소프트웨어 리파지토리 마이닝의 연구 동향 및 분야 소개

경북대학교 ▌이 은 주

1. 서 론

올해로 MSR(Mining Software Repositories) 국제학술대회가 개최된지 10년이 되었다. MSR 학술대회는, 2004년 소프트웨어 공학 분야의 탑 컨퍼런스인 ICSE (International Conference on Software Engineering)에서의 워크숍으로 시작되어 현재 소프트웨어 리파지토리 마이닝 분야의 대표적인 학술대회가 되었다. 소프트웨어 리파지토리 마이닝(Software repository mining: 이후 MSR로 칭함)은 소프트웨어 리파지토리에 저장된 풍부한 데이터들을 분석하여, 소프트웨어 시스템 및 프로젝트에 유용하고 의미있는 정보를 찾아내는 분야로서[1], 소프트웨어 유지보수 활동을 지원하며, 프로세스를 개선하고, 리서치 아이디어나 기법을 검증하며, 결함 등을 예측한다[2].

MSR관련하여 그동안 상당히 많은 연구들이 진행되어 왔다. MSR 국제학술대회 뿐 아니라 ICSE, ASE (Automated Software Engineering), ICSM(International Conference on Software Maintenance), FSE(Foundations of Software Engineering) 컨퍼런스에 발표된 MSR분야 논문들, 그리고 IEEE Trans. of Software Engineering과 같은 저널에 게재된 논문들을 통하여 MSR연구 동향을 살펴보고 당면한 이슈를 이해하는 것이, 소프트웨어 개발자나 연구자들에게 의미가 있을 것이다. 따라서 본 원고에서는 MSR 분야의 연구 동향을 전기한 학회나 저널에 게재된 논문들 중심으로 살펴보고 정리하고자 한다.

그 전에 MSR 분야에서 기 발표된 서베이(survey) 논문 몇 편을 소개한다. 서베이 논문은, 해당 분야의 개념과 전반적인 연구 흐름을 정리해서 보여주므로, 어떤 분야의 연구를 시작할 때 참고로 하면 유용하다. Kagdi의 연구 [3]은 빈번히 인용되는 MSR 분야의 서베이 논문으로 2006년 8월 이전의 연구들을 대상으로 하고 있으며, MSR의 분야들을 상세히 분류, 설명하고 있다. [4]에서는 MSR의 간략한 소개 및 2008년 기준 연구분야와 향후 연구에 대한 향방을 기술하였다. 비교적 최근에 게재된 [2]에서는 MSR 프로세스를 추출(extraction), 처리(processing), 그리고 분석(analysis)로 나누고, 각 단계별로 이슈를 설명하고 있으며, [4]에서 제기한 도전분야와 비교하여 오픈 이슈들을 정리하였다. 그리고 [5, 6]에서는 과거 MSR 학술대회에서 발표된 논문들을 다양한 각도로 분석하여 흐름을 도출해 주고 있다.

본 원고에서는 2장에서 MSR프로세스에 대해 간단히 설명 하고, 3장에서 MSR의 다양한 연구 분야를 소개한다. 그리고 4장에서 향후 이슈를 간략히 소개하며 끝을 맺는다.

* 본 연구는 2013년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2012R1A1A3011005).

2. MSR 프로세스 개요

그림 1은 MSR의 일반적인 프로세스를 보여준다. 프로세스의 시작은 데이터 추출(data extraction)인데, 여기서 데이터는 소스 컨트롤 시스템(source control system), 버그 트래킹 시스템(bug tracking system), 설계 문서(design document), 커뮤니케이션 아카이브(communication archives) 등 다양한 소스로부터 수집된다. 그런데 과거 MSR연구를 살펴보면, 다양한 데이터 소스 중에서 주로 소스 컨트롤 시스템 및 버그 트래킹 시스템이 사용되는 경향이 있었다[2]. 데이터 처리(data processing)에서는 처리할 데이터의 유형에 따라 텍스트, 트리, 자연어, 그래프, 벡터 등으로 나뉘어 처리되고 이후

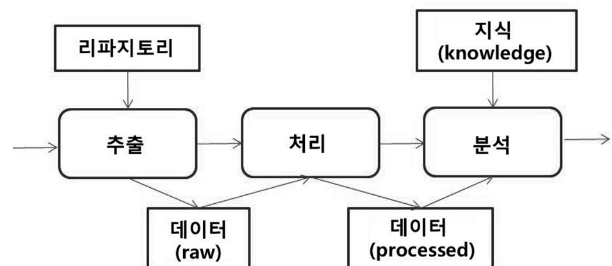

그림 1 MSR 프로세스[2]

표 1 MSR 프로세스에서의 이슈([2])

단계	이슈
추출	**리파지토리 유형** – 버전 관리 시스템(CVS, SVN, Git 등) – 버그 트래킹 시스템(BugZilla, Jira, Trac 등) – 커뮤니케이션 아카이브(이메일, 메일링 리스트, 메신저, 포럼 등) – 통합환경(Mylyn, IBM Jazz) – 코드 리파지토리(Google code, Source Forge 등) – 기타: 설계문서, 배치 로그(deployment log), 충돌 리포트(crash report)
처리	**처리 데이터 유형** – 소스코드: 텍스트, 트리 – 자연어: 코멘트, 버그 리포트, 메일링 리스트 – 그래프: 소스 코드, 개발자 관계 등 – 벡터: 속성 표현
분석	**데이터 마이닝 알고리즘** – classification, regression, association, clustering **최종 목적**(purpose) – 버그, 변경, 팀 활동, 검증(validation), 소스코드 이해, 개발 및 진화 이해 등

적절한 알고리즘에 따라 필요한 목적에 맞는 분석이 수행된다. 표 1에서는 MSR 프로세스 각 단계에서의 이슈가 되는 키워드를 요약하고 있다.

