PAPER

Industry Application of Software Development Task Measurement System : TaskPit

Pawin SUTHIPORNOPAS[†], Pattara LEELAPRUTE[†], Akito MONDEN[††], Hidetake UWANO[†††], Yasutaka KAMEI[††††], Naoyasu UBAYASHI[††††], Kenji ARAKI[†††††], Kingo YAMADA[††††††], *and* Ken-ichi MATSUMOTO[*],

SUMMARY To identify problems in a software development process, we have been developing an automated measurement tool called TaskPit, which monitors software development tasks such as programming, testing and documentation based on the execution history of software applications. This paper introduces the system requirements, design and implementation of TaskPit; then, presents two real-world case studies applying TaskPit to actual software development. In the first case study, we applied TaskPit to 12 software developers in a certain software development division. As a result, several concerns (to be improved) have been revealed such as (a) a project leader spent too much time on development tasks while he was supposed to be a manager rather than a developer, (b) several developers rarely used e-mails despite the company's instruction to use e-mail as much as possible to leave communication records during development, and (c) several developers wrote too long e-mails to their customers. In the second case study, we have recorded the planned, actual, and self reported time of development tasks. As a result, we found that (d) there were unplanned tasks in more than half of days, and (e) the declared time became closer day by day to the actual time measured by TaskPit. These findings suggest that TaskPit is useful not only for a project manager who is responsible for process monitoring and improvement but also for a developer who wants to improve by him/herself.

key words: *Processs measurement, System development, Case study*

1. Introduction

To promote the idea of "Process Improvement via Measurement" in software development, we developed a software development task measurement system called "TaskPit" in 2008 [1], and have been updating it ever since. TaskPit automatically records the time and the amount of work of an individual developer or a development team when performing daily tasks, to identify any problems in the software development process. It can record the time spent for each task such as "Programming" using Eclipse or Visual Studio, "Documentation using Word or other text editors, "E-mail" using Gmail on a browser and so on, where a task is associated with a set of applications and window titles. TaskPit records the amount of work for each task in terms of the number of keystrokes and mouse clicks. It can also record the amount of deliverables of each task as the increase in file size in a directory associated with the task.

To date, the TaskPit community has been gradually growing; user manuals and related tools have been developed by volunteers, and as of July 2015, total downloads of TaskPit version 1.0.0 to 1.0.3 has reached 1500 [1]. Now we are asking for real-world case studies to share the experience and findings about how TaskPit could be used for process improvement.

The goal of this paper is to demonstrate how the task measurement can be used to monitor daily tasks of a software development and to identify possible concerns to be improved. This paper introduces the system requirements, design and implementation of TaskPit; then presents real-world case studies of applying TaskPit in two software organizations. The first organization consists of 12 members, measurements were taken during a 9 day period (6 business days) where 7 are developers, 3 leaders, and 2 customer service agents. Analysis of the measurement data was done by a QA specialist in a different division of a same company in which measurement was taken. In the second organization, one developer was being measured for a period of 17 days (13 business days). In addition to TaskPit's automatic measurement, the planned time and self reported time of each development task for each day was also recorded. This will help in understanding the difference between the self reported time and the actual time measured by TaskPit.

This paper extends our Japanese workshop (short) paper [2] with an additional (second) case study. We also added in this paper explanations of the system requirements, design and implementation of TaskPit to clarify the design concept of TaskPit and to illustrate how TaskPit can be used in software organizations.

In the following Section 2, we will describe related work and the research background, Section 3 will be about TaskPit and its system requirements, design, and implementation. Section 4 will show measurement results and analysis in two organizations which will be split into two sub sections. Lastly in Section 5 is a summary.

2. Background and Related Work

As represented in Tom DeMarco's famous quote "You can't control what you can't measure," [3] we believe measurement is essential in the control and improvement of software development processes. For this purpose, various product process metrics have been proposed, and applied in actual software development [4] [5]. In many projects, metrics such as software size, development hours, the number of

[†]Department of Computer Engineering, Kasetsart University
[††]Okayama University
[†††]National Institute of Technology, Nara College
[††††]Kyushu University
[†††††]NCS&A Co., Ltd.
[††††††]FineBus Co., Ltd.
[*]Nara Institute of Science and Technology
DOI: 10.1587/transfun.E01.A.1

bugs are measured and used for project management and quality assurance [6] [7].

On the other hand, the majority of causes of software failure are human factors [8] [9] [10]. Therefore, in addition to measure the software products and/or processes, it is natural to measure the developers and their works to improve the process. One widely known method is the Personal Software Process (PSP) and Team Software Process (TSP), which record information on the daily tasks of the developer or development team for use in process improvement [11] [12]. However, as developers have to do this manually, the barrier for its adoption is high, and it has not been used widely.

When the PSP had been first proposed, it required measurement data to be input in to specific paper based forms, which required great cost in performing data measurement. Therefore, tools that assist in data input and calculation have been proposed such as Process Dashboard [13], Task Coach [14] and Slim Timer [15]. However, they still require human effort in measurement with a context switch between development and measurement tasks, which presented a barrier towards adoption in actual development projects [16].

Automatic measurement tools that do not require a context switch have been proposed, for example hackystat [17], EPM [18] [19] and Ginger2 [20]. These can be used to collect detailed data at a lower cost. However, the data collected by these tools are not directly associate with developers' tasks such as "programming" and "testing", and also, the recorded data are too much in detail, which require signifiacant amount of time for the analysis. The data required for process improvement based on PSP/TSP are: (1) the time spent on each task; (2) the outcome of each task (e.g. number of bugs found); and does not require detailed data. While the detailed data collected by hackystat and Ginger2 enable more detailed analysis, it also presents a weakness in increased time required for the analysis.

In this paper, we aim to present TaskPit as a tool to introduce process improvement in a development task level. The advantage of TaskPit is that it requires no pre-planning and easy to adopt. It can automatically record the time spent on each task and the output of each task together with its change over time. The data measured by TaskPit represents the amount of work and outcome of each task and is easy to analyze. Hence, it is suitable for the introductory stages of implementing process improvement methods such as PSP/TSP.

