Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 2. pp. 255–262
doi: 10.14794/ICAI.9.2014.2.255

A methodology for measuring software development productivity using Eclipse IDE

Gábor Antal, Ádám Zoltán Végh, Vilmos Bilicki

University of Szeged, Department of Software Engineering
antalg@inf.u-szeged.hu, azvegh@inf.u-szeged.hu, bilickiv@inf.u-szeged.hu

Abstract

During software development processes many methodologies and technologies are used which can be examined and compared by many points of view. One of the important aspects is the development productivity which affects development time and costs significantly. It is the composition of many factors but actually not all relevant and affecting factors and their relationships are known. Measuring the development productivity can be very useful if we would like to see that:

- How much of the total development time takes the real development?
- How long is the real development time of specific software components and layers?
- How much time does a bug fix or the implementation of a new feature take in specific components or layers during software evolution?

Beside the development time it is also worth to examine the quality of the software using various software metrics. Therefore a special tool is needed which can perform real time productivity measurements during the development but at present there are only a few tools for this task with limited measurement capabilities. The goal of this paper is to introduce a methodology for measuring productivity (even for specific software units) with defining a list of relevant factors and events to be observe (e.g. file and user interface events). Besides, this paper presents a measurement tool for monitoring development productivity in the Eclipse IDE (Integrated Development Environment). We have successfully measured a former project using this Eclipse-based tool and evaluated the measurement results to examine the development time of specific application layers.

Keywords: software development, productivity, development time, metrics

1. Introduction

Productivity is one of the most important factors in software development projects, which affects the costs and time requirements of the development process significantly. Therefore, it is a very important area of software engineering research to analyze and improve software development productivity. However, defining productivity is still a very difficult problem, because it is the composition of many factors and not all relevant and affecting factors and their relationships are known yet.

Former articles and studies [1], [2], [3], [4] analyzed and collected many affecting factors of software development productivity. These factors can be divided into the following two main categories.

- Technical factors: these factors are related to the technical attributes and characteristics of the development process. Some examples of technical factors: complexity of the software product, reusability of software components, programming language, frameworks and libraries, development practices, design patterns, use of developer tools.
- Human (soft) factors: these factors are related to the human attributes in the project. They can be analyzed in terms of an individual person or the whole team, which works on the project. Some examples of human factors: respect, communication, fairness, team cohesion, support for innovation, capabilities and experiences.

Measuring productivity is also very important for development process analysis and improvement. The main goals of productivity measurement are measuring the efficiency of the development, the skills of developers, and the real development time of specific application layers, components, or the whole project. The results of the measurement can be used to find the weak points of the development process and improvement actions can be proposed to correct these weaknesses. However, productivity analysis can be used in many areas of research, for example:

- Comparing different development models, methodologies, practices in terms of productivity.
- Comparing different programming languages, technologies, frameworks in terms of productivity.
- Detecting specific development strategies, patterns of developers.

The productivity of a software developer can be measured by observing and collecting specific types of events related to the usage of the development environment, e.g., file events, user interface events. The goal of this paper is to introduce a methodology for measuring productivity with defining some relevant factors and events to be observed in the development environment, and a measurement toolkit for monitoring productivity in the Eclipse IDE [5]. This paper also presents an analysis method for calculating the real development time of application layers with a case study using the measurement results collected during a former software development project.

2. Related work

Monitoring the usage of development environments is not a new idea, some articles and projects already exist in this research area. Most of these usage monitoring applications were developed for the Eclipse IDE, because the development of plug-ins is well supported in it and it can be extended with custom event listeners for observing developer actions.

The Eclipse IDE contained a usage monitoring tool named Usage Data Collector (UDC) [6] until version 3.7 (Indigo). It could collect data about loaded bundles, keyboard shortcuts, menu and toolbar actions, perspective changes, and usage of views and editors. The captured data were periodically uploaded to servers hosted by the Eclipse Foundation. But the databases with the collected usage data are not opened for public access.

