

Bachelor Thesis

January 13, 2013

Happy Coder

André Meyer

of Uster, Switzerland (09-736-398)

supervised by

Prof. Dr. Thomas Fritz



University of
Zurich^{UZH}



Bachelor Thesis

Happy Coder

André Meyer



University of
Zurich^{UZH}



Bachelor Thesis

Author: André Meyer, info@andre-meyer.ch

URL: www.andre-meyer.ch/happycoder

Project period: 16.07.2012 - 16.01.2013

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to thank all the people who were involved in this thesis. First of all, I thank Prof. Dr. Thomas Fritz for giving me the opportunity to write this thesis at the software evolution and architecture lab and for his excellent support. Moreover, I want to thank the other members of the lab for their valuable comments and ideas. Further appreciation goes to the developers of the tools and frameworks I have used for the creation of Happy Coder including Reverb, Rabbit, Mylyn, Wamon, jQuery mobile, canvasXpress, and a lot more. Many thanks also go to all the participants of the studies. I also would like to express many thanks to my parents, my girlfriend and my friends who gave me insights and comments from other non IT-related perspectives. Moreover, many thanks to my father, Dr. Peter Meyer, who supported me with his knowledge and insights in the writing phase of my project. Finally, I want to thank Dr. Kaspar Zimmermann (Head of Operations, Global Discovery Chemistry, Novartis) and Paul Meyer (emer. research scientist, Lawrence Livermore National Laboratory) for proof-reading my work.

Abstract

Recent advances in small inexpensive sensors, cheaper storage, and low-power processing cause an increasing popularity of trackers that quantify a user's activities throughout his everyday life. The Fitbit and the Nike+ Fuelband are two examples of commercial approaches that motivate a user to be more active by tracking his activity and visualizing the analyzed data. In the area of software engineering there are similar tools to support a developer in a single domain of his work, such as planning tools or bug repositories. Only little research has been performed on how to integrate the available data and how to focus on providing a retrospection of a developer's work day.

In order to contribute to overcome this shortcoming we introduce a tool, Happy Coder that provides developers with a retrospective analysis of their work day, by tracking predefined metrics and visualizing them on a web client. This includes a front-end with consolidated data analysis, visualizations and representations of the collected data. Two studies revealed that developers assess their productivity based on a personal evaluation of their work day. This assessment is dominated by personal preferences of different metrics like work items, meetings, web searches or activities on the computer.

Zusammenfassung

Fortschritte in der Herstellung von kleinen, leistungsstarken Sensoren und günstige Speichermöglichkeiten erlauben eine stetige Verbreitung von Loggern, die den Alltag eines Users quantifizieren. Fitbit und das Nike+ Fuelband sind zwei Beispiele solcher kommerziellen Ansätze, die den Benutzer fördern aktiver zu sein, indem sie seine Aktivität aufzeichnen und visualisieren. Ähnliche Ansätze sind auch im Softwareengineering zu finden. Diese Tools unterstützen einen Programmierer in einem einzelnen Bereich seiner Arbeit; beispielsweise in der Planung oder beim Organisieren von Bug-Reports. Allerdings gibt es bislang noch wenige Erkenntnisse darüber, wie die verfügbaren Daten vereint und integriert werden können und einen Rückblick auf den vergangenen Arbeitstag erlauben.

Um einen Beitrag zur Lösung dieses Problems zu erbringen, haben wir Happy Coder entwickelt, das Programmierern eine retrospektive Analyse ihres Arbeitstages ermöglicht, indem es vordefinierte Metriken misst und in einem Web-Client darstellt. Dieser beinhaltet eine konsolidierte Datenanalyse sowie Visualisierungen und Repräsentationen der gesammelten Daten. Zwei Studien haben gezeigt, dass Entwickler ihre Produktivität aufgrund einer persönlichen Evaluierung des vergangenen Arbeitstages machen. Diese Evaluierung basiert auf persönlichen Vorlieben der verschiedenen Metriken wie zum Beispiel Aufgaben (Work Items), Meetings, Recherchen im Internet oder Aktivitäten am Computer.

Contents

1	Introduction	1
2	Related Work	3
2.1	Activity Sensing	3
2.2	Developer Motivation	4
2.3	Developer Productivity	5
3	Retrospective Developer Analysis	7
3.1	Tracked Developer Metrics	8
3.1.1	Measuring Activities	8
3.1.2	Measuring work with Java Elements	9
3.1.3	Measuring work with Work Items	9
3.1.4	Measuring Meetings	10
3.1.5	Measuring Web Searches	10
3.1.6	Measuring Change Sets	11
3.2	Visualization of the Metrics	11
3.3	Future Metrics	14
4	Prototype	15
4.1	Architecture Overview	15
4.2	Developer Monitoring	17
4.2.1	Activities Tracker	19
4.2.2	Java Elements Tracker	19
4.2.3	Work Items Tracker	20
4.2.4	Meetings Tracker	20
4.2.5	Web Searches Tracker	21
4.2.6	Change Sets Tracker	21
4.3	Server	22
4.3.1	Database	22
4.3.2	Data Upload Service	22
4.3.3	Data Web Service	24
4.4	Web Client	26
5	Usage Study	29
5.0.1	Results	29
5.0.2	Limitations and Conclusions	30

6	Metrics and Productivity Studies	31
6.1	Productivity Study	31
6.1.1	Results	31
6.1.2	Limitations and Conclusions	32
6.2	Metrics and Productivity Study	33
6.2.1	Results	33
6.2.2	Limitations and Conclusions	37
7	Discussion	39
7.1	Discussion	39
7.2	Future Work	41
8	Conclusions	43
A	Tools and Environments	45
B	Visualization of the Metrics in the Web Client	47
C	Usage Study: Results	55
D	Productivity Study: Results	59
E	Metrics and Productivity Study: Questionnaire	61
F	Metrics and Productivity Study: Results	65
G	Contents of the CD-ROM	69

List of Figures

3.1	Visualization of multiple metrics with a Chernoff faces.	12
3.2	Visualization of the most important metrics on the overview page of the web client.	13
4.1	Architecture of Happy Coder.	16
4.2	Status view of the Happy Coder Eclipse plug-in.	17
4.3	Preferences view of the Happy Coder Eclipse plug-in.	18
4.4	Entity Relationship Model of the database.	23
4.5	UML activity diagram of the data upload to the server's database.	25
4.6	Overview page of the Happy Coder web client.	27
7.1	Ask the user for his mood after his work day.	42
B.1	Web Client Activities visualizations.	48
B.2	Web Client Java Elements visualizations.	49
B.3	Web Client Work Items visualizations.	50
B.4	Web Client Meetings visualizations.	51
B.5	Web Client Web Searches visualizations.	52
B.6	Web Client excerpt of the log-in and the about page.	53

List of Tables

2.1	Comparison of commercial and noncommercial, wearable activity sensors.	4
5.1	Results of the usage study where the Happy Coder prototype was evaluated.	29
6.1	Results for the question if the developer's last work day was productive.	32
6.2	Categorized answers to the open question (Question 2) of how developers assess productivity.	32
6.3	Results for metrics in the category Activities.	33
6.4	Results for metrics in the category Work Items.	34
6.5	Results for metrics in the category Java Elements.	34
6.6	Results for metrics in the category Change Sets.	35
6.7	Results for metrics in the category Web Searches.	35
6.8	Results for metrics in the category Meetings.	35
6.9	Categorized answers to the open question of how developers assess productivity.	36
A.1	Tools and frameworks used to monitor developers.	45
A.2	Tools used to run the server.	46
A.3	Frameworks used to present and visualize the data on the web client.	46

List of Listings

4.1	Java snippet showing the composition of the url on the example of the activity tracker.	24
4.2	PHP snippet showing how the submission type and file is obtained.	24

Introduction

‘When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.’
- Lord Kelvin

With increasing sensing capabilities, cheaper storage space and better capacities to analyze the collected data, more and more people use trackers to quantify parts of their lives. There are several commercial approaches to track the activity of a person’s life, such as Fitbit¹, Nike+ Fuelband², or Zeo³. The Fitbit and Nike+ Fuelband track the number of steps a user⁴ walks during a day and quantify useful information such as calories burned, the number of stairways climbed or scoring points. Zeo is a sleep tracker which wakes the user in a lighter phase of sleep. Common to all of these tools is that they measure everyday data, analyze it and try to give the user a better insight in his activities with the help of visualizations. These insights often motivate a user to adapt his habits. For example, Fritz et al. [FM12] found that Fitbit users “described using a device for the awareness and reflection of their activity and for providing a continuous motivational factor to be more active”.

In the area of software engineering there are numerous products and services available that assist a developer in his daily work. For example, there are tools to plan a developer’s work and track his work tasks (e.g. Mylyn⁵ or Outlook⁶), to track the Java elements a developer worked on (e.g. Rabbit⁷), and many more. While all of these approaches support a developer in a single domain of his work, only little research has been performed on how to integrate the available data and how to focus on providing a retrospection of a developer’s work day. However, it is difficult for a developer to objectively appreciate his own work progress. This assessment is necessary to motivate him for the coming day. No approaches are known which combine multiple aspects of a developer’s achievements, focussing on providing a review of his work day with meaningful visualizations.

¹<http://www.fitbit.com>, verified 01/10/13

²<http://www.nike.com/fuelband>, verified 01/10/13

³<http://www.myzeo.com>, verified 01/10/13

⁴The term developer and user can describe a female or male person. For simplicity reasons the male form was used throughout this thesis.

⁵<http://www.eclipse.org/mylyn>, verified 01/10/13

⁶<http://microsoft.com/outlook>, verified 01/10/13

⁷<http://code.google.com/p/rabbit-eclipse>, verified 01/10/13

Considering the just described problem, we formulated the following research question to summarize the goal of this thesis:

RQ: Can we provide a retrospective analysis of a developer's work that integrates useful metrics of his work day to provide a meaningful analysis of his productivity?

In this paper, we introduce a tool, called Happy Coder that allows a software developer to quickly get a summarized overview of his daily work and achievements. One part of this thesis work was to examine the feasibility of collecting data on various aspects of a developer's work, to visualize them within the prototype and evaluate it.

To this end, we developed an approach comprised of:

- A monitoring component to collect relevant data of a developer's work. For better extensibility, we defined an abstraction and modularized the implementation where it was possible.
- A data storage component to make the data available to the developer. This is implemented using a web server which aggregates and prepares the data and a database storing it. Furthermore, necessary gateways to access the data by a client are provided.
- Finally, a component to provide a front-end to visualize the collected data to the developer. Amongst others, we visualized the work items a developer worked on, the meetings a developer attended and the Java elements a developer touched. Moreover, the developer can compare the current day to previous work days to evaluate trends. The information is made available for multiple devices for convenient access from everywhere with the use of a multi-platform website.

In a second step, we carried out two studies with multiple developers to evaluate the metrics that are useful to a developer and to find out how he assesses his productivity. In a future study we want to check whether or not the use of Happy Coder assists the developer to increase the awareness of his daily activity and helps him to reflect the work day. Moreover, we want to investigate if our prototype leads to a higher motivation of the developer towards achieving personal as well as software project goals.

This thesis makes two contributions:

- it introduces a tool, Happy Coder that provides developers a retrospective analysis of their work day, by tracking predefined metrics and visualizing them on a web client, and
- it presents the results of two studies suggesting that important metrics and the assessment of productivity is based on a personal evaluation of the developer's work day.

This thesis is structured as follows: In Chapter 2 we present related work in the fields of activity sensing, developer motivation and developer productivity. Chapter 3 focuses on the metrics that are being tracked and visualized. In Chapter 4, the architecture and implementation of the three components of the prototype are explained. After the implementation of the prototype a usage study of Happy Coder was conducted which is described in Chapter 5. The metrics that are being tracked and approaches to measuring productivity are evaluated in Chapter 6. The implementation of Happy Coder and possible future work is discussed in Chapter 7. Finally, Chapter 8 concludes the results of this thesis.

In the following thesis, we refer to the developer monitoring trackers as *tracker*, *monitor* or *recorder* (e.g. activities recorder).

Related Work

This chapter presents related work in the field of activity sensing, developer motivation and developer productivity.

2.1 Activity Sensing

Recent advances in small inexpensive sensors, cheaper storage, and low-power processing cause an increasing popularity of trackers that quantify a user's activities throughout his everyday life [CMT⁺08]. The first quantified self conference being held in 2012¹ and new commercial products like Nike+ Fuelband or Fitbit One released in 2012 suggest a common trend towards quantifying our lives.

A number of studies have been conducted to find the effects and benefits of activity sensing tools like sports trackers. Delajoux et al. created a social computer game, Fish'n'Steps, to promote an increase in physical activity [LML⁺06]. The player's daily foot steps are counted and animated with a virtual character, a fish in a fish tank. By adding the fishes of other players, an environment of cooperation and competition encouraged the players to exercise more. The visualization, which was considered much friendlier than an Excel sheet, 'led the users establish new routines that led to healthier patterns of physical activity in their daily lives' [LML⁺06]. Chick Clique, developed by An et al. [TFAG06], motivated teenage girls to exercise more 'by exploiting their social desire to stay connected with their peers'. Moreover, it gave useful tips for healthy nutrition and exercise. UbiFit Garden is another example of on-body sensing. It consists of a wearable sensor, a glanceable display and an interactive application on the mobile phone providing detailed information about the user's activity and allows the user to add, edit or delete activities [CMT⁺08]. The possibility to edit the measured data is a feature that was only found in the UbiFit Garden. Striiv is another pedometer that, unlike to other commercial trackers such as Fitbit and the Nike+ Fuelband, motivates users to do exercise more by providing mini-games powered by walking and donating a little fee for every step made to a charity organization.

While the above-mentioned tools automatically measure steps as the main metric and visualize them in a purposeful manner, they follow different approaches of motivating the user. A comparison of the most important features of those trackers can be found in Table 2.1. The reason why people utilize these devices are multifaceted: some want to achieve some personal goals, better understand themselves, reflect their activity, or have other social reasons [FM12]. On the other hand, there are also many users who track an aspect of their life with an unknown goal. 'They

¹<http://quantifiedself.com>, verified 01/10/13

continue because they believe their numbers hold secrets that they can't afford to ignore.' [Wol12] This clearly emphasizes the necessity of useful metrics in combination with meaningful visualizations and simplifications. For example, the creators of Fitbit tried to abstract the collected data into a visualization to show recent activity at a glance: the user can see his progress on the device's display with a flower that grows leaves. A similar approach can be found on the Fuelband where Nike abstracted away from the step count and introduced the proprietary 'Nike Fuel' measure to indicate the progress towards a predefined goal.

	visualization of measured data	group (sharing, co-operation, etc.)	game character	goal setting and progress towards goal
Chick Clique	✓	✓(measure own group with other groups)	X	✓
Fish'n'Steps	✓	✓(see other users in fish tank)	X	✓
UbiFit Garden	✓	?	X	X
Fitbit (com.)	✓	✓(social networks, high score list)	✓(achievements)	✓
Nike+ Fuelband (com.)	✓	✓(social networks)	✓(achievements)	✓
Striiv (com.)	✓	✓	✓(mini-games, donations to charity)	✓

Table 2.1: Comparison of commercial and noncommercial, wearable activity sensors.

In the area of software engineering there are also some tools available to assist a developer in his daily work. This include tools such as Mylyn, Reverb, Rabbit or Wamon, all covered and discussed in the following chapters of this thesis.

2.2 Developer Motivation

The tools described in the last chapter can motivate a user to do more of a certain activity (e.g. walking, programming) and to achieve goals. In software engineering, motivation is considered as a crucial factor of the overall success of a software project [SSP11, HSB⁺08]. Developers find themselves in a difficult situation: they have to cope with difficult projects, a lot of pressure to perform, quality inspections, fast moving technologies, and much more. Hence, it is very important for a developer to be highly motivated to deliver high quality work [HSB⁺08]. Motivating developers is considered as one of the most important and most difficult skill managers have to master [Whi97]. Whitaker, a software project manager for more than two decades, found a successful way to motivate developers in different teams. He prioritized product features and must-have features and rewarded selected developers privately as they achieved a certain milestone. As soon as the team achieved the next milestone, several other developers received similar awards, including a developer that already had received one.

Besides rewards there are many other motivational factors like challenging and creative work, working in a great team, producing good and useful software, or doing work that is appreciated

by others [SSP10, SSP11, HSB⁺08]. Factors such as repetitive, uninteresting or tedious work, uncertainty, and missing the goals are considered as motivation killers. Moreover, disruptions prevent a developer to focus on a task and to perform the planned work [KHB06]. A retrospective analysis tool that automatically collects data relevant to a developer can improve his motivation by presenting everything he achieved during work. It might also give him insights into unproductive times, like browsing the web or writing emails. This information might help a developer to boost his efficiency and focus more on the work itself.

