Comparison of Developer's Work Efficiency between Different Editors

Sentaro Onizuka[†], Tetsuya Kanda[†], Katsuro Inoue[‡]
[†]Graduate School of Information Science and Technology, Osaka University, Osaka, Japan
Email: {s-onizuk, t-kanda}@ist.osaka-u.ac.jp
[‡]Faculty of Science and Technology, Nanzan University, Aichi, Japan, Email: inoue599@nanzan-u.ac.jp

***Abstract*—In this paper, as an example of comparing developer's work efficiency between different editors, we propose a method to collect and compare self-evaluations and quantitative evaluations of developer's work efficiency for each editor. We also practiced the proposed method on Visual Studio Code and Eclipse, and confirmed the applicability of the method.**

***Index Terms*—Coding Process, Fine-grained Edit History, Work Efficiency, Editor**

I. Introduction

Although many studies have analyzed the coding process, most studies have been limited to a single development environment, and few have compared the coding process across different development environments. However, when development environments are limited, the proficiency of the development environment may affect the results of the analysis. For this reason, it is important to develop methods to compare coding processes in different development environments.

However, to compare the developer's work efficiency between different editors, it is necessary to collect the edit history from each editor under the same conditions. In addition, more granular histories than those recorded in version control systems can provide a more detailed understanding of the coding process [1]. Therefore, we extended a fine-grained edit history collection platform proposed by Ishida et al. [2]. The fine-grained edit history in this platform is a record of all edit operations of the source code by the developer on the editor. In this platform, the history collection module is divided into a language server and editor plug-ins so that only editor plug-ins need to be newly developed for each editor. The infrastructure of the history analysis part is implemented as a server, and the server provides an API for retrieving the history, making it easy to use the history with various tools.

Editors are an essential element in the coding process and are important in influencing the efficiency of development. In addition, each developer has subjective evaluations of each editor, such as strengths and weaknesses. However, such subjective evaluations do not always match actual work efficiency. Therefore, there might be room for improving developer's development efficiency by finding a more efficient editor.

In this paper, as an example of comparing coding processes between different development environments, we propose a method to compare developer's work efficiency between different editors by collecting and comparing self-evaluations and quantitative evaluations of developer's work efficiency for each editor. We practice the method on Visual Studio Code and Eclipse, and confirm that the method is applicable in practice.

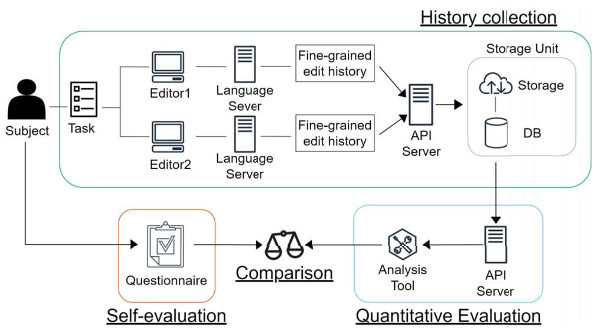

Fig. 1. An Overview of the Proposed Method

II. Proposed Method

A. Overview

We propose a method to collect and compare "self-evaluations" and "quantitative evaluations" of developer's work efficiency for each editor. The proposed method compares two editors and is applicable to any editor that supports Language Server Protocol [3], which is used in the history collection part of the platform. A schematic diagram of the proposed method is shown in Figure 1. Subjects are asked to perform small tasks of similar estimated working time with two editors and to answer questionnaires about the self-evaluations. Quantitative evaluations are collected through the tasks and compared with self-evaluations.

B. Self-evaluations

Self-evaluations are collected by taking questionnaires before and after the task. Specifically, developers are asked to answer the question, "Which editor do you think is faster at coding?" on a five-point scale from "Editor 1 is very fast" to "Editor 2 is very fast". The survey results are then converted to a numerical value on a scale of -2 to $+2$, with the higher value indicating that Editor 1 is faster and the lower value indicating that Editor 2 is faster.

