Journal of The Korea Society of Computer and Information
Vol. 21 No. 8, pp. 65-75, August 2016

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.8.065

Design and Implementation of a Data Extraction Tool for Analyzing Software Changes

Yong-Hyeon Lee*, Kisub Kim**, Jaekwon Lee***, Woosung Jung****

Abstract

In this paper, we present a novel approach to help MSR researchers obtain necessary data with a tool, termed General Purpose Extractor for Source code (GPES). GPES has a single function extracts high-quality data, e.g., the version history, abstract syntax tree (AST), changed code diff, and software quality metrics. Moreover, features such as an AST of other languages or new software metrics can be extended easily given that GPES has a flexible data model and a component-based design. We conducted several case studies to evaluate the usefulness and effectiveness of our tool. Case studies show that researchers can reduce the overall cost of data analysis by transforming the data into the required formats.

▸ Keyword : Data extraction, Change analysis, Code ownership, Mining software repositories

• First Author: Yong-Hyeon Lee, Corresponding Author: Woosung Jung
* Yong-Hyeon Lee(gd0live@cbnu.ac.kr), Dept. of Computer Engineering, Chungbuk National University
** Kisub Kim(falcon@cbnu.ac.kr), Dept. of Computer Engineering, Chungbuk National University
*** Jaekwon Lee(exatoa@cbnu.ac.kr), Dept. of Computer Engineering, Chungbuk National University
**** Woosung Jung(wsjung@cbnu.ac.kr), Dept. of Computer Engineering, Chungbuk National University
• Received: 2016. 07. 14, Revised: 2016. 07. 18, Accepted: 2016. 08. 10.
• This work was supported by the research grant of Chungbuk National University in 2014, and was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2015R1C1A1A01054994, No. 2014M3C4A7030505)

I. Introduction

소프트웨어의 종류가 다양해지고, 규모가 커짐에 따라 개발 시의 복잡성도 크게 늘어나고 있다. 특히 대규모 소프트웨어의 경우 특정 개발자 한 명이 개발하는 것이 불가능하기 때문에 개발 시의 협업을 도와주는 버전 관리 시스템(VCS; Version Control System)이 많이 이용되고 있다[1]. 이 중에도 Git 기반의 저장소를 무료로 호스팅 하는 GitHub는 일반 사용자들 누구나 오픈소스 프로젝트를 생성하고 공유할 수 있도록 하는 특징을 가지고 있기 때문에 널리 사용되고 있다. GitHub에는 2016년 초 기준으로 2백만 개 이상의 프로젝트가 활동을 보이고 있으며, 활성화된 프로젝트들의 수는 2012년 이후로부터 전년 대비 50%에서 100% 가량 증가하고 있다[2]. VCS 기반의 오픈소스 저장소는 소프트웨어 저장소 마이닝(MSR; Mining Software Repositories) 분야의 발전에도 영향을 주었다. MSR 분야에서는 소스코드 저장소로부터 얻을 수 있는 데이터를 마이닝하여 소프트웨어 변경 패턴을 추적하고[3], 코드 결함을 탐지하거나[4], 특정한 일에 대해 전문지식이 있는 개발자를 파악하는[5] 등 다양한 연구를 진행하고 있다.

마이닝 프로세스에서 데이터의 양이 충분하지 않거나 불필요한 데이터가 섞일 경우, 의도했던 결과를 얻기 힘들다. 또한 충분한 데이터양의 확보와 불필요한 데이터의 여과에는 높은 컴퓨팅 파워와 큰 저장 공간이 요구되며 네트워크, 인코딩, 그리고 파일처리 등에서 다양한 오류가 발생한다. 이런 문제 때문에 MSR 연구자들은 온라인 소스코드 저장소에서 확보하는 데이터의 양과 질을 높이기 위한 시도를 계속해왔다[6].

Linstead는 웹으로부터 소스코드 저장소를 크롤링(Crawling)하여 코드를 구문분석하고 특성을 추출하여 저장하는 자동화된 도구를 개발하였다[7]. 또한 Dyer의 연구에서는 Hadoop 및 MapReduce를 기반으로 인프라를 구축, Boa라는 독자적인 프로그래밍 언어를 개발하여 많은 양의 데이터에 대한 질의를 쉽게 할 수 있도록 제공했다[8][9]. 그러나 기존의 연구들은 몇 가지 한계점을 가진다. 첫째, 대부분 소스코드의 변경정보를 저장하지 않아서 버전 사이에 나타날 수 있는 패턴에 대한 분석이 어렵다. 둘째, 시스템의 구조가 고정적이므로 새로운 언어나 시스템에 대한 확장이 용이하지 않다. 셋째, 수집된 데이터에 직접 접근하지 못하고 전용 언어를 통해 가공된 정보에만 접근이 가능하다. 이는 사용 편의성에는 도움이 되지만 일부 지원하지 않는 정보들에 대해서는 추가적인 커뮤니케이션이 필요해진다.

본 논문에서는 소스코드 저장소로부터 다양한 데이터를 추출하여 관계형 데이터베이스에 저장하고 간단한 쿼리들을 통해 데이터를 쉽게 가공할 수 있도록 도와주는 툴인 GPES(General Purpose Extractor for Source code)를 제안한다. GPES는 정의된 스키마에 맞추어 저장소로부터 버전 정보, 추상 구문 트리(AST; Abstract Syntax Tree)를 이용한 구조 정보, 버전 간 코드 비교 정보, 메트릭 정보를 추출한다. 최종 데이터에 대해서는 사용자가 스키마를 직접 접근할 수 있도록 허용하며, 확장 가능한 데이터 모델과 컴포넌트 기반 설계를 통해 새로운 언어의 분석을 지원하거나 기능추가를 가능하도록 구성함으로써 기존 연구들의 한계점을 보완하였다.

우리가 제안하는 툴이 얼마나 다양한 연구에 활용될 수 있는지 확인하기 위해 MSR 분야의 세 가지 연구 주제에 대해 데이터 제공 가능성을 실험했다. 실험은 오픈소스 프로젝트를 기반으로 진행했고, 우리는 각각의 사례 연구에서 간단한 데이터베이스 질의만으로 각 리비전별 파일 수, LOC(Line Of Code), 함께 변경된 코드 패턴, 개발자의 패키지 전문성 등 각 연구에 필요한 입력 데이터를 추출해 보았다. 입력 데이터를 추출하는데 사용한 쿼리를 분석한 결과 평균 6개의 테이블로부터 5개의 쿼리를 이용하여 추출할 수 있었다. 이러한 결과를 토대로, 연구자들은 GPES를 통해 다음과 같은 도움을 얻을 수 있다.

- 소프트웨어의 모든 리버전에 대해 AST 수준의 구조 정보를 포함하여 품질, 관계, 변경 정보 등 분석에 필요한 기초 데이터를 제공함으로써, 특정 스냅샷뿐만 아니라 진화 과정에 대한 연구를 여러 관점 및 수준에서 지원할 수 있다.
- 제안하는 스키마를 만족하도록 소프트웨어 정보를 자동 추출하기 때문에, 적은 비용의 쿼리를 통해 분석 과정에서 필요한 다양한 형태의 데이터를 쉽게 획득할 수 있다.
- 반복적이고 오류를 발생시키기 쉬운 단계인 데이터 추출 및 정제에 대한 연구자의 부담을 줄여줌으로써 핵심 분석 단계에 보다 집중할 수 있도록 한다.
- 유연한 데이터 모델과 컴포넌트 기반 설계를 통해 새로운 VCS시스템, 언어 또는 메트릭 등에 대한 확장이 용이하다.