In the [7] article an Eclipse plug-in is introduced, which can monitor keyboard, mouse, scrollbar, and file change events into XML files, which can be converted to Excel files. The Mylar Monitor [8], [9] can capture many user interface events (e.g., view, perspective, editor events) and commands (invoked by key bindings, menus and toolbars). The CodingSpectator [10] is an Eclipse plug-in for monitoring the usage of refactoring commands. Its extended version is the CodingTracker [11] plug-in with the capability of capturing several other events, e.g., file events, interactions with Version Control Systems, test and application runs.

These toolkits can monitor many types of events in the Eclipse IDE, but they do not take some other important factors of productivity into account, such as the current task of the developer, the interruptions and idle time intervals during the development, and the quality metrics of the software product. Our goal is to develop a productivity measurement toolkit which can monitor most of the user interface events that can be captured by the existing tools, with the compensation of the mentioned deficiencies.

3. Methodology for productivity measurement

As it was previously mentioned, the productivity of individual developers can be measured by monitoring the usage of the development environment. Mostly it means capturing the events of the user interface elements and resources (e.g., files). But there can be other relevant factors, which should be monitored to measure the efficiency of the development, for example:

- The actual task of the developer is very important for analyzing the collected events and understanding the causes of event frequencies and patterns. For example, if the developer has a bug fixing task, the number of file save events can be very low, because looking for the cause of the bug can be a long process.

- Developers rarely use the development environment in one long working session. There can be interruptions during the development, so the developer has to work on another task. In other cases, there can be idle time intervals, when the developer does not work with the development tools, e.g., lunch break, reading documentations, learning from tutorials. These interruptions and idle time intervals are important to calculate real development time requirements of the project.
- Measuring the changes of the product quality can be also a useful perspective of productivity monitoring. Therefore, it is necessary to collect code quality metrics (e.g., cyclomatic complexity, coupling, lack of cohesion of methods), when any part of the source code changes.
- It can be also useful to collect data about the structure of the source code to monitor the structural changes of the software product.

For monitoring these additional factors, a new productivity measurement methodology was defined. The measurement process is performed by a plug-in, which is embedded in the development environment, and the captured data are uploaded to a data collector server for further analysis. The measurement plug-in is responsible for the following tasks:

- Monitoring all events related to files (creating, opening, saving, switching, closing, deleting) and projects (opening, closing, creating, deleting).
- Monitoring events related to the user interface (windows, dialogs, editors, views, etc.)
- Monitoring keystroke and shortcut events from the keyboard.
- Monitoring code movement events (cut, copy, paste).
- Monitoring code running events (start, debug, profile, stop).
- Monitoring the actual task of the developer, which can be selected from a predefined list downloaded from the server. The state (new, started, suspended, and finished) and progress of the actual task is also monitored.
- Monitoring interruptions and idle time intervals. The plug-in observes the keyboard and mouse actions performed by the developer, and if the last action was performed "too long ago", it is considered as the starting point of an idle time interval. The developer can confirm or reject this decision, when a new action is performed.
- If a save event is captured related to a source code file, the structure of the source code and the code quality metrics of the file are also monitored.

4. Eclipse-based productivity measurement

The previously presented productivity measurement methodology was implemented in a plug-in for the Eclipse IDE. This toolkit can monitor all the mentioned types of file and project events using custom implementations of IResourceChangeListener and IResourceDeltaVisitor interfaces. User interface events are monitored related to windows (open, close, switch), perspectives (open, close, save), views (open, close, switch), dialogs (open, close), editors (open, close), with custom implementations of built-in user interface listeners, e.g., IWindowListener, IPartListener2, IPerspectiveListener3. Keyboard events are collected with an SWT Listener, which captures keystrokes and shortcuts. Code movement events are detected with an IExecutionListener implementation; code running events are captured using an ILaunchListener implementation.

