2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing

Visualizing Software Metrics with Service-Oriented Mining Software Repository for Reviewing Personal Process

Yasutaka Sakamoto, Shinsuke Matsumoto, Sachio Saiki and Masahide Nakamura
Graduate School of System Informatics, Kobe University
1-1 Rokkodai, Nada, Kobe, Hyogo 657-8501, Japan
Email: gen@ws.cs.kobe-u.ac.jp, {shinsuke, masa-n}@cs.kobe-u.ac.jp, sachio@carp.kobe-u.ac.jp

***Abstract*—We have proposed a framework named SO-MSR: service-oriented mining software repository, which applied service oriented architecture to MSR. Following the SO-MSR, we have developed a web service, named MetricsWebAPI, for metrics calculation from a variety of software repositories and a variety source codes. In this paper, we develop and propose MetricsViewer, which is client of MetricsViewer and is a web application to support personal process improvement. MetricsViewer provides an interactive user interface for repository file exploring. Moreover the MetricsViewer visualizes change of source code metrics to support overhead view of personal process. End user can improve their development activities based on software repository data without MSR specific knowledge by using MetricsViewer. We have conducted a pilot study to evaluate the effect of proposed system for personal process improvement.**

***Keywords*-MSR, Service-Oriented Architecture, Software Metrics, Visualization, SO-MSR, MetricsViewer**

I. Introduction

Mining Software Repository (MSR) is a research field where developers analyzes a software repository by using some data mining techniques. By conducting the MSR, developers can conduct empirical and evidence-based process improvement.

In our previous work[1], we have previously proposed a MSR framework, "Service-Oriented Framework for MSR (SO-MSR)", which applied the concept of SOA (Service-Oriented Architecture) to MSR. In the SO-MSR, various MSR procedures and techniques are wrapped as an abstracted service. End user can easily get the mining results of their own software repository without some mining specific knowledge. Furthermore, the concept of SOA allows service composition. MSR practitioners can easily extend the MSR techniques by combining the MSR services and other third-party services.

We have also proposed and developed a concrete Web service[2] named MetricsWebAPI which follows SO-MSR[1]. MetricsWebAPI provides measurement results of some variety of source code metrics from a user's software repository. The system wraps detailed mining techniques such as building an AST (Abstract Syntax Tree) and natural language processing to commit logs. Also the system abstracts the difference of software repositories (SVN, CVS and Git) and the difference of programming language. End user can get some variety of source code metrics without regard to the differences by specifying a minimum requirement to conduct the measurement.

The next challenge of MetricsWebAPI is to develop and provide a GUI client. Web service can be accessed from any programming languages and from any platforms because it provides XML-based web APIs. Hence each service easily combine with other services, web service has a strong advantage of its expandability. However these advantages are just for developers. User-friendly clients such as GUI client are necessary for end users to provide interactive and intuitive MSR.

There is wide variety of purposes of MSR techniques. MSR clients also vary for the MSR purposes. For example, a client which visualizes evolution of software may support self process improvement by looking back their development activities. Other examples are: a client to visualize the development relationship between developers and source codes; a client to recommend refactoring; a client to identify system vulnerabilities based on its modification history.

In this study, we focus on supporting personal process improvement as a application of MSR. We propose a Web application, named "MetricsViewer", which provides interactive file exploring in a software repository and visualizes change of some software metrics. MetricsViewer helps developers to have a bird's-eye view of their development activity without MSR specific knowledge. The system is composed by MetricsWebAPI and Google Chart API. Google Chart API is a Web service for generating graph objects provided by Google with minimum plotting parameters. We also conducted an preliminary experiment to show the effect of the system. In the experiment, four subjects used the proposed system and evaluated the system in terms of whether the system is useful to personal software improvement.

II. Preliminaries

A. Mining Software Repository (MSR)

MSR is to help software development with the technique of data mininig by analyzing development historical data from software reporitories such as version control system and bug tracking system and so on. Various studies have been repoted in the MSR research field[3], [4], [5].

