

Bachelor Thesis

June 7, 2013

# Project Awareness and Software Quality

A Customized Visualization for Software Project  
Awareness Based on Stakeholder Roles

**Michael Kündig**

of Wetzikon, Switzerland (09-737-552)

**supervised by**

Prof. Dr. Harald C. Gall  
Sebastian Müller



University of  
Zurich<sup>UZH</sup>



software evolution & architecture lab



Bachelor Thesis

---

# Project Awareness and Software Quality

A Customized Visualization for Software Project  
Awareness Based on Stakeholder Roles

Michael Kündig



University of  
Zurich<sup>UZH</sup>



**Bachelor Thesis**

**Author:** Michael Kündig, michael.kuendig@uzh.ch

**Project period:** 10.12.2012 - 10.06.2013

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

---

# Acknowledgements

I would like to thank Prof. Harald Gall for giving me the opportunity to write this thesis at the software evolution and architecture lab. Many thanks also go to Sebastian Müller for his inputs and valuable assistance. Moreover, I would like to thank Siddhartha Arora for reviewing this thesis and everyone who took part in the evaluation.



---

# Abstract

Being aware of a project's status and its software quality is an important factor in the decision making process of a project manager. There are a number of existing tools that are designed to support and analyze software projects, but most of them focus on development tasks and fail to provide a high level project overview.

The goal of this thesis is to assess the information needs of a project manager focused on software development and create a mobile prototype application that is able to assist him in his decision making process. The application that is developed in connection with this thesis is built on the *Android* platform and focuses on a high level project overview. On the one hand, it concentrates on project awareness and makes use of a collaboration tool called *Rational Team Concert*. On the other hand, a service developed by the software evolution & architecture lab at the University of Zurich is used to receive software metrics and give an impression about the software quality of a project. A formative evaluation of the application concludes this thesis.





---

# Zusammenfassung

Den Status und die Software Qualität eines Projektes zu kennen, ist ein wichtiger Faktor im Entscheidungsprozess eines Projekt Managers. Es existieren bereits mehrere Applikationen, die darauf abzielen, Software Projekte zu unterstützen. Da diese Programme allerdings hauptsächlich Arbeiten von Entwicklern unterstützen, liefern sie ein mangelhaftes Gesamtbild eines Software Projektes.

Das Ziel dieser Arbeit ist, den Informationsbedarf von Projekt Managern zu bemessen und eine Prototyp Applikation für Mobilgeräte zu erstellen. Diese soll fähig sein, den Projekt Manager in seinen Entscheidungsprozessen zu unterstützen. Die Applikation, die in Verbindung mit dieser Arbeit erstellt wurde, wurde auf der *Android* Plattform entwickelt und liefert einen Gesamtüberblick eines Projektes. Einerseits fördert die Applikation das Wahrnehmen des Projektstatus und nimmt dabei das Kollaborationstool *Rational Team Concert* in Anspruch. Andererseits liefert ein Service, welcher am software evolution & architecture lab der Universität Zürich entwickelt wurde, Software Metriken. Diese werden genutzt, um die Software Qualität eines Projektes aufzuzeigen. Eine formative Evaluation der Applikation schliesst die Arbeit ab.



---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Goal . . . . .	1
1.3	Thesis Outline . . . . .	2
<b>2</b>	<b>Related Work</b>	<b>3</b>
2.1	Analytics for Software Development . . . . .	3
2.1.1	Reasons for Analytics in Software Development . . . . .	3
2.1.2	Project Management . . . . .	4
2.1.3	Goals of Software Development Analytics . . . . .	4
2.2	Software Metrics . . . . .	5
2.3	Software Project Awareness . . . . .	6
<b>3</b>	<b>Background</b>	<b>7</b>
3.1	The Overview Pyramid . . . . .	7
3.1.1	Thresholds . . . . .	7
3.1.2	Structure . . . . .	8
3.2	SOFAS Metrics Services . . . . .	10
3.2.1	Service Input and Output . . . . .	10
3.2.2	Interaction with the Service . . . . .	11
3.3	IBM Rational Team Concert . . . . .	11
3.3.1	Access to the Server . . . . .	11
3.3.2	Usage of Dashboard and Feeds . . . . .	12
3.4	Tablet Development . . . . .	12
3.4.1	Technology Evaluation Process . . . . .	13
3.4.2	Android Tablet Development . . . . .	14
<b>4</b>	<b>Project Status Awareness and Software Quality Application</b>	<b>15</b>
4.1	Stakeholder Role . . . . .	15
4.2	Dependencies . . . . .	16
4.2.1	Rational Team Concert Server . . . . .	16
4.2.2	SOFAS Metrics Service . . . . .	16
4.3	User Interface . . . . .	16
4.3.1	Project Overview . . . . .	16
4.3.2	Overview Pyramid . . . . .	17
4.3.3	Method and Class Metrics . . . . .	18
4.3.4	Work Item Queries . . . . .	20

4.3.5	Team . . . . .	22
4.3.6	E-Mail Notification . . . . .	22
4.4	Used Technologies and Frameworks . . . . .	23
4.4.1	AChartEngine . . . . .	23
4.4.2	QDox . . . . .	23
4.5	Use Case . . . . .	24
4.5.1	Detecting a critical Method . . . . .	24
4.5.2	Sending E-mail Notification . . . . .	24
4.5.3	Refactoring . . . . .	25
4.5.4	Keeping Track of the Work Item Status . . . . .	25
4.5.5	Limitation . . . . .	25
<b>5</b>	<b>Implementation Details</b>	<b>27</b>
5.1	Internal Structure . . . . .	27
5.1.1	Receiver Classes . . . . .	27
5.1.2	Fragments . . . . .	28
5.1.3	Data Container Classes . . . . .	28
5.1.4	Data Management Classes . . . . .	29
5.2	Receiving Data from the Metrics Services . . . . .	29
5.2.1	SPARQL Query . . . . .	29
5.2.2	Response Receiving . . . . .	30
5.2.3	Response Parsing . . . . .	31
5.3	Connect RTC with the Metrics Service . . . . .	31
5.3.1	Finding a Change Set by the Name of a Java class . . . . .	31
<b>6</b>	<b>Evaluation</b>	<b>33</b>
6.1	Setup . . . . .	33
6.2	Results . . . . .	34
6.2.1	Task Success . . . . .	34
6.2.2	Usability Rating . . . . .	36
6.2.3	Open Questions . . . . .	37
6.2.4	Suggestions for Improvement . . . . .	37
6.3	Limitations . . . . .	38
<b>7</b>	<b>Conclusion</b>	<b>39</b>
7.1	Summary . . . . .	39
7.2	Future Work . . . . .	39
<b>A</b>	<b>Evaluation Form</b>	<b>41</b>
<b>B</b>	<b>Content of the CD-ROM</b>	<b>49</b>

## List of Figures

3.1	Statistical thresholds of 45 Java and 37 C++ systems [LM06]	8
3.2	Example of the Overview Pyramid characterizing the open source project <i>ArgoUML</i> [LM06]	10
3.3	Screenshot of Rational Team Concert's Eclipse plugin	12
3.4	Screenshot of a dashboard in Rational Team Concert's web client	13
3.5	Example of Fragment usage in Android [and]	14
4.1	Screenshot of the project overview in the application	17
4.2	Screenshot of the Overview Pyramid in the application	18
4.3	Screenshot of the method metrics part in the application	19
4.4	Screenshot of the query definition part in Rational Team Concert	20
4.5	Screenshot of the work item queries in the application	21
4.6	Screenshot of a work item in the application	21
4.7	Screenshot of a user in the application	22
4.8	Screenshot of the e-mail notification user interface in the application	23
4.9	Screenshot of an automatically created e-mail template	23
4.10	Screenshot of the methods with a critical high nesting value	24
5.1	Flowchart of a user input scenario	27
5.2	The application's fragment structure	29
6.1	Bar chart showing the success rate of each task	34
6.2	Line chart showing the difficulty of each task	35
6.3	Usability rating of the application	36

## List of Listings

3.1	Sample output of the SOFAS Object Oriented Metrics Service	10
5.1	Example XML file that defines a user interface	28
5.2	Example SPARQL query of the average method weight metric	30
5.3	Implementation of communication with the metrics service	30
5.4	Parsing the content of a response from the metrics services	31
5.5	Finding the latest change set which is related to a given Java class	32



# Introduction

## 1.1 Motivation

Despite the fact that software development is an activity that is considered to be highly measurable and analyzable, it remains difficult to predict and has an overall high rate of project failure [EK08]. One reason for software project failures is that decision making is still primarily based on past experience and intuition of project managers, as assistant tools fail to deliver useful information or are too difficult to interpret [BZ11].

The task of a project manager, who is the main decision maker in a software project, consists of different tasks, *i.e.* communicating with stakeholders, staying aware of the project's status, or task planning. To execute his tasks he is dependent on information from assistant tools that measure and analyze these areas. There are a number of existing tools that are designed to support the software development process, though most of them focus on low level tasks of developers and fail to provide a higher level overview for project managers. In addition, none of the existing tools are designed for mobile devices, but rather for desktop computers.

Sales of desktop computers are declining and predicted to decline even more in the future [gar]. In contrast, the sales quantity of mobile devices has increased heavily and is predicted to do so moving on. In this context, the preconditions were ideal to exploit the advantages of a mobile device application that assists the decision making tasks of a project manager.

## 1.2 Goal

The goal of this thesis is to assess the information needs of a project manager that focuses on software development and create a prototype application for mobile devices that is able to assist him in his decision making process. The application focuses on project awareness by providing an overview over current tasks and activities of team members. Additionally, software metrics are used to give an impression about the software quality of the whole project and indicate critical code areas.

In order to achieve these goals, the information needs of a project manager, as well as existing tools that help to foster awareness and analyze source code, are analyzed. Thereupon, a concept is created how to connect these tools to use it in combination with the prototype mobile application. Finally, the prototype application is evaluated by conducting a formative study.

## 1.3 Thesis Outline

Chapter 1 gives a short introduction and defines the goals of this thesis. In Chapter 2, related work in the field of software project awareness and analytics as well as software metrics is discussed. Chapter 3 presents required background information about the visualization of a software system in form of an overview pyramid as well as the tools that were used in connection with the prototype application. The dependencies of the application as well as its user interface is presented in Chapter 4 along with the used technologies and a brief use case. Chapter 5 presents and explains informative implementation details of the prototype application and Chapter 6 covers the findings of the user evaluation. Chapter 7 concludes the thesis with a summary and an outlook on future work.



# Related Work

This Chapter presents related work in the areas of analytics for software development, software metrics, and software project awareness.

## 2.1 Analytics for Software Development

Davenport and Harris describe analytics as *"the extensive use of data, statistical and quantitative analysis, explanatory and predictive models, and fact-based management to drive decisions and actions"* [DH07]. Analytics consists of a set of technologies and processes that use data to help decision makers understand and analyze business processes. The following Section describes the reasons and goals for analytics in the field of software development and who can profit from it.

### 2.1.1 Reasons for Analytics in Software Development

Software development is considered to be an activity that is data rich. Almost every aspect in software development, from code repositories to testing environments, can be measured and analyzed. Despite a high degree of measurability, software projects remain unpredictable and have an increased risk of failure [BZ10]. This observation implies that there may be a disconnect between information that is needed for decision making and the information that is delivered by existing analysis tools [BZ11].

Buse and Zimmermann state that software engineering has many qualities that lend themselves well to a business analytics process [BZ11]:

- **Data-rich:** Software development delivers large amounts of data to analyze. Analytics works best with lots of data to analyze.
- **Labor intensive:** Software engineering is a labor intensive activity, needs a long period of education of engineers and has a short supply of graduates [Sha00]. Analytics enables correct expertise to be leveraged and helps to manage human resources.
- **Time dependent:** Analytics supports time dependent business processes by enabling decision makers to look at the past and predict future schedules.
- **Dependent on consistency and control:** In unusual circumstances analytics can help to make consistent and controlled decisions.

- **Dependent on distributed decision making:** Software development projects, in particular open source projects, have often geographically distributed teams. Analytic tools help to analyze and understand the overall state of projects with distributed teams.
- **Low average success rate:** Areas where failure rates are high have a larger chance to benefit from analytics. Although failure rates of software development projects have decreased in the past years, still every fifth software project gets cancelled [EK08].

## 2.1.2 Project Management

Project managers are the main decision makers in software projects and mainly responsible for the successful outcome of a project. Therefore, project managers in software development should have a vested interest in analytics tools that assist them in their decision making process. The challenge of developing a tool that supports a project manager's work, is to understand his tasks and information needs [LHG10].

A project manager's job includes the monitoring and controlling the work of his team of software developers. A developer mainly focuses on lower level tasks such as coding, performance enhancing or refactoring, whereas a project manager is mostly occupied with higher level tasks. Examples of higher level tasks include: monitoring the direction of the project, allocating resources, cooperating and communicating with stakeholders or fostering the motivation and team spirit [BZ11].

To perform these tasks a project manager requires information from a wide range of sources. In a survey by Buse and Zimmermann 110 employees at Microsoft were interviewed; 57 managers and 53 developers [BZ11]. When asked about the factors which influence their decision making most, managers rated *Data and Metrics* as the most important factor, whereas *Personal experience* was rated as most important by developers. Project managers also rated *Personal experience*, *Customer input*, *Product vision* and *Team planning* as important to very important factors. From this study it can be observed that data mining and analyzing holds crucial information for the decision making task of a project manager. Nevertheless it is important for project managers as well, to survey the current work of their developers, so that they can manage their teams and allocate resources properly.