3. MSR의 이용분야 및 동향

올해(2013년)에 개최된 MSR 학술대회에서, 과거 10년간 발표되었던 논문들을 대상으로 동향을 다각도로 분석한 결과가 발표되었다[5,6]. MSR의 서베이 연구들[2,3]이 MSR의 개별 연구 주제에 초점을 맞추어 소개하였다면, 이 두 연구는 이제까지 수행된 MSR 연구의 큰 흐름을 파악하는데 도움을 준다. 본 장에서는 이 두 연구의 결과를 간단히 소개하고, 이후 세부적인 연구분야에 대해 기술한다.

[5]에서는 지난 MSR 컨퍼런스에 발표된 연구들을 대상으로 텍스트 마이닝을 통해 다음과 같은 결과를 도출하였다: 연구 주제 중 '변경' 및 '소프트웨어 진화'가 차지하는 비중이 높았고, 그 대상은 오픈 소스 시스템이었으며 Eclipse가 특히 여러 연구에서의 분석 대상이 되었다. 그리고 CVS와 BugZilla가 주로 사용된 데이터 소스였으나, SVN, Git, Gira도 점차 많아지는 추세를 보였다. 또한 기존 MSR 연구는 개발자나 테스터 같은 기술인력 위주로 수행되었으나, 프로젝트 관리에 참여하는 비기술 의사결정권자(non-technical decision maker)에 대한 고려가 부족한 실정이다. [6]에서는 2005년부터 2012년까지 MSR학회에서 발표된 도합 117편의 full paper를 대상으로 리뷰하고 이들로부터 의미 있는 268개의 코멘트를 추출, 정리하였다. 여기서 코멘트는 특정 주제에 대해 논문의 저자가 직접적으로 알려주는 정보들을 말한다. 이들은 연구들의 큰 주제를 다음과 같이 MSR의 프로세스에 준하여 다음 네 가지로 분류하였다:

주제 1. 데이터 획득 및 준비(data acquisition and preparation)

다양한 데이터 소스로부터 어떤 데이터를 어떻게 추출하느냐에 대한 주제이며, 그림 1의 '추출' 작업에 해당된다.

주제 2. 통합(synthesis)

획득된 데이터를 정리(clean-up)하는데 적용되는 알고리즘들(클러스터링, 분류, 예측, 그외 기계학습 알고리즘 등)에 대한 주제이며, 그림 1의 처리 및 분석작업의 일부와 관련된다.

주제 3. 분석(analysis)

수행된 연구의 직접적인 결과와 관련된 주제이며, 그림 1의 분석 작업에 해당된다.

주제 4. 공유 및 복제(sharing and replication)

이 주제는 외부 검증 및 연구 재현을 위한 데이터와 도구 공유에 대한 것이다.

Hemmanti 등은 117개의 268 코멘트들을 분석하여 각 년도 별 주제의 개수를 산정하였다. 표 2는 최종 개수를 보여주는데, '주제 1'이 전체적으로 68%라는 높은 비중을 차지하고 있으나, 추이를 살펴보면 '주제 1'의 비중은 점차 줄고 있고, '주제2'와 '주제3'의 비중이 높아졌다. 즉 데이터 추출 연구는 충분히 성숙하였고, 이후 처리 및 분석 부분의 비중이 증대되고 있음을 알 수 있다. 또한 '주제 4'인 데이터 공유 및 재현은 상대적으로 연구가 부족한 편이다.

그 후 이들은 코멘트를 바탕으로 최종 16개의 시사점을 도출하였는데, 이 중 한 해에 다섯 편 이상의 논문들이 다룬 주요 주제들을 따로 다음과 같이 정리하였다: SCM(Source Control Management) 리파지토리는 생각하지 못한 여러 노이즈(noise)가 있을 수 있고, 추출된 데이터를 모델링할 때 편향(skew)되지 않도록 주의해야 한다. 또한 결과에 대한 수동 검증이 필요하며, 정확도(precision)와 회수율(recall)이 주요 평가지표이

표 2 주제별 코멘트 개수(2005~2012)([6])

주제 1	주제 2	주제 3	주제 4	총계
180	33	40	15	268

표 3 이용 목적에 대한 상세 분류([2])

키워드	세부 분류
버그	버그수정, 중복 버그 검출, 예측, 버그 해결자(re-solver) 추천, IR(information retrieval) 적용
변경	예측, 리팩토링, API변경, 변경 패턴
팀 활동	개발자 공헌도 및 전문성, 도구 지원, 유용한 정보 제공
이해	시각화, 식별자(identifier)관련, 연산 기록(re-cording operation)
검증	메트릭, 도구, 클론, 버그
개발 및 진화	개발관련, 진화 관련

기는 하지만, 상황에 따라 다른 적합한 척도를 찾아야 한다. 마지막으로, 통계적 유의성(statistical significance)은 두 기법을 비교할 때 필요하지만, 특정 기법의 효율성을 말해주지 않으므로 시각화(visualization)와 같은 실용적인 방법을 고려해야 한다.

다음 3.1부터 3.6절에는 기 발표된 MSR의 연구들을 분류하고 각 주제 별로 관련된 연구들을 소개한다. [2]에서는 주제별 키워드를 버그, 변경, 팀 활동, 코드 이해, 검증, 개발 및 진화 지원으로 나누었고, 본 원고에서도 이에 준하여 기술하되, 2007년부터 최근 2013년에 발표된 연구들 위주로 소개하고자 한다. 웹 페이지 [7]에서는 본 원고에서 빠진 2007년 이전 연구들을 포함, 2011년도에 발표된 연구들까지 테이블로 요약되어 있다. 표 3은 큰 카테고리에 속한 여러 기존 연구들을 주제 별로 다시 재 분류한 것으로, 실제 연구와의 연결은 [2]에서 볼 수 있다.