978-0-7695-5005-3/13 $26.00 © 2013 IEEE
DOI 10.1109/SNPD.2013.96

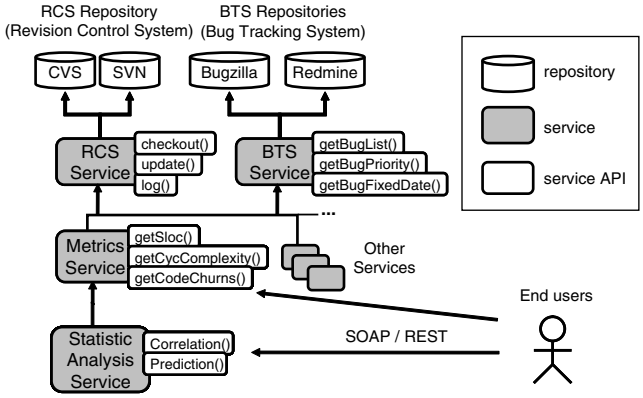

Fig. 1. Architecture of SO-MSR

Source code metric is one of the well-known and widely-studied approach in the MSR field[6], [7]. The metric is calculated by source code and/or its change history. Developer or researcher can conduct a typical data mining analysis by using some software metrics. As one of typical practical uses of the metric, there are the number of code, complexity, C&K metrics of source code and the contribution of developers and so on.

B. Service-Oriented MSR (SO-MSR)

SO-MSR is a MSR framework which applied concept of Service-Oriented Architecture (SOA). Figure 1 shows an architecture of SO-MSR. In this figure, CVS and SVN are illustrated as repositories of version control system. The difference of accessing methods between the reporitories is wrapped by RCS Service. RCS Service provides abstracted access APIs which commonly used in revision control systems such as `checkout()` and `log()`. Repositories of bug tracking system are also wrapped by BTS Service. BTS Service supports some APIs related to bug tracking system such as `getBugList()` and `getBugFixedDate()`. By integrating these two services, a service for metric measurement, named Metrics Service, can be provided. In the same manner, Static Analysis Service, which supports statistical analysis, is developed by using Metrics Service. As shown in this figure, SO-MSR have an advantage for expandability of MSR techniques.

C. MetricsWebAPI

As one of the concrete service of SO-MSR, we have developed MetricsWebAPI. MetricsWebAPI is a web service which measures variety of software metrics. Three types of measurable software metrics are shown as below.

- Code metrics: (e.g., source lines of code, cyclomatic complexity and C&K metrics)
- Change metrics: (e.g., number of code churn and number of revisions)
- Developer metrics: (e.g., number of developers and a list of developers)

```
$curl http://metrics.web.api/registerReposi
tory?repo=http://path.to.repo/

  <id>1</id>
```

(a) `registerRepository()` API

```
$curl http://metrics.web.api/getSlocs?repo=
1&src=PlayHNS.java

  <getSlocs>
    <revision>
      <revid>690</revid>
      <date>2010-08-27</date>
      <author>gen</author>
      <sloc>100</sloc>
    </revision>
    <revision>
      <revid>698</revid>
      <date>2010-08-30</date>
      <author>masa-n</author>
      <sloc>112</sloc>
    </revision>
    ...
  </getSlocs>
```

(b) `getSlocs()` API

Fig. 2. Example of call and response of MetricsWebAPI using cURL

Figure 2 represents two examples of usage result of MetricsWebAPI using cURL command. In the Figure 2(a), a developer's repository is registered to MetricsWebAPI by using `registerRepository()` API and the system returns repository id "1". Figure 2(b) calls `getSlocs()` API with the registered repository id "1" and target source code name "PlayHNS.java". As a result, the system returns measurement results with XML format. This xml represents that the "PlayHNS.java" revised at revision id "690", "698" and so on. The first revision was commited by developer "gen" at 27th Augst 2010 and its total lines of code was 100. As described in this figure, MetricsWebAPI returns measurement results for each revision with meta information of each revision.

In this way, XML-based protocol (SOAP/REST) is used as a service interface protocol. So we can easily cooperate with other services and expect to extend to various service like a bug prediction service with BTS service.

