QualiPSo

Quality Platform for Open Source Software

IST-FP6-IP-034763

Working document 5.5.4, V6.0

User guides for the WP5.5 toolset

Anja Hentschel (SIE)
Klaus P. Berg (SIE)
Vieri del Bianco (INS)
Davide Taibi (INS)
Davide Tosi (INS)
Tomasz Krysztofiak (PNSC)
Davide dalle Carbonare (ENG)
Hongbo Xu (GMRC)
Jonathan Pares (GMRC)
Auri Vincenzi (USP)
Paulo Meirelles (USP)

Due date of deliverable: 31/10/2010

Actual submission date: 31/01/2011

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 License.

To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA

creative commons

Change History

Version	Date	Status	Author (Partner)	Description
0.1	16/08/07	Draft	Anja Hentschel (SIE)	Document structure and initial content
0.2	22/10/07	Draft	Anja Hentschel (SIE)	Input from Max. Almost everything completely restructured.
0.3	31/10/07	Draft	Anja Hentschel (SIE)	Input from Marion and Abbas.
0.4	13/06/08	Draft	Anja Hentschel (SIE), Klaus Berg (SIE), Vieri del Bianco (INS)	New chapters: Testing (KB), User stories (VdB), Implementation concept (AH) Removed: description of evaluation methods
1.0	30/07/08	Release	Anja Hentschel (SIE)	
1.1	03/11/08	Draft	All	New chapters: Tool descriptions, Integration, Experiences and related appendices
1.2	07/11/08	Draft	Anja Hentschel (SIE)	Some remaining gaps filled
1.3	28/11/08	Draft	Anja Hentschel (SIE)	Input from reviews included; gaps filled
2.0	10/12/08	Release	Anja Hentschel (SIE)	
2.1	16/03/09	Draft	Anja Hentschel (SIE)	New chapter: Quality Assurance
2.2	24/03/09	Draft	Anja Hentschel (SIE)	Chapter “Tool descriptions”: template for extending the tool specific user guides
2.3	19/05/09	Draft	Anja Hentschel (SIE)	Input from reviews included
3.0	22/05/09	Release	Anja Hentschel (SIE)	
3.1	28/09/09	Draft	Anja Hentschel (SIE)	Split and update of section “Tool descriptions”; update of sections “Executive summary”, “Introduction”, “Evaluation related tasks”,

creative commons

				“User Stories”, “Quality Assurance”, “Experiences”; section “Requirements” removed
3.2		Draft	Anja Hentschel (SIE)	Input from reviews included; update of sections “Tool descriptions” and “Integration”
4.0				
5.0	31/07/10	Release		User Guide Update
6.0	31/10/10	Release		User Guide Update
7.0	29/11/10		Davide Dalle Carbonare (ENG)	Removed Spago4Q 1.0 first integration “Analysis and assessment tools” update
7.1	01/12/10		Jonathan Pares (GMRC)	Updated the description of the JUnit tool. Replaced Fossology by OSLC. Changed the figure of the integration of the tools with the Factory. Added a section describing the Spago4Q integration with the QualiPSo Factory.
7.1	02/12/10		Paulo Meirelles (USP)	Updated the description of Kalibro Metrics tool
7.1	20/12/10		Jonathan Pares (GMRC)	Updated the description of the QualiPSo factory and its architecture. Moved the factory/Spago4Q integration in the section integration.

creative commons

Executive Summary

The tools developed in work package 5.5 aim at making the evaluation of OSS trustworthiness as easy and seamless as possible for developers, end users, and managers. Specifically, the tools intend to automate as many activities as possible and provide developers, users, and managers with information that is reliable and useful for making decisions involving trust in OSS.

The A5 tool set is composed of a number of OSS tools that support the creation of a measurement plan, starting from the main actors and stakeholders objectives and goals (developers' community, user community, business needs, specific users, etc.), down to the specific static and dynamic metrics that will need to be collected to fulfill the goals.

This document contains the user guides for the WP 5.5 tool set. It describes the context in which the A5 tools work and how they can be used for trust evaluation. The term “user guide” is to be interpreted in a very broad sense in this context, since the user guide is intended to

- outline the environment in which the tools will be used
- collect requirements, paying attention to different user roles
- describe the tools belonging to the tools set and their interaction.

As a first step to describing the context of the A5 tools, a typical tool selection process is outlined. It starts with activities like “Searching a suitable OS product,” goes on with activities like “Evaluation,” “Get clearance” and “Test,” and ends with “Adaptation” and “Get approval.” The main focus of the A5 work is on Evaluation and Test.

Activity “Evaluation” is split in the following four tasks: data acquisition, data integration, evaluation, and usage. In turn, each task is broken down to several sub-tasks which help identify the requirements for the needed tools. In a complementary approach, a set of user stories is collected which help identify tool requirements from a different angle and with a finer granularity.

Though the data gained by activity “Test” are similar to other evaluation data, there are some special aspects to consider. Unlike others, test-related measures can often not be obtained by simply running a measurement tool or looking into a repository. In unfavourable cases, it may even be necessary to create new test suites before anything can be measured at all. Therefore, it is necessary to investigate which types of tests are needed (functional, non-functional, interoperability testing) and how measurement can be done in the testing context.

