Developer Experience Considering Work Difficulty in Software Development[*]

Taketo Tsunoda, Hironori Washizaki, Yosiaki Fukazawa
Department of Computer Science and

Engineering Waseda University

Tokyo, Japan

Email: tsunoda_q1ll@asagi.waseda.jp

{washizaki, fukazawa}@waseda.jp

Sakae Inoue, Yoshiiku Hanai, Masanobu Kanazawa

Fujitsu Connected Technologies Limited

Kanagawa, Japan

Email: {inoue.sakae, hanai.yoshiiku,

kanazawa.masano}@jp.fujitsu.com

Abstract

Previous studies have researched how developer experience affects code quality, but difficulty was ignored although experienced developers are more likely to work on more complex parts of a project. We examine work difficulty by focusing on revised files. Using product metrics, we evaluate file complexity for each type of file origin. Specifically, we analyze three large commercial projects (each with about 250,000 LOC) executed by the same organization to analyze the relationship between previous project experience and developer's work. Although experienced developers do not necessarily work on more complicated files, they introduce fewer defects, especially if the difference in work difficulty is not significant.

Keywords: Experience; Product Metrics; Organization change; Software quality;

1. Introduction

In software projects, work should be distributed to the appropriate developers. Some studies have suggested methods to determine experts in a particular domain [14], their roles [6], and developer relationships [16]. These studies are helpful to understand the main contributor in a specified domain and assign jobs. In addition, previous studies have researched the correlation between developer experience and code quality [2], [3], [5], [7], but the defect tendency of a developer depends on the file complexity. Other works have suggested that experienced developers are more defect-prone because they often work in the most complex regions [5], [7]. Consequently, to fairly evaluate developers, work difficulty needs to be assessed.

In this study, we analyze three large commercial projects executed by the same organization to examine the relationship between experience and defects while considering work difficulty. Work difficulty is determined by classifying files according to their origin,

[*] This paper is an extended version of a paper presented at [19].

while file complexity is evaluated using product metrics. This study makes the following contributions:

1. Developers who are unfamiliar with previous projects tend to introduce more defects even if the files are not complex.
2. Experienced developers do not always work on more complicated files or introduce fewer defects. However, when considering similar work difficulty, experienced developers introduce fewer defects.
3. Most developers do not introduce many defects. However, for all three projects, those who introduced the most defects were responsible for the most complex files.

2. Related Works

Many researchers have previously studied fault prediction in software projects [17], [18], and some studies indicate that developer experience affects software quality. Eyolfson et al. researched the correlation between developer experience (by days on a project) and defect tendency [5]. They found that the number of bugs decreases as experience increases, but a more experienced programmer is not necessarily less buggy. Rahman researched how specific experience in a target file and general experience are related to defects [2]; general experience is not related to a defect, but specific experience is closely related. Additionally, Ando et al. defined the developer experience using several projects and researched the correlation between defects and experience [7]. They suggest that other factors such as work difficulty affect the defect tendency. In the contrast to previous studies, we investigate the difference between experienced and inexperienced developers while considering file complexity.

3. Background

3.1. SZZ Algorithm

A bug tracking system can easily identify who fixed a bug, but it cannot indicate who introduced a bug. To solve this problem, we adopt the “SZZ algorithm” suggested by Sliwerski [11]. This algorithm assumes that a bug-fix-change fixes bad code, which is also the cause of the defect. Thus, if we specify the location that is changed by the bug-fix and when each line is created, we automatically identify the bug introduced change. The SZZ algorithm uses SVN and a bug tracking system.
Our method employs the following steps:

1. Retrieve the bug fix revision from the bug tracking system.
2. Use the diff command to specify which lines are changed.
3. Use the VCS annotate command to examine when changed lines are created.

In addition, the rules below are added upon considering Kim’s research [12].

1. Ignore comment changes.
2. Ignore format and blank line changes.
3. Ignore outlier bug-fix commits in which too many files were changed.

3.2. File Origin and Metrics

The file origin means how many organizations have previously modified the files. Often another organization takes over the development of a software project. Table 1 shows how many organizations are related to each file (Number of Organizations) in this study. Each organization from Ox to Oz edits the file in chronological order. For example, Oy and Oz modified file f1 originally created by Ox. Thus, the Number of Organizations of f1 is 3. Sato indicated that a file related to several organizations has higher product metric values and is more prone to defects [13]. Because our research is also related to three organizations, we classify the files according to the Number of Organizations. In addition, we adopt two representative product metrics, lines of code (LOC) and number of functions and methods defined in other files that a file calls (Call Number), to evaluate the file complexity.

In our preliminary survey, inexperienced developers working on files related to several organizations and files with high metric values are similar to experienced developers. In this research, we investigate whether there are differences in code quality and complexity between experienced and inexperienced developers using the defects and two product metrics.

