

# Mining a Developer's Workflow from IDE Usage

Constantina Ioannou

DTU



Kongens Lyngby 2018  
MSc-2018

Technical University of Denmark

DTU Compute

Matematiktorvet, building 303B, DK-2800 Kongens Lyngby, Denmark

Phone +45 4525 3031, Fax +45 4588 1399

[compute@compute.dtu.dk](mailto:compute@compute.dtu.dk)

[www.compute.dtu.dk](http://www.compute.dtu.dk) MSc-2018

# Abstract

---

Software developers interact with the Integrated Development Environment (IDE) by issuing commands that execute various programming tools from source code formatters to build tools. The aim of this thesis is to collect data on how developers interact with the IDE while developing software, and to mine developers' workflow from IDE usage using process mining techniques. To pursue this goal research was conducted to investigate available IDE and tools. A comparison was made based on this research and a promising tool was selected. The tool was capable of capturing developers' interactions within IDE. However, it had some limitations which needed to be overcome to change the perception on usage data and to determine the requirements for improvement.

The first requirement was the enablement of the collection of developers' interactions within an IDE regarding influenced files and the enablement of the collection of developers' interactions through their source code. A further requirement was to activate process mining. The desired improvements for the tool to meet these requirements were designed and implemented. Finally, the developed software was evaluated throughout a test case scenario which was given to developers. The methodology to refine the retrieved experimental data required was illustrated. The results of the methodology revealed an accurate depiction of developers' interactions and the analysis proved that mining developers' workflow through their interactions within IDE is possible.



# Preface

---

This thesis was carried out at the department of DTU Compute, at the Technical University of Denmark, in fulfillment 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. It analyses in detail the software which was selected for expansion and features that were developed during this project. In the end it presents the results of mining developers' interactions according to their IDE usage.



# Acknowledgements

---

My deepest thanks go to my advisor, Barbara Weber. Throughout these months, Barbara gave me the freedom for exploring an unknown field, challenged my findings, and offered valuable advice 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.

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 pursue 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! This thesis is dedicated to my best friend Alexis Charalambous who had a unique developing workflow.





# Contents

---

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Contributions . . . . .	2
1.2 Structure of thesis . . . . .	2
<b>2 Background and Theory</b>	<b>5</b>
2.1 Integrated Development Environment . . . . .	6
2.1.1 Overview of IDE . . . . .	6
2.1.2 Interactions within IDE . . . . .	7
2.2 Eclipse IDE . . . . .	9
2.2.1 Overview of Eclipse . . . . .	9
2.2.2 Plug-in Development in Eclipse . . . . .	11
2.2.3 Plugin Availability and Analysis . . . . .	12
2.3 The focal plug-in - Rabbit Eclipse . . . . .	18
2.3.1 Overview of Rabbit Eclipse . . . . .	18
2.3.2 Architecture of Rabbit Eclipse . . . . .	20
2.4 Process Mining and Disco . . . . .	25
2.4.1 Overview of Process Mining . . . . .	26
2.4.2 Overview of Disco . . . . .	27
<b>3 Problem Definition</b>	<b>31</b>

<b>4</b>	<b>Design and Implementation</b>	<b>35</b>
4.1	Design for Process Mining . . . . .	36
4.1.1	Include Timing interactions . . . . .	36
4.1.2	Include Developer's id . . . . .	37
4.1.3	Include Command categories . . . . .	38
4.2	Design for Locating command - resource interaction . . . . .	38
4.3	Design for Analyzing Java element interactions . . . . .	43
4.3.1	Detecting Java element interaction - Design 1 . . . . .	43
4.3.2	Detecting Java element interaction - Design 2 . . . . .	44
4.3.3	Comparison of Design 1 and Design 2 . . . . .	47
<b>5</b>	<b>Process Mining Analysis</b>	<b>49</b>
5.1	Disco Requirements Setup . . . . .	50
5.2	Mining Approach . . . . .	51
5.3	Results from Disco . . . . .	54
5.3.1	Result 1 - Most Commonly used Commands . . . . .	54
5.3.2	Result 2 - Developers workflow . . . . .	56
5.3.3	Result 3 - Compare Developers workflow between two classes . . . . .	57
5.3.4	Result 4 - Compare Developers workflow in a specific class . . . . .	60
5.4	Summary . . . . .	61
<b>6</b>	<b>Conclusion</b>	<b>63</b>
<b>A</b>	<b>Appendix 1: Rabbit Eclipse</b>	<b>67</b>
A.1	Commands . . . . .	67
<b>B</b>	<b>Appendix 2</b>	<b>81</b>
B.1	Experiment . . . . .	81
B.2	Disco Results . . . . .	89
	<b>Bibliography</b>	<b>101</b>

## CHAPTER 1

# Introduction

---

Programming is not a way of implementing a solution, but it is a way of thinking. This intricate procedure requires a great amount of human memory and cognition; thus, developers make use of IDEs which are environments responsible to ease software developing workflow. But, would it be possible to develop huge software only by using a compiler and a code editor? If so, how would development workflow be impacted if IDE did not exist and how long it would take to develop software?

Developers' workflow without an IDE becomes extremely slow and developers' efficiency is notably decreased. Other remarkable advantages of using IDEs are listed in [Ver17] and [Tra17].

Nevertheless, it seems that the process of software development has stagnated. This is since IDEs usually tend to overload developers by providing a considerable number of tools which result in a chaotic state as studies have shown [MMRL16], [PR12]. Based on these, several researchers ([KMCA06], [FKA<sup>+</sup>05], [MML15], [ZH13]), focused on understanding how developers operate to complete their jobs: how they structure their tasks, how they apply their strategies and how they use their tools. Their findings allowed modern IDE to begin recognizing the complexity and the memory requirements for developers. However, despite these findings none of these tools can as yet take the developer's cognition into account [PR12].

In addition, developers' interactions have frequently been used by researchers to examine developers' behaviour, including [MKF06] which highlights the most commonly used tools and [ML13a] which activated understanding of developers' workflow. More specifically, researchers in [ML13a], developed DFlow for recording development interactions within Pharo IDE.

Based on the afore-mentioned literature, it may be concluded that there was considerable potential in exploring developers' workflow within an IDE throughout their interactions and way of thinking. Consequently, it was decided to further investigate how the extraction of these interactions in a widely used IDE could be executed.

## 1.1 Contributions

The following contributions to the body of knowledge that concern process mining of developers' workflow through IDE usage are:

- a proposal for different perception on usage data to relate developers' interactions with influenced resources
- a design model and implementation of a tool capable of detecting and recording developers' interaction within a widely known IDE in regards to influenced resources.

## 1.2 Structure of thesis

To begin with, Chapter 2 provides a more specific background on the problem and its origin. It goes through previous related studies within the area of IDEs and the connection between a developer and an IDE. Next, knowledge for available IDEs and plug-ins is captured unravelling their capabilities and limitations. Selection between the most promising plug-ins is performed and the selected plugin is deeper analyzed. In the end, a background regarding the principles of process mining is provided.

Further, in Chapter 3 the goals are clearly stated and questions are raised regarding the current perception on usage data logs. An innovative perception of process mining usage data is suggested. Improvements required to meet the perception proposed on the chosen tool are described.

Later on, the design decisions which were made to fulfill requirements stated are defined and illustrated in Chapter 4. The design decisions can be sorted in: design decisions for allowing process mining, for giving a new perception on developers' interaction within an IDE and also for giving a perception on developers' source code interactions.

Thereafter, in Chapter 5, an experiment is materialized to record a data set with event logs derived from the improved version of the tool. These event logs are being refined to be able to be imported in process mining tool. Afterwards, the results accomplished are demonstrate and discussed in detail.

Finally in chapter 6, the thesis is concluded and an extensive look at future work is taken.



## CHAPTER 2

# Background and Theory

---

Since an IDE is very personal and is accustomed to developers' preferences, gathering information regarding interactions within an IDE is promising to reveal developers' workflow.

Before proceeding any further, it is absolutely required to build a solid background behind concepts used throughout the thesis related to IDE.

This chapter begins with section 2.1, which gives an introduction to how IDE evolved through time including the reasons proving IDE's significance. In addition this section describes the possible interactions of a developer within an IDE.

Among the various IDE mentioned in section 2.1, Eclipse was selected for further investigation in section 2.2. The capabilities of Eclipse and its advantages are elucidated. Further, the nature of Eclipse plug-in development is enhanced including an overview on Eclipse available plug-ins

After conducting a research on plug-ins, *Rabbit Eclipse* high potential was notable and the decision to give a great amount of focus on that specific plug-in was taken. In section 2.3 the architecture of the plug-in is presented to provide a mutual understanding.

Finally, a research is contained, in section 2.4, in regards to process mining techniques and what was required in order for it to be enabled.

## 2.1 Integrated Development Environment

An IDE is a software application which aims to improve developers' productivity by facilitating application development. It consolidates the Basic tools (f.x. source code editor, a compiler, debugger, etc) that developers need in order to write and test software. To further understand capabilities and to gain insights regarding IDEs availability, an investigation was carried out and is described throughout this section.

Section 2.1.1 primarily consists of a brief historical description on how IDEs evolved through time, and their major advantages. Next, section 2.1.2 analyzes the possible interactions within IDE.

### 2.1.1 Overview of IDE

When the development via a console or a terminal became possible, ideas regarding IDE arose. The first steps towards IDE is traced back to Dartmouth Basic which was the first language to be created with an IDE in 1964 [oDC18]. Dartmouth Basic IDE was command based, and without a graphical user interface. However, it integrated editing, file management, compilation, debugging and execution which are the necessary components to define an IDE.

According to [Ver17] Turbo Pascal, which was released in 1983 which integrated an editor and a compiler for Pascal. Later on Microsoft's Visual Basic (VB), launched in 1991 and made the process learning programming relatively easy and was used for teaching purposes. Further, followed Delphi in 1995, [Wik17] which could run on 16-bit Windows and it could provide Object Pascal.

Throughout the years several IDE were developed (Codelite, Codeblocks, dialogblocks, Netbeans, Komodo, Xamarin) [pWT14]. Nowadays, the most known and highly used nowadays are Visual Studio released in 1997 [Mic18] and Eclipse released in 2006 [Fou18a]. Today, Top IDE index [Car16], which is a ranking created by analysing how often IDEs are searched on Google, supports that the three most used tools employed to develop source code are Eclipse released in 2004, Visual Studio released in 1997, Android Studio released in 2013 [Fou18a] [Mic18] [2.518] .



Although IDEs are striking through the top three, it is noticeable 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.

Over the years, IDE have offered remarkable advantages to developers as mentioned in [Ver17] and [Tra17]. Some of these advantages for example are:

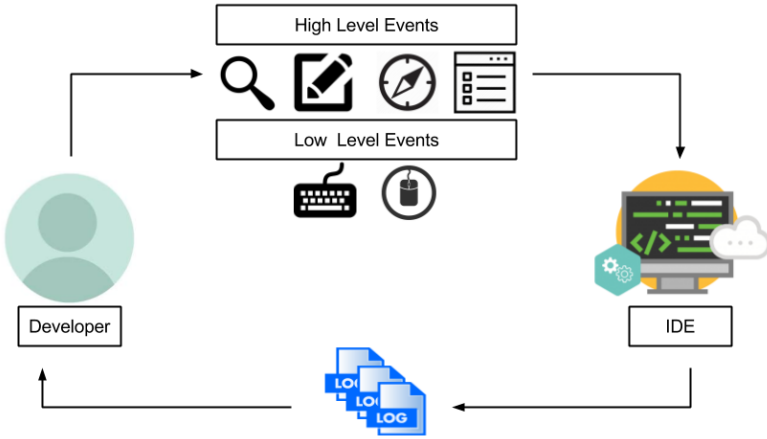
- Reduction of the setup up time required to configure multiple development tools,
- Increment of development speed and efficiency by providing instant feedback for syntax errors or code insights,
- Improvement of program management by organizing resources, providing a virtual representation of the project,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

Overall these advantages of using an IDE have endorsed the productivity of a developer and increased the number of possible interactions within an IDE significantly. However the usefulness of interactions which are further described in Section 2.1.2, IDEs lack effective support to browse between complex relationships of source code elements. This implies that developers spend 30% their time navigating through a chaotic environment, thus, their productivity is being reduced as supported by [MMRL16] [RND09] and [KMCA06]

### 2.1.2 Interactions within IDE

IDEs are highly used by developers these days to produce and maintain software. Developers use IDE to read, navigate, understand and write code. In more detail the developer interacts with IDE through low level events: mouse clicks and keyboard shortcuts but also high level events which are facilitated by context menus existing within IDE as mentioned in [KMCA06] and [ZH13]. A high

level event is a series of low level events and it allows the developer to re-factor, debug, navigate, test, and so forth without having to execute manually these steps. These interactions are referred as **usage data** in [SMHF<sup>+</sup>15] and they are an essential element of this thesis. Figure 2.1 represents the exchange of usage data (high and low level events) produced by developer’s interactions and corresponding logs are reported back from IDE.



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

An example to emphasize on what high and low level events mean as well as to show the convenience IDE offer is the following *The renaming of a class within a large project*. IDE provides contextual menus enabling fast and efficient changes. Otherwise a series of low level events such as editing several files, compiling and error seeking, executing and testing the correctness after the change will have to be done manually.

According to [ML13b] and [Gu12] gathering all the IDE usage data provides an additional perception on developers’ workflow. Analyzing and tracing the workflow of a developer throughout usage data is a challenging matter which attracted many researchers before, for example DFlow [USI14] [SMHF<sup>+</sup>15] [KMCA06].

To understand and investigate further developers’ workflow through their interactions within an IDE the selection of an appropriate IDE was required and is further described in Section 2.2.

## 2.2 Eclipse IDE

Eclipse is undoubtedly one of the most powerful currently available IDE and this is why it was selected as the IDE for this thesis. Eclipse offers various sub-project development environments, i.e., IDE and Plugin Developer Environment (PDE) and along with its structured framework it allows extensions and optimization for tools to be easily applied.

The first part section 2.2.1 presents the Eclipse IDE with a brief introduction of its capabilities including the major reasons for selecting this IDE. A basic vocabulary was provided in regards to Eclipse IDE artifacts. Adding, the next Section 2.2.2 gives an introduction to the highly adaptable Eclipse IDE and the various approaches for developing plug-in tools. Attention is then given on statistical existing plug-ins so as to select a base plug-in which allows the captivity of developer's interactions. Such plug-ins are analyzed thoroughly and presented in Section 2.2.3.

### 2.2.1 Overview of Eclipse

Eclipse IDE is a powerful UI framework used by a respectively vast amount of developers: to primarily develop Java applications or even to develop in other programming languages, and to also develop documents and packages for other software.

The standard SDK Eclipse distribution contains a base workspace with certain operations for Java development 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 multi-function framework to be used for: Embedded programming, C++ programming, javaBeans, Java application, websites, or even develop additional Eclipse plug-ins and more [Wik18].

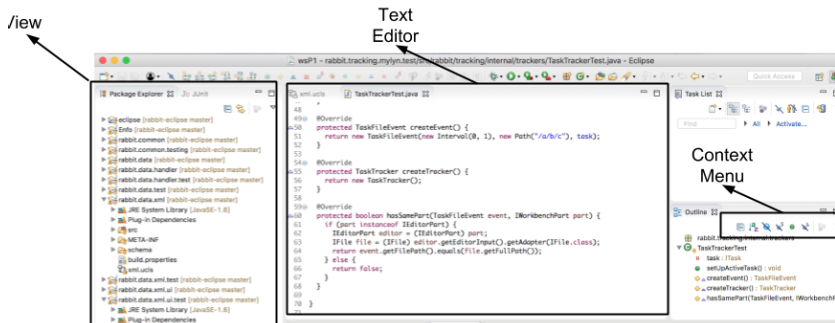
Even though Eclipse framework started as a replacement for visual age for Java from IBM [Wik18], it became an open platform for integrating tools, editors, views and plug-ins. Its source code is freely available and anyone can contribute by building their own new plug-ins or by engaging in discussions regarding integrated tools.

The most significant reasons for selecting the Eclipse IDE to analyze developers' workflow are :

- It is the top ranked used IDE by developers [Bur09] and [Car16]

- It is a home for tools, where you can build and integrate your own [Bur09]
- It is highly adjustable and enables the developers to construct its workspace according to their preferences [Bur09]
- It is an open source framework where features and plug-ins are available online so are the execution and source code files [Bur09]
- It is an open source Community [Fou18a], collection of technical professionals with a common interest in both using and contributing in the evolution of the platform

Having these reasons in mind the framework of Eclipse was chosen as the IDE for this thesis. Figure 2.2 illustrates Eclipse workspace and its surroundings, to offer understanding for the various artifacts and also to define a common vocabulary. Eclipse framework is consisted of a workbench window, which has integrated basic elements such as views, text editors, menus and active perspectives. [Fou18a]



**Figure 2.2:** A standard Eclipse workspace setup for a Java developer

A *view* is a window that enables the developer to observe and examine files or projects, like Package Explorer, Outline and Console. In general, Eclipse is in general filled up with *context menus*, the main menu is at the top of the screen while views and explorers usually provide their own context menus. Context menus enable the developer to customize the appearance of relevant information or to provide supplementary capabilities.

A *text editor* is a smart editor which recognizes the programming language, markups the syntax and allows the developer to easily modify and save files. Editors share characteristics with views, but unlike views, editors do not have context menus.

A *perspective* has its own a set of elements views and editors and menus along with an adaptable personalized layout for specific tasks such as debugging a program.

To summarize these are the main artifacts usually provided by Eclipse initial workspace setup for a java developer. These vocabulary terms are a starting point to understand concepts of IDE as well as concepts supporting plug-in development PDE. The real power of Eclipse lies within its capability of developing plug-ins for itself and is presented in the following Section 2.2.2.

### 2.2.2 Plug-in Development in Eclipse

Eclipse offers the capability of developing or evolving additional tools for itself. The PDE is based on the Open Services Gateway initiative (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 meta-data.<sup>1</sup> This meta-data includes a name, version, activator, dependencies, API and more.

Due to this technology, a good architecture is established for Eclipse. This is why developers are easily able to extend their Eclipse IDE by creating new additional operations or even improve the capabilities of already implemented plug-ins. Based on this, a thumping range of plug-in tools are available in Eclipse Marketplace client. These plug-ins can be classified in three categories according to what they offer to a developer: assistance, customization/management and statistics.

Assistance plug-ins can offer assistance to a developer while implementing a software system, like in the instance of JRebel [Zer18] which allows developer's to reload code changes instantly or, the instance of Bytecode Outline [Los17] which shows disassembled byte code of current java editor and allows developer's to have an inside look in the stack.

Moreover, there are customization and management plug-ins for example: Code-Bubbles [Rei18] which is a front end of Eclipse designed to simplify programming by generating working sets and grouping together related source fragments, or Darkest Dark [Gen18] which simply enhance and changes the user interface color to black.

Lastly, we have statistical plug-ins which provide statistics by gathering informa-

---

<sup>1</sup><https://en.wikipedia.org/wiki/OSGi>

tion from Eclipse IDE and by using graphical representations. A representative example is Metrics which calculates various statistics for your code [Flo16], or Usage data collector which captures data to help understanding how developers are using eclipse [Fou18b].

Although, as described above, Eclipse provides a variety of plug-ins to support developers, usually this results in a chaotic environment. In an attempt to reduce frustration created by the variety of plug-ins offered and in an effort to improve a developer's productivity, researchers in [GMR17] are aspiring to understand the developers' workflow by using statistical plug-ins and thereafter developing Recommendation System in Software Engineers (RSSE)s. Following ideas and concepts presented in [GMR17] the next Section 2.2.3 analyzes what available plug-ins can contribute.

### 2.2.3 Plugin Availability and Analysis

Throughout research [GMR17] the need of developing context models that can go beyond interaction events and common project artifacts is highlighted. A model consisting of thirteen contextual factors is suggested in [GMR17]. Contextual factors are variables with precise domains of possible values that are used to identify the context. The model characterizes developer situations from several perspectives, namely who, what, when and where. Using this model of

**Table 2**  
Context model.

Category	Contextual factor	Contextual factor domain
Who	Developer unique identifier	{Developer A, Developer B, ...} <sup>2</sup>
	Developer general experience Developer IDE experience	{novice, competent, expert} {novice, expert}
What	Current activity	{reading, editing, navigating, debugging, using version control, reviewing code, other}
	Previous activity	{reading, editing, navigating, debugging, using version control, reviewing code, break, other}
	Type of the artifact under development	{JavaScript file, XML file, Java class, Java method, ...} <sup>2</sup>
	Length of the artifact under development	{short, medium, long}
When	Complexity of the artifact under development	{simple, medium, complex}
	Time of the day	{morning, afternoon, evening, night}
	Day of the week	{Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
Where	IDE instance	{Eclipse Luna, Subclipse, oXygen, ...} <sup>2</sup>
	Active user interface elements	{(Java editor, XML editor, Project Explorer view), (Java editor, Console view, Outline view, Project Explorer view), ...} <sup>2</sup>
	User interface element with focus	{Java editor, XML editor, Project Explorer, ...} <sup>2</sup>

<sup>2</sup> actual domain depends on the collected data and cannot be predefined.

**Figure 2.3:** Context Model [GMR17]

contextual factors (see in Figure 2.3), as a main criteria along with taking into account the maintainability and adaptability of the plug-in, an evaluation for the capabilities of plug-ins can be derived.

Concerning the plug-ins mentioned before, they are mostly directed in gathering statistics and observing the usage of artifacts by recording activity and

interactions of a developer while developing a software system.

After conducting a research on available plug-in tools, through GitHub and the Eclipse Market, the most similar which could be used as a basis and/or be combined here are listed below and an evaluation with regards to the set criteria is done.

- Fluorite
- Rabbit
- Metrics
- Mylyn
- Time Keeper
- ITrace
- Usage Data Collector

**Fluorite** is a plug-in developed in the School of Computer Science at Carnegie Mellon university [YM11]. Its purpose is low level event logging for Eclipse when using the code editor. In other words, events such as: character type, text cursor movement, selected text modification, as well as, all the other available Eclipse commands that can be called for an editor.

In Figure 2.4 an example representing the data from Fluorite depicted from [YM11] is shown. By using the logged data, Fluorite can provide full reproduction of each source code file that has been used during a programming session. Fluorite could potentially be used to extract developers' source coding activity since it enables the detection and measurement of time for various usage patterns or events of interest like "typo correction". Among their findings, the authors provided empirical evidence that editing source code is different from editing textual documents. The authors also studied the distribution of keystrokes reporting that backspace and arrows are the most frequent keys pressed by developers.

However, this plug-in seemed promising for investigating sequences of low level events and detecting coding strategies for code editor, it was not selected for further exploitation. The main reason for not selecting this tool was its disability of recording interactions of the developer with an IDE since this tool only focused in capturing code editor interactions. A less important reason causing the rejection of this plug-in was the massive amounts of low level events which were generated and the unobtainable source code.

```
<Command __id="2" _type="MoveCaretCommand" caretOffset="142" docOffset="142" timestamp="3977"/>
<Command __id="3" _type="EclipseCommand" commandID="eventLogger.styledTextCommand.SELECT_LINE_DOWN"
timestamp="5598"/>
<DocumentChange __id="4" _type="Delete" docASTNodeCount="22" docActiveCodeLength="125" docExpression-
Count="10" docLength="151" endLine="9" length="39" offset="142" startLine="8" timestamp="7186">
  <text>
    <![CDATA[      System.out.println("Hello world!");
  ]]>
</text>
</DocumentChange>
<Command __id="5" _type="EclipseCommand" commandID="org.eclipse.ui.edit.delete" timestamp="7202"/>
<Command __id="6" _type="EclipseCommand" commandID="org.eclipse.ui.file.save" timestamp="8099"/>
```

Figure 2.4: Fluorite: Example of data log collected

**Usage Data collector** is a framework for collecting information regarding usage data of Eclipse IDE.

It was originally build by the Eclipse Foundation, as a way to measure how the community was using the IDE. Information for views usage, editor usage, changes of perspectives and actions invoked are recorded by monitors which also use a time-stamp. In Figure 2.5 an example of a data log entry is shown [SMHF<sup>+</sup>15]. Moreover this plug-in also collects basic information about the run-time 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 [Fou18b]. This project was shut down eventually due to resource constraints since hundreds of thousands of Eclipse users uploaded data, the executable code for this plug-in remains available as well as the collected data is available upon request.

vhat	kind	bundleId	bundleVersion	description	time
xecuted	command	org.eclipse.ui	3.7.0.v20110928-1505	org.eclipse.ui.edit.paste	1389111843130

Figure 2.5: Usage data collector: Example of data log collected [SMHF<sup>+</sup>15]

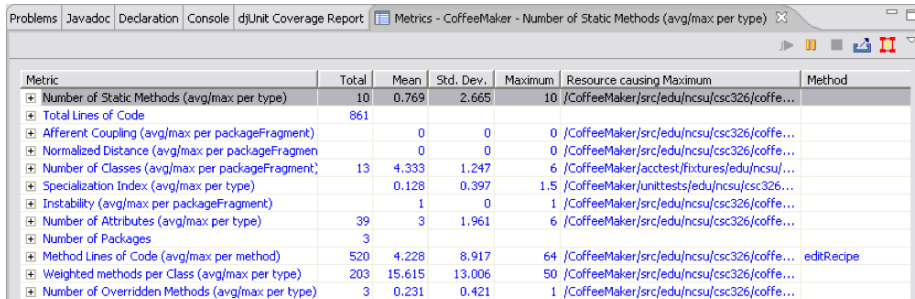
This plug-in tool records relevant information required for this thesis and for this reason is valid and useful candidate by achieving a significant amount of contextual factors 2.3. The source code was not obtainable and therefore the usage of this plug-in as a basis for this research was rejected. According to [SMHF<sup>+</sup>15] another inconvenience of this tool, is the fact that sometimes incomplete data are captured. This is because not all key bindings, menu or main toolbar bindings are addressed.