3. TaskPit

3.1 System Requirements

Based on the discussion in the previous section, the system requirements of TaskPit are defined as follows.

3.1.1 Requirement 1: Binding between Tasks and Applications

In TaskPit system, we consider that each development task to be a work performed in different applications or windows. Therefore, we need to bind a task name with its corresponding application name (the executable file name or process name). However, it is not always the case that performing a task uses only one application. So it should be possible to assign multiple applications to a task. Additionally, different tasks may use the same application. In this case, we distinguish the task based on the window name during the execution of the application. The extended BNF for defining tasks is as follows.

⟨Task⟩ ::= ⟨Application⟩ { | ⟨ Application ⟩}
⟨Application⟩ ::= ⟨Executable File Name⟩ [⟨Window Name⟩]

Tasks are defined as a set of one or more running applications, and applications are defined as a set of executable name and window name. Below shows an example of task definitions.

Mail ::= Outlook | Gmail with IE | Gmail with Chrome
Programming ::= Eclipse | Visual Studio
Web Browsing ::= Internet Explorer
Outlook ::= OUTLOOK.EXE
Gmail with IE ::= IEXPLORE.EXE Gmail
Gmail with Chrome ::= CHROME.EXE Gmail
Eclipse ::= ECLIPSE.EXE
Visual Studio ::= DEVENV.EXE
Internet Explorer ::= IEXPLORER.EXE

TaskPit assumes that a developer performs just one task at a time, i.e. tasks are not overlapping each other. In this example, the task "Mail" is associated with OUTLOOK.EXE, IEXPLORE.EXE with a window name "Gmail" and CHROME.EXE with a window name "Gmail". This means, if a developer is using the Internet Explore and a text string "Gmail" is included in its window title, then TaskPit considers that the developer is performing the task "Mail". Note that in this example IEXPLORE.EXE also appears in the task definition of "Web Browsing" without any window title specified. In this case, TaskPit considers that a developer is performing the task "Web Browsing" if he/she is using the Internet Explore and its window title does not include its associated strings indicated in other task definitions ("Gmail" in this case.)

We assume that these task definitions should be determined before starting the measurement as clearly as possible, e.g. by asking developers about which tasks they are performing and which applications they are using to work on each task. We decided to use such a definition-before-measurement policy rather than definition-after-measurement because (1) it is often the case that a data analysis is performed several weeks/months after the measurement, and in such a case (2) it is often very difficult to

Table 1 Example of output of TaskPit

Task name	start time	end time	left click	right click	keystroke
File Operation	20150204:095229	20150204:095235	3	0	0
Spreadsheet	20150204:095235	20150204:095256	1	0	0
File Operation	20150204:095256	20150204:095319	6	1	0
Programming	20150204:095319	20150204:095340	3	0	10
Programming	20150204:095340	20150204:095411	3	0	15
File Operation	20150204:095411	20150204:095415	2	0	0
Programming	20150204:095415	20150204:095455	2	0	0
Programming	20150204:095455	20150204:095612	2	0	10

contact and ask developers about tasks and applications they were using.

At the same time, it is possible to start measurement by TaskPit with a standard (default) task definition, and conduct detailed task analyses after measurement with help of developers being measured. For this purpose, TaskPit provide a functionality to record law log data of executed applications and their window titles.

3.1.2 Requirement 2: Automated Data Measurement

• Task Execution Time

The start and end time of using each application is recorded, where an application is considered *in use* if the window is currently in focus. However, even if a window is in focus, if a fixed amount of time (e.g. 10 minutes) has passed without activating computer input (mouse or keyboard), it will be recorded that no application is in use.

• Amount of Work

Amount of work is recorded as the number of key strokes and mouse clicks.

Table 1 shows an example of tasks and their amount of work measured by TaskPit.

• Amount of Deliverables

The outcome (deriverables) of each task is defined as the combination of a directory and one or more file extensions. TaskPit periodically scans the files in the specified directory and its subdirectories, and records the number of files and the total file size. The extended BNF for defining the outcome is as follows.

⟨Outcome⟩ ::= ⟨Directory⟩ ⟨Extension⟩ {⟨ Extension ⟩}

An example of outcome definition in TaskPit is shown below.

Requirement analysis documents ::=
C:\Users\monden\desktop\development\SRS doc tex txt
Program source files ::=
C:\Documents and Settings\monden\desktop\development\sources c cpp java

• Privacy

Especially when recording team activities, it is necessary to protect personal information and respect the privacy of developers. In the proposed system, the key stroke contents, application contents, window names, file names and file contents are not recorded. Additionally, recording the complete history of all applications and windows will make the user feel like being "monitored" which may cause resistance in adopting the system, so the applications and windows not bound to a task in Requirement 1 will be recorded together as "Other tasks".

However, before starting measurement for process monitoring and improvement, one may need to conduct a pilot measurement to determine task definitions. Also, one may want to use the definition-after-measurement policy. For these purposes, TaskPit optionally provides a functionality to record all executed application names and their window titles.

3.1.3 Requirement 3: Team measurement

Data measured for individual developers in their computers are gathered as a team data in a specific computer to let a manager (or a data analyst) enable analyzing the data.

