Journal of The Korea Society of Computer and Information
Vol. 19, No. 9, September 2014

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.9.103

다형의 버그 추적 시스템 마이닝 및 분석을 위한 저장소 독립 모델 설계

이 재 권*, 정 우 성*

Designing a Repository Independent Model for Mining and Analyzing Heterogeneous Bug Tracking Systems

Jae-Kwon Lee*, Woo-Sung Jung*

요 약

본 논문은 다양한 버그 추적 시스템으로부터 추출한 데이터를 통합하여 단일 저장소 모델을 제공하는 UniBAS(Unified Bug Analysis System)를 제안한다. UniBAS는 MSR(Mining Software Repositories) 연구 과정에서의 저장소 추출, 데이터 가공이나 모델 생성과 같은 공통적인 반복 작업을 줄이고, 관련 연구자가 상위 수준의 연구에 보다 집중할 수 있도록 함으로써 해당 연구 수행에 발생하는 복잡도와 비용을 줄여준다. 또한, UniBAS는 데이터 추출 뿐 아니라 질의 기반 분석에 필요한 테이블, 뷰 및 저장 프로시저 등을 자동 생성하며, 수집한 데이터 관리와 외부 도구와의 연동을 위해 다양한 형식의 파일을 생성할 수 있다. 사례 연구로 UniBAS의 유용성을 검증하기 위해 Mozilla사이트의 Firefox프로젝트를 대상으로 실제 중복 버그 리포트를 탐지하는 실험을 진행하였다. 이 과정에서 자동 추출된 자료를 대상으로 질의와 분석이 유연하게 이루어질 수 있었으며, 다양한 자연어 처리 알고리즘 적용을 통해 유효한 실험 결과를 얻을 수 있었다.

▸ Keywords : 분소프트웨어 저장소 마이닝, 저장소 독립 모델, 버그 트래킹 시스템

Abstract

In this paper, we propose UniBAS(Unified Bug Analysis System) to provide a unified repository model by integrating the extracted data from the heterogeneous bug tracking systems. The UniBAS reduces the cost and complexity of the MSR(Mining Software Repositories) research process and enables the researchers to focus on their logics rather than the tedious and repeated works such as extracting repositories, processing data and building analysis models. Additionally, the system not only extracts the data but also automatically generates database tables, views and stored procedures which are required for the researchers to perform query-based analysis easily. It can

•제1저자 : 이재권 •교신저자 : 정우성
•투고일 : 2014. 8. 26, 심사일 : 2014. 9. 4, 게재확정일 : 2014. 9. 17.
* 충북대학교 컴퓨터공학과(Dept. of Computer Engineering, Chungbuk National University)

also generate various types of exported files for utilizing external analysis tools or managing research data. A case study of detecting duplicate bug reports from the Firfox project of the Mozilla site has been performed based on the UniBAS in order to evaluate the usefulness of the system. The results of the experiments with various algorithms of natural language processing and flexible querying to the automatically extracted data also showed the effectiveness of the proposed system.

▸ Keywords : Mining Software Repositories, Repository Independent Model, Bug Tracking System

I. 서 론

소프트웨어의 규모와 복잡도는 증가하는 추세이며, 개발 과정에서 생기는 버그의 유형도 점점 다양하고 복잡해지고 있다. 이러한 버그를 효과적으로 관리하기 위해 BugZilla[1], Mantis[2], Trac[3] 등의 버그 추적 시스템(Bug Tracking System)들이 제시되었고, 버그의 내용과 발생환경, 상태 등의 정보를 사용자가 쉽게 추적할 수 있게 되었다. 버그 추적 시스템은 소프트웨어 개발 과정에서 발생하는 오류 및 개선에 대한 정보를 상세히 기록하고 있기 때문에 소프트웨어 진화(Software Evolution) 또는 소프트웨어 마이닝(Software Mining) 분야에서 다양한 연구들이 진행되었다. 그러나 다양한 버그 추적 시스템들이 각각 서로 다른 형태로 버그 리포트를 관리하기 때문에 대부분의 경우, 한 가지 버그 추적 시스템만을 대상으로 연구를 진행하거나 데이터 추출 도구 구현이나 스키마 정의 등 중복 작업을 진행하였다. 또한 버그 리포트는 상태(status), 우선순위(priority), 심각도(severity) 등의 메타데이터(meta-data)를 포함하지만, 설명(description), 댓글(comment)과 같은 텍스트 기반의 비정형적인 데이터가 주요 분석 대상이므로, 이를 위한 정제 과정에 많은 노력이 든다. 이러한 문제를 해결하기 위해 비정형 데이터로 부터 패치(patch), 스택추적(stack trace), 소스코드(source code), 순서리스트(enumeration)와 같은 정형적 텍스트를 추출하는 연구도 있었다[1]. 그러나 설명 및 댓글들에는 여전히 비정형적 자연어가 존재하고 추출된 텍스트도 메타데이터처럼 단순 데이터가 아닌 복잡한 형태이거나 자연어로 구성되어 있기 때문에 연구에 활용하기 위해서는 부가적 노력이 필요하다.

따라서 본 논문에서는 다양한 버그 추적 시스템으로부터 추출한 데이터를 일차적으로 가공하여 각각의 시스템에서 제공하는 데이터 스키마에 맞춘 저장소 종속 모델(Repository Specific Model)을 생성한 후, 이를 다시 저장소 독립 모델(Repository Independent Model)로 통합하여 연구자에게 제공함으로서 MSR(Mining Software Repositories) 분야에 대한 연구 수행 시 단일 모델로 다양한 연구를 할 수 있도록 지원하는 UniBAS(Unified Bug Analysis System)를 제안한다. 연구자들은 이 시스템을 통하여 다음과 같은 도움을 얻을 수 있다.

- 다양한 버그 추적 시스템으로부터의 데이터 추출뿐 아니라 분석에 필요한 기초 정보 및 통계를 얻기 위한 가공 작업에 대한 노력과 비용을 줄일 수 있다.
- 비정형적인 텍스트 위주로 구성되어 있는 다형의 버그 추적 시스템으로부터 통일되고, 구조적인 스키마를 따르는 데이터를 효과적으로 확보할 수 있을 뿐 아니라 쉽게 확장이 가능한 환경을 만들 수 있다.
- 저장소 독립 모델을 대상으로 다양한 질의를 실시함으로써 보다 일관성 있고, 추상적인 수준에서 MSR연구에 집중할 수 있다.

본 논문의 이후 구성은 다음과 같다. 2장에서는 다양한 연구자들의 요구사항을 예측하고 통합 모델에 담아내기 위한 관련 연구를 제시하고, 3장에서는 다양한 버그 추적 시스템에 대한 개요를 설명한다. 4장에서는 저장소 종속 모델, 저장소 독립 모델 및 확장 모델에 대하여 설명한다. 5장에서는

1) Bugzilla : http://www.bugzilla.org/
2) Mantis : http://www.mantisbt.org/
3) Trac : http://trac.edgewall.org/

UniBAS를 활용한 사례 연구를 보이고, 마지막 6장에서는 향후 연구를 포함한 결론을 내린다.

II. 관련 연구 및 배경 지식

이 장에서는 버그 추적 시스템으로부터 추출한 데이터를 이용한 연구들을 살펴봄으로써 MSR 연구를 지원하기 위한 시스템, 즉 UniBAS가 갖추어야 할 여러 속성들에 대해 조사하였다.

1. 버그 리포트 분석에 관한 연구

버그 리포트 분석에 관한 연구에는 대표적으로 중복된 버그 리포트 탐지[2][3], 버그 발생위치 추적[4][5], 버그 리포트 자동 분류[6][7][8], 버그 리포트 품질 향상[9][10][11], 버그 수정시간 예측[12] 등의 연구가 있었다.