3.1 버그 관련 활동 지원: 수정, 검출, 예측 등

버그는 소프트웨어 변경의 주요 원인 중의 하나이며, MSR에서도 버그와 관련된 다양한 연구들이 쏟아져 나왔다. 버그 수정 및 검출, 버그 예측, 버그 심각도 구분, 버그 해결자 찾기 등 다양한 주제들이 그것이다.

버그 수정과 관련하여, 버그 수정 패턴 발견 및 이해, 자동 버그 수정 등의 연구가 있다. Eyolfson 등은 커밋(commit)의 패턴과 해당 커밋의 버그 발생과의 상관관계를 연구하였다[8]. Linux Kernal과 PostgresSQL을 탐색한 결과, 자정부터 4 AM사이의 커밋이 버그가 발생할 가능성이 높고, 매일 커밋하는 개발자들의 커밋이 버그 발생 가능성이 낮으며, 요일이 미치는 영향은 프로젝트마다 가변적이라고 주장하고 있다. 전자제품내의 소프트웨어 구현에 쓰이는 HDL(Hardware description language)에서의 버그 수정 패턴을 분석한 연구가 있다[9]. 버그를 줄이기 위하여 우선 여러 에러 유형을 찾아 이해하고, 빈번히 발생하는 유형에 대처하는 것이 필요하다. [9]에서는 네 개의 하드웨어 프로젝트에서 버그 수정 히스토리를 분석하여 25개의 버그 수정 패턴을 정의하고 각 패턴의 빈도를 계산하였다. 이들은, 다수의 버그 수정 패턴이 할당문(assignment statement)과 if문에서 발생한 것을 발견하였다.

사용자가 보고한 버그가 버그 트래킹 시스템에 이미 존재할 경우, 버그 리포트가 중복되었다고 한다. 버그 리포트의 중복이 처리되지 않으면, 버그 해결 비용이 높아지게 된다. 중복 버그 리포트 검출과 관련, 대개 텍스트 기반 분석이 수행된다[10-12]. 버그 리포트를 텍스트로 간주하여, 자연어 처리를 통해 텍스트의 유사도를 측정하는 방식이다. Wang 등은 자연어 처리뿐 아니라 실행 정보(execution information)를 이용하여 중복 버그 검출의 회수율(recall)을 상승시켰다[13].

소프트웨어 리파지토리를 분석하여, 버그 수정 패턴을 찾고, 미래에 버그 발생 가능성이 높은 코드를 찾는 연구들이 있다. 소스 코드의 변경 히스토리로부터 버그 수정 패턴을 찾고, 이후 프로젝트별 버그를 예측하는 BugMem[14]이 제안되었으나, 일부 소소한 버그를 찾는 데 한계가 있고 충분한 변경 히스토리를 필요로 한다. 이후 [15]에서는 버그 예측에 다음과 같은 지역성(locality)을 고려하였다: 변경된 엔티티 및 새 엔티티는 결함을 가져올 가능성이 높으며, 최근 결함을 유도한 엔티티는 다른 결함을 초래할 가능성이 있고, 논리적으로 결합된 '인근' 엔티티들도 결함을 유발시킬 수 있다. 이러한 가정 하에, 결함이 잦은 엔티티들의 현재 리스트를 운용하는 캐시를 두고, 캐싱(caching)을 위한 몇몇 알고리즘을 제안하고 있다. 현재 코드가 캐시 내의 코드와 유사하면, 개발자들에게 버그 가능성을 알려주게 되며, 파일 레벨에서는 73-95%의 정확도를, 메소드 레벨에서는 46-72%의 정확도를 보였다.

버그 수정 패턴 및 예측 연구 이외에도, 버그 심각성이나 버그 해결자에 대한 연구들이 수행되었다. 이전에 수동으로 산정하던 버그의 심각성(severity)을 텍스트 마이닝을 통해 예측하고 검증하는 연구[16,17], 뉴럴 네트워크를 통해 버그 리포트의 우선순위를 예측하는 연구 [46] 등이 있다. 또한 발생한 버그 해결을 위해, 자동으로 적합한 개발자에게 전달하는 연구들이 다수 존재한다[18,19]. 이들 대부분은 버그 리포트와 코드의 어휘(vocabulary), 코드 수정한 개발자가 변경한 부분 등을 이용하여 개발자의 전문영역을 결정하고, 발생한 버그 리포트와의 유사도를 통해 순위별로 적합한 개발자를 추천해주는 방식을 택한다. [19]에서 제안된 기법은, 커밋 정보를 주로 이용하여, 버그를 적합한 개

발자에게 넘겨준다. 우선 어휘분석에 기반하여 버그 리포트와 코드 사이의 유사도를 결정하여 특정 버그와 관련된 코드 엔티티(예: 클래스)를 찾는다. 이후 “과거에 해당 코드에 의미있는 변경을 수행한 개발자가 이후에도 그럴 가능성이 높다”는 가정하에 커밋 시간 정보를 이용하여 개발자 코드 벡터 및 파일 변경 벡터를 구축하고 이들 사이의 유사도를 통해 적합한 개발자를 찾는다.