**Metrics** [Flo16] is a plug-in that calculates around 20 metrics for your code during build cycles, warns the developer of range violations for each metric, plus, it provides an XML file that contains the information. What is more, it provides representation of these metrics in tables inside the Eclipse IDE and also generates dependency graphs. The mentioned calculated metrics for instance



are McCabe's cyclomatic complexity, number of elements (classes, children, interfaces, statements, fields), depth of inheritance, etc. These metrics provide information related to experience and to programming style of a developer as referenced and further explained in [LW05].

This plug-in source code can be found online in [Git18] and in the Figure 2.6 a representation of the mentioned calculated metrics is illustrated.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Static Methods (avg/max per type)	10	0.769	2.665	10	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	
Total Lines of Code	861					
Afferent Coupling (avg/max per packageFragment)		0	0	0	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	
Normalized Distance (avg/max per packageFragment)		0	0	0	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	
Number of Classes (avg/max per packageFragment)	13	4.333	1.247	6	/CoffeeMaker/acctest/fixtures/edu/ncsu/...	
Specialization Index (avg/max per type)		0.128	0.397	1.5	/CoffeeMaker/unittests/edu/ncsu/csc326...	
Instability (avg/max per packageFragment)		1	0	1	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	
Number of Attributes (avg/max per type)	39	3	1.961	6	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	
Number of Packages	3					
Method Lines of Code (avg/max per method)	520	4.228	8.917	64	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	editRecipe
Weighted methods per Class (avg/max per type)	203	15.615	13.006	50	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	
Number of Overridden Methods (avg/max per type)	3	0.231	0.421	1	/CoffeeMaker/src/edu/ncsu/csc326/coffe...	

**Figure 2.6:** Metrics: Example of data collected [LWH05]

Metrics is analysing static code and although it can offer a perception of the experience of the developer and the complexity of the current implemented software system, it is not able to capture developers' current activity, therefore it was not selected.

**Mylyn** is a subsystem in Eclipse used for task management.[Com18] The original name of this project is Mylar, and was used in studies such as [MKF06]. Mylyn offers a task-focused interface shown in figure 2.7 (f.x. fixing bugs, new features, problem reports) to reduce irrelevant information to a task and makes multitasking easier. Mylyn monitors programming activity to create a "task context" in relation to workspace and automatically links all relevant artifacts to the task-at-hand. 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.

As an addition to Mylyn plug-in, was **TimeKeeper** which is responsible to track time, and report how long a developer worked on a certain task, and open a workweek view. Together these two plug-ins form a strong organization and management of working environment. The source code for Mylyn and Timekeeper plug-ins are available online and located at [Git11] and [Git17] respectively.

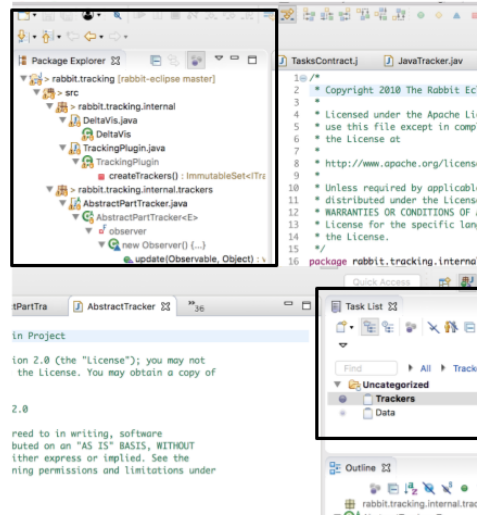


Figure 2.7: Mylyn: Example of provided interface

Even though Mylyn integrated with TimeKeeper tool and they seemed promising, the main disadvantage of using this project was that important information of developers' interaction with IDE were not captured. These information are for example: switching between views or switching between perspective. Therefore this tool was not selected.

**iTrace** is a tool integrated with an eye tracker to determine the type of element one is looking at. iTrace was released as open source project in June 7th 2015. [Uni15] While the user is reading a source code file, the tools is recording the eye movement and fixations and is able to identify the source code element for example if the user is looking at a method, or an if statement, etc. iTrace allows horizontal and vertical scrolling and is able to retrieve information only when the code is static and no interactions with the IDE through keyboard or mouse are happening. After a recording session of iTrace a data log with the fixations on the current java editor is provided in an XML form (see figure 2.8).

iTrace capability to identify source code elements by using eye movement is remarkable and the source code is available online [Git15a]. However, since developers' current activity is required to be captured to understand developers' workflow, tracking only when user is reading results is a great limitation.

**Rabbit Eclipse** is a statistics tracking tool which provides developers with information regarding their interactions within Eclipse IDE by recording low

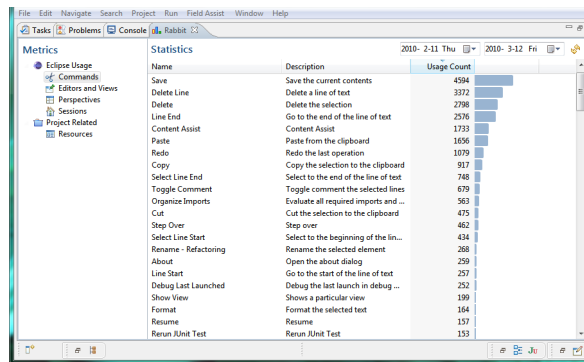
```

<response line_base_y="412" line_base_x="90" col="23" line="38"
font_height="24" line_height="37"
path="C:\program\src\tasks\program5.java"
nano_time="2969874330552" system_time="1433180043835"
tracker_time="1433178283244"
right_pupil_diameter="3.151641845703125"
left_pupil_diameter="3.095123291015625" right_validation="1.0"
left_validation="1.0" y="437" x="510" type="text"
name="program5.java"
- <sces>
  <sce type="METHOD" name="java.lang.String.length()"
end_col="30" start_col="17" end_line="38" start_line="38"
total_length="13" how="USE"/>
  <sce type="VARIABLE"
name="tasks.MiscUtilities.canonPath.trim" end_col="30"
start_col="10" end_line="38" start_line="38"
total_length="20" how="DECLARE"/>
  <sce type="IFSTATEMENT" name="IfStatement-I35c4"
end_col="5" start_col="4" end_line="51" start_line="35"
total_length="617" how="DECLARE"/>
  <sce type="METHOD" name="tasks.MiscUtilities.canonPath
(java.lang.String)" end_col="3" start_col="2" end_line="77"
start_line="17" total_length="2192" how="DECLARE"/>
  <sce type="TYPE" name="tasks.MiscUtilities" end_col="1"
start_col="0" end_line="78" start_line="15"
total_length="2270" how="DECLARE"/>
</sces>
</response>

```

**Figure 2.8:** iTrace: Example of logged data [SWW<sup>+</sup>15]

and high level events. The information is then displayed to developers by using a graphical representation and also data logs. (see figure 2.9 and 2.10). Therefore, tracking current activity of a developer within the IDE is possible.[Cod11]



**Figure 2.9:** Rabbit Eclipse UI

Rabbit eclipse carries similar ideas to Usage data collector mentioned before. Even though that its development was terminated, the source code was available on git-hub [Git15b].

Rabbit proved to be a good candidate for this thesis by achieving also a significant amount of contextual factors 2.3. The source code was obtainable and

```

commandEvents-2018-01.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<events>
  <commandEvents date="2018-01-28"/>
  <commandEvent commandId="org.eclipse.ui.edit.selectAll" count="1"/>
  <commandEvent commandId="org.eclipse.ui.edit.copy" count="4"/>
  <commandEvent commandId="org.eclipse.ui.edit.paste" count="4"/>
  <commandEvent commandId="org.eclipse.ui.edit.undo" count="17"/>
  <commandEvent commandId="org.eclipse.ui.perspectives.showPerspective" count="1"/>
  <commandEvent commandId="org.eclipse.ui.file.import" count="1"/>
  <commandEvent commandId="org.eclipse.ui.file.save" count="22"/>
  <commandEvent commandId="org.eclipse.ui.views.showView" count="2"/>
  <commandEvent commandId="org.eclipse.jdt.debug.ui.localJavaShortcut.run" count="1"/>
  <commandEvent commandId="org.eclipse.ui.newWizard" count="1"/>
  <commandEvent commandId="org.eclipse.ui.edit.delete" count="3"/>
</commandEvents>
</events>

```

**Figure 2.10:** Rabbit Eclipse: Sample of recorded event logs

easily adaptable therefore the usage of this plug-in as a basis for this research was accepted. Further reasons behind this decision, as well as how the plug-in operates, are explored and more detailed analyzed in section 2.3.

## 2.3 The focal plug-in - Rabbit Eclipse

This section aims to provide understanding regarding how *Rabbit Eclipse* plug-in operates. Firstly, an overview is contacted providing the plug-in’s capabilities and limitations in Section 2.3.1. Then, an effort to establish mutual concrete understanding for plug-in’s architecture is analyzed in Section 2.3.2. To achieve that, common usage scenarios for the plug-in and the main actors to the system are described. Further, the architecture of *Rabbit Eclipse* is depicted and the functionality of the main components is discussed.

### 2.3.1 Overview of Rabbit Eclipse

*Rabbit Eclipse* is a statistics plug-in and is responsible to collect the various usage data from the interactions performed by a developer in Eclipse IDE over a period of time. [Cod11] [jax10]

*Rabbit Eclipse* is a semi-dynamic recording tool which means that even though it silently runs in the background and gathers usage data these are only available when requested. The usage data become visible in log files, or graphical representation. These logs contain different types of interactions which are command usage; and time spent using tools such as: resources, perspectives, java elements, sessions, editors and views, launches and tasks within Eclipse.

*Rabbit Eclipse* in an effort to be more precise, it collects usage data only when Eclipse is active. In other words, if the window of Eclipse is not focused or no keyboard/mouse event has occurred during a particular amount of time then Rabbit tool consider the developer as idle and pauses tracking activities. In this way Rabbit provides more accurate data.

Due to the mentioned capabilities of deriving event logs with regards to developers' activity as well as detecting when a developer is idle led to the choice of modifying and extending this plug-in.

The main advantages behind the selection of *Rabbit Eclipse* as a base plug-in were:

- the well-structured provided online source code which could be easily understood and interpreted [Git15b]
- the availability of the main developer who willingly responded to emails.

*Rabbit Eclipse* offers these advantages and proves to be a valuable base for gathering usage data, but it also had some particular limitations.

Command's interactions within Eclipse IDE can be triggered in various ways, either by key-bindings, by contextual menus or mouse clicks. These command's interactions are further analysed by Eclipse [Bur09]. Using the information from the book an analysis in regards to which command interactions are covered by *Rabbit Eclipse* was undertaken. The results were satisfying since *Rabbit Eclipse* was able to capture the most commonly used commands [SMHF<sup>+</sup>15] A further analysis of the results regarding this procedure is presented in Appendix A.1.

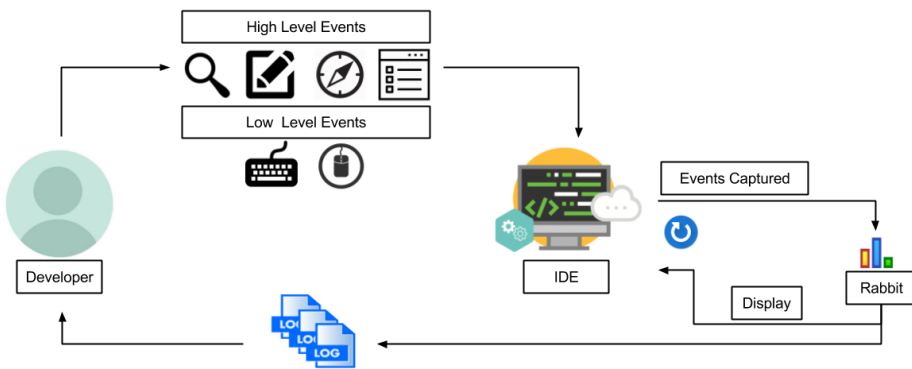
Today's developers as mentioned in [MKF06], are using only a small portion of commands available by Eclipse. Therefore, this limitation was neglected.

Moreover, *Rabbit Eclipse* was not able not recognize adjustments in terms of layout. Interactions such as maximizing window, minimizing window and adjusting workspace are not recorded. However, this limitation was not an issue, since interactions concerned with layout adjustments were out of the scope for this thesis.

As a result these limitation did not overshadow the *Rabbit Eclipse* advantages and therefore it was selected as the base plug-in for the thesis. Further, in the next section 2.3.2 the architecture of the plug-in is elaborated.

### 2.3.2 Architecture of Rabbit Eclipse

This section is destined to accomplish a mutual understanding of how Rabbit Eclipse operates, since no particular documentation of the plug-in's architecture was provided. An advantage of using *Rabbit Eclipse* was the well-structured open source code provided which enabled the ease of carrying out a study on its source code. Throughout this study the creation of a model presenting the flow of the system was possible and is displayed in Figure 2.11.



**Figure 2.11:** Model: Rabbit-Eclipse

The general followed scenario is that a developer is implementing a software within Eclipse IDE, and generates various interactions of low and high level events as described in section 2.1.2. When an interaction is initiated, *Rabbit Eclipse* is triggered to capture, and analyse the interaction. These interactions are then stored in event logs while they are exported and upon request of a developer *Rabbit Eclipse* provides captured events in a graphical view.

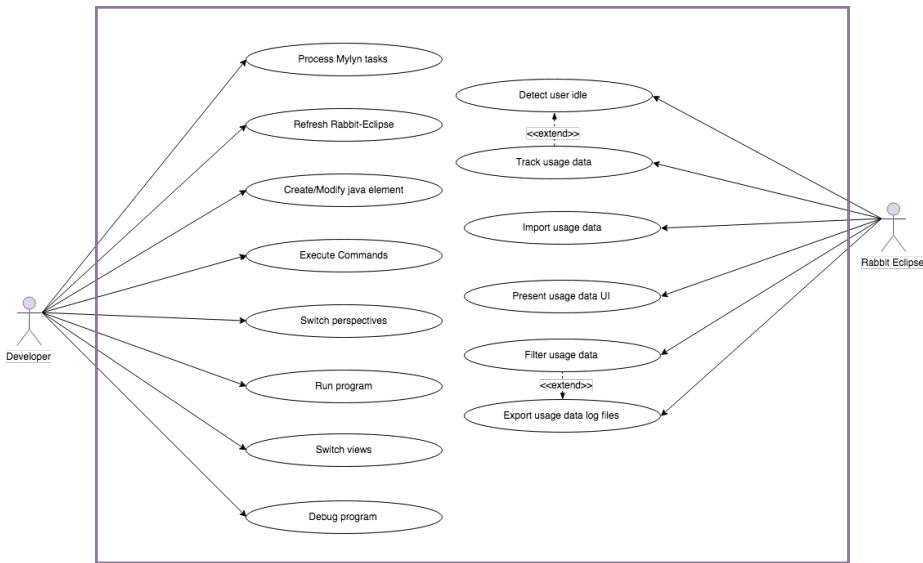
To analyse further the system, two kind of actors interacting with the Eclipse IDE, i.e., developer and plug-in are defined.

**Developer:** is a person who uses Eclipse IDE and could be running the environment through any of available operating systems (Windows, Linux, MAC). The developer does not know how to interact directly with plug-in for recording but it is only capable of obtaining through the provided UI the metrics captured. A developer can be an amateur or an expert who would like to get insights in regards to its own development workflow.

**Rabbit Eclipse** is a plug-in on the other hand, which was created to capture

activities within Eclipse IDE. The tool is built to communicate and record the data inserted in the Eclipse IDE by keeping a data structure which will then be used to output the information.

In Figure 2.12 aims to provide an idea of the most significant use cases which are accounted, for the two actors are demonstrated. The use cases for developers' interactions within Eclipse IDE are for example: execute commands, run program, debug a program, etc. On the other hand the use cases in regards to *Rabbit Eclipse* are detected user idle, track usage data, present usage data, etc.



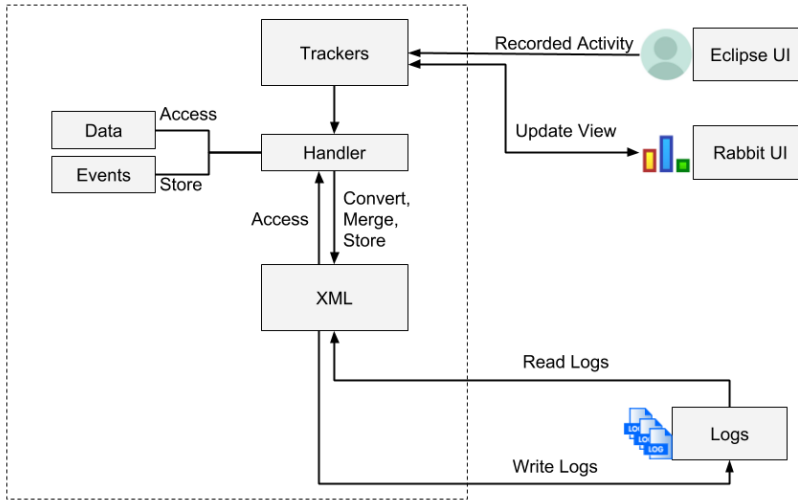
**Figure 2.12:** This figure represents an abstracted overview of use cases.

Using this information of the general scenario and how actors interact with the system the Figure 2.13 was created to represent *Rabbit Eclipse* architecture. The depicted architecture is divided into unit sections: data structure, interaction with developer, interaction with Eclipse IDE, interaction with output logs, an XML output organizers and connecting handlers. Subsequently, these units are further analyzed to explain the system thoroughly.

The data structure of *Rabbit Eclipse* is composed of the data and event units shown in Figure 2.13. The selected data structure is arranged with Immutable objects<sup>2</sup> and using Key Map Builder<sup>3</sup> which provides consistency of information

<sup>2</sup><https://docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html>

<sup>3</sup><http://lkumarjain.blogspot.dk/2013/03/why-map-key-should-be-immutable-object.html>



**Figure 2.13:** Architecture of Rabbit - Eclipse plug-in

by enabling the defensive programming technique which provides a safe usage from multiple threads. This means the objects won't run in race conditions resulting modified/corrupted states and also the objects can be easily shared by references.

The data unit contains information for interactions which has already been recorded from a previous session, whereas the event unit contains information which has just been recorded from developer's current activity. The structure of event unit is shown in the Figure 2.14. This unit consists of a variety of classes. At the top of the hierarchy is DiscreteEvent class and ContinuousEvent class which distinguish between the main types of interactions recorded, i.e. instant and continuous interactions. Command and Breakpoint are listed as Discrete interactions. On the other hand, interactions such as switching between files, views, perspectives, launching, java elements and session inherit from ContinuousEvent since the duration for such activities is relevant.

Further, the most important represented unit in Figure 2.13 is called Trackers. These units run silently in the background and record developer's interaction within Eclipse IDE.

Due to this, and as displayed in figure 2.15 recorder, observer and listener classes are used in order for tracker units to be notified when changes occur.



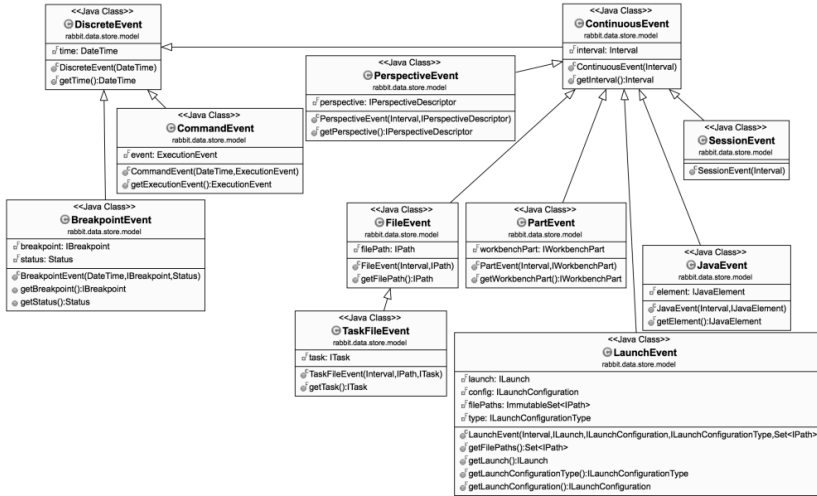


Figure 2.14: Event Classes

At the top of the hierarchy lies the AbstractTracker class which defines the necessary abilities of each tracker. These abilities are the collection of interactions, their analysis, filtering and storage.

Each tracker unit is responsible to record a different type of interaction and these types were described previously by the events structure. For example the Command tracker is responsible to explore how often each command is used (cut, copy, paste etc.), whereas, Part tracker handles time spent using different artifact within Eclipse (Java Editor, Outline View, Project Explore etc.) Also, perspective tracker is estimating how long different perspectives have been used such as Debug, Java Browsing, GitHub etc. Moreover, launch tracker is related to activities as application runs and debug runs, all relevant files to these activities are recorded as well. Sessions tracks down how much time was spent using Eclipse in total. Further, File tracker focuses on how long each file, project and other resources have been used. *Rabbit Eclipse* tracker units integrate also with Mylyn and as a result they also keep track of tasks that are listed in Mylyn. Lastly the time spent on Java Elements (classes, methods) is also captured.

These tracker units along with data handlers as presented in Figure 2.13 when a developer requests or when eclipse is open/closed will collaborate to access/store the usage data. Data handlers are in charge of keeping separated the current changes and stored changes; this means that whenever Trackers acquires that data should be updated a relevant handler is called.

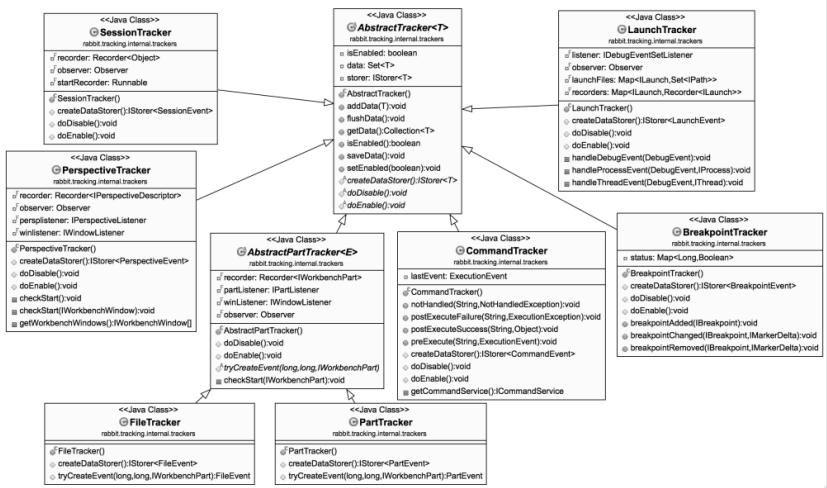


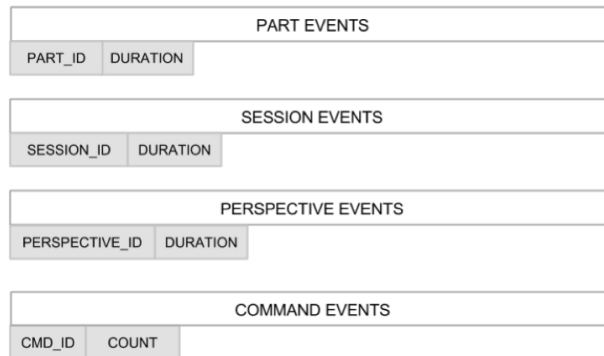
Figure 2.15: Tracker Classes

Handler units are then communicating with XML unit as it appears in Figure 2.13 to either access or store usage data. When storing information Event objects will be merged, converted and serialized using an XML schema to be exported into event logs. A depicted model to demonstrate how logs are constructed from XML writer is shown in Figures 2.16 and 2.17.



