2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement

Service Oriented Framework for Mining Software Repository

Shinsuke Matsumoto and Masahide Nakamura
Graduate School of System Informatics, Kobe University, Japan
Email: shinsuke@cs.kobe-u.ac.jp

***Abstract*—Mining software repository is one of important topic in empirical software engineering. A wide variety of mining tools are published on the Web and we can easily apply individual mining approaches. However, there is no supporting system for sharing the mining techniques, procedures, knowledge and know-how. This sharing problem also poses great difficulties for independent validation and experimental replication from mining researchers. The goal of this paper is to provide a framework that supports sharing the repository mining techniques for reducing mining effort and external validation of analysis results. This paper proposes Service Oriented Framework for Mining Software Repository (SO-MSR) which applied Service Oriented Architecture (SOA) to the repository mining. Following the SO-MSR, we also develop Metrics Web API which is a prototype system for metrics measurement. Metrics Web API can measure a variety of source code metrics without relying on any types of repositories and programming languages. The proposed system is designed and implemented as a Web service and demonstrated using actual software repository.**

***Keywords*-mining software repository; service oriented framework; version control system; source code metrics; SO-MSR; Metrics Web API**

I. Introduction

Mining software repository which empirically analyzes a development history has been widely used for improving and/or managing a software development process. Generally, a changing history of software products is stored in revision control system (RCS) repository. Likewise a bug detections history is stored in bug tracking system (BTS) repository. These historical data stored in software repositories are analyzed by wide variety of data mining techniques. The mining result are used for progress management of development, posterior analysis (e.g., causal analysis of project failure), and finding a novel knowledge. In software engineering research, a lot of mining studies have been reported[1][2][3][4][5][6][7]. The repository mining is becoming one of important topic in empirical software engineering.

One of the challenges of the repository mining is lack of system or framework to sharing mining techniques and knowledge. Therefore, we need some efforts and technical knowledge for conducting the mining. For example, when conduct a simple mining such as visualizing a change of SLOC, at least the following three steps are required: 1. Get a target source code from RCS repository, 2. Measure SLOC, 3. Visualize a graph. Researchers and developers need to use published tools (or him/herself own scripts) for each steps. However there is no compatibility and interoperability of mining tools. Though wide variety of mining tools have been published on the web, it is difficult to share techniques of data compatibility and combination of the tools.

The sharing problem also poses great difficulties for independent validation and experimental replication of the mining. For instance, a value of SLOC (source lines of code) that represents a size of software program varies with the following definitions: blanks and comments, logical or physical, a single statement written in multiple lines. Open source software (OSS) repositories are popular data source for repository mining. Even though researchers analyze same data source, the results might be different by data preprocessing, procedures and parameters of mining techniques. Some researchers pointed out[2][3] that measured metric value varies depending on each mining tool. To tackle the problem, a framework for sharing the information of *How, which tools and what parameters is used in what steps* is required.

The goal of this paper is to provide a framework that supports sharing repository mining techniques for reducing mining effort and external validation of analysis results. The key idea is applying the *service oriented architecture (SOA)* to repository mining. SOA provides loose-coupling and reusability of software systems by regarding a software system as a composed of services. The service is a unit of software functions which developers make accessible over a network. Wrapping mining tools and techniques by Web services achieves reusability and interoperability of its services. Additionally, registering and sharing a sequence of service API usages as a single service enables an independent validation and repetition of the mining procedure.

This paper proposes *Service Oriented Framework for Mining Software Repository (SO-MSR)* which applied *Service Oriented Architecture (SOA)* to repository mining. Following the SO-MSR, we also develop *Metrics Web API* which is a prototype Web service for metrics measurement. Metrics Web API can measure a variety of source code metrics without relying on any types of repositories and programming languages.

