A Study of an Automated Software Effort Measurement Method

Yeong-Seok Seo(✉) and Hyun-Soo Jang

Department of Computer Engineering, Yeungnam University, 280 Daehak-Ro, Gyeongsan, Gyeongbuk 38541, Republic of Korea
ysseo@yu.ac.kr

Abstract. Software companies have adopted project management methodologies suitable for their organizations and have made significant efforts in successfully applying them to improve the quality of software. In particular, a technology that can measure and analyze software project data is essential for effective project management and productivity improvement. Of these software project data, software effort is the key metric to be measured, given its direct relation to process improvement and quality but also have general management interest. However, in practice, there have been many difficulties in actually measuring effort data because of problems in continuous and consistent measurement. Therefore, in this paper, we propose an automated software effort measurement method that can apply during the entire software development life cycle, to overcome these problems and to achieve improvement of effort measurement outcomes. Experiments are performed to evaluate the proposed method from the viewpoint of effort measurement accuracy. The results indicate that the proposed method shows a significant improvement compared to the existing methods.

Keywords: Effort · Measurement and analysis · Software project management Software quality · Software tools and environments

1 Introduction

Software measurement and analysis can lay the foundation for effective software project management and can also have a significant impact on an organization's software development productivity [1]. Of the various software project data, the software development effort (note that "effort" is the amount of work units (time) required to complete any given task) is one of the most important indicators [2]. Based on the collected effort, we estimate the software development effort at the planning stage and perform systematic project management through the allocated effort by tasks. In order to conduct these activities, we should basically be able to measure effort accurately with less trouble. If accurate effort measurement is possible, then from an individual perspective, the effective planning and processing of each assigned task are possible, deficiencies in individual processes can be identified, and opportunities for improving these deficiencies can be provided when performing a software development project [3, 4]. From an organizational perspective, software project managers can easily

J. H. Park et al. (Eds.): PDCAT 2018, CCIS 931, pp. 371–380, 2019.
https://doi.org/10.1007/978-981-13-5907-1_40

identify and monitor the overall progress, and reliable effort estimation is possible through historical data, which will directly affect the quality of the final product [5, 6].

Many companies have been making or attempting effort measurement owing to its importance. However, the reliability of the collected effort is quite low because developers record their individual effort by directly measuring it through additional efforts other than development, or the effort collection process is not automated. Even with the existing effort measurement methods, developers should make further efforts or there are problems with accuracy. The existing effort measurement methods can be divided into a self-report method in which developers directly record the effort data and an automatic-report method using a software program. The self-report method has the possibility of fabrication because each developer has to manually enter his/her own effort or review it. Even for the automatic-report method, it is impossible to apply it to the whole stages of the software development cycle. Further, it is very difficult to collect accurate effort data because the method does not take into account "the time for thought" to solve problems and to find the necessary key information during development. Additional efforts may be required from developers to adjust this information, which may affect primary tasks assigned to them.

Therefore, in this paper, we propose a method that enables software developers to measure effort more accurately and automatically without additional efforts while performing a software project. This method, which uses Windows Application Programming Interface hooking (Windows API hooking) technology [7, 8], can collect events of working tools and input devices that developers actually use, and measure the actual work time by task, tool, and development phase by analyzing the collected data. In particular, the proposed method can measure effort by considering "the time for thought" to solve problems when developing a software product. In addition, this method can improve the effort measurement accuracy by measuring actual effort through extracting website information when using web browsers. Although web browsers are generally used for a software development, it is a difficult issue to identify the actual work time from them.

The remainder of this paper is organized as follows: Sect. 2 describes existing studies related to effort measurement. Section 3 introduces the effort measurement method proposed in this study. Section 4 shows the experimental results and analysis, and Sect. 5 concludes this paper and suggests future work.

2 Related Work

Research on software effort measurement is largely divided into self-report and automatic-report methods.