본 논문의 이후 구성은 다음과 같다. 2장에서는 연구에 필요한 배경 지식 및 관련 연구를 제시하고, 3장과 4장에서는 GPES의 설계 및 구현과 데이터베이스 스키마에 대해서 설명한다. 5장은 GPES를 이용한 사례연구를 보이고 6장에서 향후 연구를 포함한 결론을 내린다.

II. Related works

이 장에서는 버전 관리 시스템, 버그저장소 등과 같은 온라인 저장소에 축적된 데이터를 추출 및 마이닝 하여 소프트웨어 개발 프로젝트의 생산성, 품질 등에 기여하고자 하는 MSR 분야의 선행 연구들을 조사하였다.

1. MSR Challenge

MSR 분야에서는 버전 정보, 버그 정보, 프로젝트 관리 정보 등 다양한 대상의 데이터 수집을 꾸준히 시도하고 있다. 특히, 소스코드 저장소는 데이터의 양과 유형이 다양하기 때문에 지속적인 연구의 대상이 되어 왔다.

다양한 데이터를 수집하기 위한 연구로 Eclipse IDE를 통해 버전 관리 시스템에 반영되지 않은 로컬 작업 기록을 수집하여 프로젝트의 클래스별 개발 활동량을 시각화하고 클래스 간의 근접성을 파악하는 도구가 개발되었으며[10], 소스코드 정보를 AST 기반으로 추출하여 각 노드에 대해 원자적 변경을 계산하고, 여러 조합을 통해 리팩토링이나 버그 수정, 기능 추가 등을 구분하는 연구가 진행되었다[11].

한편, 대규모 데이터를 수집할 수 있는 기반에 대한 연구로, 다양한 요구사항에 바탕을 두고 소프트웨어 진화를 연구하는 Kenyon 시스템이 있었다[12]. 이 시스템은 설정에 따라 사실 추출기, 메트릭 로더의 인터페이스로 구현된 외부 도구를 통해 데이터를 추출할 수 있도록 설계되었다. 그리고 Sourcerer라는 검색엔진에서는 웹으로부터 소스코드 저장소를 크롤링하여 파서와 사실 추출기를 통해 패키지, 클래스/인터페이스, 메서드 및 필드 단위의 정보 분석과 코드 검색을 지원하였다[7]. 대규모 데이터 수집의 필요성이 점차 증가함에 따라 일부 연구자들은 분산처리를 기반으로 MSR 연구의 확장을 시도하였고, 그에 따른 문제점과 해결방안에 대해 논의하였다[13]. 이 과정을 통해, Dyer는 Hadoop 기반의 코드 추출도구인 Boa를 개발했고, GitHub, SourceForge로부터 데이터를 수집했다. 또한 독자적인 질의 언어를 제공하여 사용자가 수집된 데이터를 쉽게 획득 및 가공할 수 있도록 제공했다[8][9].

2. Software Change Analysis

소프트웨어는 시간이 흐름에 따라 많은 변경을 거치며, 축적된 변경 정보에는 다양한 지식들이 내포된다. 변경 원인을 파악하거나, 파일이나 클래스 등 소프트웨어 구성요소들의 변경 패턴을 추출함으로써 요소들 간의 연관성을 찾고, 더욱 안정적인 소프트웨어 진화를 지원하고자 하는 연구들이 진행되었다.

Mockus는 수정요청에서 서술된 변경에 대한 설명과 변경된 코드를 이용하여 적응형, 교정형, 완료형 변경으로 분류하는 방법을 제안했고[14], 변경된 파일들을 시각화하고 시각 패턴을 이용하여 불안정한 컴포넌트와 응집된 개체, 설계의 진화, 그리고 팀의 생산성 변동 등을 파악하는 방법도 소개되었다[15]. 또한, 오픈소스 프로젝트인 ARLA 시스템을 통해 소프트웨어의 진화과정을 조사하는 연구도 있었는데, 그들은 LOC, 파일 개수 등을 기반으로 애플리케이션의 규모, 디렉터리에 대한 레벨과 깊이, 너비 등의 메트릭을 정의하여 구조적 진화를 분석했다[16]. 변경의 분류와 진화에 대한 연구와는 별개로, 여러 릴리즈 버전에서 함께 변경된 횟수를 계산하여 서브시스템 또는 모듈 간의 결합도를 구함으로서 소스코드 분석으로는 파악할 수 없었던 서브시스템 간의 종속성에 관한 연구도 진행되었다[17][18]. 나아가, 함께 변경된 산출물을 그래프로 나타내고, 그에 대해 간선-반발 에너지 모델 기반의 그래프 군집방법으로 산출물들을 군집화 해본 연구와[19], 각 버전에서 함께 변경된 파일, 클래스 등에 대한 연관 규칙 마이닝을 기반으로 하는 ROSE라는 도구가 개발되었다[3]. ROSE는 기존의 코드 분석으로는 발견되지 않는 개체 간 결합성을 파악하고 함께 변경될 만한 코드를 예측하거나, 불완전한 변경을 경고하기도 한다.

3. Developer Code Ownership and Expertise

버전 관리 시스템에서는 코드를 변경한 개발자 정보를 기록하기 때문에 그들의 활동을 분석할 수 있고 개발자별 전문성에 관한 정보를 얻을 수 있다. 여러 연구에서 특정 분야에 대한 전문가 또는 유사한 수준의 개발자를 탐색하거나 추천하고, 그들의 활동이 소프트웨어 진화에 미치는 영향을 분석함으로써 생산성과 제품의 품질향상에 기여하였다.

먼저, 소프트웨어의 진화적 관점에서 개발자의 활동이나 코드 소유권을 분석한 연구들이 있었다. 시간에 따라 각 파일이 변경된 양과 소유권을 가진 개발자를 표현하는 소유권 지도라는 시각화 기법을 제안한 연구[20], 소스코드 저장소와 버그 리포트 등에서의 개발자 활동을 분석하여 소프트웨어 개발 비용 예측에 활용한 연구도 있다[21]. 특정 컴포넌트들에 대한 개발자별 소유권이 소프트웨어 품질에 끼치는 영향을 파악하는 연구도 진행 되었는데, 소유권의 높고 낮음에 따라 그룹을 나누어 조사한 결과, 소유권이 낮은 개발자가 많아지면 품질이 낮아지고 결함이 발생하기 쉬우며, 소유권이 높은 개발자가 많아지면 품질이 좋아지는 현상을 발견했다. 이 결과로, 약한 책임을 가진 다수의 개발자들보다 강한 책임을 가진 소수의 개발자들이 품질향상에 더 좋은 영향을 끼친다는 것을 알 수 있다.