2.3 Developer Productivity

In order to find a useful approach to provide a meaningful analysis of a developer's work day, the common understanding of productivity in software engineering had to be determined. Researchers have been searching for approaches to measure productivity for almost thirty years². As productivity mainly is a personal assessment of the tasks a developer achieved during a period, it is no wonder that *the* way to quantify productivity has yet to be found. In general, productivity is considered as the produced output divided by the resources consumed [Car06]. As it is difficult to quantify input and output, there are several approaches available to measure productivity of a developer producing code directly. A simple method is for example to measure the lines of code a developer writes during a day. However, this is not very accurate as the amount of code needed to solve a problem differs up to 500% between different programming languages [Jon94]. Moreover, physical lines of code do not correctly represent the effort devoted to a program as there are comments, blank lines, and other constructs which add no functionality to the code. The function-point metric is another possibility of measuring developer's productivity. This metric is more accurate as it does not depend on different programming languages. However, it only measures the code a developer produced and does not consider other aspects of the code, like functionality, complexity, maintainability, size and quality [Jon94, AL03, Dun88]. Moreover, personal aspects such as the developer's motivation, the level of arousal [KHB06] or the number of disruptions are also not measured by this approach. The key finding from a study by Hale et al. [PSHH04] was to take the code as well and the personal aspects into account to measure productivity. Additionally, they found out that teams are more productive when their members work independently and get disturbed rarely. A new approach was tried by Cui et al. [LXC12] to measure a developer's performance with the keystroke as an indicator. They learnt that programmers with a high performance usually have a high keystroke productivity. In addition, developers are more productive under pressure. This is really interesting as productivity also heavily depends on a developer's own feelings and moods [Car06]. A study by Brinkman et al. [KHB06] has shown that 86% of the developers believe that their productivity is related to their mood and motivation.

The examples and their justifications above show that no single productivity measure can be found that will work under all circumstances. This insight is also verified by other studies like the one by Card [Car06] about the challenge of productivity measurements. Hence, it is important to define various metrics and combine them appropriately. Moreover, different aspects of a developer's work day have to be considered as a developer is not always working in the code, but also writing documentations or attending meetings. This thesis presents one possible approach which takes the code, a developer's meetings, and other metrics into account. The metrics tracked by our prototype are described in Chapter 3 and the implementation of the prototype in Chapter 4.

²We found a study from 1984 on developer productivity by Lampert [Lam84].

Retrospective Developer Analysis

Most of the approaches described in the chapter about the related work are activity sensing techniques for a user's private life. Although there are tools available to measure a developer's work day, they mostly concentrate on a single aspect of his work or do not provide meaningful information to the individual developer. In this chapter, we present the metrics currently being tracked by Happy Coder, discuss their visualization and present other metrics that could be tracked in the future.

We introduce the metrics with the help of a motivating example describing an imaginary work day of a software developer. Consider Alan, an experienced software engineer with more than fifteen years of experience. He arrives at work at half past eight in the morning and writes some emails after having read the latest tech headlines in the web. He plans his work day before he returns to fix a bug he could not resolve yesterday. After one hour, the bug is fixed and he commits his work to the source code repository. He then spends the next three hours with the implementation of a new feature on the project he is currently working on. He meets his boss at one o'clock for a short talk to prepare the meeting that follows afterwards. Skipping lunch, Alan discusses a new feature request with a client. After a successful meeting he works on three other bugs that had been assigned to him recently. Unfortunately, he runs into troubles fixing one of them and spends all the time searching for the erroneous code and a way to resolve the issue. He uses the documentation as well as a couple of resources in the internet to search for hints. At five o'clock, Alan has a headache and decides to leave early; he will not be able to resolve the bug today anyway. He cannot concentrate anymore and is frustrated. Alan has the feeling of not having achieved anything today and that time literally ran away. On his way home he takes his smartphone out of his pocket and navigates to the Happy Coder web client. The metrics that have been tracked the whole day in the background are now visualized. They are described in the following chapters. Screenshots of all the visualizations of Alan's work day can be found in Appendix B. After looking at the details of his work day, Alan feels a lot better as he realizes that he has achieved a lot indeed. Being relieved, he can now enjoy his leisure time and recharge his batteries for tomorrow's work.

3.1 Tracked Developer Metrics

‘Software Metrics provide a measurement for the software and the process of software production.’ [HWY09]. They provide quantitative attributes which have an effect in the understanding, forecasting, evaluation and control of a developer’s accomplishments. This section is an overview of the metrics measured by the prototype. In addition, the importance of those metrics to a developer is discussed. It is up to future studies to determine whether the chosen metrics provide useful information to a developer.

To find useful metrics we read the related work on software metrics and included our own ideas and experience. In addition, we also determined whether measuring a certain metric is feasible or not. The metrics were then categorized into useful and intuitive groups: activities, Java elements, meetings, work items, web searches and change sets. Each of the groups contains metrics that aggregate similar information and represent different parts of a developer’s work. This includes coding, searching for information in the web or attending meetings. In Chapter 3.3 we described other possible metrics that could be measured in a future implementation.

3.1.1 Measuring Activities

In contrast to what is often assumed, a developer’s work day does not only consist of writing code. In fact, there are a lot of other activities a developer works on, such as planning, introducing the project to a new employee or presenting a milestone to a customer. In our motivating example, Alan worked for eight hours on his computer and spent most of the time in the IDE (about two thirds of it in the debug mode), two and a half hours in the browser and reading documentation and some time on planning and writing emails. Only very little of his time was used with planning and emails.

It is important to a developer to know how much time he spent for coding and to know what else he did. Hence, we have introduced an activity metric which measures how much time a developer spends in which program on his computer. As we did not want to provide every single detail but a readable and meaningful abstraction of a developer’s work day, certain activities are grouped together into the following sub categories:

- **Coding** (code): This is the time a developer spent in an IDE minus the time he spent debugging.
- **Debugging** (debug): The time a developer spent in the debug-mode. In the current implementation of the prototype for Eclipse, the display time of the debug-perspective is measured.
- **Browsing** (research): All browsers (e.g. Internet Explorer, Google Chrome, Mozilla Firefox) as well as reading PDF’s or editing a Microsoft Word or Excel document fall into that category. With the term ‘research’ not exactly ‘scientific research’ is meant but browsing the web to search for solutions or to read some documentation.
- **Planning** (plan): Coordinating with others (e.g. team members or customers) through Microsoft Outlook, communicating through Skype or Microsoft Lync or taking notes in Microsoft OneNote or Evernote fall into the category of planning.

- **Navigating** (nav): Every time a user is searching a document, managing his document with the Windows Explorer or using other similar file managers, the duration is counted in this category.
- **Other activities** (other): All programs that are currently unassigned or do not fit in one of the categories above are listed here.

As this categorization does probably not fit in all situations, the categorization could be further split into subcategories, like documenting, testing or communicating. Moreover, it might make sense to let the developer categorize his programs on his own.

3.1.2 Measuring work with Java Elements

While a developer is programming in a software project, he usually selects and edits many different elements in the code. A typical Java project in Eclipse is organized into packages containing classes and methods. The Java elements metric does not only measure the time a developer spent writing code, but also how long he was reading code. This information can be useful to a developer to see all the different Java projects he worked on and also the other elements he touched during his work. When Alan browses to the Happy Coder web client, he sees that he had only worked in one single project but on multiple packages during the day. Moreover, he had viewed a lot of code whose code ownership was not his and wasted a lot of time in understanding this code, as it is not properly documented.

Additionally, one could also track other metrics of a developer within the IDE, such as the number of debug sessions a developer starts, the number of different files he touches, how many copy and pastes he did and the number of breakpoints he set.

3.1.3 Measuring work with Work Items

Work items such as tasks, bugs or feature requests are important measures for a developer to plan his work and to see if he achieved his duties and responsibilities (on time). Besides writing a list to record all the tasks, there are professional tools to do so. Some of them (e.g. Bugzilla or Mantis) have specialized to track bugs and feature requests while others (e.g. Microsoft Outlook) provide a more common way to manage tasks. The Happy Coder prototype focuses on the work items a developer has to handle while he is programming. Currently, the work items are tracked through the Mylyn plug-in for Eclipse. In this pre-installed plug-in a user can add a task repository like Bugzilla, Jira, Microsoft TFS, Mantis or Hudson/Jenkins. The task repository can be shared and updated by other team members or users. The developer can add his own tasks, bugs or open issues and plan their execution. When a developer activates a work item Mylyn automatically maintains a task context by monitoring the interaction. Mylyn also tracks the time a user spent working on a certain work item. The work items a developer worked on, their details, current status and the time he spent on solving them is extracted from the collected data.

In the example of Alan, the web client lists all work items Alan worked on today together with the just explained details. He now recalls that he had actually successfully fixed an old bug and implemented a new feature. In addition, he had created a new feature request (work item) after discussing it with the customer in the meeting. Alan now also realizes that he had only spent about one third of his time trying to fix the bug he could not resolve and did not waste as much time as he feared.

In future versions it could also be interesting to see the number of work items a developer created, edited, viewed, deleted and solved. Moreover, a measurement of the complexity of the work item could be included. This additional information could provide the developer with better insights into his work. Other code unrelated task management systems like Microsoft Outlook or Google Tasks¹ could be added to capture tasks usually not stored in a repository like Bugzilla. This could be achievements like a presentation of a finished project in front of a customer, a conference call with a supervisor or the introduction into the project to a new employee.

3.1.4 Measuring Meetings

Besides coding and other duties, a developer has also to attend some meetings. Hence, the time a developer spends in meetings as well as the number of meetings he attends are tracked. To that end, we connect to a developer's calendar to get his events. After applying the following two heuristics, all appointments are filtered and only meetings are left to be stored:

- If the event title contains a certain keyword such as 'meeting', 'gathering', 'conference', 'discussion', 'talk', it is treated as a meeting.
- If an event has one or more invitees it is considered a meeting. That could for example be an informal lunch with a customer.

Alain also navigates to the meetings page in the Happy Coder web client. The visualizations reveal that he had spent one and a half hours in meetings, which is about the average time he usually spends in meetings each day.

3.1.5 Measuring Web Searches

'Software developers use information resources on the web to help perform many different programming tasks, including learning new programming concepts, reminding themselves of syntactic programming language details and clarifying error messages, amongst others.' [SMJ12] As those web searches are a relevant part of the developer's daily work on the computer, the time he spends searching the web (i.e. the time spent in the web browser) is tracked with the activity metric described in Section 3.1.1. Furthermore, it might also be interesting to see whether the web searches were relevant or irrelevant to the code the developer worked on. The Reverb plug-in (see 4.2.5 and [SMJ12]) connects the code with the web pages a user visited by recommending previously visited web pages. The algorithm determines whether a web site was relevant to the code or the Javadoc or not. The search history of Mozilla Firefox² and Google Chrome³ gets indexed, the content of each website downloaded and the text searched for matching certain heuristics. Those heuristics are patterns that are written in camel case or contain an underscore. Moreover, if the pattern resembles a method declaration or an invocation with parameters, it is considered as relevant to the code. If at least two of those patterns are found on a web site it is classified to be code related. Reverb uses those matching patterns to recommend useful web pages a user might revisit. However, Happy Coder only needs the list of all code relevant and irrelevant web searches.

Alan gets useful insights into his productivity, by looking at the information in the web searches section on the web client. Of the eight hours Alan worked on his computer, he spent about two

¹<https://mail.google.com/tasks>, verified 01/10/13

²<http://www.mozilla.org>, verified 01/10/13

³<https://www.google.com/chrome>, verified 01/10/13

and a half hours in the browser. The visualizations reveal an unexpected fact: most of his web searches had been irrelevant to the code he was working on. This comes as a surprise to Alan as he thought he had spent his time efficiently searching for solutions to the bug. He recalls that he had not always immediately found what he was searching for and that he actually had spent some time on news web sites, seen a couple of movies on YouTube and updated his Twitter status a couple of times. He decides to carefully check his browsing behavior - probably he loses more time there than he thought.

3.1.6 Measuring Change Sets

A change set is a container which stores everything related to a check-in operation in a version-control system. This can be file or directory revisions, check-in notes, links to related work items or the change owner details. It is a powerful instrument to a developer to have different versions of the code, for example to find a bug that occurred after recent code changes. Moreover, it helps a developer to stay aware of what other team members are currently working on. The current implementation of Happy Coder does *not* track any change sets, but we plan to implement this metric in the near future. We intend to measure the code churn of the commits and the number of commits a user made. Moreover, it might be interesting for a developer to see the classes, methods and variables he edited, removed or added. Petre et al. [SSP11] found out that a developer's motivation rises when he successfully solves a problem or assists someone in solving his problem. Developers like to mark a work item as *resolved* and to commit the changes to the source code repository. This 'happiness' or on the contrary 'anger' could probably be determined by reading the commit messages. Comments like ':)', 'yeah', 'I had problems' or 'shit' might reveal information of a developer's mood and arousal towards the committed work to draw conclusions of his work performance.

Assuming the feature is implemented, Alan would see a couple of commits which show him the bug he successfully resolved in the morning and the new feature he added. As he could not resolve the second bug in the afternoon, there is no change set available for this changes. Furthermore, he might also see what his team mates achieved in the meantime to be aware of their work.

3.2 Visualization of the Metrics

Software visualizations are a great help as they can sum up large quantities of data with simple and meaningful ways. Our goals for the visualization of the collected data are twofold. First, the essential information has to be summarized in an overview to be visible at first sight. Second, a retrospection of a developer's past work day is needed to give him the possibility to recapitulate his productivity and achievements.

There are some approaches to summarize the user's activity with one single metric, as previously described in the related work section (see Chapter 2.1). In the case of Fitbit this was the growing flower and Nike+ Fuelband has its 'Nike Fuel' which visualizes the progress. We tried to find a similar abstraction of a developer's work and recognized that it is not possible to summarize all data with a single numeric value in a useful way. This finding was also supported by the related work on developer productivity (see Chapter 2.3) and the productivity studies we conducted (see Chapter 6). We decided to visualize multiple metrics and found a way to visualize them with one single representation. One way to achieve this visualization is the use of Chernoff faces, invented by Herman Chernoff in 1973 [SMJ12]. Chernoff faces represent multivariate data

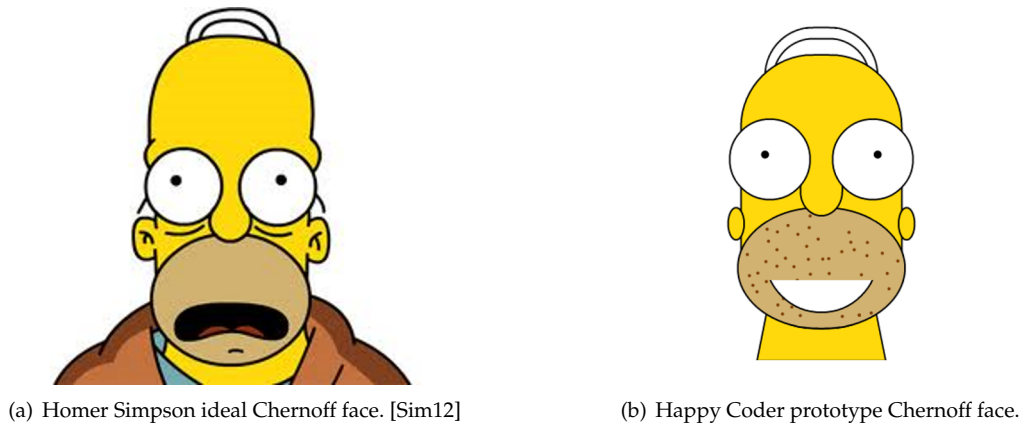


Figure 3.1: Visualization of multiple metrics with a Chernoff faces.

in a manner that is easily perceptible by the human viewer. The individual parts of a face, such as the mouth, nose, ears, eyebrows and eyes, represent values of the variables by their shape, size, placement and orientation. In the original version, they consisted of two-dimensional line drawings and could represent up to eighteen distinct facial parameters. According to Ebert et al. [MER00], humans have the ability to easily recognize faces and small changes in facial characteristics. Therefore, Chernoff faces were considered as the ideal solution to the poly-metric approach. A fun face to improve a developer's attitude towards his past work day that always smiles and probably makes the developer happier was needed. This could probably be achieved with the design of a face which resembles Homer Simpson. An example of the original Homer Simpson face can be found in Figure 3.1(a)⁴. Figure 3.1(b) shows a prototype of the visualization of three metrics in a Happy Coder Chernoff face. The mouth (which is always laughing) remains closed if the developer had no meetings, i.e. if he did not talk a lot. The more time a developer spends in meetings, the wider the mouth opens. According to a common saying 'to poke one's nose into something', the number of different projects a developer worked on is represented with the length of Homer's nose. As Homer's beard grows during the day, the face grows more stubbles the longer the developer worked. In the future, it is intended to include more metrics whose meanings can be assigned to a part in Homer's face.

Besides this poly-metric approach another overview visualization was implemented to show the most important metrics on the web client's front page (see Figure 3.2). The overview consists of a stacked line graph which shows how much time a developer spent on which activity on his computer. For the time a developer worked in the IDE (to either debug or code), the visualization is expanded to show the work items a developer worked on and the Java elements he selected or edited. Moreover, the visualization displays the meetings a developer attended. By looking at the Chernoff Happy Coder face and the stacked line graph, the developer gets an initial impression of his work day and might now be interested in some details. Multiple visualizations on the sub-pages reveal more information on the developer's activities, meetings, work items, Java elements or web searches. Every page visualizes the information of today and the last seven days from one category (e.g. web searches). As trends are usually more important than absolute numbers, the difference to the average is presented next to each metric. Additionally, there are some visualiza-

⁴Hint: In this example, Homer is not smiling but looks a bit puzzled. As previously mentioned, it is intended to let the Happy Coder Chernoff face always smile to cheer the developer up. As this image shows an ideal sector of the front of Homer's face, this image was chosen.

tions of the collected data and descriptions to explain the metrics. More information of the web client is described in Chapter 4.4. Screenshots of the different pages as well as a short description are attached in Appendix B.

