Journal of Theoretical and Applied Computer Science
ISSN 2299-2634

Vol. 6, No. 1, 2012, pp. 35-45
http://www.jtacs.org

QualitySpy: a framework for monitoring software development processes

Marian Jureczko, Jan Magott
Wrocław University of Technology, Faculty of Electronics, Poland
marian.jureczko@pwr.wroc.pl, jan.magott@pwr.wroc.pl

Abstract: *The growing popularity of highly iterative, agile processes creates increasing need for automated monitoring of the quality of software artifacts, which would be focused on short terms (in the case of eXtreme Programming process iteration can be limited to one week). This paper presents a framework that calculates software metrics and cooperates with development tools (e.g. source version control system and issue tracking system) to describe current state of a software project with regard to its quality. The framework is designed to support high level of automation of data collection and to be useful for researchers as well as for industry. The framework is currently being developed hence the paper reports already implemented features as well as future plans. The first release is scheduled for July.*

Keywords: *framework, quality assurance, software metrics, software engineering*

1. Introduction

This paper presents a framework for monitoring software development process. Failing to deliver requested feature on eligible schedule and quality may cause serious loss (e.g. financial). Therefore, it is important to keep an eye on the work progress and early identify problems. There are a number of tools employed in most of the software development environments. We believe that these tools constitute a valuable source of metrics that can be used to evaluate project current state and predict about the further development. The QualitSpy framework is designed to integrate with the issue tracking system, the version control system, the continuous integration system and the source code itself by collecting raw data as well as software metrics. The collected data is used for research (empirical experimentation, model creation) and for project evaluation.

Before the further discussion some important distinctions should be made. We refer to the information collected by the QualitySpy framework as metrics or as raw data. The term metrics refers to measurements that were conducted on at least nominal scale and describe various attributes of the investigated artifacts (e.g. the sum of modified lines in a certain class in a set of SubVersion revisions) whereas the term raw data is adequate in the case of a direct description of the artifacts (e.g. the content of a file or a class committed in a certain SubVersion revision). Furthermore, we identify two types of metrics. There are product metrics, which describe the state of an artifact on a certain time (e.g. number of lines of code in a certain class) and there are historical metrics (sometimes called process metrics), which refer to the changes in an artifact over a time period (e.g. number of SubVersion revisions committed between two dates).

The rest of the paper is organized as follows: in the next section related works are discussed; the third section presents our motivations; the fourth and fifth sections describes the QualitySpy framework; the fourth one is focused on functionalities whereas the fifth one discusses the framework architecture and the sixth one presents a simple case of the system usage; the final chapter presents future plans, specifically the schedule of forthcoming releases.

2. Related work

There are several tools that are similar to our framework. First of all we would like to mention our own solutions that laid foundation for the QualitySpy framework. These are:

- CKJM extended (*http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/*) – it calculates 19 different product metrics including CK [2], QMood [1] and Tang's [14] metric suites and Efferent Coupling, Afferent Coupling, Lack of COhesion in Methods 3 and Lines Of Code;
- BugInfo (*http://kenai.com/projects/buginfo/*) – it calculates number of defects and six other historical metrics from SubVersion and CVS repositories;
- Metrics Repository (http://purl.org/MarianJureczko/MetricsRepo) – it is a web site where a number of collected metrics are publicly available.

The entire aforementioned tools haven been used in earlier studies [11, 12, 13].

2.1. Software developer perspective

There are tools that are focused on low level measurements (close to source code). Hence, such tools are specifically interesting for developers. The basic set of features of the QualitySpy framework overlaps the main functionalities of tools like Sonar (*http://www.sonarsource.org/*). The main purpose of such tools is to calculate quality related metrics. Most of the metrics concern product (e.g. lines of code), however, there are more sophisticated ones as well. Specifically, there are metrics related to test coverage and code smells (symptoms in the source code that possibly indicate a deeper problem). Both are useful for developers since show possibilities for improving software quality

2.2. Manager perspective

SPR KnowledgePLAN (*http://www.spr.com/spr-knowledgeplanr.html*) is a tool designed to help planning software projects. It supports estimating work, schedule, and defects as well as evaluating project strengths and weaknesses to determine their impact on quality and productivity. The tool is focused on upfront estimations (expressed in figures and Gannt chart) that are based on questionnaire filled by the user. Collecting metrics and monitoring project progress are not supported. Similar features are available in the SEER for Software (*http://www.galorath.com/index.php/products/software/C5/*), although the focus is moved toward statistical methods.