Figure 2.16: Model of usage Data log

When accessing information from logs, the usage data are de-serialized and Data objects are created and will be represented on rabbit user interface.



**Figure 2.17:** Model of usage Data log

Throughout this section, the architecture of the plug-in at-hand was analyzed, and the capabilities of the tool were detailed. Even though that *Rabbit Eclipse* seems like a promising tool and was selected as a base, the current version of the plug-in does not yet enable process mining. In the next section 2.4 a closer look to requirements for process mining is given.

## 2.4 Process Mining and Disco

Process mining is the bridge between model-based process analysis and data-oriented analysis techniques such as machine learning and data mining.<sup>4</sup> Process mining is remarkable because it automatically allows the identification and visualization of processes. As supported by [RMLvdA14] and [Say14] process mining can also be used for software development and testing process.

Section 2.4.1 gives an overview of the Process mining, describing its importance as well as its application. The different types of process mining are mentioned and also its application to software engineering is analyzed. Further, the section 2.4.2 investigates the process mining tool Disco. A short explanation of how it works and which aspects can be used to derive analysis of collected data log events are elaborated.

<sup>4</sup><https://www.coursera.org/learn/process-mining>

### 2.4.1 Overview of Process Mining

Process mining is a relatively new and emerging art of using a family of techniques which automatically observe and identify trends, patterns and sequences from collected event logs to understand and produce non-trivial useful insights regarding processes.

Today, various information systems (f.x. healthcare systems, business, etc) generate multitudes of event data. This causes organizations to have a struggle with understanding the collected events. Therefore, the desirability of improving efficiency of processes as well as availability of data enhanced the integration of process mining with data science [vdAETUETN16]. Moreover, process mining can be used for conformance of processes, identification of bottlenecks, and also prediction of execution problems [Rud15].

There are three main types of process mining <sup>5</sup> :

- Discovery: where the tool receives as input an event log and produces a process model (Petri net) explaining the behavior of the recorded log.
- Conformance: where the tool receives as input an existing process model with an event log of the same process and produces a comparison revealing whether the event log conforms the process model.
- Enhancement: where the tool receives as input an existing process model and an event log with the actual process recorded and produces an improved or extended model.

Researchers in [RMLvdA14] and [Say14] proved that process mining can be used for software development. According to this, the decision to apply process mining was taken and the most suitable type is discovery since this project aims to discover and understand the workflow of a developer within an IDE.

Since process mining is a cutting edge technology, several process mining tools were invented for example most common are ProM [vDAV<sup>+</sup>05], Disco <sup>6</sup>, Celonis <sup>7</sup>.

Disco tool was proposed by the supervisors and used throughout this thesis, in the next section 2.4.2 a further description on how the tool works is provided.

<sup>5</sup><https://www.coursera.org/learn/process-mining>

<sup>6</sup><https://fluxicon.com/disco/>

<sup>7</sup>[https://www.celonis.com/?gclid=EAIaIQobChMI2fWo3fan2QIVFyjTCh17xQ\\_vEAYASAAEgK5q\\_D\\_BwE](https://www.celonis.com/?gclid=EAIaIQobChMI2fWo3fan2QIVFyjTCh17xQ_vEAYASAAEgK5q_D_BwE)

### 2.4.2 Overview of Disco

Disco is a complete process mining toolkit that makes process mining rapid, flexible and easy. Disco helps to process raw data provided in event logs using visual maps, providing metrics and statistics, as well as filtering from activities or paths.<sup>8</sup> Moreover, despite the fact that Disco offers graphical representation of the raw data, it also allows the creation of animation to visualize the process at-hand with a video animating the process. The visualisation feature is relatively helpful to identify bottlenecks of the process.

An overview of the most significant parts of Disco depicted from<sup>9</sup> follows, since the knowledge behind how Disco operates was a crucial part for this thesis.

The first step to process mining is to organize the raw data sets which will be used. Disco tool was designed to allow the user to import data easily. The tool allows to open a CSV or Excel file with the raw data sets in a read-only format. In figure 2.18 and 2.19 the 3 steps to follow when importing the raw data sets are shown. Firstly, the selection of each column and secondly configuration of each column according to what is representing is possible.

The configuration categories are defined as follows:

- **case ID**, which determines the scope of the process. Using that the process mining tool can compare several execution of the process to one another, it uniquely identifies a single execution of the process.
- **activities**, which determines the level of detail for the process steps. Using that, each step of the process is described, some of these steps might occur more than once for a single case.
- **time-stamps**, which determines when the activity took place. This is important to analyze timing behavior and also to provide a sequence between the activities.
- **additional attributes** are defined as either resources or attributes and they are optional. They are helpful to filter the analysis to become more precise.

It is mandatory to have at least one case id, and for the activities and also a time-stamp are prerequisites for the tool to be able to process mine the given

---

<sup>8</sup><https://fluxicon.com/disco/>

<sup>9</sup><https://fluxicon.com/disco/files/Disco-User-Guide.pdf>

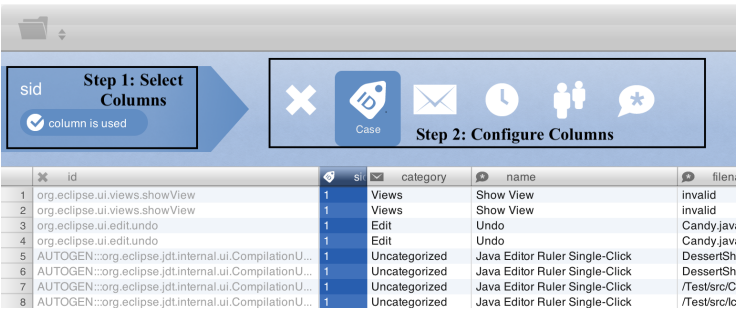


Figure 2.18: Steps 1 and 2 required to insert raw data sets into disco

event logs. The third step is to import the data as shown in 2.19. Once the event log is imported, an automatic process discovery mapping is provided by Disco and graphical representation and other statistics are generated.

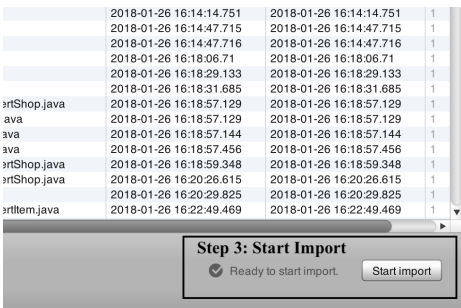
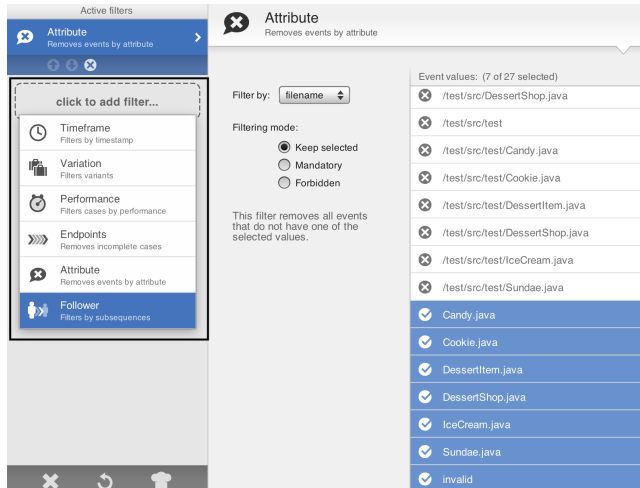


Figure 2.19: Step 3 required to insert raw data into disco

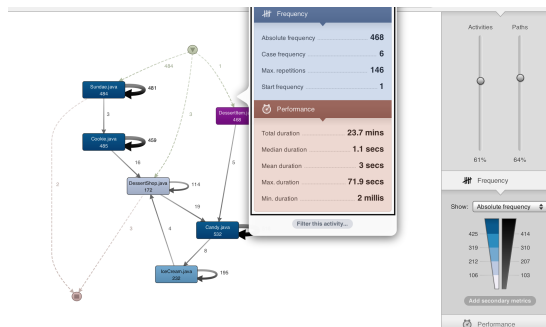
Furthermore, there are 6 different types of filters provided in Disco which are highly helpful and allow the ease of interaction, filtration and inspection of the event log( see Figure 2.20).

The *time frame filter* which enables to focus on apart of a log executed in a certain period of time. Next is the *variation filter* which enables the filtering either the most common or exceptional behavior in a process. The *performance filter* which is filtering in relation to a specific set of performance criteria. These performance criteria could be case duration, number of events, case utilization, active time, or waiting time. Following is the *end point filter* which trims out events that are not in the boundary of the beginning and finishing activity. Also the *attribute filter* which allows the selection of events based on activity and resources. Last but not least the *followers filter* which filters by sub-sequences.



**Figure 2.20:** Available filtering from Disco

This filtering capability of Disco provides all the information required to analyse more efficiently the process. While process discovery Disco offers automatic process discovery by providing information regarding an activity for Frequency and Performance as appeared in figure 2.21. Moreover, Disco allows the regulation of how detailed the mapping solution should be by providing the option to set the percentage of activities and paths to be shown.



**Figure 2.21:** Available filtering from Disco

Using this tool in a later stage of this thesis will provide understanding for developers' workflow.





## CHAPTER 3

# Problem Definition

---

**The purpose** of this thesis is to capture data on how developers interact with IDE while developing software and to mine developers' work-flows from IDE usage using process mining techniques.

To detect available and potential technology candidates that can offer recording of developers activity within IDE, a research was carried out and is presented in chapter 2. The selection of Eclipse IDE and *Rabbit Eclipse* as the tools at-hand was made and the reasons behind this decision are analyzed in previous sections 2.2 and 2.3, respectively.

Although, the chosen technologies could provide broad and profound information regarding developers' interactions within Eclipse IDE, deriving any further conclusions using process mining tool Disco regarding developers' workflow was rather inconvenient.

Therefore in order to pursue the purpose of this thesis, modifications on how the selected tools operate were applied and also the perception of captured events was changed. This means that the tools had to be modified from asking how many times an interaction occurred or how long did it last, to ask:

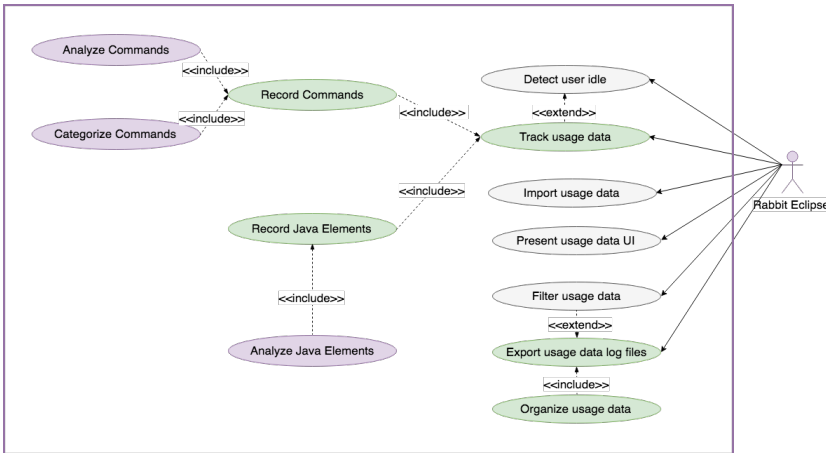
1. when the interaction occurred

2. which resource or resources were influenced during the interaction

Further, in order to enable process mining of the event logs, they are required to have a case id and time-stamps, as mentioned in Section 2.4. In addition, to provide process mining in a more abstract level, a classification of commands was also a requirement.

Moreover, since Java events have been recorded before from *Rabbit Eclipse*, as mentioned in Section 2.3.2, exploring the ability to process mine source code was set as a secondary goal.

Figure 3.1 depicts a new use case diagram which highlights the modified and added use cases required to enable process mining for *Rabbit Eclipse*.



**Figure 3.1:** Use case diagram showing new use cases (purple), modified uses cases (green), and unchanged use cases (grey)

The modified use cases presented in the diagram are related to how data and log events are captured by *Rabbit Eclipse*.

The use case *Export usage data* log files refers to the modifications made to XML schema constructing event logs and the use case *Organize usage data* was modified to record also when an interaction occurred, instead of only estimating duration by merging and converting usage data.

The new use cases created are related to how interactions are interpreted by *Rabbit Eclipse*. The use case *Categorize Commands* provides a classification for

available command events. Further, the use case *Analyze Commands* serves the purpose of identifying resources influenced by an interaction.

Lastly, *Analyze Java elements* use case identifies details in regards to recording source code interactions.

These mentioned use cases are vital to the modifications made in the architecture and components of *Rabbit Eclipse* and the design in regards to them is further elucidated in the next chapter 4.



## CHAPTER 4

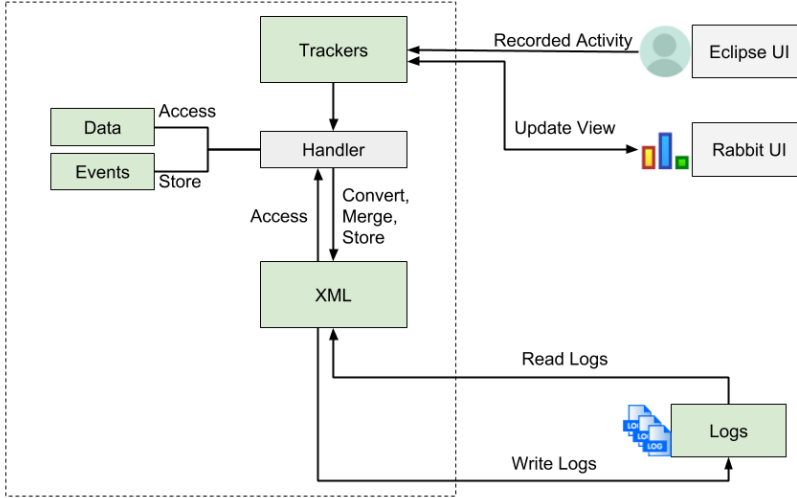
# Design and Implementation

---

In this chapter the mandatory modifications for achieving the requirements described in Chapter 3, are modeled and analyzed thoroughly.

The design derived to accommodate the requirements set is described throughout the next sections. In the first Section 4.1 of this chapter, the design which is required to enable process mining is described. Further, the next section 4.2 provides a model suggesting the correlation of command and file interactions. Last but not least, the section 4.3 provides two design solutions which enable process mining of source code.

To start with, Figure 4.1 represents the architecture of *Rabbit Eclipse* and highlights units which will be modified. In a first glance, the modified units are: XML unit which is used during storage or accessing of event logs, Data and Event units which contain information for interactions, and also Trackers which are responsible to record developers' interactions. On the other hand Handler units and the Rabbit UI unit remain unchanged.



**Figure 4.1:** Architecture of *Rabbit - Eclipse* plugin diagram showing modified units (green) and unchanged units (grey)

## 4.1 Design for Process Mining

This section presents the modeling steps towards process mining. Firstly, attention is given to model timing interactions in section 4.1.1, developers id in section 4.1.2 and classification of commands in section 4.1.3.

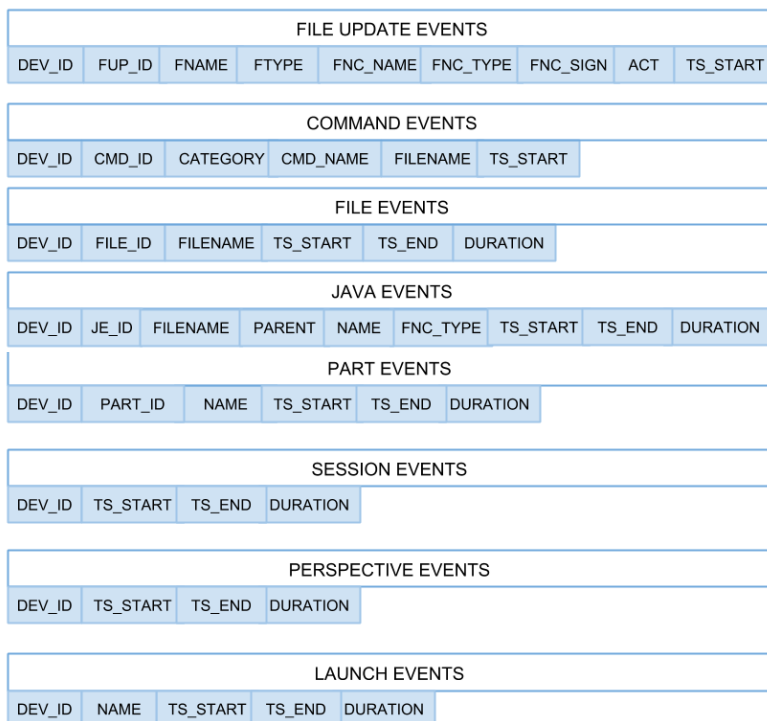
### 4.1.1 Include Timing interactions

The first requirement is to gather timing information and more specifically to obtain when a developer interacted with IDE.

Based on the previous version of *Rabbit Eclipse* plugin, elaborated in Section 2.3, interactions were collected and the total duration of an interaction was calculated. XML units, were responsible to register one entry per interaction in the event log with only the total duration of the interaction.

To approach this situation, *Rabbit Eclipse* plugin was modified. The modification made was in accordance to XML units which were directed not to merge the events but to generate and register new entry for every interaction that took

place. As it is visible in Figure 4.2, the exported usage data logs have been endorsed with fields addressing the time. A time-stamp of when an interaction was started and also a time-stamp when an interaction was ended, (TS\_START) and (TS\_END) respectively, were added. The time registered is standardized to follow the SQL format time-stamp (YYYY - MM - DD HH:MM:SS).



**Figure 4.2:** Models of event logs

### 4.1.2 Include Developer's id

A requirement to allow process mining, as mentioned in Section 2.4, is to have a case id. In order to provide a case id, the addition of a field representing the id to all entries in event logs exported by *Rabbit Eclipse* was mandatory and is visible in Figure 4.2.

The case id was designed to enable two approaches for process mining. These

approaches are (a) mining interaction of an individual developer over different development stages such as implementing, debugging and testing and (b) mining interaction of a developers group executing the same task.

To enable both, the refresh button which is visible on the interface of *Rabbit Eclipse* was enhanced. In the first case, to allow process mining of different development stages, the refresh button was designed so that when a developer presses the button a new development stage would be identified, and therefore the case id would increase to indicate the stage change. However, this provides inaccuracies since, the judgment of changing the sessions is based on the human factor.

On the other hand, to fulfill the second case the refresh button is not taken into consideration and the software assigns a constant number in the field of case id.

### 4.1.3 Include Command categories

To accompany further process mining, as described in Chapter 3, an abstraction level of command interactions was required. Command interactions were classified by following the origin categories contained in the book [Bur09]. In relation to this *Categorize Commands* use case was realized.

The previous version of *Rabbit Eclipse* gathered origin category for every command interaction. However, it was not registered in the exported event logs, mentioned in Section 2.3.2. To realize this the data structure along with the related XML schema of *Rabbit Eclipse*, were modified to accommodate the required field to include category in command interaction logs.

## 4.2 Design for Locating command - resource interaction

A significant requirement which was defined in Chapter 3 was to locate which resource is influenced by a developer's interaction within IDE.

The previous version of *Rabbit Eclipse* provided captivity of File events and Command events by using File and Command tracker units, which were analyzed in Section 2.3.2. File tracker unit was responsible to capture the duration of interaction with a file, whereas Command tracker unit was responsible to explore metrics regarding the usage of commands. However the fact that both



type of events were captured, the tool did not provide any correlation between them. Therefore, this inconvenience limits the capabilities of the tool towards process mining.

To resolve the limitation and pursue the requirement of locating which resource was influenced by developers' interaction, major changes have been made on Command tracker unit to allow the combination of these two interaction types, i.e. File and Command.

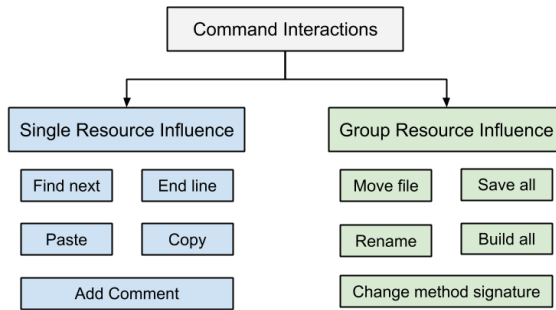
In order to combine these two interaction events, File and Command a sequence of several steps were followed.

### Step 1: command influence classification

To begin with, a list with available standard command interactions was obtained from [Bur09]. Going through the list, as mentioned in Section 2.3.2, it revealed all combinations of keyboard bindings and menu bindings which *Rabbit Eclipse* can capture. Using this information the realization of what influence a captured command interaction can have on different resources, i.e. files or projects, was possible. Therefore, all commands could be characterized by their influence on resources, and these are detailed in Appendix A.1.

This led the decision to distinguish between categories of command interactions. These categories, demonstrated in Figure 4.3, are single resource influence which refers to the influence a command has its only on a single file (f.x. copy, end line) and group resource influence which refers to the influence a command has on a project, or a group of files (f.x. move file, rename).

This procedure distinguished the categories regarding influence command interactions have on resources, and therefore provided knowledge in how many resources a command can influence. The next step to accomplish combination of the File and Commands interactions is to determine the specific resources that are influenced.



**Figure 4.3:** Examples of the two categories distinguished are single resource influence and group resource influence

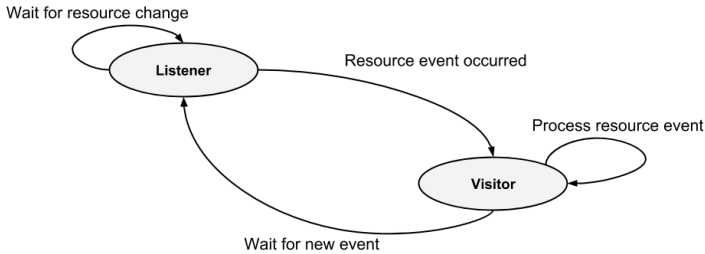
## Step 2: influenced resource detection

In order to support the need of determining the specific resources which were influenced by a command interaction, was to create a new tracker unit, specifically focusing on resource changes(i.e, an addition, a removal or an edition). This tracker unit was responsible for keeping track of when and which resources have been changed.

In Figure 4.4 a behavioral state machine to describe the behavior for this unit is depicted. The state machine is using the design patterns of a listener and a visitor [SS03]. The listener is waiting until a command execution interaction occurred which caused the change for resources. When the listener detects this interaction then it collects a resource event and triggers a visitor. A visitor is responsible to go through the resource event in order to determine what was the type of change (addition, removal, edition) and most importantly to determine the name of the resource influenced.

After the visitor processes the resource event, the relevant interaction information captured which are the name of the resource influence, the type of change on the resource and a time-stamp are stored. To accommodate this, a data structure to maintain the interaction is required was created following the *Rabbit Eclipse* data structures, mentioned in Section 2.3.2.

The described procedure explains how the resource names which were influenced by a command interaction are gathered.



**Figure 4.4:** Behavioral state diagram, to show the relation between the listener and the visitor

### Step 3: Combination of command interaction and influenced resource

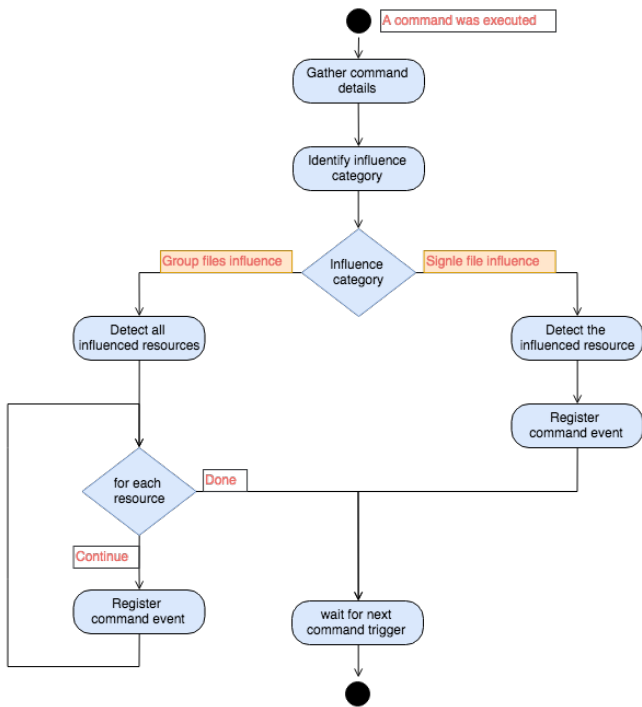
At last, in order to realize the combination of file and command interactions, there are two aspects to be considered. These aspects are how a command interaction is captured and how a command interaction is stored.