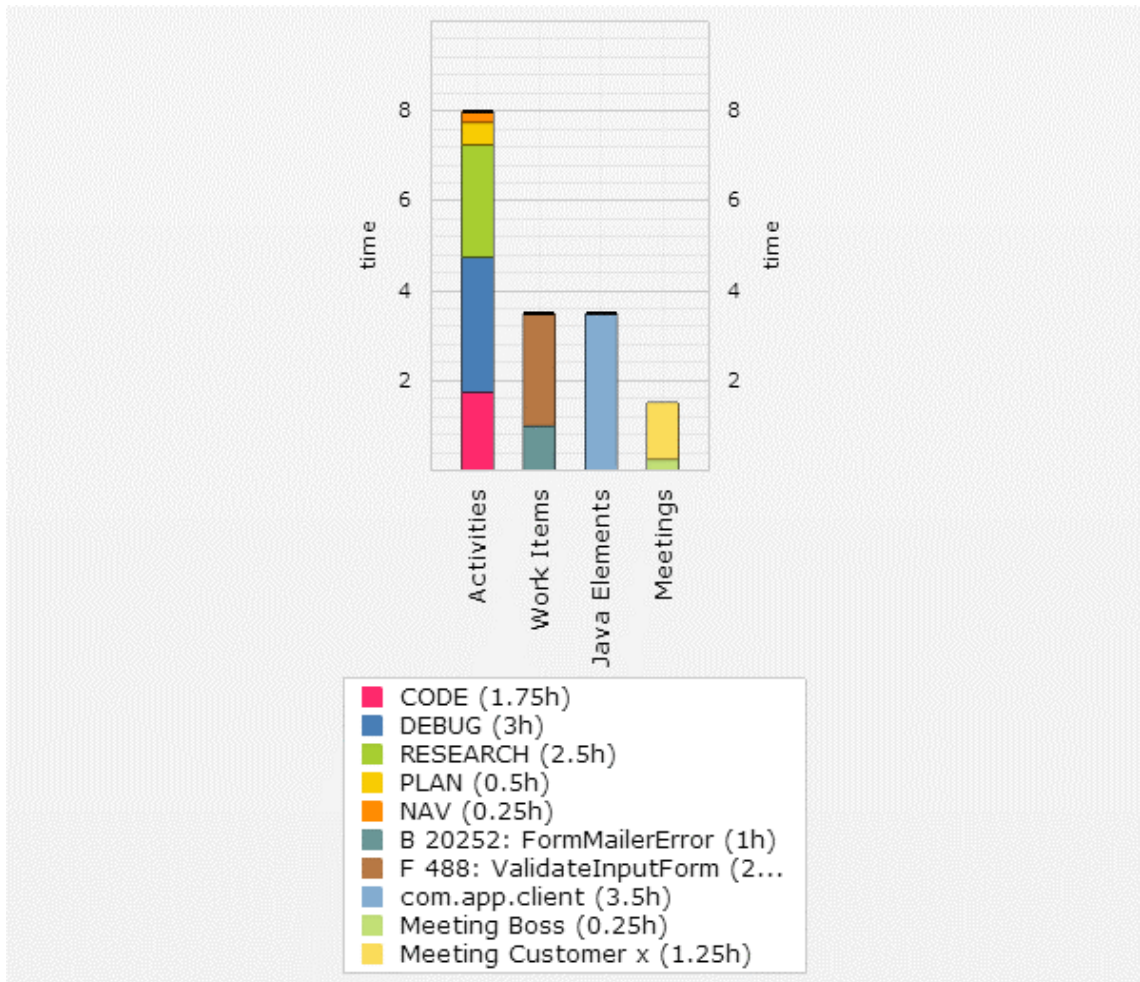


Figure 3.2: Visualization of the most important metrics on the overview page of the web client.

3.3 Future Metrics

In Chapter 3.1, we presented metrics that are tracked by Happy Coder. The architecture of the developer monitoring component allows Happy Coder to be extended with other trackers. This renders it possible to think about adding other metrics measured. Where suitable, some other metrics were already described in the corresponding category. Nevertheless, there is a couple of other innovative ideas that could and maybe should be implemented in a future version of Happy Coder. The most important ones are specified in this chapter.

It could be interesting to measure the level of collaboration the developer has together with his team mates or customers. For example, the number of times he helped a colleague by giving useful hints. Moreover, the means of communication a developer used, how often and the total time he communicated as well as the topics he talked about could be worth tracking. As good communication is a crucial part for the success of a software project, one could probably improve the user's communication effectiveness by indicating and interpreting those details. After each interruption, a developer usually needs a couple of minutes to get back to the topic he was working on before. While it is not always possible to work without getting disturbed, measuring the number of interruptions could help finding reasons for the frequent interruptions and improving the concentration on one topic.

Today, there is an increasing trend to work at home or in several other places. By tracking the user's location throughout the day one might reveal when a developer works where, how much time he travels, what kind of tasks he does at which location and where he is the most productive. As an example, one can imagine that a developer needs a quiet place to think about solving a bug.

Other interesting data like blood flow, brain waves or eye movement patterns could be gathered from the user's body and give feedback on the concentration level of the developer and his overall well-being. Other data like sustenance or healthiness could give further insights into a developer's productivity level. A combination of this data and a suitable visualization could reveal the circumstances that allow a developer to be the most productive.

Measuring software quality is a very important quantification of a developer's achievements and abilities nowadays [SSP11]. By identifying a quality metric, Happy Coder could assemble information of the code quality and give insights under what circumstances a developer produces good or bad code. According to Petre et al. [SSP11], it is very motivating for a developer to see people use the code he produced or if people find the code useful to solve their problems. Hence, it could be very rewarding to give the developer feedback on his source code, such as the number of times it was selected or edited or the number of times it was reused by others.

It was one intention of this thesis to find suitable ways to visualize multiple metrics with one representation. As constituted in Section 3.2, it is very difficult to find one single metric to summarize all achievements with a single quantitative number. The solution to this problem was to present the data with one poly-metric visualization using the Chernoff faces and a couple of visualizations showing single metrics. To find the most/least productive days in a week, we want to think further about approaches to calculate such a poly-metric.

Yet, there are many other metrics that could be measured with Happy Coder. Moreover, it is unclear if any of this metrics reveal useful information to the developer. This is the reason why we conducted a small study to find the metrics developers want to be measured (see Chapter 6.2).

Prototype

The first implementation of Happy Coder is presented in this chapter. Its basic structure and the architecture as well as some implementation details on each of the components are described.

4.1 Architecture Overview

In general, the Happy Coder prototype consists of three parts: A part where the metrics are collected by an Eclipse plug-in. Moreover a server part where the data is stored in a database and where some web services are provided for data requests. The last part consists of a web client which visualizes the data on a multi-platform website. In Figure 4.1, the architecture of the initial version of Happy Coder is visualized.

How are those three components connected to each other? When the user starts Eclipse, given that Happy Coder is installed and activated, it immediately starts tracking. When the plug-in or Eclipse gets closed, and once every hour¹, the data gets automatically uploaded to the server. In order to upload it, the plug-in serializes the assembled data into files and transmits them to the server where the data is analyzed and stored in the database. When the user wants to access the Happy Coder client he navigates to the multi-platform website, using the browser on his tablet, smartphone or computer, and logs in. The server fetches the data necessary for the visualization from the database, prepares it and sends it to the device where it is immediately presented to the user.

Appendix A shows an overview of the used tools, frameworks and environments. The different components of Happy Coder are explained in the following chapters.

¹This interval can be set in the code of the plug-in.

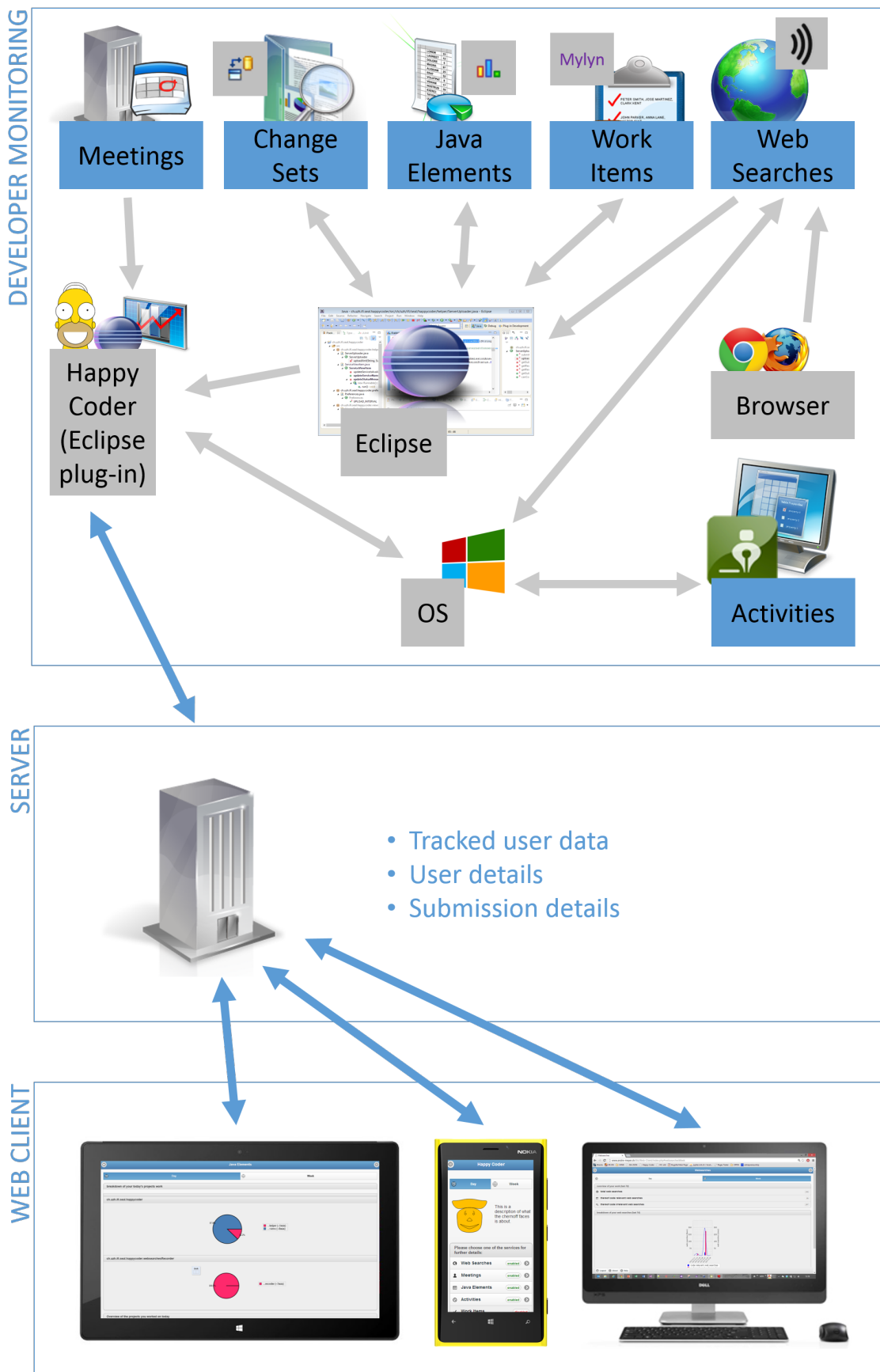


Figure 4.1: Architecture of Happy Code.

4.2 Developer Monitoring

The process of collecting the measured data from different sources is described in this section. A list of the metrics currently being tracked by Happy Coder is presented in Chapter 3. The developer monitoring is structured into multiple autonomous services, called trackers. Each of them registers itself to the Recorder where the correspondent commands get called. They all implement the same interface `IRecorderObject` which provides the important functions to prepare and collect the data. Three of those methods are important to mention: `getServiceIsAvailable` checks if the user enabled the tracker in the preferences and if the data can be accessed before each submission. `saveSubmissionData` assembles the data from the correspondent data storage (database or file) and serializes it to an XML file (one separate file for each tracker). Finally, the function `submitData` gets called by the Recorder to submit the just generated XML file to the server, assuming that a connection to the server can be established. The other functions are necessary to visualize the status in the view and coordinate the communication. A class called `FragmentStartClass` is implemented in each tracker project. This class has an extension point to `org.eclipse.ui.startup`. It creates a new service recorder and registers it to the Recorder.

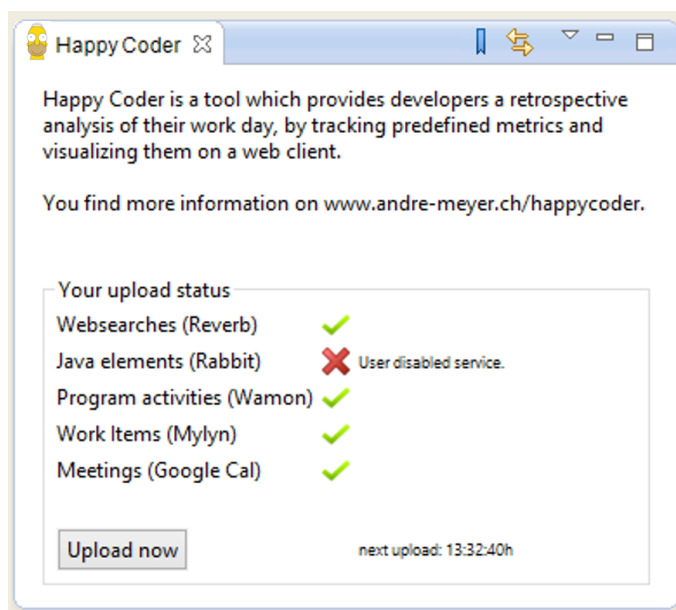


Figure 4.2: Status view of the Happy Coder Eclipse plug-in.

The Recorder coordinates the submissions and all the trackers. The structure of the whole Eclipse project is as follows: There is one master *project* (parent project) `ch.uzh.ifi.seal.happycoder.parent` where all sub-projects are saved. In addition, there is a *plug-in project* (`ch.uzh.ifi.seal.happycoder`) which stores the views, the recorder and all common libraries, models and helper classes. It also enables the whole prototype to be determined as a plug-in by Eclipse. Each of the trackers is a separate *fragment project* (e.g. `ch.uzh.ifi.seal.happycoder.activitiesRecorder`).

Happy Coder consists of one service status view and a preferences view. The service status view can be seen in Figure 4.2 and lists all registered services together with their current status. The status displays the last upload time or error messages. It is designed very simple to disturb the user as little as possible. After an initial setup, it could even run in the background without the need of user input. Moreover, the status view offers the possibility to manually upload the collected data and to get more information via the web client (see Section 4.4).

The preferences view (see Figure 4.3) can be accessed via Window > Preferences > Happy Coder. It provides the possibility to edit a user's credentials, enable/disable certain trackers and edit storage location paths. Disabling a tracker in the preferences does not stop the tracking (as this is done by other plug-ins), but prevents the data from being collected by the Happy Coder plug-in and uploaded to the server. It also determines what information is visualized on the Happy Coder web client.

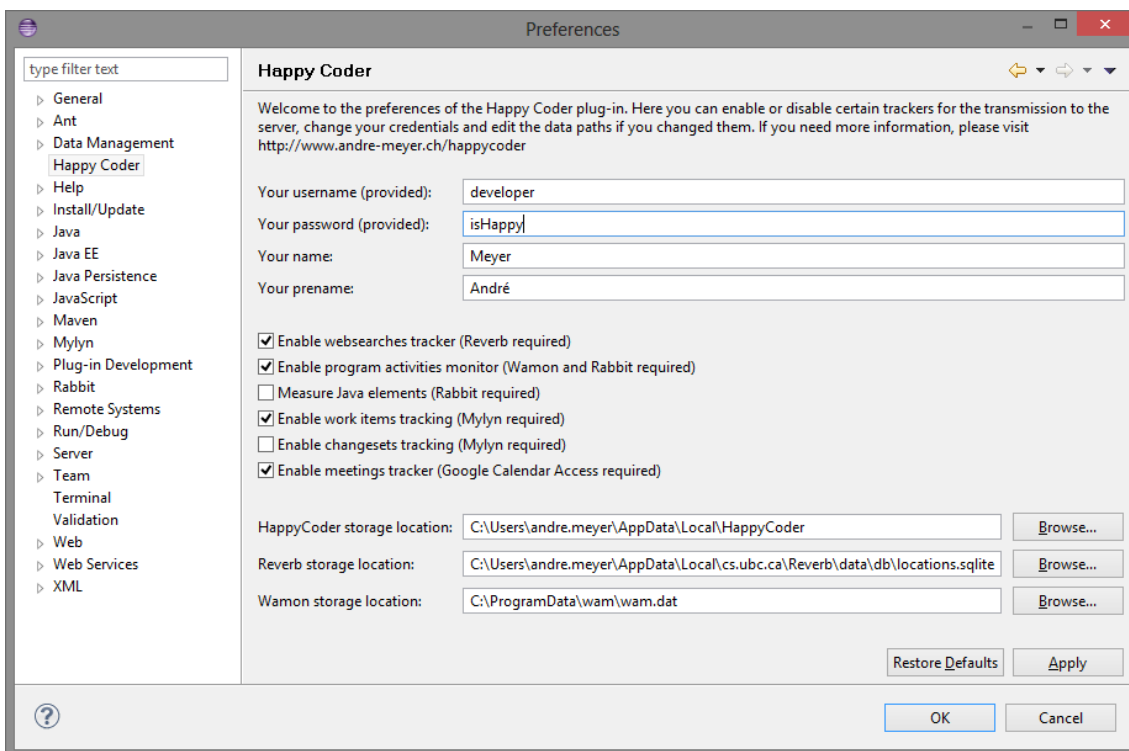


Figure 4.3: Preferences view of the Happy Coder Eclipse plug-in.

