How Does Defect Removal Activity of Developer Vary with Development Experience?

Reou Ando, Seiji Sato, Chihiro Uchida,
Hironori Washizaki, and Yoshiaki Fukazawa
Department of Computer Science and Engineering
Waseda University
Tokyo, Japan
Email: waseda-reou@suou.waseda.jp, r0d8h8i0h@asagi.waseda.jp, c.u.0224@ruri.waseda.jp, {washizaki, fukazawa}@waseda.jp

Sakae Inoue, Hiroyuki Ono, Yoshiiku Hanai,
Masanobu Kanazawa, Kazutaka Sone,
Katsushi Namba, and Mikihiko Yamamoto
Fujitsu Limited
Kanagawa, Japan
Email: {inoue.sakae, ono.hiro, hanai.yoshiiku, kanazawa.masano, sone.kazutaka, nanba, yamamoto.mikihi}@jp.fujitsu.com

***Abstract*—Because developers significantly impact software development projects, many researchers have studied developers as a means to improve the quality of software. However, most works have examined developers in a single project, and research involving multiple projects has yet to be published. Herein we propose an analysis method which investigates whether an evaluation of developers based on individual experience is feasible when targeting more than one project by the same organization transversely. Our method deals with the logs of the version control system and the bug tracking system. To support this method, we also propose two models to evaluate developer, the defect removal overhead rate (DROR) and developer's experience point (EXP). The results reveal the following. 1) DROR cannot be used to compare different projects in the same organization. 2) There is certainly a difference in DROR's between experienced and inexperienced developers. 3) EXP should be a useful model to evaluate developers as the number of projects increases. The data obtained from our method should propose the personnel distribution measures within the development framework for future developments, which might lead to improve the quality of software.**

I. INTRODUCTION

In software development projects, developers and organizations are said to significantly impact software [1-15, 17]. A 1968 study on the organization when analyzing software quality resulted in Conway's law [2], which states that "organizations that design systems are constrained to produce systems which are copies of the communication structures of these organizations." Recently, researchers have examined defect prediction in software using metrics based on hypotheses formed by the structure of an organization [12], and have investigated the effects of software in a project involving multiple organizations due to mergers and acquisitions [14], etc. Such studies have found that organizational structures greatly influence software quality [2, 3, 12, 14].

On the other hand, research on developers has proposed techniques to improve the prediction of potential defects in software by utilizing the quality of the developer. The quality of the developer is defined as how much his commits lead to defects in a project [17]. Defect prediction using metrics, such as the number of commits and LOC for each developer [6], assesses the impact of developers on the quality and reliability of software [1, 4-9, 11, 13, 15, 17].

Most studies focus on the organizational structure and the quality of the developers with respect to a single project or a group project involving different organizations. However, the results across multiple projects by the same organization have yet to be published. With regard to the experience of developers who belong to the same organization, it is easy to imagine that the development experience in past projects affects later software development. In fact, although the target of their research was a single project, A. Mockus et al. [11] found that developer's experience significantly affects the possibility of defects; more experienced developers tend to have fewer defects.

If the results about developers based on past development experience are obtained by traversing multiple projects, it may be possible to improve a new project by structuring it so that is similar to developers' previous experiences. Moreover, assuming a developer with little development experience introduces more defects, the development system should be arranged so that inexperienced developers work with experienced developers. This should improve the quality of software while simultaneously educating inexperienced developers. Therefore, we propose a technique to evaluate developers by analyzing their previous experiences from logs stored in the version control system and the bug tracking system in multiple projects. To determine how the defect removal activity of developers varies with development experience, we divided the issue into evaluable components. Hence, we formulated our study in the form of three research questions:

- *RQ1: As an organization gains project experience, does the defect removal overhead rate (DROR) of developers tend to decrease?*
- *RQ2: Is there difference in DROR based on development experience?*
- *RQ3: Is there difference in DROR between developers based on experience in a similar project?*

In order to respond to these research questions, we carried out evaluation experiments using our method. The

(DOI reference number: 10.18293/SEKE2015-221)

subjects of our study are developers in a real company involved in three projects, which do not overlap in the development periods.