ProjectCodeMeter (*http://ww.projectcodemeter.com/cost_estimation/index.html*) is a tool to measure and estimate the Time, Cost, Complexity, Quality and Maintainability of software projects as well as development team productivity by analyzing their source code. The tool supports a number of estimation models (e.g. WMFP, COCOMO) and collecting metrics from source code.

Software-Cockpit (*http://ww.de.capgemini.com/capgemini/forschung/research/software/*) is focused on moni-

toring software projects in order to early identify potential schedule or quality problems. The tool cooperates with other systems from the development environment (e.g. version control system). Nevertheless, it only slightly overlaps QualitySpy features since Software-Cockpit focus is moved toward the perspective of high level management and hence many important details that regard the employed measurement and evaluation methods are not presented.

The manager perspective is focused on high level project features and goals like the overall cost estimation or return of the investment. Therefore, it is hardly possible to draw conclusions regarding maintainability or quality issues of low level project modules or artifacts (e.g. classes). There is also no room for research since the tools that aims this perspective offer ready to use reports but limited data for user defined analysis.

2.3. Researcher perspective

Unfortunately, the aforementioned tools have limited utility for researchers. Tools from developer perspective deliver us with many interesting metrics; however there is little support for further analysis and the available metrics are almost exclusively calculated from source code (other data sources are usually ignored). One the other hand, the tools from manager perspective implement complex models but in a black-box approach and hence there is no room for self-invented analyses. In consequence the researchers usually write own script to collect the necessary data (e.g. [12, 15]).

Nevertheless, to metric collection process with regard to software development environment has already been considered in several studies. Fisher et al. [6] presented a toolset for collecting data from CVS and Bugzilla that was based on Perl scripts and the wget tool. Furthermore, heuristics for identifying merges and bugfix comments were suggested. The solution was validated on the Mozilla project. Subsequently the authors (in [7]) suggested a method for combining the data collected from CVS and Bugzilla with the program call graph. The aforementioned experiences resulted in a tool called Evolizer (*https://www.evolizer.org/*), which is successfully employed in recent studies e.g. [9]. Methods of extracting data from Bugzilla and CVS were also considered by D'Ambros et al. [3], Zimmerman and WeiBgerber [16] and German [8].

3. Motivation

Software metrics are widely employed in areas such as effort estimation, defect prediction and maintainability assessment. All of them require empirical data that is subsequently transformed to metrics and used to construct models. Unfortunately, the data collection process may be costly and laborious (Fenton and Neil estimated the cost of a software measurement program to be 4% of the development budget [4]). Therefore, it is vital to have good tool support and to limit the costs.

All the three mentioned areas of software engineering (i.e. effort estimation, defect prediction and maintainability assessment) require large data sets in order to construct models with satisfactory level of external validity. The community works on repositories that may satisfy such requirement (e.g. promisedata.org). Nevertheless, the available data is not uniform since different contributors choose different metrics.

Recent studies showed great value of process and historical metrics [5, 10, 11]. Such metrics can be usually extracted from issue tracking systems and version control systems. Unfortunately, the collection process is difficult and has unsatisfactory tool support. The researchers usually build their own solutions, e.g. [15]. Furthermore, there are no so called

industrial standard for the process and historical metrics. Therefore, different researchers use slightly different metric definitions and hence the results are difficult to compare and to replicate on other data sets. It could be also challenging to validate the results since even when the input data is published it may be not obvious how it was extracted from the source code or other artifacts. The publicly available data sets contain already calculated metrics and the mapping on software processes and artifacts may be vague (This doubt does not regard metrics definitions but missing details. E.g. let's assume that a researcher is investigating the number of distinct authors per Java class. The researcher says that he is investigating project P in version V and that he is collecting the number of distinct authors from SubVersion repository. The situation seems to be clear… However, what if there are two branches in the SubVersion repository? Did the researcher investigate both? Moreover, it is possible that some changes have been made when the investigation was finished. Specifically the source code visibility could be changed, e.g. a new branch is added to the public domain or the whole repository is shut down. In such case it may be not possible to recreate the original state.)