Table 1. File origins when three organizations are involved

File	Ox	Oy	Oz	Number of Organizations
f1	Create	Modify	Modify	3
f2		Create	Modify	2
f3	Create		Modify	2
f4			Create	1

4. Evaluation and Results

To investigate the difference between experienced and inexperienced developers, we analyzed developers who participated in three large commercial projects executed by the same organization. (Each project has about 250,000 LOC and 200 developers.) Based on chronological order, the projects are called A, B, and C. It should be noted that each project development was taken over by another organization before the project started, which is similar to using a framework or open source developed by other organizations. Therefore, each file has its own origin. Additionally, these projects released the successor model of the previous projects. About half of the developers participated in the next project. Many files were passed down to the next project (A to B or B to C). We call passed down files "inherited files". Because not all developers participate in the next project, some files are edited by developers inexperienced with such files (file-experience). We hypothesize that files edited by developers without file-experience are more likely to contain defects.

In addition, we researched the difference between files edited by developers with and without experience on previous projects. In this research, we analyze all files, including "inherited files" and "new files". Finally, we examine whether the same developers introduce many defects in each project and whether each project has its own features.

We propose the following researching questions:

RQ1) Do defect tendency and complexity of inherited files vary with file-experienced developer participation?

RQ2) Do experienced developers introduce fewer defects or work on more complicated files?

RQ3) Does the same developer introduce many defects in each project? What are the features of these developers?

4.1. *RQ1-1 Does defect proneness of inherited files vary with file-experienced developer participation?*

Figure 1 shows the rate of defect files in the inherited files. A defect file means that at least one defect is introduced. The x-axis indicates the number of related organizations, while experience group denotes whether the files are revised by file-experienced developers at least once. The findings do not support our hypothesis as the experience group introduced more defects.

4.2. *RQ1-2 Does the complexity of inherited files vary with file-experienced developer participation?*

Figure 2 shows the metrics values of inherited files. In projects A and B, experienced groups have high metrics. Table 2 indicates that the Wilcoxon rank sum test with the alternative hypothesis set to "files edited by a file-experienced developer have higher product metrics than other inherited files in each type of origin (Number of organizations)". It shows that most of the P-value are low. Although files edited by file-experienced developers are more defective, they are also more complicated.

