Mining a Developer's Workflow from IDE Usage

Constantina Ioannou

Kongens Lyngby 2017
M.Sc.-2017-

Technical University of Denmark
Applied Mathematics and Computer Science
Matematiktorvet, building 303B, DK-2800 Kongens Lyngby, Denmark
Phone +45 4525 3031, Fax +45 4588 1399
compute@compute.dtu.dk
www.compute.dtu.dk M.Sc.-2017-

Abstract

Software developers interact with integrated development environments (IDEs) by issuing commands that execute various programming tools, from source code formatters to build tools. Several studies has shown that despite the usefulness of provided tools and capabilities to perform tasks such as: navigating among classes and methods, continuous compilation, code refactoring and integrated debugging, IDEs usually tend to overload developers resulting a chaotic state. In other words, commonly developers spend time shifting between several artifacts of the working environment in order to reach their goal, and this repeated process increases the complexity of identifying the relevant information to solve their assignment.

For this reason, an assortment of usage metrics plugins to gather developers' activity have been developed in an attempt to understand a developers' workflow. Understanding the workflow, as well as the skillset and experience of a developer is the first step into improving their productivity and reducing the unneccessary complexity created within IDEs.

In this work, we analyze and provide an overview of the project itself and discuss concepts involved: by uncovering available existing plugins for recording a developer's interaction within an IDE. Further, we describe in detail the software tool which was selected for expansion and features which were developed to allow capture, mine and analyses for the process of software development. Finally, we present the results of mining deelopers' interactions according to their IDE usage.

We aim to contribute information for modelling an automatic activity detection tool capable of tracking the workflow of developers' activity which potentially

will be used as input in later studies for a reconfiguration program supporting the developer.

Preface

This thesis was carried out at the department of DTU Compute, at the Technical University of Denmark, in fulfilment of the requirements for acquiring an M.Sc. in Computer Science and Engineering.
The goal of this project was to collect data on how developers interact with an IDE while developing software and to mine developers' workflows from IDE usage using process mining techniques.

This report describes the project itself and discusses concepts involved. Further, analyses in detail the software which was selected for expantion and features that were developed during this project. Finally, presents the results of mining developers' interactions according to their IDE usage. *TODO: add signature*

Acknowledgements

My deepest thanks go to my advisor, Barbara Weber. Throughout these months, Barbara gave me the freedom for exploring an unkown field, challenged my findings, and offered advices whenever needed. Barbara taught me the importance of software methodologies and helped me to evolve my experience with agile environments: by narrowing down ideas and being selective over the most feasible ones. All these advices have been invaluable. I also thank my co-supervisor Andrea Burattin for his time and feedback. Andrea shared his knowledge regarding process mining and his questions and suggestions have enabled me to view this work from a wider perspective. Furthermore I would like to express my gratitude to my family and friends who showed patience and were supportive during moments of frustrations.

A great thank you to everyone who believed in me!

Contents

Chapter 1

Introduction

Programming is a really complex procedure and requires a great amount of human memory. Although modern program development environments have begun to recognize the complexity and the memory requirements of programmers, none of these tools are able to take into account the programmers' cognition. ADD Reference: Chris Parnin and Spencer Rugaber. Programmer information needs after memory failure. In ICPC '12: Proceedings of the 20 Interaction Conference on Program Comprehension, pages 123–132, 2012. Therefore the process of software development doesn't feel any better than it did a generation ago. Previous research focuses on understanding how developers operate to complete their jobs: how they keep their tasks structured, the strategies they apply, the tools they use

Several studies has shown that, despite the usefulness of provided tools and capabilities to perform tasks such as navigating among classes and methods, continuous compilation, code refactoring and integrated debugging, IDEs usually tend to overload de- velopers resulting a chaotic state. In other words, commonly developers spend time shifting between several artifacts of the working environment in order to reach their goal, and this repeated process increases the complexity of identifying the relevant information to solve their assignment.

Yet researchers pointed out certain aspects of the problem still the workflow and the way a programmer might think at a particular moments is still a mystery.

in order to improve the environments available we have although to understand these aspects.

ides aim to help the user however often they create a general chaos with all the navigation the studies of .. and the Cognition studies and models strategies top down bottom up,

previous research of understanding the workflow of a developer is done by ...

previous research of attempting to extracct user activities from IDE ...