개발자의 코드 소유권과는 다르게 특정 분야에 대한 전문성을 평가하고자 하는 연구들도 있었다. 소스코드에서 등장하는 어휘들을 여러 도메인으로 분류하고, 개발자별로 사용한 어휘들을 분석하여 도메인 전문성을 파악하는 연구[22], 개발자의 전문지식을 토대로 함수를 구현하는 구현지식과 함수를 호출하는 사용지식으로 구분하고, 개발자의 사용 지식을 추출함으로써 특정 API 사용의 전문가를 찾거나, 개발자 간의 근접성을 분석하는 방법[23] 등이 소개되었다.

III. GPES (General Purpose Extractor for Source Code)

GPES는 범용 목적의 소스코드 추출기로써 MSR분야의 다양한 연구를 지원하기 위해 고안된 시스템이다. 범용 목적을 위해서 대표적인 VCS인 Git을 대상으로 소스코드의 모든 정보들을 수집하는 것을 목표로 한다. 이를 위해서 소스코드의 구조정보뿐 아니라 변경 및 품질정보도 함께 수집하도록 구성하였다. GPES는 기본적으로 Git을 활용하지만 대부분의 타 VCS들이 Git으로의 변환을 지원하고 있으므로 다양한 VCS에 대한 데이터 추출도 지원 가능하다.

본 장에서는 GPES의 아키텍처와 각 정보의 추출방법에 관하여 설명한다.

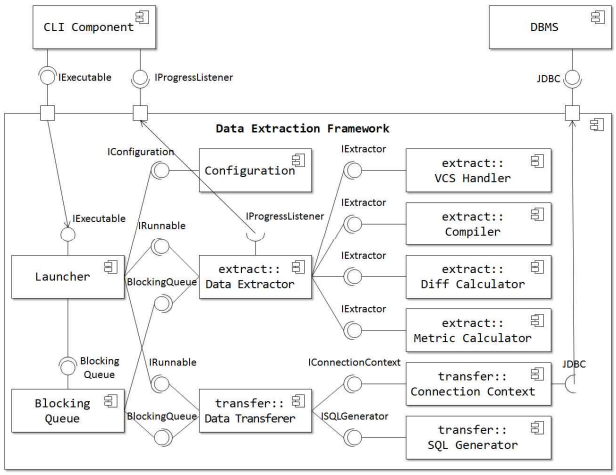

Fig. 1. GPES component based architecture

1. Software Architecture Design

GPES는 다양한 VCS, 언어, 메트릭에 대한 추출을 위하여 여러 컴포넌트로 구성되었고, 각각은 실행부, 추출부, 전송부로 나뉜다. Fig. 1은 GPES의 전체 아키텍처를 보여준다.

실행부는 커맨드라인 인터페이스(CLI)를 통해 VCS 유형, 저장소 URL, DB정보 등을 입력받아 동작하며, GPES의 환경설정과 추출작업에 필요한 객체들을 생성한다. 실행기(Launcher)는 각 입력 값에 따라 적절한 컴포넌트 객체를 생성하며 추출부와 전송부의 작업을 관리한다.

추출부와 전송부는 Data Extractor와 Data Transferer가 각 부의 중심 컴포넌트로써 하위 컴포넌트의 관리를 수행하며, 각자 별도의 스레드로 동작할 수 있다. 두 컴포넌트는 스레드 안전성(Thread safety)을 보장하기 위하여 BlockingQueue를 사용하여 데이터를 공유한다. Data Extractor는 VCS 핸들링, AST 구축, diff 계산, 메트릭 계산을 수행하는 컴포넌트를 이용하여 실제 데이터를 추출하며, Data Transferer는 JDBC를 이용하는 두 컴포넌트를 통해 데이터베이스 연결을 관리하고 쿼리문을 생성하여 데이터베이스에 추출된 정보들을 저장한다.

2. Extraction of Structural Data

구조적 정보 추출과정은 VCS제어(VCS Handler)와 컴파일러(Compiler) 컴포넌트가 담당한다. 먼저 VCS제어 컴포넌트는 원격 저장소를 로컬로 복제한 후, 로컬 저장소로부터 개발자 정보와 리비전 정보들을 추출한다. 리비전 정보에는 식별자, 작성자, 변경 시각, 작성자가 남긴 커밋 메시지, 부모 리비전 식별자, 자식 리비전 식별자를 포함한다. Git의 log명령어를 통해 각 정보들을 얻을 수 있지만 일부 제공되지 않는 정보들과 분석에 필요한 비용이 발생하여 본 논문에서는 Eclipse에서 제공하는 JGit라이브러리를 이용하여 추출한다.

VCS제어 컴포넌트는 추출된 리비전 정보를 기반으로 각 리비전별 파일 정보를 추출하게 되는데, 먼저 체크아웃을 통해 각 리비전 별 스냅샷을 별도의 디렉토리에 저장한다. 이는 이후 수행되는 추출작업에서 반복적인 체크아웃을 통해 낭비되는 비용을 줄여준다. 이 후 모든 리비전의 스냅샷 디렉터리를 순회하여 파일 목록을 작성하고, 파일 입출력을 통해 파일의 줄 단위 정보를 추출한다. 대부분의 파일정보는 Java API를 통해 얻을 수 있지만, 파일내용(FileContent)의 해시값은 Git에서 제공하는 hash-object 명령어를 통해 얻어야 한다.

컴파일러 컴포넌트는 각 리비전의 스냅샷 디렉터리로부터 소스코드의 AST를 구축한다. 스냅샷 내의 각 파일들은 서로 의존관계를 가지므로 각 파일별로 AST를 구축하는 것이 아니라 스냅샷 단위로 컴파일을 해야 한다. 컴파일러 컴포넌트는 스냅샷 AST로 부터 변수 및 객체, 객체가 의존하는 자료형 정보와 함께 전체 소스코드 구조를 추출한다. 추출된 각 정보는 Declaration, Expression, Statement 등과 같은 AST 노드로 구성된다. 현재 GPES는 Java를 대상으로 하고 있어 Eclipse의 자바 개발 툴킷(JDT; Java Development Toolkit)을 활용하며, 컴파일러의 특성에 따라 방문자(Visitor) 패턴을 적용하여 정보들을 추출한다. 다른 언어들의 정보를 추출하기 위해서는 각 언어에 맞는 컴파일러를 이용하여 컴파일러 컴포넌트 인터페이스에 맞게 제작하면 손쉽게 확장 가능하다.

3. Extraction of Evolutional Data

변경 정보 추출 과정은 Diff계산기(Diff Calculator)가 담당하고 있으며, diff 명령어를 통해 각 리비전에서 이전 리비전을 비교해 변경된 파일 목록을 작성하고, 각 파일에 대해 변경된 내용을 줄 단위와 AST노드 단위로 추출한다. Fig. 2의 (a)는 줄 단위 diff 출력결과를 +, -를 통하여 나타낸다. 첫 줄에는 줄 번호를 이용하여 변경된 코드의 전체 범위를 보여주며, 이

```
@@ -3,10 +3,10 @@ import java.util.Arrays;
 import java.util.List;

 public class RuleList {
-       public List<Rule> data = new ArrayList<>();
+       public List<Rule> rules = new ArrayList<>();

        @Override
        public String toString() {
-               String str = Arrays.toString(data.toArray(new Rule[0]));
+               String str = rules.toString();
                return str;
        }
```

(a) line-level diff

```
@@ -3,10 +3,10 @@ import java.util.Arrays;
import java.util.List;

public class RuleList {
        public List<Rule> [-data-]{+rules+} = new ArrayList<>();

        @Override
        public String toString() {
                String str = [-Arrays.toString(data.toArray(new Rule[0]));-]{+rules.toString();+}
                return str;
        }
```

(b) word-level diff

Fig. 2. Diff result of changed source code

범위에는 실제 변경이 발생한 줄 외에 ±3줄 정도를 포함한다. 예를 들어 @@ -3,10 +3,10 @@@는 이전 버전의 세 번째 줄부터 열 개의 줄이 다음 버전의 세 번째 줄부터 열 개의 줄로 변경되었음을 의미하며, 실제 변경이 발생한 줄에는 +,-기호가 표시된다. +, -가 표시된 줄은 한 줄로 본다.