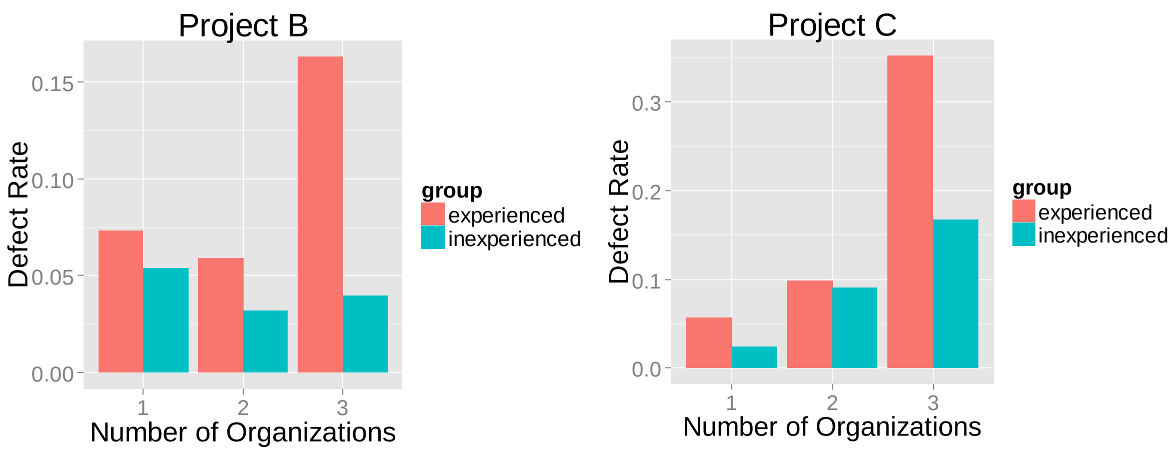

Fig. 1. Defect rate of inherited files

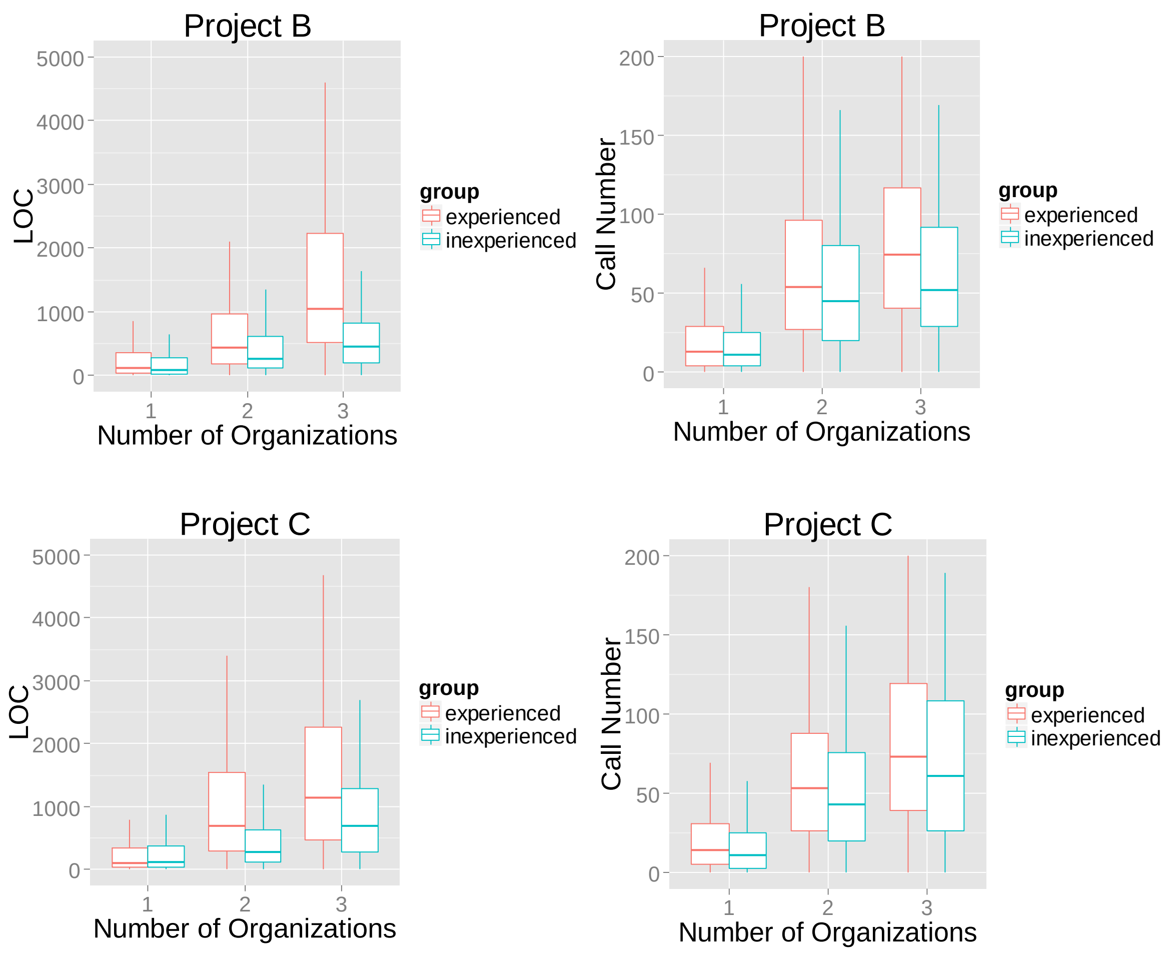

Fig. 2. Product metrics of inherited files

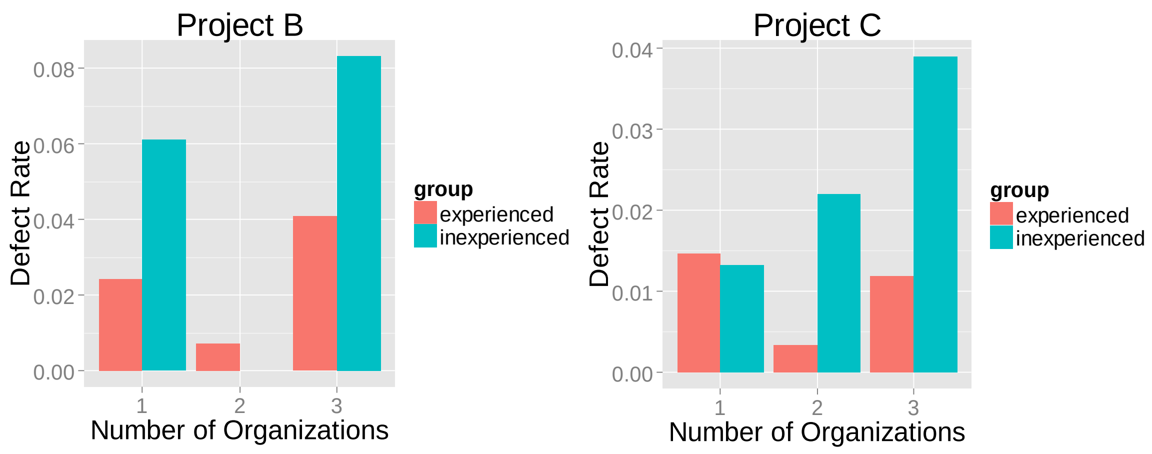

Fig. 3. Defect rate of files edited by a single developer

Table 2. Wilcoxon Rank sum test in RQ1-2

Project	Number of organizations	LOC	Call Number
B	1	P<<0.01	P<0.05
	2	P<<0.01	P<<0.01
	3	P<<0.01	P<<0.01
C	1	0.8	P<<0.01
	2	P<<0.01	P<0.01
	3	P<<0.01	P<<0.01