Runeson은 정보 검색 기법을 활용하여 중복 버그 리포트 탐지를 시도하였다[2]. 그는 비교를 위해서 프로젝트명과 요약 정보를 담고 있는 헤더(header) 및 상세 정보를 포함한 설명을 추출한 후 자연어 처리를 통해 VSM(Vector Space Model)으로 표현하였고, 코사인 유사도(Cosine Simality)를 통하여 중복 버그 리포트를 탐지하였다. 그 결과 약 60%의 중복을 식별 할 수 있었다. Nguyen은 같은 내용에 대해 다른 용어를 사용하여 설명한 리포트에 대해서 중복을 탐지할 수 없는 기존 연구의 한계를 극복하기 위해 정보 검색기법과 더불어 토픽(topic)기반 모델인 LDA(Latent Dirichlet allocation)를 통해 기존 연구 대비 20% 이상의 중복을 탐지하였다[3].

Zhou는 버그 리포트를 분석하여 VSM을 변형한 rVSM (revised Vector Space Model)을 활용하여 버그가 있는 파일들을 추천해주는 시스템을 제안하였다[4]. 시스템은 버그 리포트와 소스코드의 내용을 분석하여 검색모델을 만들고 새 버그 리포트를 분석하여 검색을 함으로써, 10개의 파일을 추천하는 경우 평균 70%의 재현율(Recall)을 보여 주었다. 이어 Kim은 기존의 연구들에 사용된 버그 리포트가 완전한 정보를 가지지 않음[11]에 착안하여 1차 분석을 통하여 정보가 불완전한 버그 리포트를 필터링 하는 2단계 추천 시스템을 제안하였다[5]. 버그 리포트를 분석하기 위해서 요약(summary), 설명, 댓글 및 버전(version), 플랫폼(platform), 운영체제(os), 우선순위, 심각도, 보고자(reporter) 등의 메타데이터를 기반으로 나이브 베이지안(naïve baysian) 알고리즘을 적용하였다. 그 결과 1단계 방식에 비해 10% 이상의 성능향상을 얻을 수 있었다.

Cubranic의 연구[6]에서는 버그 리포트의 텍스트를 분석하여 기계학습 알고리즘인 나이브 베이지안을 적용함으로써 어떤 개발자에게 버그 리포트를 할당해야 할지를 자동으로 결정하는 연구를 진행하였다. 실험 결과의 정확도는 30%정도에 불과했으나, 버그 리포트를 자동으로 선별하여 할당하는 것이 가능함을 보여주었다. 이어 Anvik은 기계학습의 SVM 알고리즘을 활용하여 버그 수정 담당자를 선정하는 방법을 제안하였다[7]. 적합한 개발자를 추천하기 위해 버그 리포트의 상태, 담당자(assignee) 및 버그를 해결한 사용자 정보를 추출하여 추천 알고리즘에 사용하였으며, VCS(Version Control System)에서 얻은 데이터로 실험을 한 결과 평균적으로 60%의 정확률(precision)을 얻을 수 있었다.

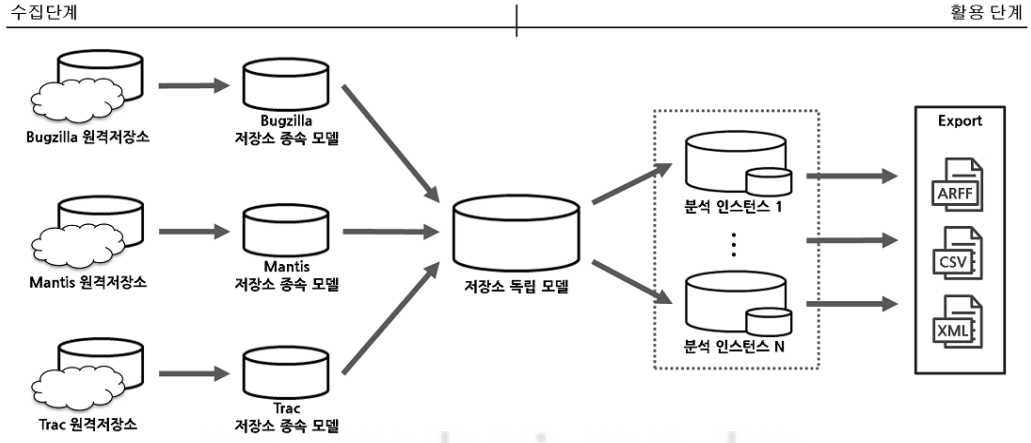

그림 1. Unified Bug Analysis System 개요
Fig. 1. The Overview of Unified Bug Analysis System

Zimmermann의 연구[11]에서는 개발자와 버그 리포트 작성자들을 대상으로 설문을 하여 그들의 생각 차이를 정리하였고, 버그 재현 과정(step of reproduce), 스택 추적, 테스트 케이스(test case), 순서리스트, 스크린샷(screenshots) 등 버그 수정에 더 도움이 되는 정보들을 식별하였다. 따라서 이 정보들을 효과적으로 입력할 수 있도록 버그 리포트 작성자들을 유도하는 방식의 시스템을 제안하였다[9][10].

Weiss는 기존에 수정된 버그 리포트의 상태 변화와 패치의 업로드 시간을 이용하여 버그 리포트를 수정하는데 걸린 시간데이터들을 확보하고, 텍스트 기반 데이터를 이용하여 버그 리포트의 유사성을 비교함으로써 버그 리포트의 결함 수정 시간을 예측하는 방법을 제안하였다[12]. 그는 각각의 버그 리포트를 비교하기 위해 kNN의 기계학습 알고리즘을 사용하였으며, 나이브 베이지안 알고리즘을 사용한 다른 연구보다 우월한 결과를 얻었고 결과는 실제로 필요한 시간과 거의 유사했다.

2. 버그 추적 시스템 통합 모델

Tran은 전체 텍스트(full-text)지원이라는 특정한 목적을 위해 버그 추적 시스템간의 통합을 시도하였다[13]. 그는 버그 리포트에서 버그 수정 정보를 쉽게 검색하기 위해 다양한 버그 추적 시스템 데이터를 통합 모델에 저장하고, 전체 텍스트 검색과 메타 데이터 검색을 결합한 검색 시스템을 제안하여 검색 성능을 개선하였다. 이는 최초로 버그 추적 시스템 통합을 제안한 논문으로 본 연구와 유사성이 있으나, 검색을 위한 단순한 정보만을 포함하고 있기 때문에, 본 논문의 연구 방향인 MSR 분야의 다양한 연구 시나리오를 지원하기에 부족한 점이 있다.

관련연구를 통하여 각 연구에서 사용한 특성(feature)들을 살펴보면 메타데이터 보다는 텍스트 데이터를 주로 이용하였으며 기본적으로 단어 주머니(bag of word) 생성 등의 자연어 처리 기법을 활용하고 있다. 본 논문에서 제시하는 통합 모델에서도 관련 연구를 지원하기 위해 단어 주머니에 대한 기본 데이터 생성을 포함한다.