Fig. 2의 (b)는 단어 단위 diff 출력결과를 보여주며, 변경된 소스코드를 {++}, [--] 기호를 통해 추가된 범위와 삭제된 범위를 나타낸다. 단어 단위 diff는 바이트(byte) 단위의 시작점과 끝점을 통해 변경된 범위를 나타낼 수 있다. 또한, AST의 각 노드에도 바이트 위치 정보가 있으므로 단어 단위 diff 결과와 범위 대조를 통해 변경된 AST노드를 파악하는 알고리즘을 Fig. 3과 같이 설계할 수 있다.

파일 f에 대한 이전 버전의 파일을 f'라고 했을 때, f'에서 f로의 변경에 대해 단어 단위 diff를 구하는 함수 $word_diff(f,f')$를 정의하고, 각각의 변경된 부분은 $diff_x \in word_diff(f,f')$로 표현한다. 이때, 임의의 변경된 부분 $diff_x = (diff_x^-, diff_x^+)$은 f'에서 삭제된 부분 $diff_x^-$과 f에 추가된 부분 $diff_x^+$으로 구성한다. 그리고 변경 범위 $[diff_x]$는 바이트 시작점부터 끝점까지의 구간으로, 삭제된 구간 $[diff_x^-] = [a,b] = \{a,b \in \mathbb{Z} \mid 0 \le a < b < |f'|\}$과 추가된 구간 $[diff_x^+] = [c,d] = \{c,d \in \mathbb{Z} \mid 0 \le c < d < |f|\}$의 쌍 $[diff_x] = ([diff_x^-], [diff_x^+])$으로 정의한다. 예를 들면, Fig. 3에 대한 단어 단위 diff는 수식 (1)로 표현된다.

$$\begin{aligned} word\_diff(f,f') &= \{diff_1, diff_2\} \\ &= \{(diff_1^-, diff_1^+), (diff_2^-, diff_2^+)\} \\ &= \left\{\begin{array}{l}(\text{“data”}, \text{“rules”}), \\ (\text{“Array....le[0]));”}, \ \text{“rules.toString();”})\end{array}\right\} \end{aligned} \quad (1)$$

파일 f에 대한 AST는 여러 개의 노드를 원소로 갖는 집합 $AST_f = \{node_x \mid 0 \le x < T,\ T \text{ is total count of nodes}\}$으로 정의하며, $node_0 \in AST_f$는 루트 노드를 나타낸다. 임의의 노드 $node_x$의 부모 노드를 구하는 함수를 $parent(node_x)$, 자식 노드를 구하는 함수를 $child(node_x)$로 표현하면, 모든 $node_x$에 대해 수식 (2), (3)을 만족한다. 또한, AST노드 $node_x$에 대한 구간은 $[node_x] = [a,b] = \{a,b \in \mathbb{Z} \mid 0 \le a < b < |f|\}$로 표현되며, 항상 수식 (4)를 만족한다.

$$N(parent(node_x)) = \begin{cases} 0, & if\ x = 0; \\ 1, & otherwise. \end{cases} \quad (2)$$

$$N(child(node_x)) = \begin{cases} 0, & if\ node_x\ is\ leaf; \\ m \in \mathbb{N} : 0 \le m < T - x, & otherwise. \end{cases} \quad (3)$$

$$\begin{aligned} &[node_0] = [0,\ |f| - 1] \\ &[node_x] \subseteq [parent(node_x)],\ if\ x \neq 0. \end{aligned} \quad (4)$$

Fig. 3의 알고리즘은 단어 단위 diff와 AST노드의 바이트 구간을 비교하는 과정을 반복한다. 노드 $node_x$에 대한 구간 $[node_x]$이 변경된 구간 $[diff_y]$에 속하면 $node_x$는 변경된 노드이다. 만약 $node_x$가 리프 노드라면, 노드의 구간이 변경된 텍스트 구간을 포함할 때만 $node_x$를 변경된 노드로 계산한다.

```
ASTNode_DiffCalculation:
FOR diff_i ∈ word_diff(f,f')
  FOR node_j ∈ AST_f'
    IF ( [node_j] ⊆ [diff_i^-] AND N(child(node_j)) > 0 )
    OR ( [diff_i^-] ⊆ [node_j] AND N(child(node_j)) = 0 )
      THEN node_j is DELETED from f' to f
  FOR node_k ∈ AST_f
    IF ( [node_k] ⊆ [diff_i^+] AND N(child(node_k)) > 0 )
    OR ( [diff_i^+] ⊆ [node_k] AND N(child(node_k)) = 0 )
      THEN node_k is ADDED from f' to f
```

Fig. 3. AST node-level Diff algorithm

4. Extraction of Qualitative Data

소프트웨어 메트릭에 관련해서는 꾸준히 새로운 연구가 진행되고 있으며 메트릭마다 측정하는 측면과 방법, 스케일이 다양하다. 여러 정적 분석 도구를 통해 메트릭을 계산할 수 있으며, GPES는 Understand로부터 파일, 패키지, 클래스, 메서드 등에 대한 LOC, 순환 복잡도, 개체 간 결합도, 메서드 응집도, 상속 깊이 등 Java에 대해 50가지 품질 정보를 추출한다.

메트릭 추출 컴포넌트(Metric Caculator)는 Understand의 커맨드라인 인터페이스를 통해 출력된 결과를 이용하여 메트릭 측정 대상의 종류와 이름, 각 메트릭에 대한 측정값을 얻는다. Understand는 기본 설정에서 대상의 이름에 대해 파일의 경우 경로를 제외한 이름만 표시하고, 패키지와 클래스 등의 논리적 요소들은 모든 패키지 정보를 포함하는 수식명으로 표시한다. GPES는 메트릭 측정 대상의 종류와 이름으로 전 단계에서 추출된 버전과 파일, AST노드 개체와 맵핑하여 측정된 값을 저장한다.

IV. Data Model Design

GPES의 데이터 모델은 구조 정보, 변경 정보, 품질 정보의 3가지로 구성되며, Fig. 4는 이들에 대한 스키마를 크로우즈 풋(Crow's Foot) 표기법을 이용한 ER-다이어그램으로 나타낸다.

1. Structural Data

구조 정보는 VCS의 구조를 나타내는 저장소(Repository), 리비전(Revision), 개발자(Developer), 파일(File), 그리고 소스코드의 구조를 나타내는 AST노드(ASTNode) 및 자료형(Type) 등으로 구성된다.