The contributions of this study are:

- A method to evaluate developers based on past development experience using logs stored in the version control system and the bug tracking system.
- Understanding the trend of DROR based on developer experience.
- Obtaining resources to help improve measures of personnel distribution within the development framework for future developments.

The rest of the paper is organized as follows. Section 2 presents the background of our study through related work. Section 3 introduces our analysis method to address the problem described in Section 2. Then two analysis models to support our method are proposed in Section 4. In Section 5, we conduct experiments to evaluate our method and investigate the proposed research questions. Next Section 6 explains summary of findings and the practical application of our method. Finally we describe the conclusion in Section 7.

II. Background And Related Work

A. Prior works focusing on developers

In software quality analysis, several works propose methods to predict defects in software based on the characteristic of developers [1, 4-9, 11, 13, 15, 17]. For example, Kamei et al. [6] observed the histories of developers commits. They proposed change measures, which extract the number of modified files recorded for each commit, lines of code added, and whether or not the change is a defect fix, etc. They found that by predicting software defects through change measures, high-risk fixes and the cost of high-quality software could be reduced.

Matsumoto et al. [9] extracted metrics such as the number of commits and LOC for each developer from the logs of the version control system. They supposed that these are useful for fault-prone analysis, which specifies the module containing defects. Besides, Y. Wu et al. [16] defined the quality for each developer from the proportion of commits that introduce defects into a project. They found that using their proposed eight metrics as parameters as lead to better fault-prone analysis compared to traditional process metrics.

B. One of the problems in related works

Developer experience varies by the individual. Numerous works deal with it [5-8, 11, 15], but the research focuses on evaluating a single project or a group of different organizations. Research on multiple projects in the same organization has yet to be published. Most prior works probably evaluate a single project, even though they considered developer experience.

If developers with experience are compared to those without experience, it is conceivable that there will be differences. In addition, it is possible that the type of experience leads to differences among experienced developers. Therefore, the research aims to evaluate developers based on their development experience in multiple projects within the same organization in a cross-sectional way.

III. Analysis Method

The participants in our study are developers involved in large-scale projects in an organization that uses a version control system and a bug tracking system.

Our analysis involves the following steps:

(i) Extract logs from the version control system and the bug tracking system used in completed projects.

(ii) Collect the names of developers, the number of files they changed, the names of the absolute path that they changed files, and the number of changes in them from the log of version control system. In addition, identify files recorded as defect fixes after detecting the defect; that is, files related with a defect (hereinafter referred to as *defect files*), from the logs of the version control system and the bug tracking system. Then collect the name of developer who changed defect files and the number of changed defect files.

(iii) Gather the number of changed files, the changed absolute path's name and its number, and the number of changed defect files by developer name.

(iv) Calculate each developer's *defect removal overhead rate* (DROR), which is detailed in Section 4, from the number of changed files for each developer.

(v) Repeat steps (i) to (iv) for each completed project.

(vi) If a developer's name exists in different projects, consider the developer to be experienced in later projects. Then calculate the *developer's experience point* (EXP) in the project, which is detailed in Section 4.

Finally, analyze each developer based on the gathered data. Incidentally, we assumed that a function should be implemented not by single file, but by all files included in the absolute path, which is why we use the number of changed absolute paths and not the number of changed files. In this method, the number of changed files includes the number of changing defect files.

IV. Analysis Model

It seems important to prepare an indicator to link developers with the number of defect to evaluate developers individually. In this paper, we present a metric called *defect removal overhead rate* (DROR) for each developer. Moreover, in order to examine precisely what area and how much ability a developer has acquired, we also suggest a measure named *developer's experience point* (EXP) for each experienced developer.