978-0-7695-4565-3/11 $26.00 © 2011 IEEE
DOI 10.1109/IWSM-MENSURA.2011.28

II. PRELIMINARIES

A. Software Development Repository

Software development repository is an online storage which contains products and development histories recorded by revision control system (RCS) and bug tracking system (BTS). Current version system (CVS) and Subversion are traditionally used as a RCS system. Recently, some distributed RCS (e.g., Git, Bazaar and Mercurial) have been published. As a BTS, Bugzilla and Redmine are well-known. Furthermore, email history is considered as one of the development repository which contains a history of communication and collaboration between developers. Each of the repositories has different data storing structure and format.

SourceForge[1], a well-known web-based software development management system, provides wide variety of development repositories for OSS development projects. An OSS project developed in SourceForge opens not only its product (e.g., source code) but also its development process (e.g., change of source code, communication history between developers and bug fixing process). Therefore, many researchers have been conduct mining studies using repository data stored in SourceForge.

B. Repository Mining

Repository mining is an approach to managing and improving software development process by analyzing development repository with applying data mining techniques. The purpose of the mining is to share experience, intuition, know-how and finding novel knowledge in software development. Then each mining result is described as formal knowledge. Traditionally, there are various mining techniques, such as measuring a software metrics[4][5], statistical analysis using the software metrics[6], feature prediction[7], association mining[8] and so on. Nowadays human-oriented analyses have been studied which analyzes a developer's communication and collaboration[9] and source code ownerships[10].

C. Challenges in Repository Mining

There are some challenges in repository mining for sharing the mining techniques.

P1: Lack of standardized access methods to the repositories

Although variety kinds of RCS and BTS have been developed, each repository requires different client tools and different mining approaches. For example, CVS repository and Subversion repository have different version numbering rules and different file units, although both repositories contain history of source code development. As shown in Figure 1, log formats have quite different. Different calculation logic or tool is required when measuring a log-based metric

[1]http://SourceForge.net/

```
----------------------------
revision 1.4
date: 2009/09/12 00:29:40;  author: shinsuke;  state: Exp;  lines: +16 -0
added for dependency metrics.
----------------------------
revision 1.3
date: 2009/09/13 00:24:15;  author: shinsuke;  state: Exp;  lines: +77 -31
incorrect results of dependency metrics.
----------------------------
...
```

(a) CVS log format

```
------------------------------------------------------------------------
r16 | shinsuke | 2009-01-13 00:40:13 +0900 (Tue, 13 Jan 2009) | 13 line

added persistence extension
------------------------------------------------------------------------
r14 | shinsuke | 2009-01-07 23:43:33 +0900 (Wed, 07 Jan 2009) | 3 lines

updated javadocs
------------------------------------------------------------------------
...
```

(b) Subversion log format

Figure 1. Difference of commit log format between CVS and Subversion

(i.e., change metrics[11]). Thus, researchers and developers need to understand about details of each repository even if they apply a same mining technique to different repositories.

P2: Lack of interoperability among mining tools

There is no data compatibility and interoperability among published mining tools. When a user measures a metric using tool A and applies a statistical analysis using tool B, he/she need to ensure data conversion between tool A and B. In general, analyzing between source code metrics stored in RCS repository and bug fixing process stored in BTS has been studied in reliability researches. Data compatibility and interoperability must be supported to analyze across multiple repositories.

P3: Lack of a system for sharing mining techniques and procedures

We can easily apply *individual* mining techniques (e.g., measuring metrics, statistical analysis, clone detection and so on) without expert knowledge because a lot of mining tools have been published on the Web. However there is no supporting system to sharing mining procedures and techniques of the tool combinations. Currently, individual researchers and developers uniquely have their own techniques and know-how without any sharing.

From a research perspective, this sharing problem also poses great difficulties for independent validation and experimental replication of the mining. Some researchers pointed out[2][3] that measuring results varies depending on each mining tool. To provide the validation and reputation sharing the information of "How, which tools and what parameters is used in what steps." is required. However, writing the detailed analysis conditions in limited paper space is unrealistic approach.