3.1.4 Requirement 4: Data visualization and analysis

The recorded data can be visualized as a graph for an individual developer or a team. The time spent on each task and total time during the specified period is shown as a bar chart or a line chart. The data visualization tool uses the task log file from TaskPit (such as in Table 1), compiles it into a

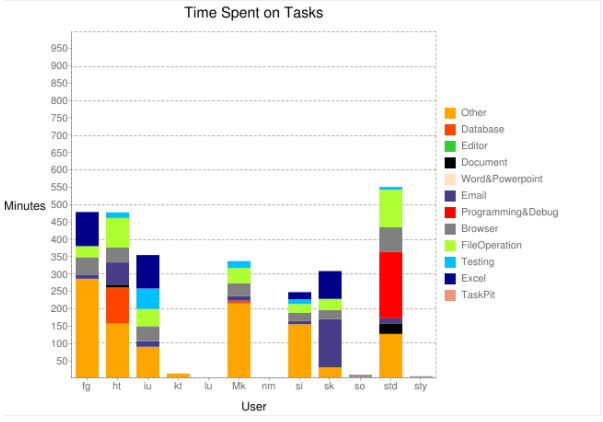


Fig. 1 Total time spent on tasks

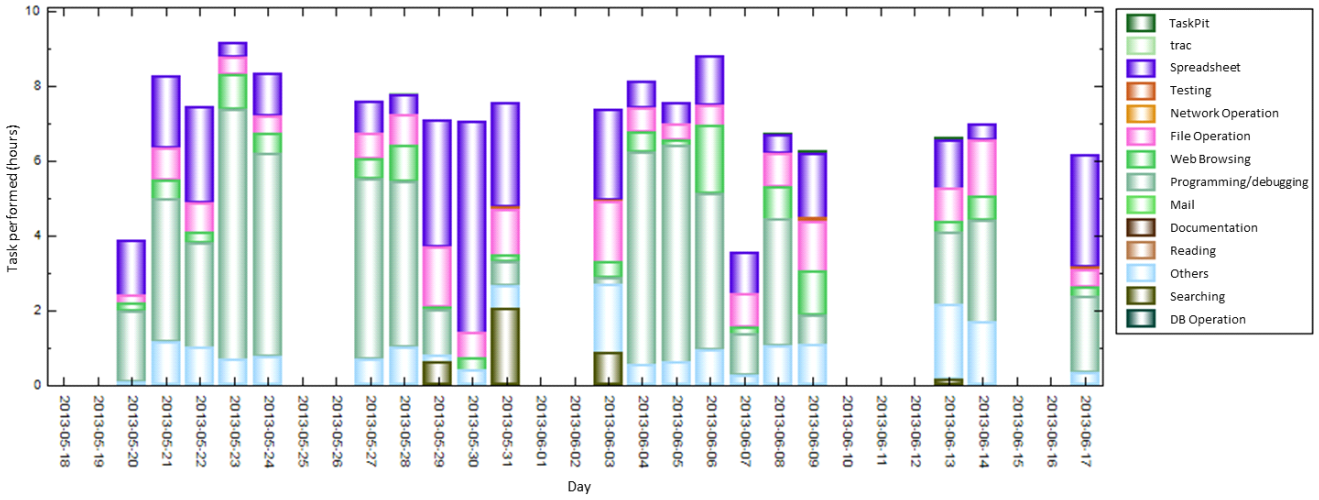


Fig. 2 Time series data of tasks

pre-processed version, and shows the result as a graph. Examples of the graphs are shown in Figure 1 and 2.

Figure 1 describes the total time spent on tasks in a given time frame, separated by each individual in the team with each task indicated with different colors. This figure can show task contribution in the big picture of the team, in which every data is gathered from TaskPit log file.

Figure 2 is a time series visualization of work hours of a specified developer. This figure will describe how a developer performed tasks in each day.

Also, to make analysis easier, TaskPit outputs a summarized log file, which consists of the number of keystrokes, mouse clicks, or the time spent for using each application in a given interval time (e.g. 10 minutes. This interval time is given in a configuration file). Table 2 shows an example of summarized log for the time spent for using each application. As shown in the table, only 6 lines of log data (in CSV format) are output in every 60 minutes.

3.2 System Design

As shown in Figure 3, TaskPit consists of a measurement component which is connected to a database, a visualization component, a configuration file, a log file and daily report files. In Figure 3, arrows indicates input/output relations. The configuration file contains the definitions of tasks and outcome, as well as the interval of outputting to the log file. The measurement component GUI contains a "Task" tab showing the accumulated time spent, key strokes and clicks for each task (see Figure 4), and a "File" tab showing the number of files and file size. The measurement results are output to the log file periodically (e.g. every 10 minutes).

To satisfy Requirement 3, a client-server architecture was considered, but client-server systems require non-trivial configuration and maintenance cost, so instead, we forgo the server and output the measurement data in a specified *shared* folder (or the user's local directory when using individually) in a log file. The visualization component will take the configuration file and shared folders as input, and display the data for a specified period, using various charts. As depicted in Figure 5, the measurement component is run on each de-

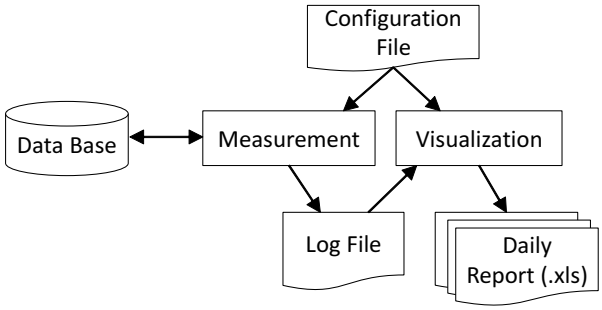


Fig. 3 System components

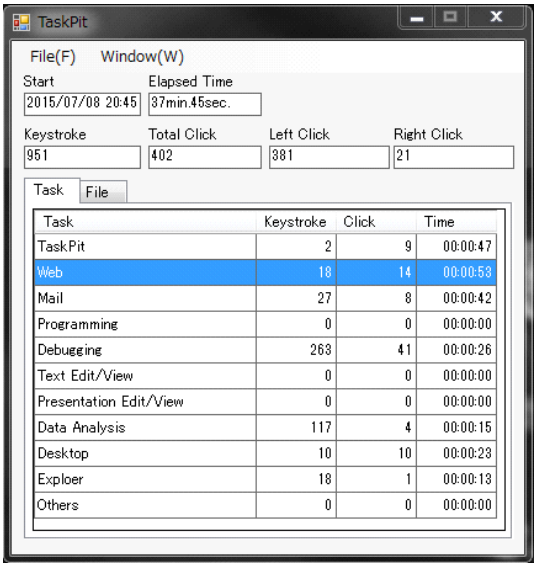


Fig. 4 TaskPit user interface

Table 2 Example of summarized log of TaskPit

	Web Browsing	Mail	Programming	Spreadsheet	File Operation	Others
20130204:100000	123	50	217	120	80	0
20130204:101000	43	80	366	47	56	0
20130204:102000	118	0	387	0	0	0
20130204:103000	0	14	463	0	9	0
20130204:104000	108	0	445	0	39	0
20130204:105000	97	0	320	167	9	0
20130204:110000	126	0	319	148	0	0