## 2.1.3 Goals of Software Development Analytics

Software development analytics assists project managers to make decisions by accentuating the information that they need. Project managers could profit from analytics in the following areas [BZ11]:

- **Project monitoring:** Analytics is a powerful means of monitoring the status of a project. Analytics can analyze large amounts of data and help to visualize the contained information in an understandable way.
- **Efficiency improvement:** Analytics helps to detect the efficiency of different working techniques. It can as well discover whether some workflow changes influence the result or the efficiency.
- **Risk management:** Large data collections allow one to create risk models. Analytics can also help to collect qualitative data and deliver output in a coherent way.
- **Anticipation of change:** Analytics models can help managers to detect and forecast trends in data.

- **Evaluation of past decisions:** Consistent decision making based on data is simpler to evaluate than decisions based on intuition. Through data based decision making it is easier to analyze how a result was caused by a decision.

The overarching goal of analytics is to help managers move beyond information and toward insight. Analytics is not about answering the question what happened in the past but rather how did it happen and what is the next best action to take. The complexity of a project manager's work is still not entirely understood. There is a lot of research still to be done until project managers can use analytics tools that come up to their great promises.

## 2.2 Software Metrics

Software metrics allow characterization and measurement of software products and processes. The purpose of software metrics is to evaluate and improve the quality and design of software systems. A lot of metrics have been proposed and validated. There are software metrics that measure the size, the complexity, the quality, the change history, and many other areas of a software engineering project [Mil98].

Fenton and Pfleeger classify software metrics into three categories: process metrics, product metrics, and resource metrics [FP97].

- **Process metrics** measure software related activities, *e.g.* the development process.
- **Product metrics** measure all artifacts that result from a process activity.
- **Resource metrics:** measure entities that are required by a process activity.

Furthermore, with each metric there is a distinction between internal and external attributes. Internal attributes can be measured relating to the product, process, or resource itself. An external attribute can be measured only with the respect how a product, process, or resource relates to its environment.

Another method is to classify software metrics into static and dynamic metrics. On the one hand, static metrics are collected only from the source code of a software project and are usually used to quantify various design or source code complexity aspects. On the other hand, dynamic metrics capture the complexity and the dynamic behaviour of a software system. They are collected from the execution traces of the source code's executable models. To clarify the differences of static and dynamic metrics, one can say that *"static metrics deal with the structural aspects of a software system, whereas runtime (dynamic) metrics also deal with the behavioural aspects of the system"* [KCG10]. The large advantage of static over dynamic metrics is the fact that they are simpler to collect. The collection of dynamic trace data is far more complex and the data is accessible only late in a software development lifecycle [KCG10].

Michel Lanza and Radu Marinescu analyzed the usage of object-oriented metrics in practice [LM06]. In their book, they only deal with static, internal product metrics although they refer to them as design metrics. Lanza and Marinescu state that metrics are used *"for many specific purposes, like quantifying and qualifying the code that has been written, or predicting future development efforts that must be invested into a project."* [LM06]. Their focus lies on the purpose to make software metrics more interpretable. They introduce different approaches that visualize software projects with the aid of software metrics or introduce characteristics that aim to detect problems in the

source code. One of their approaches, the Overview Pyramid, is utilized in this thesis and described in more detail in Section 3.1.

## 2.3 Software Project Awareness

Dourish and Belotti describe awareness as “*an understanding of the activities of others, which provides a context for your own activity*” [DB92]. In a software project, awareness includes being aware of the technical aspects, *e.g.* source code status, but also being informed about the current and upcoming tasks and the team members’ workload. Project awareness allows better coordination and control of work tasks, planning future tasks, and allocating resources. Especially in distributed development environments, where face-to-face meetings and synchronous communication is rare, it is crucial that developers and managers keep track of a project’s status [GPS04].

Project awareness in software development projects is supported by assistant tools. Early awareness assistant tools focused on source code and information that is available from source code. Some of the tools raise awareness by highlighting code fragments that were recently changed. For example, *FASTDash* is a project awareness tool that allows developers to quickly see which team members checked out source files, what files are being viewed and which methods and classes are being changed [BCSR07]. Another assistant tool is *Seesoft* which concentrates on software visualization. It maps each source code line to a thin row and indicates changes by using colors [ESS92].

These tools focus on code changes and current activities in a project. However, “*in large projects, they often fail to provide an overview of the overall status of a project.*” [TS10].

For this reason, more recent attempts to assist project awareness focus on higher level and more work related project insights. Software project awareness tools such as Microsoft’s *Team Foundation Server* and IBM’s *Rational Team Concert* provide a development environment that keep users up to date about current work tasks, bug statuses, code modifications, and the team’s productivity and workload. Section 3.3 describes the *Rational Team Concert* in more detail. Unfortunately, these higher level assistant tools fail to give a detailed overview over code related artifacts.

The prototype application developed in conjunction with this thesis connects two existing tools, IBM’s *Rational Team Concert* and University of Zurich’s *Object Oriented Metrics Service*, to combine source code oriented awareness with work task related awareness.

# Background

In this Chapter all necessary background information that is needed in connection to the prototype application is presented. First, an introduction about the Overview Pyramid which is used in the prototype application is provided. Sections 3.2 and 3.3 present the SOFAS metrics services as well as IBM's *Rational Team Concert*. The last Section gives a brief introduction to tablet application development.

## 3.1 The Overview Pyramid

The Overview Pyramid by Michel Lanza and Radu Marinescu is a visualization of an object-oriented system in its entirety. The goal is to show the most significant measurements about an object-oriented project in a single place. The Overview Pyramid displays and compares size and complexity, coupling, and inheritance metrics. The displayed metric ratios are classified using thresholds that are introduced in the following Section. The Overview Pyramid consists of metric values and ratios. One option to make the metric ratios of the pyramid interpretable is to give thresholds that make them comparable. The following Sections describe how the metric thresholds are defined and how the Overview Pyramid is structured.

### 3.1.1 Thresholds

Software development and design can only benefit from metrics if the metric values can be interpreted and evaluated. Thresholds are a fundamental instrument to analyze and compare software code. A software metric is almost useless in practice if it is not possible to tell whether the calculated value is too high, too low, or just right. Unfortunately, the typical values of most software metrics are still unknown [FBB<sup>+</sup>11]. How is it even possible to determine a threshold? What is the criteria to tell whether a project is small, large, or even too large?

Lanza and Marinescu state that there are two major sources to identify thresholds: statistical information and generally accepted semantics. Thresholds based on statistical measurements are primarily useful for size metrics. For example, a Swiss man has a height of 175.4 cm on average [CKG<sup>+</sup>00]. With this piece of information one can tell that every person over 175.4 cm can be considered tall. The more statistical data there is, *e.g.* the distribution or height compared to origin, the more it is possible to tell about a person's height.

Thresholds based on information that is considered normal are typically also a result of former statistical observations. The difference is that the information or the result is so widely accepted that it implicitly classifies the reference points. For example, if we would measure the number of

meals we consume per day, we would classify 3 as the normality threshold without statistically measuring our behaviour.

## Statistics-Based Thresholds

It is hard to tell beyond which number of code lines a method is considered too large. There are some factors that need to be considered before this question can be answered. In addition, the code language that has to be identified, since it has an influence on the number of code lines, it needs to be defined how the lines will be counted. As soon as a definite method is determined to collect the metrics, data needs to be statistically interpreted. Lanza und Marinescu defined a metric result to be average if its value was between the standard deviation subtracted from the average and the standard deviation accumulated to the average. A result is considered to be high if it exceeds the value of the upper limit and very high if it exceeds the upper limit value multiplied by 1.5. Same applies to the lower margins.

## Meaningful Thresholds

The maximum nesting level of a method tells how many conditional statements are nested within a method. The value 0 illustrates that there is no conditional statement in a method, whereas a value of 1, 2, or 3 displays that there is some nesting but it is still possible to be overlooked. Values over 4 indicate that the method has a deep nesting structure and it is hard to follow the control flow. Hence, there are three thresholds that are not statistics based but they give the desired information about the metric.

The top four metrics in Figure 3.1 show thresholds that were used for the Overview Pyramid in the prototype application.

Metric	Java			C++		
	Low	Average	High	Low	Average	High
CYCLO/Line of code	0.16	0.20	0.24	0.20	0.25	0.30
LOC/Operation	7	10	13	5	10	16
NOM/Class	4	7	10	4	9	15
NOC /Package	6	17	26	3	19	35
CALLS/Operation	2.01	2.62	3.2	1.17	1.58	2
FANOUT /Call	0.56	0.62	0.68	0.20	0.34	0.48
ANDC	0.25	0.41	0.57	0.19	0.28	0.37
AHH	0.09	0.21	0.32	0.05	0.13	0.21

**Figure 3.1:** Statistical thresholds of 45 Java and 37 C++ systems [LM06]

### 3.1.2 Structure

The Overview Pyramid is divided into three sections: size and complexity (yellow), inheritance (green), and coupling (light blue). These sections give an informative insight about how large and complex the system is, to which extent the classes within the system are coupled, and how much inheritance was used. The threshold from Figure 3.1 are used to color the calculated metric ratios

in the pyramid. Dark blue indicates a low, green an average, and red a high value. All ratios are calculated the same: the metric value in the lower row is divided by the value in the upper row.

### Size and Complexity metrics

- **Cyclomatic complexity (CYCLO):** The total number of possible independent paths summed from all methods in the system. Based on McCabe's cyclomatic complexity [McC76].
- **Lines of code (LOC):** The total number of all user-defined code lines. Only the code lines that contain functionality are counted (e.g. lines of code belonging to methods).
- **Number of methods/operations (NOM):** The total number of user-defined methods within the system.
- **Number of classes (NOC):** The overall number of defined classes in the system, not counting library classes.
- **Number of packages (NOP):** The total number of packages that are used to organize the code.

The metric values only reveal how large the project is, whereas the metric ratios show how well structured and how complex it is. The ratios let the beholder quickly know the average size of the methods, classes and packages within the system.

### Inheritance metrics

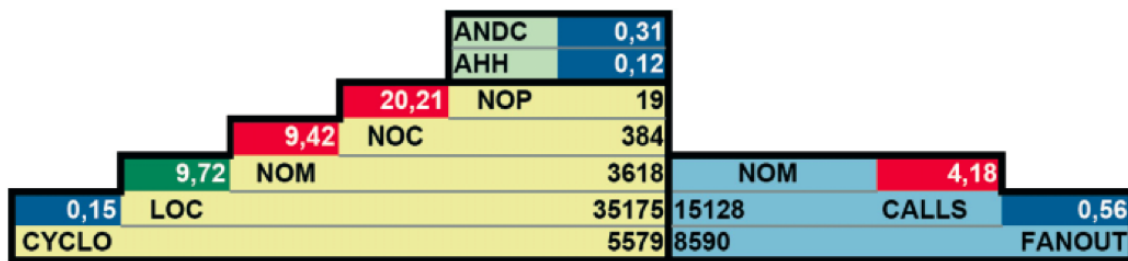
- **Average number of derived classes (ANDC):** The average number of direct subclasses of a class, interfaces do not count. The number of subclasses of every user-defined class is summed up and divided by the number of classes in the system.
- **Average hierarchy height (AHH):** The average height of the inheritance tree of every root class in the system. A class is a root class if it is not derived from any other class belonging to the system. The height of a root classes inheritance tree is zero if it has no subclasses.

These two metrics give an overview how inheritance is used in a system. The ANDC metric shows in how many classes of a system inheritance is used and the AHH tells how deep the inheritance structure is. For example, if the ANDC value is low and the AHH value is high, then there are only few but very deep inheritance structures in the project.

### Coupling metrics

- **Number of called classes (FANOUT):** The sum of all fanout values of every class. The fanout value illustrates how many other classes are called from operations within the class.
- **Number of method/operation calls (CALLS):** The total number of all distinct method invocations in the system. For example, if a method *foo()* is called three times by the method *m1()*, the metric value for the method *foo()* is 1. If *foo()* is called at least once from within each method *m1()*, *m2()*, and *m3()*, the metric value for method *foo()* is 3.

To gain an insight about the coupling amount of a system, the calculated metrics have to be compared with each other. The FANOUT/CALLS ratio is an indicator of how coupling is used beyond classes. A higher value indicates that there are a lot of calls between classes. The CLASS/NOM ratio shows the average number of method calls from within a method.



**Figure 3.2:** Example of the Overview Pyramid characterizing the open source project *ArgouML* [LM06]

## 3.2 SOFAS Metrics Services

The *SOFAS Size & Complexity Metrics Service* and the *SOFAS Object Oriented Metrics Service* are RESTful web services provided and developed by the software evolution & architecture lab (s.e.a.l.) at the University of Zurich. A RESTful web service is a web service implemented using HTTP and the REST (Representational State Transfer) conventions. The metrics services are part of the Software Analysis Services (SOFAS) which wrap already existing software analysis tools exposing their functionalities and data through a web service [sof].

The SOFAS project was started because no convenient way existed to compare the results of different software analysis tools. Existing tools used different meta-models for their input and output representation, therefore making it complicated and time-consuming to compare the results. Additionally, manual installation and configuration of the analysis tools is avoided, due to the fact that the software analysis services are available through a web service [sof].