Repository는 원격 저장소의 URL을 통해 객체를 구별하며 가독성을 위해 URL을 축약한 이름(name) 속성을 지닌다. Revision은 VCS에서 부여된 해시 값을 고유값으로 하며, 리비전의 저자와 커미터, 변경 시간, 그리고 커밋(commit) 메시지 등 VCS의 log로부터 추출한 모든 정보를 포함한다. Developer는 저자 또는 커미터로서 각자의 이름과 이메일 주소를 속성으

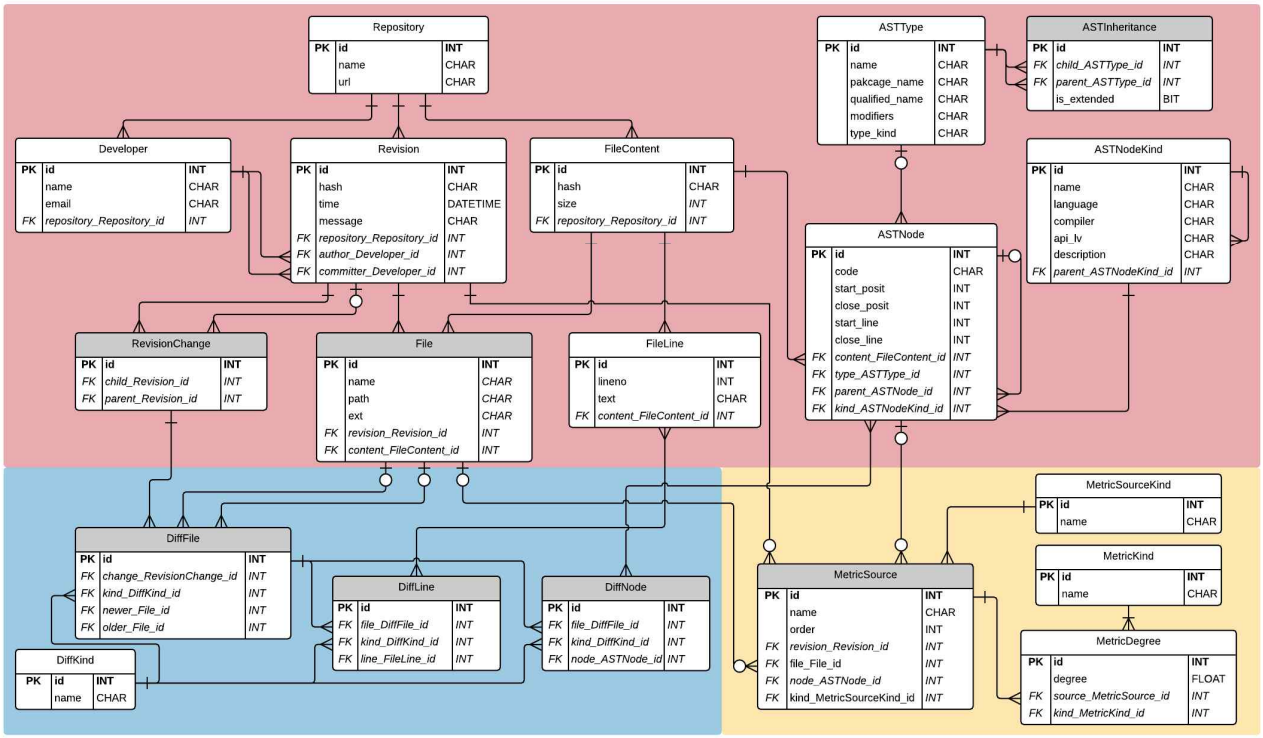

Fig. 4. GPES database schema

로 지닌다. RevisionChange는 변경 전후의 리비전을 연결하여 히스토리 그래프를 나타내며 구조 정보와 변경 정보를 잇는 구심점 역할을 한다. GPES는 Git 파일 관리 구조를 그대로 추출하므로, 파일의 메타정보를 가지는 File과 파일의 내용을 나타내는 FileContent로 나누어 추출하도록 하였다. 이는 리비전별로 중복된 파일을 다시 추출하지 않도록 해준다. File은 파일명(name), 상대경로(path), 확장자(ext)를 속성으로 가지고 외래키로 소속된 Revision과 FileContent를 참조하며, FileContent는 이진 데이터를 묘사하는 해시 값과 바이트 크기의 속성을 가진다. 텍스트 파일의 내용은 줄 번호(lineno)와 텍스트 값(text) 속성을 통해 줄 단위로 FileLine 테이블에 저장된다.

2. Evolutional Data

변경 정보는 리비전 변경 시 달라진 파일과 줄 단위, AST 단위의 파일내용으로 구성된다. DiffKind는 상수 테이블로 추가, 삭제, 수정 중 어떤 변경이 이루어졌는지를 나타내며, 향후 변경 계산 알고리즘이 개선된다면 이름변경이나 코드복제, 리팩토링 등의 변경 유형도 추가될 수 있을 것이다. FileDiff는 두 리비전 간의 변경된 파일을 가리키며 RevisionChange를 참조하여, 이전 파일(older_File_id)의 값이 없는 경우 파일의 추가를, 새로운 파일(newer_File_id)이 없는 경우 파일의 삭제를, 그리고 양쪽 모두 값이 존재하는 경우 파일의 수정을 나타낸다. FileLineDiff와 ASTNodeDiff는 줄 단위와 AST노드 단위로 추가되거나 삭제된 소스코드 내용을 가리키고, 각각에 해당하는 변경된 파일 내용(line_FileLine_id)과 변경된 AST노드(node_ASTNode_id)를 참조한다.

3. Qualitative Data

품질 정보는 패키지, 파일, 클래스, 메서드에 대해 소프트웨어 메트릭을 측정한 정보로 구성된다. 다양한 메트릭을 지원하기 위해 메타모델링을 도입하여 측정 대상과 측정치를 나타내는 MetricSource와 MetricDegree 테이블로 나누고 각각의 유형을 나타내는 MetricSourceKind, MetricKind을 상수 테이블로 정의하였다. MetricSource은 대상의 이름과 순서(order), 유형 정보를 가지며, MetricDegree는 측정값과 측정된 대상, 메트릭 유형을 속성으로 지닌다. 측정 대상의 순서를 파악하는 이유는 소스코드 상에 등장하는 메서드 오버로딩, 익명 클래스 등으로 인해 이름만으로 분별 불가능한 경우가 있기 때문이다.

V. Case Studies

본 논문에서는 GPES의 유용성을 입증하기 위해 서로 다른 주제의 기존 MSR 연구를 참고하여 GPES의 데이터를 가공해 보고하는 실험을 수행하고, 쿼리의 수를 통해 데이터 추출비용을 평가 하였다.