4.3. RQ2-1 Do experienced developers introduce fewer when they edit files alone?

Figure 3 shows the rate of defect files edited by a single developer. An experienced developer means that the developer worked on a previous project. In some cases, experienced developers introduced slightly more defects. In many cases, they introduced fewer defects, especially if they are related with all three organizations. Thus, if a developer solely edits a file, few defects are introduced on the whole. However, inexperienced developers tend to introduce more defects in complex files due to the lack of experience.

4.4. RQ2-2 Do experienced developers work on more complicated files when editing files alone?

Figure 4 shows the product metrics of files edited by a single developer. Table 3 indicates that the Wilcoxon rank sum test with the alternative hypothesis set to “experienced developers work on higher metrics files in each type of origin”. Experienced developers do not work on more complicated files. Additionally, there is not a significant difference between experienced and inexperienced developers.

Table 3. Wilcoxon Rank sum test in RQ2-2

Project	Number of organizations	LOC	Call Number
B	1	P<0.05	0.9
	2	P<0.05	0.3
	3	0.7	0.9
C	1	1	P<<0.01
	2	P<0.05	0.8
	3	P<<0.01	0.1

4.5. RQ2-3 Do experienced developers introduce fewer defects when they edit files with the other developers?

Figure 5 shows the rate of defect files edited by several developers. In each group, “experienced group” means that files are revised by experienced developers at least once. In the experienced group, only defects introduced by “experienced developers” are considered. Similarly, only defects introduced by “inexperienced developers” are considered in the inexperienced group. The experienced group introduced more defects in project B.
In addition, the experienced group introduced more defects in both projects when the three organizations are related. This contradicts the results of RQ2-1. However, it is possible that files related to the three organizations and several developers induce more defects regardless of developer experience