### 3.2.1 Service Input and Output

Both Metrics Services are similarly constructed. They both accept two types of inputs: raw source code and FAMIX meta-models. FAMIX is a code language independent meta-model, developed by Serge Demeyer, that characterizes the static structure of object-oriented software systems [DTS99]. The output data is described by the Software Evolution Ontologies (SEON). They define and represent the consumed and produced data by the Software Analysis Services [sof]. Their goal is *"to facilitate the implementation of tools that help software engineers to manage software systems over their entire life-cycle"* [seo]. Listing 3.1 shows a sample output of a system's ANDC metric calculated by the *SOFAS Object Oriented Metrics Service*. It is represented in the SEON Code Metrics Ontology.

```
<rdf:Description rdf:about="http://../famixMetrics/analyses/Prototype/ANDC">
  <rdf:type rdf:resource="http://../softwaremetrics.owl#SoftwareDesignMetric"/>
  <rdfs:label>total ANDC</rdfs:label>
  <j.0:hasName>ANDC</j.0:hasName>
  <j.0:hasValue rdf:datatype="http://../XMLSchema#double">0.19</j.0:hasValue>
  <j.0:isMetricOf rdf:resource="http://../famixParser/analyses/Prototype"/>
</rdf:Description>
```

**Listing 3.1:** Sample output of the SOFAS Object Oriented Metrics Service



### 3.2.2 Interaction with the Service

Both, the *SOFAS Size & Complexity Metrics Service* and the *SOFAS Object Oriented Metrics Service*, provide access through a web interface. The source code is uploaded to the *FAMIX meta-model extractor* service in a zipped file. The generated FAMIX meta-model is then used by the metrics service to analyze the source code and calculate all the metrics. As soon as the analyses completed, the output is available in a downloadable RDF Schema file. Additionally, the output is accessible through SPARQL queries that are sent to the REST interfaces of the services.

## 3.3 IBM Rational Team Concert

*Rational Team Concert (RTC)* is a team collaboration tool that was developed by IBM's Rational brand and initially released in 2008. It is part of the IBM Jazz Platform which combines tools for improving collaboration and lifecycle management. *RTC's* features such as task tracking, source control, and agile planning help to improve the software development process [rtc]. In addition, *RTC* provides dashboards and feeds that foster project and team awareness.

A main component in *RTC* is the work item which is used to track the work tasks in a project. A work item may be a task, a defect, a story, or one of many other types that are provided by *RTC*. Usually, a title, description, priority, plan, due date, and user are assigned to a work item. A newly created work item is automatically added to the user's unfinished work section. As soon as the user starts working on an assigned task, he marks the task as his current work item. Once he has finished the task, he commits the changes in form of a change set to the source control. The committed change set has to be associated with a work item which will automatically be marked as resolved and linked to change set.

*Rational Team Concert* also supports agile, waterfall, or hybrid approached planning that is closely connected to the work task management. Furthermore, it has a build management functionality that allows control and traceability of software builds.

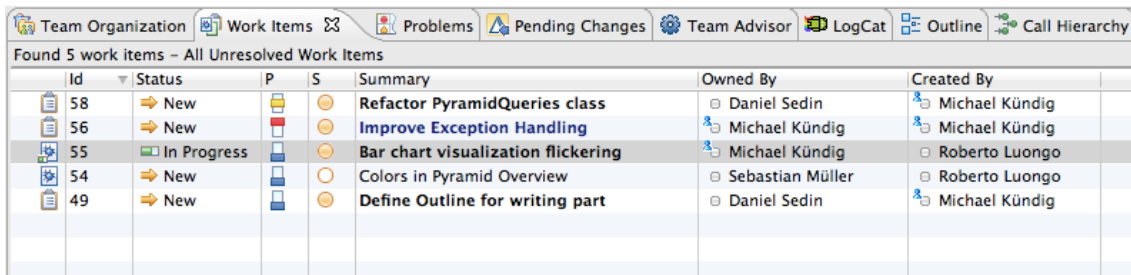
There are several reasons why *Rational Team Concert* was used in connection to the prototype application of this thesis. First of all, it was immediately available and free for up to ten developers which was enough in context of the prototype. It also fulfilled the requirement to be a software awareness tool whose focus lies on a high level project overview. Most importantly, it offered various interfaces that allowed integration in the prototype application.

### 3.3.1 Access to the Server

There are several different ways to interact with the *RTC* server: through an IDE plugin for *Eclipse* and *Visual Studio*, a web client, a command line interface, a Java and REST API. The following three ways of interaction were used in the context of this thesis.

#### Eclipse Integrated Development Environment plugin

The plugin for the *Eclipse* integrated development environment (IDE) is mainly intended for developers. All *RTC* features are embedded into the *Eclipse* workbench, including views, editors and status line trims.



The screenshot shows the Rational Team Concert Eclipse plugin interface. At the top, there is a toolbar with icons for Team Organization, Work Items, Problems, Pending Changes, Team Advisor, LogCat, Outline, and Call Hierarchy. Below the toolbar, a status bar indicates "Found 5 work items - All Unresolved Work Items". The main area displays a table with the following data:

Id	Status	P	S	Summary	Owned By	Created By
58	New			Refactor PyramidQueries class	Daniel Sedin	Michael Kündig
56	New			Improve Exception Handling	Michael Kündig	Michael Kündig
55	In Progress			Bar chart visualization flickering	Michael Kündig	Roberto Luongo
54	New			Colors in Pyramid Overview	Sebastian Müller	Roberto Luongo
49	New			Define Outline for writing part	Daniel Sedin	Michael Kündig

**Figure 3.3:** Screenshot of Rational Team Concert's Eclipse plugin

## REST interface

The IBM Jazz platform which underlies the *Rational Team Concert* is part of the *Open Services for Lifecycle Collaboration (OSLC)* community. The goal of the *OSLC* community is to "standardize the way that software lifecycle tools can share data with one another" [osla]. Therefore, all products belonging to the Jazz Platform have to abide the specification defined by the Open Services for Lifecycle Collaboration. *OSLC* follows the REST architectural pattern [oslb]. This means that all services of the *OSLC* provide a standardized REST interface for communication. Data can be received and sent to the *RTC* server through simple HTTP GET and PUT requests.

## Web client

The web client only requires a web browser to access *RTC*. Every user has its own customizable home screen where dashboards and feeds of interest can be added.

### 3.3.2 Usage of Dashboard and Feeds

In a survey Christoph Treude and Margaret-Anne Storey asked approximately 150 developers and project managers at IBM about their usage of dashboard and feeds in their daily work with *Rational Team Concert* [TS10]. The survey showed that developers and managers customize their dashboards for their own advantage. Whereas developers mainly use feeds to stay aware of current changes of their work items or work items they subscribed to, managers use dashboards and feeds to compare developers and teams. Team and project dashboards are also used to reveal the productivity of different developers and teams. However, managers and developers are aware that dashboards are only as good as the data that they display.

Treude and Storey also discovered that dashboards and feeds are especially used in critical phases of a project. Work items are indicated as "mustfix" before a release to indicate their importance. Dashboard and feeds are then used to observe the status of the critical work items.

Nevertheless, the survey also afforded that about half of the developers do not make use of dashboards and about one third do not use feeds.

## 3.4 Tablet Development

A recent forecast published by the technology research firm Gartner, predicted that by 2017 more tablet computers will be sold than desk-based computers and notebooks combined [gar]. Whereas the number of annually sold desktop and notebook computers is going to decrease, the sales quantity of tablet devices will increase rapidly. This prediction indicates the growing importance

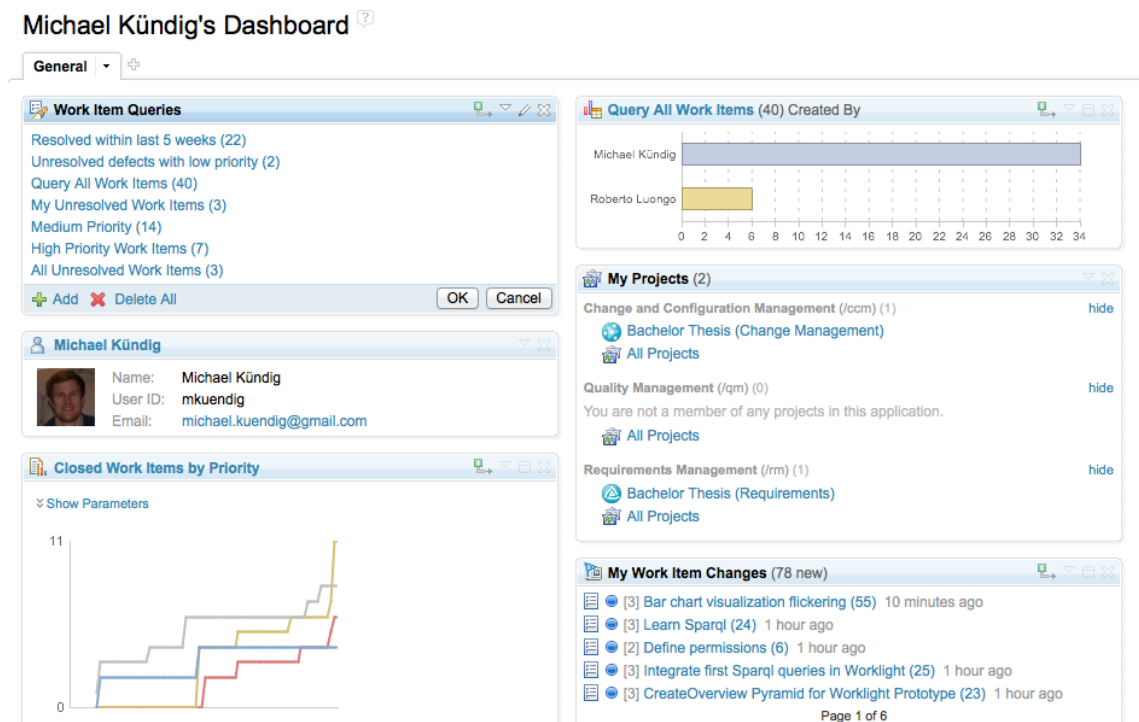


Figure 3.4: Screenshot of a dashboard in Rational Team Concert's web client

of tablet application development.

There are two main types of mobile applications: native and web. Native applications use the mobile platform's SDK which offers the widest range of functionalities and best utilization of device resources. The only disadvantage of native applications is that they only run on the platform they were developed for, whereas web applications run on all platforms with a sufficient mobile browser. The downside of web applications is their dependency on web technologies which offer less functionality and performance compared to native developed applications [CL11].

### 3.4.1 Technology Evaluation Process

In the beginning the goal was to implement a mobile web prototype application for this thesis with IBM *Worklight*, a development environment for mobile applications. The main reason to build a web application was the ability to test and use the application on various platforms and devices. *Worklight* allows one to develop a server-side and a client-side part which communicate through predefined interfaces. Therefore, the possibility existed to develop the client-side mobile application in JavaScript/HTML5 and the server-side application in Java. For that reason, a large piece of the coding part could have been written in Java, which was the preferred language. After about three weeks of testing and developing with *Worklight*, the decision was made to switch from web to native application development. The development environment for mobile web applications did not seem to be technically mature yet. For example, the rendering of web components within the IDE was oftentimes flawed or not working at all, although the code was correct and worked perfectly well in a web browser. Debugging was much more cumbersome as well, since it was only possible through a web browser and not within the development environ-

ment itself.

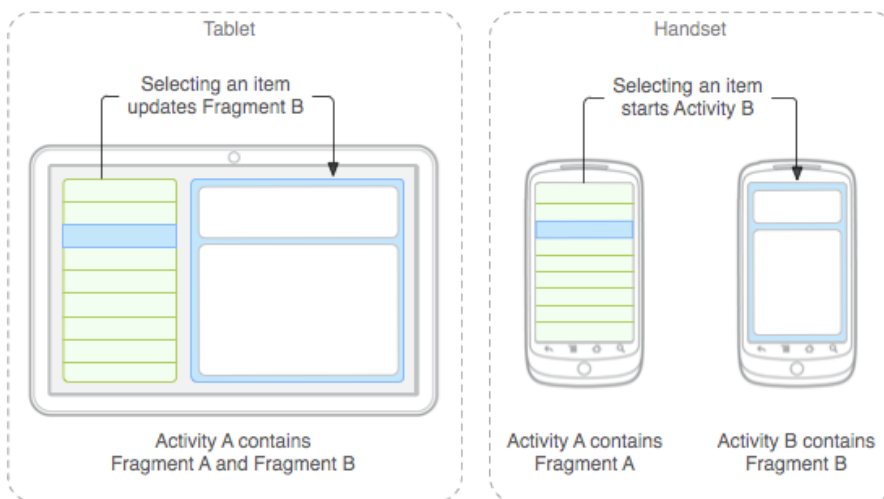
For these reasons a switch was made to native *Android* application development. The choice of *Android* was obvious, as apps in *Android* are developed in Java which was the preferred code language.

### 3.4.2 Android Tablet Development

With *Android 3.0 Honeycomb (API level 11)* Google presented the first and last tablet-only *Android* version. The main architectural change was the introduction of fragments that allowed better code reuse and animated effects when dynamically replacing other fragments from the screen. Since the release of *Android 4.0*, Google does not distinguish between tablet and mobile development anymore. The developer has the possibility to define different user interfaces for different screen sizes. Depending on what device the application is running, it chooses the user interface suitable to the screen size [hon].

#### Design Guidelines

Google proposes to take advantage of extra screen area available on tablets [and]. As poorly developed tablet applications use a stretched layout style, tablet applications sometimes have exceedingly long line lengths or a poor use of white space. Figure 3.5 shows an example how Fragment allows one to build a multi-pane layout to ideally use the whole screen of a device. The size of fonts, buttons and other UI components needs to be optimized for all different screen configurations as well.