III. 통합 버그분석 시스템(UniBAS) 개요

본 논문은 연구자들의 다양한 연구 수행을 위하여 여러 버그 추적 시스템을 일관된 방법으로 접근, 분석할 수 있는 UniBAS (Unified Bug Analysis System)를 제안한다. 제안한 시스템은 크게 수집단계와 활용단계로 나뉜다. 수집단계에서는 다양한 버그 추적 시스템이 제공하는 데이터를 각각의 시스템 모델에 맞춘 저장소 종속 모델로 확보한 후, 일관된 분석 방법을 제공할 하나의 저장소 독립 모델로 데이터를 변환한다. 활용단계에서는 연구자들이 수집된 데이터를 필터링하여 생성한 분석 인스턴스를 통해 다양한 연구를 수행하도록 지원한다. 수집된 데이터는 저장소 독립 모델 뿐 아니라 저장소 종속 모델도 접근이 가능한데, 이는 시스템 종속적인 데이터를 이용한 연구도 가능하도록 한다. 이로써 연구자들은 데이터 추출이나 모델링과 같은 공통적이고 반복적인 작업에 신경 쓰지 않고 상위 수준에서 다양한 질의를 통해 다형의 버그 추적 시스템으로부터 확보한 버그 리포트와 추가 정보들을 분석할 수 있다. 그림 1은 UniBAS의 주요 데이터 흐름을 보여준다.

1. 수집단계

데이터 수집을 위한 사이트가 지정되면 UniBAS는 해당 사이트에 맞는 웹 봇(web bot)을 이용하여 수집을 시작한다. 웹 봇은 다형의 원격 저장소에 맞는 버그 리포트 획득 알고리즘을 갖도록 별도 설계되었으나, 1) 로그인과 환경설정 2) 속성값 추출 3) 버그 리포트 수집 순으로 공통된 흐름을 가진다.

대부분의 오픈소스 프로젝트는 버그 리포트가 공개되어 있지만, 로그인을 하지 않는 경우 일부 식별자의 제한적 제공, 사용자에 따른 언어 불일치, 타임존의 차이로 인한 시간데이터의 불일치가 발생하여 올바른 추출이 어렵다. 따라서 로그인 후 고정된 환경설정을 이용하도록 한다.

속성 값 추출은 버그 리포트에 존재하는 제품(product), 우선순위, 심각도, 상태, 운영체제 등 메타데이터에서 사용가능한 값들의 추출을 말한다. 각 버그 추적 시스템은 별도의 페이지를 통해 이 값들을 제공하여 순서정보, 부가설명들을 추가로 얻을 수 있다. 우선순위나 심각도의 순서정보는 활용단계에서 속성값 매핑에 활용해야 하므로 중요하다.

다음으로, 사이트에 존재하는 모든 버그 리포트를 수집한다. 버그 리포트는 최초의 문서부터 차례로 수집을 하며, 각 버그 추적 시스템에 맞는 분석기에 의해 필요한 데이터를 추출하여 저장소 종속모델에 저장한다. 분석기는 메타 데이터 뿐만 아니라 댓글, 첨부 파일들도 추출하는데, 파일과 같이 추가적인 분석이 필요한 데이터는 웹 봇에 요청하여 다운로드를 받는다.

버그 리포트를 수집하기 위한 프로토콜에는 HTML, RPC, SOAP 등이 있으나 RPC, SOAP은 시스템 관리자가 중단할 수 있는 선택적 기능[13]이므로 기본적으로 HTML을 사용하였다. 이 과정에서 웹에 드러나지 않는 내부 레코드

식별자, 버그 리포트의 상태전이도(status-workflow), 관리 데이터 등 원격지 저장소의 일부 데이터가 손실될 수 있다. 식별자의 경우 버그 리포트 식별자 등 중요한 값들은 추출 가능하므로 임의로 새로운 식별자를 할당해서 사용하고, 상태 전이도나 관리 데이터의 경우 분석에 큰 영향을 주지 않으므로 권한으로 인한 미공개 버그 리포트와 함께 수집 대상에서 제외하였다.

이 과정을 마친 후에 저장소 종속 모델에 확보된 레코드를 가공하여 여러 시스템을 하나의 방식으로 다룰 수 있도록 통합한 저장소 독립 모델 인스턴스로 복제하였다. 작업 완료 후에는 필요에 따라 업데이트를 통해 변경된 정보들을 새로 수집하여 추출할 수 있기 때문에, 원본 데이터의 유지 및 원격지의 잦은 접근 방지를 위해 로컬에 웹페이지 캐시를 생성하였다.

2. 활용 단계

수집된 데이터들을 활용하여 원본 데이터를 보존하고 빠른 연구 수행을 위하여 분석 인스턴스를 생성하였다. 분석 인스턴스는 기본적으로 속성값 매핑, 확장 모델 생성 및 처리를 한다. 연구자는 분석 인스턴스 생성시 데이터를 선택할 조건들의 지정과 함께 각 기능의 수행 여부도 선택을 할 수 있다.

위와 같은 기능들을 지원하기 위해 UniBAS에는 관리모델을 가진다. 이 모델은 데이터 수집을 위한 사이트의 관리, 분석 인스턴스 관리를 책임진다. 그림 2는 UniBAS의 데이터들을 관리하기 위한 모델을 간략히 보여주고 있다. 사이트 테이블(site)은 수집단계의 진행상황을 포함한 사이트의 정보들을 저장하고 프로젝트요약 테이블(project_summary)과 더불어 각 사이트에서 수집된 데이터의 요약정보들을 가짐으로써 사이트 관리를 수행한다. 분석 인스턴스의 관리는 데이터 선택에 사용한 필터링 조건 및 프로젝트 정보들을 저장하는 분석 테이블(analysis)과 선택한 프로젝트에 대한 정보를 저장하는 분석프로젝트 테이블(analysis_project), 분석 인스턴스의 상태 변화를 기록하기 위한 상태기록 테이블(analysis_history)을 이용한다. 또한 사용자가 분석시에 자주 사용하는 질의나 UniBAS에서 기본적으로 제공할 질의에 대한 관리를 위해 질의 정보(analysis_query)보 보관할 수 있도록 한다.

추가적으로 분석 인스턴스 뿐 아니라 UniBAS내의 모든 모델로부터 맞춤형 질의를 통해 데이터를 살펴볼 수 있고, 타블로(tablau)와 같은 시각화 도구 및 웨카(weka)같은 마이닝 도구와의 연동을 위하여 arff, csv, xml 등 다양한 형식으로 데이터 내보내기를 할 수 있다.

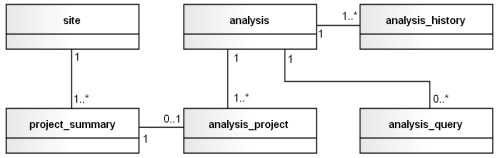

그림 2. 데이터 관리 모델
Fig. 2. The Data Management Model

IV. UniBAS 주요 저장소 모델

이 장에서는 개별 원격 저장소에 최적화된 저장소 종속 모델과 다형의 종속 모델을 비교, 분석하여 통합한 저장소 독립 모델 및 분석 기능을 추가하기 위한 확장 모델에 대하여 자세히 살펴본다. 각각의 저장소 종속 모델은 오픈소스 프로젝트에서 많이 사용하는 Bugzilla, Mantis 및 Trac을 선정하였다[13].

1. 저장소 종속 모델

다형의 버그 추적 시스템은 각 시스템을 위한 데이터 스키마(schema)를 공개[4][5][6]하고 있다. 이 스키마에는 시스템 관리를 위한 내부 데이터와, 웹에는 공개되지 않는 데이터들이 함께 존재한다. 우리는 저장소 종속 모델을 정의하기 위해, 이런 데이터들을 제외하고 분석에 활용 가능한 정보들을 식별하였다. 그림 3, 그림 4, 그림 5는 각각 선정한 시스템들의 저장소 종속 모델들을 간략히 보여준다. 각 버그 추적 시스템은 개별적인 방식으로 시스템을 구축하고 서비스를 제공하지만 버그 추적이라는 동일한 목표 때문에 실질적으로는 유사한 정보와 기능을 제공한다. 즉, 일반적으로 버그 추적 시스템은 사용자(보고자, 개발자, 관리자 등)가 프로젝트 별로 버그 리포트를 관리할 수 있도록 지원하는 시스템으로 각 시스템의 데이터 모델도 프로젝트, 사용자, 버그 리포트 세 부분으로 그룹화 된다.