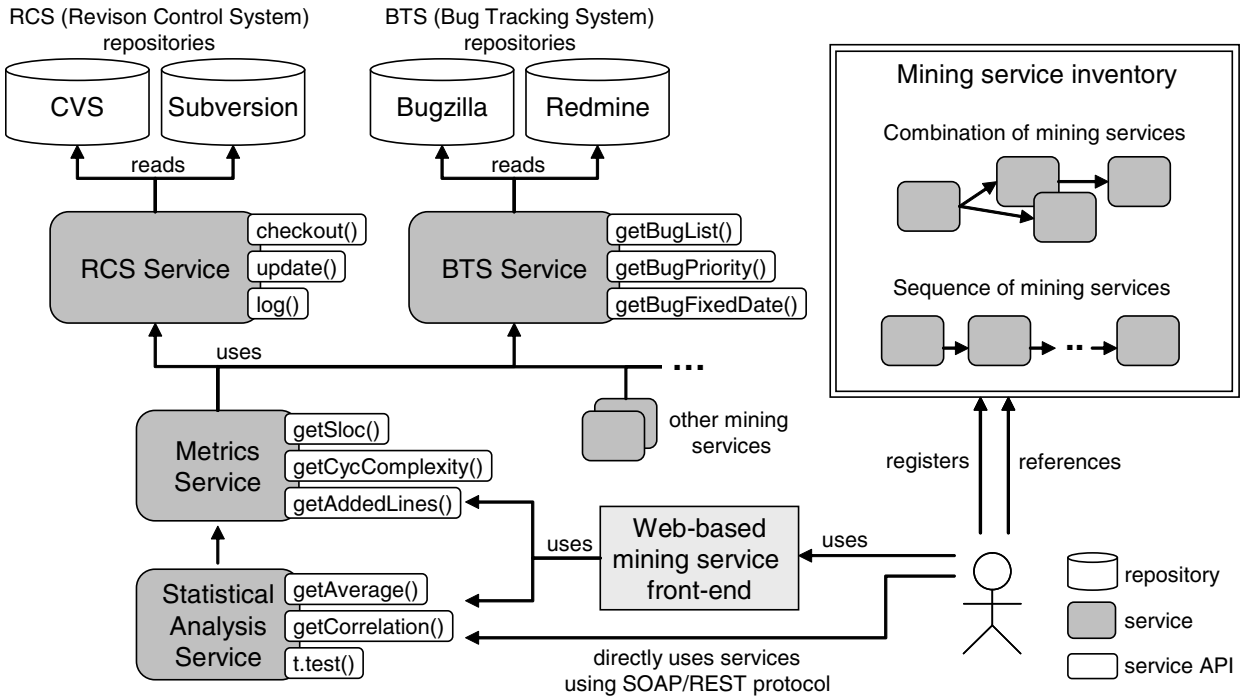

Figure 2. Architecture of SO-MSR (Service Oriented framework for Mining Software Repository)

III. Framework for Mining Software Repository

A. Requirements

In this paper, we consider that the framework for sharing the mining techniques needs to fulfill the following requirements.

RQ1: Supporting repository-independent access method

To solve the problem P1, the framework need to provide repository-independent access method. For example, when accessing to a RCS repository, the following common RCS commands should be standardized: `checkout()` which obtains stored source codes, `update()` which updates to new version, `log()` which obtains a commit log. Additionally, common information written in RCS log message such as *revision number, commit developer, date and number of edited lines* (shown in Figure 1) also should be represented as same data format.

RQ2: Supporting abstract mining techniques

To tackle the problem P2, the framework need to abstract and encapsulate mining techniques, data format and data processing logic in the system. This approach makes users to conduct a mining by meaningful method calls. For example, `getSloc(http://path_to_repository)` and `getCyclomaticComplexity(http://path_to_repository)` should achieve that user can measure each metric without regard to repository access procedure, measurement logic and data compatibility.