To start with, Figure 4.5 demonstrates the use case *Analyze Command* which realizes the first aspect of how a command interaction will be captured.

In detail, once a command interaction is detected, the details(command name, time stamp, etc.) for that interaction are gathered. Then, the command interaction is being processed to identify in which influence category it belongs to, single or group resource influence. According to which influence category the command belongs to, the influenced resources are captured. In case of group resource influence, for each resource a new command interaction is registered, otherwise only one new command interaction is registered.

Further, to realize the second aspect, of how a command interaction is registered, data structure along with the related XML schemas of *Rabbit Eclipse*, mentioned in Section 2.3.2, were modified to accommodate the required field to include file name.

An example of a common command - file interaction entry is shown in Figure 4.6



**Figure 4.5:** Activity diagram - Analyze Command describes the functionality required to capture a command-file interaction

COMMAND EVENTS	
DEV_ID	0
CMD_ID	org.eclipse.ui.edit.copy
CATEGORY	Edit
CMD_NAME	Copy
FILENAME	IceCream.java
TS_START	2018-01-24 18:10:04.401

**Figure 4.6:** Command - file interaction entry

## 4.3 Design for Analyzing Java element interactions

As a secondary purpose of this thesis as mentioned in Chapter 3, the requirement of analyzing Java element interactions was set to enable process mining of source code.

Based on the previous version of *Rabbit Eclipse* plugin, elaborated in Section 2.3.2, java element interactions were recorded. Java Element interactions are interactions on source code elements, (f.x a change of method signature or the class). Although *Rabbit Eclipse* recorded information in regards to these interactions, only an identifier and duration for each element was exported into the event logs.

To undertake this situation, currently there are two methodologies that can be followed.

### 4.3.1 Detecting Java element interaction - Design 1

The first methodology was to modify the data structure along with the related XML schema of *Rabbit Eclipse*, to accommodate the additional required information. The addressed additional information fields are shown in Figure 4.7.

The fields included are listed:

- JE\_ID which is the handle identifier id
- FILE\_NAME which is simply the name of the resource which contains the element
- PARENT which resource is the parent of the current resource
- FILE\_TYPE which indicates whether the resource is a class or an interface (5 - class, 9 - interface)
- TS\_START which indicates when this java element interaction started
- TS\_END which indicates when this java element interaction finished
- DURATION for how long this element has been modified

JAVA EVENTS	
DEV_ID	0
JE_ID	Test/src&lt;{DessertItem.java
FILE_NAME	DessertItem.java
PARENT	DessertItem.java
FILE_TYPE	5
TS_START	2018-01-24 18:03:31.986
TS_END	2018-01-24 18:03:34.229
DURATION	2243

**Figure 4.7:** Java Element - Java interaction entry

### 4.3.2 Detecting Java element interaction - Design 2

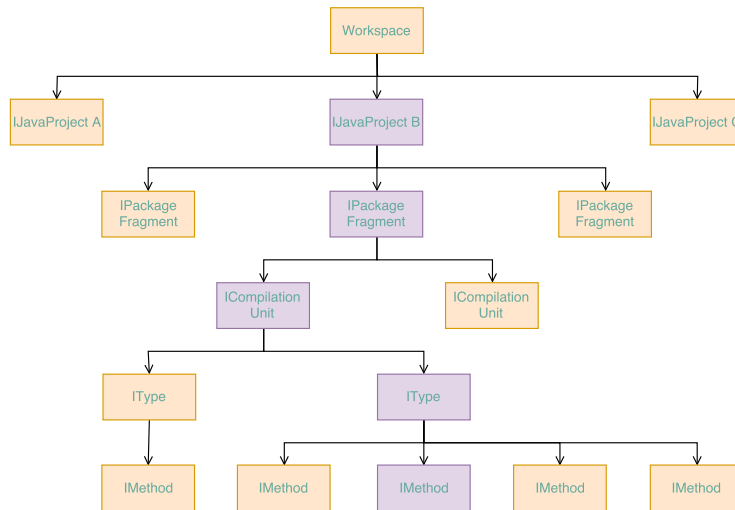
The second methodology was to built another tracker responsible to retrieve java element interactions through the Abstract Syntax Tree (AST) constructed by Eclipse [SS03].

In order to do that several steps were followed. Firstly, the detection of when a java element interaction occurred, and secondly to define information for the interaction are required.

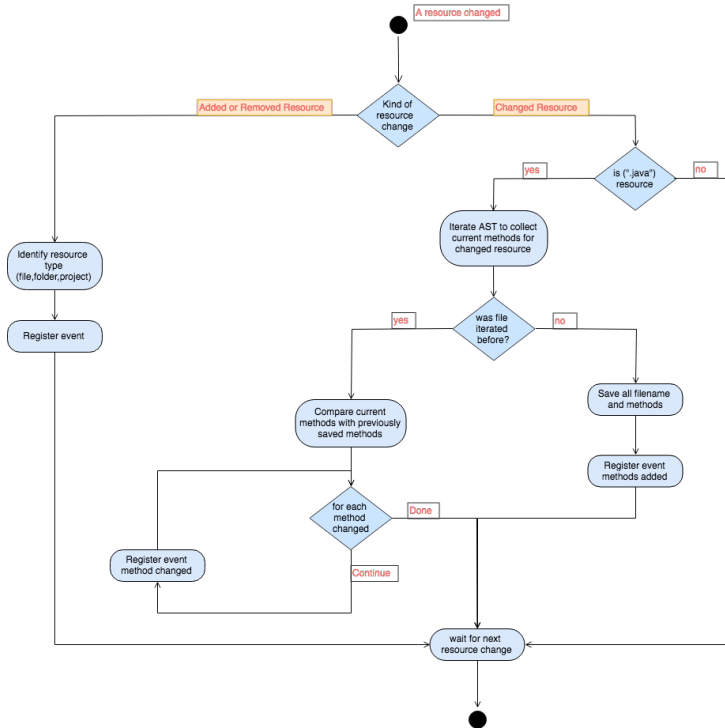
To start with, a java element interaction implies a change of source code element, within a resource file. Therefore, to detect when a java element interaction occurred, recording when a resource file was modified is sufficient. The design from the previous Section 4.2 for detecting updated files is used.

In order to define information for the interaction, the AST is used. The AST is a detailed tree representing the Java source code. For each project inside a workspace the AST is able to recognize the java elements, as mentioned in the book [SS03]

Further, the AST is updated when a resource file has been modified. The structure of AST is demonstrated in Figure 4.8, and it suggests that to retrieve information for a java element resource from AST, is required to iterate the tree. By using the AST to retrieve information for java element interaction and the resource file, the diagram represented in Figure 4.9 demonstrates the use case *Analyze Java Elements*.

**Figure 4.8:** AST example

In detail once a resource change is detected, timing information and resource name are gathered for the java interaction. If the resource was added/removed then identification of what was added/removed is done (file, folder) and these events are registered. In case of an entire project deletion this branch of the diagram will be executed several times. On the other hand, if the changed resource was a type of java source code file then is further examined. An iteration of the updated AST and the current methods of the java file at-hand are gathered. Further, if the file was not previously modified, then the file at-hand is a new entry. This means that all the methods detected from the previous stage need to be stored and registered as new events. Otherwise, a comparison between the newly retrieved methods and the stored methods for the file-at hand. Finally the results from this comparison are registered as a new entry in the event log for all methods added, removed or changed. Thereafter, the process awaits for a next resource change.



**Figure 4.9:** Activity Diagram: Analyze java elements

In addition the XML units had to be modified to accommodate this new type of recording java element interactions. An additional data structure along with the related schema for *Rabbit Eclipse* was created. In Figure4.10 shows the listed fields:

- FUP\_ID which is the full path to the corresponding resource file
- FILE\_NAME which is simply the name of the resource
- FILE\_TYPE which indicates whether the resource is a file, a folder, or a project
- METHOD\_NAME which is the name of the method
- METHOD\_SIGN which is set as the signature of the method
- METHOD\_TYPE which is set as the return type of the method



- METHOD\_ACT which identifies whether the method was added, removed, or edited.
- TS\_START which indicates when this java element interaction occurred

FILEUPD EVENTS	
DEV_ID	0
FUP_ID	/Test/src/IceCream.java
FILE_NAME	IceCream.java
FILE_TYPE	file
METHOD_NAME	getCost
METHOD_SIGN	()I
METHOD_TYPE	()I
METHOD_ACT	ADDED METHOD
TS_START	2018-01-24 18:07:29.519

**Figure 4.10:** Fileupd - java interaction entry

### 4.3.3 Comparison of Design 1 and Design 2

To sum up, both designs mentioned in Section 4.3.1 and in Section 4.3.2 have been implemented to enhance the secondary requirement this thesis, which was to enable process mining of source code.

To pursue this, the exported event logs were built to server the minimum requirements for process mining, case ID, activity and time-stamp as referenced in Section 2.4.

The exported java and fileupd event logs are similar, and serve the same purpose of tracking a java element interaction. However, there are two fundamental differences. The first difference is that Design 1 provides a compact version of the java interaction, whereas Design 2 is more detailed and provides gathered method information (i.e. sign, type, name, action). The other difference is that Design 1 provides a starting time-stamp and ending time-stamp whereas Design 2 only provides a starting time-stamp.

Even though, both of the designs could be used to provide process mining for java element interactions.



## CHAPTER 5

# Process Mining Analysis

---

As discussed before in Section 2.4, process mining is an evolutionary technique which can also be used to provide valuable knowledge for software engineering [RMLvdA14] and [Say14]. This Chapter is dedicated to apply process mining on the recorded event logs from the new implemented version of *Rabbit Eclipse*. The attempt to reveal that process mining developers' through interactions within Eclipse IDE is possible is pursued by initiating an experiment. During this experiment, *Rabbit Eclipse* was able to retrieve the event logs from participants and enable process mining successfully.

In the first Section 5.1 of this Chapter the requirements to enable process mining are analyzed. Further in Section 5.2, the approach followed to refine the raw event logs retrieved from *Rabbit Eclipse* plugin is explained. Thereafter, the refined event logs are imported into Disco and the derived conclusions are presented in Section 5.3. Last but not least, Section 5.4 converse about the overall results of this procedure.

## 5.1 Disco Requirements Setup

To enable process mining using Disco, the selection among the three different techniques analyzed in Section 2.4 was required. Among them the most suitable was the discovery technique. This technique is used when only an event log is available for input and produces a process model explaining the behavior of the recorded log.

Further, Disco requires the event log to be provided in a CSV or EXCEL format and also to contain at least a case ID, an activity, and a time-stamp as mentioned in Section 2.4.

For this reason, the most promising event logs generated by the *Rabbit Eclipse*, modeled in the previous Chapter 4, are shown in Figure 5.1. These event logs were modeled and implemented to meet the requirements for Disco. These logs precisely collect when (TS\_START, TS\_END) and where (FILENAME) interactions have occurred. Further, to establish a case ID the events were provided with an identifier (DEV\_ID), where this identifier is used to distinguish which developer invoked the interaction.

FILE UPDATE EVENTS								
DEV_ID	FUP_ID	FNAME	FTYPE	FNC_NAME	FNC_TYPE	FNC_SIGN	ACT	TS_START

COMMAND EVENTS					
DEV_ID	CMD_ID	CATEGORY	CMD_NAME	FILENAME	TS_START

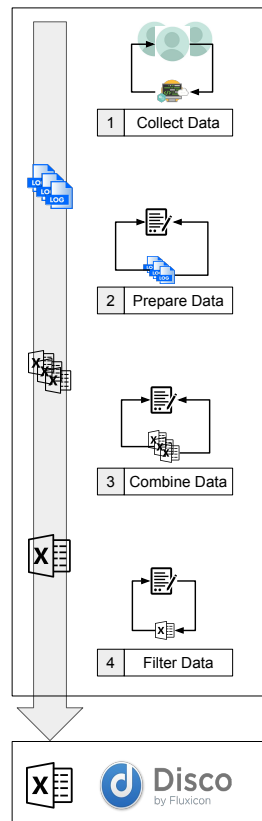
JAVA EVENTS								
DEV_ID	JE_ID	FILENAME	PARENT	NAME	FNC_TYPE	TS_START	TS_END	DURATION

**Figure 5.1:** The event logs used for process mining

Since, the modified version of *Rabbit Eclipse* can successfully retrieve these organized event logs, the decision to retrieve a larger data set was taken. An experiment to reveal developers' workflow was designed and a plan of action was constructed to refine the raw event logs retrieved from the experiment. In the following Section 5.2 the experiment and the plan of action are detailed.

## 5.2 Mining Approach

In this section, the experiment to retrieve raw event logs from developers is designed and the methodology to refine the data set retrieved from *Rabbit Eclipse* is indicated. The steps required to be followed in order to design and materialize the experiment are shown in 5.2.



**Figure 5.2:** Process Mining approach

### Step 1: Collect data

The first step in the approach is the collection of event logs to create a data set. The current data set was derived from 6 participants who have an academic

background. Most of the participants were familiar with Eclipse IDE, however, their level of experience within Eclipse IDE differs.

Participants were assigned a specific task and were given guidelines to setup the tool in their Eclipse IDE. In Appendix B, the task and the guidelines are presented in detail. Participants had to import the plugin to their Eclipse first and then follow the task. The task required the participant to create a small project implementing an inheritance java example. A description as well as a class diagram were provided, in order to clarify the task and to define names to be used for classes and methods.

The reason behind requiring participants to follow the predefined naming is to allow feasibility of process mining with regards to affected resources. Otherwise, if participants were given the permission to use their own naming, then is highly probable that a great variation between resources naming would be generated, and therefore correlation of data would be hardly possible.

## Step 2: Prepare data

Once the tool is established in Eclipse then it begins to capture interaction events. This usually results the creation of noise events such as:

- data set information, (f.x. date see Figure 5.3), which is defined when a developer restarts Eclipse. This information is recorded, but is not useful, since the desired event log is required to only contain the fields defined previously in Section 5.1.
- unrelated interactions, which are interactions with other projects existing in the current workspace
- folder, project interactions (see Figure 5.4), which are out of the scope of this analysis due to the nature of the assigned task.

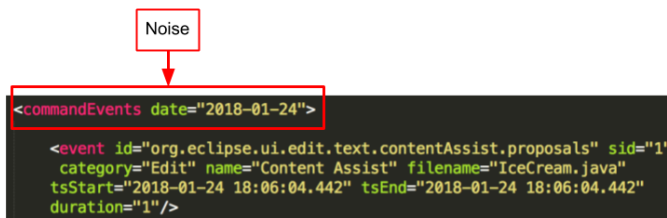
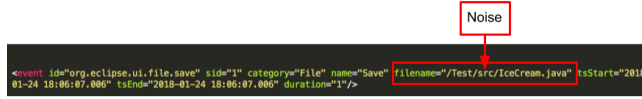


Figure 5.3: Noise created by data set information



**Figure 5.4:** Noise created by interactions with folders and projects

These noise events are removed manually from event logs. The last modification to prepare the data set is to convert the exported XML format event logs to CSV format. To make this possible a program which converts an XML file to a CSV file was created.

At the end of this step, the noise have been removed and the XML logs have been converted to CSV logs.

### Step 3: Combine Data

The third step in the process as suggested in Figure 5.2 is to combine event logs of the same type (Command logs, FileUpd logs, etc.). As a result one CSV file for every type is created.

### Step 4: Filter Data

The last step show in Figure 5.2, is to filter the data of CSV files. This means that all entries generated which are invalid will be filtered out. An invalid entry is an entry which either it has been wrongly named or it has been marked with the filename field as invalid. A wrongly named entry is for example, when a participant, instead of using correctly the requested naming *"Test.java"*, used *"test.java"*. While the filename field is marked as invalid when the participant begins to interact with Eclipse IDE without initially selecting a file.

Furthermore, one of the participants produced a considerable amount of interactions compared to the others. Consequently, the choice to filter out this participant's event logs was taken.

### Step 5: Disco

The final step is to access and shed more light into the data set which have been captured and refined. The data set is imported to Disco process mine tool and the results obtained are elaborated in Section 5.3.

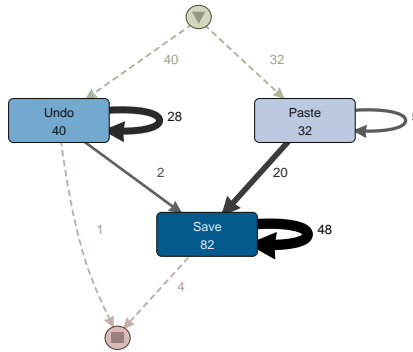
## 5.3 Results from Disco

Throughout this section, the refined event logs gathered as described in the previous Section 5.2 and are used to derive results for understanding of developers' workflow within the environment of Eclipse. To do that, the refined event logs are imported to Disco. As introduced in Section 4.1, the option for case-id, activity, time-stamp, resource are defined and this enables the discovery process. The derived disco diagrams are illustrated and discussed.

### 5.3.1 Result 1 - Most Commonly used Commands

The aim for this part is to reveal the most commonly used commands by developers.

To do that firstly, the command event log exported from Rabbit Eclipse was imported to Disco. The selected options for the discovery process were: case-id = developer id, activity = Command names, time-stamp = TS\_START.



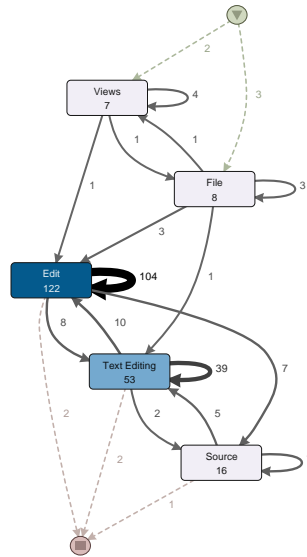
**Figure 5.5:** The most commonly used commands.

Figure 5.5 shows that the most common commands used by participants, are Save, Undo, Paste. Similarly, in a previous study [MKF06] which was related to Java developers' command usage within Eclipse, the most commonly used commands were Save, Undo, Paste.

Secondly to investigate the commands further, the classification of commands was exploited, in Section 4.1.3. The command event log was used to derive this result. The selected options for the discovery process were: case-id = developer



id, Activity = Categories, time-stamp = TS\_START, Other Attributes = command names and Filenames. In Figure 5.6 the most commonly used categories are demonstrated. These categories are Edit and Text-Editing which are related to interactions of a developer within a java file.



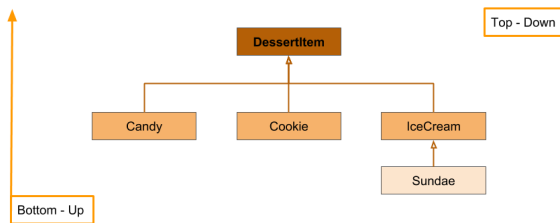
**Figure 5.6:** The most commonly used categories of commands.

### 5.3.2 Result 2 - Developers workflow

This part focuses on presenting the developers workflow within Eclipse IDE.

To start with, there are different workflow techniques a developer can follow. Two common workflow techniques for object oriented programming, addressed in [DK03], are a top-down approach and a bottom up approach. In a top-down approach, a developer begins with the main problem and then subdivides it into sub-problems. Whereas, in a bottom-up approach, a developer begins with sub-problems building up to the main problem. In relation to the assigned task, Figure 5.7 displays the techniques.

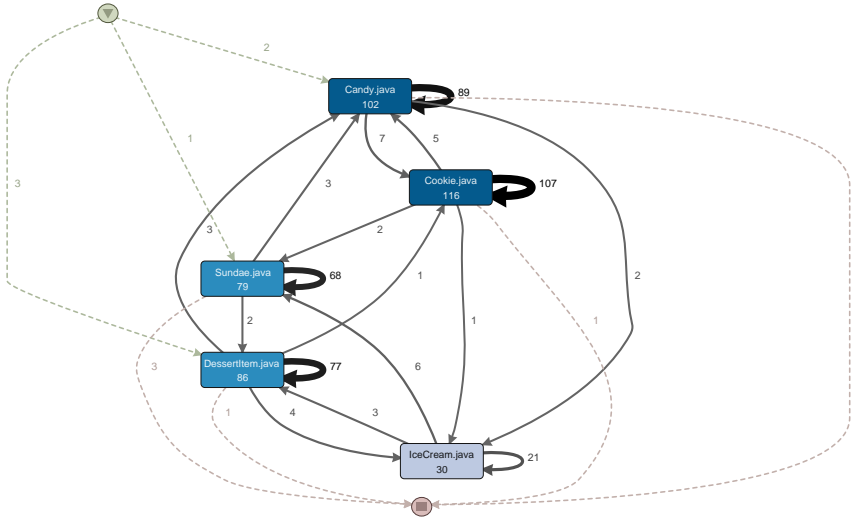
In order to exploit these techniques, the command event log was used. The selected options for the discovery process were: case-id = developer id, Activity = File names, time-stamp = TS\_START, Other Attributes = command names.



**Figure 5.7:** Class diagram related to bottom up and top down techniques for the given task.

In Figure 5.8 the results generated from Disco are illustrated. Half of the participants followed top down approach by selecting to begin with *DessertItem.java* which was the superclass. The other half of the participants followed a bottom up approach by selecting to begin implementation with *Candy.java* or *Sundae.java* which inherit from the superclass *DessertItem.java*.

Moreover, from this figure 5.8 is also observable that *Cookie.java* is the file with the most interactions in comparison to the *IceCream.java* file which had the less interactions. This is due to the reason, that *Cookie.java* was slightly more complicated than *IceCream.java*.



**Figure 5.8:** The class files (Candy, Cookie, Sundae, DessertItem, IceCream) which are requested to be created by the developer throughout the given task

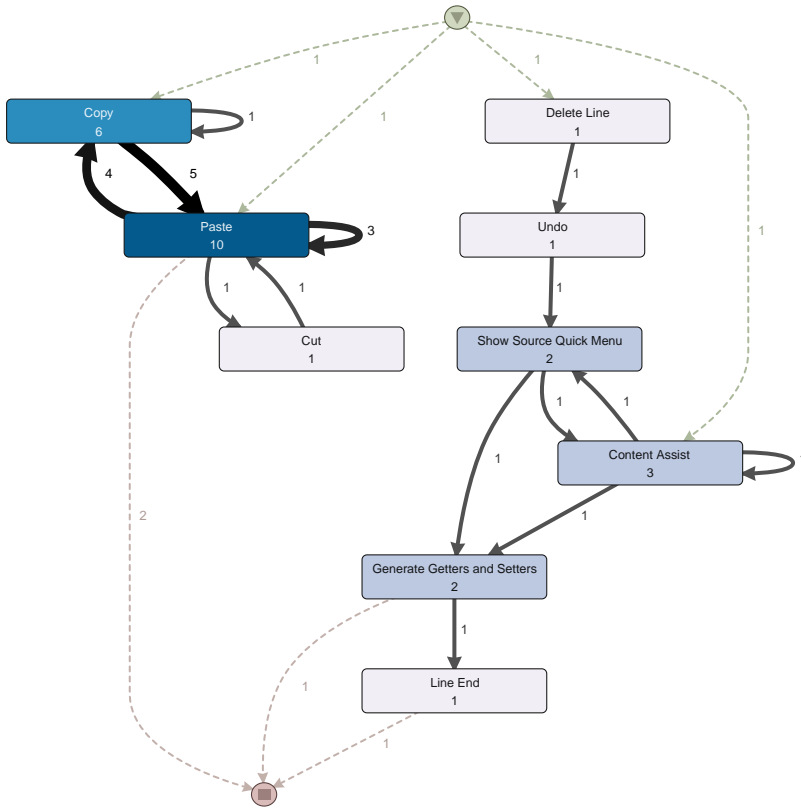
### 5.3.3 Result 3 - Compare Developers workflow between two classes

With regards to the observation made in Section 5.3.2 for the amount of interactions in *Cookie.java* and *IceCream.java*, an investigation was carried out on these two files. Scope of this investigation is the comparison of how developers treat files with different difficulty level.

To make this possible the command event log was imported to Disco. The selected options for the discovery process were: case-id = developer id, activity = Command names, time-stamp = TS\_START and attributes = filename and categories.

For Figure 5.9 a filter to keep only the interactions related to *Cookie.java* was enabled and similarly for Figure 5.10 the filter was set to *IceCream.java*.