The QualitySpy framework attempts to address the aforementioned issues. The framework is designed to collect raw data and to allow the user (e.g. researcher) to define metrics. Therefore, it will be possible to collect uniform data (i.e. the raw data; it will be a detailed copy of project state and hence when the project evolves it is not necessary to recreate previous state for investigation purposes since the previous state is stored in QualitySpy repository) define various metrics on top of them and conduct a variety of experiments. Furthermore, the results could be compare among different experiments (even experiments that use different metrics) since the data source (the raw data) will be in the same format. The QualitySpy approach increases external validity of experiments by providing uniform data. Let's assume that there is a researcher that investigates a certain phenomenon and he collects empirical data in order to conduct an experiment. Since the collected data is stored in uniform format another researcher that is investigating another phenomenon (maybe using different metrics) can employ the already collected data in his experiment to extend the external validity.

There are two main goals for the QualitySpy framework. The more important one (mentioned above) regards collecting uniform data for research. The second one is connected with industry. The QualitySpy framework should be in assistance for assuring quality in software development processes as well. The tool will cooperate with most common development tools and systems in order to deliver valuable information with regard to project estimation and planning or software quality and maintainability. Nevertheless, we believe that accomplishing the second goal requires further research. Therefore, the first release of QualitySpy corresponds only with the first goal. The first version will be used to collect empirical data. Further investigation (based on the collected data) will show how the data can be effectively used by industry. The investigation results will be employed to define reports with regard to different project aspects.

4. Features

There are two main groups of features. The first is designed for data acquisition when the later for data analysis and reporting (Fig. 1).

4.1. Acquisition

The acquisition related features are encapsulated in an user interface where the configuration for all connectors can be set. The user can define several different configurations; specifically there can be many software projects under investigation and a separate configuration for each of them.

Using the connectors the user can collect data from different sources:

- software metrics are calculated from Java classes,
- raw data regarding history of the source code is collected from source version control system (e.g SubVersion),
- raw data regarding project history is extracted from the issue tracking system (e.g. Jira),
- raw data regarding project history is collected from continuous integration system (e.g. Hudson).

Three of the four aforementioned connectors collect raw data. It means that the data is stored in a textual form without transformation into metrics. However, later the user has the possibility to define metrics on top of the raw data using the reporting interface.

All the collected data is stored in a central repository and is available for further investigation. Specifically, the collected data will be available through the acquisitor's interface.

4.2. Reporting

The framework offers a publicly available web interface. All the collected data is available through the interface. Furthermore, it will be also possible to define new metrics on top of the available data and generate reports that are based on these metrics. There will be a set of predefined report templates (based on research results), however the user will have the possibility to define his own. The reporting interface of the QualitSpy first release will be mainly used for research. We are going to formulate the research results into report templates that will be used to reflect current state and estimations for the software project. The reports are scheduled for one of the future releases.