3.2 변경 관련 활동 지원: 변경 탐지, 예측, 가이드

소프트웨어 노화(aging)의 부정적인 영향을 줄이는 방법으로, 변경 자체를 개발 프로세스의 중심에 두어야 한다 는 주장([20])이 있을 만큼, 변경은 프로젝트 수행 시의 필수 활동이다. 이전 절의 주제인 버그 수정도 코드 변경이며 다음 절에 소개한 개발자 관련 지원 활동도 개발자가 수행한 변경 작업과 연관이 있다. 본 절에서는 ‘변경’ 그 자체에 초점을 맞춘 MSR 작업에 대해 소개하고자 한다.

함께 커밋된 변경 데이터들의 연관성을 고려해야 할 필요성을 탐색한 연구가 있다[21]. 이 연구에서는 각기 다른 목적으로 수행된 변경들이 함께 커밋된 경우, 이들을 ‘뒤엉킨 변경(tangled change)’라고 하며 이 뒤엉킨 변경들을 구분할 필요가 있다고 주장한다. 즉 소스 코드 저장소 안에서 연관성이 낮은 변경 정보들이 함께 통합되어 관리된 경우에 변경 정보들을 기반으로 향후의 변경 및 결함 예측을 수행하는 분석 방법에 영향을 미칠 수 있다. 예를 들어 한 커밋 내의 세 변경 중에서, 한 변경 A만 특정 버그에 관련된 것이고 나머지는 그와 무관하다고 할 때, 해당 버그 관련 분석에서는 A 만 분석하는 것이 낫다는 것이다. [21]에서는 뒤엉킨 변경들 중에서 파일거리, 패키지 거리, 호출 그래프, 변경 결합도 등을 이용하여 연관성 있는 변경 정보를 구분하는 방법을 제안하였다. 또한 실험을 통하여 연관성이 구분된 변경 집합이 그렇지 않은 경우보다 더 나은 결과를 보였다.

리팩토링(refactoring)은 코드 변경의 또 하나의 주요 원인이다. [22]에서는 리팩토링과 결함과의 연관성을 탐색한 결과, 리팩토링과 결함이 역으로 관련이 있음을 발견하였고, 결과적으로 리팩토링이 결함의 개수를 줄이는데 중요한 역할을 한다고 주장한다. API 변경은 리팩토링과 밀접한 관련이 있다. Taneja 등은 API변경의 80%가 리팩토링과 관련이 있음을 보였으며, 어플리케이션을 자동으로 업그레이드 할 수 있도록 리팩토링 후보를 찾아주는 방법을 제안하였다[23]. Nguyen 등은 이미 새로운 라이브러리로 마이그레이션(migration)을 한 다른 클라이언트 코드들로부터 API 사용 적응 패턴(API usage adaptation pattern)을 학습하는 프레임워크를 수립하였다[45]. 제안된 프레임워크의 입력으로는, 클라이언트 어플리케이션의 현재 버전, 예전 및 현재 버전의 라이브러리, 그리고 이미 제대로 마이그레이션된 코드집합들이다. API 사용 적응 패턴에 대한 지식 베이스(knowledge base)를 수립한 후, 변경된 API를 이용하는 클라이언트 코드에 대해 가장 적합한 API 사용 패턴을 연결시켜 준다.

그외, 버그 수정 패턴과 마찬가지로 변경에 대해서도 변경 유형을 분류하고 변경 패턴을 검출하는 연구들이 다수 존재한다. 버전관리 시스템은 일반적으로 변경에 대해, 의미를 고려하지 않고 단순히 텍스트 차원에서 변경을 알려준다. 따라서 변경의 유형을 분석하여, 소스 코드 변경의 속성을 이해하는 것은 소프트웨어 진화 이해에 큰 도움을 준다. 함께 변경될 가능성이 높은 엔티티들이 서로 변경 결합되었다(change coupled)고 하는데, 변경 결합도는 향후 함께 변경될 엔티티들을 예측하거나 구성요소들 사이의 의존관계를 이해하는데 이용될 수 있다. Fluri와 Gall은 소스 코드 변경을 트리의 에디트(edit) 연산 조합으로 표현하고, 각 변경 유형을 중요도 레벨(significant level)로 나누었다([20]). 예를 들어 〈[body]removed functionality, DEL(M), crucial〉은, 처음부터 각각 “변경유형/적용된 단위연산/변경의 중요도”를 뜻한다. 이후 이들은 [24]에서 이전에 정의한 변경 유형을 이용하여, 프로젝트에서 함께 발생하는 변경 유형들이 있는지 추출하고 구체적으로 어떤 식으로 함께 변경되었는지 분석하였다.

3.3 개발 및 유지보수 인력 지원

리파지토리 마이닝을 이용하여, 개발자의 구현 작업에 직접적으로 도움을 주거나 관리자 입장에서 개발자들에 대한 기여 등의 정보를 제공하여 프로젝트 관리에 도움을 주기 위한 연구들이 수행되었다.

오픈소스 프로젝트에서 커밋을 주로 수행하는 핵심팀(core team)에 외부 개발자가 포함되려면 일정 이상의 신뢰가 쌓여야 하는데, 실제 어떤 경우에 신뢰가 수립되는지에 대해 고찰할 필요가 있다. Sinha 등은 개발자의 신뢰가 수립되는 경우에 대해 소셜(social) 관점과 테크니컬 관점으로 세 가설을 세우고 Eclipse 시스템을 대상으로 검증하였다[25]. H1은 버그와 관련하여 공헌을 세우는 경우이고 H2는 과거 프로젝트 경험이며 H3은 핵심 팀과 같은 조직에 있는 경우이며, 각 경우에 대해 이후 핵심 팀으로의 진입과 연관이 있을 것이라는 가설이다. 결과적으로 버그에 대한 공헌이 가장 중요한 요소라고 할 수 있었으며 소셜 요소(social