Abstract—Developers use the Integrated Development Environ- ment (IDE) to develop a system at hand, by reading, understand- ing, and writing its source code. They do so by exploiting the tools

and facilities provided by the IDE. This also allows them to build a mental model of the system to perform informed changes. It is however not clear how and when developers use which facility and tool, and to what extent the current services oered by the IDE appropriately support the navigation. We present an approach to visualize the activities of developers within the IDE, implemented in a tool: DFlow. DFlow records all IDE interactions that occur during a development session and

By better understanding actual work practices, looking for patterns in them, and in performing empirical studies based on this knowledge, we can better address how to create the next generation of supporting tools for software developers.

Previous research of understanding how software developers do their jobs has focused mainly on studies of programmer cognition and work practices. Cognition studies try to ex- plore the cognitive models, strategies, and patterns used by developers performing program comprehension. Mostly based on laboratory experiments, these studies have shown that programmers apply multiple comprehension strategies, such as top-down [Bro77], bottom- up [Pen87] , and integrated [vMV95], and also that patterns in the code, such as beacons [Bro83, GC91], plays an important role in comprehension. These findings have guided the design of some research development tools (e.g., Shrimp [SBM01]) in supporting program comprehension. The study of developer work practices, on the other hand, focuses on a big picture of industrial developers working in the context of realistic development tasks. Sur- veys, diaries, interviews, and other methods have led to several surprising insights about work practice; for example, Singer et al. found that developers spend significant effort on 1 searching for code elsewhere in the system that relates in some way to the code they are working on [SLVN97]. Seaman found that the information sources that developers rely on most are source code, colleagues, and various development artifact

repositories [Sea02]. These findings suggest that supporting improved information searching, interruption man- agement, and repository mining can be helpful to programmers.

1.1 The problem

Although, the mentioned findings are a bit irrelevant because we are awesome

A fine-grained understanding of developers a fine-grained understanding of industrial programmers working in an industrial context remains limited. Cognitive studies often involve student programmers working on small problems. It is hard to generalize from such small-scale studies to industrial software development practice, where developers typically have a much larger code base and more complicated tasks to consider, and where they will be ultimately be held accountable for their decisions. Detecting high-level comprehension strategies has been the main focus of cognitive studies. But detailed work patterns — such as what artifacts are usually used for solving certain typical sub-tasks, and what relationships exist between those artifacts — remain mostly unexplored. Moreover, the relation between programmers' work and contextual factors is poorly understood. As shown in Figure 1.1, driven by maintenance tasks, programmers interact with artifacts of a software system using tools in the work domain.

Tasks may largely determine which parts of the software system are relevant, but other factors in software development, such as the expertise level of programmer, the design of the software system, the features of the tool, and interruptions during tasks, may all affect programmer's detailed work pattern. To understand how industrial software developers perform their job — and how we can help them do better — we should not ignore these contextual factors.

1.2 The approach

1.3 Integrated Development Environment

1.4 Structure of thesis

Chapter 2

IDE

During programming tasks developers often interact with a variety of facilities and tools: browsers, IDEs, mail clients, virtual machines, etc. All these interactions are of high significance when it comes to understand the workflow of a developer. An IDE is very personal and therefore analysing developers' behavior is feasible.

This chapter begins with an introduction to the history of IDEs to support their importance and a description of their future is given. Further, the main focus is the analysis and discussion of possible interaction with today's IDEs.

2.1 Quick IDE History

An IDE is a software application which aims to improve developers' productivity by facilitating application development. It consolidates the basic tools developers need to write and test software. Typically, an IDE consists of a source code editor, a compiler, a debugger and build automation tools. According to ADD Reference: reference: the idea behind IDE was realeased as Turbopascal which integrated an editor and a compile. However many believe Microsoft's Visual Basic (VB), launched in 1991 was the first real IDE, and by the time it was relatively easy to learn and use.

Over the years several IDEs have been implemented and released. IDEs offered remarkable advantages to developers mentioned in ADD Reference: reference: and ADD Reference: reference: for example:

- Reducing the setup up time required to configure multiple development tools,
- Increasing development speed and efficiency by providing instant feedback for syntax errors or code insights,
- Increasing program management by organising resources, providing a virtual representation of the project, and automatically adding appropriate imports.
- Increasing the quality of programs by offering the ability for debugging, testing and source control.