4.2.1 Activities Tracker

To track the time a developer spends in each program on his computer, a third-party tool was evaluated. The Windows Activity Monitor (Wamon) is an open source tool that runs on the Windows operating system which allows the access of the data it measures. Wamon was developed in CSharp and tracks active windows and processes. Once a minute, it checks what program (process) is currently active or if it is idle and saves the data into a SQLite database. The database is stored in the ProgramData on the user's computer from where it is accessed by the `ActivitiesRecorder`. By default, Wamon does not track idle times. By changing one boolean in the setting, this feature can be enabled. The `ActivitiesRecorder` in the Happy Coder plug-in automatically attempts to enable this feature to make the activity measuring much more accurate. While it is difficult to determine the times the computer did not run, Wamon solved this problem niftily: As there is one entry every minute (program or idle) no entry will be made if the device is off. If the timespan between two entries is more than two minutes, this must have been a time the computer was unused.

When the `saveSubmissionData` function gets called, the tracker queries the database to get all today's entries. For each entry a helper class (`WamonProcessMapper`) is used to determine the program name. This step is executed with a simple mapper because Java does not allow to receive the file description name from the operating system like it is possible using CSharp². Most of the programs a developer uses on a daily basis are covered by this mapper. The mapper's list could be easily extended if required. As previously mentioned in 3.1.1, the processes are also grouped into some sub categories like *CODE*, *RESEARCH*, *PLAN*. This grouping is also managed by the `WamonProcessMapper`.

The *DEBUG* category is a bit different and will be explained in the following paragraph. Wamon cannot determine if a user is coding or debugging as he works within one or more instances of Eclipse. After the user set some breakpoints and started a debugging session, Eclipse asks (by default) to open the debug perspective. In order to determine the debug time, the time a developer spent in this perspective is measured. This approach of measuring the debug time only in the debug perspective is actually not very accurate. For example, if a user debugs in the 'normal' Java or Plug-in Development perspective, this time will not be measured. This issue could presumably be further addressed in a later implementation. The Rabbit plug-in, which is also used to track Java elements (see Section 4.2.2), offers the possibility to get the duration a user spent in each perspective within Eclipse. By accessing the perspective store (`DataStore.PERSPECTIVE_STORE`), the debug time is extracted and added to the submission item which gets serialized afterwards.

Another metric that is measured by the activities tracker is the total time a developer worked on his computer. This measurement is calculated by summing up all activities' durations and subtracting the idle time. Moreover, the web searches metric contains the total duration a user spent in the browser which is also determined with the activities tracker.

4.2.2 Java Elements Tracker

The Java elements tracker collects all the projects, packages, classes and methods and the duration a developer worked on during his work day. This can be achieved by reading all the elements from the Java data store (`DataStore.JAVA_STORE`) via Rabbit. The `DataStore.JAVA_STORE`

²In CSharp one could simply call `String fileDescription = System.Diagnostics.FileVersionInfo.GetVersionInfo(file).FileDescription;` to get the file description property.

returns a list where the needed elements are selected by accessing hardcoded indices. This might not be the best method, but was the only one that worked. The duration of each of those items gets added up and serialized for the submission.

4.2.3 Work Items Tracker

The work items are managed by Mylyn and are either stored locally or within a task repository. Due to some problems with the Mylyn API and the lack of documentation, the access to the work item repository is established via the `DataStore.TASK_STORE` from the Rabbit plug-in, similar to the approach in the Java elements tracker. The data obtained via the Rabbit plug-in offers the information needed, such as the duration a developer worked on each work item. It could optionally also present all Java elements that are connected to the work item. To get updated work item data from Rabbit, the work item repositories are updated before they are accessed. The data is then gathered and subsequently serialized to the XML file. This workaround functions stably and fulfills the goals. In a future implementation we plan to directly access the Mylyn task repositories through the Mylyn APIs (using the `ITaskDataManager` API). With that change we could also get other interesting metrics like the number of work items a user added, edited, deleted and resolved.

To avoid an information overflow and to reduce the submission time, only the following necessary data is submitted to the server's database:

- The work item id
- A web link to the online representation of the work item
- The duration the developer worked on a work item
- A short description of the work item
- A boolean to show if the work item was completed or not

4.2.4 Meetings Tracker

The developer's digital agenda has to be accessed to track the number of meetings he attended during a work day. In the current implementation of Happy Coder, the developer's calendar is accessed by using the Google Calendar API. This API offers client libraries for all major programming languages including Java. In order to access the calendar, a free API key (with 10'000 requests a day) has to be obtained. The Java client offers an authentication functionality which works after some workarounds³. The API needs a file called `calendar_api_secrets.json` where the API key is saved and it then creates a `calendar_api_user_credentials.json` where the user credentials are saved in a hashed format together with an expiration date. If the user authorization expires, the API automatically starts a new authorization with the Google account. After successfully authenticating with the Google calendar, appointments can be added, removed, edited and fetched. Happy Coder currently only needs a list of all appointments a developer has a day. After fetching the list of appointments from Google, some heuristics had to be applied to each appointment to find out whether it is a meeting or not. For details, see Section 3.1.4.

³The Google API's authentication method does not work in an Eclipse plug-in project but now works in the fragment project. In order to write the user credentials into the mentioned JSON file, the file has to be generated first and '{}' has to be added. Otherwise a 'java.io.IOException' is thrown because it is unable to set file permissions.

In a future implementation we intend to add native Microsoft Outlook support to provide a broader range of supported calendar products, as Outlook probably is the most widely used calendar tool at the moment. A short while ago, Outlook users could have installed a sync tool between Outlook and the Google Calendar, but Google recently ended the support of this tool ⁴.

4.2.5 Web Searches Tracker

To track the websites a user visits during his work, the data is gathered from the Reverb Bookmarks plug-in for Eclipse. The Reverb Bookmarks service consists of three components:

- **The indexing service.** This service is used to index the content of web pages a user visits and to respond to queries from the Eclipse plug-in.
- **The browser extension.** A browser extension for Google Chrome and Mozilla Firefox runs a background agent which transfers the page content of the web sites to the indexing services. As there are only two browsers supported, the user has to use them in order to be able to get information about his web searches.
- **The Eclipse plug-in.** The plug-in recommends related web pages according to the Java code a developer is currently working on by generating queries.

Reverb stores the collected data in a SQLite database, where it is accessed by the web searches tracker. Reverb only stores a summary of the visits of different web searches: for each visited web site there is one entry containing the last visit time, the number of visits and two flags if the web site is code or Javadoc relevant or not. As Happy Coder only needs the number of visits today and not the total number, this makes it more complicated to obtain the needed data. Upon start-up of the Happy Coder plug-in, the tracker creates a backup list of all web requests a user conducted. This list is then serialized to an XML file to restore it if the user closes Eclipse and restarts it the same day to restore the original data. To prepare the submission data, the current database entries (i.e. the visits count) is compared to the visits count from the yesterday backup list. The difference is the number of times a user visited the web site today. Using the code or Javadoc relevance flag it is determined if this web site is relevant to the written code or not. This data is subsequently stored in the submission item and serialized for the submission.

4.2.6 Change Sets Tracker

As previously mentioned, the change sets tracker is currently not implemented in Happy Coder, but has a high priority for a future release. The Mylyn plug-in used in the work items tracker offers a task-focused view of the change sets; the *Synchronize* view. It contains information of the change sets of all code changes in combination with a work item. Multiple connectors offer access to a broad range of source code management systems like SVN, CVS or Git. Another advantage is, that no additional plug-in has to be installed.

⁴<http://support.google.com/calendar/bin/answer.py?hl=en&answer=98563>, verified 01/10/13

4.3 Server

The server and its database are responsible for the management of the data feeds, either incoming or outgoing. The infrastructure utilized to run the server and the database is described in Appendix A. Generally, the server offers two access points:

- **The data upload service.** This service is used to transfer the collected data from the Eclipse plug-in to the server's database. Each of the plug-in's services has one XML file to submit to the server. The data is then saved in at least one SQL table.
- **The data web service** offers a web service to access the saved data from the database. A user can authenticate with the system via the multi-platform web site where his personal data is visualized. The service is offered in form of a REST⁵ web service.

Before going into further details of those web services, the database structure is presented which is crucial for the understanding of those web services.

4.3.1 Database

The database stores all the data necessary for the visualization to provide the developer with a retrospective analysis of his work day. Just as the web server, it is currently hosted on `www.andre-meyer.ch`. It basically contains a table for the users, the uploads and at least one table for each tracker. The `hc_users` table stores all necessary information of the user: some personal information like the name or the username, a flag if the user is enabled or blocked, a password (which is MD5 hashed), and a unique id. As the user wants to submit the collected data he first sends the configuration file to the server which is double-checked with the data in the database. The `hc_uploads` table stores the settings data of each submission. It also contains a unique id to identify the upload as well as a connection to the user. Moreover, it has a flag for each tracker to store if it was enabled or not at the time of the submission. In addition, it has a record of the exact date and time (timestamp) of the submission as well as a flag to see whether the upload is the today's latest. As soon as a new submission overrides the old one, this flag gets changed for all previously uploaded submissions of the correspondent day. Finally, there are multiple other tables storing the data of each tracker, like `hc_activities`, `hc_meetings`, or `hc_websearches`. A detailed representation of the database, its entities, relationships and attributes can be found in Figure 4.4. The visualization is in the form of an entity-relationship model (ERM).

4.3.2 Data Upload Service

The data web service consists of a couple of PHP files and a connection to the database. The process of uploading the data is visualized in Figure 4.5 and is described in this chapter.

As previously described, the data upload gets initiated through the `Recorder` as soon as Eclipse or the plug-in shuts down, and in a certain interval (e.g. once an hour). In order to start a new submission the configuration gets serialized and transferred to the server. To initiate the XML data upload from the Eclipse plug-in to the database, the `ServerUploader` connects to the server by creating a new `HttpClient`. It adds the url to access, some request headers and the file input stream of the XML file. Moreover a submission type is added to the url. For example, this could be 'CONFIGURATION', 'WORKITEMS' or 'ACTIVITIES'.

⁵SOAP and XML-RPC are the most common types of web services. REST (Representational State Transfer) is another web service which receives requests via parameters in the url and returns the data as XML, JSON or HTML.

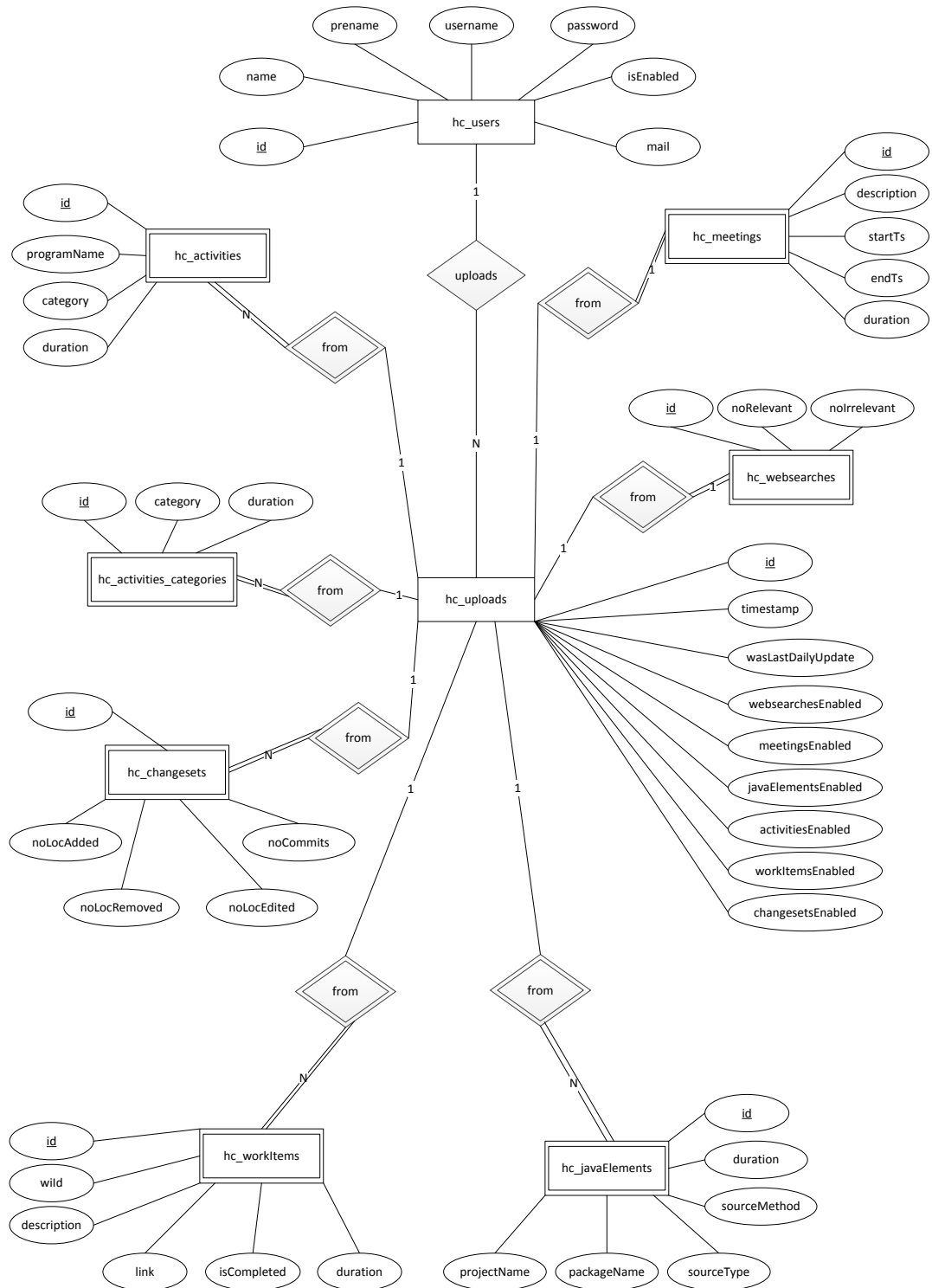


Figure 4.4: Entity Relationship Model of the database.

The following snippet (see Listing 4.1) shows the composition of the url:

```
String baseUrl = "http://www.andre-meyer.ch/BA/Server/services"
                + "/dataUploadService.php";

SubmissionType type = SubmissionType.ACTIVITIES;
String inputUrl = String.format("%s?type=%s", baseUrl, type.toString());

// the resulting url looks like this:
// http://www.andre-meyer.ch/BA/Server/services/
// dataUploadService.php?type=ACTIVITIES
```

Listing 4.1: Java snippet showing the composition of the url on the example of the activity tracker.

The `dataUploadService.php` receives the request containing the submission type and the submission data (see Listing 4.2).

```
$getSubmissionType = Helper::getGETValue( "type" );
$getSubmissionData = file_get_contents('php://input');
```

Listing 4.2: PHP snippet showing how the submission type and file is obtained.

As the server gets the configuration file it verifies the user and saves the submission information to the database. The information includes the user id and the enabled trackers. If no error occurred, the submission id will be returned and the submission can proceed. The server will then response to the request. We introduced a simple response protocol in the form `STATUS#ID#MESSAGE`. The status is either *SUCCESS*, *WARNING* or *ERROR*. The id represents the submission id needed to authenticate the submission of the files. The message is used to tell the plug-in some details in case of errors which will be saved into the log files. As soon as the `Recorder` knows the submission id, it spreads the command to each registered tracker to submit their data. Every tracker collects the necessary data and serializes it (including the submission id) to an XML file. Those XML files are then transferred in the same way as just explained: the server needs a submission id, submission type and the submission data. The XML data will be parsed and stored in the database. Finally, the server returns the status of the request. Back in the plug-in, the server updates the view to show the status of the submission and writes the status into the log file. It then resets the interval timer to start a new submission once it expires.

4.3.3 Data Web Service

The web service receives the request details via the `GET` variables from the web client. After a successful authentication the web client sends requests to the server when a user navigates to different pages. Each request contains the user id, the request type, and a sub-request type if necessary. The inquiry is then checked and sent to the database connector (`database.php`) which handles all requests to the database with correspondent methods. The predefined query is executed and the result prepared and enriched with status information to the wished format. The answer object is serialized to the JSON format and printed by PHP. JSON has the advantage of minimized data usage which is ideal for data transmission to mobile devices. Furthermore, it is based on JavaScript and offers direct access on the data objects via JavaScript. In some cases (e.g. to draw the work items pie), the web client starts a request where the server directly outputs the answer as JavaScript code which is the web client runs. This is due to some limitations of the `CanvasXpress` visualization framework. This is probably not the best way to implement, but works very stable and was recommended by the creator of the framework.