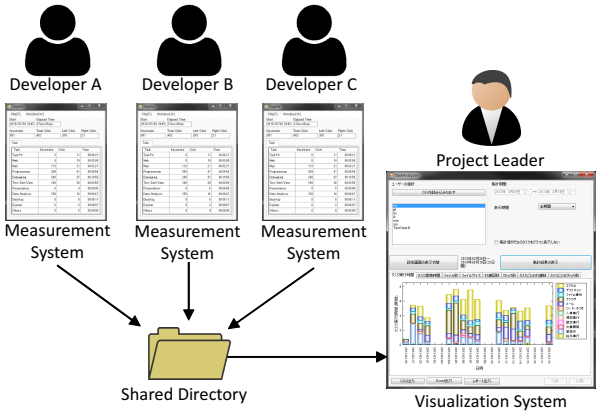


Fig. 5 Measurement for a team

veloper's PC, and the visualization component is run when the developer or team leader wishes to see the measurement results.

In addition, an overview report of daily work can be generated in both CSV format and in Excel .xls format.

3.3 Implementation

TaskPit 1.0.X runs on the .Net Framework on Windows XP, Vista, 7, 8 and 8.1, and uses the SQLite database. The Windows API is called to identify tasks; specifically, GetForegroundWindow is called to obtain the active window handle; GetWindowText is called to obtain the window title. The GetProcessById method from the System.Diagnostics.Process class is called to obtain the process, in order to get the execution file name.

Similarly, to record key strokes and mouse clicks, we register methods to be hooked to record keyboard and mouse events using SetWindowsHookEx. In addition, we also measure outcomes with GetFiles method of System.IO.Directory class. The specified directory is scanned at predefined time intervals to record the number of files and file size.

Measured task log is recorded in CSV file for visualization and further analysis. Free charting library NPlot is used for visualization (see Fig.2). An Excel file is also created for an overview report. Excel file is generated from multiple classes in Microsoft.Office.Interop.Excel namespace.

4. Measurement

4.1 Case Study 1: Team Measurement

4.1.1 Goal of Measurement

The first case study envisions a scenario where a software development department wants to identify problems in the development process and find ways to improve. Analysis of the recorded data is performed by a veteran employee from the Project Management Office (PMO). This person is responsible for development support and consulting, from the point of view of company-wide project management standardization, quality control, and human resource development. The analyst understands the role of each person, but does not know the daily assgined tasks of each person.

The aim of this case study is to observe daily software development activities in industry without any hypothesis, and to find out if any symptoms for possible improvements could be observed by a company's analyst without any prerequisite effort before the analysis.

4.1.2 Measurement Target and Time

The target organization is a software development department consisting of 12 members. Their roles are shown in Table 3.

The department uses Trac as a project management tool which is run on the Web browser. They have been encouraged to use Trac for manipulating variouos documents related to the development.

For data measurement, we used a customized TaskPit 1.0.1, which was modified to hide the GUI to prevent distraction during work. Also, before the measurement we interviewed the members on their application usage to define the binding between task and application. The measurements were carried out over for period of 9 days (6 business days).

Table 4 shows measured time for each member. The table also shows the lines of (summarize) log for each member with an inverval time = 5 min. As shown in the Table, the lines of log to analyze is relatively small. On the other hand, in case of other related work such as Ginger 2 [20], more than 80,000 lines of log data were recorded in a 50-min experiment [21]. The log of Ginger 2 includes various fine-grained real time data such as mouse cursor movement,

Table 3 Team roles

Role	Member	Note
Project Leader	L1, L2, L3	
Member of Project A	A1, A2, A3	A3 is a recruit.
Member of Project B	B1, B2, B3, B4	B1 transferred from different section.
Customer Support	CS1, CS2	Both are part time worker.

Table 4 Measured time and lines of log of each member

	Measured Time (Hours)	Lines of Log
L1	26.3	568
L2	33.1	610
L3	32.3	789
A1	56.6	818
A2	30.5	479
A3	35.8	513
B1	46.1	633
B2	25.7	386
B3	32.0	647
B4	28.3	575
CS1	35.0	674
CS2	26.0	411

mouse clicks, all the changes of texts in kterm window, key strokes, editor commands such as cut, copy and paste, text cursor movement, window movement, and eye-gazing point movement. Therefore, we believe data of TaskPit is much easier to analyze than that of Ginger 2.

4.1.3 Configuration file

Part of the configuration file used for the measurement is shown in Figure 6. The configuration shows that multiple applications can be assigned to a single task.

4.1.4 Measurement Results and Analysis

Figure 7 shows the one day average of time spent on each task, total time for all tasks and total working time. Working time is estimated from the start up and shutdown time of TaskPit. From the results we can give the following observations. These observations were derived by an analyst within a day, which we believe it a realistic time for the analysis.