factor)인 H3도 어느 정도 타당성이 있었다. 이 결과를 기반으로 향후 팀원을 충원할 때 필요한 기준을 세울 수 있을 것이다. 개발자들의 기여도를 산정하는 연구도 있었다[26]. 여기서는 기존의 LoC(lines of code)와, 개발자들의 CF(contribution factor)를 결합하여 메트릭을 정의하였는데, CF는 개발자들의 각 행동들을 긍정적/부정적으로 가중치를 부여하여 정의하였다. 이 메트릭은 이론적으로는 검증하였으나, 실험적인 검증이 완료되지는 않았다. 개발자가 변경한 파일 정보를 바탕으로 전문성(expertise)을 평가하는 연구들이 있다[27,28]. [27]에서는 특정 파일이 그 디렉토리 정보를 이용하여 미리 전문영역이 결정되고, 해당 파일에 대한 개발자의 커밋 횟수로 그 전문영역을 정하였다. [28]에서는 개발자가 변경한 메소드와 이용한 메소드에 대해 개수 기반으로 각각 구현전문성, 사용전문성으로 나누어 평가하였다. 또한 유사도를 이용하여 개발자들 사이의 전문성 유사도도 평가하였다.

개발자들은, 어떤 코드가 왜 이렇게 작성되었으며 왜 이렇게 동작하는지 알기 위하여 버그 리포트, 체크인 메시지, 이메일 아카이브 등과 같은 다양한 데이터를 보고 상호연관 시키려 노력한다[29]. 이러한 작업을 수동으로 하는 것은 쉽지 않으므로 Holmes와 Begel은 단일 코드 요소에 대한 여러 히스토리 정보들을 보여주는 도구인 Deep Intellisense를 개발하였다[29]. 이는 current item view, people veiw, and event history view를 제공해 준다. Bradley와 Murphy도 역시 소스 코드의 히스토리를 통합해서 보여주는 도구인 Rationalizer를 개발하고 인터페이스 비교를 위한 모델을 수립한 후 이것을 Deep Intellisense와 비교하였다[30]. Rationalizer는 particular code line에 대하여 ‘when/who/why’의 관점으로 해당 데이터를 보여준다. 그리고 Deep Intellisense와의 비교를 통하여 두 도구의 장점을 취한 미래의 도구개발 가능성을 시사하고 있다.

3.4 시스템 이해도 향상

본 절에서는 리파지토리 데이터를 이용하여 소프트웨어 진화나 소프트웨어 자체의 이해도를 강화시켜주는 연구들을 소개한다. Davis 등은 소프트웨어 엔티티의 원출처(origin)를 Bertillonage 분석에 기반한 anchored signature matching을 통해 찾아주는 기법을 제안하였다[31]. 소프트웨어 엔티티의 원출처는, 유지보수뿐 아니라 라이선스 문제와도 얽혀있으므로 이를 알아둘 필요가 있다. [31]에서는, 이진 클래스 파일(binary class file)을 소스파일과 매치시켜주기 위하여 이진 자바 아카이브(binary JAVA archives)에 대한 Bertillonage 메트릭을 제안하였다. Bertillonage는 정확성보다 효율성을 중시하는 기법으로, 찾으려는 대상의 특징적인 부분을 데이터로 추린 후, 후보군들을 추린 데이터에 기반하여 조직하고 불필요한 후보들을 삭제하여 탐색 범위를 좁히려는 접근법이다. [31]에서는 이를 위해 anchored signature matching 기법을 이용한다. 클래스 각각에 대해, 대상 클래스 자체의 타입 시그니처와 포함된 메소드들의 타입 시그니처들로 표현한다. 그 후, 유사도와 포함율(inclusion)을 이용하여 후보군에서 찾으려는 클래스에 대응되는 타겟을 찾는다.

코드 내의 식별자(identifiers)는 개발자의 의도와 같은 중요한 정보를 담고 있어 프로그램 이해에 있어 중요한 역할을 한다[32]. 식별자 분석에서, 식별자를 적절히 쪼개는(splitting) 것이 필요한데, 단순히 네이밍 관습(naming convention)만을 이용해서는 좋은 결과를 얻기가 어렵다. Enslen 등은 소스 코드 내의 단어 빈도수에 기반한 스코어 테크닉(scoring technique)을 사용하여 식별자를 자동으로 나누는 Samurai 접근법을 제안하고, 실험을 통해 기존 접근법보다 효율적임을 보였다[33]. [34]에서는 식별자를 적절히 분할했는지 비교 평가해주는 데이터 집합 구축 과정을 보여주었다. Eshkevari 등은 프로그램에서의 식별자 명칭짓기(naming)에 대해 동의어(synonym), 반의어(antonym), 상위어(hypernym), 하위어(hyponym)의 관점으로 탐색하고 이들이 프로그램 이해에 미치는 영향을 살펴보았다[32]. TomCat과 Eclipse-JDT를 분석한 결과, 재명명(renaming)은 특정 시간 프레임 내 일부 커미터들에 의해 많이 발생하였다. 또한 클래스 인터페이스에 많이 발생하였고, 동의어뿐 아니라 반의어나 부분어(meronym)도 많이 있었는데, 이는 에러 수정을 의미할 가능성이 높다고 할 수 있다. 그리고 오자수정이나 접두어/접미어 변경과 같은 짧은 문자열의 수정이 많았으나, 명사에서 동사로 바뀌는 등의 문법변환은 많지 않았다. 최종적으로 이들은 재명명이 개발자의 도메인 모델에 대한 변경을 반영한다고 주장한다.