**Figure 3.5:** Example of Fragment usage in Android [and]

# Project Status Awareness and Software Quality Application

The following Chapter covers a detailed description of the prototype application. We are going to describe the dependencies and the user interface of the application, as well as the frameworks that were used during the development process. In the last part a use case describes how the functionalities of the prototype application could be utilized.

All the data that is analyzed and displayed by the prototype application was generated in the context of this thesis. The used code metrics are calculated from the source code of the application itself and the work items and users were created in the course of the thesis.

## 4.1 Stakeholder Role

In the beginning of this thesis the information needs of different stakeholders in a software project, such as software testers, developers, project managers or requirements engineers, were analyzed. Thereafter, an application for one specific stakeholder was going to be created.

After studying the different stakeholders and researching about available tools to use in connection with the mobile application, the decision was made to focus on the project manager. The project manager's stakeholder role was selected because the tools to cover his information needs were available right away. Section 2.1.2 covers these findings.

The goal of this application is to provide an assistant tool for project managers. The perceptions gained from the literature studies were used as a foundation to design the application. Since the application uses external tools to receive, store and analyze data, it is dependent on their features, reliance and availability. Moreover, the findings of the literature study had to be reconciled with the features of the used services.

## 4.2 Dependencies

The application uses two main services which provide data in raw and analyzed form. On the one hand, there are the SOFAS metrics services that are responsible for the analysis of the project's source code. On the other hand, there is the *Rational Team Concert* server which is used to track and organize the work tasks and team members.

### 4.2.1 Rational Team Concert Server

The *Rational Team Concert* server is used to manage all work items and users as well as the source control. As soon as the developer installed the *Rational Team Concert* plugin in his IDE and connected it to the *RTC* server, he can create new projects or load existing ones. As the prototype application is only connected to one example project, this exact project has to be loaded in order to use the tablet application.

The application is thus dependent on the existence and accessibility of the example project on the *RTC* server. As soon as either the project is archived, the server is not running or has no network connectivity, the application will fail to load any data.

### 4.2.2 SOFAS Metrics Service

Even though the SOFAS metrics services are provided and hosted by the University of Zurich, they are, unlike the *RTC* server, accessible from the internet. In Section 3.2 the services and the interaction with them is described in more detail.

The SOFAS metrics services are used to calculate all the software metrics that are presented within the application. The source code for analysis has to be uploaded to the metrics service manually in a zip file. As soon as the analysis has finished, the data is available through the REST or the web interface.

A large part of the application is dependent on the SOFAS metrics services. Without the service, multiple parts of the application will not display any data. The fact that the user has to upload the source code manually brings some user dependency to the application as well. The actuality of the calculated metrics is only given if the source code used for analysis is updated on a regular basis.

## 4.3 User Interface

The application consists of six main user interface parts which are available through the application's menu bar on the left. They are introduced in the following Sections.

### 4.3.1 Project Overview

The project overview part gives a brief survey of the project. The left column shows all work items in the project that need to be resolved. This list is ordered by the creation date, new work items appear at the top, older ones at the bottom. A click on a work item displays the work item view where more information about a work item is available.

To the right there are two bar charts. The upper one shows the work items categorized by priority. The chart gives an overview over the distribution of the work item priorities. The lower chart shows the number of resolved work items sorted by users. It is a simple way to get an overview

over the productivity of each user. However, it has to be considered that the bare number of resolved work items does not tell anything about the amount of work behind each work item. Nonetheless, the chart can be used to get a glimpse on how many work items each user has resolved. To get more detailed information about a user's work, the user's own view which is described in Section 4.3.5 has to be considered.

The reason why the project overview was implemented, is to give an up-to-date overview over what is currently happening in the project and see how the team members performed in the last weeks.

Figure 4.1 shows a screenshot of the project overview in the prototype application.

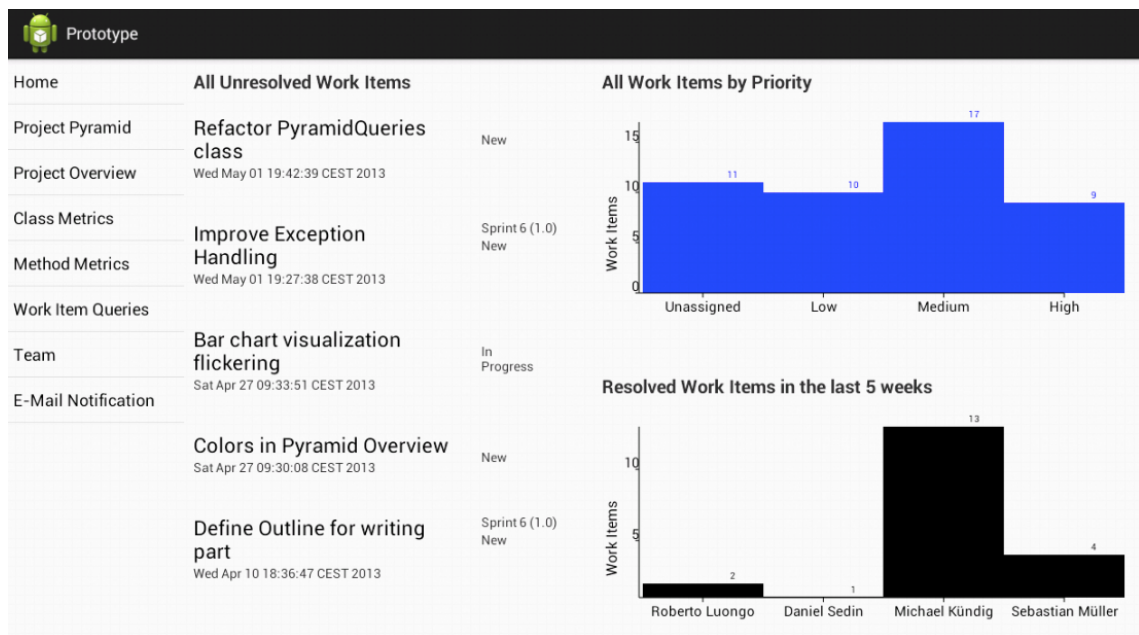


Figure 4.1: Screenshot of the project overview in the application

### 4.3.2 Overview Pyramid

This user interface part implements the Overview Pyramid designed by Michel Lanza and Radu Marinescu. A more detailed introduction is presented in Section 3.1. In the upper right, there is a scrollable legend that explains the abbreviations and metrics for those who are not familiar with them.

The metrics abbreviations with a yellow background describe size and complexity metrics, those with a green background describe inheritance metrics, and those with a blue background describe coupling metrics. The metric ratios have a short description on top of them and are colored in either dark blue, dark green, or red. Dark blue indicates a too low, green an average, and red a too high value. The thresholds that are used to set the background color of the ratios are the same as in the original Overview Pyramid and presented in Figure 3.2.

The reason why the Overview Pyramid was selected to be a part of the application, was to show a high level overview about the source code of a project. Lanza and Marinescu describe that the basic idea of the Overview Pyramid is "to put together in one place the most significant measurements

about an object-oriented system, so that an engineer can see and interpret in one shot everything that is needed to get a first impression about the system" [LM06]. The idea is that a project manager can get a good impression of the system as a whole by taking a look at the Overview Pyramid.

As for this project, the Overview Pyramid reveals that it has low inheritance metric values. A reason why the *Average Hierarchy Height* and the *Average Number Of Derived Classes* values are low, might be because it is an *Android* project and a lot of the classes in the project extend from the *Android* framework. Since only inheritance from user-defined classes is considered, classes that extend from the *Android* framework are ignored in the evaluation of the inheritance metrics. Further, the pyramid shows that there are too many calls per method and that the coupling between classes (FANOUT/CALL ratio) is too low. The size and complexity metrics ratios are in the average threshold range or just slightly below.

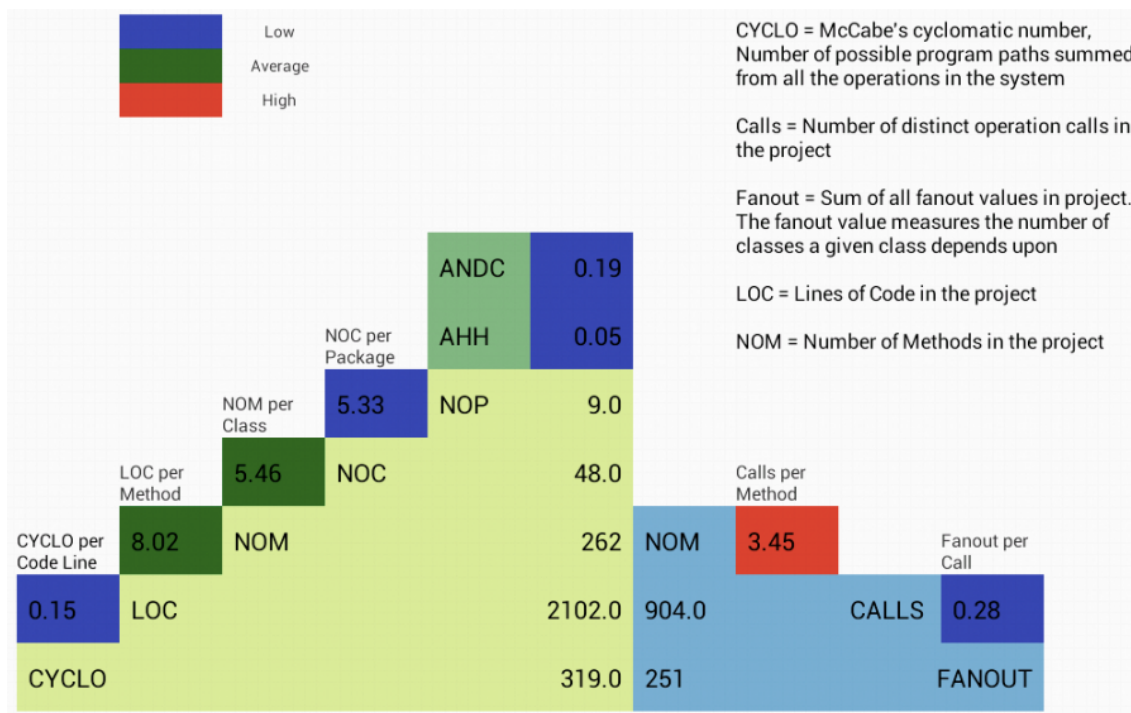


Figure 4.2: Screenshot of the Overview Pyramid in the application

### 4.3.3 Method and Class Metrics

This part connects the SOFAS metrics services, introduced in Chapter 3.2, with the *Rational Team Concert*, described in Chapter 3.3. The class metrics part consists of five metrics which each display five classes. The shown classes have the highest metric value of the respective metric in the entire project. They are ordered by their value. Therefore, the class with the highest and accordingly most crucial value is on top of the list, the class with the fifth highest value at the bottom. The method metrics part consists of four metrics and is structured in the same fashion.



The goal of this user interface part is to see when a class that has, or contains a method with a high metric value, was last modified, by whom it was modified, and in relation to which work item. The first column shows a message icon. A click on that item adds the respective line to the e-mail notification list, which is more precisely described in Chapter 4.3.6. The class or method names are displayed in the second column. A click on the class or method name shows the Java source code of the related class. The third column contains the metric value calculated by the metrics service. The last three columns contain the name of the related work item, the latest modification date, and the user who changed the work item last.

The following two Sections present the metrics that are displayed in the method and class metrics part [fam] [jav]. The metrics in the two Sections were chosen to be displayed because they are all related to the metrics from the Overview Pyramid. Besides, the variety of available metrics about a single method or class was limited.

## Method Metrics

- **Maximum nesting level (MAXNESTING):** Maximum number of nested conditional statements within a method.
- **Coupling intensity (CINT):** The number of distinct methods called by the measured method.
- **Changing classes (CC):** The number of classes in which the methods that call the measured method are defined in [LM06].
- **Changing methods (CM):** The number of distinct methods calling the measured method.