What is prominent in these Figures, 5.9 and 5.10, is the commands appeared belong mainly to Edit and Text - Editing categories (see Appendix A.1). More

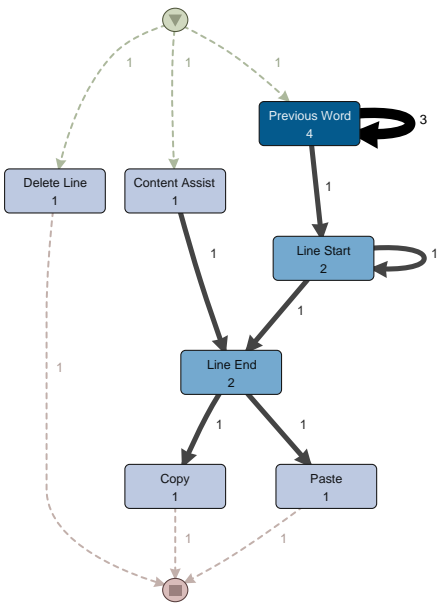


**Figure 5.9:** The commands used during the implementation of *Cookie.java*

specifically, commands like Undo, Copy, Paste belong to Edit category and commands like Line start, Line End, Previous word, Delete Line belong to Text Editing category.

It is noteworthy that *Cookie.java* has more Edit commands than *IceCream.java* whereas, *IceCream.java* has more Text-Editing commands than *Cookie.java*. From these findings is shown that the developers' workflow changes when dealing with larger source files.

Another distinct observation drawn from Figure 5.9 is the implementation approaches for *Cookie.java*. One approach is, when participants used Edit com-



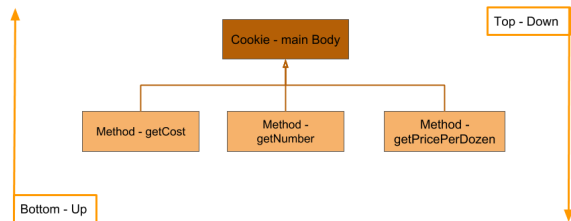
**Figure 5.10:** The commands used during the implementation of *IceCream.java*

mands (Copy, Paste, Cut) and the other approach is when participants used Text Editing and Source commands (Quick Menu, Content Assist, Generate getters and setters). The variation of participants' experience within Eclipse IDE is highlighted by the variety of usage of commands participants chose while implementing the task.

5.3.4 Result 4 - Compare Developers workflow in a specific class

In this part developers’ workflow through the implementation of a specific class is studied.

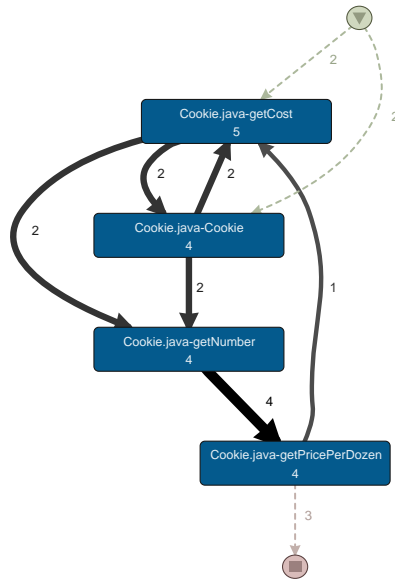
The java event log was imported to Disco and the selected options for the discovery process were: case-id = developer id, activity = method name and file-name, time-stamp = TS\_START and attributes = action (f.x. added, removed, changed as described in Section 4.3).



**Figure 5.11:** Methods and main body of *Cookie.java* related to bottom up and top down techniques

The workflow techniques top down and bottom up, as mentioned before in Section 5.3.2, also apply for java source code. An example is shown in Figure 5.11, to illustrate that top down approach means implementing the main body of the class first and then implementing the methods in detail. Whereas, bottom up approach, means implementing method by method the main body of the class.

For Figure 5.12 a filter to keep only the interactions related to *Cookie.java* was enabled and similarly for Figure 5.13 the filter was set to *Sundae.java*. These figures clearly show the different approach developers had while implementing these files.



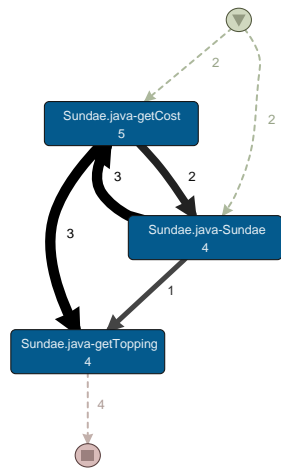
**Figure 5.12:** The java events interaction in regards to a specified class called *Cookie.java*

## 5.4 Summary

Overall this experiment, *Rabbit Eclipse* was able to retrieve the event logs from participants and enable process mining successfully.

An understanding of developers' workflow throughout their interactions in Eclipse IDE was accomplished by:

- Identifying the most commonly used commands and categories of commands, (see result 1 in Section 5.3.1)
- Identifying the two object oriented programming techniques, top down and bottom up (see result 2 in Section 5.3.2 and result 4 in Section 5.3.4)
- Identifying how developers' workflow might diverse when building a small or a large source code (see result 3 in Section 5.3.3)



**Figure 5.13:** The java events interaction in regards to a specified class called *Sundae.java*

Nevertheless, while inspecting the retrieved interactions, the realization that the particular given task to developers was relatively small came through. The capabilities of the implemented version of *Rabbit Eclipse* to capture the influenced files were not fully exploited, since the task did not require any high re-factoring, or navigation.



# Conclusion

---

In this study, the approach towards mining developers' workflow, throughout their interactions within an IDE, was investigated. This topic was chosen due to its considerable potentials and due to the fact that there was a notable room for improvement. The methodology was structured in three main axes. Firstly, the investigation, which contained the analysis and selection of potential tools, suitable for the extraction developers' interactions from IDE. Secondly, the design and implementation, which included the changes required for the enablement of the collection of developers' interactions within an IDE and process mining. Thirdly, the process mining analysis, in which the implemented tool was evaluated, by initiating an experiment with external users - developers, the workflow of whom was revealed. Finally, conclusions were carried out examining to which extent the initial targets were accomplished and identifying future improvements.

Throughout the conducted investigation, the most widely known IDEs were reviewed and the IDE with the most adjustable framework - Eclipse - was selected. In more detail existing plug-ins provided by Eclipse were analyzed. In total 7 plug-ins capable to retrieve developers' interaction were identified and compared based on their capabilities and adaptability. With respect to this, Rabbit Eclipse plugin was chosen to be further investigated for several reasons, most important of which were its well-structured online provided source code including its accuracy in recording developers' interactions performed by a developer.

Subsequently, the architecture of Rabbit Eclipse was overviewed thoroughly, in order for its operation to be understood and its limitations to be detected. The last part of the research contained the investigation of process mining technique and what was required in order for it to be enabled.

Although, Rabbit Eclipse provided statistical results on developers' interaction, it did not enable process mining. Due to this limitation, two major modifications were required on its operation in order to accommodate process mining. The first was the adjustment of the tool in order to record time-stamps, a case id, and a classification. The second regarded the revision of tool's perception on developers' interactions. In order for the latter to be achieved, it was essential that developers' interaction were combined with the influenced resources. Furthermore, a change of secondary importance was made in order to provide process mining on source code. Finally, since the guidelines for implementing the modifications and the faced challenges for Rabbit Eclipse were designed, the implementation was realized.

After the implementation of the mentioned modifications, Rabbit Eclipse was capable of recording the developers' interactions with respect to process mining requirements as desired. To realize this, an experiment was initiated in order to prove the valid functionality of the tool. A data set was created, which contained recorded developers' interactions from six participants. This data set was refined following a created by the author methodology in order to remove noise and enable the application of process mining.

Ultimately, three significant derived results were analyzed and illustrated in diagrams. The first one was the successful identification of the most commonly used commands and categories. The second one was the identification of two object oriented programming techniques, which are top down and bottom up. The last one was the identification of how developers' workflow might diverse when implementing a small or a large source code. These results indicate that finally it is possible to mine developers' workflow from their interactions within an IDE.

It is expected that this thesis work will inspire successors and encourage them to conduct a further research towards mining developers' workflow by means of their interactions within an IDE. Being a promising, adaptable and easily adjustable tool, Rabbit Eclipse can be utilized for further investigations with regards to mining developers' workflows.

It is proposed that a more elaborated research can be carried out in order to expose the full capabilities of the tool implemented during the thesis. Specifically, an experiment that explores refactoring and correcting bugs via an unknown code, could potentially reveal promising results in terms of the workflow.

Another crucial feature embedded into the Rabbit Eclipse is the capability of recording developers' source code interactions. Hence, this feature enables an onward investigation of the process mining source code.



## APPENDIX A

# Appendix 1: Rabbit Eclipse

---

In this appendix, additional information regarding the *Rabbit Eclipse* are presented.

## A.1 Commands

Throughout this appendix section, tables representing the available commands are gathered. These tables are separated according to [Bur09] different categories. Each table contains 7 columns, command name, key binding, v, menu binding, v and located. For each command, the key binding and menu binding are retrieve from the book [Bur09]. Further these two columns are followed by a column called V, which is indicating whether or not *Rabbit Eclipse* is capable of capturing the information from the specified command. Last and most important the column Located defines on which files the command at-hand has an effect when executed.

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Add Bookmark</i>		-	Edit-> add bookmark	+	ON FILE
<i>Add Task</i>		-	Edit-> add task	+	ON FILE
<i>Content Assist</i>	ctrl+space	+	Edit-> content assist	+	ON FILE
<i>Context Information</i>	ctrl+shift+space	+	Edit-> parameter hints	+	ON FILE
<i>Copy</i>	ctrl+c   ctrl+Insert	+	Edit->copy	+	ON FILE
<i>Cut</i>	ctrl+x   shift+delete	+	Edit->cut	+	ON FILE
<i>Delete</i>	Delete	+	Edit->Delete	+	ON FILE
<i>Find and Replace</i>	Ctrl+F	+	Edit->Find/Replace	+	ON FILE
<i>Find Next</i>	ctrl+K	+	Edit->Find Next	+	ON FILE
<i>Find Previous</i>	ctrl+shift+K	+	Edit->Find Previous	+	ON FILE
<i>Incremental Find</i>	ctrl+J	+	Edit-> Incremental Find Next	+	ON FILE
<i>Incremental Find Reverse</i>	ctrl+shift+J	+	Edit-> Incremental Find Previous	+	ON FILE
<i>Paste</i>	ctrl+V   shift+Insert	+	Edit-> Paste	+	ON FILE
<i>Quick Diff Toggle</i>	ctrl+shift+Q	+		+	ON FILE
<i>Quick Fix</i>	ctrl+1	+	Edit->Quick Fix	+	ON FILE
<i>Redo</i>	ctrl+Y	+	Edit->Redo	+	ON FILE
<i>Restore Last Selection</i>	alt+shift+down	+	Edit->Expand Selection To -> Restore Last Selection	+	ON FILE
<i>Revert Line</i>		-		-	
<i>Revert to Saved</i>		-	File->Revert	-	
<i>Select All</i>	ctrl+A	+	Edit->Select All	+	ON FILE
<i>Select Enclosing Element</i>	alt+shift+up	+	Edit->Expand Selection To -> Enclosing Element	+	ON FILE
<i>Select Next Element</i>	alt+shift+right	+	Edit->Expand Selection To -> Next Element	+	ON FILE
<i>Select Previous Element</i>	alt+shift+left	+	Edit->Expand Selection To -> Previous Element	+	ON FILE
<i>Shift Left</i>		-	Source->Shift Left	+	ON FILE
<i>Shift Right</i>		-	Source->Shift Right	+	ON FILE
<i>Show Line Numbers</i>		-		-	
<i>Show Tooltip Description</i>	F2	+	Edit->Show Tooltip Description	+	ON FILE
<i>Toggle Insert Mode</i>	ctrl+shift+insert	+	Edit->Smart Insert Mode	+	ON FILE
<i>Undo</i>	ctrl+Z	+	Edit->Undo	+	ON FILE
<i>Word Completion</i>	alt+/	+	Edit->Word Completion	+	ON FILE

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Close</i>	ctrl+F4   ctrl+W	+	File->Close	+	NEXT FILE
<i>Close All</i>	ctrl+shift+F4   ctrl+shift+W	!	File->Close All	!	CLOSE ALL - ERROR
<i>Exit</i>		-	File->Exit	+	
<i>Export</i>		-	File->Export	+	Active file but may affect another file
<i>Import</i>		-	File->Import	+	Active file but may affect another file
<i>Move</i>		-	File->Move	+	Active file but may affect another file
<i>New</i>	ctrl+N	+	File->New->Other	+	Active file but may affect another file
<i>New Menu</i>	alt+shift+N	+	File->New	+	Active file but may affect another file
<i>Open File</i>		-	File->Open File	+	Active file but may affect another file
<i>Open Workspace</i>		-	File->Switch Workspace	+	Active file but may affect another file
<i>Print</i>	ctrl+P	+	File->Print	+	ON FILE
<i>Properties</i>	alt+enter	+	File->Properties	+	ON FILE
<i>Refresh</i>	F5	+	File->Refresh	+	ON FILE
<i>Rename</i>	F2	+	File->Rename	+	Active file but may affect another file
<i>Revert</i>		-	File->Revert	+	Active file but may affect another file
<i>Save</i>	ctrl+S	+	File->Save	+	Active file but may affect another file
<i>Save All</i>	ctrl+shift+S	+	File->Save All	+	Active file but may affect another file
<i>Save As</i>		-	File->Save As	+	ON FILE

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Back</i>		-	Navigate -> Goto -> back	+	ON PROJECT
<i>Backward History</i>	alt+left	+	Navigate->Back	+	ON FILE
<i>forward</i>		-	Navigate ->Goto->Forward	+	ON PROJECT
<i>Forward History</i>	alt+right	+	Navigate->Forward	+	ON FILE
<i>Go Into</i>		-	Navigate->Go Into	+	ON PROJECT
<i>Go to Line</i>	ctrl+l	+	Navigate->GotoLine	+	ON FILE
<i>Go to matching bracket</i>	ctrl+shift+P	+	Navigate->Goto-> Matching Bracket	+	ON FILE
<i>Go to Next member</i>	ctrl+shift+down	+	Navigate-> Goto -> Next Member	+	ON FILE
<i>Go to Package</i>		-	Navigate-> Goto->Package	+	ON PROJECT
<i>Go to Previous member</i>	ctrl+shift+up	+	Navigate->Goto->Previous Member	+	ON FILE
<i>Go to resource</i>		-	Navigate->Goto->Resource	+	ON PROJECT
<i>Go to Type</i>		-	Navigate->Goto->Type	+	ON PROJECT
<i>Next</i>	ctrl.	+	Navigate->Next	+	ON FILE
<i>Open external Javadoc</i>	shift+f2	+	Navigate->Open External javadoc	-	
<i>Open Declaration</i>	f3	+	Navigate->Open declaration	+	ON FILE
<i>Open cAll Hierarchy</i>	ctrl+alt+h	+		-	ON FILE
<i>Last edit location ctrl+Q</i>		-	Navigate->Last Edit Location	+	ON FILE
<i>Open structure</i>	ctrl f3	+		-	Active file but may affect another file
<i>Open super implementation</i>		-	Navigate -> Open Super Implementation	+	Active file but may affect another file
<i>Open Type</i>	ctrl+shift+T	+	Navigate->Open Type	+	ON FILE
<i>Open Type Hierarchy</i>	f4	+	Navigate->Open Type Hierarchy	+	Active file but may affect another file
<i>Open Type in Hierarchy</i>	ctrl+shift+H	+	Navigate->Open Type in Hierarchy	+	Active file but may affect another file
<i>Previous</i>	ctrl,	+	Navigate->Previous	+	ON FILE
<i>Quick Hierarchy</i>	ctrl+T	+	Navigate->Quick Type Hierarchy	+	ON FILE
<i>Quick OutLine</i>	ctrl+O	+	Navigate->Quick Outline	+	ON FILE
<i>Show in Menu</i>	alt+shift+w	+	Navigate->Show In	+	Active file but may affect another file
<i>Show in Package</i>		-	Navigate->Show In ->Package explorer	+	Active file but may affect another file
<i>Up</i>		-	Navigate->Goto->Up one level	-	



<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>CSV Repository Exploring</i>		-	Window->Open Perspective->Other	+	ON FILE
<i>Debug</i>		-	Window->Open Perspective->Debug	+	ON FILE
<i>Java</i>		-	Window->Open Perspective->Java	+	ON FILE
<i>Java Browsing</i>		-	Window->Open Perspective->Java Browsing	+	ON FILE
<i>Java Type Hierarchy</i>		-	Window->Open Perspective->Java Type Hierarchy	+	ON FILE
<i>Team Synchronizing</i>		-	Window->Open Perspective->Team Synchronizing	+	ON FILE

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Build All</i>	ctrl+b	+	Project->Build All	+	ON PROJECT
<i>Build Clean</i>		-	Project->Clean	+	ON PROJECT
<i>Build Project</i>		-	Project->Build Project	+	ON PROJECT
<i>Close Project</i>		-	Project->Close Project	+	ON PROJECT
<i>Generate Javadoc</i>		-	Project->Generate Javadoc	+	ON PROJECT
<i>Open Project</i>		-	Project->Open Project	+	ON PROJECT
<i>Properties</i>		-	Project->Properties	+	ON PROJECT
<i>reBuild All</i>		-		-	
<i>reBuild Project</i>		-		-	
<i>repeat Working Set Build</i>		-		-	

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Add Class Load Breakpoint</i>		-	Run->Add Class Load Breakpoint	+	ON FILE
<i>Add Java Exception Breakpoint</i>		-	Run->Add Java Exception Breakpoint	+	ON FILE
<i>Debug ant Build</i>	alt+shift+d   q	+	Run->Debug ant Build	+	ON FILE
<i>Debug Eclipse Application</i>	alt+shift+d   e	+	Run->Debug Eclipse Application	+	ON FILE
<i>Debug Java applet</i>	alt+shift+d   a	+	Run->Debug Java applet	+	ON FILE
<i>Debug Java Application</i>	alt+shift+d   j	+	Run->Debug Java Application	+	ON FILE
<i>Debug JUnit Plug-in Test</i>	alt+shift+d   p	+	Run->Debug JUnit Plug-in Test	+	ON FILE
<i>Debug JUnit Test</i>	alt+shift+d   t	+	Run->Debug JUnit Test	+	ON FILE
<i>Debug Last launched</i>	F11	+	Run->Debug Last launched	+	ON FILE
<i>Debug SWT Application</i>	alt+shift+d   s	+	Run->Debug SWT Application	+	ON FILE
<i>Debug</i>		-	Run->Debug	+	ON FILE
<i>Display</i>	alt+shift+d	+	Run->Display	+	ON FILE
<i>EOF</i>	ctrl+z	+	console view only	+	ON FILE
<i>Execute</i>	ctrl+u	+	Run->Execute	+	ON FILE
<i>External Tools</i>		-	Run->External Tools	+	ON FILE
<i>Inspect</i>	ctrl+shift+i	+	Run->Inspect	+	ON FILE
<i>Profile Last Launched</i>		-	Run->Profile Last Launched	+	ON FILE
<i>Profile</i>		-	Run->Profile	+	ON FILE
<i>Remove All Breakpoints</i>		-	Run->Remove All Breakpoints	+	ON FILE
<i>Resume</i>	F8	+	Run->Resume	+	ON FILE
<i>Run ant Build</i>	alt+shift+x   q	+	Run->Run ant Build	+	ON FILE
<i>Run Eclipse Application</i>	alt+shift+x   e	+	Run->Run Eclipse Application	+	ON FILE
<i>Run Java applet</i>	alt+shift+x   a	+	Run->Run Java applet	+	ON FILE
<i>Run Java Application</i>	alt+shift+x   j	+	Run->Run Java Application	+	ON FILE
<i>Run JUnit Plug-in Test</i>	alt+shift+x   p	+	Run->Run JUnit Plug-in Test	+	ON FILE
<i>Run JUnit Test</i>	alt+shift+x   t	+	Run->Run JUnit Test	+	ON FILE
<i>Run Last launched</i>	ctrl+F11	+	Run->Run Last launched	+	ON FILE
<i>Run Last launched external tool</i>		-	Run->Run Last launched external tool	+	ON FILE
<i>Run SWT Application</i>	alt+shift+x	+	Run->Run SWT Application	+	ON FILE
<i>Run to Line</i>	ctrl+R	+	Run->Run to Line	+	ON FILE
<i>Run</i>		-	Run->Run	+	ON FILE
<i>Skip All Breakpoints</i>		-	Run->Skip All Breakpoints	+	ON FILE
<i>Step Into</i>	F5	+	Run->Step Into	+	ON FILE
<i>Step Into selection</i>	ctrl+F5	+	Run->Step Into selection	+	ON FILE
<i>Step Over</i>	F6	+	Run->Step Over	+	ON FILE
<i>Step Return</i>	F7	+	Run->Step Return	+	ON FILE
<i>Suspend</i>		-	Run->Suspend	+	ON FILE
<i>Terminate</i>		-	Run->Terminate	+	ON FILE
<i>Terminate and Relaunch</i>		-	Run->Terminate and Relaunch	+	ON FILE
<i>Toggle Line Breakpoint</i>	ctrl+shift+b	+	Run->Toggle Line Breakpoint	+	ON FILE
<i>Toggle Method Breakpoint</i>		-	Run->Toggle Method Breakpoint	+	ON FILE
<i>Toggle Step Filters</i>	shift+F5	+	Run->Toggle Step Filters	+	ON FILE
<i>Toggle watchpoint</i>		-	Run->Toggle watchpoint	+	ON FILE

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Change Method Signature</i>	alt+shift+c	+	Refactor->Change Method Signature	+	Active file but may affect another file
<i>Convert Anonymous Class to Nested</i>		-	Refactor->Convert Anonymous Class to Nested	+	ON FILE
<i>Convert Local Variable to Field</i>	alt+shift+f	+	Refactor->Convert Local Variable to Field	+	ON FILE
<i>Encapsulate Field</i>		-	Refactor->Encapsulate Field	+	ON FILE
<i>Extract Constant</i>		-	Refactor->Extract Constant	+	ON FILE
<i>Extract Interface</i>		-	Refactor->Extract Interface	+	ON FILE
<i>Extract Local Variable</i>	alt+shift+l	+	Refactor->Extract Local Variable	+	ON FILE
<i>Extract Method</i>	alt+shift+m	+	Refactor->Extract Method	+	ON FILE
<i>Generalize Type</i>		-	Refactor->Generalize Type	+	ON FILE
<i>Infer Generic Type Arguments</i>		-	Refactor->Infer Generic Type Arguments	+	ON FILE
<i>InLine</i>	alt+shift+i	+	Refactor->Inline	+	ON FILE
<i>Introduce Factory</i>		-	Refactor->Introduce Factory	+	ON FILE
<i>Introduce Parameter</i>		-	Refactor->Introduce Parameter	+	ON FILE
<i>Move Refactoring</i>	alt+shift+v	+	Refactor->Move	+	Active file and affects other files
<i>Move Member Type to New File</i>		-	Refactor->Move Member Type to New File	+	Active file and affects other files
<i>Pull Up</i>		-	Refactor->Pull Up	+	ON FILE
<i>Push Down</i>		-	Refactor->Push Down	+	ON FILE
<i>Rename</i>	alt+shift+r	+	Refactor->Rename	+	Active file and affects other files
<i>Show Refactor Quick Menu</i>	alt+shift+t	+		-	
<i>Use Supertype where possible</i>		-	Refactor->Use Supertype where possible	+	

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Declaration in Hierarchy</i>		-	Search->Declaration in Hierarchy	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Declaration in Project</i>		-	Search->Declaration in Project	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Declaration in Working Set</i>		-	Search->Declaration in Working Set	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Declaration in Workspace</i>		-	Search->Declaration in Workspace	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>File Search</i>		-	Search->File Search	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Implementors in Project</i>		-	Search->Implementors in Project	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Implementors in Working Set</i>		-	Search->Implementors in Working Set	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Implementors in Workspace</i>	ctrl+h	+	Search->Implementors in Workspace	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Open Search dialog</i>		-	Search->Open Search dialog	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Read Access Hierarchy</i>		-	Search->Read Access Hierarchy	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Read Access in Project</i>		-	Search->Read Access in Project	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Read Access in Working Set</i>		-	Search->Read Access in Working Set	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Read Access in Workspace</i>		-	Search->Read Access in Workspace	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>References in Hierarchy</i>		-	Search->References in Hierarchy	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>References in Project</i>		-	Search->References in Project	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>References in Working Set</i>		-	Search->References in Working Set	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>References in Workspace</i>		-	Search->References in Workspace	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Referring Tests</i>		-	Search->Referring Tests	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Search All occurrences in File Identifier</i>		-	Search->Search All occurrences in File Identifier	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Search All occurrences in Throwing Exception</i>		-	Search->Search All occurrences in Throwing Exception	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Search All occurrences in Implementing Methods</i>	ctrl+shift+u	+	Search->Search All occurrences in Implementing Methods	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Search All occurrences in File Quick Menu</i>		-	Search->Search All occurrences in File Quick Menu	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Write Access in Hierarchy</i>		-	Search->Write Access in Hierarchy	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Write Access in Project</i>		-	Search->Write Access in Project	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Write Access in Working Set</i>		-	Search->Write Access in Working Set	+	ON FILE BUT MIGHT NAVIGATE ELSE
<i>Write Access in Workspace</i>		-	Search->Write Access in Workspace	+	ON FILE BUT MIGHT NAVIGATE ELSE