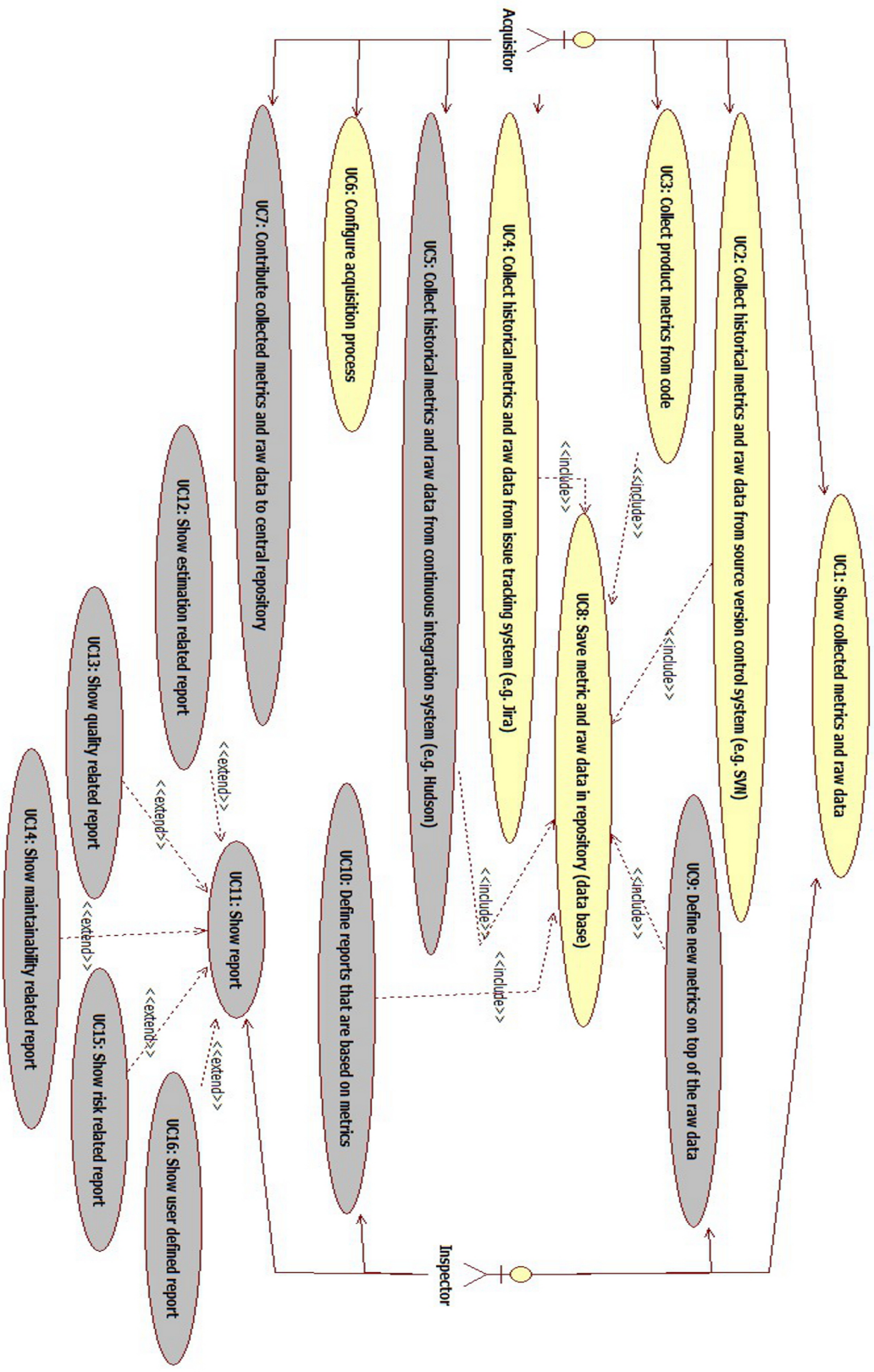

Figure 1. Use Case diagram. Use cases with grey background are scheduled for second release

5. Architecture

The QualitySpy framework consists of several modules. There is a central repository with two user interfaces one for data acquisition and one for reports (Fig. 2.) and a set of connectors, which are designed to collaborate with development tools and artifacts in order to collect the data.

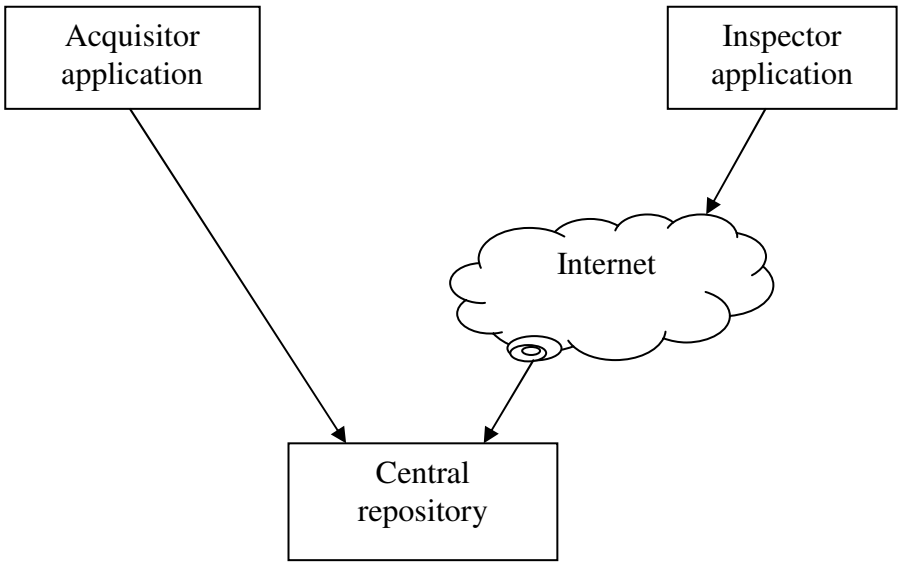

Figure 2. QualitySpy high level architecture

5.1. Repository

The repository is built on a relational database, where all the collected data is stored. There is an interface for data acquisition where the user can configure how the framework should collaborate with the development tools to collect necessary data. There is also an independent web interface that gives access to the collected data through internet. The later one is designed to replace our earlier service, i.e. Metrics Repository (http://purl.org/MarianJureczko/MetricsRepo).