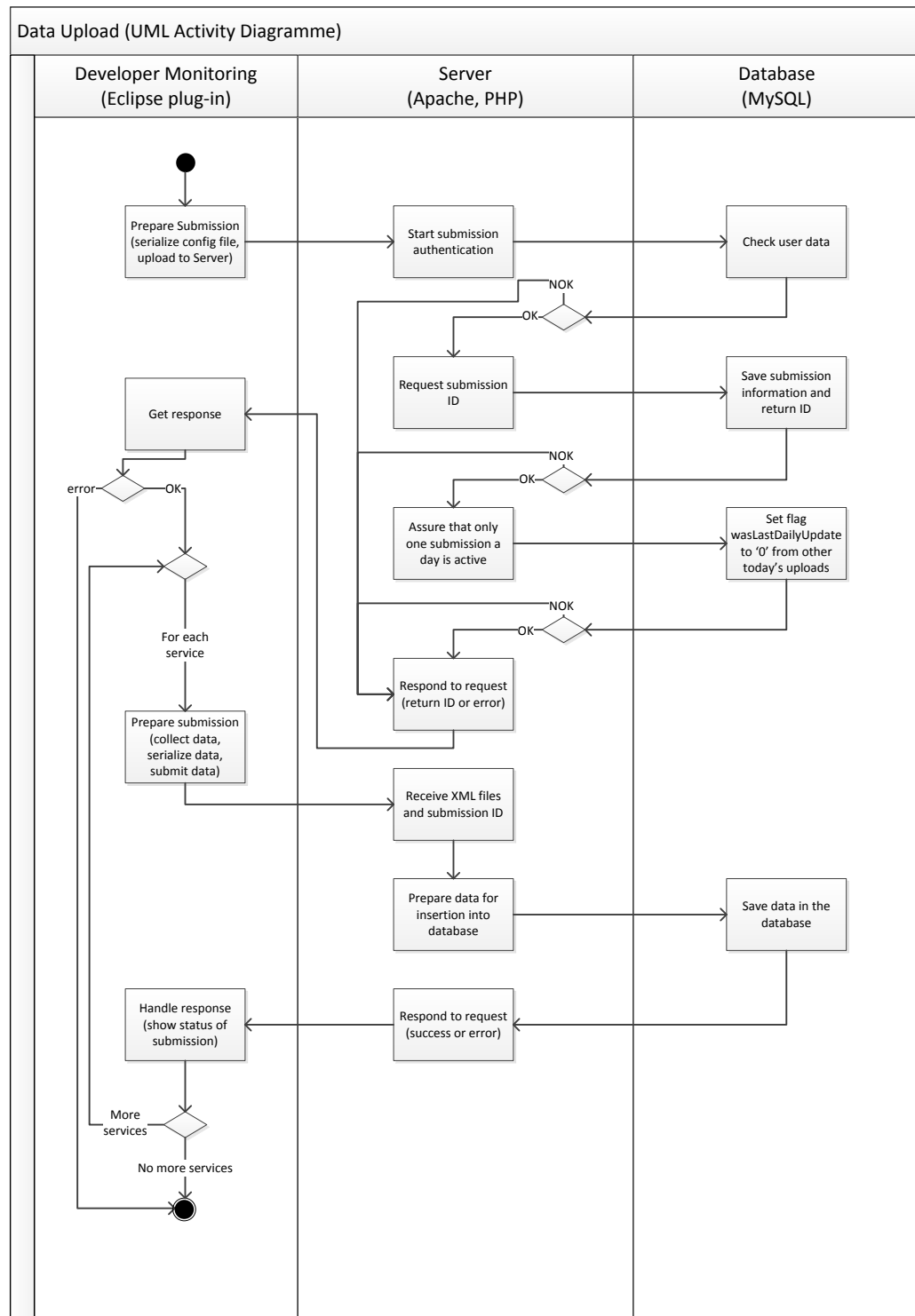


Figure 4.5: UML activity diagram of the data upload to the server's database.

4.4 Web Client

While the Eclipse plug-in and the server work predominantly in the background, the web client is the most important component to the user. It visualizes the collected data to the developer in a comfortable way on almost all browser enabled devices, such as smartphones, tablets and computers. This has the advantage of offering almost device and location independent access. Moreover, the client can be updated and maintained without redeploying it to the user. Disadvantages of using a web site are the single point of failure (server), some privacy concerns and the complex development (e.g. difficulty of debugging and testing). As web clients usually have browser compatibility issues, this problem could mostly be solved with jQuery mobile. Additionally, the rapid evolution of the technologies, libraries and tools need a consistent maintenance.

One could imagine that a developer wants to see the retrospection of his work day on the elevator ride down to the exit of the company's building or while commuting home by train. Therefore, the visualizations are optimized for viewing on a device with a small screen, like a smartphone, and for viewing on a bigger screen like a tablet or computer. To that end, we evaluated multiple frameworks to create browser independent websites and decided to use jQuery mobile because of its broad and stable use in other commercial and non-commercial applications. To generate the visualizations, we use canvasXpress because of its easy-to-use library. Due to some drawbacks including scrolling issues on certain devices and limited adjustment capabilities, we will use the more powerful and stable D3 framework in a future implementation. The Chernoff face is also created with the D3 framework.

The web client consists of multiple pages, all designed with the same layout; optimized for touch, user friendly and containing animations. The user is always redirected to a log-in page if he was not yet authenticated. He can insert his credentials which are immediately checked with the database. After a successful log-in the overview page is the usual entry point to the website. As discussed in Chapter 3.2, it summarizes the collected information of the developer's work day using the Happy Coder Chernoff face and a stacked line graph. On the bottom of the main page there is also a list containing all services that are currently supported by Happy Coder together with their current status (enabled or disabled). By clicking on one of the enabled services, the user is redirected to a new page containing detail information of the selected service. A toggle button enables the possibility to switch between today and the last seven days. On every page are buttons to open the navigation menu, to log-out, to get help and to get information about the authors. A screenshot of the overview page is presented in Figure 4.6. Screenshots of the other pages are attached in Appendix B. In the current implementation of the web client, the data is presented in boxes in a list and the user needs to scroll to see all visualizations. In a future version, it is planned to enhance the web client to better use the available width of the window by presenting the boxes side by side as it is represented in the screenshot.

This paragraph describes the most important details of the implementation of the web client. The website is based on the `index.php` file. The majority of the code is HTML, and there is also some JavaScript and PHP code in it. To follow the jQuery framework guidelines, the whole website is stored in this single file. Thus, the file is more than a thousand lines long. This drawback gets rather insignificant in regard to all the advantages that jQuery offers with its out-of-the-box features. The `index.php` file contains multiple `div`-tags, one for each page. This has the advantage that after an initial loading, the page can be navigated back and forth very fluently and nimbly. The visualization data is fetched from the server as soon as the user navigates to a certain page to reduce the initial loading time. A loading sign and some status texts are displayed while the data is downloading. `data.js` is another file worth mentioning. It is responsible for the com-



Figure 4.6: Overview page of the Happy Coder web client.

munication between the server and the web client. The web client runs a PHP method (described in Chapter 4.3.3) via Ajax call and prepares the received data to visualize it on the website in a list or graphical representation. If a visualization contains a dynamical number of items (e.g. if all projects a user worked on are visualized), it is generated on the server and then sent to the client to be run. In the day view, the numbers (e.g. number of web sites a developer visited) are usually presented together with an average value and illustrate trends. Long lists, like the Java elements, are presented in a collapsible list to reduce the need to scroll in long lists and to let the user see only what is important to him. Appropriate messages are shown in case of errors or if no data is available.

The web client can be accessed on <http://www.andre-meyer.ch/hc-web-client>. Alan's imaginary work day, which was introduced in this chapter, is presented with a demo account. The username is `developer` and the password is `isHappy`.

Usage Study

After the implementation of the prototype a usage study of Happy Coder was conducted. The goal of this study was to investigate if productivity could be measured with the chosen metrics. Therefore, the importance of the metrics and the correlation of each metric with the productivity was evaluated. For 30 days, the author of this thesis collected data of the predefined metrics together with an estimation and justification of his productivity.

5.0.1 Results

Table 5.1 specifies the metrics that were measured together with the correlation coefficient between the metric and the personal rating of the productivity. Moreover, the p-value from the t-test was calculated to determine if the correlation coefficient is significant or not. Most of the metrics are already measured by the prototype while others were measured out of curiosity (marked with a '*'). To prevent outliers to distort the correlations and to get more accurate results, two days were ignored from the calculations. On those two days the author of the thesis did not work. An overview about all the results of the study can be found in Appendix C.

Metric	Correlation	p-value
Correlation meetings <-> personal rating	-0.299	7.682E-13
Correlation total time worked <-> personal rating *	-0.163	1.155E-05
Correlation time on computer <-> personal rating	-0.077	5.039E-04
Correlation time planning & communicating <-> personal rating	-0.349	1.448E-10
Correlation time coding <-> personal rating	0.043	1.077E-07
Correlation time debugging <-> personal rating	0.210	2.518E-14
Correlation time researching <-> personal rating	-0.332	2.853E-08
Correlation time navigating <-> personal rating	-0.236	2.779E-14
Correlation time doing other things <-> personal rating	0.145	1.377E-13
Correlation number of solved tasks (Mylyn, Outlook, OneNote) <-> personal rating *	-0.135	1.318E-04
Correlation different Java projects <-> personal rating	-0.106	5.679E-10
Correlation number of commits <-> personal rating	0.246	5.605E-10
Correlation lines of code committed <-> personal rating	0.147	7.126E-06

Table 5.1: Results of the usage study where the Happy Coder prototype was evaluated.

The results, all with a p-value smaller than 0.05 and thus considered as significant, indicate

the tendency that there is no single metric to measure productivity. No metric had a very high correlation towards the personal rating of productivity. Some results are interesting to point out though. The highest correlation is calculated between the time a developer spent planning and communicating (-0.349) and the personal rating of his productivity. This negative correlation shows that it is difficult for a developer to concentrate on his work if he has a lot to plan, coordinate and communicate. The time a developer was searching for information has also a high negative correlation to productivity (-0.332). This was expected, as searching the web for help or answers related to a problem is very time consuming and sometimes painful. If the developer finds the right answer and is able to solve the problem (i.e. fix the bug, complete the work item), we expected this to have a positive influence on the assessment of the productivity. According to a correlation of only -0.135, the number of solved tasks has almost no influence on how productive a developer was. This is an unexpected result. We assume the reason to reside in the definition of a work item. In this self-evaluation, we considered Mylyn tasks, Microsoft Outlook tasks, emails flagged to answer and some to-do's in Microsoft OneNote as work items. A day full of important emails to answer would lead to a lot of completed work items but probably not to a feeling of a productive work day. Another interesting correlation was observed between the total time a developer worked and the personal rating of his productivity. The negative correlation of -0.163 could indicate that sometimes a short but intensive work day might also be a productive one.

5.0.2 Limitations and Conclusions

The usage study suggested that the productivity cannot be measured with a single metric. The results indicate that there is a certain correlation between the metrics and the personal rating of productivity. This supports our decision to visualize all the metrics with a poly-metric approach and let the user choose the metrics he wants to be measured. Limitations in this study could occur due to the fact that only one participant was tracked and due to the relatively short period of the study (30 days). Moreover, removing two outliers could also have an effect on the results as well as the fact that some metrics were manually measured at that time. Additionally, only the metrics were tracked that were already implemented in the prototype at the time of the study or could be tracked by hand. As described in Section 7.2, we intend to conduct a bigger study over a longer period of time and with more participants to learn more about the correlation between the metrics and a developer's productivity.

Metrics and Productivity Studies

In the last chapter, we introduced a tool, called Happy Coder that allows a software developer to get a summarized overview of his daily work and achievements. We examined the feasibility of collecting data on various aspects of a developer's work, visualized them on a multi-platform website and evaluated it with a usage study. In a second step, we conducted two small studies with multiple developers to evaluate the metrics that are useful to a developer and how he assesses his productivity. The results are explained in this chapter.

6.1 Productivity Study

In a first study, professional software developers were asked about their productivity as a part of a software engineering training course. They had to answer eight questions about their work routines. Two of the questions concerned the productivity of a developer and are listed as follows:

- **Question 1:** Do you think that you have been productive during your last work day?
(very unproductive, unproductive, undecided, productive, very productive)
- **Question 2:** How do you assess your productivity at the end of a work day?
e.g. number of achieved tasks, etc.

The participants group consisted of the seventeen participants of the seminary who work for the same company. They were asked to answer the questions on an online survey website. The first question consisted of the question itself and five given answers using the Likert scale (1 = 'very unproductive', 5 = 'very productive'). The second question included an example and had to be answered in an empty text box.

6.1.1 Results

The results for Question 1 are presented in Table 6.1. We categorized the answers for Question 2 and counted the number of mentions. The results for Question 2 are presented in Table 6.2. An overview about all the results of the study can be found in Appendix D.

An average of 3.29 and a standard deviation of 0.75 indicate that developers have difficulties assessing their productivity and therefore are mostly 'undecided'. They have the feeling that they

Metric	Mean	Median	STDV	Min	Max
Do you think that you have been productive during your last work day?	3.29	3.00	0.75	2.00	4.00

Table 6.1: Results for the question (Question 1) if the developer's last work day was productive. (1 = 'very unproductive' / 5 = 'very productive')

tended to be productive. The same conclusion could be made of the answers of the second question. 4 of the 17 asked participants (24%) did not have an idea how they assess their productivity although an example was given with the question. Most of the others gave their answer based on this example. One participant gave an answer that was mentioned only once and therefore is listed in the category 'other'. He assesses his productivity just on his 'personal feeling'. This is probably the technique that the other developers with no idea of their productivity also do.

Answer	Number of mentions	Hints
Number of finished planned tasks	7	current phase of a project, e.g. bug fixing near the end, last minute changes the customer wants, ensure the team knows the plan in the beginning
Number of completed work items (no planning)	2	
Coding versus communication time	2	
Depends on the kind of the task	2	
Other	1	personal feeling
None	4	

Table 6.2: Categorized answers to the open question (Question 2) of how developers assess productivity.

6.1.2 Limitations and Conclusions

The study's participants work for the same company. Due to this selection, it remains unclear whether the results can be applied to software developers in general or not. It was unexpected to see most of the participants not knowing if their previous work day has been productive or not. The results of the second question might be a bit biased as an example was attached to the question. Most of the developers gave their answer based on this example and did not come up with other explanations of their productivity assessment. Additionally, some answers do not further specify what a developer defines as a task. This could be tasks within a bug repository like Mylyn, a task management system like Outlook and might even include the number of answered emails. Finally, the answers given in this study indicate the trend of heavily relying on the tasks the developers completed; sometimes in comparison with the work they planned to do.

6.2 Metrics and Productivity Study

We performed a second study to probe the metrics that are important to a developer. To use the opportunity to perform a survey with developers, we again asked them to explain how they assess their productivity. For this purpose, a questionnaire was designed that consists of three parts. In the first part the participants received a list containing the same metrics that are categorized and described in Chapter 3.1. There are also some additional metrics, we intended to add to Happy Coder in the future but first wanted to find out if they are important and relevant to the developer. The participants had to rate each metric on a scale from 1 ('not important') to 5 ('very important'). In each category, they had the possibility to suggest other metrics. The second part of the questionnaire consists of a single open question on how the developer assesses his productivity. The third part is an empty text field where the participant could write some comments or suggestions. The suggestions are already included in the future work as described in Chapter 3.1. The questionnaire can be found in Appendix E.

The study was performed with 10 members of our software engineering and architecture lab (s.e.a.l.). Two of the participants are professors, two are senior research assistants, two are research assistants and four are master students all with a background in computer science. Due to the limited amount and easy to answer questions, all participants answered the questionnaire in less than 10 minutes.

6.2.1 Results

In this chapter the study's results are presented in more detail. An overview about all the results of the study can be found in Appendix F.

Metrics

The first part of the questionnaire addresses the metrics that are important to a developer. The participants had to rate each metric on a scale from 1 ('not important') to 5 ('very important').

Activities The results in Table 6.3 indicate that for most participants measuring the activity on the computer is very important. Participants also suggested to measure the time spent for communication (e.g. chat, voice-over-ip), reading or writing the documentation and testing. In a future study, it would also be interesting to find out if the developer likes the categorization as it is provided now, or if he prefers another categorization.

Metric	Mean	Median	STDV	Min	Max
How much time did the developer spend in which program? (Categorization: Code, Debug, Plan, Research, Navigate, Other)	3.40	4.00	1.28	1.00	5.00

Table 6.3: Results for metrics in the category Activities. (1 = 'not important' / 5 = 'very important')

Work Items Overall, the participants rated the work items metric between 3 ('undecided') and 4 ('important'). The mean (3.7 and 3.4) and the median (4 and 3.5) indicate that they are mostly interested in the number of work items a developer worked on and the number of work items he

solved, edited, created or deleted. The average (3.1) and median (3) as well as the high standard deviation (1.14) indicate that it is a personal assessment whether the duration a developer worked on a work item is important or not. One participant suggested to take the difficulty of a work item into consideration while another proposed to include estimates of the duration to solve a work item to compare with the actual duration.

Metric	Mean	Median	STDV	Min	Max
Number of work items a developer worked on	3.70	4.00	0.90	2.00	5.00
How long a developer was working on a certain work item	3.10	3.00	1.14	1.00	5.00
Number of work items a developer solved/edited/created/deleted	3.40	3.50	0.92	2.00	5.00

Table 6.4: Results for metrics in the category Work Items. (1 = 'not important' / 5 = 'very important')

Java Elements The results in Table 6.5 indicate that most participants rate the measurement of Java elements (projects, packages, classes, and methods) as less important. Nevertheless, the relatively high standard deviation (between 0.92 and 1.27) shows that some participants value that feature highly. With an average of 2.2 and a median of 2, most of the participants do not want the number of commands they executed to be counted. One participant proposed to measure the access level modifiers (public, private, protected) to know how they influence the rest of the project. A metric to measure the number of files touched was also desired by one participant.