RQ3: Sharing mining procedures

For the problem P3, a mechanism for sharing mining techniques, procedures and used parameters are required. This mechanism fulfills independent validation and experimental replication in the mining research area. Furthermore, mining efforts might be reduced by the knowledge sharing system.

RQ4: Support machine-processable interface

Supporting every mining techniques by system is actually impossible. In some case, a user may conduct a mining by partially using the proposed system and partially using his/her own script. To support this usage, the system should be accessed by machine-processable interface such as XML-based protocol. The machine-processable interface achieves program invocation from a variety of programs or systems.

RQ5: Reusing existing mining tools

A mount of mining tools have been proposed and used by some researchers. By reusing the existing tools, the proposed system assures reliability for mining results.

B. Key Idea

We propose a *Service Oriented Framework for Mining Software Repository (SO-MSR)* based on *Service Oriented Architecture (SOA)* to meet the above requirements RQ1 to RQ5. The SOA is an approach to build a flexible information system. SOA considers a software system as integrated *services*. The *service* is a unit of software functions and has a coarse granularity than object and module. Each service can invoke from the internet using standardized XML-based protocol such as SOAP/REST. Constructing a

software system from a set of services allows flexibility and reusability.

Generally, SOA based system is implemented as a Web service. Web service provides interoperable machine-processable interface using web-based protocols. Wide variety and number of Web services have been published on the web. We can use these services such as weather report, daily news, schedule management and language translation through the internet.

Wrapping each repository-depended accessing method to a service enables standardized repository-independent access. This approach solves requirement RQ1. Likewise, encapsulating each repository-depended data format and data process in a service solves requirement RQ2. To solve requirement RQ3, we provide an inventory service which registers and shares a mining procedure (e.g., procedure and combination of mining service invocation). Requirement RQ4 is solved by supporting XML-based and machine-processable protocol.

C. Architecture

The architecture of SO-MSR is shown in Figure 2. Cylinder means a repository, gray-colored box means a service and white-colored box means service API.

This architecture wraps accessing method for CVS and Subversion repository as `RCS Service` which has common RCS methods (i.e., checkout(), update() and log(). Similarly, accessing procedure to BTS repositories are wrapped as `BTS Service`. Mining services use these repository accessing services. This figure illustrates two mining services which are `Metrics Service` and `Statistical Analysis Service`. User can call mining services through a Web application front-end and can directly call the services using SOAP/REST protocol. Service inventory, located in the right-upper, supports sharing a sequence and combination of service invocations. Referring the registered service invocations enables independent validation, mining replication and customizing the mining procedure.

IV. Metrics Web API

A. Overview

In this paper, we develop a prototype system for software metrics measurement system based on SO-MSR. The metrics measurement is one of famous approach in the repository mining. *Metrics Web API*, described in this section, is a Web service for source code metrics measurement which corresponds to `Metrics Service` in Figure 2. Metrics Web API can measure a variety of source code metrics without relying on any types of repositories and programming languages.