실험을 위하여 Github에서 데이터가 충분할 것으로 예상되는 12개의 프로젝트를 선정하였다. 오픈소스 라이선스를 따르며, 여러 번의 릴리즈를 통해 안정화되고, 인지도가 높으면서

Table 1. Target projects summary

Project	rev.	dev.	init. date	last commit
auto	644	44	2013-05	2016-04
blade	486	13	2015-07	2016-04
Java-Chronicle	306	13	2012-01	2016-01
javapoet	657	56	2012-06	2016-03
nanohttpd	511	66	2012-08	2016-05
netty-socketio	695	24	2012-01	2016-04
okio	385	24	2011-05	2016-02
reactive-streams-jvm	351	21	2014-02	2016-04
retrolambda	434	11	2013-07	2016-01
scribejava	545	85	2010-09	2016-04
socket.io-client-java	233	10	2013-04	2016-02
spark	662	81	2011-05	2016-04

Table 2. The statistics of the main tables among data extracted from GPES

Table	Total Records	Average Records
Revision	5,909	492.4
Developer	448	37.3
File	577,473	48,122.8
FileContent	20,811	1,734.3
ASTNode	11,063,423	921,951.9
Type	40,616	3384.7

여러 개발자가 참여한 프로젝트를 첫 번째 기준으로 세웠다. 또한, 현재까지는 Java에 한해 AST 추출이 가능하기 때문에 Java 사용률이 90%가 넘는 조건을 포함하였다. 그중에서 안드로이드 프로젝트는 참조하는 클래스 및 인터페이스를 정의하는 코드가 저장소에 포함되지 않아서 Type에 관한 데이터가 추출되지 않기 때문에 제외하였다.

Table 1은 실험 대상으로 선정된 12개 프로젝트에 대한 요약 정보를 나열하며, 각 프로젝트들은 평균적으로 492개의 리비전을 포함하고 37명의 개발자가 기여하였고, 최근까지 활성화된 프로젝트임을 보여준다. Table 2는 GPES를 통해 추출된 데이터 중 주요 테이블에 대한 통계정보를 나타낸다. 총 5,909개의 리비전에 대해 577,473개의 파일 스냅샷과 20,811개의 파일내용, 그리고 11,063,423개의 AST노드와 40,616개의 자료형이 추출된 것을 알 수 있다.

1. Experimental Results

1.1 Software Evolution

Capiluppi의 연구에서는 파일의 개수, 크기, LOC 등을 통해 소프트웨어의 규모와 디렉터리 구조의 너비와 깊이를 파악하였고, 소프트웨어 구조의 안정성을 측정하여 릴리즈 변경에 따라 각 속성이 변화하는 과정을 분석하였다[16].

Fig. 5는 리비전에 따른 파일 개수의 변동 추세를 보여준다.

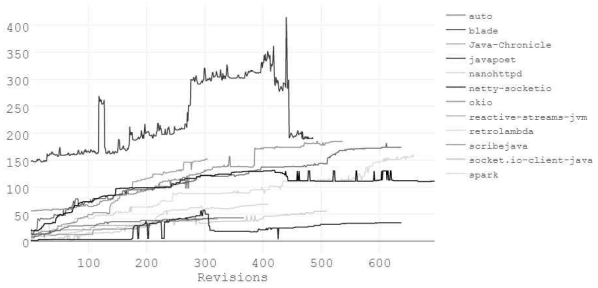

Fig. 5. Total number of files

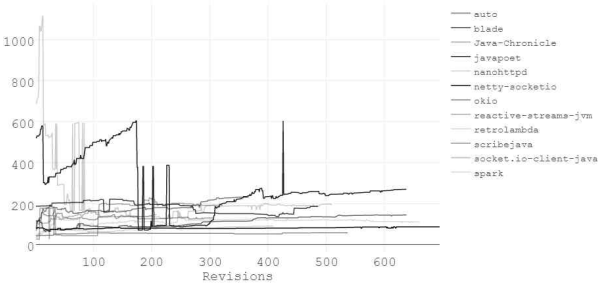

Fig. 6. Average LOC of files

대부분 프로젝트에서 파일 개수가 선형으로 증가하는 양상을 보이며, 각 데이터의 선형 추세선을 그렸을 때 평균 R^2 값은 0.82이었다. R^2이 높은 9개 프로젝트는 모두 0.83 이상인 반면, 나머지 프로젝트는 0.50 이하의 값을 보였다. 그 중에서도 netty-socketio 프로젝트는 규모가 일정하게 수렴하는 양상을 보이는 것과 반대로 blade, javapoet 프로젝트는 특정 시기에 파일 개수가 급격하게 변화하는 경우가 존재하였다. 해당 리비전의 커밋 로그를 분석한 결과 새로운 기능과 테스트용 소스코드의 추가, 기존 기능 삭제와 서브 모듈의 저장소 이전, 그리고 리팩토링 등의 작업으로 파일 개수가 급격히 변경된 것을 확인하였다. 또한 일정 수준으로 상승과 하락을 반복하는 경우는 브랜치(branch)가 나뉨으로써 발생하는 현상임이 파악되었다.

Fig. 6은 리비전별 소스코드의 평균 LOC를 그래프로 표현한 결과이다. 많은 프로젝트에서 평균 LOC가 초기에 큰 폭으로 변화하다가 후반으로 갈수록 변동 폭이 줄어드는 양상을 보인다. 커밋 로그를 살펴본 결과 초기에는 새로운 기능을 추가하는 작업이 많지만, 뒤로 갈수록 소스코드를 최적화하고 버그를 수정하는 작업이 증가하여 프로젝트가 점차 안정화되었음을 확인하였다.

1.2 Change Patterns

Zimmermann의 연구에서는 리비전마다 함께 변경된 파일, 클래스, 메서드 간에 대해 연관 규칙 마이닝을 적용하여 다음 변경을 예측하거나 불완전한 변경을 경고하는 도구를 제안하였다[3]. Fig. 7은 GPES를 사용하여 얻은 4개 프로젝트에 대한 파일 간 상호변경된 관계를 시각화 한 그림이다. 각 노드는 파일을 나타내고 노드의 크기는 상대적인 파일 크기를 나타내며, 간선이 짙을수록 함께 변경된 횟수가 많음을 나타낸다.

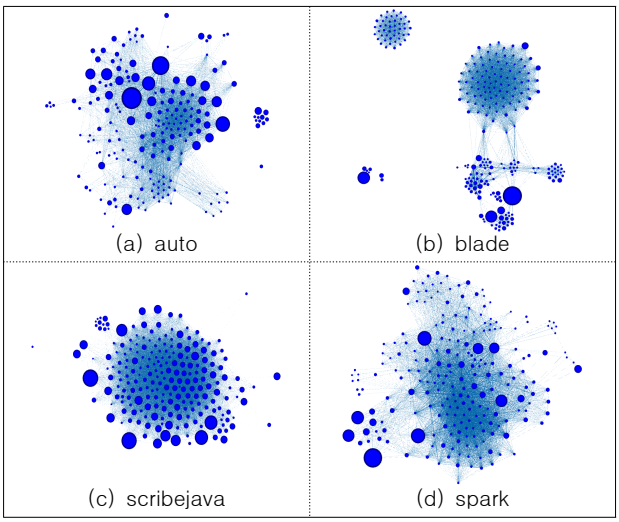

Fig. 7. Visualization of co-changed patterns

시각화 결과는 프로젝트마다 다른 패턴을 보였다. Fig. 7의 (a) auto 프로젝트와 (d) spark 프로젝트는 상호변경도가 높은 여러 군집이 겹쳐진 형상을 보였고, (b) blade 프로젝트는 각 군집을 뚜렷하게 확인할 수 있었다. 반면에, (c) scribejava 프로젝트는 군집이 구분되지 않았고, 이 프로젝트에서는 파일이 함께 변경된 횟수가 비슷하여 상관관계가 높은 파일들을 찾을 수 없었다.