Metric	Mean	Median	STDV	Min	Max
Time a developer spent on which project	2.60	2.50	1.02	1.00	5.00
Time a developer spent on which package (Java)	2.40	2.50	0.92	1.00	4.00
Time a developer spent on which class	3.10	3.00	1.14	1.00	5.00
Time a developer spent on which method	2.70	2.50	1.27	1.00	5.00
Number of commands a developer executed (copy, paste, save, open, close, step-into (debugging), etc.)	2.20	2.00	1.08	1.00	4.00

Table 6.5: Results for metrics in the category Java Elements. (1 = 'not important' / 5 = 'very important')

Change Sets Measuring the number of commits a developer made was rated as less important by most of the participants, with a relatively low mean (2.6) and median (2). Nonetheless, a relatively high standard deviation (1.11) indicates that some participants think this metric is important while others would not use it at all. The participants considered the lines of code a developer submitted, added or removed as a bit more important. With relatively high standard deviations (1.22 and 1.04) it still remains up to the discretion of the individual developer to decide if he uses it. With an average value of 3.1 the most important information the participants wanted to take out of the change set metric was the number of selects and edits a developer made. A participant indicated that for him, measured data is only important in relation to other metrics. We assume, that the combination with work items would make sense in this case. Another participant suggested to measure the code complexity of the code churn.

Metric	Mean	Median	STDV	Min	Max
Number of commits a developer made	2.60	2.00	1.11	1.00	5.00
Lines of Code (LoC) a developer submitted	2.90	2.50	1.22	1.00	5.00
Lines of Code (LoC) a developer added/removed (Code Churn)	2.90	3.00	1.04	1.00	4.00
Number of classes/methods/variables a developer added/removed/edited	2.90	2.50	0.94	2.00	4.00
Number of code elements a developer looked at (selects/edits)	3.10	3.00	0.70	2.00	4.00

Table 6.6: Results for metrics in the category Change Sets. (1 = 'not important' / 5 = 'very important')

Web Searches Most participants considered tracking the number of code related and code unrelated web searches as important. Nevertheless, a standard deviation of 1.19 indicates that this metric is not important for all developers. One participant rated the metric with 1 ('not important') and two participants rated the metric with 2 ('not very important'). A participant suggested to include the duration a developer spent browsing the web to the metric. We considered this suggestion as very important and already implemented this feature. Two of the participants also wanted to know the types of web searches they did, like searching about code or about documentation. This could be an interesting metric, but very difficult to measure.

Metric	Mean	Median	STDV	Min	Max
Number of web searches a developer made	2.90	3.00	1.30	1.00	5.00
Number of code related / code unrelated web searches a developer made	3.30	4.00	1.19	1.00	5.00

Table 6.7: Results for metrics in the category Web Searches. (1 = 'not important' / 5 = 'very important')

Meetings The results in Table 6.8 indicate that most of the participants want to see the meetings they attended as well as the duration and topic of each of these meetings. However, a high standard deviation (1.47) denotes that this metric is not important to all participants - probably, because they remember it after their work day. One participant wanted to see an overview of his meetings not per day, but per month. A similar feature is already implemented with the view to look back at the meetings of the past seven days. Another suggestion of a participant indicates the need to track the number of support requests a developer has to handle.

Metric	Mean	Median	STDV	Min	Max
Number of meetings a developer had	3.60	3.50	1.11	2.00	5.00
Duration and topic of each of these meetings	3.20	3.00	1.47	1.00	5.00

Table 6.8: Results for metrics in the category Meetings. (1 = 'not important' / 5 = 'very important')

Productivity

In the second part of the study the participants had the possibility to describe how they assess the productivity of their past work day. This part of the questionnaire consisted of the following question: 'How do you assess your productivity at the end of a work day?'. In contrast to the other productivity study we conducted (see Chapter 6.1), we gave the participant no example or hint to answer the question as we wanted to get a less influenced answer than we got in the previous study. To evaluate the answers, they were grouped into the same categories as in the previous study. The results are categorized in Table 6.9 and counted according to the number of mentions.

Answer	Number of mentions	Hints
Number of finished planned tasks	5	number of bugs found/done/resolved, number of features implemented
Number of completed work items (no planning)	6	
Coding versus communication time	2	
Depends on the kind of the task	1	soft factors, formula, important, feeling accomplished something
Other	4	
None	0	

Table 6.9: Categorized answers to the open question of how developers assess productivity.

One developer stated that he distinguishes between a work day with a lot of meetings and a day with only a few ones to assess his productivity. In case there were a lot of meetings and little time to work he considers his day as productive. Otherwise he compares the planned work items with the completed ones. Four answers could not be classified in the predefined categories and were only mentioned once by a participant. Those are:

- Soft factors that determine productivity
- A productivity formula: 'my productivity = solved issues/work items/tasks + effort spent on classes & bugs - 'distractions' by web & chat'
- 'If I know that I have achieved something important. Does not matter if it is the target I aimed for at the beginning of the day.'
- 'If you feel you accomplished something - then it was a good day'

In contrast to the previous productivity study (see Chapter 6.1, all participants reflected on what they understand of a productive day. This is interesting as we gave no hint or example this time. The other answers were very similar. Most of the developers assess their productivity either by the number of work items they solved. In both studies, about half of them compare their completed work items to the planned ones. Furthermore, both studies contained a participant who assesses his productivity based on feelings.

Feedback and Suggestions

Only two participants used the field for suggestions. Both of them indicated that most of the presented metrics are quantitative. They recommended to add some qualitative metrics like software quality. This is an important hint we also added to the metrics we want to add in future as described in Chapter 3.3.

6.2.2 Limitations and Conclusions

Due to the selection of the participants the results of this study could be biased. All participants work or study in the academic area (software engineering) in our research group. Some work part-time in a company outside the university or their own start-up. Another possible limitation could be the relatively small group of participants (10). This could make it difficult to guarantee a representative distribution and find significant relationships. Nevertheless, the answers were within our expectations from the previous productivity study (see Section 6.1), other studies (see [Car06,PSHH04]) and our own experience. Relating to the first part of the questionnaire, there was no metric that they strongly declined or strongly rated as very important. We learnt that the importance of a metric depends on personal preferences and experience. As there was no single metric that was significantly more important than the others. We did not find a single metric to represent the developer's work day. This was again as expected. The results encourage our decision of working with a poly-metric approach for the overview visualization and to leave the choice to the user to see only the metrics he needs. Expecting this outcome, this features are already implemented in our prototype. Moreover, we got a lot of suggestions for other metrics that could be implemented in a future version. In the second part of the study the participants had to explain how they assess a productive work day. Similar to the first productivity study, they heavily rely on the work items they completed; sometimes in comparison with the planned work. The very similar answers in both productivity emphasize the versatility of different attitudes towards important metrics and measuring productivity. In a future study, we want to scrutinize if Happy Coder measures the right combination of metrics and offers enough flexibility for each individual developer. Furthermore, we want to assess if a developer could evaluate his productivity with the retrospective analysis, offered by Happy Coder, and if this makes him happier in the long run.

Discussion

Major drawbacks and possible limitations of the thesis are discussed in this chapter. In addition, the importance of the tool is emphasized and some future work concerning the Happy Coder prototype is presented.

7.1 Discussion

A developer's work day usually consists of some ups and downs - productive times and productivity leaks. Especially in a down phase it is important to get some objective information on the achievements of today's work. As presented throughout the last chapters, Happy Coder is one of the first approaches to combine multiple aspects of a developer's work, focussing on providing a review of his work day. This makes it almost impossible to compare Happy Coder to another, similar tool. Other approaches in the area of software engineering concentrate on providing information on a single domain of a developer's work (e.g. debugging). While sport activity trackers like the Fitbit have a single metric (step count) to analyze and visualize the user's activity, this simplification could not be assigned to measure a developer's productivity. This observation was also confirmed by the two studies we conducted and the related work. As the participant group in both studies was relatively small and many of the participants work in the academic area and not in a company, it remains unclear whether these results could be generalized to all developers. Moreover, the small evaluation and the two studies did not check if Happy Coder can increase a developer's motivation or make him happier. This is to be probed in a future study.

The visualizations in the web client are based on the metrics, the feasibility of the implementation, and the assumptions of useful combinations of the metrics. In a future study, we intend to ask developers to reflect if the right metrics are visualized in the appropriate way.

There are some possible improvements concerning the implementation of the tracker, the server and the web client which could improve the stability and usability of the prototype. They are described in the next chapter. At the moment, the tracker only works if the plug-in within Eclipse is activated and it depends on multiple external plug-ins. The installation of those plug-ins might be tedious and could prevent a developer from using it. After the installation and an initial configuration, the Happy Coder tracker runs in the background without engaging the user. This restriction could not be solved otherwise due to the limited duration the project was allowed to last.

As the Java open source environment offered the most tools whose data could be accessed, the decision was made to develop for this environment and not for another. This caused some

troubles while implementing the prototype. For example, those tools (e.g. Mylyn, Rabbit) are almost not documented and it was very difficult to find out how to access their data. Furthermore, getting help and support for such open source tools was extremely difficult and time consuming. This entailed multiple drawbacks, like delays, postponing features to future releases and changing the architecture of the prototype. Another example was the Google Calendar API that offered old documentation and old examples which did not work anymore. Furthermore, it was difficult to get a contact to inform Google about a bug in their API and to get some help to find a workaround. In order to get the meetings for as many developers as possible, we initially decided to use the Google Calendar API. The reason for this decision was that it can access the appointments from all Google Calendar users and, over the Google Sync Client, from Outlook users. Unfortunately, a very short time before the release of this thesis, Google discontinued the support of this tool. As a consequence, only Google Calendar users can currently use the Happy Coder's meetings tracker, unfortunately. Such drawbacks might not have occurred with the use of commercial tools, which usually announce the end of a software product many years in advance. In contrast to open source tools, commercial tools might not allow the use of their tools in a project like Happy Coder or insist on complex and costly licensing structures.

Sending private data to a server reveals possible privacy concerns. On the one hand, developers might not want the data to be revealed to their boss and on the other hand, the data must be secured from unauthorized access. Currently, the data is sent without encryption to the server but can only be accessed when a user authenticates. Another possible solution to any privacy concerns could be to save the data only locally and visualize them directly on the work device. Finally, a possible limitation that comes with all those trackers of this type is that a user might always work towards the metrics to get better scores and that the metrics are only as good as the individual evaluating and interpreting them.

According to Delajoux et al. [LML⁺06], goal-setting, self-assessment and monitoring of the achieved progress are three techniques to motivate a user to change his behavior towards his goals. The last two are already implemented with Happy Coder while the goal-setting is an important feature we plan for a future implementation. The above-mentioned drawbacks and limitations from above could be solved in future work. Many approaches in the area of software engineering and other aspects of a person's life as well as our studies underline the need for a tool like Happy Coder. If a user decides to use Happy Coder, he gets a great tool to monitor his achieved work and to self-assess his productivity. After an initial set-up Happy Coder works in the background without the need to engage. The collected data is visualized with understandable, aesthetic representations and contains trending information to give the opportunity for self-reflection. Those representations might have a positive reinforcement on a developer's motivation and might let him integrate Happy Coder into his everyday life. The possibility to expand Happy Coder with other trackers enables the developer to measure what is important to him. Therefore, Happy Coder might fit every developer's needs and interests for metrics.

7.2 Future Work

There is a number of other features which could be implemented to extend the Happy Coder prototype to improve its usability and functionality. As previously mentioned, the plug-in was developed to be extended by other modules (i.e. trackers). This extensibility has to be refined on the server and the web client to enable others to easily expand the metrics that are measured and visualized. In Chapter 3.3 a multitude of metrics that could be measured are described. Among them is the idea to include other task management systems like Microsoft Outlook or Google Tasks and to implement the change sets tracker as described in Chapter 4.2.6. One might also include some human sensing tools to add data like brain waves or eye movement patterns, to include measures of concentration assessments or means of interruption to improve a developer's efficiency. To overcome possible privacy concerns one might add the possibility to see the visualizations locally and avoid the need to upload the data to the server.

Most of the activity sensing tools described in the related work (see Chapter 2.1) provide the functionality to set a daily goal (e.g. 7000 steps) and see the progress towards the goal (e.g. Fit-bit flower, Nike+ Fuel level indicator). Different studies indicate the motivating effect of setting a goal and seeing the progress [LML⁺06,FM12]. We therefore intend to enhance Happy Coder with a similar feature to add the possibility to forecast today's work by setting goals in form of tasks. With this feature, the developer could use Happy Coder not only as a retrospection but also as a planning tool.

Another interesting feature to cheer a developer on could be the creation of a game feeling. Using trophies or rewards for achieving certain goals (e.g. successfully solve a bug or accurately plan the work day) could improve a developer's motivation to achieve the goal. Moreover, the dimension could be expanded to include other team members. This could be either accomplished by competing against the others or by creating teams which compete against other teams. Rewarding group performance could be a powerful method to motivate a developer to work more efficiently and to incentivize the team to achieve ambitious goals [TFAG06]. Again, this might cause some privacy issues or demotivate a developer even more after an unproductive day. This might be an interesting point to determine with a future study.

In the future, we intend to conduct a user study with multiple developers to find out if Happy Coder measures the right combination of metrics and offers enough flexibility for each developer. Furthermore, we want to assess if a developer can evaluate his productivity with the retrospective analysis offered by Happy Coder and if this makes him happier and motivates him to work. Additionally, we want to ascertain that the visualizations show useful information and to find out what developers think of the Chernoff faces approach. Finally, the ease of use of the plug-in should be assessed. This could be achieved by letting developers use Happy Coder while they perform their usual work. To determine if Happy Coder has an influence on a developer's motivation, it could be interesting to ask him for his mood before showing him the retrospection of his work day. One could imagine the presentation of several faces where a user can chose the face that suits his mood best. The faces could look similar to the poly-metric faces approach (resembling Homer Simpson), see Figure 7.1. Afterwards, the developer can reflect his past work day with the help of the Happy Coder Chernoff face, the overview visualizations and all the detail pages. Before the developer logs out, he gets again asked about his mood (using the same five faces from Figure 7.1. The change could indicate if the data visualized by Happy Coder makes a developer happier. Other findings could be made by interviewing the developer after the installation, configuration and usage of Happy Coder.

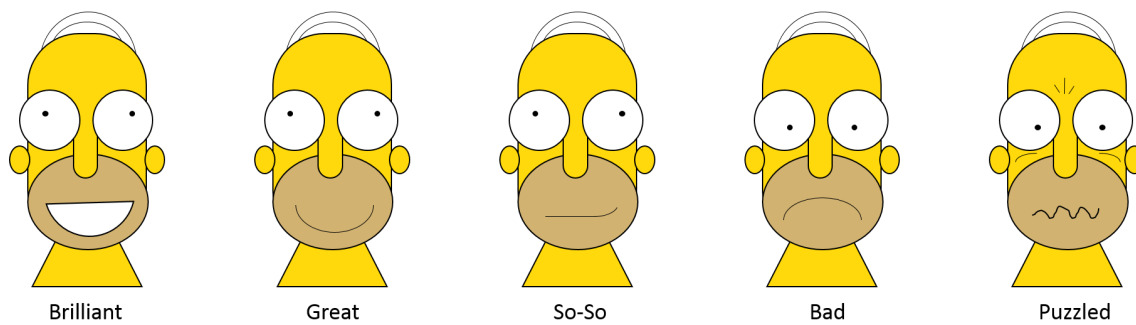


Figure 7.1: Ask the user for his mood after his work day.

As Happy Coder collects a lot of important data of a developer's work behavior, one could imagine a new project, Happy Analyst, to improve a developer's productivity by pointing out tips and suggestions. For example, a developer could be warned if he gets disturbed frequently. It could be suggested to move to a quieter place or to take a break if he is not able to concentrate anymore (to reduce the risk of writing erroneous code). The lessons learnt while working on this thesis revealed the need of an API to gather common metrics from a developer's IDE. This could be another interesting task for a future project.

Conclusions

While there are a lot of studies about supporting a developer in a single domain of his work, only little research has been performed on how to integrate the available data and focus on providing a retrospection of a developer's work day. Our Happy Coder prototype contributes to overcome this issue. Our approach consists of a monitoring component to collect the relevant data of a developer's work and a server component to make the data available with web services. Finally, we implemented a multi-platform website which provides a front-end with consolidated data analysis, visualizations and representations of the collected data.

The usage study and the two small productivity studies revealed that developers assess their productivity based on a personal evaluation of their work day. Therefore, it is not possible to measure productivity with one single metric. Useful information could be provided to the developer to get a quick overview of his past work day, by visualizing a combination of multiple metrics within one representation applying the Chernoff faces approach.

For the future we plan to extend Happy Coder with other useful metrics and a feature to define goals to see the progress towards those objectives. In addition, we intend to conduct a user study to assess if the use of Happy Coder enables the developer to increase the awareness of his work and helps him to reflect the work day. Moreover, we want to find out if our prototype leads to a higher level of motivation of the developer towards achieving personal and software project goals.

Tools and Environments

In order to implement the three components, previously described in Chapter 4, multiple tools, third-party libraries and frameworks were required.