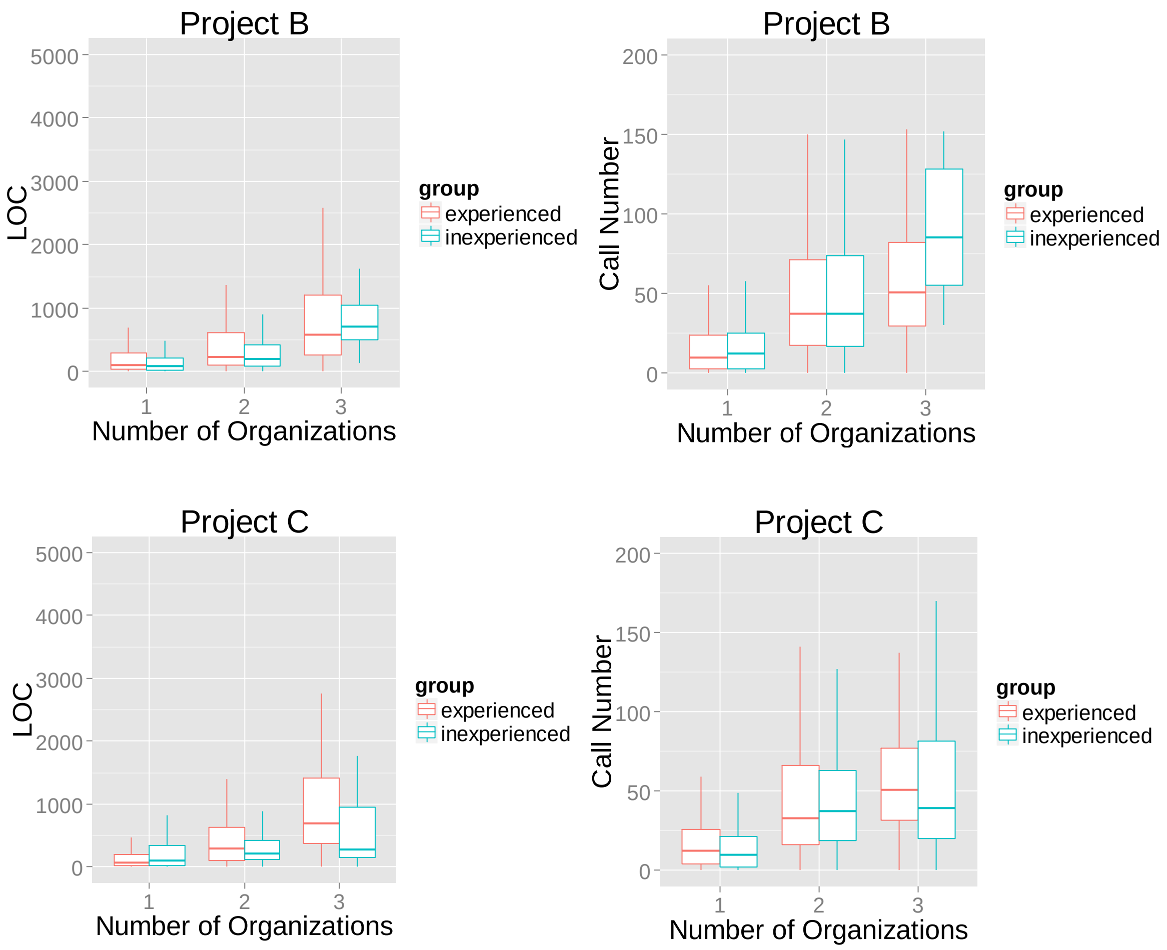

Fig. 4. Defect rate of files edited by a single developer

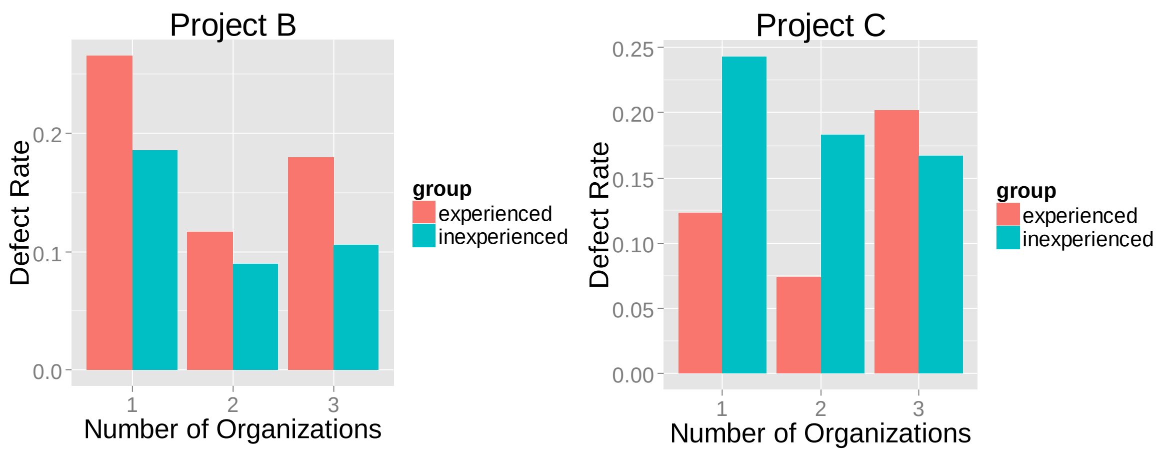

Fig. 5. Defect rate of files edited by several developers

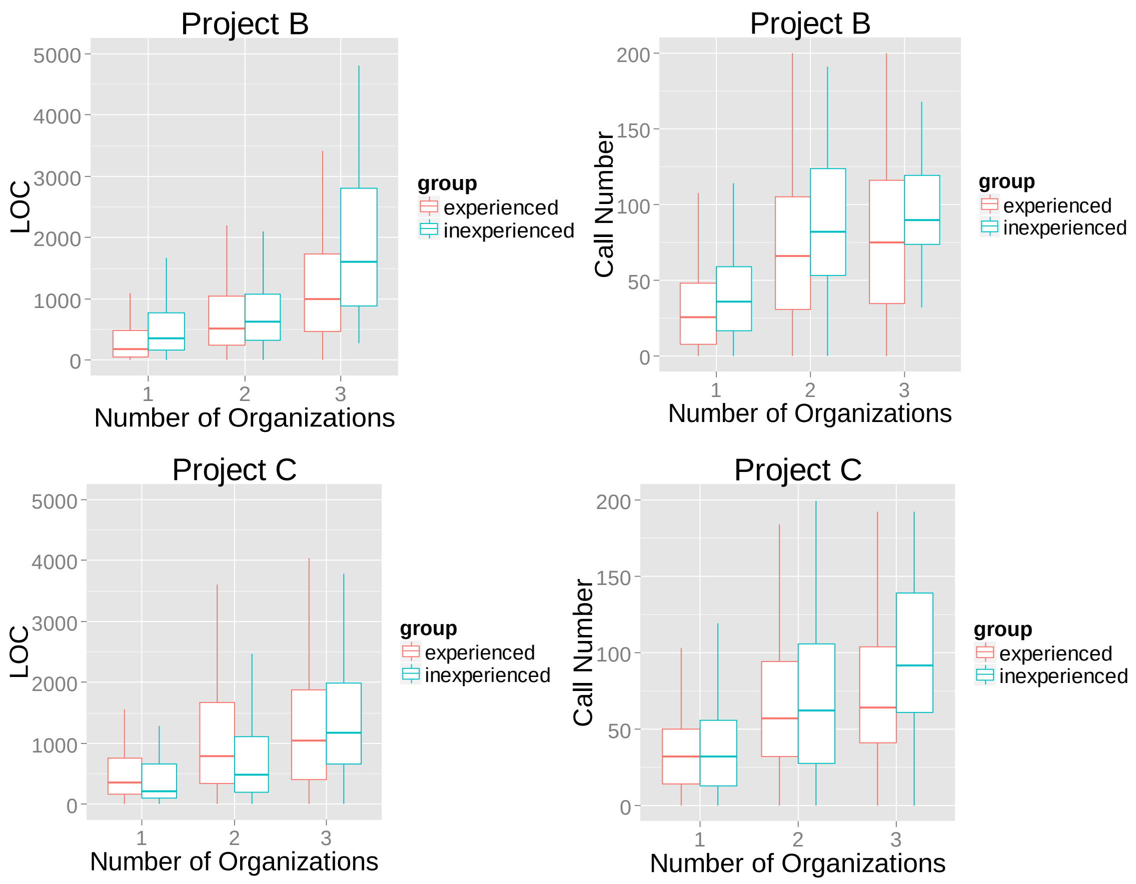

Fig. 6. Product metrics of files edited by several developers

4.6. *RQ2-4 Do experienced developers work on more complicated files when they edit files with the other developers?*

Figure 6 shows the product metrics of files edited by several developers, while Table 4 represents the Wilcoxon rank sum test with the alternative hypothesis set to “inexperienced developers work on higher metrics files in each type of origin”.

Table 4. Wilcoxon Rank sum test in RQ2-4

Project	Number of organizations	LOC	Call Number
B	1	P<<0.01	P<0.05
	2	P<0.05	P<0.01
	3	P<0.01	P<0.05
C	1	0.9	0.15
	2	0.9	0.12
	3	0.06	P<0.05

Experienced developers do not work on more complicated files in all projects. In project B, inexperienced developers worked on the more complicated files.

4.7. *RQ2-5 Do experienced developers contribute more to the files edited by several developers?*

Figure 7 and Figure 8 show how many times experienced and inexperienced developers modified the files edited by several developers compared with the number of total file changes. Table 5 shows the Wilcoxon rank sum test with the alternative hypothesis set to “experienced developers contribute more to files in each type of origin”. There is not a significant difference between experienced and inexperienced developers in project C, but experienced developers contribute more in project B. These results indicate that inexperienced developers introduce fewer defects and work on more complicated files, but they do not contribute much to these files. In project C, the differences in complexity and the contribution of