A. Defect removal overhead rate (DROR)

In a large-scale development, it is important that people who are not engaged in implementation are involved in detecting defects. For this reason, it is probable that developers differ from testers. A developer who modifies certain defect files should have changed it because he induced the defects that testers requested to be fixed.

Table 1. Example each d_{P1}'s DROR calculation

Date	d_{P1}			***P1***			
	x	**y**	**z**	**f**$_1$	**f**$_2$	**f**$_3$	**f**$_4$
2015/1/10	✓			○	○	○	
2015/1/11		✓				◎	
2015/1/12			✓				◎
2015/1/15		✓		●	●	⦿	
2015/1/16			✓				⦿
$DROR(d_{P1})$	$\frac{0}{3}$	$\frac{3}{4}$	$\frac{1}{2}$	○: Changed file ◎: Changed file (defect occurred) ●: Defect-fixed file ⦿: Defect-fixed file (defect removed)			

Hence, we assume that the person who injected defects into a file is the person who changed it. This assumption is used to define defect removal overhead rate (hereinafter referred to as DROR) of a developer.

DROR of a developer is calculated as the proportion of fixing defect files compared to the total number of files that he changed. When developer d_R involved in project R, d_R's DROR is defined as

$$DROR(d_R) = \frac{NDF_{d_R}}{NF_{d_R}} \qquad (1)$$

R: The project which evaluates developer
d_R: Developer who involved in R
NF_{d_R}: The total number of d_R changing file in R
NDF_{d_R}: The number of d_R fixing defect file in R

Equation (1) can also be understood as the probability that the developer fixes a defect file when changing a file. The higher DROR, the more the developer is evaluated badly. It is because developers who write low-quality code should change more files related with a defect than developers who write high-quality program. Table 1 gives an example of each d_{P1}'s DROR calculation. When developer x changed files f$_1$, f$_2$ and f$_3$ on 2015/1/10, DROR of x is calculated as $\frac{0}{3}$ because he didn't fix defect files. Besides, developer y changed file f$_3$, in which he might have induced a defect at that time, on 2015/1/11. And he fixed defect files f$_1$, f$_2$ and f$_3$ on 2015/1/15. Then, DROR of y is measured as $\frac{3}{4}$. In addition, developer z changed file f$_4$, in which he might have introduced a defect, on 2015/1/12. If he fixed file f$_4$ on 2015/1/16, DROR of z is figured out as $\frac{1}{2}$. If comparing these developers, developer y is evaluated the worst.

B. Developer's experience point (EXP)

Developer's experience point (hereinafter referred to as EXP) is measurement that considers his development experience. When there is developer $d_{P,R}$ who has experienced past projects P and is involved in the project R, $d_{P,R}$'s EXP in R is defined as