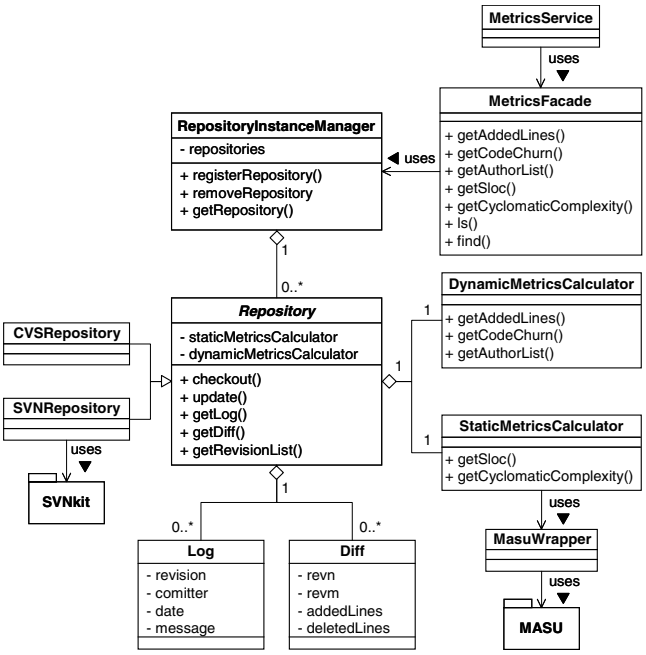

Figure 3. UML class diagram of Metrics Web API

B. Design

Figure 3 shows a UML class diagram of Metrics Web API (only the essential classes and methods are shown).

A `MetricsService`, located in the top, is a service-layered class which has all service APIs. API calls from service users are invoked through this class. A `MetricsFacade` aggregates all mining operations such as metrics calculation and file operation. Common RCS repository operations are abstracted in `Repository`. Each repository specific logics are encapsulated in `CVSRepository` and `SVNRepository` which extend the `Repository`. Some `Repository` are managed by `repositoryInstanceManager`. Additionally, `Repository` has two metrics measurement classes, `DynamicMetricsCalculator` and `StaticMetricsCalculator`. The two metrics measurement classes respectively measures history-based metrics (we call dynamic metrics) and snapshot metrics (we call static metrics).

C. Implementation

Based on the UML class diagram, we have developed Metrics Web API. Development language was Java. The total SLOC and the total number of classes were 4,245 and 19 respectively. As shown in Figure 3, the proposed system reuses SVNKit[12] as a Subversion client and reuses MASU[13] (Metrics assessment plugin platform for software unit of multiple programming languages) as a C&K metrics calculator. MASU can calculate some metrics from

multiple programming languages because MASU constructs language-independent data when applying source code analysis. Measuring logic for other metrics such as line-based metrics and dynamic metrics are developed using our original programs.

Currently implemented APIs for metrics measurement and file operation are shown in Table I. Square brackets written in "Parameters" row mean optional parameters. When a user measures a metric, he/she must use API `registerRepository()` which registers his/her repository to Metrics Web API. User ID and password are required as the API parameters because `registerRepository()` logins to the repository at the same time. The response of the `registerRepository()` is a registered ID. For supporting multiple users, Metrics Web API assumes that some repositories are registered in the system. Therefore a user must specify his/her target repository ID when calling metrics measurement APIs. APIs for metrics measurement are shown in the table. For example, when a user measure SLOC metric, the user only have to call two APIs (i.e., `registerRepository()` and `getSloc()`). The proposed system implemented file operation APIs such as `ls()` which lists files or directories in current directory, `find()` which finds specified files or directories, `getJavaFileList()` which lists source codes written in Java.

D. Case Study of Metrics Web API Usage

We demonstrate a case study of repository mining using the Metrics Web API. In this case study, we developed a program which captures a change of SLOC of a source code and visualize growth of the code. The program invokes two Web services, Metrics Web API and Google Chart API[2]. Google Chart API can create variety of graph images such line chart, pie chart, scatter chart and so on by calling simple http request.

Figure 4 shows a developed source code. Development language is Perl. To invoke these Web services, we used SOAP::Lite module and Google::Chart module respectively. The development takes about 20 minutes and the total SLOC is 39 (including blanks and comments). This code consists of two phases, the data obtain phase which uses Metrics Web API and the visualize phase which uses Google Chart API. The 7th and 8th line generate SOAP::Lite object to preparing the Metrics Web API invocation. The 16th and 30th line actually invoke the service APIs which are correspond to repository login and SLOC obtaining. Although the type of target repository is Subversion and the target source code is Java, the developed Perl code can conduct repository mining by only invoking abstracted APIs without relying on the repository type and the programming language.