C. Quantitative evaluations

Quantitative evaluations are based on the fine-grained edit history collected through the task process. To collect histories

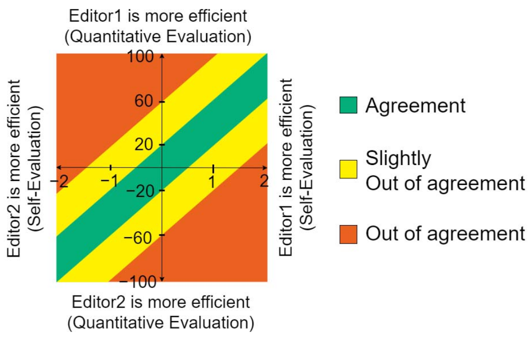
Editor1 is more efficient
(Quantitative Evaluation)
Editor2 is more efficient
(Self-Evaluation)
Editor1 is more efficient
(Self-Evaluation)
Editor2 is more efficient
(Quantitative Evaluation)
100
60
20
−20
−60
−100
−2
−1
1
2
Agreement
Slightly
Out of agreement
Out of agreement

Fig. 2. Meaning of each area in a scatter plot

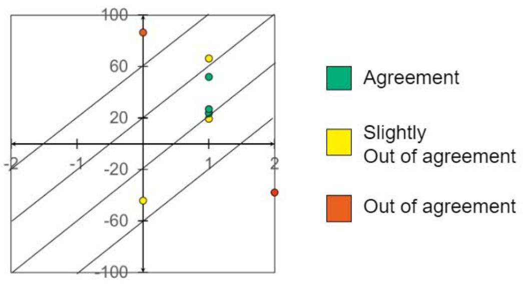

Fig. 3. A scatter plot of comparing post-development self-evaluations and quantitative evaluations of a source code transcription task

from multiple editors under the same conditions, the platform described in section I is used. We extended the platform to provide a history API so that we can easily access the collected history thereafter.

The data of the time required for each task in the progress graph are used for evaluations. Specifically, the time required for each task is normalized from a minimum value of 0 to a maximum value of 1. The difference between the average of normalized time in editor 1 ($avg1$) and editor 2 ($avg2$) is calculated as "$-(avg1 - avg2) \times 100$". The range of value calculated here is -100 to $+100$, where a larger value means that editor 1 is more efficient and a smaller value means that editor 2 is more efficient.

Using the collected history, we can conduct various visualization of the development process. For example, visualizing changes in the development progress rate over time based on the edit distance to the files in their final state at each time.

D. Comparison

Scatter plots are created based on the values of self-evaluations and quantitative evaluations, and the scatter plots are used to determine whether they are in agreement or not. Each point on the scatter plot corresponds to a developer's self-evaluations and quantitative evaluations. The x-axis is the self-evaluation and the y-axis is the quantitative evaluation. Since the self-evaluation is conducted twice, before and after the task, we can create two types of scatter plots: "comparison of pre-development self-evaluations and quantitative evaluations" and "comparison of post-development self-evaluations and quantitative evaluations".

Two criteria are used to evaluate the agreement between the self-evaluations and the quantitative evaluations based on the scatter plots: one is the correlation coefficient of the scatter plots, and the other is the classification by area. The scatter plots are divided into three areas as shown in Figure 2, and the agreement is judged by which area each point is located.

III. Method Example

We conducted a small experiment with the proposed method on Visual Studio Code and Eclipse. According to the proposed method, we first developed plug-ins for each target editor to collect history from different environments. We also developed a tool to visualize the time-series change of progress rate based on the collected history using the API provided by the platform. Eight participants were asked to do two tasks: the first task required them to answer programming problems of similar difficulty, and the second task required them to transcribe source code. Self-evaluations are collected by taking questionnaires before and after the task. Finally, we created scatter plots for comparison based on self-evaluations and quantitative evaluations.

As an example of the output obtained by the method, Figure 3 shows a scatter plot comparing post-development self-evaluations and quantitative evaluations of a source code transcription task. The correlation coefficient calculated from the plot is -0.27, indicating that there is no positive correlation. Also, regarding the self-evaluations and quantitative evaluations, three respondents were in agreement, three were slightly out of agreement, and two were not in agreement, indicating that only a small number of respondents were in agreement. These results indicate that there is a gap between the post-development self-evaluations and the quantitative evaluations of the source code transcription task.

IV. Conclusion

We proposed a method to realize the comparison of developer's work efficiency between different editors. We showed an example output of the proposed method with two editors.

In future work, we need to improve the method of quantitative evaluations. Currently, only information on time spent coding is used, hence it is planned to consider information on the coding process to more accurately evaluate work efficiency.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers JP19K20239 and JP18H04094.

References

[1] S. Negara, M. Vakilian, N. Chen, R. E. Johnson, and D. Dig. "Is it dangerous to use version control histories to study source code evolution?," In Proc. of ECOOP2012, pp. 79–103, 2012.
[2] N. Ishida, T. Kanda, K. Shimari, and K. Inoue. "Concept of Fine-Grained Edit History Collection Platform Using Language Server"", SES2020 Workshop 5, no.5, September 2020 (in Japanese).
[3] "Language Server Protocol," Microsoft, 2022. Accessed: October 12, 2022. Available: https://microsoft.github.io/language-server-protocol/