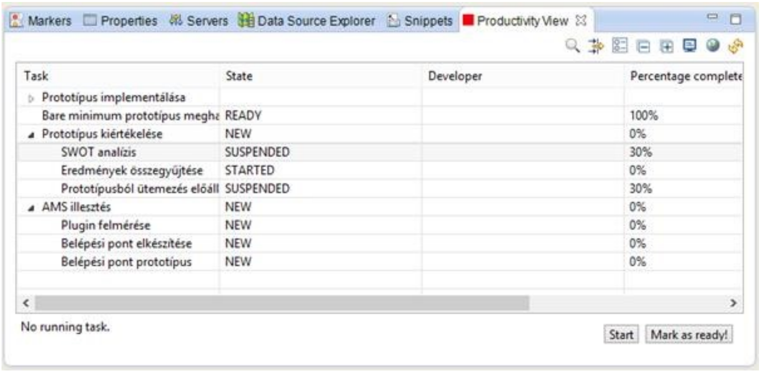
The list of selectable tasks can be defined on the server and the plug-in instances can download it. When the developer starts the Eclipse, the previously started task can be continued. Task suspending, changing and finishing can be performed on a special view (Figure 1), with the possibility to set the progress of the task and write a comment related to it. When the developer closes the Eclipse, the actual task is automatically stopped. The plug-in can be also used in passive mode, which means there are no tasks to manage and only the previously mentioned types of events are collected.

Interruptions and idle time intervals are detected using sessions. A session is started when a task is started or a keyboard or mouse action is performed. A session is finished when a task is suspended or the elapsed time since the last keyboard or mouse action is higher than a predefined session timeout value.

When a save event is detected related to a Java source file, the plug-in collects about 21 code quality metrics about the file and its package, with maximum, average and standard deviation values if it is possible, e.g., Total Lines Of Code (TLOC), coupling, Depth of Inheritance Tree (DIT), Nested Block Depth (NBD), Specialization Index in methods, McCabe Cyclomatic Complexity, Weighted Methods per Class (WMC). Metrics are collected using the Metrics plug-in for Eclipse [12]. The structure of the source file (names and relations of classes, fields, methods, annotations, etc.) is also captured using the high level Java model of Eclipse JDT (Java Development Tools).

5. Calculating real development time for application layers

Using the previously explained methodology, an algorithm was developed for calculating the real development time of a file in the project for a specific developer. This method is based on the analysis of the file events, which can have one of the following types: open, switch, save, close, delete, left. Left file events are created when a session is finished, so it is the starting point of an idle time interval. The



Task	State	Developer	Percentage complete
▷ Prototípus implementálása			
Bare minimum prototípus meghá	READY		100%
◢ Prototípus kiértékelése	NEW		0%
SWOT analízis	SUSPENDED		30%
Eredmények összegyűjtése	STARTED		0%
Prototípusból ütemezés előáll	SUSPENDED		30%
◢ AMS illesztés	NEW		0%
Plugin felmérése	NEW		0%
Belépési pont elkészítése	NEW		0%
Belépési pont prototípus	NEW		0%

No running task.

Figure 1: Task selector view of the productivity measurement toolkit

algorithm collects all types of file events for the given file and the open and switch events for other files, ordered by the date of the events. After that, it iterates over these events and determines the relevant intervals for the development of the file, according to the type of events related to the given file. If the type of the actual event is open, switch or save, and it is related to the given file, its date is the starting point of a relevant interval, and the endpoint is the date of the next event in the list. The sum of the lengths of these relevant intervals gives the real development time of the given file.

These real development time results can be aggregated for specific application layers. If the naming conventions are well defined and strictly used in the project, then the application layer for the file can be determined using simple rules according to file names, extensions, class names and package names, e.g., if class name contains “DAO”, then it is a class of the DAO (Data Access Object) layer. In other cases, the application layer can be determined by analyzing the structure of the file, e.g., @Entity annotation on the class means that the source file belongs to the entity layer.