The most commonly known self-report methods are Personal Software Process (PSP) [3, 4] and Team Software Process (TSP) [5, 6]. Software developers record their daily effort to use in process improvement. However, there is a limitation in that actual effort must be recorded manually by developers, and thus it is very difficult to guarantee reliability and validity of the collected data. Although a variety of tools have been developed to support manual data collection, such as Process Dashboard [9],

PSP.NET [10], and WBPS [11], they still need to input data manually with additional efforts through a context switch between development and measurement activities.

There are several types of automatic-report methods. The first type is a chunk-based method [12–14]. This method divides 24 h per day into certain time intervals. When an event related to a software task occurs at a specific interval, that interval is measured as valid effort. Although the chunk-based method enables automated effort measurement, it is not very sophisticated since it measures even one software task event at a certain time interval as the entire work time. Further, additional reviews should be made periodically to check if the measured effort by developers is suitable. The review activity puts a lot of burden on developers. The second type of automatic-report method is an interval-based method [15, 16]. It first measures events related to software tasks (e.g., compilation in development support tools), and then measures the final effort by developing a discriminant to identify actual effort within the intervals between events. This method may be more automated than the chunk-based method, yet it can only be applied at the implementation stage where compilation is necessary but not the whole software development life cycle. The interval-based method can be used in combination with the self-report method. The interval-based method measures actual effort according to effort decision formulas. Here, if there is any additional work time recorded by developers, it can be included as work time. In this way, the final effort is drawn by combining the effort collected through the interval-based method and the effort recorded by an individual developer. The last type is a task-based method [17]. The method is developed as a software development task measurement system, called as TaskPit. It can record the time spent for each task such as programming, testing, and documentation, by binding between tasks and applications. In addition, it can record the amount of work and the amount of deliverables of each task. TaskPit is helpful for effort measurement and process improvement in a development task level. However, this mainly focuses on grasping how developers are working, rather than achieving accurate effort measurement by identifying and analyzing the actual work time in detail.

3 Personal Effort Measurement System (PEMS)

The proposed automated personal effort measurement method is named the personal effort measurement system (PEMS). PEMS enables the members participating in a software development project to measure effort, that is, actual work time, by analyzing working activities on each computer while working on their computers. To this end, this study applied API hooking provided by the Windows operating system (OS). The Windows OS is used by many people and is run by events as an event-driven OS. Here, hooking refers to a mechanism that can identify and monitor all events such as messages, mouse movements, and keystrokes occurring in programs such as operating systems and application software [7, 8]. If we create a function that acts as a hook for an application program running on the Windows operating system, we can extract the process names of the program or events of input/output (I/O) devices occurring in the program.

The overall approach of PEMS using these characteristics is shown in Fig. 1. In the following subsections, we present each step of PEMS in more detail.