D. Challenge

One of the problems of MetricsWebAPI is to develop a client for end-users which has the interactive graphical interface. As you can see in Figure 2, XML-based protocol is used as a communication language between the machines. So it is difficult for end-users to understand what the service response means and the XML protocol is not assumed that directly used by humans. In order to support end-users or software developers use MetricsWebAPI without MSR specific

knowledge, it is necessary to develop a client which has an graphical user interfacce and visualization features.

This client can be provided for various software metrics usages and its purposes. First, from the perspective of process improvement, there is an example of a clinet which visualizes changes of some software metrics collected by MetricsWebAPI. This client helps users to intuitively and graphically look back their development activities. Other examples include a client for visualizing a relationship between developers and source codes. This client supports understand a developer who has specific knowledge for a source code that should be fixed. The client also used in bug assignment process. Moreover, it is thought that we can provide a client to help software product improvement. The client identifies a buggy or less-vulnerability code and finds a code that should be refactored.

III. MetricsViewer

A. overview

In this paper, we focus on supporting self process improvement as one of the some clients of MetricsWebAPI. So, we develop a GUI client named **MetricsViewer**. MetricsViewer is a web application for visualizing software evolution represented by source code metric and process metric. This system is implemented as a mashup application combinating MetricsWebAPI and Google Chart Tools. MetricsViewer just requires an internet browser and network connection. User can easily look back their own development activities from any computer environment.

B. Requirements

When we developed the proposed sytem, we difined two requirements for our own review based on source code metrics.

- **Requirement 1 (R1):** MetricsViewer provides interactive operation to search a file in a repository easily.
- **Requirement 2 (R2):** MetricsViewer shows the visualizing result of metrics and other information which help our own review about a selected file.

As a lot of source codes and documents are stored in a repository, finding a soource code in the repository is not easy. When a user reviews himself or herself, a interactive file exolorer function like R1 is necessary. And we consider four kinds of metrics as visualized metrics in R2 as belows.

- Meta information in recent revision (date, revision ID, committer, comment)
- A history list for each revision
- A growth graph of source code metrics value
- A percentage of commitments users make

It is generaly thought that when you search your problems and improvemnts, you search extraordinary or abnormal points about them and confirm their detail later. To help this series of reviews, first, it is nessary to visualize metrics like the help of a bird's-eye view of the entire action. The four kinds of metrics are the easy information which user can understand intuitivily in the information about development activities. We try to help to review ourselves by showing these informations on a screen.

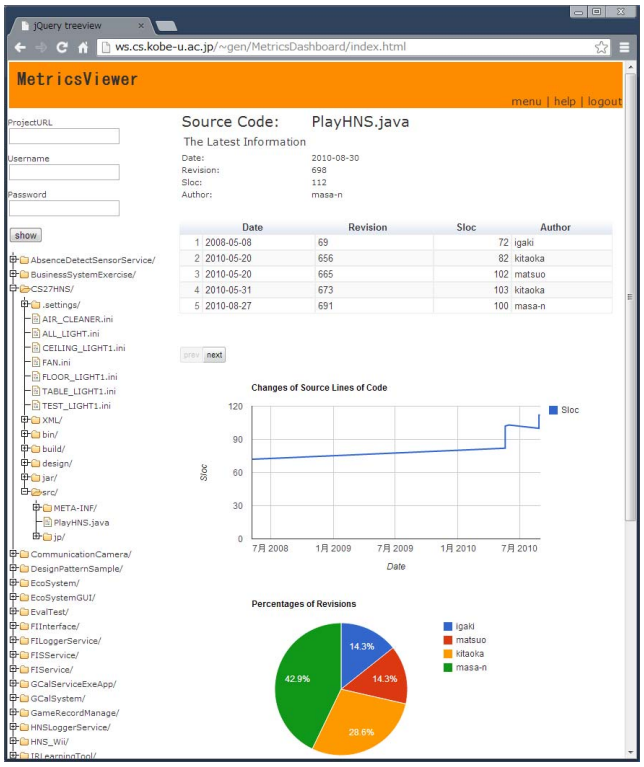

Fig. 3. Screenshot of MetricsViewer

In particular, a user can confirm the growth process of source code with a growth graph and the detail of each revision from the history information.