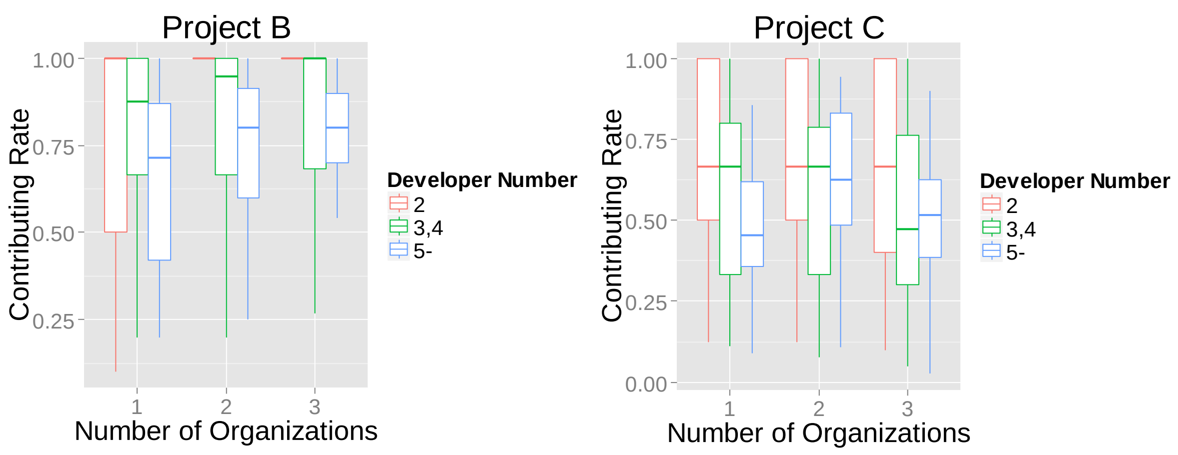

Fig. 7. Contribution rate of experienced developers

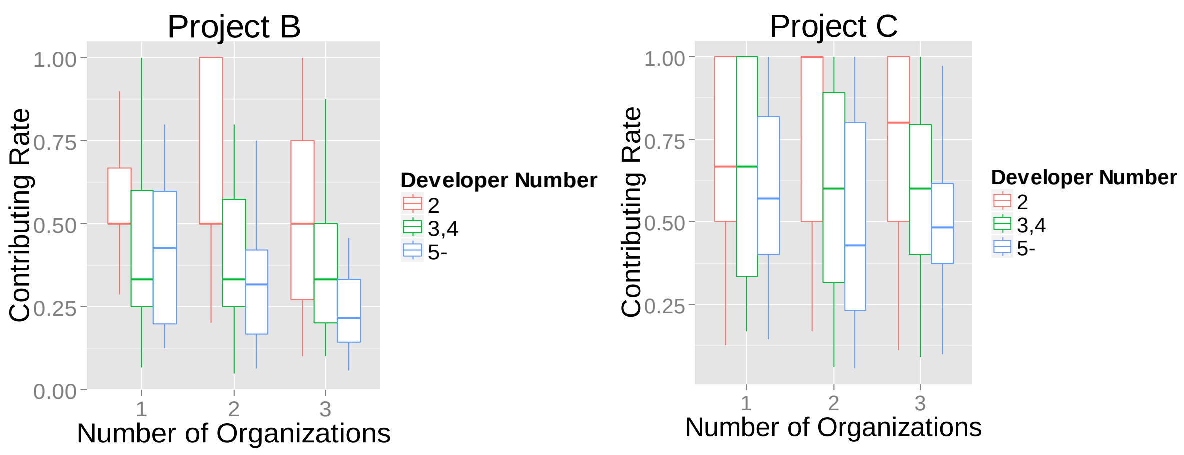

Fig. 8. Contribution rate of inexperienced developers

developer experience are insignificant. Hence, experienced developers introduce fewer defects in some origins.

Table 5. Wilcoxon Rank sum test in RQ2-5

Project	Number of organizations	Contributing Rate
B	1	P<<0.01
	2	P<<0.01
	3	P<<0.01
C	1	P<<0.01
	2	0.9
	3	0.6

4.8. RQ3-1 Do the same developers introduce many defects in each project?

Figure 9 shows the number of defects by developer who participated in the projects continuously. The x-axis means the developer, while the y-axis shows the number of introduced defects. If developer X fixes two files to remedy a bug and both files are written by developer Y, we consider that developer Y introduced one defect. This graph indicates that developers, who did not introduce or only introduced a few defects, introduced many defects in the next projects. Therefore, introducing many defects in a previous project is not an indicator of introducing many defects in the next project.

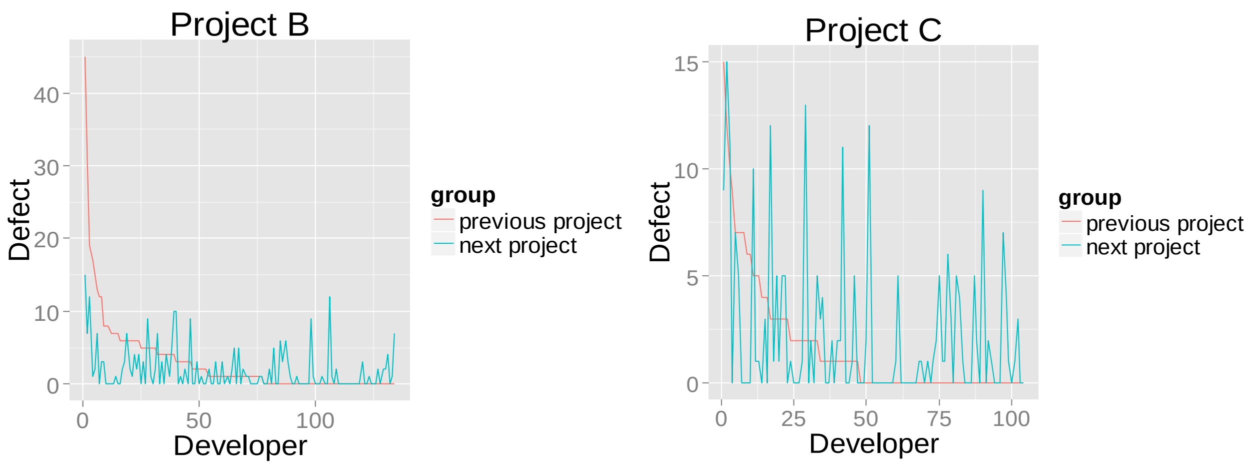

Fig. 9. Transition of the developer introducing defects

4.9. RQ3-2 Does a developer who introduces many defects in all project work on complex parts?

In the three projects, most developers did not consistently introduce many defects. However, “one developer” introduced many defects in all projects, and was the most defective developer in projects A and B. He was responsible for the inherited files with the highest metrics values for both LOC and Call Number in all projects. These files introduced many defects in each project. However, as he gained experience, the number of introduced defects decreased. In project C, he introduced about half the number of defects compared to the most defective developer. We speculate that the developer who takes charge of the most complicated files may introduce many defects, but the number of introduced defects decreases as he gains “experience” in the specified domain.