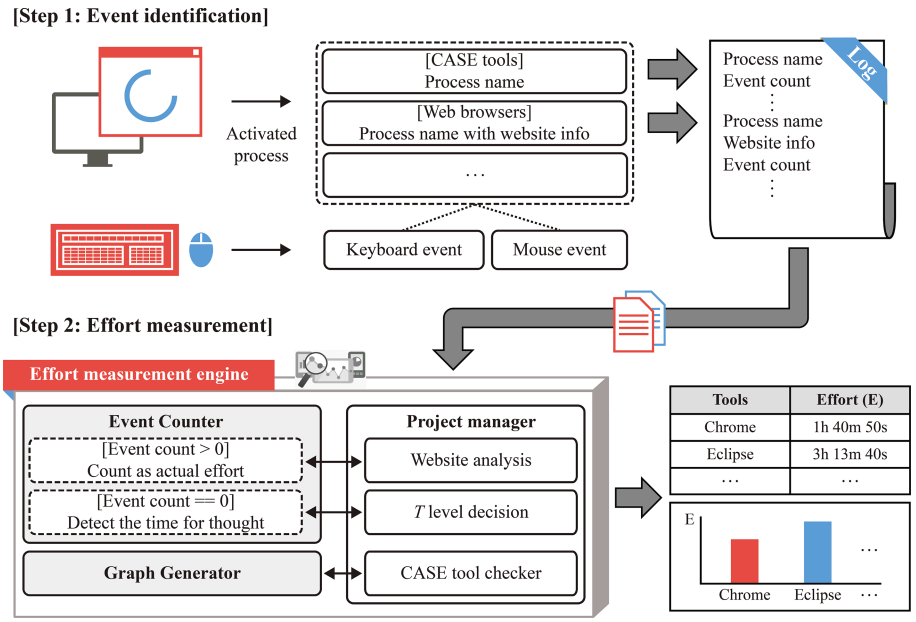

Fig. 1. Overall approach

3.1 Step 1: Event Identification

Step 1 is the stage of collecting the keyboard and mouse events generated from various software tools that software project participants (named as developers) use while working on a computer, and recording the information in a log file.

Developers use various software tools related to specific tasks assigned to them. Of the various tools that developers run on the computer for their tasks, PEMS identifies the process names of the tools that the developers are currently using. In other words, PEMS identifies the process names of the tools currently activated by the developers. Keyboard or mouse events are generated to activate and use the tools, and PEMS also identifies the counts of these events. These identified process names and event counts are recorded in the log file every second.

As shown in Fig. 1, process names and event counts can be collected as mentioned above to measure effort for the usage time of various Computer-Aided Software Engineering (CASE) tools that are used as general working tools at each stage of the software development process. Further, even for tools that each software development group wants to additionally include in effort measurement through discussion, project managers can input them into PEMS to collect process names and event counts. Tools that are not specified as working tools (tools not related to an assigned software project task) are recorded in the log file as NULL because they are not related to effort measurement.

For web browsers, unlike CASE tools, the title bar information of web browsers is recorded in the log file in addition to process names and event counts. Using web browsers can be an activity that is not related to an assigned task. However, there is a strong possibility that they can be used in web search activities relevant to the task in many ways. Thus, by collecting the titles of websites as well, PEMS was designed to enable identifying additional effort in the next step to detect the time related to actual work while using web browsers. In the case of web browsers, just like CASE tools, project managers can input work-related website title information to PEMS so that it records websites not related to work as NULL in the log file.

3.2 Step 2: Effort Measurement

Step 2 is a stage of providing the measured effort by minutely analyzing the log file obtained through Step 1.

As shown in Fig. 1, successive computations and algorithms are constructed in the form of an effort measurement engine to automate the analysis of the input log file. When the log file is inputted, Event Counter continuously analyzes keyboard and mouse event counts occurring for each process recorded in the log file every second, and derives the measured effort.

The entire pseudocode for effort measurement is shown in Table 1. First, the working tool of the corresponding process is determined to have performed a software project task if the event count of a specific process is greater than 0. Then, one second is assigned to actual effort for the working tool, because the log is recorded in every second, as described in Step 1. In the case of web browsers, although only websites related to the work can be recorded in the log file, it is also necessary to analyze further whether the search engines used for the tasks performed the search for the actual work. Note that the search words used in the general search engines are presented in the title bar of web browsers. Thus, by analyzing the title bar recorded in the log file, the execution time of web browsers is assigned to actual effort if the terms or names related to software project tasks are included in the title bar information.

Next, if event counts for a specific process are equal to 0, it can be determined that developers are usually taking a break without using the working tool of the corresponding process. However, it can also be determined that they are thinking to solve a problem while they are using the working tool for the process. We designate "time for thought" (T) to measure even these occurrences as actual effort. If the event count of the corresponding process is continually 0 for less than or equal to T, the time for the event counts is allocated to actual effort of the working tool of the corresponding process. If the event count of the corresponding process is continually 0 for more than to T, the time for the event counts exceeded T is classified as a break time rather than actual effort. When the developers return from a break and start working again, actual effort is measured again. In practice, T should be customized to obtain the best measurement results in software companies.