We have measured one of our former projects with the previously presented toolkit, and calculated the real development time for application layers by a lead software developer weekly. The results can be seen on a stacked area diagram on Figure 2. It can be seen that the developer had to work a lot in the implementation of the software in the preparation phase, until week 20. From week 21, he had less implementation tasks and he could concentrate on his management tasks, too.

6. Summary and future plans

In this paper we discussed some existing productivity measurement toolkits for the Eclipse IDE, and introduced a new methodology with an Eclipse-based toolkit for measuring the productivity of software developers. An algorithm was also

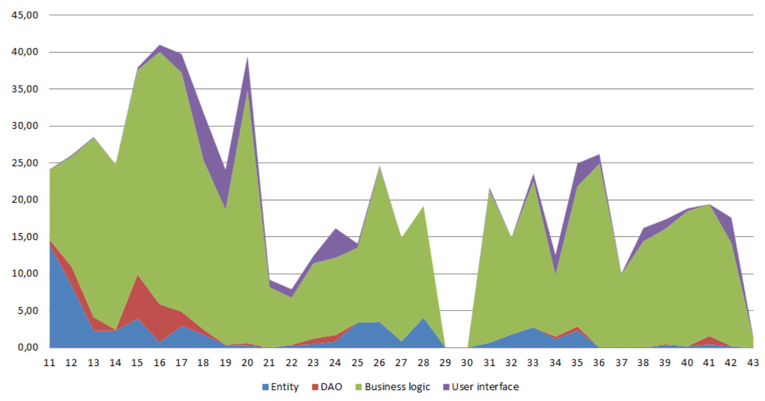


Figure 2: Real development time for application layers by a lead software developer weekly

developed for calculating the real development time by a specific developer for files and application layers, using the collected measurement results. In the future we would like to perform additional productivity measurements on our projects, and analyze the results in terms of the following research areas:

- Comparing different development models (e.g., waterfall, prototype-based, agile).
- Comparing different Java Enterprise Edition technologies.
- Detecting and comparing different developer strategies, patterns.

Acknowledgements. The publication is supported by the European Union and co-funded by the European Social Fund.

Project title: “Telemedicine-focused research activities on the field of Mathematics, Informatics and Medical sciences”

Project number: TÁMOP-4.2.2.A-11/1/KONV-2012-0073

References

[1] CHINUBHAI, A., Efficiency in Software Development Projects, International Journal of Software Engineering and its Applications, Vol. 5 No. 4 (October, 2011), 171-179

[2] WAGNER, S., RUHE, M., A Systematic Review of Productivity Factors in Software Development, Institut für Informatik, Technische Universität München, Technical Report TUM-I0832

[3] MAXWELL, K. D., FORSELIUS, P., Benchmarking Software Development Productivity, IEEE Software, Vol. 17, No. 1 (January/February, 2000), 80-88

[4] JIANG, Z., COMSTOCK, C., The Factors Significant to Software Development Productivity, International Journal of Computer, Information Science and Engineering, Vol. 1, No. 1 (2007), 68-72

[5] `http://eclipse.org/`, retrieved on: 2014. 04. 30.

[6] `http://www.eclipse.org/org/usagedata/`, retrieved on: 2014. 04. 30.

[7] MCKEOGH, J., EXTON, C., Eclipse plug-in to monitor the programmer behavior, Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange (2004), 93-97

[8] KERSTEN, M., MURPHY, G. C., Mylar: a degree-of-interest model for IDEs, Proceedings of the 4th International Conference on Aspect-oriented Software Development (2005), 159-168

[9] MURPHY, G. C., KERSTEN, M., FINDLATER, L., How Are Java Software Developers Using the Eclipse IDE?, IEEE Software, Vol. 23, No. 4 (July/August, 2006), 76-83

[10] `http://codingspectator.cs.illinois.edu/`, retrieved on: 2014. 04. 30.

[11] `http://codingtracker.web.engr.illinois.edu/`, retrieved on: 2014. 04. 30.

[12] `http://metrics2.sourceforge.net/`, retrieved on: 2014. 04. 30.