프로젝트 그룹은 사이트 내 수행중인 프로젝트 구분 정보를 저장한다. Bugzilla는 범주(classfications), 제품(products), 컴포넌트(components) 테이블을 계층적으로 이용해 프로젝트를 구분한다. 이 중 범주 테이블은 선택적 분류이므로 제품이 중심 분류 역할을 한다. Mantis의 경우, 프

4) Bugzilla 스키마 : http://www.ravenbrook.com/tool/bu gzilla-schema/
5) Mantis 스키마 : http://www.mantisbt.org/docs/master /en-US/Developers_Guide/html-single/
6) Trac 스키마 : http://trac.edgewall.org/wiki/TracDev/ DatabaseSchema

로젝트(project), 카테고리(category) 테이블을 이용하며, 프로젝트는 관계를 통한 계층구조를 표현하고, 카테고리는 프로젝트의 세부 분류 정보를 저장한다. Trac은 단일 프로젝트를 위한 시스템으로 컴포넌트(component) 테이블 만을 이용해 구분한다.

사용자 그룹의 경우, 원격 저장소에서 다양한 정보를 가질 수 있으나 실제 추출 가능한 정보가 부족하여 모든 저장소 종속 모델에서 1개의 사용자 테이블(profiles, user)과 사용자 식별을 위한 로그인명 필드(login_name, username)로 정의하였다. Mantis의 경우 접근 수준 필드(access_level)가 존재하며 원격 저장소 접근 수준이 보고자(reporter), 개발자(developer) 등으로 표현되기 때문에 사용자의 역할을 유추하는데 유용하다. 각 사용자는 버그 리포트에 대해 보고자, 담당자(assignee, handler, owner), 검토자(qa), 관심사용자(cc)의 역할을 가지고 있어서 각 역할에 대해 버그 테이블(bugs, bug, ticket)과의 관계를 정의하였다.

버그 리포트는 보고자가 제출하는 문서로써 발생한 버그에 대한 실행환경, 상세설명, 키워드 및 첨부파일까지 포함한다. 사용자는 발생한 버그에 대한 토의 및 투표, 일정 관리 등을 실시할 수 있다. 각 시스템은 이런 정보들을 추적하기 위해 여러 테이블을 이용하게 되는데, 대표적으로 메타데이터를 저장하는 버그 테이블, 토의 정보를 저장하는 댓글 테이블(longdescs, bugnote), 첨부파일을 위한 첨부파일 테이블(attachements, bug_file, attachement) 등이 있다. 또한 버그 리포트간의 중복, 종속, 참조 관계 표현을 위한 관계 테이블(duplicates, dependencies, relationship)도 존재한다.

각 시스템은 대부분 유사한 스키마를 가지나, 일부는 차이를 보인다. 가장 많은 정보 필드를 가진 Bugzilla는 Mantis가 가진 재현가능성(reproducibility), 수정된 버전(fixed_in_version) 필드가 없다. 반대로, Mantis는 버그 리포트에 대한 재개확인(everconfirmed), 버그 파일 위치(bug_file_loc) 필드가 없다. 그 외 Bugzilla에는 투표(votes), Mantis에는 예상 수정 비용(projection) 등 각 시스템별 독자적인 기능들을 반영한 필드들이 있다. 공통적인 필드 중에서는 Mantis가 운영체제 정보를 두 필드(os, os_build)로 구분하는 차이점을 가진다. 단순한 구조를 가지는 Trac 스키마는 티켓변경 테이블(ticket_change)이 변경 이력, 댓글을 포함하도록 복합적으로 구성되어 있기 때문에, 테이블 수에 비해 많은 정보들을 담을 수 있다. 또한 댓글 내에는 버그 리포트 간 관계 정보를 포함하고 있어 분석을 통해 추출가능하며, 메타데이터에서는 운영체제와 플랫폼 정보가 제외 되었다.

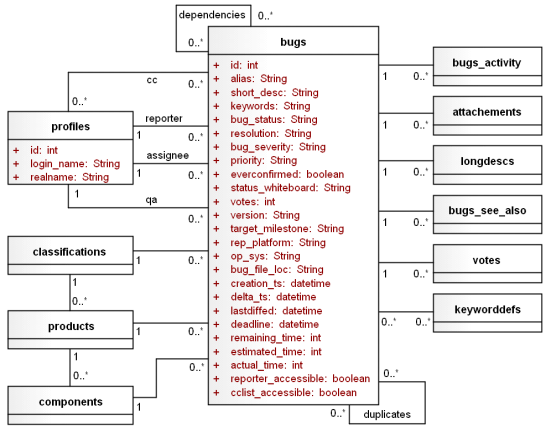

그림 3. Bugzilla의 저장소 종속 모델
Fig. 3. The Repository Specific Model of Bugzilla

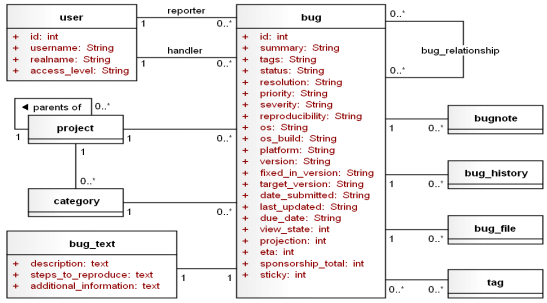

그림 4. Mantis의 저장소 종속 모델
Fig. 4. The Repository Specific Model of Mantis

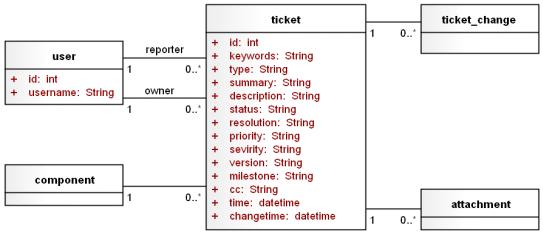

그림 5. Trac의 저장소 종속 모델
Fig. 5. The Repository Specific Model of Trac

2. 저장소 독립 모델

앞서 정의한 다형 저장소 종속 모델을 비교, 분석함으로써 이번 장의 저장소 독립 모델을 구축하였다. 저장소 독립 모델은 다형의 데이터를 일관된 방법으로 다룰 수 있도록 테이블 매핑, 필드 매핑 및 속성 값 매핑을 통한 변환을 거치게 된다. 또한 저장소 독립 모델은 연구자의 효과적 데이터 활용과 유용성 확보를 위해 추가적으로 구조화된 정보를 추출하여 저장한다.

그림 6은 저장소 독립 모델을 UML로 표현한 것으로, 다형의 데이터들을 관리하고 모델간의 정보격차를 줄이기 위해 필요한 테이블을 추가하고 부가적인 작업을 수행하였다. 사이트 테이블(site)은 다형의 저장소 종속 모델의 데이터를 구분하기 위한 것으로, 저장소 종속 모델의 각 테이블은 사이트 식별자를 필드로 추가하여 데이터 출처를 구분하였다. 목록맵 테이블(enum_map)은 사이트 내의 속성값 목록과 매핑정보를 저장하기 때문에 속성값 매핑 정보를 필요에 따라 쉽게 변경할 수 있다. 사용자 테이블(user)은 사용자의 이름 외에 역할(role), 활동기간(creation_ts, change_ts)을 포함하는데, 이들은 저장소 종속 모델에서 추출되지 않는 정보로서 각 사용자의 활동을 분석하여 정보를 저장한다[6]. 구조화된 정보(structured_info)테이블은 정형적인 텍스트의 분리를 위하여 따라 비정형 텍스트로부터 소스코드 등을 추출한다[1].