5. Threats to Validity

One threat to internal validity is that only three projects were considered. Although experience in previous projects was considered, we did not consider whether the developer worked in the same domain in the next project. In the future, we need to confirm whether most developers work in similar domains in the next project. Additionally, we examined defects, two metrics, and experience, but these factors only capture part of the code quality, complexity, and experience. In the future, we need to investigate other factors.

A threat to the external validity is that we analyzed three projects executed by a single organization. Although we hypothesize that the results will be applicable to continuous projects executed by other organizations, we need to confirm that the same features appear in other projects.

6. Conclusion and Future Work

This research reveals that experienced developers do not always work on more complex files or introduce fewer defects. However, if inexperienced and experienced developers work on projects with similar complexities, experienced developers tend to introduce fewer defects. Additionally, the most defective developer was responsible for the most complex files, but the number of defects decreased with experience. We did not confirm that experience affects the code quality.

In the future, we want to investigate which domain truly requires reliable knowledge or experience as this should lead to more effective strategies when managing developers.

References

1. R. M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations* (Plenum, New York, 1972), pp. 85–104.

2. Foyzur Rahman, Premkumar Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," Proceedings of the 33rd International Conference on Software Engineering, 2011 pp. 491–500.
3. Audris Mockus, David M. Weiss, "Predicting risk of software changes, Bell Labs Technical Journal, vol. 5, no. 2, 2000 pp. 169-180
4. Christian Bird et al., "Don't Touch My Code! Examining the Effects of Ownership on Software Quality," Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, 2011, pp.4-14.
5. Jon Eyolfson, Lin Tan, Patrick Lam, "Do Time of Day and Developer Experience Affect Commit Bugginess?" Proceedings of the 8th Working Conference on Mining Software Repositories, 2011, pp. 153-162.
6. Pamela Bhattacharya, Iulian Neamtiu, Michalis Faloutsos," Determining Developers' Determining Developers' Expertise and Role: A Graph Hierarchy-based Approach", Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME 2014), 2014, pp. 11-20.
7. Reou Ando et al., "How Does Defect Removal Activity of Developer Vary with Development Experience?" Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE 2015), 2015, pp. 540-545.
8. Ekrem Kocaguneli, Ayse T. Misirli, Bora Caglayan, Ayse Bener, "Experiences on Developer Participation and Effort Estimation" Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011, pp. 419 – 422.
9. Shinsuke Matsumoto et al., "An Analysis of Developer Metrics for Fault Prediction," Proceedings of the 6th International Conference on Predictive Models in Software Engineering, No.18, 2010
10. Thomas J. Ostrand, Elaine J. Weyuker, Robert M. Bell, "Programmer-based Fault Prediction," Proceedings of the 6th International Conference on Predictive Models in Software Engineering, No.19, 2010.
11. Jacek Sliwerski, Thomas Zimmermann, Andreas Zeller, "When Do Changes Induce Fixes?" Proceedings of the 2005 international workshop on Mining software repositories, 2005 pp.1-5.
12. Sunghun Kim, Thomas Zimmermann, Kai Pan, E. James Whitehead, "Automatic Identification of Bug-Introducing Changes", Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, 2006, pp. 81-90.
13. Seiji Sato et al., "Effects of Organizational Changes on Product Metrics and Defects," Proceedings of the 20th Asia-Pacific Software Engineering Conference, 2013, pp.132-139.
14. Renuka Sindhgatta, "Identifying domain expertise of developers from source code", Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, 2008, pp. 981-989.
15. Audris Mockus, "Organizational volatility and its effects on software defects," Proceedings of the 18th ACM SIGSOFT international symposium on Foundations of software engineering, 2010, pp. 117–126.
16. Mitchell Joblin et al., "From Developer Networks to Verified Communities: A Fine-Grained Approach" Proceedings of the 37th International Conference on Software Engineering - Volume 3, 2015 pp. 563-573
17. Osamu Mizuno et al., "Fault-Prone Module Prediction Approaches Using Identifiers in Source Code" International Journal of Software Innovation - Volume 3, 2015 pp. 36-49
18. Golnoush Abaei et al., "Increasing the Accuracy of Software Fault Prediction using Majority Ranking Fuzzy Clustering" International Journal of Software Innovation - Volume 2, 2014 pp. 60-71
19. Taketo Tsunoda et al, "Evaluating the work of experienced and inexperienced developers considering work difficulty in sotware development" 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2017, pp. 161-166.