A System for Information Integration between Development Support Systems

Soichiro Tani[1] Akinori Ihara[1] Masao Ohira[1] Hidetake Uwano[1,2] Ken-ichi Matsumoto[1]

[1]Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara, JAPAN
+81-743-72-5318

[2]Nara National College of Technology
22 Yata,Yamatokoriyama,Nara,JAPAN
+81-743-55-6000

{ soichiro-t, akinori-i, masao, matumoto } @ is.naist.jp, uwano @ info.nara-k.ac.jp

ABSTRACT

Many software projects use dezvelopment support systems such as bug tracking system (BTS) or version control system (VCS) to manage development information. Such the support systems preserve information according to a type of information (e.g., bug information in BTS and change information of source code in VCS). Since the systems do not provide developers with a feature to integrate several types of information required to complete development tasks, the developers need to collect the information by themselves that would result in inefficient development. In this paper, we demonstrate a system called SUSHI that helps developer integrate the information between multiple development support systems. Our system collects information which belongs to the same development context and provides developers with hyper links to related information in the support systems. Since our system also runs as a proxy server, developers can continue to use existing systems and stored information without any conversion.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *software configuration management, software quality assurance, programming teams.*

General Terms

Management

Keywords

information sharing, development context, bug tracking system, version control system

1. INTRODUCTION

With increasing outsourcing of software development or open source software projects, developers need to collaborate with geographically distributed co-workers and share information between them. In such the environment, they communicate each other to share information with asynchronous communication tools/systems such as mailing list (ML), bug tracking system (BTS) or version control system (VCS) that we call development support systems (DSS) in this paper.

In a large-scale software development organization, several DSSs are often integrated into a single system in order to optimize the efficiency of collaboration and information sharing among developers. The optimized integration of DSSs helps developers find and understand the past and current progress of development is dispersed in each DSS.

Meanwhile, a small-scale organization and open source software project does not have the integrated DSSs because they do not afford to build it by themselves or buy it. They often use a rental hosting service such as sourceforge.net that can be used at no fee. In case of using a rental hosting service, it is difficult to change/modify each DSS and composition of DSS to make the performance of information sharing better. Each developer needs to search several kinds of information in DSSs to understand the development context. For instance, when a developer receives a new patch via ML, s/he has to find a bug report which asks developers to make a patch to fix a certain bug, understand discussions about the bug on ML and BTS, and identify a file or module in VCS to apply the patch, and commit it into VCS. As just described, using DSSs as a combination of independent DSS sometimes consumes developers' time and effort [1, 2].

In order to resolve the issue on the use of DSSs in a small organization, in this paper we propose a system called SUSHI that supports information integration between multiple DSSs without changing in-use independent DSS in the organization. Our system serves as a proxy server which detects user's access to DSSs, collects the same information that the user refers to, and make an association with several kinds of the referred information in multiple DSSs. Using the association of development information, the next developer can easily find information relevant to his tasks.

2. SUSHI: INTEGRATING INFORMATION BETWEEN MULTIPLE DSSs

2.1 Overview

Figure 1 shows the information flow in using SUSHI. SUSHI works as a proxy server, collects information from BTS, and provides users with related information while the users look for information in each DSS through a web browser. Due to this configuration, SUSHI can deal with several kinds of information

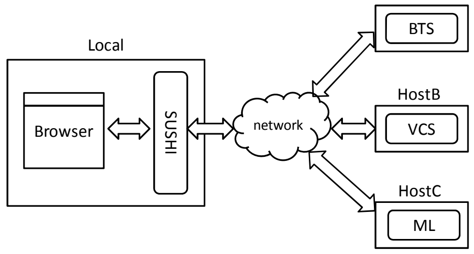

Figure 1. Information flow in SUSHI.

in multiple DSSs. At the same time, the users of SUSHI are not required to intentionally operate SUSHI to have information relevant to the development context.

2.2 Architecture of SUSHI

Figure 2 shows the architecture of SUSHI. SUSHI has four components: (1) **Access Detector**, (2) **Data Collector**, (3) **Estimator**, and (4) **Formatter**.

Access Detector automatically detects user's access to DSSs and notifies Data Collector of what information are browsed by the user. At this moment, Access Detector eliminates information irrelevant to DSSs such as web searching results.