Overal these advantages of using an IDE have endorsed the productivity of a developer. Today, according to the Top IDE index ADD Reference: reference:github , a ranking created by analysing how often IDEs are searched on Google, the 3 most used tools employed to develop source code are Eclipse, Visual Studio, Android Studio. Although s are stricking through the top 3, it is noticable that a significant amount of developers prefer to use highly configurable text editors such as Vim and Sublime Text. The different types of code, specific languages, cloud based, mobile application, apple or microsoft development, led IDEs to expand in a variety of directions. Consequently not all IDEs offer the same capabilities and the same collection of tools. Therefore one must be selective when choosing an IDE, in accordance to its development task and requirements.

Even though that IDEs are highly helpful to a developer, they lack effective support to browse between complex relationships between source code elements. This implies that developers spend 30% their time navigating through a chaotic environment and therefore their productivity is being reduced as supported by ADD Reference: Taming the IDE with Fine-grained Interaction Data. ADD Reference: Autumn leaves and ADD Reference: An explatory study

For this thesis, in an attempt to understand the developers' workflow the extraction of user interactions with an IDE is required. The decision to analyse and develop for Eclipse IDE was taken for a vast amount of reasons listed in 3.

2.2 Interactions within IDE

IDEs are highly used by developers nowadays to develop and maintain a software. Developers use IDE to read, navigate, understand and write code. In more detail the developer interacts with IDE through low level events such as: mouse clicks and keyboard shortcuts but also high level events facilitated by the tools provided in the toolbar such as: refactoring, debugging, navigating, testing, inspecting as mentioned in ADD Reference: An explatory study and ADD Reference: How Much Integrated Development Environments Improve Productivity. A high level event is a series of low level events. In other words the toolbar allows the developer to code, compile and execute source code without having to follow manually these steps. An example is the process of renaming a class within a large project. IDE provides contextual menues enabling fast and efficient changes where otherwise a series of low level events: editing several classes, compiling and error seeking, executing and testing the correctness after the change should be done manually.

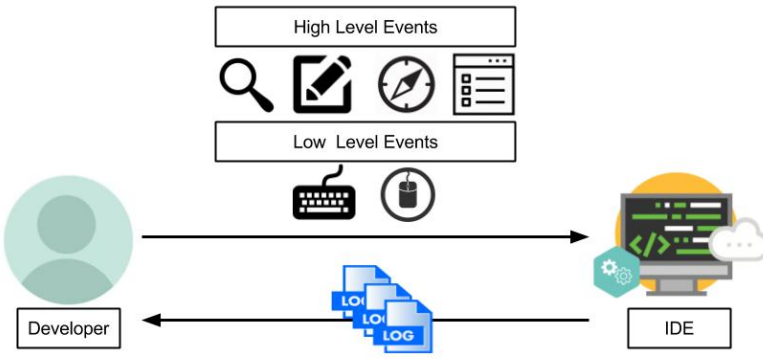

Figure 2.1: Flow of usage data between developer and IDE.

According to ADD Reference: Visualizing the Workflow of Developers and ADD Reference: Capturing and Exploiting IDE Interactions by gathering all the IDE usage data provides an additional view of what the workflow of a developer is to solve a certain task. A task (implementing a class, correcting an error) is consisting of a series of high and low level events These interaction events are called usage data ADD Reference: A Practical Guide to Analyzing IDE

Usage Data and they are an essential element for this thesis. All the usage data generated by the activity of a developer within an IDE are recorded. The developer initiates the different interactions and IDE responds back with logging particular events. The model in 2.1 shows the exchange of usage data between developer and IDE.

2.3 The usage data model

Chapter 3

The Eclipse IDE

In the following chapter a general overview of Eclipse framework and its most prominent components is provided. A description and analysis of available plug-ins from which essential information could be captured is presented. Last but not least a great amount of focus is given on a specific plug-in called Rabbit, which is used and enhanced throughout the thesis.

3.1 Eclipse

Eclipse workbench is a powerful UI framework for IDEs fairly used by several developers: to primarily develop Java applications or even to develop in other programming languages, and to also develop documents and packages for other softwares. It provides several services for highly integrated and extensible user interfaces. Eclipse fulfills the requirement of an entire product developer team to use one platform where all their tools integrate and work together. This is one of the main reasons Eclipse is still the top IDE available ADD Reference: top index ide

The standard SDK Eclipse distribution contains a base workspace with certain functionalities, Java Developemnt tooling plug-ins and layout. Thereafter

additional plug-ins can be included or created to allow the extension of the workspace. As a result Eclipse can become a multifunction framework to be used for: Embedded programming, C++ programming, javaBeans, Java application, websites, or even develop additional Eclipse plug-ins, etc. The plug-ins can be removed or replaced according to the needs and preferences of developers.