Changing Classes		Value	Related Work Item	Date	User
<input checked="" type="checkbox"/>	com.prototype.jazz.JazzDataManager.getInstance()	17.0	Chart to show open Work Items by priority	Wed Apr 10 18:34:00 CEST 2013	Roberto Luongo
<input checked="" type="checkbox"/>	com.prototype.jazz.JazzDataManager.getWorkItem(java.lang.String)	6.0	Chart to show open Work Items by priority	Wed Apr 10 18:34:00 CEST 2013	Roberto Luongo
<input checked="" type="checkbox"/>	com.prototype.rest.HttpUtils.getXPath()	6.0	Code Refactoring 1	Tue Apr 09 14:38:44 CEST 2013	Michael Kündig
<input checked="" type="checkbox"/>	com.prototype.jazz.User.getName()	5.0	Show open Work Items per User	Tue Apr 09 18:06:36 CEST 2013	Roberto Luongo
<input checked="" type="checkbox"/>	com.prototype.rest.HttpResponseReceiver.receiveJazzResponse(java.lang.String,int)	4.0	Update Pyramid View	Wed Apr 24 12:55:48 CEST 2013	Michael Kündig
Changing Methods		Value	Related Work Item	Date	User
<input checked="" type="checkbox"/>	com.prototype.jazz.JazzDataManager.getInstance()	25.0	Chart to show open Work Items by priority	Wed Apr 10 18:34:00 CEST 2013	Roberto Luongo
<input checked="" type="checkbox"/>	com.prototype.rest.HttpUtils.getXPath()	21.0	Code Refactoring 1	Tue Apr 09 14:38:44 CEST 2013	Michael Kündig

**Figure 4.3:** Screenshot of the method metrics part in the application

## Class Metrics

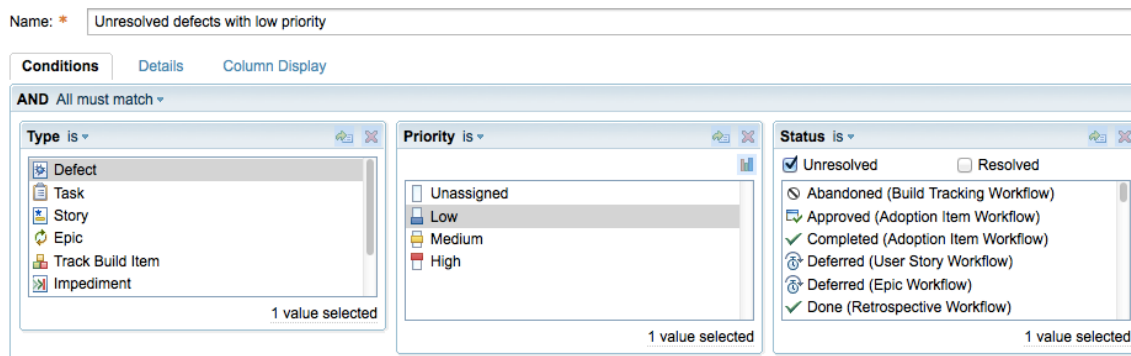
- **Average method weight (AMW):** The average of McCabe's static complexity [McC76] of the methods of the measured class.
- **Weighted method count (WMC):** The sum of McCabe's static complexity of the methods of the measured class.
- **Tight class cohesion (TCC):** The cohesion between the public methods of a class. "Two methods are related through instance variable(s) if both methods use the same instance variable(s)" [BK95]. The result value ranges between 0 and 1. The higher the value, the more methods are related through instance variables and the more cohesive is the class.
- **Number of attributes (NOA):** The number of static and non static attributes of a class.
- **Number of methods (NOM):** The number of static and non static methods of a class.

### 4.3.4 Work Item Queries

*Rational Team Concert* allows to define queries that help to find work items more quickly. Queries are a powerful tool to keep a clear view over all work items, even if a huge number of them are defined. Through conditional statements, work items can be filtered in the desired way.

Work item queries are either created through the web client or the IDE plugin of *Rational Team Concert*. Any number of conditions can be defined and one can choose if either all of them (AND operator) or any of them (OR operator) must match. A query is accessible to the user who created it, but can be shared to the whole team as well.

Figure 4.4 shows how a query is defined through the web interface. There are three conditions defined for the query *Unresolved defects with low priority*: 'Type is Defect', 'Priority is Low', and 'Status is Unresolved'. A work item must match all conditions in order to be found by the query.



**Figure 4.4:** Screenshot of the query definition part in Rational Team Concert

In Figure 4.5, a screenshot of the work item queries part from the prototype application is shown. In the column to the left, all shared and user-defined work item queries are listed and can be selected. The shared queries appear at the top of the list, whereas the user-defined queries are found at the bottom. After a query has been selected, the right column displays the title, the creation date, and the status of the work items that match the conditions of the selected query. Two work items match the criterias of the query defined above, as Figure 4.5 illustrates: *Colors in Pyramid Overview* and *Bar chart visualization flickering*.

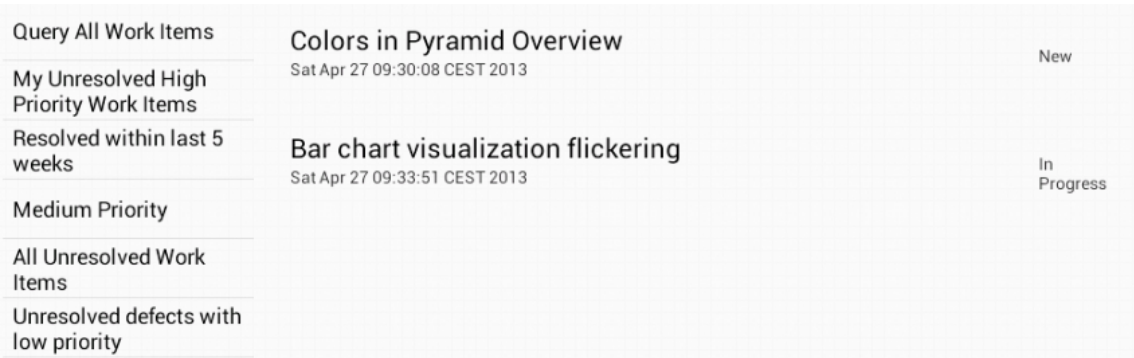


Figure 4.5: Screenshot of the work item queries in the application

Figure 4.6 shows how the information about the selected *Bar chart visualization flickering* work item is displayed in the prototype application.

## Bar chart visualization flickering

Description	Type	Defect
Bar charts in the Project Overview and in the User view are sometimes flickering. Check the bar chart template in the <i>ChartHelper.java</i> class.	Creator	Roberto Luongo
	Creation Date	Sat Apr 27 09:33:51 CEST 2013
	State	Reopened
	Last Modified	Sat May 18 09:13:04 CEST 2013
	Close Date	
	Planned For	Sprint 6 (1.0)

Figure 4.6: Screenshot of a work item in the application

The idea behind the work item queries part is to offer a quick navigation through all work items in the project. The project manager should have the ability to use his own defined queries. Since the queries are highly customizable, it is possible for the project manager to quickly get an overview about different specific parts of the project.

### 4.3.5 Team

The team user interface part allows to take a closer look at every single member of the team. In the team view, the application presents a list with all team members. A click on a user displays the selected user's view which is shown in Figure 4.7.

The column to the left presents information about the user as well as a list of unresolved work items that are assigned to the user. In the right column, there is a bar chart that shows the distribution of how many work items the user has resolved in the past five weeks. Below the bar chart there is a scrollable list that shows the user's resolved work items. The list is descendingly ordered by the resolve date of the work items.



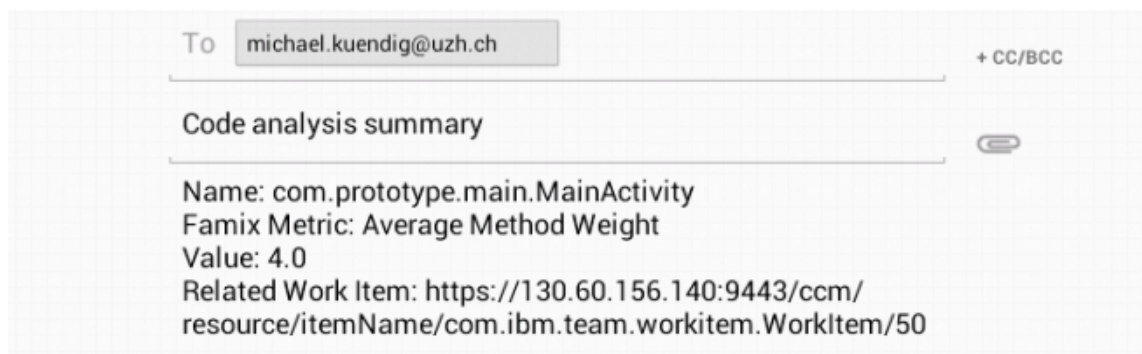
**Figure 4.7:** Screenshot of a user in the application

### 4.3.6 E-Mail Notification

The e-mail notification part allows to send an e-mail containing information about method and class metrics directly from the application. Every line in the method and class metrics part, which is described in Section 4.3.3, contains a message icon all to the left. A metric is added to the e-mail notification list by clicking on the message icon. If the metric has been added to the notification list, the message icon is replaced by a check mark. Figure 4.3 shows how two different metrics have been selected. All selected metrics will appear in the e-mail notification list as shown in Figure 4.8. As soon as all desired metrics have been added to the notification list, a click on the *Send E-Mail* button automatically creates an e-mail with information about the desired metrics and the work items related to them. The idea is that the project manager can use this ability to send himself or somebody else a reminder about discoveries he made with the application. Figure 4.9 shows an example of a automatically created e-mail template.

Send E-Mail		
Average Method Weight	com.prototype.main.MainActivity	4.0
Weighted Method Count	com.prototype.jazz.receiver.WorkItemReceiver	25.0
Changing Classes	com.prototype.rest.HttpUtils.getXPath()	6.0
Changing Methods	com.prototype.jazz.JazzDataManager.getWorkItem(java.lang.String)	6.0

**Figure 4.8:** Screenshot of the e-mail notification user interface in the application



**Figure 4.9:** Screenshot of an automatically created e-mail template

## 4.4 Used Technologies and Frameworks

The following two Sections describe the frameworks that were used in the development of the prototype application. Both frameworks are not part of the *Android* API and had to be imported manually.

### 4.4.1 AChartEngine

*ACHartEngine* is an open source software charting library for *Android*. It can be used to build chart views of many different kinds, for example bar charts, line charts, pie charts, or scatter charts. The chart is defined all in Java and added to a layout element which can be defined statically or programmatically [ach].

*ACHartEngine* is used in the prototype application to draw bar charts in the project overview and the user view. Figure 4.1 shows the user interface of the project overview containing two bar charts created with *ACHartEngine* to the right.

### 4.4.2 QDox

*QDox* is a lightweight parser for extracting class, interface, and method definitions from source code files. The source code files are skimmed by the parser and categorized by class definitions, import statements, member declarations, and JavaDoc comments. The result is a document model containing the useful information about the parsed file [qdo].

In the prototype application, *QDox* is used to parse the source code received through the REST

interface of *Rational Team Concert*'s source control. *QDox* parses the plain text source code files and provides the information from the imported code in a `JavaClass`. Information about the Java code can now comfortably accessed through the methods of the `JavaClass`.

## 4.5 Use Case

The following Section describes a use case of how the functionalities of the prototype application could be utilized. The code snippets and metrics used in this use case represent the prototype application itself.

Assume a small team of software developers and a project manager are working on the development of an *Android* application. The project manager is responsible for the outcome of development process, but as a matter of fact, he is engaged a lot with communicating with stakeholders outside the project and has almost no time to assist the development process himself. To stay aware of the quality and the status of the project, he uses the prototype application and manages the developers work through *Rational Team Concert*.

The tablet application gives him the opportunity to get a view on the source code quality and the workload of his team, even if he is not in front of his desktop computer.

### 4.5.1 Detecting a critical Method

While browsing through the prototype application and taking a look at the project's status, the project manager spots a method with a critically high nesting value. By clicking on the method's name, he can immediately take a look at the code of the class that contains the method. He is also able to see the work item that is related to the latest class change and the user who updated the work item. By taking a closer look at the work item, the project manager discovers that it has been changed just recently and marked as done. The project manager is not satisfied with this result and decides that the method has to be refactored.

Maximum Nesting Level	Value	Related Work Item	Date	User
 com.prototype.main.MainActivity.onItemSelected(java.lang.String)	9.0	Update Pyramid View	Fri Apr 19 15:19:08 CEST 2013	Michael Kündig
 com.prototype.jazz.receiver.UserReceiver.receiveResolvedWorkItems(com.prototype.jazz.User)	5.0	Update Pyramid View	Wed Apr 24 12:55:48 CEST 2013	Michael Kündig

**Figure 4.10:** Screenshot of the methods with a critical high nesting value

### 4.5.2 Sending E-mail Notification

The project manager adds the method and the corresponding work item to the e-mail notification list and keeps on browsing. Before he finishes using the application, he sends a notification e-mail containing the information about the method with the critical high nesting level to his work address.

As soon as he is at his work computer again, he clicks on the link in the notification e-mail and finds himself immediately at the right place on the web client of the *Rational Team Concert*. Since it is not a very urgent issue, he creates a low priority task and assigns it to the user who modified

the class last and who is responsible for that part of the code.

The project manager could also send the e-mail directly to the developer and ask him to create a new work item.

### 4.5.3 Refactoring

Soon after, the developer checks his current work and finds that a newly created task has been assigned to him. As soon as he has some spare time to work on that task, he updates the status of the work item from *new* to *start working*. All other team members are now able to see that he is currently working on that task.

As long as the user is working on that work item, everyone in the team can see that the status of the work item is *in progress*. Unfortunately, the developer can not resolve the issue due to some dependency issues. Therefore, he does not mark the work item as resolved, but sets its status to *stop working* and adds a note to its description why it has not been marked as resolved yet.

### 4.5.4 Keeping Track of the Work Item Status

The project manager has several ways to keep track of the work item's status in the prototype application. As long as the work item is unresolved he can find it in the project overview. He can also use a query or check the developer's user view to check the status of the work item.

As soon as the project manager sees that the developer stopped working on the task, he checks the work item from the prototype application. He can see what was added to its description and thus knows why it has not been marked as resolved yet. For checking purposes, he takes a look at the method metrics part and finds the method gone from the list of the methods with the highest maximum nesting level.

### 4.5.5 Limitation

The inability to create the work items from the application directly limits the efficiency of the project manager. If a work item could directly be created from the application, the project manager would not have to send a notification e-mail to himself to create the work item later at his work computer.