[2]http://code.google.com/intl/en/apis/chart/

```
1  #!/usr/bin/perl -w
2  use strict;
3  use SOAP::Lite;
4  use Google::Chart;
5
6  # Service instantiation                          Data obtaining phase
7  my $service = SOAP::Lite->uri('http://metrics.kobe-u.jp');
8  $service->proxy('http://cs27.kobe-u.ac.jp/MetricsAPI');
9                                                   Service instantiation step
10 # Repository settings
11 my $repo = 'http://cs27.kobe-u.ac.jp/svn/metricsApi/';
12 my $user = 'shinsuke';
13 my $pass = '***';
                                  API invocation steps
15 # Register repository to MetricsAPI by invoking a SOAP API
16 my $id = $service->registerRepository($repo, $user, $pass)
17                 ->result;
18
19 # Parameters for getting SLOC
20 my $year = 2010;
21 my $month = 12;
22 my $src = 'src/jp/kobe_u/cs27/metricsAPI/main.java';
23
24 # Get SLOC list in Dec. 2010
25 my @sloc;
26 for (my $day=1; $day<30; $day+=1) {
27   my $date = sprintf("%04d-%02d-%02d", $year, $month, $day);
28
29   # Invoke SOAP API
30   my $res = $service->getSlocByDate($id, $src, $date)
31                     ->result;
32   push(@sloc, $res);
33 }
34
35 # Line plot using ChartAPI                       Visualizing phase
36 my $chart = Google::Chart->new(type=>'line', data=>[@sloc]);
37
38 # Save a png file
39 $chart->render_to_file(filename=>'growth.png');
```

Figure 4. Perl code for visualizing growth of source code using Metrics Web API and Google Chart API

V. Current Status and Future Works

Currently, Metrics Web API supports only Subversion repository. We have to support other RCS repositories such as CVS and Git. Generally, repository mining based on source code metrics are studied with bug detection information stored in BTS repository. To provide a practical system for analyst who conducts a repository mining, a service for accessing BTS repositories is required.

Separating the metrics service and RCS service is also our future work. As shown in Figure 3, Metrics Web API internally has not only metric calculation logic but also processing logic for RCS repository. However, in terms of service oriented framework, RCS service must separate from Metrics Web API as shown in Figure 2. This approach enables reusing the RCS services from other mining services such as clone detection service.

Developing a service client is also necessary for improving the service usability. As shown in Figure 4, Web service API can directly invoked from program using SOAP/REST protocol. However, to improve the usability of Web services, we must provide a service client such as interactive CUI client and/or GUI Web application client. These service clients might significantly reduce mining efforts and assist a developer's self analysis of his/her own work.

Table I
SUMMARY OF IMPELEMENTED APIS

Category	Sub categ.	API name	Parameters	Responses
Service management		registerRepository	repository url, uid, pass	Registered id
		deleteRegisteredRepository	id	—
		getRegisteredRepositoryList	—	Registered repository list
Static metrics measurement	Line-based	getSloc	id, file [, date]	Physical SLOC
		getLogicalSloc	id, file [, date]	Logical SLOC
		getCommentSloc	id, file [, date]	Comment SLOC
		getBlankSloc	id, file [, date]	Blank SLOC
	McCabe	getCyclomaticComplexity	id, file [, date]	Cyclomatic complexity
	C&K	getWmc	id, file [, date]	Weighted methods per class
		getNoc	id, file [, date]	Number of childrens
		getCbo	id, file [, date]	Coupling between object classes
		getDit	id, file [, date]	Depth of inheritance tree
		getRfc	id, file [, date]	Response for a class
		getLcom	id, file [, date]	lack of cohesion in methods
Dynamic metrics measurement	Line-based	getAddedLines	id, file [, term]	# of added lines
		getDeletedLines	id, file [, term]	# of deleted lines
		getCodeChurn	id, file [, term]	# of code churn (added + deleted)
	Commit mssage-based	getRefactoringList	id, file [, term]	List of refactorings
		getBugfixList	id, file [, term]	List of bug fixes
		getNRefactorings	id, file [, term]	# of refactorings
		getNBugfixes	id, file [, term]	# of bugfixes
	Author	getAuthorList	id, file [, term]	List of authors
		getNAuthors	id, file [, term]	# of authors
	Others	getRevisionList	id, file [, term]	List of revision
		getNRevisions	id, file [, term]	# of revisions
		getAge	id, file [, term]	Age (days)
Metrics summary		getDynamicMetricsSummary	id	Summary of dynamic metrics for all codes
		getStaticMetricsSummary	id	Summary of static metrics for all codes
File operation		ls	id, path	File list (equals ls command)
		lsr	id, path	File list (equals $ls --recursive)
		find	id, file	File list (equals find command)
		getJavaFileList	id, path	File list (equals $find . -name "*.java")
		getCFileList	id, path	File list (equals $find . -name "*.c")