Even though Eclipse framework started as a replacement for Visual Age for Java from IBM ADD Reference: wikipedia, it became an open platform for integrating tools, editors, views and plug-ins. Its source code is freely available and anyone can contribute by building their own new plug-ins or by engaging in discussions regarding integrated tools.

The most significant reasons of selecting the Eclipse IDE as a tool for gathering information for processing developers' workflow are:

- It is the top ranked used IDE by developers ADD Reference: BOOK and ADD Reference: the top 10 index
- It is a home for tools, where you can build and intergrate your own ADD Reference: the book
- It is an open source framework, he features and plug-ins are available online, ADD Reference: the book

3.2 Eclipse User Interface

3.2.1 Eclipse UI as Application Developer

ADD Reference: The eclipse foundation. eclipse documentation:release oxygen. online url// eclipse.org Eclipse framework consists of workbench windows, which have integrated basic elements such as active perspectives, editors, views and menus.

When starting up Eclipse, if its a newly initiated greets the developer with a welcoming screen, otherwise the previously workspace is loaded. Figure **??** displays a standard Eclipse setup for a Java IDE.

A *perspective* has its own a set of elements views and editors and menus along with a adaptable personalized layout for specific tasks such as debugging a program. A *view* is a window that enables the developer to examine something, eclipse offers various of views f.x. Java packages and their files. An *editor* is a

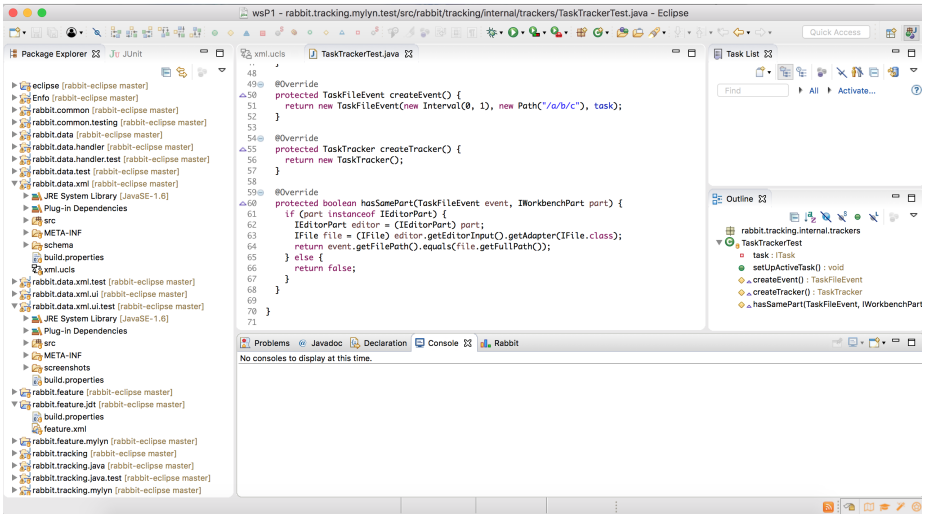

Figure 3.1: The Java Development perspective in Eclipse.

smart editor which recognizes the programming language markups the syntax and allows you to modify and save files. Editors share characteristics with views, but unlike views, editors don't have tool-bars. Eclipse is filled up with *menus*, the main menu is at the top of the screen while views and explorers usually provide their own context menus.

Eclipse IDE is highly adjustable and enables the developer to construct and save his own perspectives according to his preferences. Having these advantages in mind one can optimise working capabilities.

3.2.2 A closer look to Eclipse

A closer look into Eclipse and how it works is provided. Each workbench is consisted by several windows as mentioned. These windows have their own selection service instance to be handled. The selection service instances are responsible to track selection activity and to propagate selection changes to all registered listeners. Selection events can be triggered either through user interaction or programmatically, and they occur when the selection in the current active part is changed or when a different part is activated.

The view where elements or text is selected does not need to know who is

interested in the selection. This gives the capability of creating new views that depend on the selection of an existing view without changing the context of the existing view.

The next sections explain who provides what kind of selections to whom.

