로파일 명세 모델에는 애플리케이션의 ID, 애플리케이션의 이름, 관리자 ID, 운용중인 상태, 등록 날짜, 마지막 수정 날짜, 애플리케이션의 유형(Type)별 ID와 6가지의 QoS 요소의 기준값, 애플리케이션이 요구사항의 ID와 그에 해당하는 6가지의 QoS 요소를 정의하였다. 아래 [그림 1]은 애플리케이션의 프로파일 명세 모델이다.



[그림 1] 애플리케이션 프로파일 명세 모델

명세 모델 중에 애플리케이션 유형을 구분하기 위해 CISCO에서 정의한 것을 참고하였고 Network Control, Telephony 등 총 12가지 애플리케이션 유형으로 구분하였다. *RFC 5127*에서는 애플리케이션 QoS를 Loss, Delay, Jitter 3가지 요소를 기반으로 각 애플리케이션 유형별 기준값을 정의하였다. 본 논문에서는 추가로 Bandwidth, Availability, Security 3가지의 애플리케이션 QoS 요소를 추가하여 [표 2]와 같이 총 6가지의 애플리케이션 QoS를 정리하였다.

[표 2] 애플리케이션 QoS 분류

QoS	Description	Unit
Loss	전송 과정에서 손실되는 패킷량	bytes
Bandwidth	네트워크에서 이용 가능한 최대 전송속도	bps
Delay	패킷 전송에서 걸리는 지연 시간	ms
Jitter	전송 처리 시간의 변동폭	ms
Availability	전송 처리의 가용성 (안정적인 전송 처리 수행 시간 비율)	percent
Security	보안 지원 수준	level

6가지 QoS요소를 기반으로 *RFC 5127*에서 정의한 12가지의 애플리케이션 유형별 기준값을 정의하였다. [표 3]은 *RFC 5127* 에서 정의한 12가지 애플리케이션 유형중 하나인 Network Control에 대한 QoS 기준값을 예시로 나타낸 표이다.

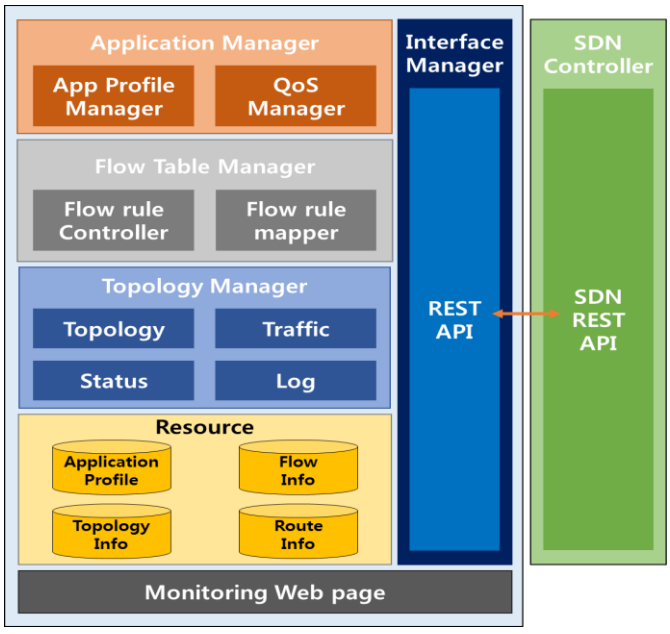
해당 애플리케이션 유형별로 각각의 애플리케이션 QoS 요소에 대한 기준값이 다르다. 예를 들어 Network Control인 경우 실시간으로 네트워크를 제어해야 하므로 Loss, Delay는 Low 해야 하며 Availability는 High, Security는 Very High 해야 한다. 이처럼 12가지 애플리케이션 유형별 QoS 기준값을 정의하였다.

[표 3] 애플리케이션 유형에 따른 예시

Feature	Value
Type	Network Control
Loss	Low
Delay	Low
Jitter	Yes
Bandwidth	Low-Medium
Availability	High
Security	Very High

3.3 모니터링 아키텍처

모니터링 요구사항을 먼저 도출하였고 요구사항을 기반으로 애플리케이션의 프로파일 명세 모델을 정의하였다. 요구사항과 애플리케이션 프로파일 명세 모델을 기반으로 SDN 환경에서 애플리케이션 모니터링을 위한 아키텍처를 제시한다. 모니터링 아키텍처는 [그림 2]와 같다.