- As shown in Figure 7 all developers worked on unregistered tasks. With A3, B2, B4 in particular who exceeded 100 minutes in this category. One of the cause of this was, there were unidentified applications installed on the PC, despite prohibiting the installation of unapproved applications in this organiation. Moreover, the organization used an attendance management system, which was not registered in the configuration file. Furthermore, TaskPit can not record testing or debugging that is performed on a virtual OS. (TaskPit 1.0.3 added the ability to record applications not registered in the configuration file.)
- Total working hours and the time spent on the PC-related tasks for project A members (A1, A2, A3) were longer than other project members, due to the proximity to the deadline of the project. On the other hand, project B members (B1, B2, B3, B4) had shorter working times due to their project's satisfactory progress. This shows that TaskPit can be used to grasp the busyness of a project. This is very important to the company because if someone is not working, than that means company's management (task assignment) is bad. Also, this company uses a cooperative company in abroad, and it was very difficult for the company to grasp whether or not the cooperative company was busily working or not. Indeed, after this case study, this company applied TaskPit to an overseas company and observed its work situation.
- A1 had the longest working hours and the time spent on the PC, spending on average 184 minutes programming/debugging, 106 minutes editing and 94 minutes testing. This shows A1 works mainly on coding. Meanwhile, A2 who belongs to the same project spent 0 minutes programming/debugging (or not more than 76 minutes including time for unregistered tasks), and 156 minutes testing. This suggests A2 is responsible for testing. Therefore, TaskPit can be used to grasp the work content of each developer. This is also important because if a developer is not working on the assigned task, then the task assignment is bad and need improvement.
- The relationship between total working hours and the time spent on PC is shown in Figure 8. From this figure, we learn that the 3 leaders spent less time on the PC compared to developers and customer support agents. This is because they need to spend more time in business and management tasks. However, looking at the task breakdown, leader L2 had engaged on average 71 minutes in testing and 103 minutes using excel (which is considered to be involved with test case management). This can be said to be too much engagement in development task for a leader. To improve this situation, a manager need to ask the leader why he/she needed to work on the development task so much, identify its underlying problem, and seek for possible solutions for the problem.
- Two customer support agents (CS1, CS2) are mostly engaged with mailing and browsing. This is because their work requires communicating and interacting with customers via e-mail. The data suggests that they are using Trac for document control and communication in the browser. (This measurement helped us discover the problem that different tasks performed in the browser cannot be reflected in the configuration file.)
- Looking at mail tasks, L2, A3, B1, B2, B3, B4 all spent less than 15 minutes. The department encouraged members to use e-mail rather than telephone for communication (to leave a record of communication).

```
Web Browser = iexplore.exe|firefox.exe|Safari.exe|chrome.exe
Mail = thunderbird.exe|Outlook.exe|iexplore.exe:gmail
File Operation = explorer.exe
Text Editor = sakura.exe|noediter2.exe
DB = SqlWb.exe
Excel = EXCEL.EXE|scalc.exe
Word/Power Point = WINWORD.EXE|POWERPNT.EXE
Document = AcroRd32.exe
Test = mstsc.exe|Beyond32.exe|DF.exe|reverse.exe|reverseserver.exe
Program/Debug = eclipse.exe|devenv.exe|VPC.exe|VMWindows.exe|hh.exe
```

Fig. 6 Configration file

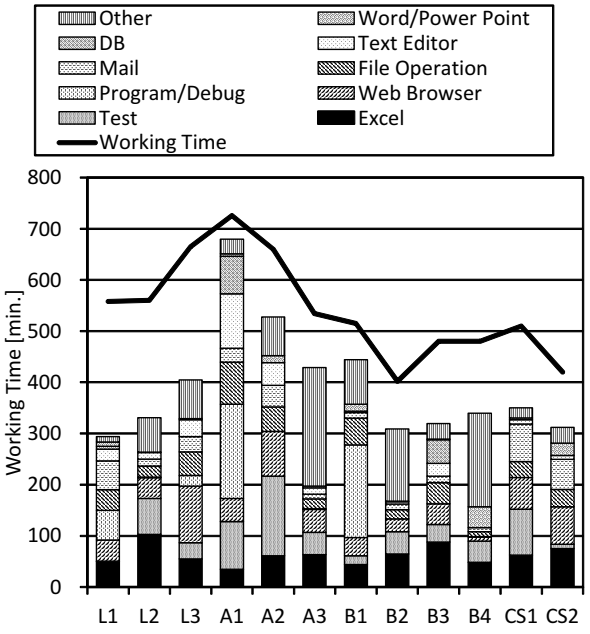


Fig. 7 Average time spent on tasks in a day of each developer

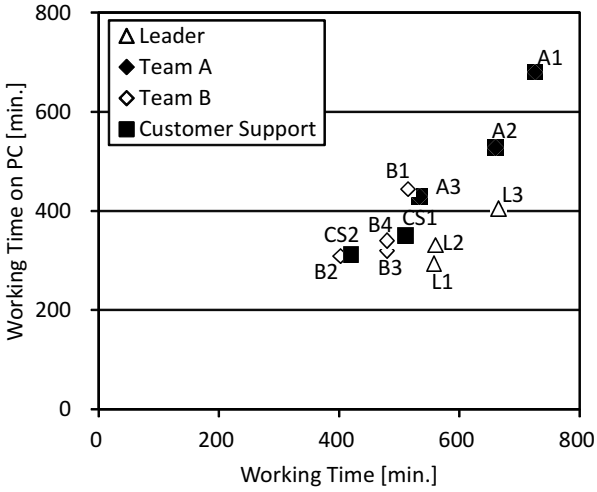


Fig. 8 PC work time and working hour relationship

From the data it can be said that leader L2 spent too little time using e-mail, and after further investigation, we learn that L2 uses the telephone more frequently.

- Figure 9 shows time spent in mailing and programming for each member along with their respective typing speed (or key strokes). Comparing key strokes per minute for the e-mail tasks, CS1 and CS2 show a high value. These two also have high total key strokes, which suggests they send out long e-mails. When responding to e-mails with customers it is preferred to write concisely, covering the main points; additionally, the message should be checked and polished, so it can be seen that the manner in which they write e-mail can be improved.
- Focusing on key strokes for programming/debug task, it can be seen that A1 has the highest number of key strokes and key strokes per minute. Thus, apart from its quality, it can be considered that A1 is the most productive developer.

Our analysis on the measured data revealed that some member possibly violated some policies such as "prohibiting the installation of unapproved applications" and "use e-mail rather than telephone" used in a company. Since the actual violation of policies cannot be judged by TaskPit's data only (e.g. the time spent to use telephone is not recordable by TaskPit), the analyst need to interview a team member if he/she actually violated the policy or not. If an actual violation to a company's policy was identified, then the improvement is to guide members to comply with the policy.