$$EXP(d_{P,R}) = \sum_{p_i \in R} C_R(p_i) \times C_P(p_i) \qquad (2)$$

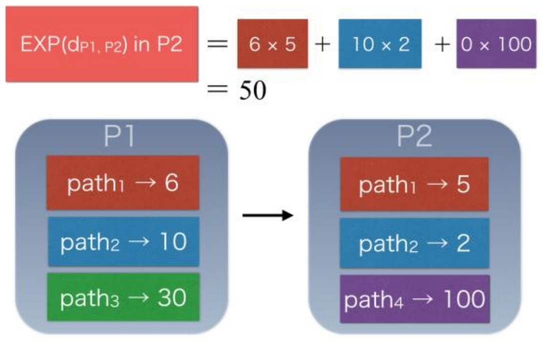


Figure 1. Example $d_{P1,P2}$'s EXP calculation

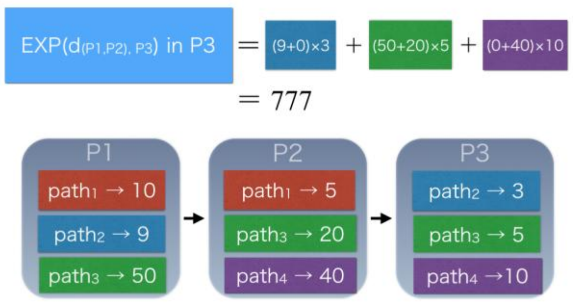
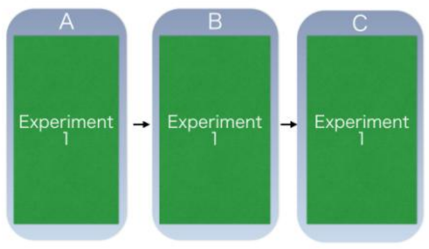


Figure 2. Example $d_{(P1,P2),P3}$'s EXP calculation

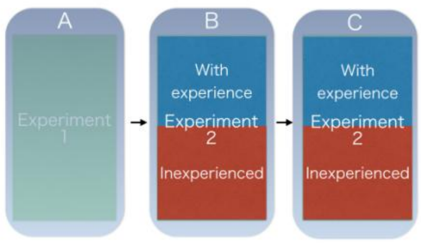
P: Past projects
R: The project which evaluates developer
$d_{P,R}$: Developer who experienced P and R
$C_R(p_i)$: The number of appearing absolute path p_i which $d_{P,R}$ changed in R

Equation (2) means that if absolute path p_i in which $d_{P,R}$ changed files in R also exists in P, the number of changing files in p_i in R, defined $C_R(p_i)$, is weighted by that in P, which defined $C_P(p_i)$. The higher EXP, the more experience the developer has. The thought of (2) is developed by referring to A. Mockus et al. [11] and Y. Kamei et al. [6]. Figure 1 gives an example calculation of EXP when $d_{P1,P2}$ changed the contents of path$_1$ 6 times, path$_2$ 10 times, and path$_3$ 30 times in P1. Moreover, he also edited path$_1$ 5 times, path$_2$ 2 times, and path$_4$ 100 times in P2. Then, his EXP in P2 is calculated as 50 (i.e. (6×5) + (10×2) + (0×100)). Note that path$_3$ in P1 is not used in this example because he did not change it in P2. Figure 3 gives another example of EXP calculation. When $d_{(P1,P2),P3}$ changed the contents of files shown in Fig. 2, his EXP in P3 is figured out as 777 (i.e. ((9+0)×3) + ((50+20)×5) + ((0+40)×10))).

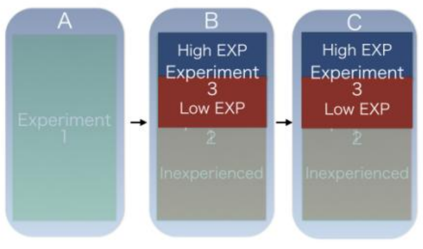
There are two purposes to define EXP using Eq. (2). First developers can be separated according to development experience. Although many developers have some experience, the amount likely varies by developer. If they are treated equally, the evaluation of developers can be mistaken. The other purpose is to consider developers with some experience but not in the type of project. As a result of taking these purposes into account, we adopted the system that $C_R(p_i)$ is weighted by $C_P(p_i)$. Scale type of EXP is ratio scale; EXP takes value from 0 indicating that the corresponding developer has no experience.



(a) Subject of evaluation experiment 1



(b) Subject of evaluation experiment 2



(c) Subject of evaluation experiment 3

Figure 3. Subject of each evaluation experiment

V. EMPIRICAL EVALUATION

To evaluate the proposed method in this paper, we analyzed hundreds of developers who were involved in three different completed projects of embedded system development in a real company. In these three projects, developers released software that is enhanced seasonally. Thus, it is reasonable that the order of time is about the same for each project and it is unlikely that the projects were carried out simultaneously. In the following, three projects by this company are named project A, B, and C in order of time. Incidentally, the scale of this company's project ranges from 200,000 LOC to 300,000 LOC per project. There were hundreds or a few thousands of defects and thousands of commits per project[1].