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Located</b>
<i>Clear Mark</i>		-	NOT CAPTURED
<i>Collapse</i>	ctrl+numpad_substract	+	ON FILE
<i>Copy Lines</i>	ctrl+alt+down	-	NOT CAPTURED
<i>Cut Line</i>		-	NOT CAPTURED
<i>Cut to Beginning of Line</i>		-	NOT CAPTURED
<i>Cut to End of Line</i>		-	NOT CAPTURED
<i>Delete Line</i>	ctrl+D	+	ON FILE
<i>Delete Next</i>	Delete	+	ON FILE
<i>Delete Next Word</i>	ctrl+Delete	+	ON FILE
<i>Delete Previous</i>		-	NOT CAPTURED
<i>Delete Previous Word</i>	ctrl+Backspace	+	ON FILE
<i>Delete to Beginning of Line</i>		-	NOT CAPTURED
<i>Delete to End of Line</i>	ctrl+shift+Delete	+	ON FILE
<i>Duplicate Lines</i>	ctrl+alt+up	+	ON FILE
<i>Expand</i>	ctrl+Numpad_add	+	ON FILE
<i>Expand All</i>	ctrl+numpad_multiply	+	ON FILE
<i>Insert Line Above Current Line</i>	ctrl+shift+enter	+	ON FILE
<i>Insert Line Below Current Line</i>	shift+enter	+	ON FILE
<i>Line Down</i>	Down	+	ON FILE
<i>Line End</i>	End	+	ON FILE
<i>Line Start</i>	Home	+	ON FILE
<i>Line Up</i>	Up	+	ON FILE
<i>Move Lines Down</i>	alt+down	+	ON FILE
<i>Move Lines Up</i>	alt+up	+	ON FILE
<i>Next Column</i>		-	NOT CAPTURED
<i>Next Word</i>	ctrl+right	+	ON FILE
<i>Page Down</i>	Page Down		NOT CAPTURED
<i>Page Up</i>	Page Up		NOT CAPTURED
<i>Previous Column</i>		+	ON FILE
<i>Previous Word</i>	ctrl+left	+	ON FILE
<i>Scroll Line Down</i>	ctrl+Down		NOT CAPTURED
<i>Scroll Line Up</i>	ctrl+up		NOT CAPTURED
<i>Select Line Down</i>	shit+down		NOT CAPTURED
<i>Select Line End</i>	shift+end		NOT CAPTURED
<i>Select Line Start</i>	shift+home		NOT CAPTURED
<i>Select Line Up</i>	shift+up		NOT CAPTURED
<i>Select Next Column</i>			NOT CAPTURED
<i>Select Next Word</i>	ctrl+shift+right	+	ON FILE
<i>Select Page Down</i>	shift+Page Down		NOT CAPTURED
<i>Select Page Up</i>	shift+Page Up		NOT CAPTURED
<i>Select Previous Column</i>			NOT CAPTURED
<i>Select Previous Word</i>	ctrl+shift+left	+	ON FILE
<i>Select Text End</i>	ctrl+shift+End	+	ON FILE
<i>Select Text Start</i>	ctrl+shift+home	+	ON FILE
<i>Select Window End</i>			NOT CAPTURED
<i>Select Window Start</i>			NOT CAPTURED
<i>Set Mark</i>			NOT CAPTURED
<i>Swap Mark</i>			NOT CAPTURED
<i>Text End</i>	ctrl+End	+	ON FILE
<i>Text Start</i>	ctrl+home	+	ON FILE
<i>To Lower Case</i>	ctrl+shift+Y	+	ON FILE
<i>To Upper Case</i>	ctrl+shift+X	+	ON FILE
<i>Toggle Folding</i>	ctrl+numpad_divide	+	ON FILE
<i>Toggle Overwrite</i>	Insert	+	ON FILE
<i>Window End</i>		-	NOT CAPTURED
<i>Window Start</i>		-	NOT CAPTURED

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Ant</i>		-	Window->Show View->Ant	+	ON FILE
<i>Breakpoints</i>	alt+shift+Q,B	+	Window->Show View->Breakpoints	+	ON FILE
<i>Cheat Sheets</i>	alt+shift+Q,H	+	Window->Show View->Other->Cheat Sheets	+	ON FILE
<i>Classic Search</i>		-	Window->Show View-> Other->Basic->Classic Search	+	ON FILE
<i>Console</i>	alt+shift+Q,C	+	Window->Show View->Console	+	ON FILE
<i>CVS Annotate</i>		-	Window->Show View->Other->CVS Annotate	+	ON FILE
<i>CVS Editors</i>		-	Window->Show View->Other->CVS Editors	+	ON FILE
<i>CVS Repositories</i>		-	Window->Show View->Other->CVS Repositories	+	ON FILE
<i>CVS Resource History</i>		-	Window->Show View->Other->CVS Resource History	+	ON FILE
<i>Debug</i>		-	Window->Show View->Debug	+	ON FILE
<i>Display</i>		-	Window->Show View->Display	+	ON FILE
<i>Error Log</i>		-	Window->Show View->Error Log	+	ON FILE
<i>Expressions</i>		-	Window->Show View->Expressions	+	ON FILE
<i>Java Call Hierarchy</i>		-	Window->Show View->Other-> Java-> Call Hierarchy	+	ON FILE
<i>Java Declaration</i>	alt+shift+Q,D	+	Window->Show View->Java Declaration	+	ON FILE
<i>Java Members</i>		-	Window->Show View->Other->Java Browsing->Members	+	ON FILE
<i>Java Package Explorer</i>	alt+shift+Q,P	+	Window->Show View->Package Explorer	+	ON FILE
<i>Java Packages</i>		-	Window->Show View->Other->Java Browsing->Packages	+	ON FILE
<i>Java Projects</i>		-	Window->Show View->Other->Java Browsing->Projects	+	ON FILE
<i>Java Type Hierarchy</i>	alt+shift+Q,T	+	Window->Show View->Type Hierarchy	+	ON FILE
<i>Java Types</i>		-	Window->Show View->Other->Java Browsing->Types	+	ON FILE
<i>Javadoc</i>	alt+shift+Q,J	+	Window->Show View->Javadoc	+	ON FILE
<i>JUnit</i>		-	Window->Show View->Other->Java->JUnit	+	ON FILE
<i>Memory</i>		-	Window->Show View->Other->Debug->Memory	+	ON FILE
<i>Outline</i>	alt+shift+Q,O	+	Window->Show View->Outline	+	ON FILE
<i>Plug-in Dependencies</i>		-	Window->Show View->Other->PDE->Plug-in Dependencies	+	ON FILE
<i>Plug-in Registry</i>		-	Window->Show View->Other->PDE Runtime->Registry	+	ON FILE
<i>Plug-ins</i>		-	Window->Show View->Other->PDE->Plug-ins	+	ON FILE
<i>Problems</i>	alt+shift+Q,X	+	Window->Show View->Problems	+	ON FILE
<i>Registers</i>		-	Window->Show View->Other->Debug->Registers	+	ON FILE
<i>Search</i>	alt+shift+Q,S	+	Window->Show View->Search	+	ON FILE
<i>Synchronize</i>	alt+shift+Q,Y	+	Window->Show View->Other->Team->Synchronize	+	ON FILE
<i>Variables</i>	alt+shift+Q,V	+	Window->Show View->Variables	+	ON FILE

<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Active Editor</i>	F12	+	Window->Navigation->Active Editor	+	ON FILE
<i>Close All Perspectives</i>		-	Window->Close All Perspectives	+	ON FILE
<i>Close Perspective</i>		-	Window->Close Perspective	+	ON FILE
<i>Customize Perspective</i>		-	Window->Customize Perspective	+	ON FILE
<i>Hide Editors</i>		-			
<i>Lock the Toolbars</i>		-			
<i>Maximize Active View or Editor</i>	ctrl+M	+	Window->Navigation->Maximize Active View or Editor	+	ON FILE
<i>Minimize Active View or Editor</i>		-	Window->Navigation->Minimize Active View or Editor	+	ON FILE
<i>New Editor</i>		-	Window->New Editor	+	ON FILE
<i>New Window</i>		-	Window->New Window	+	ON FILE
<i>Next Editor</i>	ctrl+F6	+	Window->Navigation->Next Editor	+	ON FILE
<i>Next Perspective</i>	ctrl+F8	+	Window->Navigation->Next Perspective	+	ON FILE
<i>Next View</i>	ctrl+F7	+	Window->Navigation->Next View	+	ON FILE
<i>Open Editor Drop Down</i>	ctrl+E	+	Window->Navigation->Switch to Editor	+	ON FILE
<i>Pin Editor</i>		-			
<i>Preferences</i>		-	Window->Preferences	+	ON FILE
<i>Previous Editor</i>	ctrl+shift+F6	+	Window->Navigation->Previous Editor	+	ON FILE
<i>Previous Perspective</i>	ctrl+shift+F8	+	Window->Navigation->Previous Perspective	+	ON FILE
<i>Previous View</i>	ctrl+shift+F7	+	Window->Navigation->Previous View	+	ON FILE
<i>Reset Perspective</i>		-	Window->Reset Perspective	+	ON FILE
<i>Save Perspective As</i>		-	Window->Save Perspective As	+	ON FILE
<i>Show Key Assist</i>	ctrl+shift+L	+	Help->Show Key Assist		
<i>Show Ruler Context Menu</i>	ctrl+F10	+			
<i>Show Selected Element Only</i>					
<i>Show System Menu</i>	alt+-	+	Window->Navigation->Show System Menu		
<i>Show View Menu</i>	ctrl+F10	+	Window->Navigation->Show View Menu		
<i>Switch to Editor</i>	ctrl+shift+E	+	Window->Navigation->Switch to Editor		



<b>CMD Name</b>	<b>Key Binding</b>	<b>V</b>	<b>Menu Binding</b>	<b>V</b>	<b>Located</b>
<i>Add Block Comment</i>	ctrl+shift+/ 	+	Source->Add Block Comment	+	ON FILE
<i>Add Constructors from Superclass</i>		-	Source->Add Constructors from Superclass	+	ON FILE
<i>Add Import</i>	ctrl+shift+M	+	Source->Add Import	+	ON FILE
<i>Add Javadoc Comment</i>	alt+shift+J	+	Source->Add Comment	+	ON FILE
<i>Comment</i>		-		-	ON FILE
<i>Externalize Strings</i>		-	Source->Externalize Strings	+	ON FILE
<i>Find Strings to Externalize</i>		-	Source->Find Strings to Externalize	+	ON FILE
<i>Format</i>	ctrl+shift+F	+	Source->Format	+	ON FILE
<i>Format Element</i>		-	Source->Format Element	+	ON FILE
<i>Generate Constructor using Fields</i>		-	Source->Generate Constructor using Fields	+	ON FILE
<i>Generate Delegate Methods</i>		-	Source->Generate Delegate Methods	+	ON FILE
<i>Generate Getters and Setters</i>		-	Source->Generate Getters and Setters	+	ON FILE
<i>Indent Line</i>		-	Source->Correnct Indentation	+	ON FILE
<i>Organize Imports</i>	ctrl+shift+O	+	Source->Organize Imports	+	ON FILE
<i>Override/Implement Methods</i>		-	Source->Override/Implement Methods	+	ON FILE
<i>Quick Assist - Assign parameter to field</i>		-		-	NOT CAPTURED
<i>Quick Assist - Assign to field</i>	ctrl+2,F	-		-	NOT CAPTURED
<i>Quick Assist - Assign to local variable</i>	ctrl+2,L	-		-	NOT CAPTURED
<i>Quick Assist - Rename in file</i>	ctrl+2,R	-		-	NOT CAPTURED
<i>Quick Assist - Replace statement with block</i>		-		-	NOT CAPTURED
<i>Quick Fix - Add cast</i>		-		-	NOT CAPTURED
<i>Quick Fix - Add import</i>		-		-	NOT CAPTURED
<i>Quick Fix - Add non -NLS tag</i>		-		-	NOT CAPTURED
<i>Quick Fix - Add throws declaration</i>		-		-	NOT CAPTURED
<i>Quick Fix Change to static access</i>		-		-	NOT CAPTURED
<i>Quick Fix Quality field access</i>		-		-	NOT CAPTURED
<i>Remove Block Comment</i>	ctrl+shift+/ 	+	Source->Remove Block Comment	+	ON FILE
<i>Remove Occurrence Annotations</i>	alt+shift+U	+		-	ON FILE
<i>Show Source Quick Menu</i>		-		-	NOT CAPTURED
<i>Sort Members</i>		-	Source->Sort Members	+	ON FILE
<i>Surround with try/catch Block</i>		-	Source->Surround with try/catch Block	+	ON FILE
<i>Toggle Comment</i>	ctrl+/  ctrl+7 ctrl+shift+C	+	Source->Toggle Comment	+	ON FILE
<i>Toggle Mark Occurrences</i>	alt+shift+O	+		+	ON FILE
<i>Uncomment</i>		-		-	NOT CAPTURED



## APPENDIX B

# Appendix 2

---

This appendix includes the experiment applied for process mining: the given installation guide, the given guidelines, and the main class for their task and also it includes the diagrams produced by the process mining tool.

## B.1 Experiment

### Instalation guide

This text contains installation instructions. After the installation, start Eclipse and you will see a new view (named Rabbit) under the "Rabbit" category in the show views dialog in Eclipse. A folder also named Rabbit will appear under your home folder, it's used as the XML database, it's important that you don't modify files under this directory.

=== System Requirements: === \* Eclipse 3.4, 3.5, or higher \* JavaSE 1.6 or higher \* Mac, Linux, or Windows

=== Windows, Linux, Mac: === To install, unzip the zip file, then move the two jar files into the \*dropins\* (or \*plugins\*) folder under your Eclipse's folder,

you may create the `*dropins*` folder if it's not already created.

Note: The plug-in will not be loaded if the Eclipse is placed under a system folder, for example, the "Program Files" directory under Windows Vista/7, the simplest way to fix this is to move your Eclipse to another location such as your home user folder, or see the alternative below.

=== Linux Alternative: === If you've installed Eclipse through the command line (such as `"sudo apt-get install eclipse"` on Ubuntu), then you can install the plug-in by moving the two jar files into: `/.eclipse/_-your-current-version-_/dropins` (or `*plugins*`).

The purpose of this short experiment is to gather data with regards to Developers' workflow in the Eclipse IDE. Therefore, please follow the task carefully and develop the required units. It is important to follow the given steps to setup the environment properly. **It is crucial to follow the given naming for your variables methods and classes.**

## Step 1 : Download

- Download the jar files provided on the google drive.
- Follow InstallationGuide.txt
- Create a new workspace inside folder **“Test”**

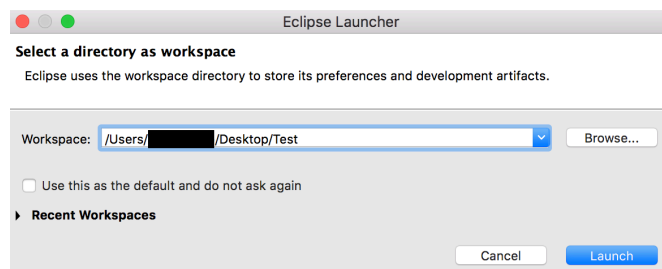


Figure 1: Create Workspace **“Test”**

## Step 2: The main task

At this point you are ready to start. The tool is represented in the Figure 2. You can see information regarding the different activities being recorded if you press button refresh on the tool. **Keep in mind that is highly important to follow the naming notations provided.** To **obtain valid results** is expected that you follow (see Figure 3)

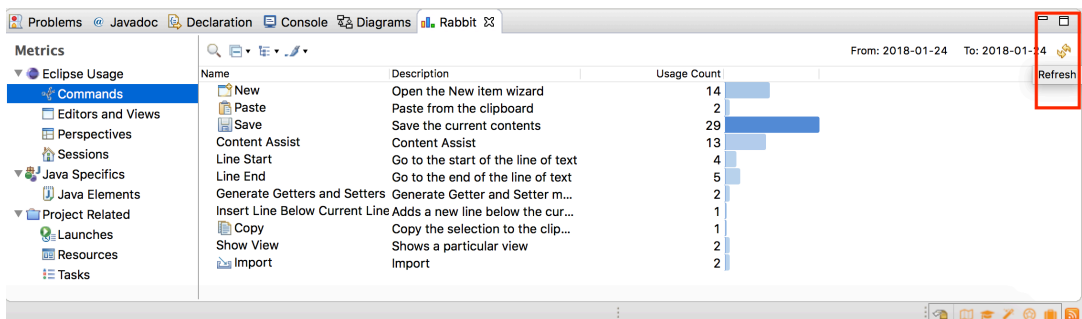


Figure 2: Rabbit Eclipse tool

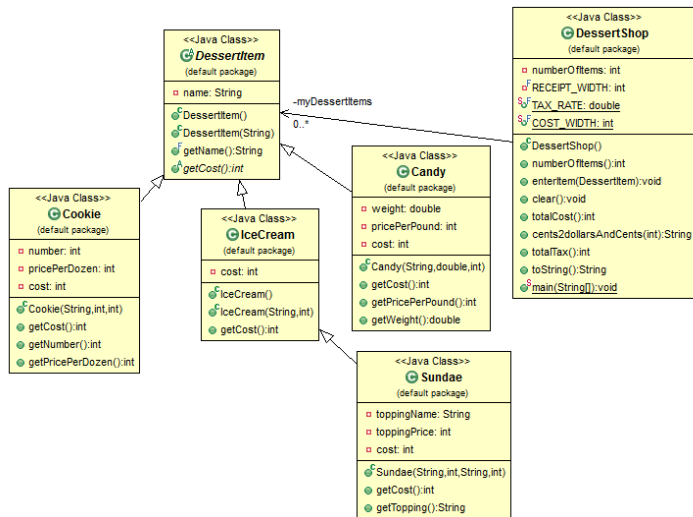


Figure 3: Class Diagram

**Create a project named Test.** To activate and see the results of your programming session Follow the steps in Figure 4.

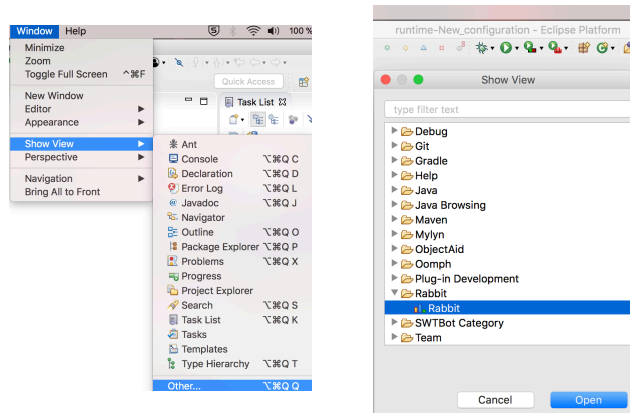


Figure 4: Activate Rabbit

The task is fairly simple, the purpose is the implementation of inheritance hierarchy of classes derive from an abstract superclass.

**DessertShop** class is given to you and contains the main functions of the shop and the test for your classes. Further your task is to implement:

1. **DessertItem** abstract superclass.
2. **Candy, Cookie, IceCream** classes which derive from **DessertItem** superclass
3. **Sundae** class which derives from **IceCream** class

**The Candy class** should be derived from the **DessertItem** class. A Candy item has a weight and a price per pound which are used to determine its cost. The cost should be calculated as  $(\text{cost}) * (\text{price per pound})$ .

**The Cookie class** should be derived from the **DessertItem** class. A Cookie item has a number and a price per dozen which are used to determine its cost. The cost should be calculated as  $(\text{cost}) * (\text{price per dozen})$ .

**The IceCream class** should be derived from the **DessertItem** class. An IceCream item simply has a cost.

**The Sundae class** should be derived from the **IceCream** class. The cost of a Sundae is the cost of the IceCream plus the cost of the topping.

#### Step 4: Extracting Data

Your last step before finishing is to **press the button refresh**.

The Plugin has now recorded your activity. The documents created can be found in Rabbit folders. For both mac and windows users, the data files will be found in `/Users/<UserName>/Rabbit/<Users. <username>. runtime-New_configuration>`.

Mac users might not be able to see these documents because they are considered as hidden files, therefore you'll need to enable hidden files.

Please zip all the files created and send them to [s150954@student.dtu.dk](mailto:s150954@student.dtu.dk)

The main file given to developers was *Dessertshop.java*

```

1
2 public class DessertShop {
3     private DessertItem[] myDessertItems;
4     private int numberOfItems;
5     private final int RECEIPT_WIDTH = 30;
6
7     public DessertShop() {
8         myDessertItems = new DessertItem[100];
9         numberOfItems = 0;
10    }
11
12    public int numberOfItems() {
13        return numberOfItems;
14    }
15
16    public void enterItem(DessertItem item) {
17        this.myDessertItems[numberOfItems] = item;
18        numberOfItems++;
19    }
20
21    public void clear() {
22        for(int i = 0; i < numberOfItems; i++)
23            this.myDessertItems[i] = null;
24        numberOfItems = 0;
25    }
26
27    public int totalCost() {
28        int sum = 0;
29        for(int i = 0; i < numberOfItems; i++)
30            sum += myDessertItems[i].getCost();
31        return sum;
32    }
33
34    public final static double TAX_RATE = 6.5;    // 6.5%
35    public final static int COST_WIDTH = 6;
36
37    public String cents2dollarsAndCents(int cents) {
38        String s = "";
39
40        if (cents < 0) {
41            s += "-";
42            cents *= -1;
43        }
44
45        int dollars = cents / 100;
46        cents = cents % 100;
47
48        if (dollars > 0)
49            s += dollars;
50
51        s += ".";

```



```

52
53     if (cents < 10)
54         s += "0";
55
56     s += cents;
57
58     return s;
59 }
60
61 public int totalTax() {
62     return (int)Math.round(this.totalCost() *TAX_RATE / 100.00);
63 }
64
65 public String toString() {
66     String s = "";    // receipt
67
68
69     s += "      " + "Derlicious!!! " + "\n";
70     s += "      " + "_____ " + "\n";
71
72     for(int j = 0; j < numberOfItems; j++){
73
74         String l = myDessertItems[j].getName();    // items of every
              line
75
76 //         String item = myDessertItems[j].getClass().toString().
              substring(6);    // get the item category
77
78         String p = cents2dollarsAndCents(myDessertItems[j].getCost())
              ;    // price of every item
79         if (p.length() > COST_WIDTH)    // verify the price is within
              length
80             p = p.substring(0, COST_WIDTH);
81
82         if (myDessertItems[j] instanceof IceCream) {    // print if
              ice cream
83             while(l.length() < RECEIPT_WIDTH - p.length()){
84                 l += " ";
85             }
86             s += l + p + "\n";
87         }
88         else if (myDessertItems[j] instanceof Sundae) {    // print if
              Sundae
89
90             s += ((Sundae)myDessertItems[j]).getTopping() + " Sundae
              with\n";
91
92             while(l.length() < RECEIPT_WIDTH - p.length()){
93                 l += " ";
94             }
95             s += l + p + "\n";
96         }
97         else if (myDessertItems[j] instanceof Candy){    // print if
              Candy
98             s += ((Candy) myDessertItems[j]).getWeight() + " lbs @ " +