4.2 Case Study 2: Personal Measurement

4.2.1 Measurement Purpose and Methods

The second case study envisions a scenario where an individual developer records and analyzes his own performance to identify problems and lead to improvement. In this case study, we will record the planned time, measured time, and self reported time for each task, daily, and identify the differences in these values. The planned time is the time the developer sets at the beginning of the day. The measured time is the time reported by TaskPit. The self reported time

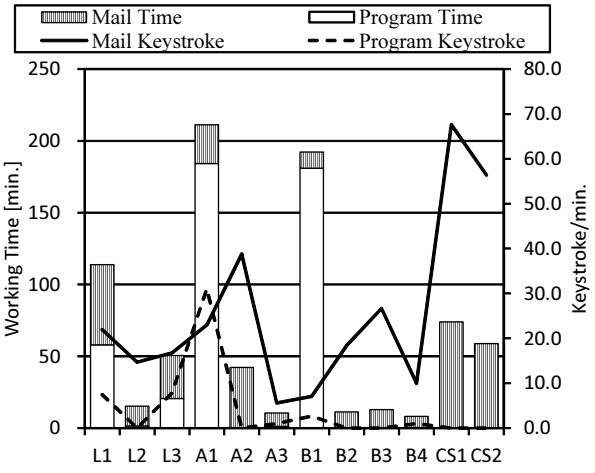


Fig. 9 Mail and programming

is the time the developer reports at the end of the day (before seeing the result from TaskPit).

If the value of planned, measured and self reported time are identical, it means the developer has performed the task according to plan, and has a good grasp on the situation without using TaskPit. However, if a large difference between measured and reported time is present, it means the developer is not able to grasp the time spent on each task. Especially when there is a large difference between measured and self reported time, this is a serious problem as the developer cannot manage his time.

4.2.2 Measurement Target and Period

The target personnel of this measurement belongs to a different organization than the first case study. The target is a developer doing web-based programming. The measurement period was 17 days (13 business days). Prior to the measurement, 3 days were spent adjusting the configuration file and performing preliminary measurements. Also at the end of each day, the developer was asked to look at the planned, measured and self reported times, and write a comment.

4.2.3 Measurement Results and Analysis

Measured value, planned value and self reported value are shown in Figure 10, Figure 11 and Figure 12 respectively. The horizontal axis of each graph represents the days and the vertical axis represents number of hours. Measured values tend to be less than what is planned, except for day 11, which shows time is not spent according to plan. From the developer's comments, this is because time was spent in activities not using the PC (answering the phone) on 4 days, and because there was unplanned work on 3 days. It is usual to have unplanned work/activities during the work day, the data from TaskPit shows that 5 to 10 percent of extra time should be allotted for this during planning. Note that on day 11 the measured time exceeded planned time, this was caused by a mistake in planning (the developer forgot a task that should be done that day).

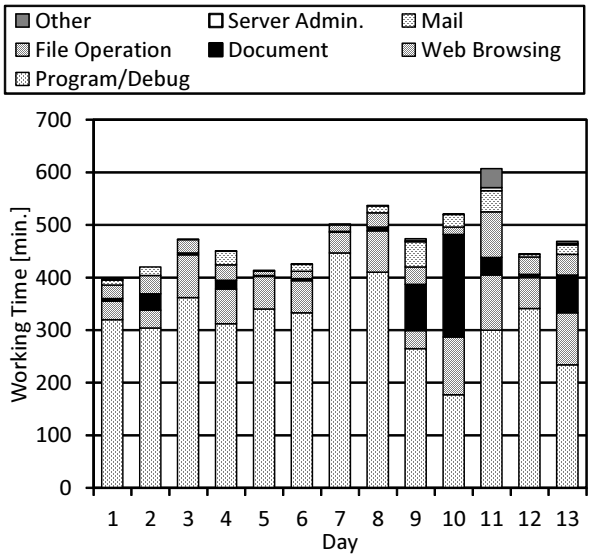


Fig. 10 Measured value

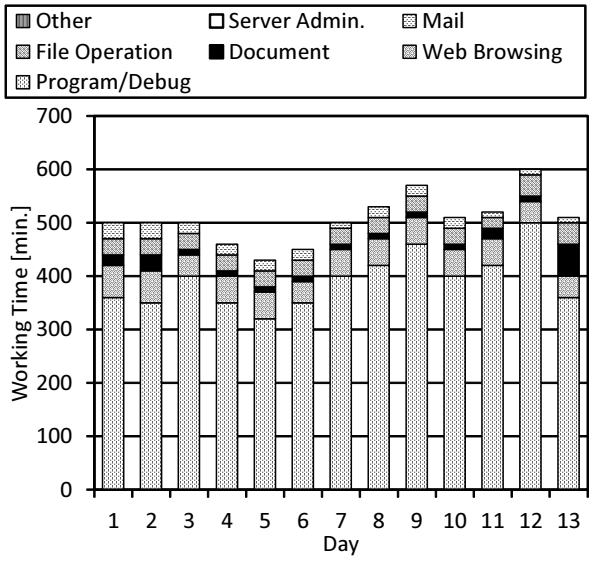


Fig. 11 Planned value

Figure 13 shows measured time minus self reported time. This shows that at the beginning of using TaskPit, the developer overestimated the time spent, and gradually, has been able to better estimate the actual (measured) time. Therefore, from this case study, the developer was able to grasp his actual workload more accurately with the continued use of TaskPit. This is reflected in the developer's comment on day 5: "I was consciously aware of the task I was doing, and my planned times and self-reported times are becoming closer to the measured time." Also on day 7: "If there are no unplanned tasks, the gap between planned, self reported and measured time is becoming smaller. " This shows that by using TaskPit the developer will be more con-