A. Experiment

We obtained the logs of the version control system (Perforce[2]) and the bug tracking system (Prismy[3]) used in projects A, B, and C. Then, we gathered data for developers in each project according to procedure described in Section 3. Next, we set up evaluation experiments to correspond to each research question presented in Section 1. Finally, we divided the developers into several groups (Fig. 3).

- **Evaluation experiment 1 corresponding to RQ1** divides the developers into three groups depending on whether they are involved in project A, B, or C.
- **Evaluation experiment 2 corresponding to RQ2** divides the developers into two groups according to whether they have experience in previous projects.
- **Evaluation experiment 3 corresponding to RQ3** divides developers into two groups with median of EXP as a boundary on those who have experience in projects B and C.

With respect to the results, we created boxplots and graphs of the empirical cumulative distribution function, that is to say ECDF, for DROR of a developer for each evaluation experiment. Reading the vertical axis of an ECDF graph when the horizontal axis is fixed allows the proportion of developers who have DROR up to a value that the horizontal axis indicates to be determined. On the other hand, if the graph is read through the horizontal axis with the vertical axis fixed, the maximum DROR can be grasped in proportion of developers.

[1] Due to a confidentiality agreement, we do not show precise numbers.
[2] http://www.perforce.com
[3] http://www.tjsys.co.jp/page.jsp?id=742

B. Results and discussion

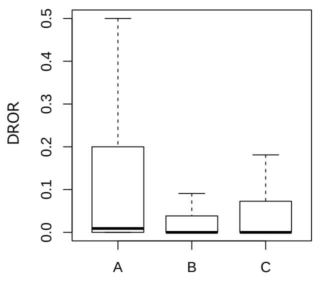
RQ1: As an organization gains project experience, does the DROR of developers tend to decrease?

Figure 4 shows a boxplot and an ECDF of DROR in evaluation experiment 1. Many developers in project A have a higher DROR than those in project B and C. Considering this and the fact that projects B and C are derived from project A, DROR seems to depend on organization experience. However, comparing ECDF of project B with that of project C shows that the proportion of developer in project B with a 0.2 or less DRORs is more than that in project C. Furthermore, comparing the boxplot of project B and that of project C indicates that the DROR of project C is more scattered than that of project B, suggesting that the DROR of developers varies with factors other than their development experience. It might be because some of developers are already at their peak and did not improve significantly.

These findings show that we cannot affirm that DROR tends to decrease as an organization experiences projects.

RQ2: Is there difference in DROR based on development experience?

Figure 5 shows the boxplots of the DROR in evaluation experiment 2. There is a gap in the DROR's for both project B and C according to developer experience. In addition, the width of boxplots for DROR of inexperienced developers in project B differs from that in project C, but the width of the boxplots of experienced developers in project B is in good agreement with that in project C. These results suggest that experienced developers are free not influenced by changes in the development system or software design in between projects B and C.