The final effort is derived by collecting all effort of the working tool of each process, as presented in Table 1. The derived effort is visualized in the form of graphs and tables by Graph Generator to easily identify effort by each tool, each task, and each development phase. In particular, the working tools that perform similar functions are grouped and represented together by CASE Tool Checker when it effort visualized by

Table 1. Pseudocode for effort measurement

Pseudocode

```
Require: a log file derived from Step 1
Ensure: effort from the log file

 1: Map(process_name, Effort)
 2: i
 3: idx_time_i
 4: cumulative_effort
 5: event_count_i
 6: T

 7: idx_time_1 = 0;
 8: CP = process_name_1;

 9:for i = 1 to EOF do

10: if event_count_i > 0 then
11:   if CP == process_name_i then
12:      idx_time_(i+1) ← idx_time_i + 1;
13:   else
14:      Map(CP, cumulative_effort for CP + idx_time_i);
15:      idx_time_(i+1) ← 1;
16:      CP ← process_name_i;
17:   end if

18: else if (continually)event_count_i == 0 for ≤ T then
19:   idx_time_(i+1) ← idx_time_i + 1;

20: else if (continually)event_count_i == 0 for > T then
21:   Map(CP, cumulative_effort for CP + idx_time_i);
22:   idx_time_(i+1) ← 0;

23: else if i == EOF then
24:   Map(CP, cumulative_effort for CP + idx_time_(i+1));
25: end if

26:end for
```

(Comments)
1: // a Map data structure to save effort
2: // index of (process name, (website info), event count) in the log file
3: // temporary variable to calculate effort of the *i*th process name(*process_name*$_i$)
4: // total effort cumulated by *idx_time*$_i$ (initial value is 0)
5: // the number of keyboard and mouse events for the *i*th process name
6: // the time for thought
8: // *CP* is a current process
9: // *EOF* is End Of File
11: // In case that the existing process name is continued
12: // 1 is one second
13: // In case that the existing process name is changed
18: // Identification of the time of thought
20: // Identification of a break time
24: // *i* in *idx_time*$_{(i+1)}$ is an index value just before *EOF*

tasks and development stages. For example, using the category "Search", the measured effort from various search engines is represented by integration. This allows users to easily and intuitively understand the results of effort measurement at a glance, and to analyze these results from various angles.

4 Experiment

4.1 Experimental Design

In order to verify the performance of PEMS for software development tasks, we conducted a tool development project for algorithmic problem-solving using a hash structure. Experiments were performed on three participants who have one year work experience as software developers in a CMMI level 3 [18] software company. They are skilled in programming languages, such as Java (J2SE, J2EE, J2ME), C#/.NET, Python, HTML5. In our experiment, PEMS were compared with the self-report method, the chunk-based method, and the interval-based method, which are mentioned in Sect. 2. Task-based method could not applied because the specific algorithm for effort measurement was not expressive enough to implement and experiment it.

For the self-report method, apart from the three participants, three more self-report record monitoring personnel were assigned to supervise effort measurement recording of the participants. This is intended to reduce the possibility of bias mainly arises from self-reporting and to improve the reliability of effort measurement results.

For the existing methods, the chunk-based method that is the basis of this study and the interval-based method were implemented as program modules and activated with PEMS during our experiment. The time interval of the chunk-based method was set as one second to make fair comparison with PEMS.

4.2 Experimental Results

The experimental results for participants A, B, and C are shown in Fig. 2. The x-axis of the graph represents the tools each participant used while performing the project. Web browsers used in the search for solving problems while conducting tasks include Internet Explorer, Chrome, and Microsoft Edge. The y-axis of the graph represents effort expressed in seconds. The table below the graph describes the exact numerical values for each graph item. SR is the self-report method, IbM is the interval-based method, and CbM is the chunk-based method.