5.2. Issue tracker connector

The issue tracker connector is designed to collect data from an issue tracker called Jira (*http://www.atlassian.com/software/jira*). We are going to develop support for other issue trackers (Bugzilla, Mantis and Redmine) in the future. Currently it is possible to connect with a Jira running instance and collect detailed data about selected issues (including history). QualitySpy collects data through the user interface (web page) using Selenium. Therefore the process does not depend on Jira configuration (i.e. Jira CLI or Jira Web Service which could be disabled).

5.3. Source version controller connector

The source version controller connector is designed to collect data from one of the most commonly used version control systems, namely SubVersion (http://subversion.tigris.org). Support for CVS and GIT will be implemented in one of the next versions. This connector is

derived from our earlier tool – BugInfo (*http://kenai.com/projects/buginfo*). It is possible to connect with a SubVersion server and read detailed data about specified revisions (modified files and modified lines within those files, authors of modifications…). Specifically, the connector can identify which files represent Java classes. Therefore, the collected data can be easily combined with data obtained from other connectors.

5.4. Metrics connector

The metric connector wraps the CKJM extended tool (*http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/*) in order to calculate software metrics form the byte code of compiled Java classes. The connector supports all the 19 metrics calculated by CKJM extended and connects them with certain Java classes and methods.

5.6. Continuous integration connector

The continuous integration connector is scheduled for the second release. The connector is designed to collaborate with Hudson continuous integration system (*http://hudson-ci.org/*) and to collect detailed data regarding build successes and failures as well as other available data. The scope of available data depends on Hudson configuration. Most common are test results. However, others (e.g. test coverage) are sometimes available as well. We are considering integration with other continuous integration systems, e.g. Atlassian Bamboo, Apache Continuum, CruiseControl.

5.7. Reporting module

The collected data and reports are available in the reporting module. We are going to implement some predefined report templates according to research results. However, the tool is intended to be flexible and hence the user could define own templates as well. The reports will be based on metrics and the metrics will be defined on top of the raw data. We are going to suggest a dedicated language (or an extension to existing one) that will be used by the user to define metrics and report templates. Report is an instance of report template that is filled with a combination of metric values.

The reporting module is implemented as light web client, which communicate with the server (central repository) using Representational State Transfer (REST). This module is operated through a web browser and hence it will become a part of the public domain once it has been deployed on an application container.

6. Instead of evaluation

This paper deadline is a couple weeks before QualitySpy release and hence some refactorization and integration activities are still in progress. In consequence, it is not possible to conduct a formal evaluation. Instead a simple case of the system usage is presented.

QualitySpy was executed on the version control system of the Apache Ivy project (*http://ant.apache.org/ivy/*) with following settings:

- SubVersion repository url: http://svn.apache.org/repos/asf/ant/ivy/core/trunk/
- Java classes prefix (necessary to identify whether a file represents the object of study e.g. a Java class and to cut off this part of file path that does not belong to the Java packages): /(ant|incubator)/ivy/(core/)?trunk/src/java/
- Java classes postfix: .java
- start revision: 734618

- end revision: 811778

In result an amount of data was collected and stored in the repository. For the sake of readability we present only a small subset of those data, namely description of revision number 737330 (Tab. 1). The content of Tab. 1. slightly differ from the real commit number 737330. Four files from this commit are not present in the table. Those are a xml and a txt file and two JUnit test classes located in the directory test/java. Those files were excluded since they do not correspond with the given Java classes prefix and postfix.

Table 1. Data collected with regard to Apache Ivy revision 737330

Revision number	Date	Author	Comment	Added Lines	Removed Lines	Tags	Class name	New content	Old content
737330	2009-01-24 12:00:40	xavier	FIX: TTL does not work as expected (IVY-1012)	2	6		org.apache.ivy.core.cache.DefaultRepositoryCacheManager	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....
737330	2009-01-24 12:00:40	xavier	FIX: TTL does not work as expected (IVY-1012)	3	3		org.apache.ivy.plugins.resolver.BasicResolver	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....
737330	2009-01-24 12:00:40	xavier	FIX: TTL does not work as expected (IVY-1012)	8	0		org.apache.ivy.core.cache.RepositoryCacheManager	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....
737330	2009-01-24 12:00:40	xavier	FIX: TTL does not work as expected (IVY-1012)	7	0		org.apache.ivy.plugins.resolver.AbstractResolver	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....	/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file.....

The collected data consist of:

- Revision number – the revision number that is used in the SubVersion repository.
- Date – date of the commit, when the revision was committed.
- Author – identifier of the developer that committed the revision.