C. Features

We show the screenshot of MetricsViewer in Figure 3.

The left side of this figure shows a feature for searching a file in a repository. This feature meets requirement R1. The right side of the figure represents a feature of requirement R2 which visualizes measurement results and other repository information. We explain the five main features of MetricsViewer after this section.

1) Package Explorer: Package Explorer provides a feature to search a file in a specified repository. When a user selects a repository, MetricsViewer retrieves a all source code list in the repository automatically and generates explorer view. If a user selects a file from this explorer, the user can use the visualization function of following sections interactively. When a user wants to register a new repository, the user only has to input a URL, user ID and password of the repository throw the left upper text fields of MetricsViewer. This feature uses two APIs, `lsr()` and `registerRepository()`, provided by MetricsWebAPI.

2) Latest Information: Latest Information shows meta information in the most recent revision (revision ID, commit date, number of lines and committer) about a selected file. This function enables users to understand the most recent status

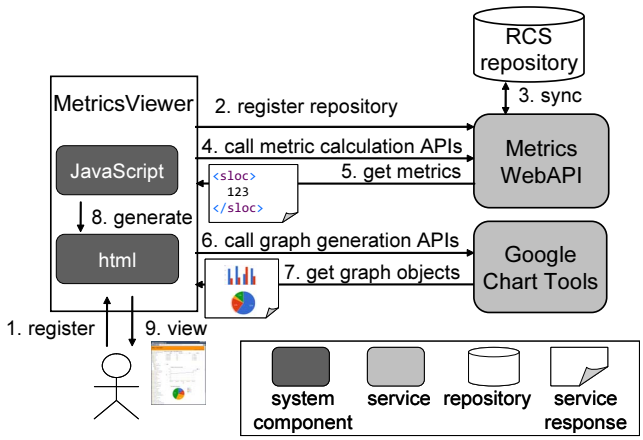

Fig. 4. Architecture and process flow of MetricsViewer

of the file. `getLatestRevision()` and `getSloc()` are used to get meta information.

3) Revision History: Revision History shows a history for each revision as a table format. A user can understand detailed information which is difficult to read by a visualized graph. `getRevisions()` is used in this feature.

4) SLOC Line Chart: SLOC Line Chart visualizes the growth of a source lines of code about a selected file as a line chart. A vertical axis shows number of lines and horizontal axis shows dates. A user can understand how the selected code growths day by day.. `getSlocs()` is used to get metric of source lines of code. This feature will be extended to generate a graph with other process metrics.

5) Contributor Pie Chart: Contributor Pie Chart shows the degree of contribution of a user which edits addtionaly and deletes a selected file by using a pie chart. This degree of contribution is calculated from the percentage of number of committments. A user can show a degree of cotributions about collaborative developers. `getContributors()` is used as API.

D. Process Flow of Visualization

The visualization process flow of MetricsViewer is shown as follows.

Step1 : A user registers his/her repository through MetricsViewer's html page.
Step2 : JavaScript component invokes repository registering API.
Step3 : MetricsWebAPI synchronizes the specified RCS repository.
Step4 : JavaScript component invokes metrics measurement APIs.
Step5 : MetricsWebAPI returns calculated metrics.
Step6 : JavaScript component invokes graph generation APIs provided by Google Chart Tools.
Step7 : Chart Tools returns graph objects.
Step8 : JavaScript component generates visualization html page.

Question 1. Please evaluate three points as below.
| 4: very satisfied | 3: satisfied | 2: dissatisfied | 1: very dissatisfied |

- usability [4 3 2 1]
- visibility of information [4 3 2 1]
- implementation time [4 3 2 1]

Question 2. Please describe "self-discovery" which is obtained in your writing action.
Question 3. Please describe "improvement and reflection" about your writing action.
Question 4. Please describe "complaint and improvement" about the system.
Question 5. Please describe something else you found.

Fig. 5. Questionnaire

Step9 : Generated html is visualized on the browser.

E. Implementation

MetricsViewer is implemented by html, css and Javascript. jQuery is used in main procedure. Development effort is about two man months. Number of program steps is four hundred thirty.