As shown in Fig. 2, PEMS shows a more accurate effort measurement than the existing methods based on SR. In addition, PEMS shows outstanding results of effort measurement for all tools represented on the x-axis, and the effort differences between PEMS and SR is minimal. Because PEMS can be affected by T, we also present the effort measurement results in the variation of T ($T = 10$ and $T = 15$), as provided in Fig. 2. The results show that the concept of T applied in this study affects the actual effort measurement and the measured effort increases gradually a little as T increases. We additionally analyzed the log file by setting from $T = 5$ to $T = 50$, and the best

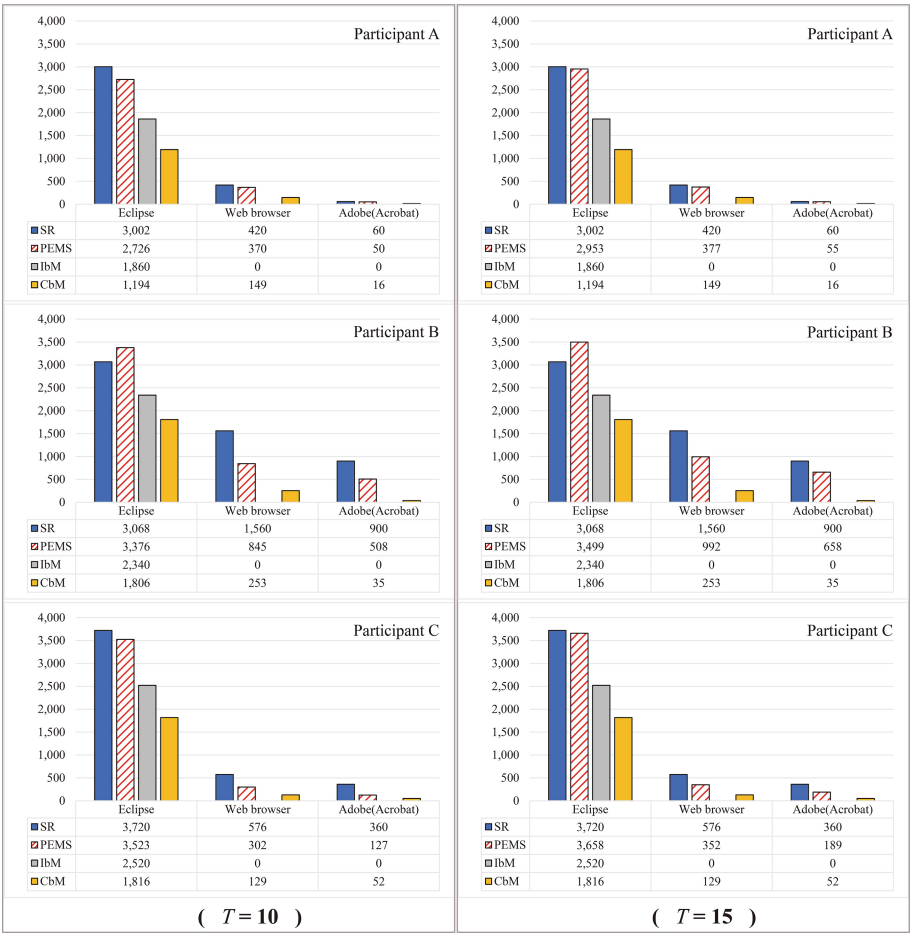

Fig. 2. Experimental results from three participants in the variation of T

T for Eclipse were 5 for the participant B (SR: 3,068, PEMS: 3,059) and 15 for the participant A (SR: 3,002, PEMS: 2,953) and C (SR: 3,720, PEMS: 3,658).

Note that the results of Web browser and Adobe (Acrobat) are zero in IbM. This is because IbM can be only applied for the implementation tools that include a compile function for source codes.

Table 2. Effort measurement accuracy from the participants (%, T = 15)

	Participant A			Participant B			Participant C		
	SR vs. PEMS	SR vs. IbM	SR vs. CbM	SR vs. PEMS	SR vs. IbM	SR vs. CbM	SR vs. PEMS	SR vs. IbM	SR vs. CbM
Eclipse	98.37	61.96	39.77	114.05	76.27	58.87	98.33	67.74	48.82
Web b.	89.76	0.00	35.48	63.59	0.00	16.22	61.11	0.00	22.40
Adobe	91.67	0.00	26.67	73.11	0.00	3.89	52.50	0.00	14.44

Table 2 presents the accuracy rates of effort measurement based on the numerical values represented in Fig. 2. The rate is derived by dividing the measured effort of each method by that of the self-report method. The best accuracy rate of PEMS is 98.37% for Eclipse of Participant A, which is 36.41% and 58.59% higher than the results of the existing methods.