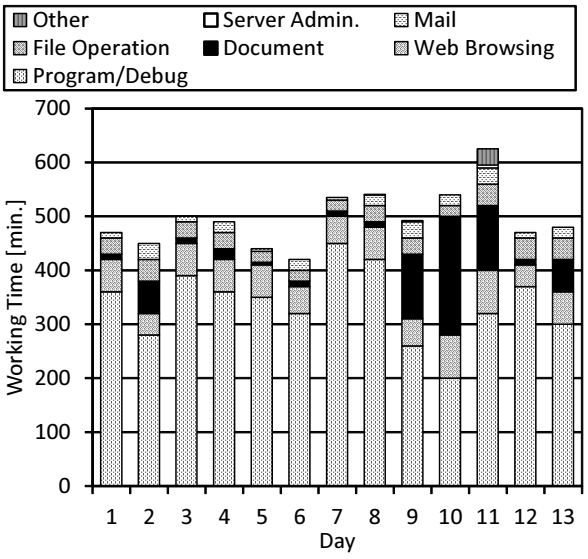


Fig. 12 Self-reported value

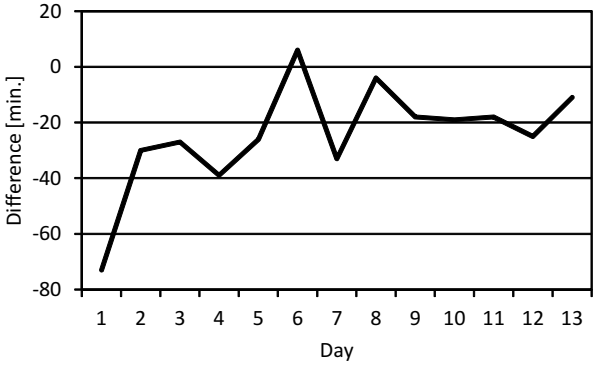


Fig. 13 Difference between measured and self reported time

scious to planned and measured time, and should be able to better manage his tasks.

As more detailed analysis, Figure 14 shows measured time minus self reported time of each task. As shown in the figure, we found that there was no improvement when we focus on individual task. This is partly because, for example the task "web browsing", "document" and "file operation" is sometimes a part of program/debugging and so the developer could not distinguish between such tasks defined by the used applications, although the developer could become able to properly report the "total" working time. We recognize such a gap between application software-based task and real task is a big threats to validity to conduct detailed task-wise analyses, and therefore we are currently working to improve TaskPit to reduce the gap by a machine learning approach [22].

5. Conclusion

In this paper, we illustrated the effect of using automatic

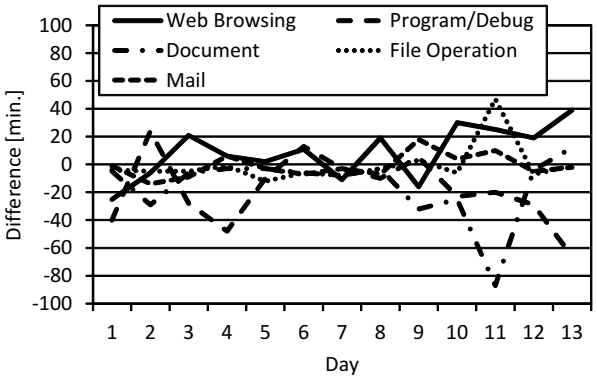


Fig. 14 Difference between measured and self reported time of each task

measurement in software development. This was done through the use of the development task recording system TaskPit in two software development organizations, with the aim to accumulate and publish the findings. In the first organization, 12 developers were recorded for 6 business days. The result, from the PMO's perspective, was the ability to grasp the busyness of each project, and understand the work content of each member. Additionally, the following points of improvements were identified: (a) a leader spent too much time on development tasks, (b) several developers rarely used e-mails, (c) some members spent a short amount of time sending multiple long e-mails to customers, suggesting some room for improvement in e-mail writing.

We believe that the main benefit to employ TaskPit for the purpose of management is to grasp how developers are working. This is especially important for the company who work with overseas cooperative (subsidiary) companies. In case of the company of case study 1, they applied TaskPit to an overseas subsidiary company (after the case study 1), and we that they could observe the busyness of teams in each week, as well as major tasks of each developer (e.g. programming, testing, documenting, etc). Also they observed that developers tend to use web browsers in less busy terms, in which more tasks could be assigned.

In the second organization, an individual developer's planned, measured and self reported times were recorded for 13 business days. As a result, it was discovered that (d) there were unplanned tasks in more than half of the days recorded, (e) the declared time became closer day by day to the actual time measured by TaskPit. These suggest that using TaskPit continuously can be useful in managing the developer's own tasks.

We hope the findings in the above case studies will be useful for organizations considering using TaskPit for measuring and improving their processes. The authors plan to continue to collect more case studies and publish newly obtained findings.

References

[1] "Taskpit." http://taskpit.jpn.org/.