그래프의 패턴을 보았을 때 blade 프로젝트는 파일 간의 의존 관계가 여러 군집으로 명확히 분류되므로 임의의 파일을 변경할 때, 전체 프로젝트에 끼치는 영향이 가장 적을 것으로 예측 된다. 하지만, scribejava 프로젝트의 경우 하나의 파일이 변경될 때 어떤 파일이 함께 변경될지 예측하기 어려우므로 하나의 파일에서 변경이 일어날 때, 프로젝트에 끼치는 영향이 클 것으로 추측할 수 있다.

1.3 Code Ownership

Girba의 연구에서는 리비전에 따른 각 파일에 대해 LOC 단위로 개발자들의 변경량을 계산하여 누적 변경량이 가장 많은 개발자에게 해당 파일의 소유권을 부여하고, 모든 파일에 대해 리비전에 따른 변경량과 소유권을 나타내는 Ownership Map이라는 시각화 방법을 제안하고 개발자들이 프로젝트에 기여하는 패턴들을 분석하였다[20]. 본 실험에서는 변경된 AST노드의 개수를 통해 변경량을 계산하고, 개발자별 소유권의 비율이 변화하는 과정을 시각화하였다. 파일 단위가 아닌 프로젝트 전체에 대해 코드 소유권을 계산하는 과정에서 파일 이름 변경이나 메서드 이동 등 리팩토링으로 인해 코드 소유권이 비약적으로 높아지는 경우를 발견하였다. 이를 배제하기 위해 추가된 AST노드의 수에서 삭제된 AST노드의 수를 뺀 값을 코드 변경량으로 정의했고, 삭제한 AST노드의 수가 더 많은 경우는 0으로 계산하여 음수가 나오지 않도록 하였으며 개발자별 코드 소유권은 누적 변경량으로 계산하였다.

Fig. 8, Fig. 9, Fig. 10은 각각 auto, NanoHttpd, spark 프로젝트에 대해 개발자의 코드 소유권 변동 추이를 보여준다. 각 그래프는 코드 소유권이 가장 높은 상위 5명의 개발자의 이름과 그 외의 개발자로 나누어 나타냈으며, 전체 소유권의 총합 대비 각 개발자의 소유권 비율을 나타내고 있다.

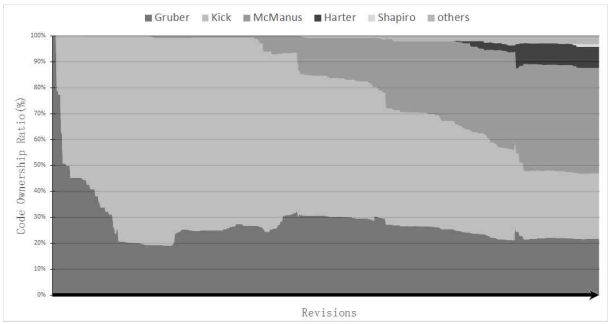

Fig. 8. Code ownership ratio in auto project

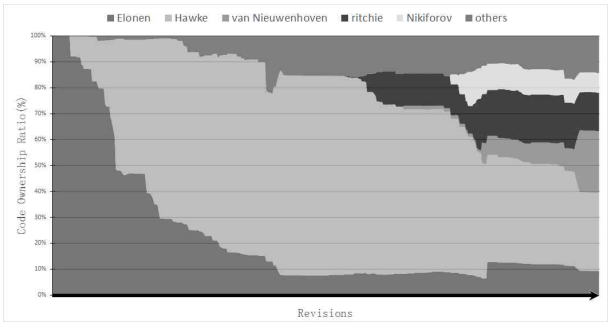

Fig. 9. Code ownership ratio in NanoHttpd project

Fig. 10. Code ownership ratio in spark project

Fig. 8의 auto 프로젝트와 Fig. 9의 NanoHttpd 프로젝트는 초기부터 활동한 개발자보다 2번째, 3번째로 활동을 시작한 개발자의 코드 소유권이 더 많아지는 것을 볼 수 있다. 이러한 패턴은 Girba의 연구에서 정의한 행동 패턴 중에서 한 명의 개발자가 독자적으로 개발하는 Monologue 패턴보다는 파일의 코드 소유권이 다른 개발자에게 점진적으로 전이되는 Familiarization 패턴 또는 빠르게 전이되는 Takeover 패턴에 해당하는 것으로 보인다.

앞서 제시된 두 프로젝트와는 다르게 Fig. 10의 spark 프로젝트는 초기부터 활동한 개발자의 코드 비율이 굉장히 높고, 상위 5인을 제외한 다른 개발자들의 활동량이 많은 것을 알 수 있다. 실제 데이터를 확인한 결과, 상위 1인을 제외한 여러 개

발자의 활동량이 비교적 고르게 분포된 것을 파악하였다. 이는 spark 프로젝트가 Wendel이 생성한 소규모의 프로젝트이며 다른 개발자들이 조금씩 참여해왔기 때문이다. Girba의 행동 패턴을 기준으로 분석하면 Monologue 패턴을 보이며, 여러 명에 의한 Teamwork 패턴 또는 파일 일부분을 변경하는 Bugfix 패턴, Edit 패턴에 해당한다.

2. Discussion

Table 3은 각 실험에서 하나의 그래프를 그리기까지 사용한 테이블과 쿼리의 수를 보여준다. 예를 들면, 개발자의 코드 소유권을 추출하기 위해서 6개의 테이블과 중첩을 포함하여 5번의 SELECT 문이 사용되었고, 임시테이블에 대해 4번의 INSERT 또는 UPDATE 문, 그리고 CURSOR를 이용한 1번의 루프 구문이 사용되었다. 이러한 결과는 GPES를 활용함으로써 적은 비용으로도 다양한 연구에 필요한 데이터를 가공할 수 있음을 보여준다.

Table 3. Used queries per experiment

Experiment	join	select	insert/ update	cursor
Software evolution	2	1	0	0
Change patterns	4	7	3	0
Code ownership	6	5	4	1

VI. Conclusions

본 논문에서는 소스코드 저장소로부터 데이터를 추출하고 데이터베이스에 저장하여 코드분석 기반의 MSR 연구에 필요한 데이터를 쿼리를 통해 제공하는 GPES를 고안하였다.

GPES는 기존의 연구들에 비해 소프트웨어의 모든 버전에 대한 소스코드의 AST 정보를 분석하여, 각 AST 구문에서 의존하는 클래스 정보와 해당 클래스의 상속관계 등을 제공한다. 또한, 각 버전에서 수정된 파일에 대해 변경정보를 소스코드의 행 단위와 AST노드 단위로 확인할 수 있기 때문에 변경의 목적을 파악하는 시간을 단축할 수 있다. 다른 강점으로, 기존의 도구들이 고정적이고 폐쇄적인 구조를 가지는 것에 비해 GPES는 유연한 데이터 모델과 컴포넌트 기반 설계를 통해 쉽게 확장 가능하다.