5 Conclusion

Measuring effort accurately is one of the most challenging issues for software project management. Thus, this study proposed PEMS that could automatically measure effort without additional efforts from participants during the entire software development life cycle. Using API hooking, process names and keyboard/mouse events currently running on the computer were identified. Based on the log file that records them, a more accurate effort measurement was possible through Effort measurement engine of PEMS. In particular, we achieved a more accurate effort measurement by reflecting the time for thought in effort measurement and by analyzing the utilization of website information for software development tasks.

Although there were considerable encouraging results in this study, we will conduct further studies on the following issues. By actually applying the proposed method to small- and medium-sized companies, we will investigate the generalization of our results. In addition, we will develop a more accurate and automated effort measurement method on computers that use operating systems other than Windows.

Acknowledgement. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2017R1C1B5018295).

References

1. Sommerville, I.: Software Engineering, 10th edn. Pearson, Boston (2016)
2. Singh, J.: Functional Software Size Measurement Methodology with Effort Estimation and Performance Indication, 1st edn. Wiley-IEEE Computer Society Press, Hoboken (2017)
3. Humphrey, W.S.: PSP(sm): A Self-Improvement Process for Software Engineers, 1st edn. Addison-Wesley Professional, Upper Saddle River (2005)
4. Blokdyk, G.: Personal Software Process: Best Practices Guide. CreateSpace Independent Publishing Platform, USA (2017)
5. Humphrey, W.S.: TSP: Leading a Development Team, 1st edn. Addison-Wesley Professional, Reading (2005)
6. Blokdyk, G.: Team software process: Fast Track. CreateSpace Independent Publishing Platform, USA (2017)
7. Richter, J.: Programming Applications for Microsoft Windows. Microsoft Press, Redmond (1999)
8. Yosifovich, P., Russinovich, M.E., Solomon, D.A., Ionescu, A.: Windows Internals, 7th edn. Microsoft Press, Redmond (2017)
9. Process Dashboard. https://www.processdash.com. Accessed 29 June 2018

10. Nasir, M.H.N.M., Yusof, A.M.: Automating a modified personal software process. Malays. J. Comput. Sci. **18**(2), 11–27 (2005)
11. Thisuk, S., Ramingwong, S.: WBPS: a new web based tool for personal software process. In: 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, pp. 1–6. IEEE, Nakhon Ratchasima (2014)
12. Johnson, P.M., Kou, H., Paulding, M., Zhang, Q., Kagawa, A., Yamashita, T.: Improving software development management through software project telemetry. IEEE Softw. **22**(4), 76–85 (2005)
13. Fauzi, S.S.M., Nasir, M.H.N.M., Ramli, N., Sahibuddin, S.: Software Process Improvement and Management: Approaches and Tools for Practical Development, 1st edn. IGI Global, Hershey (2011)
14. Artemev, V., et al.: An architecture for non-invasive software measurement. In: Petrenko, Alexander K., Voronkov, A. (eds.) PSI 2017. LNCS, vol. 10742, pp. 1–11. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74313-4_1
15. Hochstein, L., Basili, V.R., Zelkowitz, M.V., Hollingsworth, J.K., Carver, J.: Combining self-reported and automatic data to improve programming effort measurement. In: 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 356–365. ACM, New York (2005)
16. Hochstein, L., Basili, V.R., Vishkin, U., Gilbert, J.: A pilot study to compare programming effort for two parallel programming models. J. Syst. Softw. **81**(11), 1920–1930 (2008)
17. Suthipornopas, P., et al.: Industry application of software development task measurement system: TaskPit. IEICE Trans. Inf. Syst. **E100.D**(3), 462–472 (2017)
18. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI for Development, 3rd edn. Addison-Wesley Professional, Amsterdam (2011)