- Comment – the commentary that was assigned to the commit.
- Added lines – number of lines that were added to the committed file (Java class).
- Removed lines – number of lines that were removed from the committed file (Java class).
- Tags – the tags from version control system that are assigned to the revision.
- New content – the whole content of the committed file; Tab.1 presents only the beginning of the file.
- Old content – the whole content of the previous state of the committed file (i.e. the state before the commit); Tab.1 presents only the beginning of the file; Old content together with New content may be used to track down every single change in the committed file.

7. Schedule and future work

The QualitySpy framework is developed by undergraduate students of computer science on the Faculty of Electronics. The students created a 16-member team divided into 4 subteams. They were working during one semester. The software development process is based on Scrum. The first release will be made after two semesters of development (there is a new group of students for the second semester) and is scheduled for July 2012.

The first release contains only the basic features regarding data collection, namely integration with CKJM extended (software metrics), version control system (i.e. SubVersion) and issue tracking system (i.e Jira). There is a database where all the collected data is stored and very simple reporting module that gives access to the collected data in a read only mode. All the other mentioned features are scheduled for future releases. Specifically, we are going to use the first release to collect data for experiments that will constitute solid basis for the reporting module.

The first release as well as all the forthcoming will be available through project web page: http://purl.org/MarianJureczko/QualitySpy. We believe that we will be able to make a release each year. In consequence, the second release is scheduled for July 2013.

Acknowledgement

We would like to thank to all the students that were involved in the QualitySpy development process. A complete list of all the students is available online: http://purl.org/MarianJureczko/QualitySpy/Acknowledgement.

References

[1] Bansiya J. and Davis C.G. *A Hierarchical Model for Object-Oriented Design Quality Assessment.* IEEE Transactions on Software Engineering, No 1(28), 2002, 4-17.

[2] Chidamber S. and Kemerer C.. *A metrics suite for object oriented design.* Software Engineering, IEEE Transactions on. No 20(6), 1994, 476–493.

[3] D'Ambros M., Lanza M., Pinzger M. *"A Bug's Life" Visualizing a Bug Database*, 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, 113-120.

[4] Fenton N. and Neil M. *Software Metrics: Successes, Failures and New Directions.* Journal of Systems and Software, No 47(2-3), 1999, 149-157.

[5] Fenton N., Neil M., Marsh W., Hearty P., Radliński Ł and Krause P. *Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction*. Proceedings of the 29th International Conference on Software Engineering Workshops. 2007.
[6] Fischer M., Pinzger M. and Gall H. *Populating a Release History Database from Version Control and Bug Tracking Systems.* International Conference on Software Maintenance (ICSM '03). Washington, DC, USA, 2003.
[7] Fischer M., Pinzger M. and Gall H. *Analyzing and Relating Bug Report Data for Feature Tracking.* 10th Working Conference on Reverse Engineering (WCRE '03). Washington, DC, USA, 2003.
[8] German D.M. Mining CVS repositories, the softChange experience. IEE Seminar Digest. No 917. 2004. 17-21.
[9] Giger E., Pinzger M, and Gall H. *Using the gini coefficient for bug prediction in eclipse.* 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution (IWPSE-EVOL '11). New York, USA, 2011.
[10] Illes-Seifert T. and Paech B. *Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs*. Information and Software Technology. No 52(5), 2010, 539-558.
[11] Jureczko M. *Significance of Different Software Metrics in Defect Prediction*. Software Engineeering: An International Journal. No 1(1), 2011, 86-95.
[12] Jureczko M. and Madeyski L. *Towards identifying software project clusters with regard to defect prediction*. Sixth International Conference on Predictive Models in Software Engineering, Timisoara, Romania, September 12-13, 2010.
[13] Jureczko M. and Spinellis D. *Using object-oriented design metrics to predict software defects*. Fifth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX 2010. s. 69-81.
[14] Tang M.-H., Kao M.-H. and Chen M.-H. *An Empirical Study on Object-Oriented Metrics*. Sixth IEEE International Symposium on Software Metrics. 1999.
[15] Weiss C., Premraj R., Zimmermann T., and Zeller A. How Long Will It Take to Fix This Bug? Proceedings of the 4th International Workshop on Mining Software Repositories. 2007.
[16] Zimmermann T. and WeiBgerber P. *Preprocessing CVS data for fine-grained analysis.* IEE Seminar Digests. No 917. 2004. 2-6.