3.3 Plug-ins

3.3.1 Plug-in Structure

Eclipse is a multifunction framework and it provides several software components (plug-ins). Eclipse Marketplace client gives the advantage of searching, and discovering popular extensions for developers to enhance their working environment. Eclipse is based on the OSGi technology, and the software components are usually packaged and distributed as OSGi bundles. OSGi bundles are very similar to standard JAR packages. An OSGi bundle must contain a manifest file with the mandatory metadata.ADD Reference: Richard hall et al osgi in action creating modular applications in java 1st greenwich ct usa manning publication co 2011 isbn 1933988916, 9781933988917. This metadata includes a name, version, activator, dependencies, API, etc.

The Plug-in Developer Environment (PDE) enables developers to extend their Eclipse IDE by creating new additional functionalities or even improve the capabilities of already implemented plug-ins. Although the reason Eclipse provides this thumping range of tools is to support developers, usually instead a chaotic environment is generated. For this reason several researchers and developers instrumented plug-ins with the aim of gathering information for developers' activity, and also to further develop Recommendation System in Software Engineers (RSSE)s.

3.3.2 Plug-in Analysis

For this thesis the plug-ins in concern are mostly observing the usage of Eclipse artifacts and recording activity as well as interactions of a developer. Several of these type of plug-ins can be found in git-hub and the Eclipse Market, emphasize was given to the following plug-ins:

Fluorite is a plug-in developed in the School of Computer Science at Carnegie Mellon university. Its purpose is low level event logging for Eclipse when us-

ing the code editor. In other words, events such as: ADD Reference: add link http://www.cs.cmu.edu/ fluorite/ character type, text cursor movement, selected text modification, as well as, all the other available Eclipse commands that can be called for an editor. Fluorite could potentially be used to extract developers' coding activity and use to detect and measure the time for various usage patterns or events of interest. However, this plug-in seemed promising and a good candidate for process mining, it was not possible to gain access to its source code.

Usage Data collector is a framework for collecting information about how developers are using the eclipse platform. Information for views usage, editor usage, changes of perspectives, actions invoked are recorder by monitors which are being used including also a timestamp. Moreover this plug-in also collects basic information about the runtime environment (OS, system architecture, window system, locale, etc). This information are uploaded periodically to servers hosted by The Eclipse Foundation with the purpose to be processed in a later stage ADD Reference: https://www.eclipse.org/org/usagedata/. Since this plug-in records relevant information it could offer a useful structure for mining the developers workflow, however, due to the fact that the development stopped a long time ago, access to source code was not achieved. The project was shut down mainly due to resource constraints. Despite the efforts of researchers, not valuable knowledge was obtained from the data.

Metrics is a plug-in that calculates various metrics for your code during build cycles and warns you of range violations for each metric. ADD Reference: https://marketplace.eclipse.org/content/eclipse-metrics. Metrics examples are the number of classes, children, interfaces, depth of inheritance, overridden methods,etc. This information can be extracted as an xml and therefore some data mining could be performed.

Mylyn is a subsystem in Eclipse used for task management. The original name of this project is Mylar, and was used in studies such as ADD Reference: how are java softwaere developers usingthe eclipse IDE because it captures events, such as preference changes, perspective changes, window events, selections, periods of inactivity, commands invoked through menus or key bindings and URLs viewed through the embedded Eclipse browser. This tool allows the developers to work in a task-focused interface, such tasks are f.x. fixing bugs, problem reports, new features. Mylyn allows developers' tasks to be organized and monitors its activity. It provides awareness for the progress of tasks, and increases the productivity by reducing unnecessary navigation, searching and scrolling. Further Mylyn allows the creation of graph elements and relationships of program artifacts.
TimeKeeper this plug-in was an extension to Mylyn in order to track time, and report how long a developer worked on a certain task.

Rabbit is a statistics tracking plug in for Eclipse. Rabbit collects data on which operation have been performed over a set period of time. Rabbit view provides developers with information regarding their command usage (copy,paste, build) used as well as their tools usage (resources, perspectives, launches). The development of Rabbit was also terminated, however documentation and source code were available on github. Rabbit was selected to further explored, more detailed analysis is provided in 3.4.

ITrace interfaces with an eye tracker, determines the location of eye gaze and map to source code element

3.4 The focal plug-in: Rabbit

Chapter 4

The focal plugin - Rabbit

4.1 what it is already and how does the plugin work - class diagram describe important methods and the logic

4.2 which features will be added and why

4.3 design and analysis of the new features

4.4 what we could achieve with process mining

TODO: explain what capturing events, patterns of interaction could be done to show the workflow

Chapter 5

Analysis and Discussion for results of process mining and rabbit

Bibliography