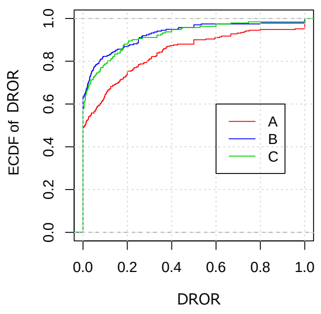


Figure 4. Boxplot and graph of ECDF of DROR in experiment 1

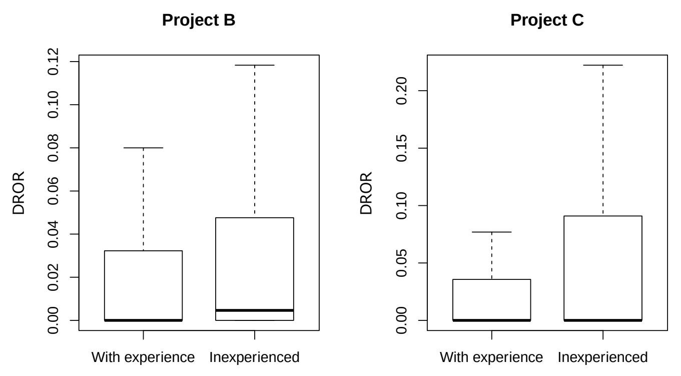


Figure 5. Boxplots of DROR in experiment 2

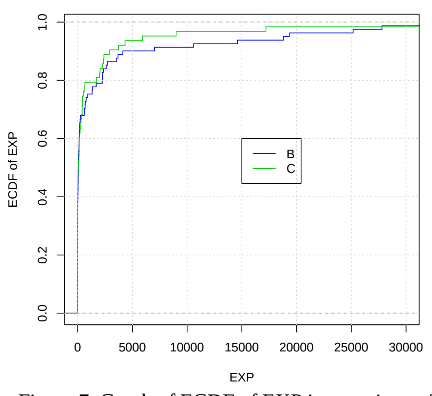


Figure 7. Graph of ECDF of EXP in experiment 3

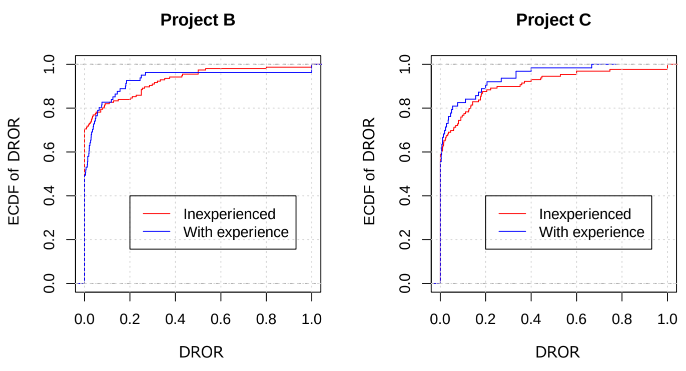


Figure 6. Graphs of ECDF of DROR in experiment 2

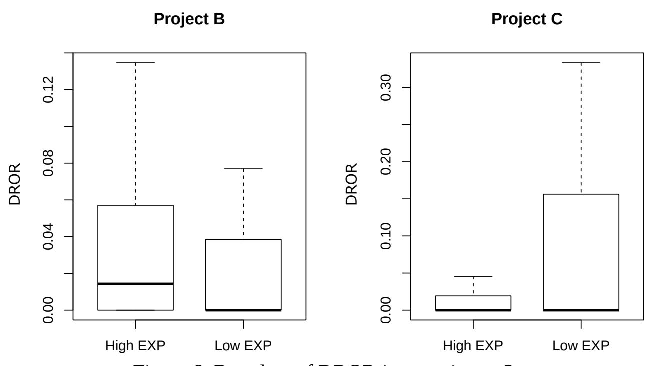


Figure 8. Boxplots of DROR in experiment 3

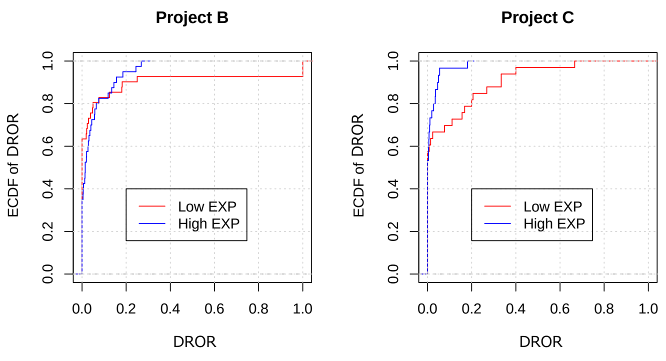


Figure 9. Graphs of ECDF of DROR in experiment 3

Figure 6 represents the ECDF graphs of DROR in evaluation experiment 2. The proportion of developers with experience and a DROR of 0.1 or less is higher than that in project B. On the other hand, the relation is opposite if the proportion is more than 0.1. In project C, regardless of reading the vertical axis with any position of a horizontal axis fixed, Fig. 6 indicates that DROR's of developers with experience is lower than those without experience. Moreover, judging from ECDF of inexperienced developers in both of project B and C, their DROR differs by project.