Data Collector collects information in DDSs that the user referred to and sends them to the internal database. More importantly, it also extracts information which is used in *Estimator* to estimate which information are related each other. In the current implementation, *Data Collector* of SUSHI gathers bug ID, reporter's name, reported time and descriptions of a reported bug from BTS, revision number, committer's name, commit message and name of committed file from VCS, and poster's name, posted time, message ID and etc. from ML.

Estimator estimates the development context which consists of several events around the same time. *Estimator* makes an association with several kinds of information using such the events recorded in DSSs. The next subsection describes *Estimator* in detail.

Formatter provides a web page (html) which has the original information that the user is about to browse and relevant information from other DSSs. The user can browse several kinds of relevant information at a time to understand the development context.

2.3 Procedure of Estimation

In *Estimator*, making an association with relevant information, that is, estimating the development context is conducted as follows.

1. Developers specify characteristic words which are often used in their project.
2. *Estimator* counts the number of the specified key/characteristic words used in both the information that a user is browsing and the information that SUSHI's database has already stored.

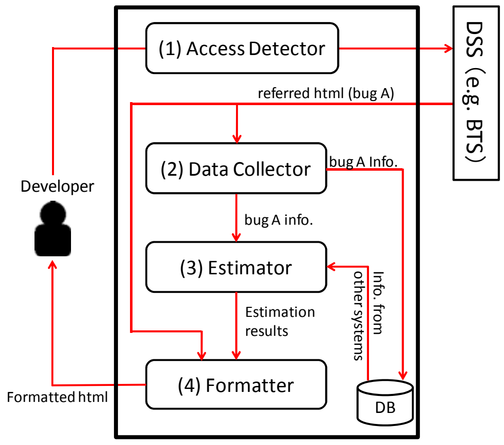

Figure 2. System architecture.

3. *Estimator* calculates the used rate of a specified word by dividing the number of the specified word used in each information by the total number of all the specified words used in each information.
4. *Estimator* calculates the estimated score by squaring the difference of the used rate between the information that a user is browsing and the information in SUSHI's database.
5. *Estimator* considers combinations of the information with lower estimated score as relative information each other.

3. PRELIMINARY EVALUATION

So far, we have applied SUSHI to two small-scale open source projects to confirm if the system works properly as we intended. Due to the space limit, we introduce a summary of the preliminary evaluation.

We asked open source developers to make links between information recorded in several DSSs to define which information is relevant each other. We also applied SUSHI to DSSs used in their projects and extracted links which were created by SUSHI. Comparing links manually defined by the developers with the links created by SUSHI, we found that the links (BTS→BTS, VCS→BTS, and VCS→VCS) created by SUSHI covered over 50% of the links defined by the developers. However, we also found that the links (BTS→CVS) less matched the developers' links (17%). This result showed that finding and recovering the missing links [3] are still difficult in our system.

4. FUTURE WORK

In the current, SUSHI only shows a simple output with estimated relevant information to uses. We need to consider more user-friendly user interface and visualization as uses can easily understand and remember the development context. We also improve the estimation algorism for relevant information. The current algorism is too simple to estimate relevant information correctly. Although the preliminary evaluation showed good results basically, we believe we can enhance the algorism (e.g., using TF-IDF) to make developers' information retrieval much more efficient.

5. ACKNOWLEDGMENTS

This work was conducted as part of StagE Project (the Development of Next Generation IT Infrastructure), Grant-in-Aid for Scientific Research (B), 23300009, 2011, and Grant-in-aid for Young Scientists (B), 22700033, 2011 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

6. REFERENCES

[1] Ohira, M,. Yokomori R., Sakai M., Matsumoto K., Inoue K., and Torii K. 2004. Empirical Project Monitor: A Tool for Mining Multiple Project Data. In *Proceedings of International Workshop on Mining Software Repositories (MSR'04)*. pp.42-46.

[2] Johnson, P.M. 2007. Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM'07)*. pp.81-90.

[3] Bachmann, A., Bird, C., Rahman, F., Devanbu, P., and Bernstein, A. 2010. The missing links: bugs and bug-fix commits. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (FSE'10)*. pp.97-106.