VI. CONCLUSION

This paper proposes a framework SO-MSR for sharing mining software repository techniques based on SOA. Following the SO-MSR, we have also developed *Metrics Web API* which is a prototype Web service for metrics measurement. Metrics Web API has following features.

- Platform-independent access.
- Machine-processable interface.
- Repository accessing APIs are independent from repository type.
- Metrics calculation APIs are independent from programming language.
- User can conduct mining without regard to internal logic and data conversion.

We have some future works for providing a practical system because Metrics Web API is at experimental stage. The future works include supporting a variety of repositories and developing a service client. Additionally, we are planning an experimental evaluation of Metrics Web API to verify the system usability and validity of the framework.

ACKNOWLEDGMENT

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (B) (No.23300009), Young Scientists (B) (No.21700077), Research Activity Start-up (No.22800042)], and Hyogo Science and Technology Association.

REFERENCES

[1] A. G. Koru, D. Zhang, K. E. Emam, and H. Liu, "An investigation into the functional form of the size-defect relationship for software modules," *IEEE Transaction on Software Engineering*, vol. 35, no. 2, pp. 293–304, 2009.

[2] L. H. Etzkorn, S. E. Gholston, J. L. Fortune, C. E. Stein, D. Utley, P. A. Farrington, and G. W. Cox, "A comparison of cohesion metrics for object-oriented systems," *Information and Software Technology*, vol. 46, no. 10, pp. 677–687, 2004.

[3] R. Lincke, J. Lundberg, and W. Lowe, "Comparison software metrics tools," in *Proc. Int'l Symposium on Software Testing and Analysis*, 2008, pp. 131–141.

[4] S. G. Crawford, A. A. McIntosh, and D. Pregibon, "An analysis of static metrics and faults in C software," *Journal of System Software*, vol. 5, no. 1, pp. 37–48, 1985.

[5] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proc. Int'l Conference on Software Engineering (ICSE '08)*, 2008, pp. 531–540.

[6] P. Bourque and V. Cote, "An experiment in software sizing with structured analysis metrics," *Journal of System Software*, vol. 15, no. 2, pp. 159–172, 1991.

[7] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, pp. 897–910, 2005.

[8] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proc. Int'l Conference on Software Engineering (ICSE '04)*, 2004, pp. 563–572.

[9] A. Meneely and L. Williams, "Socio-technical developer networks: Should we trust our measurements?" in *Proc. Int'l Conference on Software Engineering (ICSE '11)*, 2011, pp. 281–290.

[10] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transaction on Software Engineering Methodology*, vol. 11, pp. 309–346, 2002.

[11] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. Int'l Conference on Software Engineering (ICSE '08)*, 2008, pp. 181–190.

[12] SVNKit, http://svnkit.com.

[13] T. Miyake, Y. Higo, S. Kusumoto, and K. Inoue, "Masu: A metrics measurement framework for multiple programing language," *Journal of IEICE*, vol. J92-D, pp. 1518–1531, 2009 (in Japanese).