```

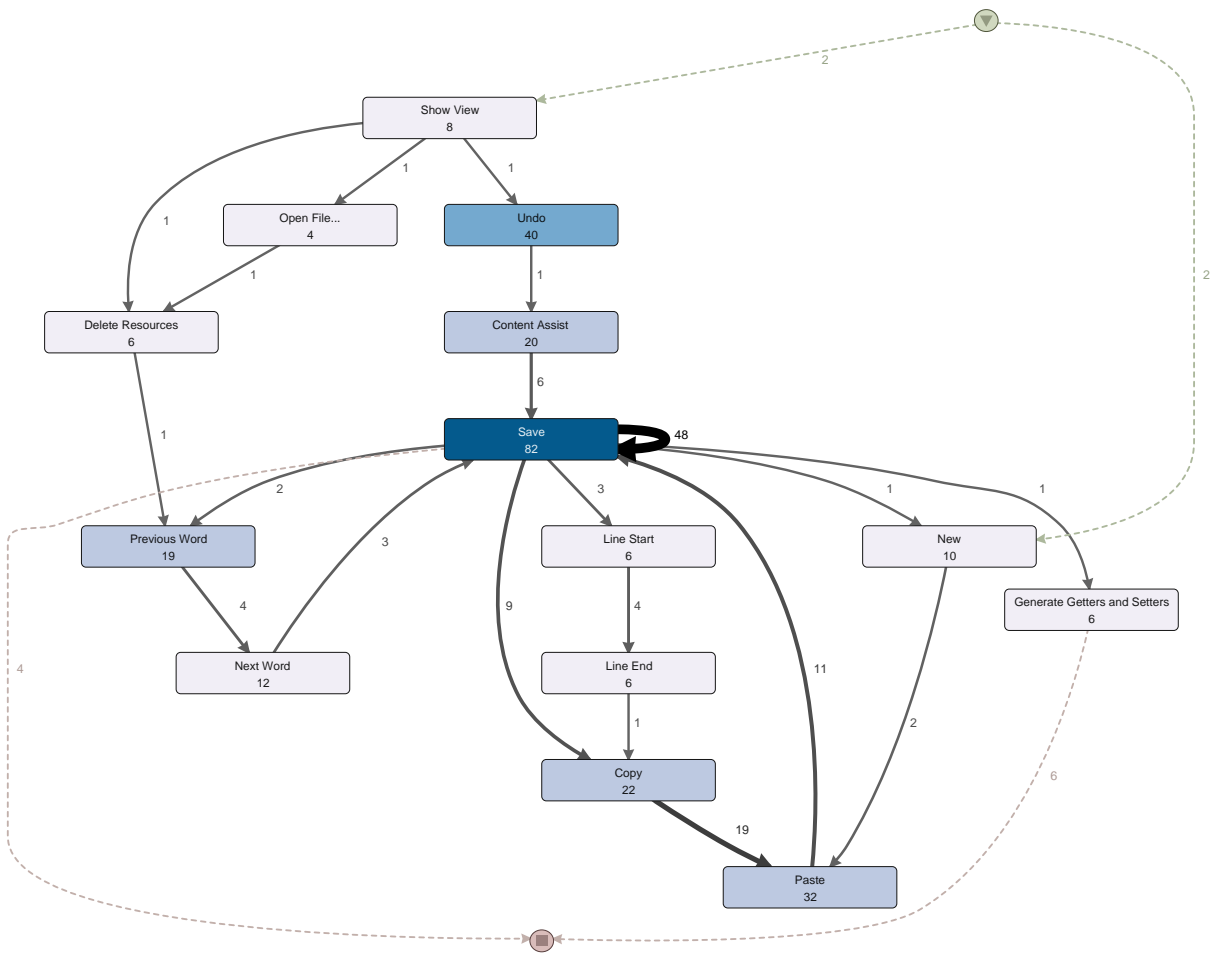
```

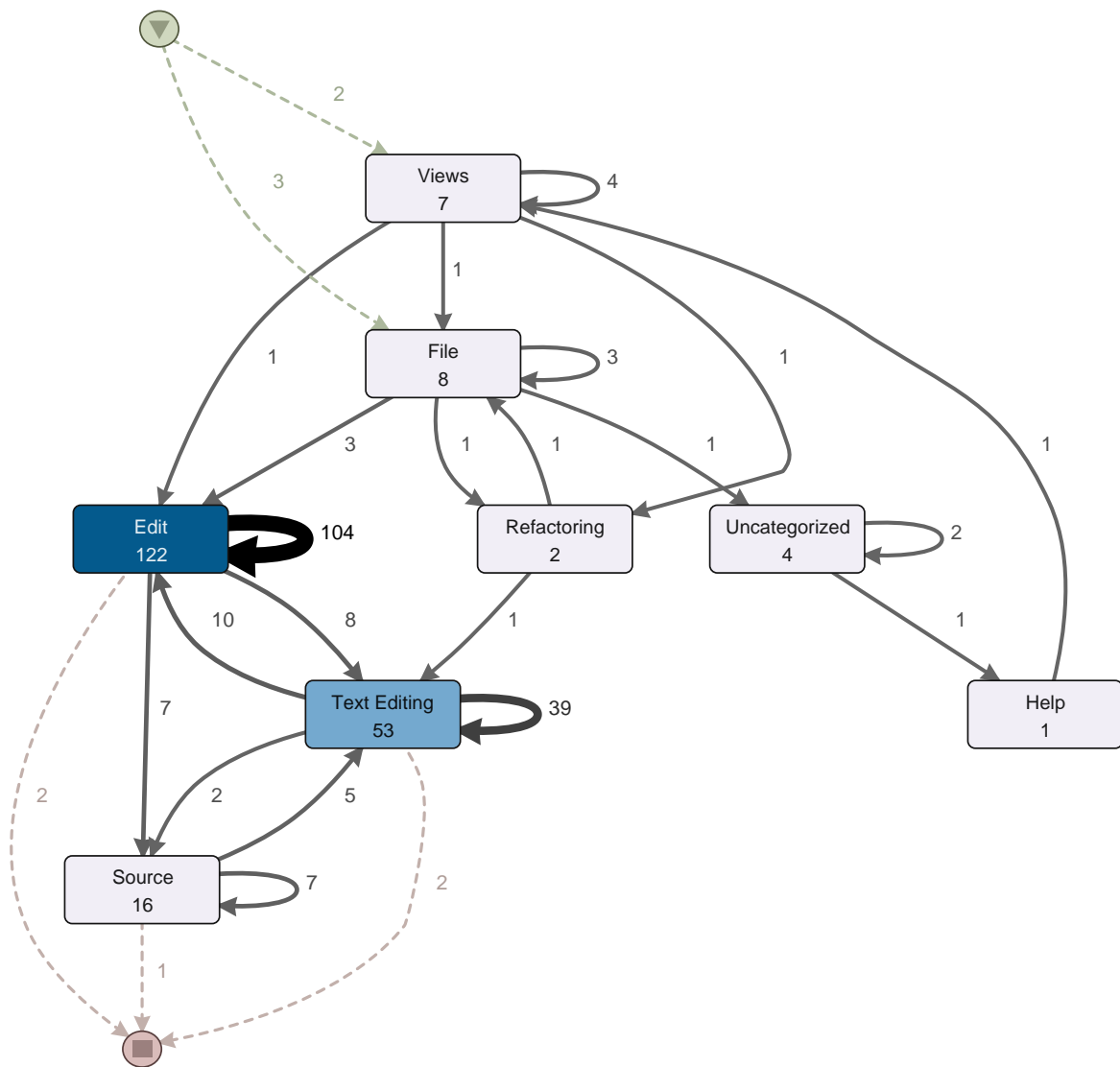
99         cents2dollarsAndCents(((Candy) myDessertItems[j])).
           getPricePerPound()) + " /lb.\n";
100
101         while(l.length() < RECEIPT_WIDTH - p.length()){
102             l += " ";
103         }
104         s += l + p + "\n";
105     }
106     else { // print if Cookie
107         s += ((Cookie)myDessertItems[j]).getNumber() + " @ " +
108             cents2dollarsAndCents(((Cookie)myDessertItems[j])).
109                 getPricePerDozen()) + " /dz\n";
110
111         while(l.length() < RECEIPT_WIDTH - p.length()){
112             l += " ";
113         }
114         s += l + p + "\n";
115     }
116 }
117 String line = "\nTax";
118 String tax = cents2dollarsAndCents(this.totalTax()); // print
           tax
119 while(line.length() <= RECEIPT_WIDTH - tax.length())
120     line += " ";
121 s += line + tax;
122
123 String totalCost = cents2dollarsAndCents(this.totalCost() +
           this.totalTax()); // print total cost
124 line = "\nTotal Cost";
125 while(line.length() <= RECEIPT_WIDTH - totalCost.length())
126     line += " ";
127 s += line + totalCost;
128
129 return s;
130 }
131 public static void main(String[] args) {
132
133     DessertShop checkout = new DessertShop();
134
135     checkout.enterItem(new Candy("Peanut Butter Fudge", 2.25,
136         399));
137     checkout.enterItem(new IceCream("Vanilla Ice Cream",105));
138     checkout.enterItem(new Sundae("Choc. Chip Ice Cream",145, "
139         Hot Fudge", 50));
140     checkout.enterItem(new Cookie("Oatmeal Raisin Cookies", 4,
141         399));
142
143     System.out.println("\nNumber of items: " + checkout.
144         numberOfItems() + "\n");
145     System.out.println("\nTotal cost: " + checkout.totalCost()
146         + "\n");
147     System.out.println("\nTotal tax: " + checkout.totalTax() +
148         "\n");
149     System.out.println("\nCost + Tax: " + (checkout.totalCost()

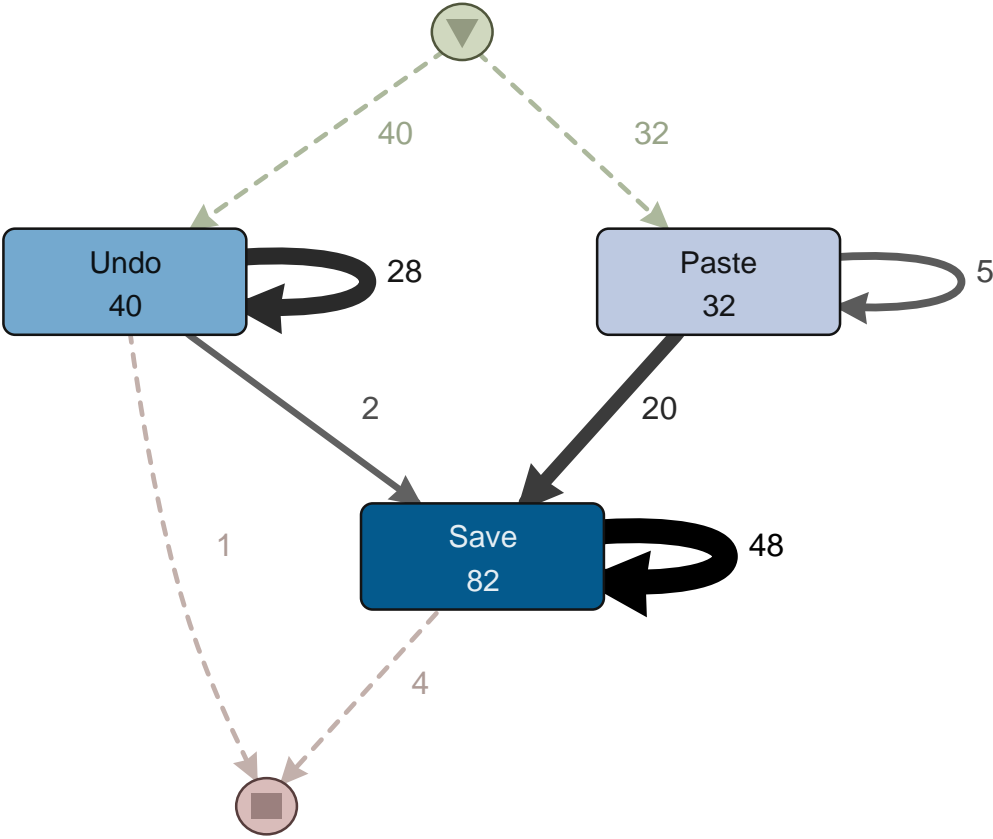
```

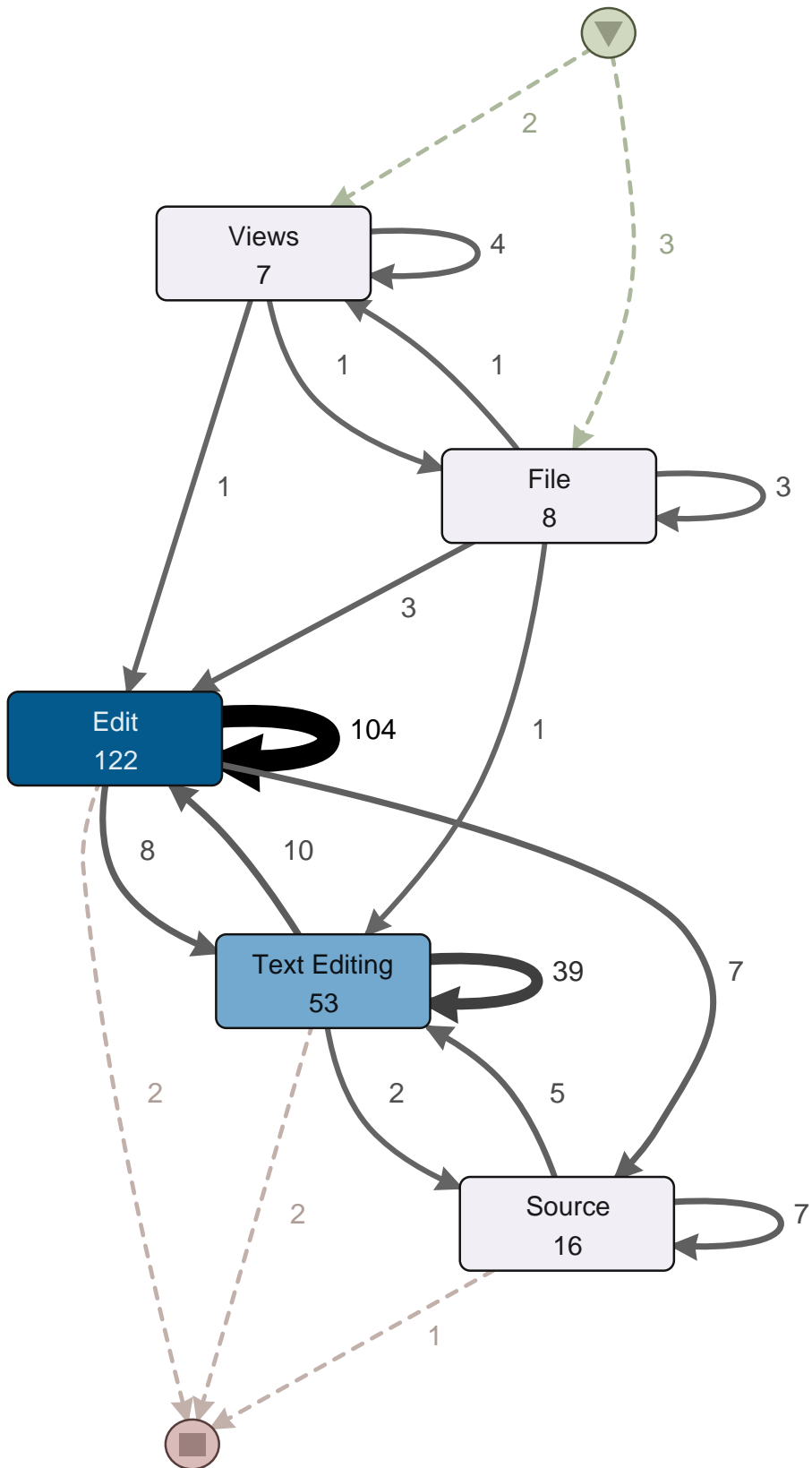
```
144         + checkout.totalTax()) + "\n");
145     System.out.println(checkout);
146
147     checkout.clear();
148
149     checkout.enterItem(new IceCream("Strawberry Ice Cream",145)
150     );
151     checkout.enterItem(new Sundae("Vanilla Ice Cream",105, "
152     Caramel", 50));
153     checkout.enterItem(new Candy("Gummy Worms", 1.33, 89));
154     checkout.enterItem(new Cookie("Chocolate Chip Cookies", 4,
155     399));
156     checkout.enterItem(new Candy("Salt Water Taffy", 1.5, 209))
157     ;
158     checkout.enterItem(new Candy("Candy Corn",3.0, 109));
159
160     System.out.println("\nNumber of items: " + checkout.
161     numberOfItems() + "\n");
162     System.out.println("\nTotal cost: " + checkout.totalCost()
163     + "\n");
164     System.out.println("\nTotal tax: " + checkout.totalTax() +
165     "\n");
166     System.out.println("\nCost + Tax: " + (checkout.totalCost()
167     + checkout.totalTax()) + "\n");
168     System.out.println(checkout);
169 }
170 }
```