Developer Monitoring The Eclipse plug-in was developed using Java in Eclipse (Juno). We also used a couple of third party libraries like JUnit (4.10) and Hamcrest (1.3) for Unit tests, SQLite-JDBC (3.7) to access the SQLite databases from other plug-ins, XStream (1.4) to serialize to XML files, and others. In the following table, you see the tools and frameworks we used:

Tool	Description	Version	Author(s)	Source
Google Calendar API	Used to access a user's appointments from the Google calendar.	3.0	Google	https://developers.google.com/google-apps/calendar
Mylyn	Used to get the work items a developer worked on and the change sets he committed.	3.8.2	Steffen Pingel, et al.	http://www.eclipse.org/mylyn
Rabbit	Used to get the debug time, Java elements a developer edited or selected and work items a developer worked on.	1.2.1	Lae Chen	http://code.google.com/p/rabbit-eclipse
Reverb	Used to get the number of code relevant and irrelevant web searches.	2012	Gail Murphy, Nick Sawadsky	http://code.google.com/p/reverb-bookmarks
Standard Widget Toolkit (SWT)	Used to create the single status view of the Eclipse plug-in.	4.2.1	Eclipse Foundation	http://www.eclipse.org/swt
Windows Activity Monitor (Wamon)	Used to track the time a user spent in each program.	1.3	Archae	http://code.google.com/p/wamon

Table A.1: Tools and frameworks used to monitor developers.

Server The server was implemented using PHP (5.2) as the programming language and MySQL as the query language in the database. The following tools currently run on the server:

Tool	Description	Version	Author(s)	Source
Apache	The http web server where Happy Coder runs.	2.4	Apache Software Foundation	http://www.apache.org
MySQL	The database to store the data.	5.0	Oracle	http://www.mysql.com
phpMyAdmin	A useful tool to administer the database.	3.5.3	phpMyAdmin development team	http://www.phpmyadmin.net

Table A.2: Tools used to run the server.

Web Client To implement the web client, we needed PHP (5.2), HTML (5.0), SVG (1.1), CSS (3.0), Ajax and JavaScript (1.8). We used the following frameworks to visualize the data collected by the developer monitor:

Tool	Description	Version	Author(s)	Source
CanvasXpress	JavaScript library used for visualizations.	7.0	Isaac Neuhaus	http://canvasxpress.org
jQuery	JavaScript library that simplifies HTML traversing and other HTML manipulations.	1.8.3	jQuery Foundation	http://jquery.com
jQuery mobile	Framework to create multi-platform websites.	1.1	jQuery Foundation	http://jquerymobile.com

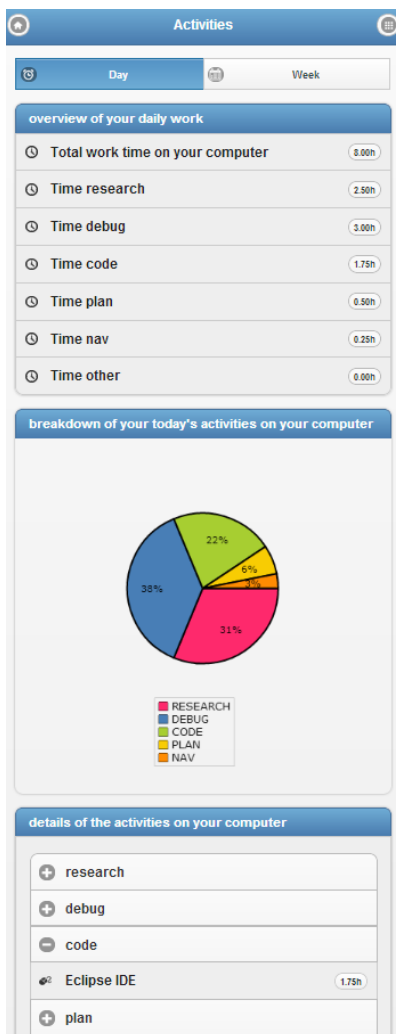
Table A.3: Frameworks used to present and visualize the data on the web client.

Visualization of the Metrics in the Web Client

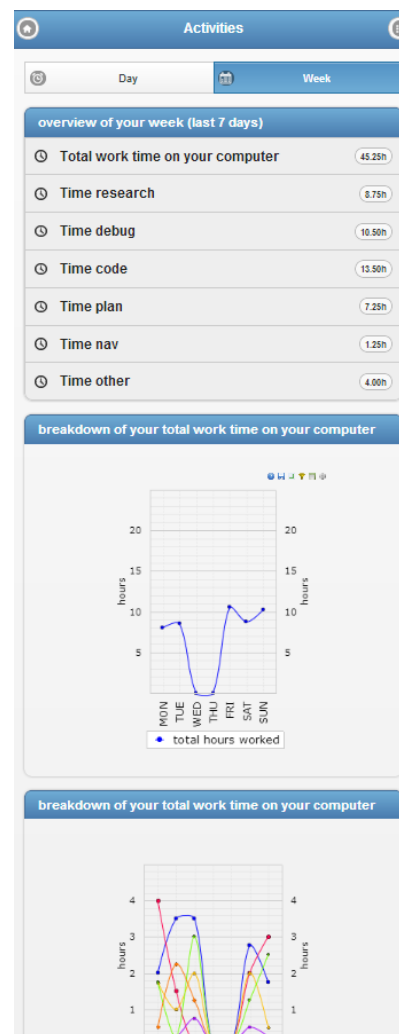
The web client can be accessed on <http://www.andre-meyer.ch/hc-web-client>. Alan's imaginary work day, introduced as a motivating example in Chapter 3, is presented with a demo account. The username is `developer` and the password is `isHappy`.

Overview You find a detailed description and a screenshot of the Happy Coder overview page in Chapter 4.4.

Activities The first box in the activities day page (see Figure B.1(a)) shows a list of the categorized activities a user worked on together with the duration. This data is visualized in a pie chart. The third box contains a detailed list of the programs a developer used during his day. This categorized list is collapsed and can be expanded to see the details. The activities week view (see Figure B.1(b)) contains the same information but for the last seven days. The total duration a developer worked on the computer is visualized in a line chart in the second box. In addition, the second visualization presents the time a developer spent in each activity category compared to the last seven days.



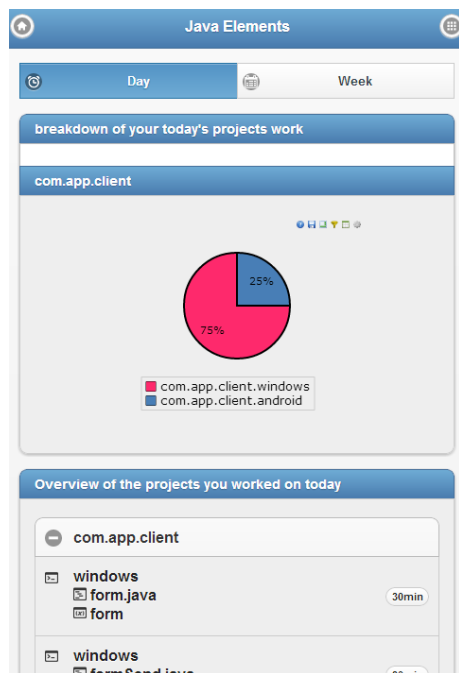
(a) Activities Day



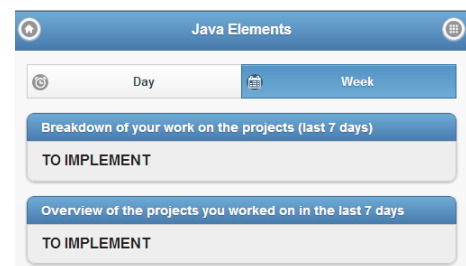
(b) Activities Week

Figure B.1: Web Client Activities visualizations.

Java Elements The java elements day page (see Figure B.2(a)) presents a pie chart for each project, the developer worked on and visualizes each package he selected or edited. The second box consists of a collapsible list presenting all projects, packages, classes and methods a user selected or edited during his work day. The java elements page for the week view has to be implemented in the future.



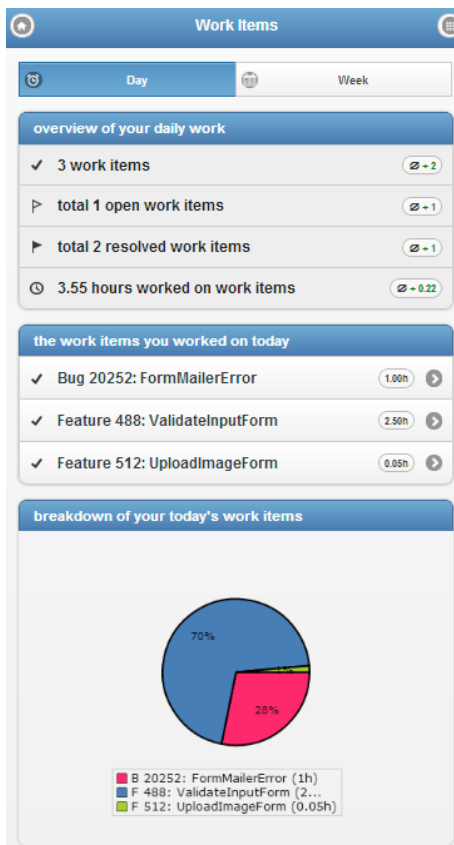
(a) Java Elements Day



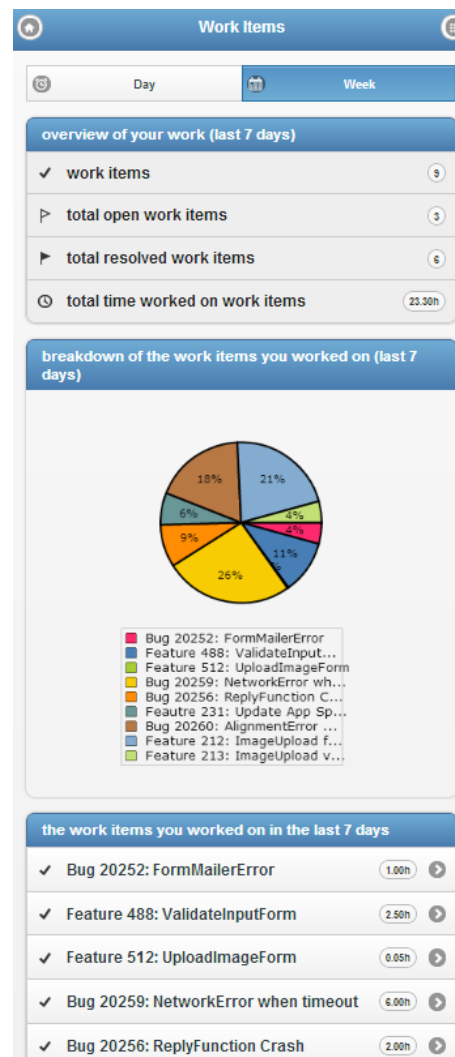
(b) Java Elements Week

Figure B.2: Web Client Java Elements visualizations.

Work Items Via the work items day page (see Figure B.3(a)), the user has access to all the work items, he worked on today. The number of open and resolved work items as well as the time he worked on those work items is presented. The durations are further visualized in a pie chart. In addition, a list contains all the work items the developer worked on. Selecting one of those work items in the list opens a new page with further details. The week view (see Figure B.3(b)) presents the same information for the last seven days.



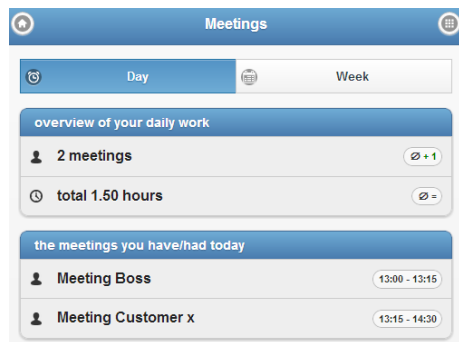
(a) Work Items Day



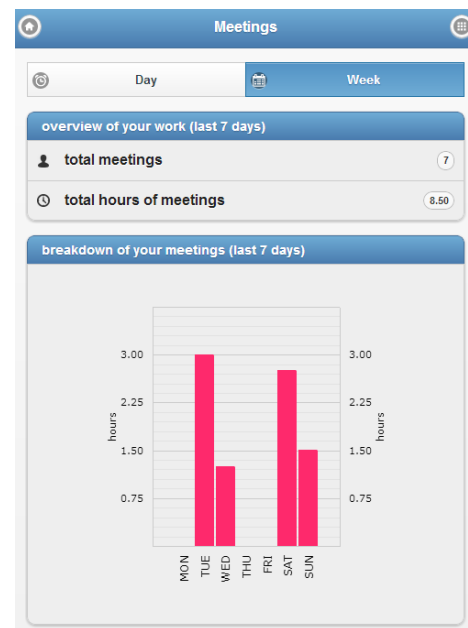
(b) Work Items Week

Figure B.3: Web Client Work Items visualizations.

Meetings The number of meetings, the duration a developer spent in planned meetings and the details of all those meetings are presented in the meetings day page (see Figure B.4(a)). The week view (see Figure B.4(b)) adds a bar chart to easily compare the time a developer spent in meetings with the last seven days.



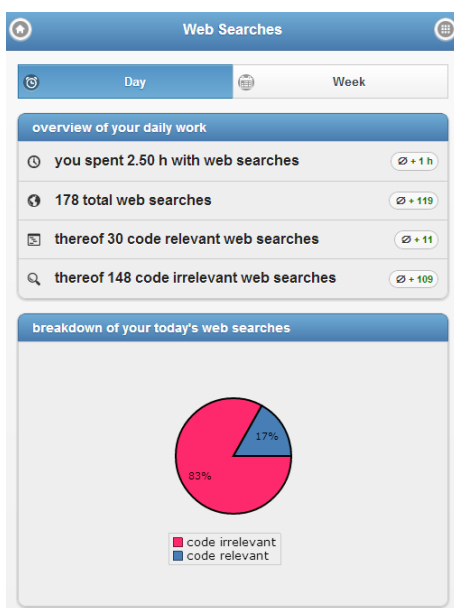
(a) Meetings Day



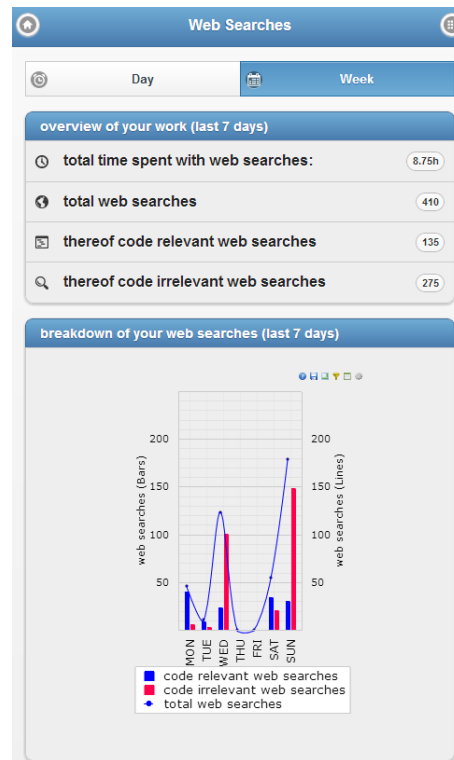
(b) Meetings Week

Figure B.4: Web Client Meetings visualizations.

Web Searches The total time the developer spent in the browser searching for information and the number of code relevant and irrelevant web searches are presented in the web searches day view in form of a list and a pie chart (see Figure B.5(a)). The week view presents the same information for the last seven days and compares the total web searches and the code relevant and irrelevant web searches in a line-bar chart (see Figure B.5(b)).



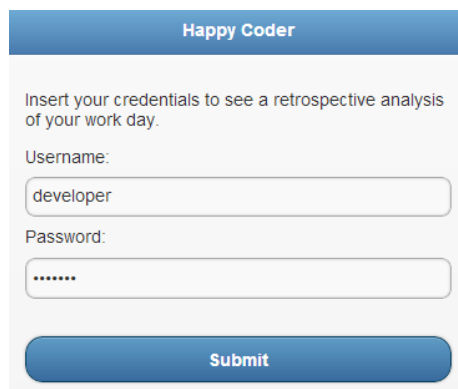
(a) Web Searches Day



(b) Web Searches Week

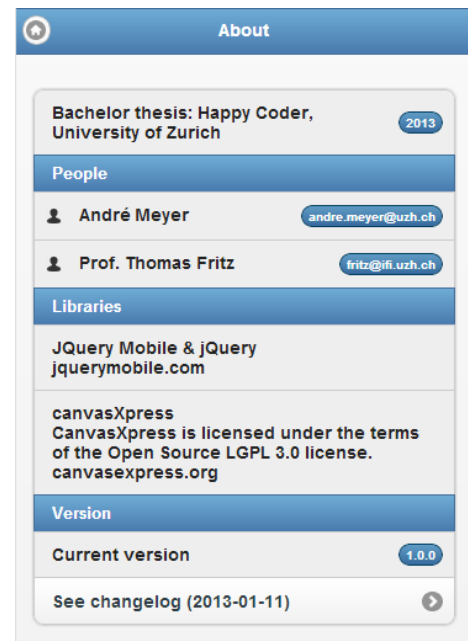
Figure B.5: Web Client Web Searches visualizations.

Others There are a couple of other pages like the log-in page (see Figure B.6(a)) and a help page (no screenshot). The about page (see Figure B.6(b)) contains details about the authors, the frameworks used and information about the latest changes on the web client.