저장소 종속 모델의 데이터는 테이블 간의 매핑정보나 필드간의 매핑정보를 이용하여 변환을 수행한다. 이후 일관성을 높이기 위해 속성 값에 대한 매핑도 수행하는데 이는 분석 인스턴스 생성시 수행하므로 별도로 설명한다. 표 1은 각 모델간의 테이블 매핑관계를 보여주며, 표 2는 버그 리포트에 속한 대부분의 메타데이터를 가지는 버그(bug)테이블에 대한 필드명의 매핑을 보여주고 있다.

테이블 매핑은 대부분 1:1 매핑을 통하여 정보의 변환이 가능하지만, 테이블을 간소화한 Trac의 경우 많은 분석 작업이 수반된다. 그러나 댓글과, 변경기록은 잘 구분되어 있고, 관계정보는 댓글 내에 구조화 되어있기 때문에 큰 비용이 들지 않는다. 키워드와 모니터(monitor)테이블에 대해서는 별도 테이블이 없이 티켓 테이블에 필드만 두고 있어서 이들을 분리하여 저장한다. 또한 존재하지 않는 프로젝트에 대한 데이터는 사이트 정보를 복사하여 사용한다.

Mantis는 관심 사용자 정보를 내부적으로 사용하기 때문에 집적 정보 추출을 할 수 없다. 대신 사용자의 버그 리포트 등록, 댓글 등록, 정보 수정 같은 행동을 추적하여 정보를 가공한다. Bugzilla는 관계에 대한 데이터를 여러 테이블에 분산 저장했지만 관계 정보를 살펴보기 위해 여러 테이블을 확인해야 하는 문제 때문에 하나로 합쳤다. 그리고 제품군은 Bugzilla에서만 사용하므로 저장소 독립 모델에서는 제외하였다.

필드 매핑은 각 테이블 내의 정보들을 의미에 맞게 변환하도록 해준다. 일부 필드는 다른 테이블에서 정보를 가져오거나 부가적인 작업을 수반하는 경우도 있다. Bugzilla와 Mantis의 경우 버그 리포트에 대한 상세설명(description, longdescs)을 다른 테이블에 분리하여 저장하고 있어서 별도로 가져와야 한다. 재개확인(everconfirmed) 필드는 Bugzilla에만 존재하는 필드로 해당 버그 리포트의 수정 후 열림 여부를 나타낸다. 다른 시스템들은 변경기록 추적을 통해 정보를 생성하며, 유추 불가능한 필드는 NULL값으로 데이터의 부재를 표현한다.

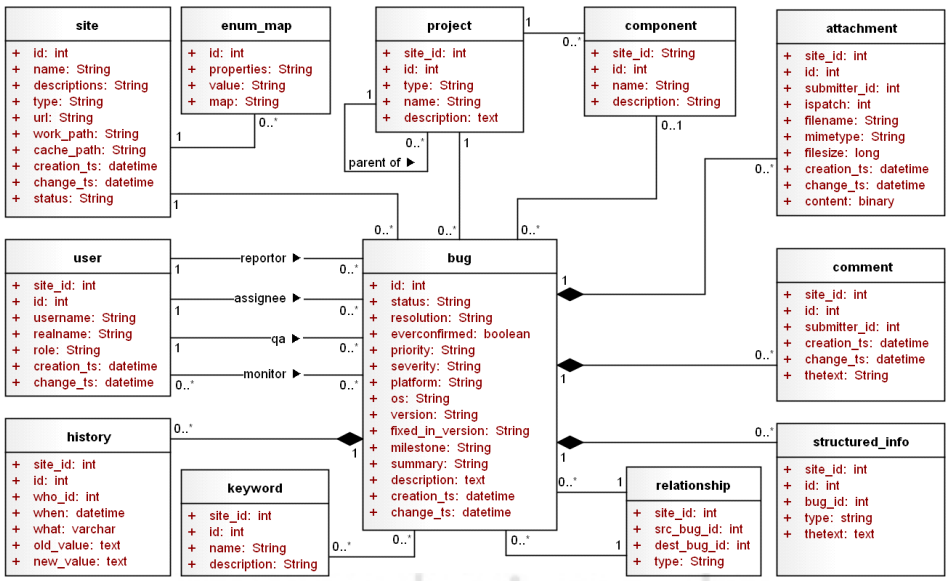

그림 6. UML로 표현한 저장소 독립 모델 스키마
Fig. 6. Repository Independent Model represented as a UML Diagram

표 1. 모델별 테이블간의 매핑
Table 1. Table Mapping for Each Model

저장소 독립모델	Bugzilla	Mantis	Trac
-	classifications	-	-
project	products	project	-
component	components	category	component
user	profiles	user	user
bug	bugs	bug	ticket
attachement	attachments	bug_file	attachment
comment	longdescs	bugnote	ticket_change
history	bugs_activity	bug_history	ticket_change
keyword	keyworddefs	tag	ticket (keywords)
relationship	dependencies, duplicates, bugs_see_also	relationship	ticket_change
monitor	cc	-	ticket (cc)

표 2. 버그테이블의 필드 매핑
Table 2. Field Mapping for 'bug' Table

저장소 독립모델	Bugzilla	Mantis	Trac
id	id	id	id
project_id	product_id	project_id	-
component_id	component_id	category_id	component
summary	short_desc	summary	summary
description	〈longdescs〉	〈bug_text (description)〉	description
reporter	reporter	reporter	reporter
assignee	assignee	handler	owner
qa	qa	-	-
status	bug_status	status	status
resolution	resolution	resolution	resolution
severity	bug_severity	severity	severity
priority	priority	priority	priority
everconfirmed	everconfirmed	〈bug_history〉	〈ticket_change〉
os	op_sys	os, os_build	-
platform	rep_patform	platform	-
version	version	version	version
fixed_in_version	_	fixed_in_version	_
milestone	target_milestone	target_version	milestone
creation_ts	creation_ts	date_submitted	time
change_ts	delta_ts	last_updated	changetime

3. 속성 값 매핑

속성 값 매핑은 데이터를 일관된 방식으로 이해할 수 있게 해준다. 매핑 할 속성에는 상태, 해결방법, 우선순위, 심각도 필드가 해당하는데 이들은 버그 리포트 처리현황이나 등급 표현에 유용하게 활용된다. 단, 같은 명칭이지만 다른 값을 나타내거나 같은 의미이지만 다른 명칭으로 표현되는 등의 문제가 있어 일관된 기준이 필요하다. 이에 대해 Tran의 연구[13]나 Capraro의 연구[14]에서도 필드 값의 통합을 시도했으나 특정 목적을 위한 기준으로, 여러 목적으로 이용하기에는 적합하지 않았다. 따라서, 본 논문에서는 속성 값에 대한 기준을 제시하고 사용자의 요구에 따라 변경이 가능하도록 하였다. 사용자는 기준에 대한 변경이나 매핑에 대한 구성을 변경할 수 있는데, 표 3~표 6은 4가지 필드에 대한 매핑 테이블을 보여준다.