# Implementation Details

## 5.1 Internal Structure

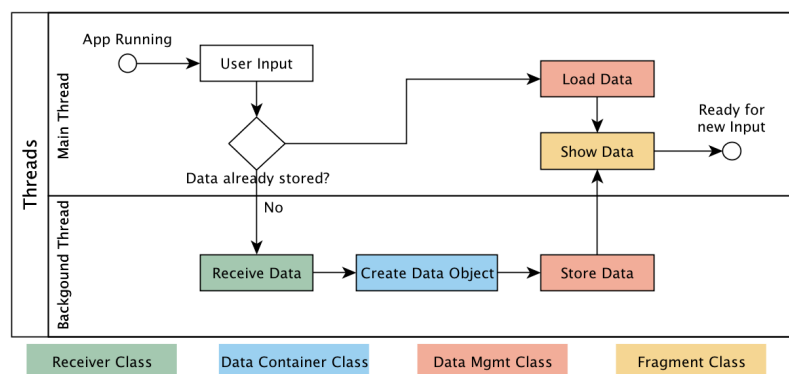
There are four different types of components that build up the application. The four components and their functionality are described in the following Section.

### 5.1.1 Receiver Classes

The main job of receiver classes is to receive data from the Metrics Service and the *Rational Team Concert* Server. An abstract receiver class which contains the information of how to communicate with the RESTful web services is defined in the project. Several other receiver classes that are specialized to receive one specific data object, for example a workitem or a user, inherit from the abstract receiver class.

To prevent the user interface from freezing, *Android* does not allow network operations on the main application thread [Mei12]. Therefore, all receiver classes are called from an `AsyncTask`, a class defined in the *Android API* that allows to perform operations on a background thread. While receiving the data, a `ProgressDialog` appears and shows that the app is loading. As long as the `ProgressDialog` is showing, user input is ignored.

Network operations are not allowed on the main application thread, whereas Activities and their Fragments are always handled by the main thread.



**Figure 5.1:** Flowchart of a user input scenario

### 5.1.2 Fragments

The `Fragment` class is provided by the *Android* API and represents some behaviour or a portion of user interface within an activity. Figure 3.5 shows how Fragments are intended to create multi-pane user interfaces.

To create a new fragment, a subclass of `Fragment` has to be created. A fragment is always embedded in an activity and dependent on its lifecycle. If the activity is paused or stopped, the fragment is paused or stopped as well. Every activity has a `FragmentManager` that can be used to interact with fragments, for example hiding, showing, or replacing them. The user interface of an activity or a fragment is either defined statically in an XML file or programmatically in the source code. It is also possible to define part of the user interface statically and add some elements or content programmatically [Mei12]. A typical XML file that defines a user interface is shown in Listing 5.1.

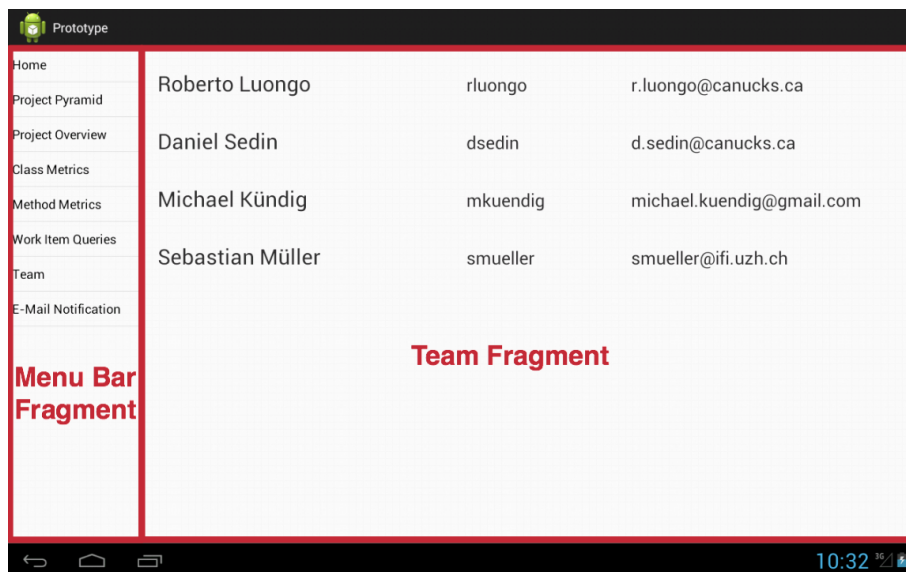
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/main_activity_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:weightSum="100" >
    <LinearLayout
        android:id="@+id/menu_bar_container"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="15" />
    <LinearLayout
        android:id="@+id/central_container"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="85" />
</LinearLayout>
```

**Listing 5.1:** Example XML file that defines a user interface

The prototype application consists of only one activity. Listing 5.1 contains the layout structure of that activity. There are two layouts aligned horizontally, the left layout uses 15 percent of the screen width and contains a fragment that displays the menu bar of the application. The right layout takes 85 percent of the screen width and holds the currently selected fragment. Figure 5.2 shows how the layout specifications in Listing 5.1 build up the fragment structure in the prototype application.

### 5.1.3 Data Container Classes

Some classes are intended to store data loaded by the receiver classes. A data container class represents an entity defined by the service, such as a metrics result or a single work item. Every container class is interconnected with a receiver class. A data container object is only instantiated from the receiver class that is supposed to load that data container class.



**Figure 5.2:** The application's fragment structure

### 5.1.4 Data Management Classes

Data management classes are the connector between receiver classes and fragments. The job of these classes is to organize the received and created data container objects and provide them to the fragments. There are only three data management classes in the project. One to handle the data from *Rational Team Concert*, one to handle the data from the Metrics Service and one to handle the connected data from both tools.

## 5.2 Receiving Data from the Metrics Services

Data is received from the SOFAS metrics services by sending an encoded SPARQL Query to the RESTful interface of the metrics service and parsing the received response. This procedure is explained in the following Sections.

### 5.2.1 SPARQL Query

The SOFAS metrics services are able to handle HTML encoded SPARQL (SPARQL Protocol and RDF Query Language) queries through their RESTful interface. The services execute the query and provide the result in a SPARQL Query Results XML Format <sup>1</sup>.

The following Listing shows a SPARQL query that receives the five largest values of the average method weight metric (AMW) and their corresponding label.

<sup>1</sup><http://www.w3.org/TR/2007/CR-rdf-sparql-XMLres-20070925/>

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX j.0: <http://habanero.ifi.uzh.ch/seon/softwaremetrics.owl#>
SELECT ?label ?value
WHERE {
    ?description rdfs:label ?label .
    ?description j.0:hasName "AMW" .
    ?description j.0:hasValue ?value
}
ORDER BY DESC(?value)
LIMIT 5

```

**Listing 5.2:** Example SPARQL query of the average method weight metric

## 5.2.2 Response Receiving

In order to communicate with the metrics services, Apache's `HTTPComponents2` library is used. The following code snippet shows how a metrics result in form of a `HttpResponse` object is received from the service.

First, a `DefaultHttpClient` and a `HttpHost` object are created. Next, the authentication credentials are added to the `DefaultHttpClient` object and a new `HttpGet` object is created by passing the entire request URL to the constructor. The URL consists of the `serviceName` which is one of the two names of the metrics services, the `analysisName` that is defined on the service, as well as the HTML encoded SPARQL query. After that, the HTTP Accept header field is added to the `HttpGet` object. In order to receive a valid response from the metrics service, the HTTP accept request-header field has to be `application/sparql-results+xml`. Last, the `HttpHost` and the `HttpGet` objects are executed by the `DefaultHttpClient` object.

```

DefaultHttpClient httpClient = new DefaultHttpClient();
HttpHost targetHost = new HttpHost("habanero.ifi.uzh.ch", 80, "http");
httpClient.getCredentialsProvider().setCredentials(
    new AuthScope(targetHost.getHostName(), targetHost.getPort()),
    new UsernamePasswordCredentials("username", "password"));
HttpGet httpGet = new HttpGet("/") + serviceName + "/analyses/"
    + analysisName + "?query=" + URLEncoder.encode(sparqlQuery, "UTF-8"));
httpGet.setHeader("Accept", "application/sparql-results+xml");
HttpResponse response = httpClient.execute(targetHost, httpGet);

```

**Listing 5.3:** Implementation of communication with the metrics service

---

<sup>2</sup><http://hc.apache.org/>

### 5.2.3 Response Parsing

The last step to make the metrics results easily accessible is to parse the received SPARQL Query Results XML file and store the values and names in a self defined `MetricsResult` data container object. The received `HttpResponse` object from Listing 5.3 is handled by the method shown in Listing 5.4.

First, an `ArrayList<MetricsResult>` is created to store the parsed results. XPath is used to navigate through the XML content of the `HttpResponse` object and create a `NodeList` object result entities of the XML response. In our case, the `NodeList` will have five entries since the average method weight query from Listing 5.2 limits the result entities to five. Each `NodeList` entry will again be evaluated with XPath and the result values are going to be stored in a `MetricsResult` which will be added to the initially created `ArrayList`.

```
private ArrayList<MetricsResult> parseResponse(HttpResponse httpResponse) {
    ArrayList<MetricsResult> resultList = new ArrayList<MetricsResult>();
    XPath xPath = HttpUtils.getXPath();
    InputSource source = new InputSource(httpResponse.getEntity().getContent());
    NodeList nodeList = (NodeList) xPath.evaluate("/:sparql:results/:result",
        source, XPathConstants.NODESET);
    for (int i = 0; i < nodeList.getLength(); i++) {
        Node node = nodeList.item(i);
        node.getParentNode().removeChild(node);
        String value = xPath.evaluate("./:binding[@name='value']/:literal", node);
        String name = xPath.evaluate("./:binding[@name='label']/:literal", node);
        MetricsResult famixResult = new MetricsResult(name, value);
        resultList.add(famixResult);
    }
    return resultList;
}
```

**Listing 5.4:** Parsing the content of a response from the metrics services

## 5.3 Connect RTC with the Metrics Service

The connection of the *Rational Team Concert* and the SOFAS metrics services works due to the fact that both services know the source code of the project. The metrics services obviously know the code because they have to analyze it and the *Rational Team Concert* knows the code because the project's source control runs through it. Source code changes are committed to *Rational Team Concert's* source control in form of change sets and have to be linked with a work item. How the connection is implemented in the application is shown in the following Section.

### 5.3.1 Finding a Change Set by the Name of a Java class

The `ChangeSet` class has four private fields: a `Date` field for the date when the code change was committed, a `String` that stores the URL which is used as the identifier of the related work item, a `JSONObject` that holds all the information about the change set, and a `HashMap<String, JavaClass>` that is used to store the source code of the change set's Java classes.

At startup of the application, a list of the URLs of every change set in the project is received

and an `ArrayList<ChangeSet>` containing the creation date, the work item URL and the `JSONObject` of each change set is created. The list is sorted descending by the commit date, hence the latest change set is at index 0.

In order to find the Java classes related to a change set, it is necessary to analyze the `JSONObject` of the change set and look for every URL that points to a text file containing Java source code and parse that file. Since this is a time consuming task, the `HashMap<String, JavaClass>` remains unset until the Java classes of that change set are needed.

Below, it is explained how to find the change set by the name of a java class with the help of an example.

```
public ChangeSet findChangeSet(String fullClassName, int changeSetIndex) {
    // changeSetToAnalyze is a static int variable
    if (changeSetToAnalyze <= changeSetIndex) analyzeNextChangeSet();
    if (changeSets.get(changeSetIndex).containsJavaClass(fullClassName)) {
        return changeSets.get(changeSetIndex);
    } else if (changeSetToAnalyze < changeSets.size()) {
        return findChangeSet(fullClassName, changeSetIndex + 1);
    }
    return null;
}
```

**Listing 5.5:** Finding the latest change set which is related to a given Java class

The method `findChangeSet` in Listing 5.5 is called after the data from the metrics service is loaded and stored in a `MetricsResult` object. After that, the Java class name is parsed and stored as follows: `com.prototype.main.MainActivity`. This class name serves as the identifier to find the linked Java class in a change set.

Next, the `findChangeSet` method is called with the full class name and the `changeSetIndex` 0. The `changeSetIndex` refers to the `ArrayList<ChangeSet>` that holds all change sets in descending chronological order. The first thing the method does, is to check if the change set at `changeSetIndex` has already been analyzed. The purpose of the static variable `changeSetToAnalyze` is to ensure that the `JSONObject` of every change set gets analyzed at most once. On application startup, the value of the variable is 0 and increases with every change set that has been analyzed. Since both values are 0 in our case, the `JSONObject` of the latest change set is analyzed and the `HashMap<String, JavaClass>` of the latest change set will be set.

The next `if` condition checks if the change set at index `changeSetIndex` of the `ArraxList` contains the Java class we are looking for. If this is the case, the found change set is returned. If not, the `findChangeSet` method is called recursively with the same class name, but a `changeSetIndex` increased by 1.

Assuming that the Java class corresponding to our class name has not been found in the first analyzed change set, the `findChangeSet` method will be called again. Next, the second latest change set will be analyzed since both the `changeSetToAnalyze` and the `changeSetIndex` values are 1. After that, the value of `changeSetToAnalyze` variable is 2.

Under the assumption that the second latest change set contains the wanted Java class, the found change set will be returned. If the `findChangeSet` method is called again with another class name, the first two change sets will not have to be analyzed again since `changeSetToAnalyze` value will be 2 and the hash maps of the first two change sets will be set.

# Evaluation

To evaluate and test the usability of the prototype application a formative evaluation was conducted. The goal was to analyze the functionality of the application and find out if it is easy and intuitive to use and suitable for the given tasks. Additionally, the participants were asked if they could imagine to use such an application in a productive environment and which functionalities would need improvement. The setup, the results, some suggestions for improvement, as well as the limitations of the evaluation are described in this Chapter.