The screenshot shows the 'Happy Coder' log-in page. It has a blue header with the text 'Happy Coder'. Below the header, there is a message: 'Insert your credentials to see a retrospective analysis of your work day.' followed by two input fields: 'Username:' with the value 'developer' and 'Password:' with masked characters '.....'. At the bottom, there is a blue 'Submit' button.

(a) Log-in page



The screenshot shows the 'About' page of the web client. It has a blue header with the text 'About'. Below the header, there is a section titled 'Bachelor thesis: Happy Coder, University of Zurich' with a '2013' badge. This is followed by a 'People' section listing 'André Meyer' (email: andre.meyer@uzh.ch) and 'Prof. Thomas Fritz' (email: fritz@ifi.uzh.ch). Next is a 'Libraries' section listing 'jQuery Mobile & jQuery' (jquerymobile.com) and 'canvasXpress' (licensed under the terms of the Open Source LGPL 3.0 license, canvasexpress.org). This is followed by a 'Version' section showing 'Current version' as '1.0.0' and a link to 'See changelog (2013-01-11)' with a right arrow.

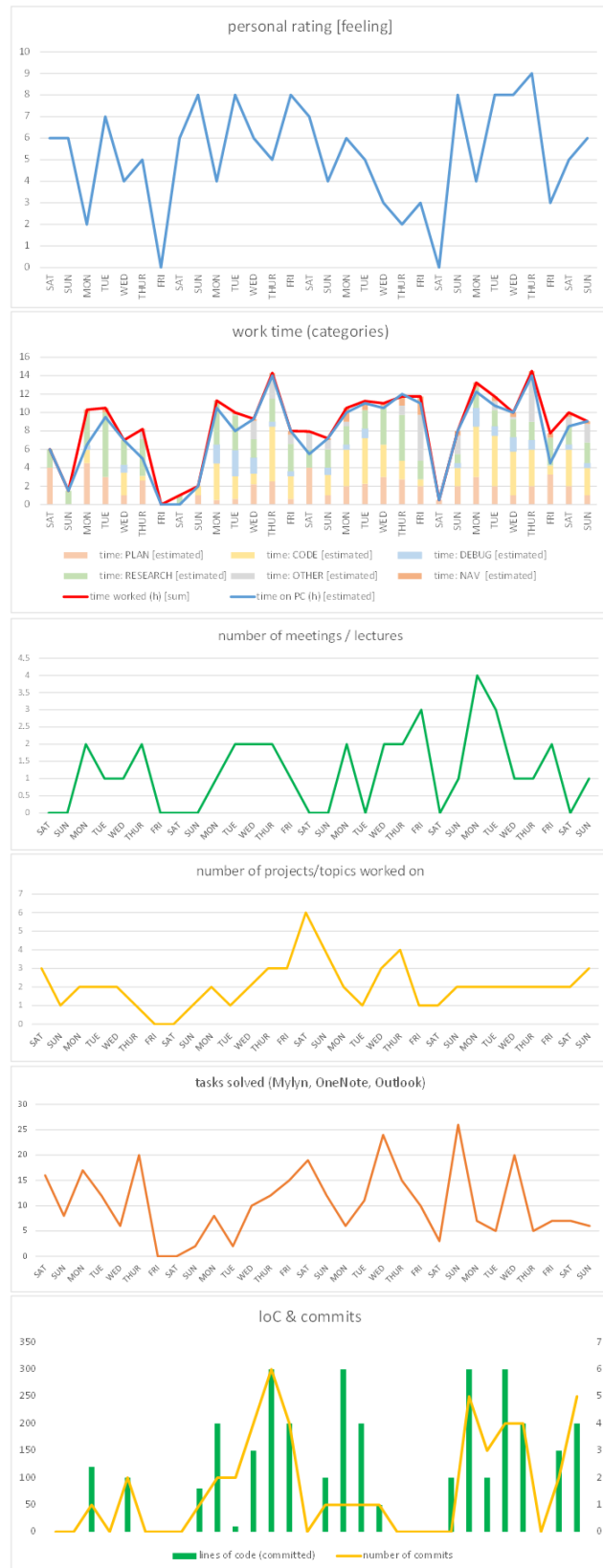
(b) About page

Figure B.6: Web Client excerpt of the log-in and the about page.

Usage Study: Results

Date	22.09.	23.09.	24.09.	25.09.	26.09.	27.09.	28.09.	29.09.	30.09.	01.10.	02.10.	03.10.	04.10.	05.10.	06.10.
	SAT	SUN	MON	TUE	WED	THUR	FRI	SAT	SUN	MON	TUE	WED	THUR	FRI	SAT
personal rating [feeling]	6	0	6	2	7	4	5	0	6	8	4	8	6	5	7
number of meetings / lectures	0	0	0	2	1	1	2	0	0	0	1	2	2	2	0
time worked (h) [sum]	6	1.5	10.3	10.3	10.5	7	8.2	0	1	2	11.3	10	9.3	14.3	8
time on PC (h) [estimated]	6	1.5	6.5	6.5	9.5	7	5	0	0	2	10.5	8	9.3	14	7.95
time: PLAN [estimated]	4	0	4.5	3	3	1	2.6	0	0	2	0.5	0.6	2.2	2.5	0.6
time: CODE [estimated]	0	0	1.5	0	0	2.5	0.5	0	0	1	4	2.5	1.2	6	2.5
time: DEBUG [estimated]	0	0	0.4	0	0	0.8	0	0	0	0	2	2.8	1.7	0.5	0
time: RESEARCH [estimated]	2	1.5	3.5	7	7	2.7	4	0	1	0	4.6	3.8	2	2.5	3
time: OTHER [estimated]	0	0	0	0	0	0	1	0	0	0	0.1	0.1	2	2.5	1
time: NAV [estimated]	0	0	0.4	0.5	0	0	0.1	0	0	0	0.1	0.2	0.2	0.3	0.4
tasks solved [WynN, OneNote, Outlook]	16	8	17	12	12	6	20	0	0	2	8	2	10	12	15
number of projects/topics worked on	3	1	2	2	2	2	1	0	0	1	2	1	2	3	6
number of commits	0	0	1	0	0	2	0	0	0	1	2	2	4	6	4
lines of code (committed)	0	0	120	0	0	100	0	0	0	80	200	10	150	300	200

Date	07.10.	08.10.	09.10.	10.10.	11.10.	12.10.	13.10.	14.10.	15.10.	16.10.	17.10.	18.10.	19.10.	20.10.	21.10.
	SUN	MON	TUE	WED	THUR	FRI	SAT	SUN	MON	TUE	WED	THUR	FRI	SAT	SUN
personal rating [feeling]	4	6	6	5	3	2	3	0	8	4	3	8	9	3	5
number of meetings / lectures	0	2	2	0	2	2	3	0	1	4	3	1	1	2	0
time worked (h) [sum]	7.2	10.5	11.25	11	11	11.75	11.75	0.5	8	13.25	11.75	10	14.5	7.75	10
time on PC (h) [estimated]	7.2	10	11	10.5	10.5	12	11	0.5	8	12.25	10.75	10	14	4.5	9.05
time: PLAN [estimated]	1	2	2.25	3	3	2.75	2	0.5	2	3	2	1	2	3.25	8.5
time: CODE [estimated]	2.25	4	5	3.5	3.5	2	0.75	0	2	5.5	5.5	4.8	4	1	1
time: DEBUG [estimated]	0.75	1	1	0	0	0	0	0	0.5	2	1	1.5	1	0	0.5
time: RESEARCH [estimated]	2	2	2	4	4	5	2	0	1	2.25	1.75	2	2	3	2
time: OTHER [estimated]	1	0.5	0	0.2	1	1	5	0	0	0	1	0.2	5	0	1
time: NAV [estimated]	0.2	1.5	1	0.3	1	1	2	0	0.5	0.5	0.5	0.5	0.5	0.5	0.3
tasks solved [WynN, OneNote, Outlook]	12	6	11	24	15	10	5	3	26	7	5	20	5	7	6
number of projects/topics worked on	4	2	1	3	4	4	1	1	2	2	2	2	2	2	3
number of commits	1	1	1	1	0	0	0	0	0	5	3	4	4	0	2
lines of code (committed)	100	300	200	50	0	0	0	0	100	300	100	300	200	0	150



Productivity Study: Results

Question 1		
Do you think that you have been productive during your last work day? (Very unproductive, unproductive, undecided, productive, very productive)		
Question 2		
How do you assess your productivity at the end of a work day? e.g. number of achieved tasks, etc.		
Answers	Rating	Statement
	4	
	4	progress, solved a task depends on kind of task
		It depends on the current phase of the project, near the end it's about bug fixing or last minute changes the customer wants,
	4	near the beginning its about the ramp up of the project and ensuring that the team know the plan of attack and how this is broken up into sprints, etc..
	4	During my current project the number of bugs found and documented
	4	finished tasks. more productive --> coding less productive --> communication & clarifications
	3	whether I fulfilled the planned tasks or not
	2	
	4	oh my... measure if planned items are resolved financial targets met, ... customer needs met, kundenzufriedenheit >8/10 employees needs met, motivation >8/10 ...
	4	improvement of usability benefit for user to use our product (time saving, reduce complexity)
	2	by checking which tasks selected for the day could be done at the end
	2	less meetings, less discussions about worthless topics
	3	I do nothing.
	3	Number of resolved bugs
	3	
	3	reached what I planned to do
	3	Progress in Relation to planning
	4	personal feeling
Results		
Very unproductive	0	
Unproductive	3	
Undecided	6	
Productive	8	
Very productive	0	
Min	2	
Max	4	
Mean	3.2941	
Median	3	
STDV	0.7487	

Metrics and Productivity Study: Questionnaire

EXPRESS STUDY

WHICH METRICS ARE IMPORTANT TO YOU?

In your opinion, how important is each of the metrics in the following table to be measured and visualized on the Happy Coder multi-platform website? – Please rate it **from 1 (not important) to 5 (very important)**. If you want, you can also add some metrics and there is also space for comments in the end of the second page. Thank you!

Description	1	2	3	4	5
Activities					
How much time did the developer spend in which program? (Categorization: Code, Debug, Plan, Research, Navigate, Other)					
Other:					
Work Items					
Number of work items a developer worked on					
How long a developer was working on a certain work item					
Number of work items a developer solved/edited/created/deleted					
Other:					
Java Elements					
Time a developer spent on which project					
Time a developer spent on which package (java)					
Time a developer spent on which class					
Time a developer spent on which method					
Number of commands a developer executed (copy, paste, save, open, close, step-into (debugging), etc.)					
Other:					

HAPPY CODER | BACHELOR THESIS

Change sets					
Number of commits a developer made					
Lines of Code (LoC) a developer submitted					
Lines of Code (LoC) a developer added/removed (Code Churn)					
Number of classes/methods/variables a developer added/removed/edited					
Number of code elements a developer looked at (selects/edits)					
Other:					
Web searches					
Number of web searches a developer made					
Number of code related / code unrelated web searches a developer made					
Other:					
Meetings					
Number of meetings a developer had					
Duration and topic of each of these meetings					
Other:					

How do you assess your productivity at the end of a work day?

Comments, Suggestions, etc.

Thank you!

Metrics and Productivity Study: Results

Productivity	<p>my productivity = solved issues/work items/tasks + effort spent on classes & bugs - "distractions" by web & chat</p> <p>"work unites" completed (as planned in the morning)</p> <p>targets met for teaching</p> <p>number of features implemented (in relation to estimated effort)</p> <p>number of bugs fixed (in relation to estimated effort)</p> <p>number of implemented work items + a lot of soft-factors</p> <p>if I reached my day-goals. And I'm super happy if I was able to do even more</p> <p>if you feel you accomplished something - then it was a good day</p> <p>number of implemented features</p> <p>if I know that I have achieved something important.</p> <p>doesn't matter if it is the target I aimed for at the beginning of the day.</p> <p>the amount of tasks I complete. By tasks I mean issue/feature request of an issue tracker, or tasks set by myself</p> <p>if I made progress on my tasks that I wrote down for the day</p> <p>if I had productive meetings (in case there were lots of meetings and little time to work)</p> <p>usually I have a plan of what problems to solve during a day/week.</p> <p>I normally measure the number of problems solved and number of more problems found</p>
Feedback	<p>most of these measurements seem to be quantitative. What about qualitative measurements?</p> <p>use code quality to assess productivity</p>

Contents of the CD-ROM

- **Zusfsg.txt**
German version of the abstract of this thesis..
- **Abstract.txt**
English version of the abstract of this thesis
- **Bachelorarbeit.pdf**
Copy of this thesis.
- **DeveloperMonitoring.zip**
Eclipse projects and Javadoc files containing the source code of the plug-in described in this thesis. Moreover, the necessary plug-ins and an installation guide.
- **Server.zip**
PHP files containing the source code of the server part described in this thesis.
- **WebClient.zip**
PHP, HTML, CSS and JavaScript files containing the source code of the web-client (multi-platform website) described in this thesis.

Bibliography

- [AL03] Donald Anselmo and Henry Ledgard. Measuring productivity in the software industry. *Commun. ACM*, 46(11):121–125, November 2003.
- [Car06] David N. Card. The Challenge of Productivity Measurements. In *Pacific Northwest Software Quality Conference*, 2006.
- [CMT⁺08] Sunny Consolvo, David W. McDonald, Tammy Toscos, Mike Y. Chen, Jon Froehlich, Beverly Harrison, Predrag Klasnja, Anthony LaMarca, Louis LeGrand, Ryan Libby, Ian Smith, and James A. Landay. Activity sensing in the wild: a field trial of ubifit garden. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1797–1806, New York, NY, USA, 2008. ACM.
- [Dun88] Anne S. Duncan. Software development productivity tools and metrics. In *Software Engineering, 1988., Proceedings of the 10th International Conference on*, pages 41–48, April 1988.
- [FM12] Thomas Fritz and Gail C. Murphy. Get Steppin'! What Motivates the Use of Activity Sensors? *Submitted to CHI'13*, 2012.
- [HSB⁺08] Tracy Hall, Helen Sharp, Sarah Beecham, Nathan Baddoo, and Hugh Robinson. What Do We Know about Developer Motivation? *Software, IEEE*, 25(4):92–94, 2008.
- [HWY09] Tu Honglei, Sun Wei, and Zhang Yanan. The Research on Software Metrics and Software Complexity Metrics. In *Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum on*, volume 1, pages 131–136, 2009.
- [Jon94] Capers Jones. Software metrics: good, bad and missing. *Computer*, 27(9):98–100, 1994.
- [KHB06] Iftikhar Ahmed Khan, Rob M. Hierons, and Willem-Paul Brinkman. Programmer's mood and their performance. In *Proceedings of the 13th European conference on Cognitive ergonomics: trust and control in complex socio-technical systems*, ECCE '06, pages 123–124, New York, NY, USA, 2006. ACM.
- [Lam84] Geoffrey N. Lambert. A Comparative Study of System Response Time on Program Developer Productivity. *IBM Systems Journal*, 23(1):36–43, 1984.
- [LML⁺06] James Lin, Lena Mamykina, Silvia Lindtner, Gregory Delajoux, and Henry Strub. Fish'n'Steps: Encouraging Physical Activity with an Interactive Computer Game. In Paul Dourish and Adrian Friday, editors, *UbiComp 2006: Ubiquitous Computing*, volume 4206 of *Lecture Notes in Computer Science*, chapter 16, pages 261–278. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2006.

- [LXC12] Dapeng Liu, Shaochun Xu, and Zengdi Cui. Programmer's Performance with the Keystroke as an Indicator: A Further Study. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 577–583, 2012.
- [MER00] Christopher J. Morris, David S. Ebert, and Penny Rheingans. An experimental analysis of the effectiveness of features in chernoff faces. In *28th AIPR Workshop: 3D Visualization for Data Exploration and Decision Making, Proceedings of SPIE*, pages 12–17, 2000.
- [PSHH04] Allen Parrish, Randy Smith, David Hale, and Joanne Hale. A Field Study of Developer Pairs: Productivity Impacts and Implications. *IEEE Softw.*, 21(5):76–79, September 2004.
- [Sim12] The Mail Online. D'oh! Divers find buried fish in the seabed that looks just like Homer Simpson. <http://www.dailymail.co.uk/news/article-2120550/Homer-Simpson-Stargazer-fish-buried-seabed-divers-Lembeh-Strait.html>, image accessed in December 2012.
- [SMJ12] Nicholas Sawadsky, Gail C. Murphy, and Rahul Jiresal. Reverb: Recommending Code-related Web Pages. *to appear in ICSE'13*, 2012.
- [SSP10] Rien Sach, Helen Sharp, and Marian Petre. Continued involvement in software development: motivational factors. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '10*, pages 44:1–44:4, New York, NY, USA, 2010. ACM.
- [SSP11] Rien Sach, Helen Sharp, and Marian Petre. Software Engineers' Perceptions of Factors in Motivation: The Work, People, Obstacles. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 368–371, 2011.
- [TFAG06] Tammy Toscos, Anne Faber, Shunying An, and Mona P. Gandhi. Chick clique: persuasive technology to motivate teenage girls to exercise. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1873–1878, New York, NY, USA, 2006. ACM.
- [Whi97] Ken Whitaker. Motivating and keeping software developers. *Computer*, 30(1):126–128, January 1997.
- [Wol12] Gary Wolf. The Data-Driven Life. <http://www.nytimes.com/2010/05/02/magazine/02self-measurement-t.html>, accessed in October 2012.