[2] A. Monden, H. Uwano, K. Araki, K. Yamada, and K.i. Matsumoto, "Automatic development task measurement in a software company," JSSST Workshop on Foundation of Software Engineering, pp.257–262, 2013.
[3] T. DeMarco, Controlling software projects: Management, measurement, and estimates, Prentice Hall PTR, 1986.
[4] V.R. Basili and D.M. Weiss, "A methodology for collecting valid software engineering data.," IEEE Trans. Software Eng., vol.SE-10, no.6, pp.728–738, 1984.
[5] R.B. Grady, Practical software metrics for project management and process improvement, Prentice-Hall, Inc., 1992.
[6] E.B. Rini van Solingen, The Goal/Question/Metric method: A practical guide for quality improvement of software development, McGraw-Hill, 1999.
[7] L. Putnam and W. Myers, Five core metrics: the intelligence behind successful software management, Addison-Wesley, 2013.
[8] R.N. Charette., "Why software fails," IEEE Spectrum, vol.42, no.9, pp.42–49, 2005.
[9] B. Curtis, ed., Human factors in software development, IEEE Computer Society, 1985.
[10] S. Flowers., Software failure: management failure: amazing stories and cautionary tales, Wiley, 1996.
[11] W.S. Humphrey, A discipline for software engineering, Addison-Wesley Longman Publishing Co., Inc., 1995.
[12] W.S. Humphrey, Introduction to the personal software process, Addison-Wesley, 1996.
[13] "Process dashboard." http://www.processdash.com/.
[14] "Task coach - your friendly task manager." http://members.chello.nl/f.niessink/.
[15] "Slimtimer - time tracking without the timesheet." http://www.slimtimer.com/.
[16] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, "Collecting, integrating and analyzing software metrics and personal software process data," Euromicro Conference, 2003. Proceedings. 29th, pp.336–342, IEEE, 2003.
[17] "hackystat - a framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data." http://www.hackystat.org/.
[18] M. Ohira, R. Yokomori, M. Sakai, K.i. Matsumoto, K. Inoue, and K. Torii, "Empirical project monitor: A tool for mining multiple project data," International Workshop on Mining Software Repositories (MSR2004), pp.42–46, IET, 2004.
[19] A. Monden, T. Matsumura, M. Barker, K. Torii, and B. Victor R., "Customizing gqm models for software project monitoring," IEICE Transactions on Information and Systems, vol.E95-D, no.9, pp.2169–2182, 2012.
[20] K. Torii, K.i. Matsumoto, K. Nakakoji, Y. Takada, S. Takada, and K. Shima, "Ginger2: An environment for computer-aided empirical software engineering," Software Engineering, IEEE Transactions on, vol.25, no.4, pp.474–492, 1999.
[21] A. Monden, Y. Takada, and K. Torii, "An experiment to observe debugging process by using an eye tracking device," Technical Report of IEICE, vol.96, no.172, pp.1–8, 1996.
[22] R. Ohashi, H. Uwano, A. Monden, K. Araki, K. Yamada, and K. Matsumoto, "Task purpose estimation in software development based on automatic measurement data and machine learning," Computer Software, vol.33, no.2, pp.139–150, 2016.

Pawin Suthipornopas is a student at Kasetsart University, Thailand in Software and Knowledge Engineering faculty, in which he will graduate in 2016. Had two months period of internship in 2015 at Nara Institute of Science and Technology in Software Engineering laboratory. His research interests include software measurement and software process.

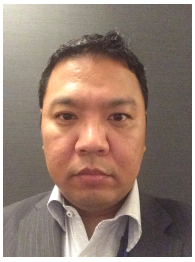
Pattara Leelaprute is a lecturer at the Faculty of Engineering, Kasetsart University, Thailand. He received his B.E. (2001) in Information and Computer Science, M.E. (2003) in Computer Science, and Ph.D. (2006) in Information and Systems Engineering from Osaka University, Japan. His research interests include Feature Interactions, Telecommunication services and Home Network Systems. He is a member of IEICE.

Akito Monden is a professor in the Graduate School of Natural Science and Technology at Okayama University, Japan. He received the BE degree (1994) in electrical engineering from Nagoya University, and the ME (1996) and DE (1998) degrees in information science from Nara Institute of Science and Technology (NAIST). His research interests include software measurement and analytics, and software security and protection. He is a member of the IEEE, ACM, IEICE, IPSJ and JSSST.

Hidetake Uwano is an associate professor at National Institute of Technology, Nara College, Japan. He received BE degree (2004) in Software and Information Sciences from Iwate Prefectural University, and ME (2006) and DE (2009) degrees in information science from Nara Institute of Science and Technology (NAIST). His research interests include human computer interaction, human factor, and software measurement. He is a member of IEEE, ACM, IEICE, JSSST, and HIS.

Yasutaka Kamei is an associate professor at Kyushu University in Japan. He has been a research fellow of the JSPS (PD) from July 2009 to March 2010. From April 2010 to March 2011, he was a postdoctoral fellow at Queen's University in Canada. He received his B.E. degree in Informatics from Kansai University, and the M.E. degree and Ph.D. degree in Information Science from Nara Institute of Science and Technology. His research interests include empirical software engineering, open source software engineering and Mining Software Repositories (MSR). More information about him is available online at http://posl.ait.kyushu-u.ac.jp/ kamei/

Naoyasu Ubayashi is a professor at Kyushu University since 2010. He is leading the POSL (Principles of Software Languages) research group at Kyushu University. Before joining Kyushu University, he worked for Toshiba Corporation and Kyushu Institute of Technology. He received his Ph.D. from the University of Tokyo. He is a member of ACM SIGPLAN, IEEE Computer Society, and Information Processing Society of Japan (IPSJ). He received "IPSJ SIG Research Award 2003" from IPSJ.



Kenji Araki joined Lnet Co., Ltd. in 1993 and moved to ACCESS Co., LTD. in 1996. He was a part of many large projects and was selected as a team and a project leader. During his career, he worked on different aspects of both the CASE tool and the reverse engineering tool such as research, modeling, and utilization of them. He was appointed as a head of an offshore development group in 2011. Not only he has been an important coordinator between the company and its Shanghai branch, but also he engaged in the BSE management extensively. As a result of a consolidation with NCS Co., Ltd., the company has been renamed as NCS & A Co., Ltd in 2014.

Kingo Yamada majored in industrial engineering and graduated Kansai University in 1988. In the same year, he joined Nissho Electronics and worked on system analysis, development and consulting. In 1994, he joined ACCESS Co.,Ltd. and engaged in research, planning, and development of the CASE tool and the reverse engineering tool, REVERSE PLANET. In 2002, he obtained a patent on the reverse engineering tool. He became a director and an executive director of the company in 2006 and in 2012 respectively. He was appointed to an executive vice-president of a subsidiary branch, Finebus Co., Ltd. in 2013 and has been a CEO since 2014.

Ken-ichi Matsumoto is a professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. He received the B.E., M.E., and Ph.D. in Information and Computer sciences from Osaka University, Japan, in 1985, 1987, and 1990, respectively. His research interests include software measurement and software process. He is an IEICE and IPSJ Fellow, an IEEE senior member, and a member of ACM and JSSST.