[그림 2] 모니터링 아키텍처

모니터링 아키텍처는 모니터링 요구사항을 바탕으로 *Application Manager*, *Topology Manager*, *Flow Table Manager*, *Interface Manager* 총 4개의 모듈과 4개의 *Resource*로 구성되어 있다. *Application Manager*는

애플리케이션의 프로파일을 수집 및 관리하고 애플리케이션 유형별 QoS 측정 및 비교하는 모듈이다. *Topology Manager*는 현재 네트워크를 구성하는 Topology의 연결 상태, 트래픽 전송 상태, 이벤트 로깅을 하는 모듈이다. *Flow Table Manager*는 애플리케이션이 요구하는 QoS 수준을 보장받지 못할 때 다른 Routing 경로로 변경하기 위한 Flow rule을 구성하여 Flow Table에 추가하는 모듈이다. *Interface Manager*는 네트워크 정보 요청 또는 제어 요청이 발생하면 SDN 컨트롤러에서 제공하는 REST API를 호출하여 해당 기능을 수행하는 역할을 하는 모듈이다. 각각의 SDN 컨트롤러에서 제공하는 API를 기능별로 정리하였고 모니터링에서 해당 요청에 따라 컨트롤러를 제어할 수 있는 구조를 정의하였다.

4. 아키텍처 프로토타입 구축 및 기존 연구와의 차이점

본 장에서는 3장에서 제안한 모니터링 아키텍처와 기존 연구와 비교평가를 제시한다. 기존의 연구들은 데이터 평면(data plane)을 모니터링 하기 위한 연구들이 진행됐다. 하지만 본 논문에서 제시하는 모니터링 아키텍처는 네트워크에 운영되고 있는 애플리케이션들의 요구사항을 기반으로 QoS를 위반하지 않게 네트워크 모니터링하여 경로를 제어할 수 있는 구조를 제안하였다. 기존의 관련 연구의 차이점을 정리하면 아래와 같다.

1. 특정 컨트롤러에 종속적이지 않은 모니터링 아키텍처를 제공 (ONOS, Floodlight 등 다양한 유형의 컨트롤러를 지원하도록 설계함)
2. 기존의 모니터링 연구들이 OpenFlow 기반의 Southbound에 국한된 것을 확장하여 OpenFlow 기반의 모니터링 정보를 포함하고, 애플리케이션의 QoS 관점에서의 모니터링 부분을 고려하였음
3. 애플리케이션이 요구하는 QoS의 충족 여부를 모니터링하여 SDN 애플리케이션이 QoS를 위반하지 않고 최적 경로에서 동작할 수 있도록 모니터링 하는 측면에 초점을 두고 있음

또한, 제시한 모니터링 아키텍처를 바탕으로 실제 프로토타입을 구현하였다.

5. 결론 및 향후 연구

본 논문에서는 애플리케이션의 프로파일을 수집하여 애플리케이션이 요구하는 QoS 보장을 위한 모니터링 아키텍처를 제안하였다. SDN 환경에서 애플리케이션 모니터링 요구사항을 도출하였고 애플리케이션의 요구사항을 수집하기 위해 프로파일의 명세 모델을 정의하였다. 이를 바탕으로 모니터링 아키텍처를 구성하였다. 기존 모니터링 연구들은 데이터 평면에서 데이터를 어떻게 가져오고 부하를 어떻게 줄일 것인지에 대해 연구가 진행되었다. 본 논문에서는 이러한 기존 연구에서 확장하여 애플리케이션의 요구사항을 수집하여 SDN 환경에서도 QoS를 보장하며 원활한 서비스를 제공하기 위한 모니터링 아키텍처를 제시하였다. 이로 인해 애플리케이션 관리자는 자신이 원하는 네트워크 환경에서 지속해서 고품질의 서비스를 제공 할 수 있게 될 것이다.

향후 연구로는 상황 추론 기법을 적용하여 SDN 환경에서 애플리케이션 요구사항 기반의 지능적 최적 경로 추론에 대한 연구를 수행할 예정이다.

6. 참고문헌

[1] 황제승, 윤동규, 박준석, 염근혁, “SDN 환경의 가상 네트워크 제어를 위한 토폴로지 구현 모델링”, 한국통신학회, 제 26 권 제 1 호, pp.96~97, 2016

[2] TTA, www.tta.or.kr/data/reportDown.jsp?news_num=3756,

[3] 정세연, 이도영, 리건, 유재형, 홍원기, “다중 컨트롤러 환경에서의제어 평면 모니터링 및 스위치 배치 방법에 대한 연구”, KNOM Review, 제 18 권 제 1 호, 2015

[4] 우태희, 유한상, 박준석, “SDN 의 네트워트 모니터링 기술 동향에 대한 연구”, 한국통신학회, 제 60 권, pp. 984-985, 2016

[5] Shihabur Rahman Chowdhury, Md. Faizul Bari, Reaz Ahmed, Raouf Boutaba, “Payless: A low cost network monitoring framework for software defined networks”, Network Operations and Management Symposium, pp. 1-9, May, 2014

[6] Taesang Choi, Sejun Song, Hyungbae Park, Sangsik Yoon, Sunhee Yang, “SUMA: software-defined unified monitoring agent for SDN”, Network Operations and Management Symposium, pp. 1-5, May, 2014

[7] Daniel Raumer, Lukas Schwaighofer, Georg Carle “Monsamp: A distributed sdn application for qos monitoring”, In Computer Science and Information Systems, pp. 961-968, September, 2014