제안하는 매핑 값은 각 시스템 본연의 값을 유지하도록 동적 변경이 가능하다. 먼저 상태필드는 버그 리포트 상태의 흐름을 표현하기 위해 6가지 상태를 가지도록 하였으며, 각 상태는 보고자, 개발자, 검토자, 관리자가 해당 역할을 수행하는 흐름과 일치한다. 해결 방법은 상태 필드가 RESOLVED 이후의 값을 가지는 경우, 추가 현황을 나타내기 위해 6가지 값을 가지며, NOTHERE 값은 다른 프로젝트에 속한 버그를 표시한다.

우선순위와 심각도 필드는 기존 시스템의 구분에 따라 각각 6가지, 7가지로 정의하였다. 높은 등급을 가지는 값이 상위에 있고 낮은 등급을 가지는 값이 맨 하위에 있다. 심각도의 경우 버그는 아니지만 기능개선에 해당하는 기능강화(ENHANCEMENT)라는 값을 가진다.

표 3. 상태(status)에 대한 매핑
Table 3. Naive Mapping of 'status'

저장소 독립모델	Bugzilla	Mantis	Trac
UNCONFIRMED	unconfirmed, needinfo	new, feedback	new, pending
CONFIRMED	new, confirmed, reopened	confirmed, acknowledged	accepted, reopened
ASSIGNED	accepted, assigned, in_progress	assigned	assigned
RESOLVED	resolved	resolved	-
VERIFIED	verified	-	-
CLOSED	closed	closed	closed

표 4. 해결방법(resolution)에 대한 매핑
Table 4. Naive Mapping of 'resolution'

저장소 독립모델	Bugzilla	Mantis	Trac
FIXED	fixed	fixed	fixed
INVALID	invalid	no change required	invalid
WONTFIX	wontfix, expired, later, obsolete, unmaintained	won't fix, suspended	wontfix
DUPLICATE	duplicate	duplicate	duplicate
NEEDINFO	worksforme, incomplete, irreproducible, waitingforinfo, support, remind, needinfo, backtrace	not fixable, reopened, open, unable to reproduce	worksforme, cantfix, needsinfo
NOTHERE	moved, not_eclipse, downstream, upstream		

표 5. 우선순위(priority)에 대한 매핑
Table 5. Naive Mapping of 'priority'

저장소 독립 모델	Bugzilla	Mantis	Trac
P1	P1, VHI	immediate	hightest, blocker, critical
P2	P2, HI	urgent	high, important, critical
P3	P3, NOR	high	normal, major
P4	P4, LO	normal	low, minor
P5	P5, VLO	low	lowest, wish, trival
NONE	—	none	undecided

표 6. 심각도(severity)에 대한 매핑
Table 6. Naive Mapping of 'severity'

저장소 독립 모델	Bugzilla	Mantis	Trac
BLOCKER	blocker	block	blocker, release broker
CRITICAL	critical, regression	crash	critical, regression
MAJOR	major	major	major, problem
NORMAL	normal	minor	normal, optimization
MINOR	minor	tweak, text	minor, cosmetic
TRIVIAL	trivial	trivial	trivial, not applicable, unimportant
ENHANCEMENT	enchancement, wishlist, task	feature	

4. 확장 모델

본 논문의 목적인 다양한 연구 활동 지원을 위해 저장소 독립 모델에 필요한 확장 모델들을 추가하였다. 본 논문에서는 자연어 처리에 기반이 되는 단어주머니 생성 관련 확장 모델을 제시하였으며, 연구 주제에 맞추어 다양한 모델을 쉽게 확장하여 사용할 수 있다.

버그 리포트에서 제목, 상세설명, 댓글 및 구조화된 정보는 자연어로 이루어진 필드로써 버그 리포트의 핵심정보들을 담고 있다. 이 과정에서 주로 TF-IDF를 계산하지만, 모든 연구에 일관된 방법으로 TF-IDF를 계산하기에는 대상에 따른 최적화 문제가 있다[15]. 따라서 우리는 다양한 알고리즘을 적용하여 비교 및 분석에 용이하도록 단어주머니 모델을 정의하였다. 그림 7은 단어주머니를 위한 주요 모델 및 관계를 UML로 보여준다.

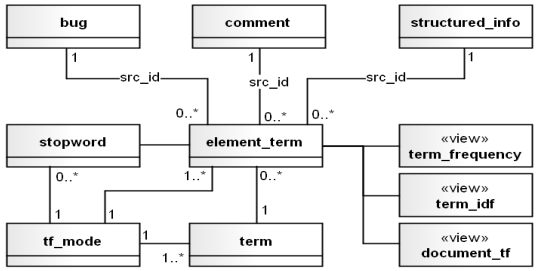

그림 7. 단어주머니에 대한 확장모델
Fig. 7. The Extended Model for 'bag of words'

단어주머니 모델은 버그, 댓글, 구조화된 정보 테이블을 요소(element)로 두고, 이 요소들 내에 존재하는 용어(term)들의 관계를 나타내는 용어관계(element_term)테이블로 구성된다. 용어관계 테이블은 각 요소 내 존재하는 어휘들의 토큰 정보를 순서 정보와 함께 저장한다. 저장된 데이터는 계산에 유용하도록 단위 정보인 용어 빈도수(term_ frequency), 문서 내 용어수(document_tf), 용어 IDF(term_idf)와 같은 뷰와 프로시저를 함께 정의하고 있다.

UniBAS는 토큰화 및 어근추출을 위해 파이썬(Python) 기반의 NLTK 라이브러리[7]를 이용하고, 불용어 처리를 위해 별도의 stopword 테이블을 사용한다. 우선, UniBAS는 검증된 NLTK를 활용하여 PunktWord, TreebankWord, WordPunkt, WhiteSpace 등의 토큰화 알고리즘과 Porter, Lancaster, Snowball 등의 어근추출 알고리즘을

7) 자연어 처리 알고리즘 : http://text-processing.com/

선택할 수 있으며, 정규식을 이용한 사용자 정의 패턴을 이용하여 토큰화 및 어근추출도 가능하다. 단어주머니 모델에서는 다양한 NLTK 알고리즘에 따른 자연어 처리 결과를 비교하고 관리할 수 있도록 TF모드(tf_mode)테이블을 사용한다.

V. 사례 연구

본 장에서는 UniBAS의 유용성을 검증하기 위해, 자연어 처리 기법을 통해 중복 버그 리포트를 식별하는 Runeson의 연구[2]를 진행하였다. UniBAS가 확보한 저장소 독립 모델에 질의를 통하여 적절한 대상 데이터를 선정하고 Runeson의 접근법에 따라 연구를 수행한 결과를 비교해 보았다.

표 7. UniBAS를 통해 수집한 버그 리포트
Table 7. The Collected Bug Reports using UniBAS

사이트	종류	프로젝트 수	버그 수	수집기간
Mozilla	Bugzilla	49	320,203	1994/09 ~ 2005/12
Phplist	Mantis	11	2,379	2003/01 ~ 2014/06
Scribus	Mantis	3	12,034	2003/11 ~ 2014/06
Trac	Trac	1	10,803	2003/08 ~ 2014/07
dJango	Trac	1	22,533	2005/07 ~ 2014/07

실험 데이터 선정을 위해 주요 사이트로부터 수집한 데이터들을 표 7에 정리하였다. 이 중, 데이터의 다양성면에서 효과적이라고 판단한 Mozilla 사이트를 대상으로 사례 연구를 진행하였다. Mozilla 사이트는 현재 107개의 프로젝트를 운영중이며, 수집한 데이터는 49개의 프로젝트, 320,203개의 버그 리포트이다. 기존에 중복으로 식별된 버그 리포트 정보들을 이용하여 각 프로젝트별 중복 버그 리포트의 현황을 알아보았다.