IV. Experiment

A. Overview

As a preliminary experiment that ascertain the efficacy of MetricsViewer for the own review, the proposed system used by subjects. For a theme of preliminary experiment, writing action of the workshop manuscript is chosen instead of the software development action because of the lack of enough history data and subjects. However, from the evaluation of proposed method point of viewing, flexible theme can be acceptted for the experiment and the writing action, exactly growth of a tex file, is enough too.

The subjects are four graduate school students belong to master's course. The procedure of the expriment is as follows. First, subjects visualized the first own tex file of the manuscript by using the proposed system. Then, the most recent subject's tex file is visualized too. Finally, they fill in the questionnaire about review of own writing with a comparison of visualization results.

Figure 5 shows actual questionnaries used in the experiment. The questionnarie includes quantitative evaluation such as usability, visibility of the information and implementation time, and questions about self-review and self-discovery, and dissatisfied point of the sysytem.

B. Result

We show the result of the questionnaire in Figure 6 and of visualization about recent writing action of a subject in Figure 7 as a example of the visualizing results. From Question 2 about self-discovery, most subjects answered, "I made final push hard before the deadline of the writing.". And about the question about his own improvement and reflection points, the opinions such as "I should plan the writing schedule to write my manuscript in good time." and "I found that I was lenient myself." were obtained. About complains of the system, opinions like "I want to compare some files at the same time." and "I want to review my action on not a per file

Question 1: four-grade evaluation
- usability 3.5 point on average
- visibility of information 3.3 point on average
- implementation time 2.3 point on average

Question 2: self-discovery
- Recent writing action is hard about final push before the deadline.
- I remembered the help of my teacher.
- I wrote the first manuscript in plenty of time.
- I found that I roughly finished writing the manuscript and after that, I edited the detail part of it

Question 3: self-improvement and reflection
- I should make a plan to write in plenty of time.
- When I reviewed recent writing action, I found that I was lenient with it even if it was before the deadline.
- I should have committed frequently in the viewpoint of the review. If the idea and trouble at that time are recorded correctly, they are useful

Question 4: complaint and improvement of the system
- I want to visualize some files at the same time.
- I want to visualize my writing action on a per person, not a per file.
- I want to add the function to compare with not only the past review but also the schedule.
- I want to show the comment of each revision.

Question 5 : other opinion
- I was interested because my writing action was visualized intuitively.

Fig. 6. Result of Questionnaire

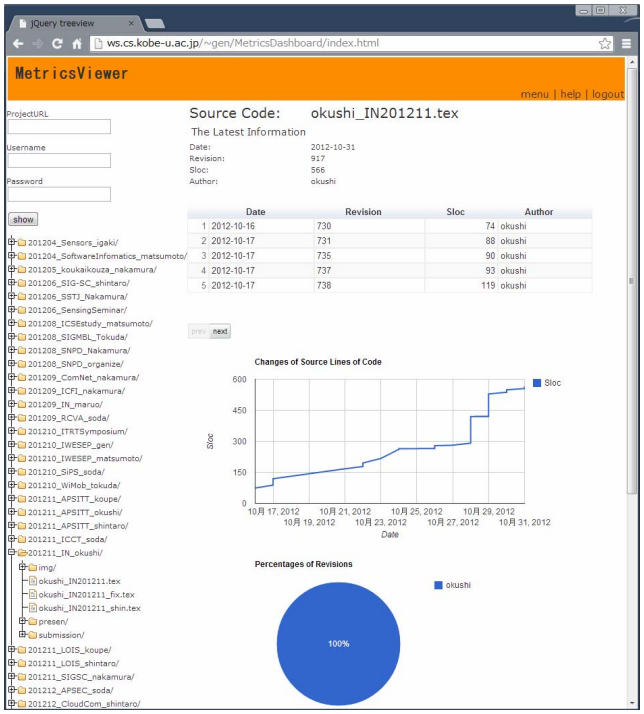

Fig. 7. Visualization Result Conducted by a Subject

but a per person." were obtained. From other opinions, the opinion that the review itself is interesting was obtained.