## 6.1 Setup

The evaluation consists of three parts: a tasksheet that covers the capability of the application, a usability rating, and an open question part. These three parts are briefly described in the following:

- **Tasks:** The tasks section consists of eight different tasks that demand using all different parts of the application described in Section 4.3. The user is asked to rate the difficulty and the time intensity of each task on a five point rating scale.
- **Usability rating:** The user was asked to rate eight polarity profiles [KH] about the general usability, the intuitivity, and the tablet optimization of the application.
- **Open questions:** The open questions focus on the detection of improvement suggestions and missing features.

Nine persons participated in the evaluation. All of the participants were either students or postgraduates in the field of computer science and therefore ideally equipped to evaluate the application. Those who had no experience with software metrics or team collaboration tools like the *Rational Team Concert* were shortly informed about the topic.

At the beginning, the participants were briefly introduced to the setup of the evaluation and a tablet with the running application was handed to them. The application itself was not explained. They were allowed to ask questions during the evaluation, but the goal was to let them solve the tasks by themselves.

The procedure of the evaluation was similar to the lab test introduced by Tullis and Albert, where *"the moderator [...] gives them (participants) a set of tasks to perform on the product in question"* and the test *"requires a relatively small number of participants (typically four to ten)"* [TA08]. The data collected in the evaluation consists of the success of each task, the results of the usability rating and the answers to the open questions.

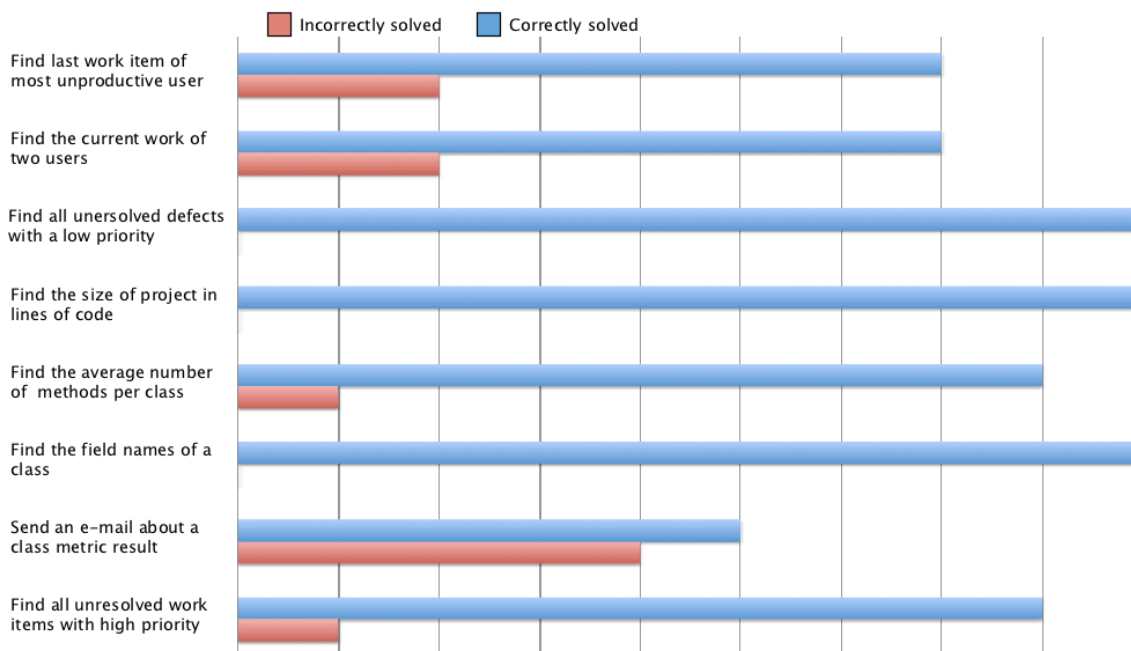
The handed out evaluation sheet is found in Appendix A.

## 6.2 Results

The result section is divided into the three parts of the evaluation. First, the success rate, the difficulty, and the time intensity of the tasks are analyzed. Second, the results of the usability rating are presented, and third, the answers to the open questions are analyzed.

### 6.2.1 Task Success

The following bar chart shows how successful each task was solved. To the left of the chart, there is a brief description of each task. The tasks are ordered in the same way as in the original evaluation form (see Appendix A).



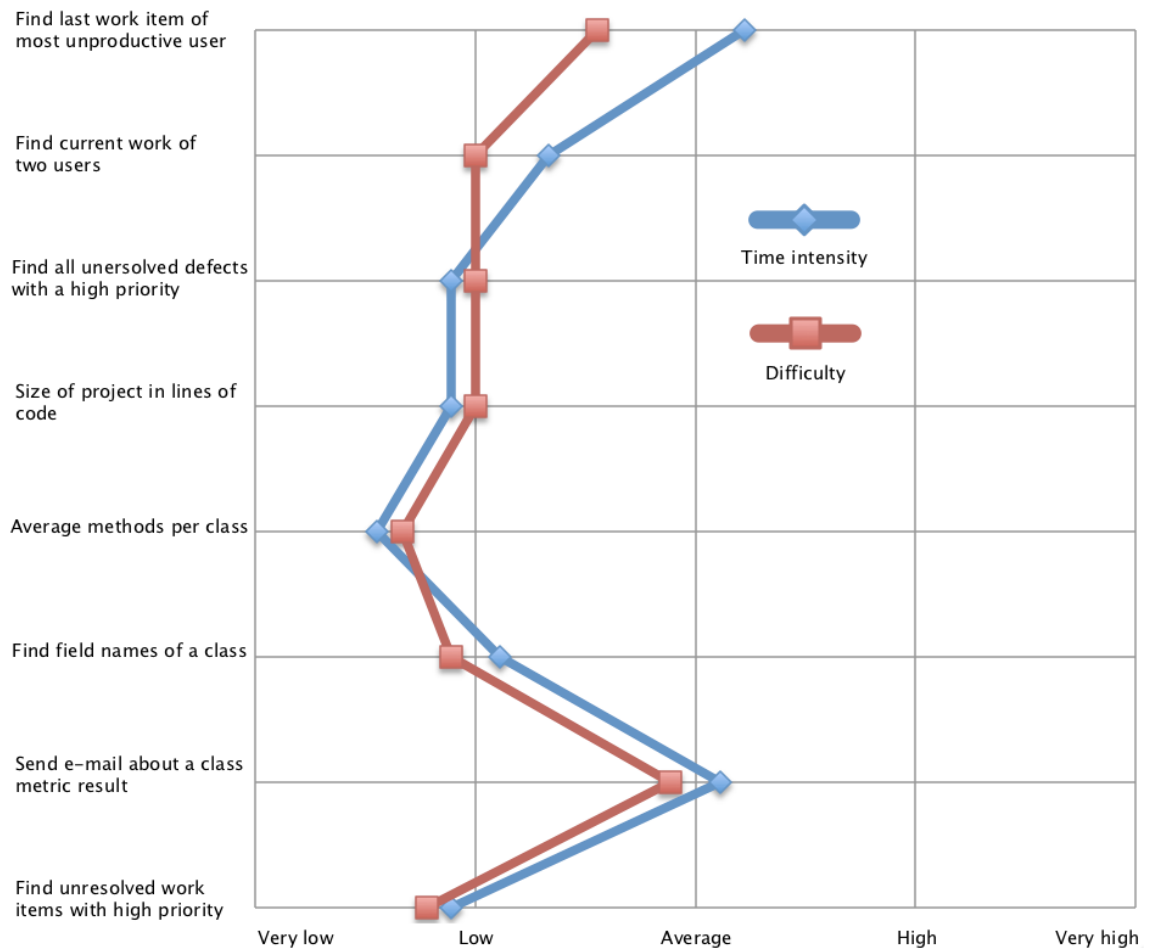
**Figure 6.1:** Bar chart showing the success rate of each task

The chart in Figure 6.1 shows that three out of eight tasks were solved correctly by every participant and four tasks were solved correctly by at least seven out of the nine participants. The main source of error in those four tasks was that the answers were not complete. Somehow, the participants wrote down only one part of the answer, which was correct in every case, and forgot about the rest of the answer.

Only one task, where the participants had to send an e-mail notification about a given metric result, caused difficulties. The idea was that they would find the specified metric in the class metrics view and click on the message icon (see Figure 4.3), so the metric is added to the e-mail notification list. Unfortunately, four out of nine participants did not discover that it is possible to click on the message icon and were not able to send the e-mail. There was no hint in the e-mail notification view that explained first time users how to use the e-mail send ability.



The following line chart shows the average participant rating about the difficulty and time intensity of each task. The participants were asked to rate difficulty and time intensity directly after each task on a five point scale.



**Figure 6.2:** Line chart showing the difficulty of each task

The line chart in Figure 6.2 shows a clear correlation between the participants' rating of the difficulty and the time intensity of each task.

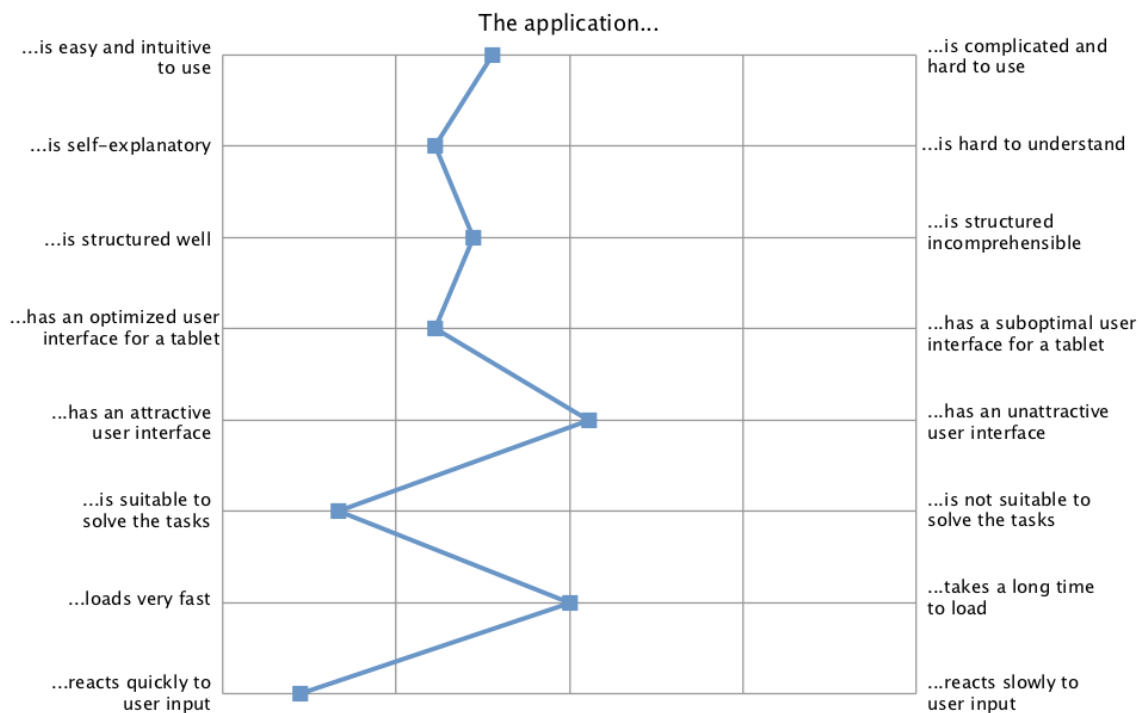
What attracts attention in the chart is the fact that the participants rated the first task as far more time intensive as most of the other tasks. Some participants even annotated on the time intensity rating of the first task that it took them some time to get to know the application during the first task. For this reason, one can suppose that the time intensity and difficulty rating of the first task would have been lower if the participants had some time to become acquainted with the application before they started the tasks.

Another thing that stands out is, like in Figure 6.1, task number seven where the participant had to send an e-mail notification. Since four out of nine participants failed to send the notification, it does not surprise that this task was rated as the most difficult one.

Altogether, one can come to the conclusion that only one task had an unsatisfying outcome and that most of the tasks were solved correctly with fairly low effort.

## 6.2.2 Usability Rating

The following chart shows how the participants rated the usability of the application on a five-point polarity profile.



**Figure 6.3:** Usability rating of the application

The most negatively ranked feature in the polarity rating of Figure 6.3 is the attractiveness of the user interface. Since the application was built as a prototype, no focus was laid on the attractiveness of the application. Still, some improvements of the user interface design would also increase the intuitive use of the application. One participant noted in the open question section that it was sometimes unclear to him which elements in the application were clickable and which were not. A revision of the user interface would not only make the application more attractive, but also help to get rid of such unclarity.

Another comment needs to be made about the loading time of the application because it is the second worst rated feature and has the largest rating diversity. The application took about one to two minutes to initially load all data. While some participants browsed through the evaluation form or asked some questions during the application was loading, others just waited for it to finish loading, so they could start. This clearly indicates that not all participants paid attention to the loading time in the same extent.

The overall rating of the usability shows a fairly satisfying result which proves that the application is a valuable instrument to stay aware of a project's status and software quality.

### 6.2.3 Open Questions

The open questions section consisted of four questions. The first question asked the participants if they could imagine using a fully developed application like this prototype in a productive environment. Generally, all participants liked the idea of the application and could imagine using it. Some stated that the application is no replacement for a tool that is integrated in the development environment, but rather a nice addition for situations when they are out of office. One participant mentioned that the application could be useful for a project leader and someone else noted that it could be convenient in a scrum meeting situation.