프로그램 및 그 진화에 대해 이해할 때, 단지 현재의 스냅샷(snapshot)만 보거나, 연속된 두 버전 사이의 차이를 보는 경우가 많은데 이 차이는 수많은 변경을 포함하므로 개별 변경을 알기가 어렵다. Omori와 Maruyama는 소스 코드에서 행해지는 모든 edit 연산을 기록하는 기법을 제안하고 있다[35]. 그리고 OperationRecorder라는 도구를 Eclipse 플러그인으로 구현 및 적용하여 소규모 프로그램에서는 충분히 실용적임을 보이고 있다.

3.5 아이디어 및 테크닉의 실험적 검증

본 절에서는, 소프트웨어 진화에 대해 다양한 가설을 세우고 이에 대해 실험적으로 검증하거나, 비교적 널리 이용되는 도구를 검증하는데 MSR을 적용한 연구들을 소개한다.

소프트웨어 내의 중복 코드를 나타내는 코드 클론(code clone)의 속성 관찰에 대해 다수의 연구들이 수행되었다. 코드 클론은 일반적으로 유지보수에 좋지 않은 영향을 미치므로 클론 검출 및 리팩토링 등을 통한 제거가 필요하다고 알려져 있다. 코드 클론들이 일관성있게 변경되어야 한다는 주장도 있지만[36], Krinke 등은 단지 50%의 클론들만이 함께 변경되며, 이렇게 함께 변경되는 비율은 버전이 증가해도 늘어나지 않았다는 것을 보였다[37]. Lozano 등은 클론을 포함한 어플리케이션의 변경률을 찾는 CloneTracker라는 도구를 개발하고 DnsJAVA에 적용하여 클론 코드들이 변경 가능성(changeability)이 높다 것을 발견하였으나 대상 시스템 자체가 2인의 개발자에 의해 개발되었으므로 일반화시키기 어려운 부분이 있다[38]. 이후 연구 [39]에서도 clone이 변경 가능성에 미치는 영향력을 분석한 결과 클론이 있을 경우 유지보수노력이 늘어날 수 있다는 것을 발견하였으나 어떤 체계적인 관계(systematic relation)는 찾기 어렵다고 하였다. 그외 Rahman 등은 클론의 속성에 대해 실험적으로 검증하였다[40]. 클론과 결함발생(defect-proneness)사이의 관계를 분석한 결과, 대부분의 버그는 클론들과 심각한 관계는 없었으며, 클론들은 비클론 코드들보다 오히려 결함 발생 경향이 덜하였다. 또한 빈번히 복제되는 코드들이 에러발생률이 높다는 것 역시 검증되지 않았다는 것을 보였다. 즉 이 연구에 따르면 통념과는 달리 클론은 리팩토링을 필요로 하는 코드 악취(bad smell)가 아니라는 것이다.

코딩 표준(coding standard)은 프로그래밍 시에 일종의 경험적 규칙으로, 소프트웨어의 품질을 향상시킨다고 알려져 있다. 이러한 코딩 표준와 결함발생과의 상관관계를 보인 연구도 있다[41]. '릴리즈/파일/모듈' 단위에 대해, 표준을 어기는 것이 결함유발률을 높이는가에 대하여 탐색하였다. 그 결과, 코딩 표준 중 10 규칙들에 대해, 이들의 준수 여부가 결함의 주요 예측자(predictor) 역할을 함을 보였다. 버그 수정 프로세스가 프로세스 데이터 품질에 어느 정도 영향을 받는지를 검증한 연구가 있다[42]. 오픈 소스와 비 오픈 소스에 적용한 결과, 프로세스 데이터의 품질 및 속성이 버그 수정에 영향을 미치는 것이 드러났다. 예를 들어, 빈 커밋(empty commit)의 비율이 버그 리포트 품질과 연관이 있으며, 버그리포트 개수를 이용하여 측정한 결과물(product) 품질은 프로세스 데이터 품질에 영향을 미치는 것을 보였다.

3.6 소프트웨어 개발 및 진화 양상 이해

MSR 연구는 장기간에 축적된 리파지토리 데이터를 분석하는 것에 기반하므로, 대체로 실험적, 실증적인 성향이 강하다. 대량의 데이터 분석을 통하여 소프트웨어의 개발 및 진화의 보편적 속성을 실험적으로 도출하려는 시도들이 있어왔다.

개발자들은 작업상태를 기억하고 공유하기 위해 작업세션(work session)에 대해 기록하는데, 이 역시 노력이 소요되는 일이다. Maalej와 Happel은 이러한 기록들의 패턴을 탐색하여 작업일지 생성을 자동화 할 수 있는 단초를 마련하고자 하였다[43]. 이들은 8년간 2000명의 개발자들이 작성한 75000건의 작업기술(work description)을 분석하였다. 그 결과, 어휘나 기술패턴 등의 컨텐츠와 작업 노트를 작성하는 시간(day time) 및 소요된 시간 등에서 유사점이 있음을 발견하였다. 그리고 작업 기술에 빈번히 사용되는 다음과 같은 패턴을 발견하였다: "ACTION concerning ARTIFACT because of CAUSE". 이들은 개발자들 전체 시간의 3-6%를 작업기술을 작성하기 위하여 쓰고 있으며, 작업 기술도 크게 상세하지는 않다는 것을 발견하고, 이들이 발견한 결과를 자동 작업일지 생성에 적용할 수 있다고 주장하였다. Hindle 등은 비교적 간과된, 많은 파일을 포함한 대형 커밋(large commit)의 속성을 이해하기 위한 실험을 수행하였다[44]. 9개의 오픈 소스에서 대형 커밋을 수동으로 분류하고 소형 커밋(small commit)과 비교하였다. 비교 결과 대형 커밋은 '완전적(perfective)'인 경향이 있고, 소형 커밋은 '교정적(corrective)'인 경향이 있었다. 교정적 속성은 국소적으로 에러를 수정하는데 반해, 완전적 속성은 보다 전역적(global)이며, 효율성, 성능, 유지보수성 향상과 관련이 있다. 이들은 대형 커밋이 개발자의 개발 습관을 반영하며, 프로젝트가 개발되는 방식을 통찰할 수 있게 해준다고 주장한다.