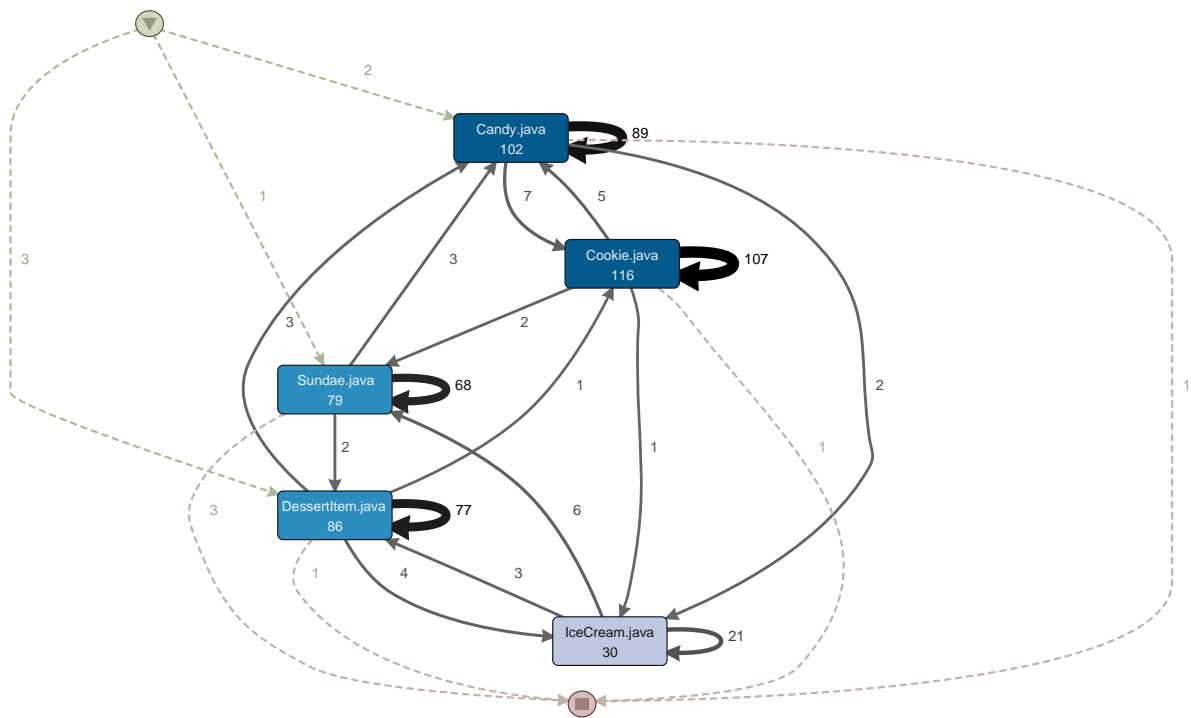
## B.2 Disco Results



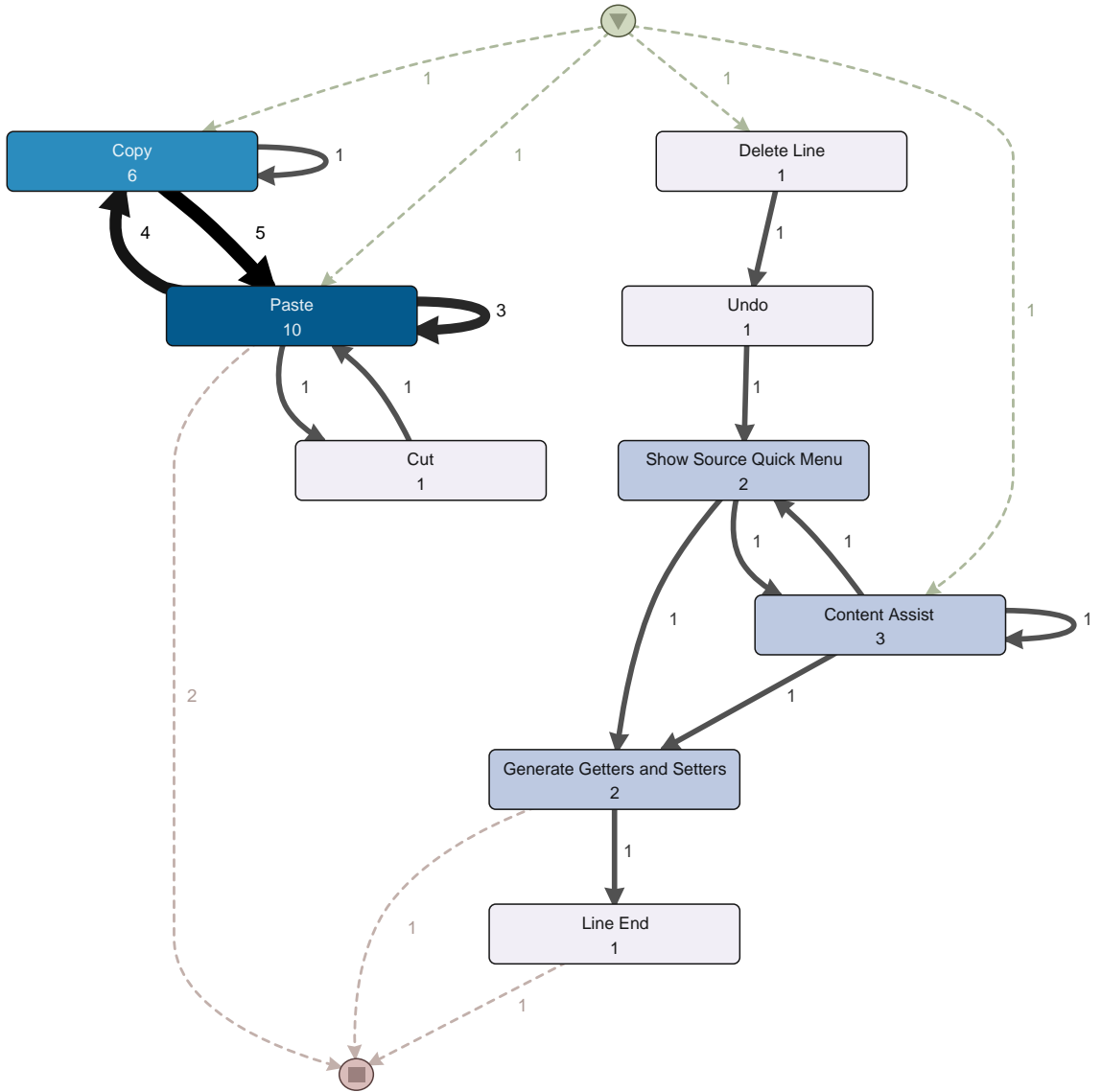


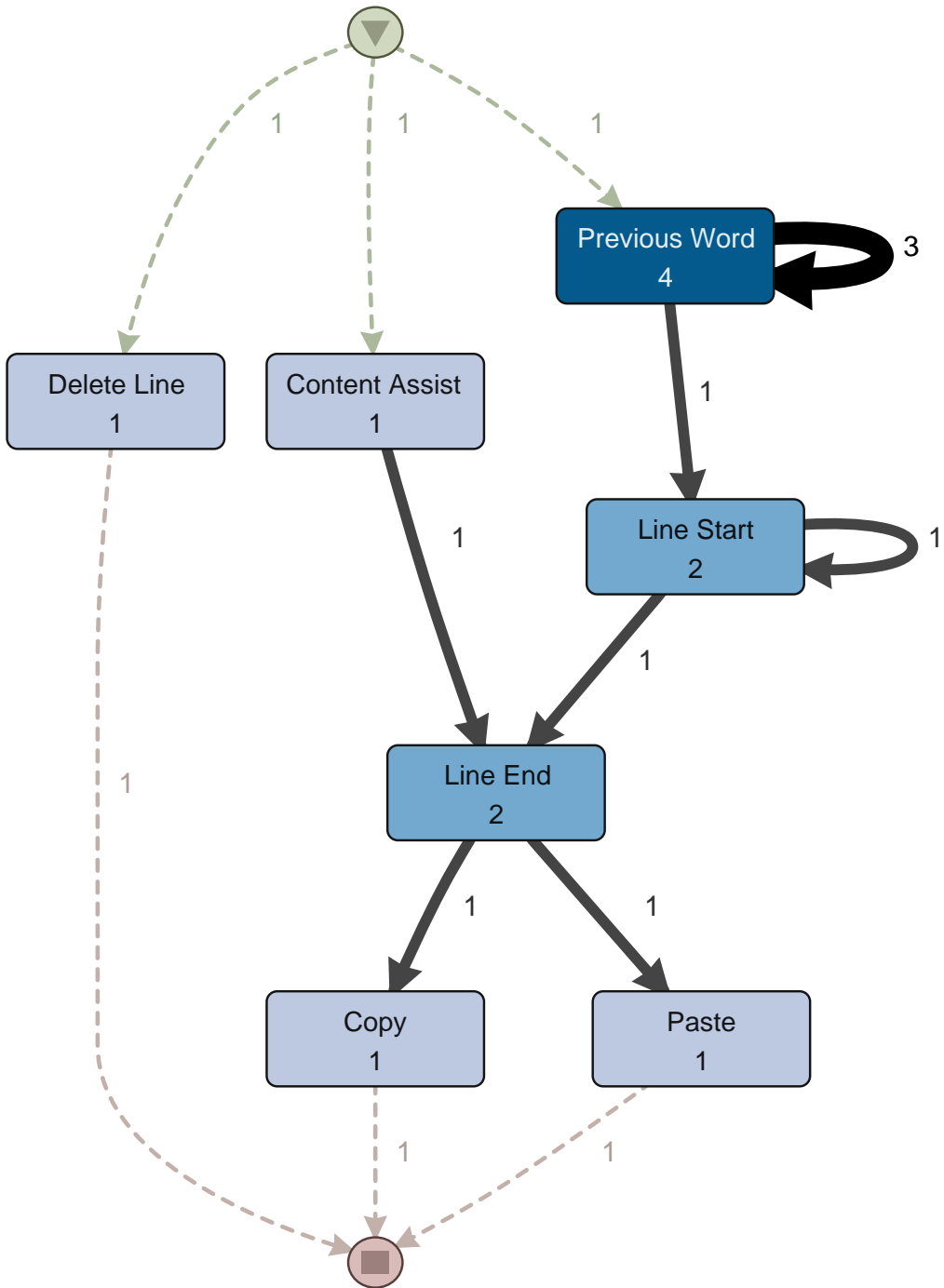


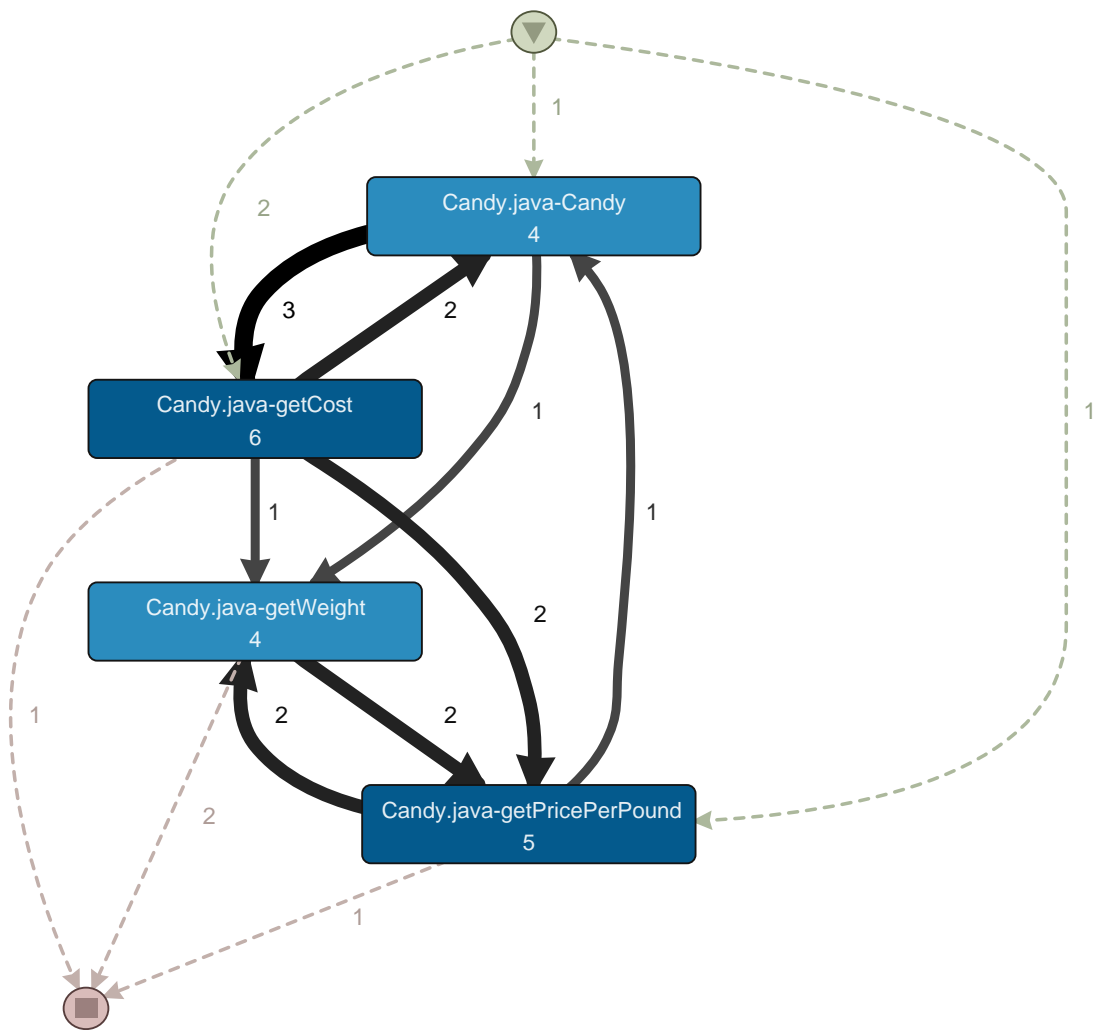


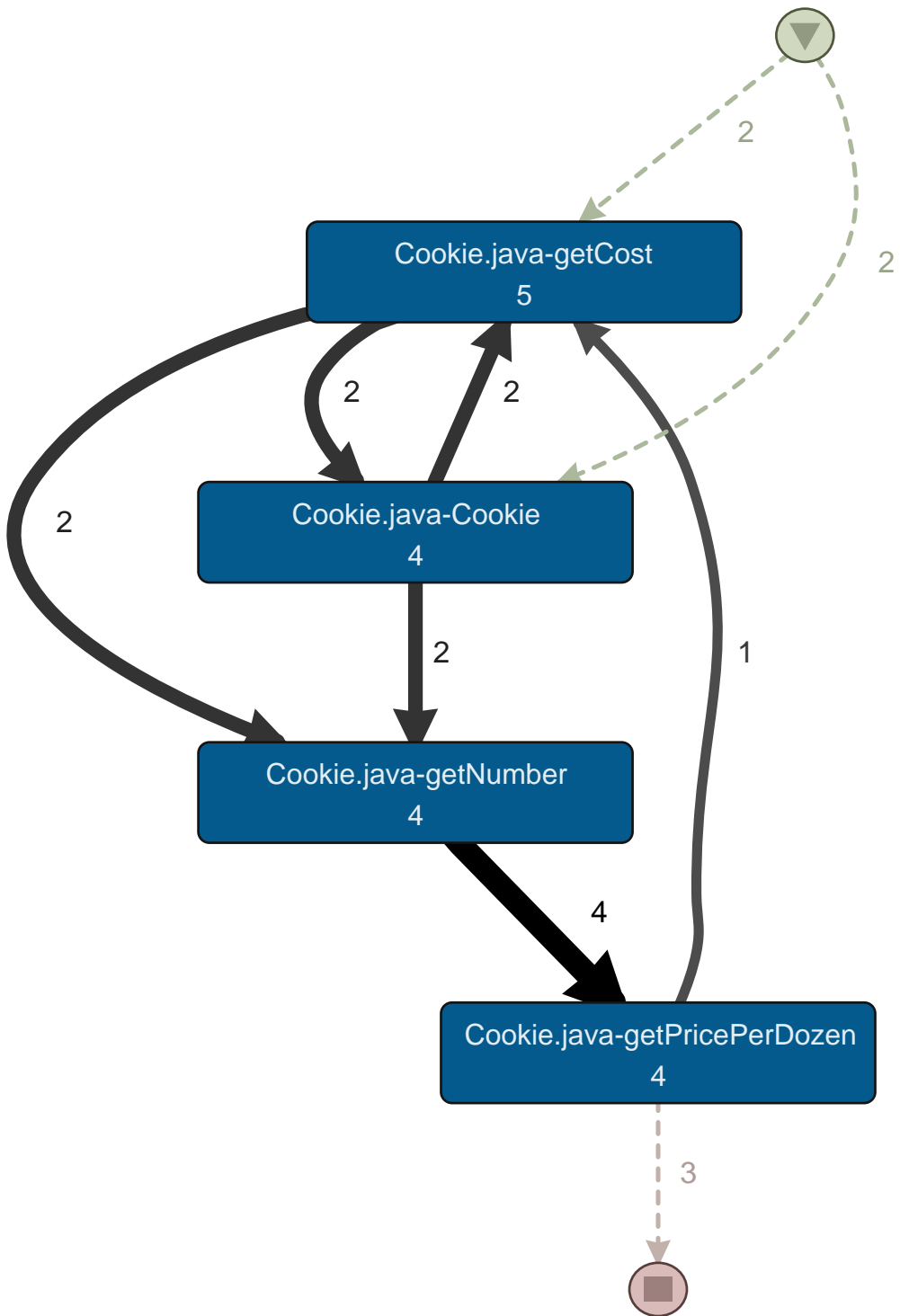


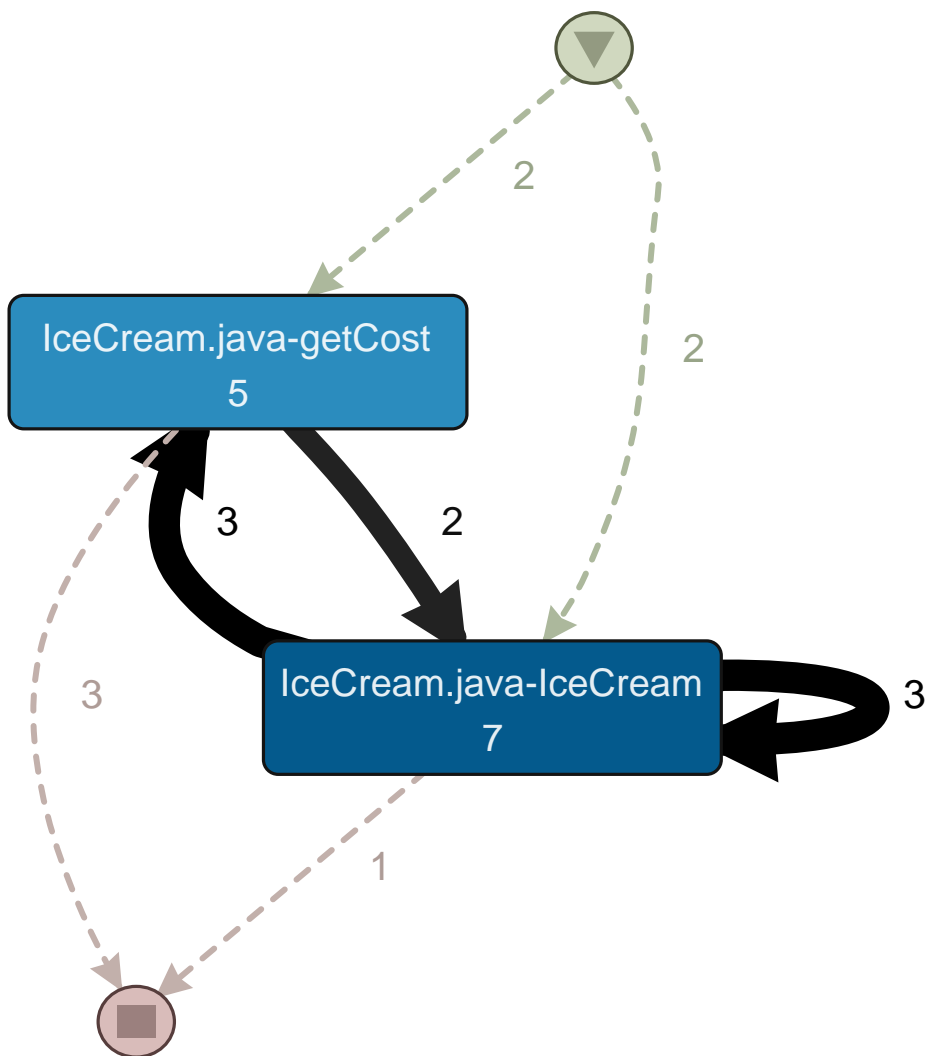


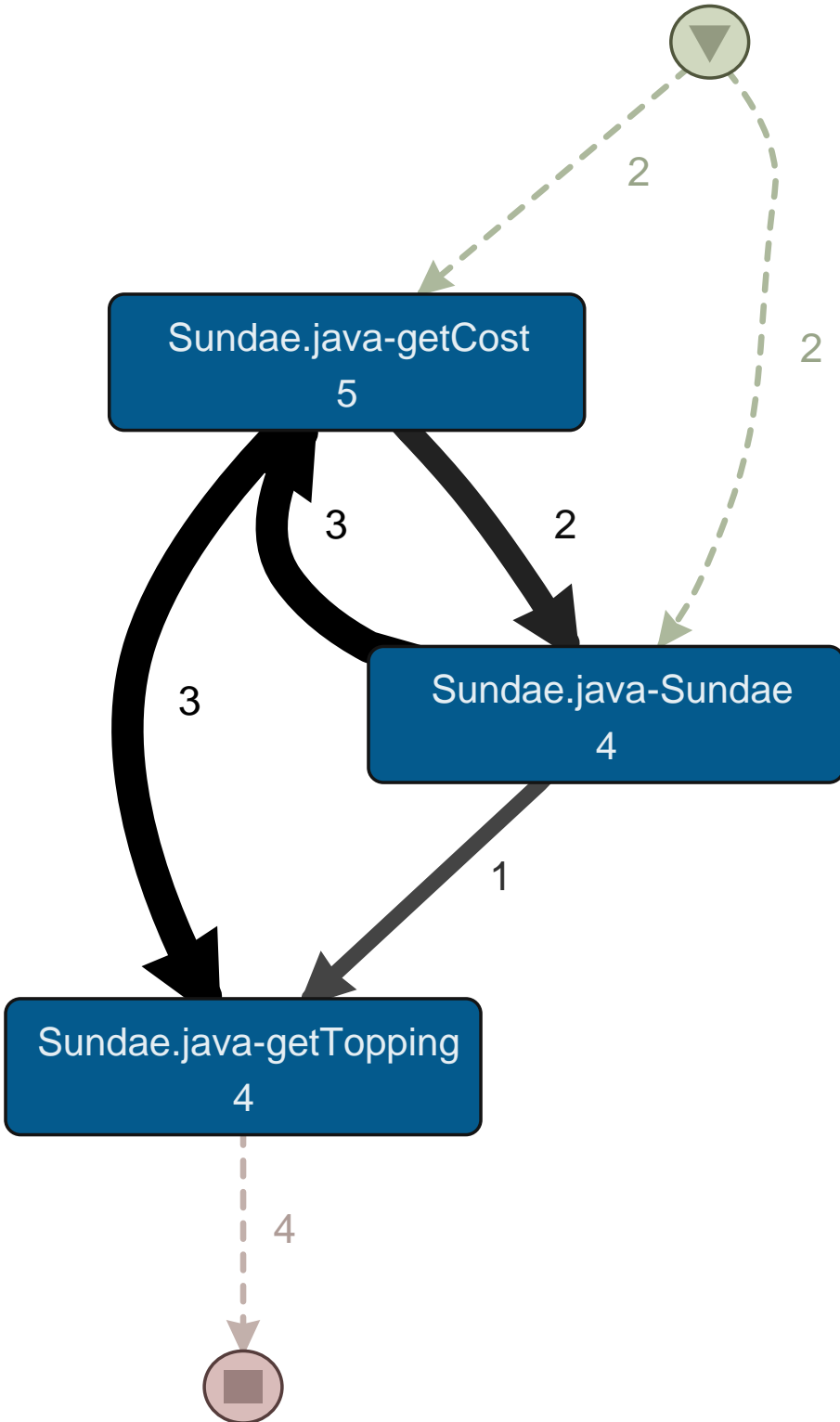












# Bibliography

---

- [2.518] Creative Commons Attribution 2.5. Android Studio. <https://developer.android.com/studio/index.html>, 2018. [Online; accessed March 1, 2018].
- [Bur09] Ed Burnette. *Eclipse IDE Pocket Guide*. O'Reilly Media, Inc., 2009.
- [Car16] Pierre Carbonnelle. Top IDE index is created by analyzing how often ide's are searched on google. <https://pypl.github.io/IDE.html>, 2016. [Online; accessed March 1, 2018].
- [Cod11] Google Code. Rabbit Eclipse Description. <https://code.google.com/archive/p/rabbit-eclipse/>, 2011. [Online; accessed March 1, 2018].
- [Com18] Eclipse Community. Mylyn Description. <https://www.eclipse.org/mylyn/>, 2018. [Online; accessed March 1, 2018].
- [DK03] Yves Pigneur Shusma Patel Dimitri Konstantas, Michel Leonard. *Object-Oriented Information Systems*. Springer, Geneva, Switzerland, 2003.
- [FKA<sup>+</sup>05] James Fogarty, Andrew J. Ko, Htet Htet Aung, Elspeth Golden, Karen P. Tang, and Scott E. Hudson. Examining task engagement in sensor-based statistical models of human interruptibility. *Proceedings of the SIGCHI conference on*

- Human factors in computing systems - CHI '05*, page 331, 2005.
- [Flo16] State Of Flow. Eclipse Metrics. <https://marketplace.eclipse.org/content/eclipse-metrics>, 2016. [Online; accessed March 1, 2018].
- [Fou18a] Eclipse Foundation. Eclipse IDE. <https://www.eclipse.org/>, 2018. [Online; accessed March 1, 2018].
- [Fou18b] Eclipse Foundation. Usage Data Collector. <https://www.eclipse.org/org/usedata/>, 2018. [Online; accessed March 1, 2018].
- [Gen18] LLC Genuitec. Darkest Dark Theme w/DevStyle. <https://marketplace.eclipse.org/content/darkest-dark-theme-wdevstyle>, 2018. [Online; accessed March 1, 2018].
- [Git11] GitHub. Mylyn Repository. <https://github.com/smilebase/org.eclipse.mylyn.github>, 2011. [Online; accessed March 1, 2018].
- [Git15a] GitHub. iTrace Repository. <https://github.com/trshaffer/iTrace/blob/master/README.md>, 2015. [Online; accessed March 1, 2018].
- [Git15b] GitHub. Rabbit Eclipse Repository. <https://github.com/aronlurie/rabbit-eclipse>, 2015. [Online; accessed March 1, 2018].
- [Git17] GitHub. TimeKeeper Repository. <https://github.com/turesheim/eclipse-timekeeper>, 2017. [Online; accessed March 1, 2018].
- [Git18] GitHub. Metrics Repository. <https://github.com/qxo/eclipse-metrics-plugin/tree/master/net.sourceforge.metrics>, 2018. [Online; accessed March 1, 2018].
- [GMR17] Marko Gasparic, Gail C. Murphy, and Francesco Ricci. A context model for IDE-based recommendation systems. *Journal of Systems and Software*, 128:200–219, 2017.
- [Gu12] Zhongxian Gu. Capturing and exploiting fine-grained IDE interactions. *Proceedings - International Conference on Software Engineering*, pages 1630–1631, 2012.



- [jax10] jaxenter. New Time-Tracking Plugin For Eclipse. <https://jaxenter.com/new-time-tracking-plugin-for-eclipse-100491.html>, 2010. [Online; accessed March 1, 2018].
- [KMCA06] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987, 2006.
- [Los17] Andrey Loskutov. Bytecode Outline. <https://marketplace.eclipse.org/content/bytecode-outline>, 2017. [Online; accessed March 1, 2018].
- [LW05] Channing Walton Lance Walton. State of flow, Eclipse Metrics. <http://www.stateofflow.com/projects/16/eclipsemetrics>, 2005. [Online; accessed March 1, 2018].
- [LWH05] Dright Ho Laurie Williams and Sarah Heckman. Metrics Tutorial. <http://realsearchgroup.org/SEMaterials/tutorials/metrics/>, 2005. [Online; accessed March 1, 2018].
- [Mic18] Microsoft. Visual Studio. <https://www.visualstudio.com/>, 2018. [Online; accessed March 1, 2018].
- [MKF06] G.C. Murphy, M. Kersten, and L. Findlater. How are Java software developers using the Elipse IDE? *IEEE Software*, 23(4):76–83, 2006.
- [ML13a] Roberto Minelli and Michele Lanza. Visualizing the workflow of developers. *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013*, pages 2–5, 2013.
- [ML13b] Roberto Minelli and Michele Lanza. Visualizing the workflow of developers. *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013*, pages 2–5, 2013.
- [MML15] Roberto Minelli, Andrea Mocchi, and Michele Lanza. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. *IEEE International Conference on Program Comprehension*, 2015-Augus:25–35, 2015.

- [MMRL16] Roberto Minelli, Andrea Mocci, Romain Robbes, and Michele Lanza. Taming the IDE with fine-grained interaction data. *IEEE International Conference on Program Comprehension*, 2016-July(Dcc):1–10, 2016.
- [oDC18] Trustees of Dartmouth College. BASIC Begins at Dartmouth. <http://www.dartmouth.edu/basicfifty/basic.html>, 2018. [Online; accessed March 1, 2018].
- [PR12] Chris Parnin and Spencer Rugaber. Programmer information needs after memory failure. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 123–132, 2012.
- [pWT14] profitbricks William Toll. Top 48 Integrated Developer Environments (IDEs) & Code Editors. <https://blog.profitbricks.com/top-integrated-developer-environments-ides/>, 2014. [Online; accessed March 1, 2018].
- [Rei18] Steve Reiss. Code Bubbles. <http://cs.brown.edu/~spr/codebubbles/>, 2018. [Online; accessed March 1, 2018].
- [RMLvdA14] Vladimir A. Rubin, Alexey A. Mitsyuk, Irina A. Lomazova, and Wil M. P. van der Aalst. Process mining can be applied to software too! *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, pages 1–8, 2014.
- [RND09] David Rathlisberger, Oscar Nierstrasz, and Stephane Ducasse. Autumn leaves: Curing the window plague in IDEs. *Proceedings - Working Conference on Reverse Engineering, WCRE*, pages 237–246, 2009.
- [Rud15] Julia Rudnitskaia. *Process Mining - Data Science in Action*. 2015.
- [Say14] O. K. Sayiam, R., Sahingoz. A process mining approach in Software Development and Testing Process: A case study. *World Congress on Engineering and Computer Science*, 1:407–411, 2014.
- [SMHF<sup>+</sup>15] Will Snipes, Emerson Murphy-Hill, Thomas Fritz, Mohsen Vakilian, Kostadin Damevski, Anil R. Nair, and David Shepherd. A Practical Guide to Analyzing IDE Usage Data. *The Art and Science of Analyzing Software Data*, pages 85–138, 2015.

- [SS03] Scott Fairbrother Dan Kehn John Kellerman Pat McCarthy Sherry Shavor, Jim D’Anjou. *The Java Developer’s Guide to Eclipse*. Addison - Wesley, 2003.
- [SWW<sup>+</sup>15] Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, Michael Falcone, and Bonita Sharif. iTrace: enabling eye tracking on software artifacts within the IDE to support software engineering tasks. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, pages 954–957, 2015.
- [Tra17] Vantage One Sam Travarca. Reasons for an Integrated Development Environment. <http://vantageonesoftware.com/advantages-disadvantages-integrated-development-environment/>, 2017. [Online; accessed March 1, 2018].
- [Uni15] Youngstown State University. iTrace – Tracking Eye Movements on Software Artifacts in the Eclipse IDE. <http://seres1.csis.ysu.edu/iTrace/>, 2015. [Online; accessed March 1, 2018].
- [USI14] Lugano. USI. DFlow. <http://dflow.inf.usi.ch/>, 2014. [Online; accessed March 1, 2018].
- [vdAETUETN16] Wil van der Aalst Eindhoven Technical University Eindhoven The Netherlands. *Process Mining*. Springer, Berlin, Heidelberg, 2016.
- [vDAV<sup>+</sup>05] B. van Dongen, a K Alves de Medeiros, H M W Verbeek, a J M M Weijters, and Will van der Aalst. The ProM framework: A new era in process mining tool support. *Application and Theory of Petri Nets 2005*, (3536):444–454, 2005.
- [Ver17] Veracode. APPSEC knowledge base. <https://www.veracode.com/security/integrated-development-environments>, 2017. [Online; accessed March 1, 2018].
- [Wik17] Wikipedia. Delphi (IDE). [https://en.wikipedia.org/wiki/Delphi\\_\(IDE\)](https://en.wikipedia.org/wiki/Delphi_(IDE)), 2017. [Online; accessed March 1, 2018].
- [Wik18] Wikipedia. Eclipse (IDE). [https://en.wikipedia.org/wiki/Eclipse\\_\(software\)](https://en.wikipedia.org/wiki/Eclipse_(software)), 2018. [Online; accessed March 1, 2018].
- [YM11] YoungSeok Yoon and Brad a. Myers. Capturing and analyzing low-level events from the code editor. *Proceedings of the*

*3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools - PLATEAU '11*, page 25, 2011.

[Zer18] ZeroTurnaround. PDE. <https://zeroturnaround.com/software/jrebel/>, 2018. [Online; accessed March 1, 2018].

[ZH13] Iyad Zayour and Hassan Hajjdiab. How much integrated development environments (IDEs) improve productivity? *Journal of Software*, 8(10):2425–2431, 2013.