The second question was about the features the participants particularly liked about the application. They mentioned the project and the user view, the ability to send e-mail notifications, the clearly arranged user interface, the possibility to query for work items, and the fact that the application fosters team collaboration.

When asked about things they disliked about the application, the participants mentioned the interactivity of the application, *i.e.* more elements should be clickable and it should be obvious which elements are clickable. Further, the missing ability to create work item queries and the design of the user interface were also brought up.

The last question referred to suggestions for improvement which are described in the following Section.

### 6.2.4 Suggestions for Improvement

The following list contains the improvement suggestions brought up by the participants.

- **Add ability to edit work items:** Allow to add comments or change the description of a work item.
- **Make e-mail notification feature more intuitive:** Simplify the way to add metrics to the notification list.
- **Revise home screen:** Add more content to the home screen. It only shows the name of the application so far.
- **Add ability to search or group queries:** Avoid the long list of queries by adding the ability to group or search through them.
- **Add syntax highlighting:** Make Java Code better readable by adding syntax highlighting.
- **Add ability to see metrics of all classes and methods:** Add the metrics of all classes and methods in the project and make them browseable, *e.g.* through the package structure of the project.
- **Make user interface more interactive:** Add interactive elements to the user interface like zooming and sliding. Make more elements clickable.
- **Add more background information:** Help users who are not familiar with software metrics and work items by adding more hints and background information.

## 6.3 Limitations

The evaluation was conducted with only nine participants. For this reason, the degree to which the results of this evaluation can be generalized, is limited. Moreover, the results of this evaluation could also be biased due to the selection of the participants which were composed of students and postgraduates. The postgraduates have obviously more experience with collaboration tools and software metrics which might cause a different rating of the application. Further, the fact that the participants were directly asked to participate in the evaluation might lead to too positive answers. Tullis and Albert state that the *"participants in a usability lab essentially want to tell us what they think we want to hear, and that is usually positive feedback about our product"* [TA08]. The evaluation would have had to be conducted in an anonymous environment to avoid this effect which is called the social desirability bias [TA08].

# Conclusion

## 7.1 Summary

The work task of a project manager is complex and consists of many different activities. To support him in his decision making progress, he needs analyzed data from his project that is presented to him in an easy to understand fashion.

The prototype application that is developed in connection with this thesis focuses on a high level project overview. It is used in conjunction with *Rational Team Concert* that helps raising project awareness and the SOFAS metrics services which provide software metrics to analyze the software quality of the project. The application is developed as a mobile application on the *Android* platform and its user interface is optimized for tablets.

The Overview Pyramid that represents one part of the application provides a visualization of a software project in its entirety and is designed to present an overall status of a project's software quality. In addition, several software metrics are used to indicate potentially critical code areas. Further parts of the application provide an overview of the tasks of a single developer and the current work tasks in the project. Moreover it is possible to display specific work items using self-defined queries. To remind oneself of critically high software metrics, the application allows one to send notification e-mails.

The results of the evaluation illustrated that the prototype application was intuitive to use. Further, all participants could imagine to use a fully developed version of the application in a productive environment to keep track of a software project's status. Nevertheless, the evaluation also brought several problems and limitations to light. To eliminate the limitations, the Future Work Section below presents some of the initial thoughts, along with other improvement ideas.

## 7.2 Future Work

Multiple suggestions for improvements arised from the open questions in the evaluation. While some of the suggestions partially miss the idea of the use case, others seemed to be very particular for a prototype application, *e.g.* the improvement of the user interface design. The following list covers the three most crucial limitations that need to be dealt with in future work.

- **Add local database:** The evaluation showed that the initial loading time was considered to be too long by some of the participants. The reason for this is that the application loads most of the required data on startup and does not store it on the device after it has been closed. For this reason, the loading process at application startup takes a long time. *Android* allows one to store application data in a local database on the mobile device. If such a database was

implemented, the application would only need to check if some of data has been updated and would not have to load all of the data again.

- **Allow creation and editing of work items:** The use case as well as some of the participants in the evaluation suggested to add the functionality to create work items. The RESTful interface of the *Rational Team Concert* does not only allow one to request data via HTTP Get requests, but also to send and store data on the server via HTTP Put requests. The functionality to create and edit work items through the application would certainly increase the productivity of a project manager.
- **Search or group queries:** The prototype application presents the work item queries in a simple list. Since any number of queries can be defined, the list might get disorganized and confusing. Adding the ability to either search for queries or group them would prevent this.

## Appendix A

---

# Evaluation Form

## Usability Evaluation

Evaluation number:.....

---

### Productivity

---

#### Task 1

Take a look at the productivity of all users. Go to the user page of the most unproductive user of the last five weeks and check when he resolved his last work item.

Title and **close** date of the last resolved work item:

.....  
.....

How much time did it take to complete this task compared to what you would expect?

Too much

☐

Plenty

☐

Average

☐

Little

☐

Very little

☐

How hard was it to solve the task?

Very Easy

☐

Easy

☐

Average

☐

Hard

☐

Very Hard

☐

#### Task 2

What is Michael Kündig's and Daniel Sedin's current work?

Work Item title and type of Michael Kündig's work:

.....  
.....

Work Item title and type of Daniel's work:

.....  
.....



How much time did it take to complete this task compared to what you would expect?

Too much	Plenty	Average	Little	Very little
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How hard was it to solve the task?

Very Easy	Easy	Average	Hard	Very Hard
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

### Task 3

Find all unresolved defects that have a low priority.

Defect titles:

.....  
.....

How much time did it take to complete this task compared to what you would expect?

Too much	Plenty	Average	Little	Very little
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How hard was it to solve the task?

Very Easy	Easy	Average	Hard	Very Hard
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

---

**Metrics**

---

**Task 4**

Figure out the size of the project measured in lines of code. How many lines of code does the project consist of?

Lines of code:

.....

How much time did it take to complete this task compared to what you would expect?

Too much

☐

Plenty

☐

Average

☐

Little

☐

Very little

☐

How hard was it to solve the task?

Very Easy

☐

Easy

☐

Average

☐

Hard

☐

Very Hard

☐**Task 5**

How many methods does a class of the project contain on average?

Methods per class:

.....

How much time did it take to complete this task compared to what you would expect?

Too much

☐

Plenty

☐

Average

☐

Little

☐

Very little

☐

How hard was it to solve the task?

Very Easy

☐

Easy

☐

Average

☐

Hard

☐

Very Hard

☐

**Task 6**

Take a look at the java code of the class that has the **second** highest number of attributes. What are the names of the first three attributes/fields of the class?

Attribute names:

.....

.....

.....

How much time did it take to complete this task compared to what you would expect?

Too much

☐

Plenty

☐

Average

☐

Little

☐

Very little

☐

How hard was it to solve the task?

Very Easy

☐

Easy

☐

Average

☐

Hard

☐

Very Hard

☐

---

**RTC Connectivity**

---

**Task 7**

Assume that you want Michael to take a closer look at the class that has the highest average method weight and its related work item. Send an E-Mail to [michael.kuendig@gmail.com](mailto:michael.kuendig@gmail.com) that contains the information about this work item and about the work item related to the method with the highest coupling intensity.

How much time did it take to complete this task compared to what you would expect?

Too much	Plenty	Average	Little	Very little
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How hard was it to solve the task?

Very Easy	Easy	Average	Hard	Very Hard
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Task 8**

Suppose that you have created a query in RTC that shows your unresolved work items that have a high priority. Find that query in the tablet application. What's the name of the open work item that has a high priority? How long has it been unresolved, respectively when was it created?

Name and creation date of the work item:

.....

How much time did it take to complete this task compared to what you would expect?

Too much	Plenty	Average	Little	Very little
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How hard was it to solve the task?

Very Easy	Easy	Average	Hard	Very Hard
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

---

**General usability rating**


---

<i>The application...</i>						<i>The application...</i>
...is easy and intuitive to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...is complicated and hard to use
...is self-explanatory	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...is hard to understand
...is structured well	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...is structured incomprehensible
...has an optimized user interface for a tablet	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...has a suboptimal user interface for a tablet
...has an attractive user interface	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...has an unattractive user interface
...is suitable to solve the tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...is not suitable for the tasks
...loads very fast	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...takes a long time to load
...reacts quickly to user input	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	...reacts slowly to user input

---

**Further Questions**

---

Could you imagine to use a fully developed application like this in a productive environment? Why or why not?

.....

.....

.....

.....

What did you like in particular?

.....

.....

.....

.....

What did you dislike in particular?

.....

.....

.....

.....

What do you think should be improved?

.....

.....

.....

.....

Additional comments:

.....

.....

.....

# Content of the CD-ROM

- **Abstract.txt**  
English version of the abstract of this thesis
- **Zusfsg.txt**  
German version of the abstract of this thesis
- **Bachelorarbeit.pdf**  
This thesis
- **PrototypeApplication.zip**  
Project of the prototype application developed in conjunction with this thesis
- **Readme.txt**  
Readme file for the prototype application
- **PrototypeApplication.apk**  
Android installer package of the prototype application





---

# Bibliography

- [ach] <http://www.achartengine.org/>.
- [and] <http://developer.android.com/distribute/googleplay/quality/tablet.html>.
- [BCSR07] Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George G. Robertson. Fastdash: A visual dashboard for fostering awareness. In *Software Teams. SIGCHI conference on Human Factors in computing systems*, pages 1313–1322, 2007.
- [BK95] James M. Bieman and Byung-Kyoo Kang. Cohesion and reuse in an object-oriented system. *SSR '95 Proceedings of the 1995 Symposium on Software reusability*, pages 259–262, 1995.
- [BZ10] Raymond P.L. Buse and Thomas Zimmermann. Analytics for software development. 2010.
- [BZ11] Raymond P.L. Buse and Thomas Zimmermann. Information needs for software development analytics. 2011.
- [CKG<sup>+</sup>00] A. E. J. M. Cavelaars, A. E. Kunst, J. J. M. Geurts, R. Cialesi, L. Grötvedt, U. Helmert, E. Lahelma, O. Lundberg, A. Mielck, N. Kr. Rasmussen, E. Regidor, Th. Spuhler, and J. P. Mackenbach. Persistent variations in average height between countries and between socio-economic groups: An overview of 10 european countries. *Annals of Human Biology*, 2000.
- [CL11] Andre Charland and Brian Leroux. Mobile application development: Web vs. native. 2011.
- [DB92] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. 1992.
- [DH07] Thomas H. Davenport and Jeanne G. Harris. *Competing on Analytics: The New Science of Winning*. Harvard Business School Press, Boston, MA, USA, 1st edition, 2007.
- [DTS99] Serge Demeyer, Sander Tichelaar, and Patrick Steyaert. Famix 2.0 - the famoos information exchange model. 1999.
- [EK08] Khaled El Emam and A. Güneş Koru. A replicated survey of it software project failures. *IEEE Computer Society*, 2008.
- [ESS92] S.G. Eick, J.L. Steffen, and Jr. Sumner, E.E. Seesoft-a tool for visualizing line oriented software statistics. *Software Engineering, IEEE Transactions on*, 18(11):957–968, 1992.

- [fam] <http://habanero.ifi.uzh.ch/famixmetrics/>.
- [FBB<sup>+</sup>11] Kecia A.M. Ferreira, Mariza A.S. Bigonha, Roberto S. Bigonha, Luiz F.O. Mendes, and Heitor C. Almeida. Identifying thresholds for object-oriented software metrics. *The Journal of Systems and Software*, 2011.
- [FP97] Norman Fenton and Shari Lawrence Pfleeger. *Software metrics (2nd ed.): a rigorous and practical approach*. PWS Publishing Co., Boston, MA, USA, 1997.
- [gar] <http://www.gartner.com/newsroom/id/2408515>.
- [GPS04] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. 2004.
- [hon] <http://developer.android.com/about/versions/android-3.0-highlights.html>.
- [jav] <http://habanero.ifi.uzh.ch/javafamixmetrics/>.
- [KCG10] Jitender Kumar Chhabra and Varun Gupta. A survey of dynamic software metrics. *Journal of Computer Science and Technology*, 25(5):1016–1029, 2010.
- [KH] Souheil Khaddaj and G Horgan. The evaluation of software quality factors in very large information systems.
- [LHG10] Michel Lanza, Lile Hattori, and Anja Guzzi. Supporting collaboration awareness with real-time visualization of development activity. *Conference on Software Maintenance and Reengineering*, 2010.
- [LM06] Michel Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice*. Springer, 2006.
- [McC76] T. J. McCabe. A complexity measure. *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, 1976.
- [Mei12] Reto Meier. *Professional Android 4 Application Development*. Wrox, 2012.
- [Mil98] Everaldo E. Mills. *Software metrics*. 1998.
- [osla] <http://open-services.net/>.
- [oslb] <http://open-services.net/bin/view/main/oslccorespecification>.
- [qdo] <http://qdox.codehaus.org/>.
- [rtc] <https://jazz.net/products/rational-team-concert/>.
- [seo] <http://www.se-on.org/>.
- [Sha00] Mary Shaw. Software engineering education: A roadmap. *Proceeding ICSE '00 Proceedings of the Conference on The Future of Software Engineering*, pages 371 – 380, 2000.
- [sof] <http://www.ifi.uzh.ch/seal/research/tools/sofas.html>.
- [TA08] Thomas Tullis and William Albert. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
- [TS10] Christoph Treude and Margaret-Anne Storey. Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds. 2010.