Krishnan 등은 소프트웨어 프로덕트 라인에서의 공통(common) 및 가변(variable) 컴포넌트의 고장(failure)/변경(change) 트렌드를 분석하고 고장과 변경 사이의 관계를 탐구하였다[45]. 진화하는 프로덕트 라인으로 간주하고 있는 Eclipse 시스템을 대상으로 실험한 결과 공통 컴포넌트의 경우 시스템이 진화할수록 주요 고장(serious failure)의 수도 줄고 변경도 줄지만 5개 이상의 프로덕트에서 재사용되는 가변 컴포넌트의 경우 예상과는 달리 균일한(uniform) 감소 패턴을 보이지는 않아, 좀더 정밀한 연구의 필요성을 이야기한다.

4. 결 론

본 원고에서는 MSR분야에서 기존 연구의 동향 파악에 도움을 주기 위하여, MSR의 개요를 설명하였다. 그리고 기 발표된 MSR 연구들을 목적 별로 분류하여 소개하였다. 마지막으로, 최근 MSR 서베이 및 동향분석 연구([2,4,5])에서 제시된, 향후 MSR 연구가 직면한 이슈에 대해 소개를 하고 끝맺는다.

1) 마이닝 과정은 상당한 시간 및 자원이 소요되는 작업으로, 프로세스를 최적화해 줄 수 있는 도구나 프레임워크가 필요하다.

2) 기존 연구들이 대부분 CVS나 SVN에 기반하므로, 다른 데이터 소스들을 쉽게 이용할 수 있도록 해야 한다. 그리고 데이터를 추출하여 이용할 때, 버전관리 시스템, 버그 트래킹 시스템 등 다양한 리파지토리 사이의 연결은 보다 의미있는 결과를 얻기 위하여 꼭 필요하다.

3) 프로젝트에 비기술인력의 동참을 위하여, MSR 연구에 관리(management) 관련 용어를 좀더 채택할 필요가 있다.

4) 기존 연구들은 획득용이성 등의 이유로 오픈 소스 위주로 분석하였으나 그 대상을 비오픈소스(closed source)와 상용 소스(commercial source)로 확장할 필요가 있다.

참고문헌

[1] MSR, http://www.msrconf.org

[2] Jung, W., Lee, E., and Wu, C., "A Survey on Mining Software Repositories," IEICE Trans. Inf. & Syst., vol. E95-D, no. 5, pp. 1384-1406, 2012.

[3] Kagdi, H., Collard, M. L., and Maletic, J. I., "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, no. 2, pp. 77-131, 2007.

[4] Hassan, A.E., "The road ahead for mining software repositories," Proc. Frontiers of Software Maintenance, pp. 48-57, 2008

[5] Demeyer, S., Murgia, A., Wyckmans, K., and Lamkanfi, A., "Happy Birthday! A Trend Analysis on Past MSR Papers," Proc. of MSR2013, pp. 353-362, 2013.

[6] Hemmanti, H., Nadi, S., Baysal, O., Kononenko, O., Wang W., Holmes R., and Godfrey M.W., "The MSR Cookbook: Mining a Decade of Research," Proc. of MSR 2013, pp. 343-352, 2013.

[7] http://zorba.knu.ac.kr/research/MSR_Survey/overall_table.html

[8] Eyolfson, J., Tan, L., and Lam, P., "Do time of day and developer experience affect commit bugginess," Proc. of MSR2011, pp.143-152, 2011.

[9] Sudhakrishnan, S., Madhavan, J. T., James, Whitehead, Jr. E., Renau, J., "Understanding bug fix patterns in Verilog," Proc. of MSR2008, pp.39-42, 2008.

[10] Jalbert, N. and Weimer, W., "Automated duplicate detection for bug tracking systems," Proc. Conf. on Dependable Systems and Networks, pp. 52-61, 2008.

[11] Runeson, P., Alexandersson, M., and Nyholm, O., "Detection of duplicate defect reports using natural language Processing," Proc. Int'l Conf. on Software Eng., pp.499-510, 2007.

[12] Alipour, A., Hindle, A., and Stroulia, E., "A Contextual Approach towards Mode Accurate Duplicate Bug Reports Detection," Proc. of MSR 2013, pp. 183-192, 2013.

[13] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J., "An approach to detecting duplicate bug reports using natural language and execution information," Proc. Int'l Conf. on Software Eng., pp.461-470, 2008

[14] Kim, S., Pan, K., and James, Whitehead, Jr. E., "Memories of bug fixes," Proc. Int'l Symposium on Foundations of Software Engineering, pp. 35-45, 2006.

[15] Kim, S., Zimmermann, T., James, Whitehead, Jr. E., and Zeller, A.. "Predicting faults from cached history," Pro. of Int'l Conf. on Software Engineering, pp. 489-498, 2007.

[16] Lamkanfi, A., Demeyer, S., Giger, E., and Goethals, B., "Predicting the severity of a reported bug," Proc. of MSR2010, pp. 1-10, 2010.

[17] Gegick, M., Rotella, P., and Xie, T., "Identifying security bug reports via text mining: an industrial case study," Proc. of MSR2010, pp. 11-20, 2010.

[18] Matter, D., Kuhn, A., and Nierstrasz, O., "Assigning bug reports using a vocabulary-based experties model of developers," Proc. of MSR2009, pp. 131-140, 2009.