The above results show that the DROR of experienced developers is lower than that of inexperienced developers. The difference between the groups depends on the inexperienced developers and varies by project.

RQ3: Is there difference in DROR between developers based on experience in a similar project?

Figure 7 shows a graph of ECDF of EXP in evaluation experiment 3. EXP depends greatly on the number of changing files in a project due to its definition. Thus, ECDF of EXP differs by project, indicating that EXP cannot be used to compare traversing projects of developers.

Figure 8 shows the boxplots of the DROR in evaluation experiment 3, while Fig. 9 graphs ECDF of the DROR. With regard to project B, the width of the boxplot of developers with a high EXP is wider than those with a low EXP (Fig. 8). In addition, more than 60 percent of developers with high EXP in project B have greater than 0 DRORs (Fig. 9). These results suggest that other factors, which cannot be measured in terms of EXP, lead to defects. On the other hand, there is a gap between developers with high EXP and those with low EXP. This result suggests that when a developer involved in one project decides to engage in a similar one, his DROR should be reduced.

It remains to be seen if there is the difference of DROR's between developers with different experience levels. However, as the number of projects increases, our analysis method and EXP should be a useful metric to evaluate developers.

C. *Threats to validity.*

Internal validity:

This research focused on projects B and C, which were derived from the development of project A. Except for the notation variability of the absolute path among projects, if the absolute paths of a file in current development corresponded to that in past development, they were regarded as the same development function. Otherwise, they were viewed as quite different functions. This is a threat to internal validity. In the future, the influences of this assumption on this analysis method must be confirmed by comparing the similarity between path names or function names inferred from path name, not the coincidence between absolute paths.

In addition, we determined the DROR based on the hypothesis that the developer who changed a file related to defect also induced the defect. As a result, some developers had a DROR of 1.0; in other words, some developers always caused defects. Although they worked as debuggers in actual development, this may affect the experimental results. This is also a threat to internal validity. In the future, who induced a defect must be more accurately identified by applying the SZZ algorithm proposed by J. Sliwerski et al. [12]. This algorithm infers commits, which brought about defects from *diff* and *annotate* commands of the version control system. By the additional investigation, we will clarify the correctness and limitation of the above-mentioned hypothesis in detail.

External validity:

In this experiment, we used Perforce as the version control system and Prismy as the bug tracking system. This is a threat to external validity. However, the analysis method of this paper is not designed for this experiment. So it may be effective in the same way for the domain that uses both a version control system and a bug tracking system. In the future, the efficiency of other domains and companies that handle version control systems and bug tracking systems must be verified.

VI. Summary of Findings and Usage

Summary of findings are: 1) DROR cannot be used to compare different projects in the same organization. 2) There is certainly a difference in DROR's between experienced and inexperienced developers. 3) EXP should be a useful model to evaluate developers as the number of projects increases.

If the next development project is similar to past projects, our method provides useful information to improve personnel assignments. It can arrange the system so that experienced developers guide inexperience ones as they work on development together. This should improve the quality of the software developed in the next project.

VII. Conclusion and Future Work

To examine the tendency of DROR of developers based on development experience, we propose an analysis method and two models, which evaluate developers across multiple projects using their records in the same organization. The research found that despite being the same domain, comparing projects directly is not useful and that DROR of the developers with experience is lower than those without experience. Although it is unclear where there is a difference in the defect flow rates between developers with much and some experience, our proposed analysis model, EXP, should help evaluate developers for future projects.