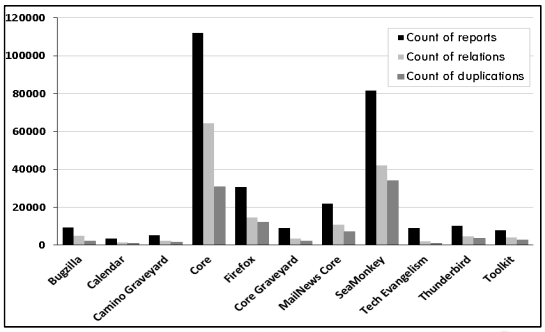

그림 8. 프로젝트별 버그 리포트 수
Fig. 8. The Count of Bug Reports for Each Project

그림 8에 나타난 프로젝트는 버그 리포트가 3000개 이상 제출된 프로젝트에 대해 중복 버그 리포트의 수와 모든 관계에 대한 버그 리포트의 수를 보여준다. 제출된 버그 리포트 중 평균 45%가 다른 버그 리포트와 관계를 가지고 있고, 그 중 평균 60%가 중복 버그 리포트인 것으로 나타났다. 특히 Firefox 프로젝트의 중복 버그 리포트는 80%이상으로 높은 수치를 보여서 연구 대상 프로젝트로 선정하였다.

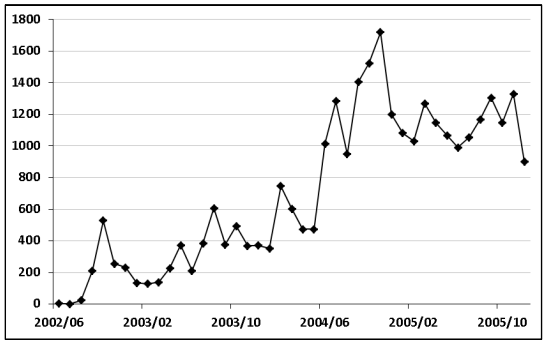

그림 9. Firefox 프로젝트에 대한 월별 버그 리포트 제출 수
Fig. 9. The Monthly Count of Bug Reports in Firefox

Firefox 프로젝트는 2002년부터 시작했으며 월별 버그 리포트의 제출량의 변화를 살펴보면 그림 9와 같이 나타났다. 각 월별 버그 리포트의 량은 증감을 반복하였는데 2004년부터 급격하게 증가하다가 2004년 말부터 안정적인 추세로 바뀌는 것을 볼 수 있다. 최종적으로는 2004년도에 제출된 버그 리포트를 대상으로 중복 버그 리포트 식별을 수행하였으며, 해당 기간 동안의 전체 버그 리포트는 11,742개, 중복 버그 리포트는 4,691개였다.

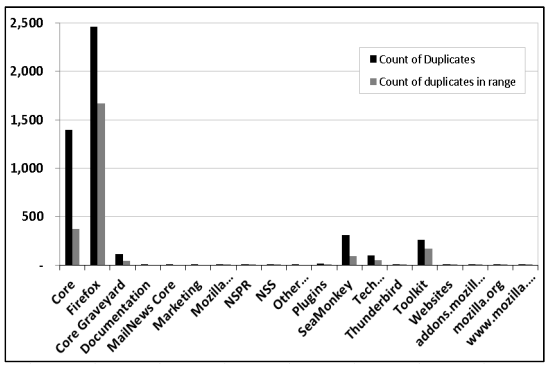

그림 10. 타 프로젝트와의 중복 버그 리포트 수 (Firefox 프로젝트 내)
Fig. 10. The Count of Duplicate Bug Reports in Firefox

그림 10은 중복 버그 리포트들의 관계들을 보여주고 있다. 중복 버그 리포트는 같은 프로젝트 뿐 아니라 다른 프로젝트와도 중복이 발생한다. 따라서 선택한 영역 내에서 발생한 중복 버그 리포트와 같은 프로젝트에서 발생한 중복 버그 리포트를 필터링 하면 확인할 수 있는 중복 버그 리포트는 1,668개이다.

중복 버그 리포트 탐지를 위하여 우리는 버그 리포트의 제목과 상세설명을 이용하였고, 자연어 처리는 UniBAS에서 선택할 수 있는 다양한 알고리즘을 적용하였다. 처리된 결과에 대해서 대명사, 관사, 관계사, 구두점 등의 불용어는 제거하고, Luhn의 가설[16]에 따라 어휘들의 IDF값을 이용하여 고빈도, 저빈도 어휘들도 추가로 제거하였다.

유사도 비교는 각 문서가 포함한 용어를 VSM으로 표현하고 코사인 유사도를 통하여 문서간의 비교를 하였다. Runeson은 비교 비용을 줄이기 위하여 중복 버그 리포트가 발생하는 시간 차이를 조사하였다. 그 결과 20일 이내에 90%의 중복이 발생하는 것으로 나타났으나 우리의 데이터에서는 200일이상의 차이에서도 70%를 넘지 못하여 전체를 대상으로 비교하였다.

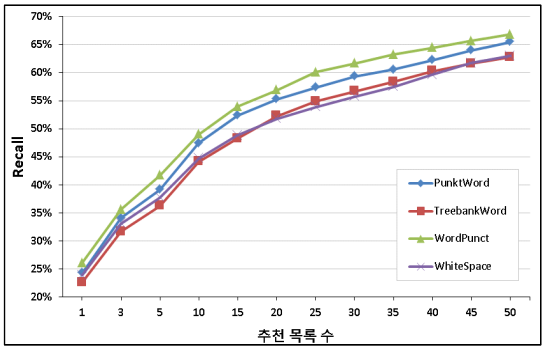

그림 11. 토큰화 알고리즘별 추천목록 수에 따른 재현율
Fig. 11. The Recall Rates of Tokenization Algorithms

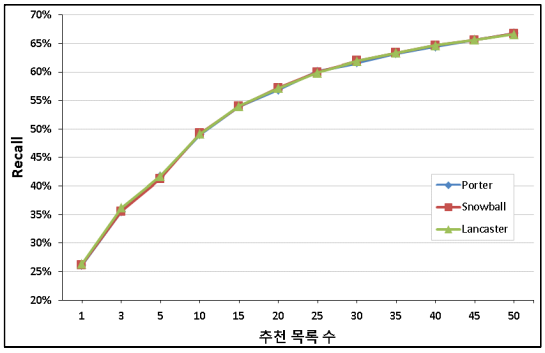

그림 12. 어근추출 알고리즘별 추천목록 수에 따른 재현율
Fig. 12. The Recall Rates of Stemming Algorithms

전체 문서의 비교 결과 가장 유사도가 높은 N개 문서를 추천하였고, 평가는 Runeson의 측정대로 기존에 식별된 중복 버그 리포트를 얼마나 많이 찾는지에 대한 재현율로 측정하였다.

그림 11은 토큰화 알고리즘별로 추천 수에 따른 재현율의 변화를 나타내고 있다. 전반적으로 WordPunct알고리즘이 재현율이 높게 나타났다. 그림 12는 어근추출 알고리즘에 대한 재현율을 나타내며 각 알고리즘 별로 큰 차이는 나타나지 않았으나 대체로 Snowball 알고리즘이 재현율이 높았다.