[19] Kagdi, H., Gethers, M., Poshyvanyk, D., and Hammad, M., "Assigning change requests to software developers," Journal of Software: Evolution and Process, vol. 24, no. 1, pp. 3-33, 2012.

[20] Fluri, B. and Gall, H.C., "Classifying change types for qualifying change couplings," Proc. of Int'l Conf. on Program Comprehension, pp. 35-45, 2006.

[21] Kim, H., and Zeller, A., "The Impact of Tangled Code Changes," Proc. of MSR2013, pp. 121-130, 2013.

[22] Ratzinger, J., Sigmund, T., and Gall, H. C., "On the

relation of refactoring and software defects," Proc. of MSR2008, pp.35-38, 2008.
[23] Taneja, K., Dig, D., and Xie, T., "Automated detection of api refactorings in libraries," Proc. Int'l Conf. on Automated Software Eng., pp. 377-380, 2007.
[24] Fluri, B., Giger, E., and Gall, H.C., "Discovering patterns of change types," Proc. of Automated Software Eng., pp. 463-466, 2008.
[25] Sinha, V.S., Mani, S., and Sinha, S., "Entering the circle of trust: developer initiation as committers in open-source projects," Proc. of MSR2011, pp.133-142, 2011.
[26] Gousios, G., Kalliamvakou, E., and Spinellis, D., "Measuring developer contribution from source repository data," Proc. of MSR2008, pp.129-132, 2008.
[27] Alonso, O., Devanbu, P.R., and Gertz, M., "Expertise Identification and Visualization from CVS," Proc. of MSR2008, pp. 125-128, 2008.
[28] Schuler, D. and Zimmermann, T., "Mining Usage Expertise from Version Archives," Proc. of MSR2008, pp. 121-124, 2008.
[29] Holmes, R. and Begel, A., "Deep Intellisense: a tool for rehydrating evaporated information," Proc. of MSR2008, pp. 23 -26, 2008.
[30] Bradley, A. W. J. and Murphy, G.C., "Supporting software history exploration," Proc. of MSR 2011, pp.193-202, 2011.
[31] Davies, J., German, D.M., Godfrey, M.W., and Hindle, A., "Software bertillonage: finding the provenance of an entity," Proc. of MSR2011, pp.183-192, 2011.
[32] Eshkevari, L.M., Arnaoudova, V., Di, Penta, M., Oliveto R., Guéhéneuc, Y., and Antoniol, G., "An exploratory study of identifier renamings," Proc. of MSR2011, pp. 33-42, 2011.
[33] Enslen, E., Hill, E., Pollock, L., and Vijay-Shanker, K., "Mining Source Code to Automatically Split Identifiers for Software Analysis," Proc. of MSR2009, pp. 71-80, 2009.
[34] Binkley, D., Lawrie, D., Pollock, L., Hill, E., and Vijay-Shanker K., "A Dataset for Evaluating Identifier Splitters," Proc. of MSR2013, pp. 401-404, 2013.
[35] Omori, T. and Maruyama, K., "A Change-Aware Development Environment by Recording Editing Operations of Source Code," Proc. of MSR2008, pp. 31-34, 2008.
[36] Aversano, L., Cerulo, L., and Penta, M. D., "How clones are maintained: an empirical study," Proc. European Conf. on Software Maintenance and Reengineering, pp. 81-90, 2007.
[37] Krinke, J., "A study of consistent and inconsistent changes to code clones," Proc. Working Conf. on Reverse Eng., pp. 170-178, 2007.
[38] Lozano, A., Wermelinger, M., and Nuseibeh, B., "Evaluating the harmfulness of cloning: a change based experiment," Proc. of MSR2007, pp.18-21, 2007.
[39] Lozano, A. and Wermelinger, M., "Assessing the effect of clones on changeability," Proc. of Int'l Conf. on Software Maintenance, pp.227-236, 2008.
[40] Rahman, F., Bird, C., and Devanbu, P., "Clones: what is that smell?," Proc. of MSR 2010, pp. 72-81, 2010.
[41] Boogerd, C. and Moonen, L., "Evaluating the relation between coding standard violations and faults within and across software versions," Proc. of MSR 2009, pp. 41-50, 2009.
[42] Bachmann, A. and Bernstein, A., "When process data quality affects the number of bugs: correlations in software engineering datasets," Proc. of MSR 2010, pp. 62-71, 2010.
[43] Maalej, W. and Happel, H., "From work to word: how do software developers describe their work?," Proc. of MSR2009, pp. 121-120, 2009.
[44] Hindle, A., German, D. M., and Holt, R., "What do large commits tell us? A taxonomical study of large commits," Proc. of MSR2008, pp. 99-108, 2008.
[45] Nguyen, H. A., Nguyen, T. T., Wilson, Jr. G., Nguyen, A. T., Kim, M., and Ngyuen, T. N., "A Graph-based approach to API usage adaptation," Proc. Int'l Conf. on Object-oriented programming systems languages and applications, pp.302-321, 2010.
[46] Yu, L., Tsai, W.T., Zhao, W., and Wu, F., "Predicting defect priority based on neural networks," Proc. Int'l Conf. on Advanced Data Mining and Applications, pp. 356-367, 2010.

약 력

이 은 주

1997 서울대학교 계산통계학과(학사)
1999 서울대학교 전산과학과(석사)
2005 서울대학교 전기컴퓨터공학부(박사)
2005~2005 서울대학교 공과대학 BK 박사 후 연구원
2005~2006 2월 삼성종합기술원 전문연구원
2006~현재 경북대학교 컴퓨터공학과 부교수
관심분야 : 소프트웨어 메트릭, 소프트웨어 리파지토리 마이닝, 변경 분석, 웹 공학
E-mail : ejlee@knu.ac.kr