Figure 6 and Figure 7 shows results of the questionnaire and one example of visualization on recent writing action of a subject. From Question 2 about self-discovery, most subjects answered, "I made final push hard before the deadline of the writing.". About the question about own improvement and reflection points, opinions such as "I should plan the writing schedule to improve my manuscript." and "I found that I was lenient myself." were obtained. For dissatisfied points of the system, "I want to compare some files at the same time." and "I want to review my action on not a per file but a per person." were enumerated. As for other opinions, "the act of self review is very interesting" was obtained.

From other opinions, the opinion that the review itself is interesting was obtained.

V. Discussion

A. Usability

From quantitative question items, the usability was 3.5 point on average and the visibility was 3.3 point on average and the evaluation of them was mostly high. And from other opinions the opinion that the act of self-review was interesting was obtained. Moreover from qualitative questionnaire, there were some self-discoveries, improvements and reflections in fact. consequently it is thought that the usability of the proposed system could be verified.

Additonally, most subjects answered that the recent writing action was harder about the final push before the deadline. Especially about the subject whose result we showed in Figure 7, he answerd that if it was plenty of time until the deadline, he was lenient about the writing action. From the view of the work of a per laboratory, he should make a schedule in advance to write his manuscript in plenty of time. In this way, it is thought that the proposed system contributes not only self-review but also review of a per team.

B. Limitations

About the evaluation in Question 1, there were a lot of complains about implementation time. This is because the update procedure of a repository and calculation procedure are implemented every time API of calculating metrics is call and the result of procedure is never reused. In this point, it is considered that improvements of MetricsWebAPI such as the cache, not of the proposed system are needed.

From the viewpoint about the help of self-review, the improvement points to the proposed system such as the display of the comment of the commitment to confirm reason for change of each revision and comparison and visualization of metrics of some files at the same time were obtained. These functions are already provided by MetricsWebAPI, so the extension is comparatively easy.

Now visualization of only a per file is supported, but visualization of a per developer is absolutely necessary in

terms of self-review. For example, we think that visualizing the informations like list of edited files and co-developers and number of commitments of the developer can help more effective self-review. API provided by MetricsWebAPI now can't help the function of collecting metrics about this developer because basically they only access the information about mainly a file. To visualize metrics of a per developer, we need to develop a new service following the framework to collect metrics of every developer.

VI. Conclusion

In this paper, we developed a web application, MetricsViewer, which visualizs some variety of source code metrics to help self process improvements. Developer can easily look back their own development process without mining specific knowledge and other tools. From the experiment by four subjects, we confirmed that they could fined their impovements and reflections by doing so.

In our future works, we will extend MetricsViewer to support other visualization features that commonly used in development process improvement. For instance, experimental subjects required a feature to visualize a comment of each revision, a feature to compare some source codes and a feature to visualize developer's activities. Moreover, we are considering a granularity of visualizations. Both visualization of coarse-grained (e.g., project and package level) and fine-grained (source code and revision level) should help look back of developer's activity.

Acknowledgments

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No.24500079), Scientific Research (B) (No.23300009)], and Kansai Research Foundation for technology promotion.

References

[1] S. Matsumoto and M. Nakamura, "Service oriented framework for mining software repository," in *Proceedings of the Joint Conference of the 21st International Workshop on Software Measurement (IWSM) and the 6th International Conference on Software Process and Product Measurement (Mensura)*, 2011, pp. 13–19.

[2] E. Cerami, *Web Services Essentials*. O'Reilly, 2002.

[3] C. C. Williams and J. K. Hollingsworth, "Automatic mining of source code repositories to improve bug finding techniques," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 466–480, 2005.

[4] R. W. Selby, "Enabling reuse-based software development of large-scale systems," *IEEE Transactions on Software Engineering*, vol. 31, pp. 495–510, 2005.

[5] H. Kagdi, Y. S, and J. I. Maletic, "Mining sequences of changed-files from version histories," in *Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR '06)*, 2006, pp. 47–53.

[6] A. P. Nikora and J. C. Munson, "Understanding the nature of software evolution," in *Proceedings of the 19th International Conference on Software Maintenance (ICSM '03)*, 2003, pp. 83–93.

[7] 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.