As a future work, we will investigate files or absolute paths changed by developers who had high DROR despite having a lot of experience. If this is understood, it might be possible to evaluate the difficulty of functions to be developed, which may improve the precision of EXP. Moreover, we would like to try to discuss relations among defect removal overhead, defect inflow (how many defects a developer introduced in files he changed) and defect removal efficiency (how many fixes a developer processed in all defects) to improve the precision when evaluating developer.

Acknowledgment

Our thanks go to anonymous reviewers who gave us a lot of valuable comments to improve this paper.

References

[1] C. Bird, N. Nagappan, B. Murphy et al., “Don’t Touch My Code! Examining the Effects of Ownership on Software Quality,” *ESEC/FSE '11 Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp.4-14, 2011.

[2] M. Conway, “How Do Committees Invent?”, *Datamation*, vol.14, no.4, pp.28-31, 1968.

[3] P. Donzelli, R. “Handling the knowledge acquired during the requirements engineering process - a case study -,” *SEKE '02 Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pp. 673-679, 2002.

[4] J. Eyolfson, L. Tan, P. Lam, “Do Time of Day and Developer Experience Affect Commit Bugginess?”, *MSR '11 Proceedings of the 8th Working Conference on Mining Software Repositories*, pp. 153-162, 2011.

[5] F. Fagerholm, M. Ikonen, P.Kettunenet al., “How do Software Developers Experience Team Performance in Lean and Agile Environments?”, *EASE '14 Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, No.7, 2014.

[6] Y. Kamei, E. Shihab, B. Adams et al., “A Large-scale Empirical Study of Just-in-Time Quality Assurance,” *IEEE Transactions on Software Engineering*, vol.39, no.6, pp. 757-773, 2013.

[7] E. Kocaguneli, A. T. Misirli, B. Caglayan et al., “Experiences on Developer Participation and Effort Estimation,” *SEAA 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp.419-422, 2011.

[8] R. Latorre, “Effects of Developer Experience on Learning and Applying Unit Test-Driven Development,” *IEEE Transactions on Software Engineering*, vol.40, No.4, pp. 381-195, 2014.

[9] S. Matsumoto, Y. Kamei, A. Monden et al., “An Analysis of Developer Metrics for Fault Prediction,” *PROMISE '10 Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, No.18, 2010.

[10] A. Mockus, “Organizational Volatility and its Effects on Software Defects,” *FSE '10 Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pp.117-126, 2010

[11] A. Mockus, D. M.weiss, “Predicting Risk of Software Changes,” *Bell Labs Technical Journal*, Vol.5, No.2, pp.169-180, 2000.

[12] N. Nagappan, B. Murphy, and V. Basili, “The Influence of Organizational Structure on Software Quality: An Empirical Case Study,” *ICSE ’08 Proceedings of the 30th international conference on Software engineering*, pp.521–530, 2008.

[13] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Programmer-based Fault Prediction,” *PROMISE '10 Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, No.19, 2010.

[14] S. Sato, H. Washizaki, Y. Fukazawa, S. Inoue, H. Ono, Y. Hanai and M. Yamamoto, et al., “Effects of Organizational Changes on Product Metrics and Defects,” *APSEC 2013 20th Asia-Pacific Software Engineering Conference*, vol.1, pp.132-139, 2013.

[15] E. Shihab, A. E. Hassan, B. Adams et al., “An Industrial Study on the Risk of Software Changes,” *FSE '12 Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, No.62, 2012.

[16] J. Sliwerski, T. Zimmermann, and A. Zeller, “When Do Changes Induce Fixes?” *MSR '05 Proceedings of the 2005 international workshop on Mining software repositories*, pp.1-5, 2005.

[17] Y .Wu, Y. Yang, Y.Zhao et al., “The influence of developer quality metrics for fault prediction,” *SERE 2014 Eighth International Conference on Software Security and Reliability*, pp.11-19, 2014.