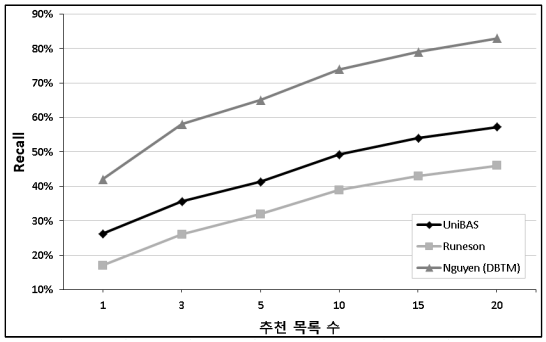

그림 13. 기존 연구와 재현율 비교
Fig. 13. Comparison of Recall Rates with the Previous Approaches

그림 13은 앞서 실험한 결과에서 높은 재현율을 보이는 알고리즘의 조합을 이용한 방법과 기존 연구들의 결과를 비교한 그래프이다. UniBAS의 결과가 Runeson의 결과보다는 10% 이상의 재현율을 보이고 Nguyen의 결과[3] 보다는 25%정도 낮은 결과를 보이고 있다. 각 연구가 다른 프로젝트를 대상으로 수행한 영향도 있어 단순히 그래프로만 비교하기는 어려운 점이 있어 상세 데이터를 살펴보았다. 그 결과 Runeson의 연구와 같이 UniBAS는 같은 내용이지만 다른 어휘를 사용하여 작성된 버그리포트를 탐지하지 못하는 문제점을 가지고 있었고, 이는 정보검색 기법을 통한 연구의 한계이므로 두 연구결과는 유사한 것으로 평가된다. 우리의 결과가 높게 나타난 것은 Firefox 프로젝트에는 같은 어휘를 사용하여 작성된 중복 버그리포트가 많기 때문이었다. 반면 Nguyen의 연구는 주제모델링(Topic Modeling)기법을 사용하여 위 문제점을 해결하여 높은 재현율을 보였다.

앞서 저장소 독립모델을 통하여 분석 대상 데이터를 살펴보고 선택한 후 분석 인스턴스를 생성하여 중복 버그 리포트를 찾는 연구를 수행하였다. 그 결과 데이터베이스에 저장된 데이터를 몇몇 질의를 통하여 다양하게 분석해 볼 수 있었고, 중복 버그 리포트를 추출하는데 있어 다양한 알고리즘을 적용

하여 비교해보기가 용이함을 알 수 있다. 다만 유사도 계산 등 대용량의 계산을 DBMS에 직접 하는 경우 불필요한 테이블 계산이 추가되어 계산 속도를 현저히 저하시켰다. 이 문제는 데이터베이스의 데이터를 부분 가공하여 내보내기한 후 계산을 수행하는 것으로 해결 하였다. 따라서 본 논문에서 제안한 시스템은 다양한 연구자들에게 기초적인 데이터를 제공하고 관리하는데 있어 편리성과 유연성을 제공함을 보여주었다.

VI. 결 론

버그 추적 시스템은 안정적인 고품질 소프트웨어 개발을 위해 필수적이기 때문에 전 세계 수많은 프로젝트에 활용되고 있다. 실제로 사용되는 데이터 형식은 상이할 수 있으나 프로젝트에 대한 오류 상황이나 해결책 등 개발자나 사용자들의 경험과 지식이 버그 추적 시스템에 내포되어 있기 때문에, 해당 저장소로부터 추출한 정보를 활용하여 소프트웨어 개발에 도움을 줄 수 있다. 이러한 MSR 연구 활동을 지원하기 위하여 본 논문에서는 주요 버그 추적 시스템으로부터 다형의 데이터를 수집하고 통합하여 저장소 독립모델을 구축하고, 수집된 데이터를 활용하여 연구를 수행하는데 필요한 기능을 지원하는 UniBAS 시스템을 제안 하였다.

UniBAS는 다양한 버그 추적 시스템으로부터 데이터를 추출하여 가공하고 그 결과로 연구자가 원하는 형태의 정보를 손쉽게 획득할 수 있게 할 뿐 아니라 외부 도구나 환경에서 활용할 수 있는 형태로의 출력을 지원함으로써 MSR 연구 활동에 도움을 줄 수 있다. UniBAS의 유용성을 검증하기 위한 사례 연구에서는 Mozilla 사이트의 Firefox 프로젝트에서 중복 버그 리포트를 탐지하는 연구를 수행하였다. 질의를 통하여 데이터를 살펴보고, 자연어 처리를 위하여 다양한 알고리즘을 간단히 선택할 수 있었기 때문에, 데이터를 추출하여 필요한 계산을 추가하는 것 만으로 관련 연구를 쉽게 수행할 수 있었다.

향후에는 첨부파일에 대한 분석을 추가하고, 소스 저장소 정보와 연계함으로써 보다 정밀한 수준에서 개발자의 지식 정보를 일반화하고 구체적인 버그 수정 패턴까지 연구에 활용할 수 있도록 UniBAS를 확장할 계획이다.

Acknowledgment

이 논문은 2012년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 연구되었음(This work was supported by the research grant of Chungbuk National University in 2012).

참고문헌

[1] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting structural information from bug reports," *Proc. 5th Working Conference on Mining Software Repositories* pp. 27-30, 2008.

[2] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," *Proc. 29th International Conference on Software Engineering*, pp. 499-510, 2007.

[3] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," *Proc. 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 70-79, 2012.

[4] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," *Proc. 34th International Conference on Software Engineering*, pp. 14-24, 2012.

[5] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where Should We Fix This Bug? A Two-Phase Recommendation Model," *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1597-1610, Nov. 2013.

[6] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," *Proc. 16th International Conference on Software Engineering & Knowledge Engineering*, pp. 92-97, 2004.

[7] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," *Proc. 28th International Conference on Software Engineering*, pp. 361-370, 2006.

[8] F. Servant and J. A. Jones, "WhoseFault: Automatic developer-to-fault assignment through fault localization," *Proc. 34th*

International Conference on Software Engineering, pp. 36-46, 2012.

[9] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in Eclipse," *Proc. 5th OOPSLA Workshop on Eclipse Technology eXchange*, pp. 21-25, 2007.

[10] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," *Proc. 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 308-318, 2008.

[11] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What Makes a Good Bug Report?," *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618-643, Sep. 2010.

[12] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How Long Will It Take to Fix This Bug?," *Proc. 4th International Workshop on Mining Software Repositories*, pp. 1-8, 2007.

[13] H. M. Tran, C. Lange, G. Chulkov, J. Schönwälder, and M. Kohlhase, "Applying Semantic Techniques to Search and Analyze Bug Tracking Data," *Journal of Network and Systems Management*, vol. 17, no. 3, pp. 285-308, May. 2009.

[14] M. Capraro, "Towards a Representative and Diverse Analysis of Issue-Tracker Related Code and Process Metrics," Friedrich-Alexander-University Erlangen-N¨urnberg, Germany, 2013.

[15] N. Kaushik and L. Tahvildari, "A Comparative Study of the Performance of IR Models on Duplicate Bug Detection," *Proc. 16th European Conference on Software Maintenance and Reengineering*, pp. 159-168, 2012.

[16] H. P. Luhn, "The Automatic Creation of Literature Abstracts," *IBM Journal of Research and Development*, vol. 2, no. 2, pp. 159-165, 1958.

저 자 소 개

이 재 권

2013: 충북대학교 컴퓨터공학과 공학사.
현 재: 충북대학교 대학원 컴퓨터공학과 석사과정
관심분야: 소프트웨어공학, 소프트웨어 저장소 마이닝
Email : exatoa@cbnu.ac.kr

정 우 성

2003: 서울대학교 컴퓨터공학부 공학사.
2011: 서울대학교 대학원 컴퓨터공학부 공학박사
현 재: 충북대학교 컴퓨터공학과 조교수
관심분야: 소프트웨어공학, 소프트웨어 저장소 마이닝
Email : wsjung@cbnu.ac.kr