GPES를 구축함에 있어 개선해야 할 사항들도 존재한다. 도구 내부의 변경된 AST노드를 계산하는 과정에서 diff 결과에 따라 정확도가 다를 수 있는데, 이는 정확도가 높은 도구를 사용함으로써 해결이 가능하다. 그리고 사용자에게 최대한 상세한 정보를 주고자 하기 때문에 추출 속도가 상대적으로 느리다. 이는 GPES를 Hadoop과 같은 분산컴퓨팅 환경에서 동작할 수 있도록 구축함으로써 극복할 수 있는 문제이다. 분산컴퓨팅으로의 확장을 위해서는 기존의 추출과정을 MapReduce 기반으로 변경하고, 데이터 전송을 위해서 Sqoop 등의 Hadoop Echo 시스템을 사용해야 할 것이다.

향후에는 GitHub로부터 버그 또는 기능추가와 같은 이슈관리 정보 등을 추출해보고 GPES의 소스코드 정보와 맵핑함으로써 더 정교한 연구를 진행할 수 있을 것이다. 또한 분산처리 기반으로 단 시간 내에 많은 저장소 정보를 추출하고, Google BigQuery 등을 통해 데이터를 공유함으로써 더 많은 사람들에게 양질의 데이터를 제공하기 위한 시도를 계속할 것이다.

REFERENCES

[1] S. Oh, "A Study on the Efficient Configuration Thread Control Modeling in Version Control using Object Oriented System," Journal of the Korea Society of Computer and Information, Vol. 10, No. 4, pp. 123-132, Sep. 2005.

[2] GitHubArchive Event Dataset on Google BigQuery, https://www.githubarchive.org/

[3] 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, June 2005.

[4] C. Görg, and P. Weißgerber, “Error Detection by Refactoring Reconstruction,” ACM SIGSOFT Software Engineering Notes, Vol. 30, No. 4, pp. 1-5, July 2005.

[5] C. Teyton, M. Palyart, J. R. Falleri, F. Morandat, and X. Blanc, “Automatic Extraction of Developer Expertise,” in Proceedings of the 18th ACM International Conference on Evaluation and Assessment in Software Engineering, May 2014.

[6] S. Oh and C. Park, "Development of Automatic Rule Extraction Method in Data Mining : An Approach based on Hierarchical Clustering Algorithm and Rough Set Theory," Journal of the Korea Society of Computer and Information, Vol. 14, No. 6, pp. 135-142, June 2009.

[7] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, “Sourcerer: Mining and Searching Internet-Scale Software Repositories,” Data Mining and Knowledge Discovery, Vol. 18, No. 2, pp. 300-336, April 2009.

[8] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, “Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories,” in Proceedings of the 35th ACM/IEEE International Conference on Software Engineering, pp. 422-431, May 2013.

[9] Dyer, R., H. A. Nguyen, H. Rajan, and T. N. Nguyen, “Boa: Ultra-Large-Scale Software Repository and Source-Code Mining,” ACM Transactions on Software Engineering and Methodology, Vol. 25, No. 1, pp. 178-212, Dec. 2015.

[10] A. K. Schneider, C. Gutwin, R. Penner, and D. Paquette, “Mining a Software Developer’s Local Interaction History,” in Proceedings of the 1st International Workshop on Mining Software Repositories, pp. 106-110, May 2004.

[11] R. Robbes, “Mining a Change-Based Software Repository,” in Proceedings of the 4th International Workshop on Mining Software Repositories, 2007.

[12] J. Bevan, E. J. Whitehead, Jr., S. Kim, and M. Godfrey, “Facilitating Software Evolution Research with Kenyon,” ACM SIGSOFT Software Engineering Notes, Vol. 30, No. 5, pp. 177-186, Sep. 2005.

[13] W. Shang, B. Adams, and A. E. Hassan, “An Experience Report on Scaling Tools for Mining Software Repositories using MapReduce,” in Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering, pp. 275-284, Sept. 2010.

[14] A. Mockus, and L. G. Votta, “Identifying Reasons for Software Changes using Historic Databases,” in Proceedings of the 16th IEEE International Conference on Software Maintenance, pp. 120-130, Aug. 2000.

[15] F. Van Rysselberghe, and S. Demeyer, “Studying Software Evolution Information by Visualizing the Change History,” in Proceedings of the 20th IEEE International Conference on Software Maintenance, pp. 328-337, Sept. 2004.

[16] A. Capiluppi, M. Morisio, and J. F. Ramil, “Structural evolution of an open source system: A case study,” in Proceedings of the 12th IEEE International Workshop on Program Comprehension, pp. 172-182, June 2004.

[17] H. Gall, K. Hajek, and M. Jazayeri, “Detection of Logical Coupling Based on Product Release History,” in Proceedings of the 14th IEEE International Conference on Software Maintenance, pp. 190-198, March 1998.

[18] H. Gall, M. Jazayeri, and J. Krajewski, “CVS Release History Data for Detecting Logical Couplings,” in Proceedings of the 6th IEEE International Workshop on Principles of Software Evolution, pp. 13-23, Sept. 2003.

[19] D. Beyer, and A. Noack, “Clustering Software Artifacts Based on Frequent Common Changes,” in Proceedings of the 13th IEEE International Workshop on Program Comprehension, pp. 259-268, May 2005.

[20] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, “How Developers Drive Software Evolution,” in Proceedings of the 8th IEEE International Workshop on Principles of Software Evolution, pp. 113-122, Sept. 2005.

[21] J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona, “Effort Estimation by Characterizing Developer Activity,” in Proceedings of the 8th ACM International Workshop on Economics Driven Software Engineering Research, pp. 3-6, May 2006.

[22] R. Sindhgatta, “Identifying Domain Expertise of Developers from Source Code,” in Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 981-989, Aug. 2008.

[23] D. Schuler, and T. Zimmermann, “Mining Usage Expertise from Version Archives,” in Proceedings of the 5th ACM International Working Conference on Mining Software Repositories, pp. 121-124, May 2008.

Authors

Yong-hyeon Lee received his B.S.E. degree in Computer Engineering from Chung- buk National University, Korea, in 2014.

He is an M.Eng. student at the Dept. of Computer Engineering, Chungbuk National University. His research interests include software evolution and source code analysis.

Kisub Kim received his B.S.E. degree in Computer Engineering from Chungbuk National University, Korea, in 2014.

He was a student researcher at Natural Language Processing Lab in Chungbuk National University from 2012 to 2014. He was a developer in Kyunghee University Medical Center from 2014 to 2015. He is currently an M.Eng. student at the Dept. of Computer Engineering, Chungbuk National University. His research interests include mining software repositories, code search, and software evolution.

Jaekwon Lee received his B.S.E. and M.Eng. degrees in Computer Engineering from Chungbuk National University, Korea, in 2013 and 2015, respectively.

Currently, he is a Ph.D Student in the Department of Computer Engineering, Chungbuk National University. His research interests include mining software repositories and search-based software engineering.

Woosung Jung received his B.S. and Ph.D. degree in Computer Science and Engineering from Seoul National University, Korea, in 2003 and 2011, respectively.

He was a researcher in SK UBCare from 1998 to 2002. He was a senior research engineer at Software Capability Development Center in LG Electronics from 2011 to 2012. He is currently an associate professor at the Dept. of Computer Engineering, Chungbuk National University. His research interests include software evolution, software architecture, adaptive software system and mining software repositories.