The A5 tool set includes tools from external providers as well as tools developed by the partners. The choice of the testing tools is inspired to the concepts investigated in Task 5.4.2 and the choice of the measurement tools is based on the trustworthiness factors identified in WD 5.3.2 [3].

- JaBUTi and JUnit provide information about testing
- PMD, Checkstyle, MacXim, and Kalibro are code analysis tools that deliver information related to maintainability and reliability

creative commons

- StatCVS and StatSVN provide data on the development process of the OSS product
- OSLC collects data on licensing and intellectual properties
- GQMTool supports the definition of a Goal-Question-Metric model
- The Spago4Q platform is used to collect the data coming from these sources and present the data according to the GQM defined in WP5.3.
- Quality platform provides an easy way to access to the quality report and to analyze new projects.

All tools have been fully integrated into Spago4Q while OSLC and JUnit were partially integrated.

Spago4Q has been partially integrated into Qualipso Factory as proof of concept.

To ensure the quality of the A5 tools, a quality concept has been introduced and used.

creative commons

Table of Contents

1. INTRODUCTION 10
2. SELECTION PROCESS 11
3. EVALUATION-RELATED TASKS 13
3.1 Data Acquisition 13
3.2 Data Integration 14
3.3 OSS evaluation 14
3.4 Result usage 14
4. TESTING 16
4.1 Test types 16
4.1.1 Functional Testing 16
4.1.2 Non-functional Testing 16
4.1.3 Interoperability Testing 17
4.2 Measurement in the testing context 17
5. USER STORIES 19
5.1 Legend 19
5.2 User Story – General context 19
5.2.1 Evaluation of the trustworthiness of a product 19
5.3 List of user stories 20
5.3.1 Insert new project 20
5.3.2 Get OSS Project list 20
5.3.3 Start the toolset 20
5.3.4 Change role after login 21
5.3.5 Get OSS overview information if available 21
5.3.6 Update OSS information 21
5.3.7 Search for relevant information of a product 21
5.3.8 Analyze presented information 22
5.3.9 Define a search and analyze profile 22
5.3.10 Create trustworthiness metrics information 22
5.3.11 Create and execute trustworthiness test-based information 22
5.3.12 Execute trustworthiness test-based information 22
5.3.13 Create and execute trustworthiness dynamic measures based information 23
5.3.14 Get quality and trustworthiness trend and history information 23
5.3.15 Get recommendations to improve trustworthiness 23
5.3.16 Quick compare 23

creative commons

5.3.17 Deep compare ... 24
6. TESTING TOOLS ... 25
6.1 JaBUTi ... 25
6.1.1 Installation (Desktop GUI) ... 26
6.1.2 Installation (Service) ... 27
6.1.3 Tool Usage (Desktop) ... 28
6.1.4 Tool Usage (Service GUI) ... 30
6.1.5 Integration with Spago4Q ... 31
6.2 JUnit Statistics Tool ... 31
6.2.1 Available metrics ... 31
6.2.2 Installation ... 32
7. CODE ANALYSIS TOOL ... 34
7.1 Macxim ... 35
7.1.1 External Tools integrated into Macxim ... 36
PMD ... 36
CHECKSTYLE ... 36
7.1.2 The available metrics ... 37
7.1.3 Installation ... 39
7.1.4 Tool Usage ... 43
7.1.5 MacXim Spago4Q Integration. ... 54
7.2 Kalibro Metrics ... 55
7.2.1 Installation ... 58
Installation prerequisites ... 58
Kalibro Service ... 58
Spago4Q integration ... 59
Kalibro Desktop – Quick install and usage ... 59
7.2.2 Tool Usage ... 63
Kalibro Desktop ... 63
Settings ... 70
7.2.3 Developer Guide ... 71
7.3 OSLC ... 74
7.3.1 Available metrics ... 74
7.3.2 Installation ... 75
8. ANALYSIS OF PROJECT DEVELOPMENT ... 77
8.1 StatCVS and StatSVN ... 77
8.1.1 Installation ... 78
8.1.2 Tool Usage ... 78
9. ANALYSIS AND ASSESSMENT TOOLS ... 80
9.1 Spago4Q ... 80
9.2 Spago4Q Extension for QualiPSo ... 80

creative commons

Spago4Q-QualiPSo bundle 1.2 81
9.2.1 Installation 81
9.2.2 Tool Usage 82
9.3 Quality Platform 83
9.3.1 Installation 83
9.3.2 Tool Usage 84
9.4 GQMTool 86
10. INTEGRATION 88
10.1.1 Integration mechanism with Spago4Q 2.x 88
10.1.2 Toolset and Spago4Q 2.x Integration process 89
10.2 Example of tool integration in Spago4Q 2.3 90
10.3 Integration of the A5 tools into the QualiPSo Factory 93
11. QUALITY ASSURANCE 95
11.1 Software quality overview 95
11.1.1 Coding Conventions 97
11.1.2 Testing Activities 98
11.2 Coding conventions for the A5 toolset 100
11.2.1 Naming conventions 100
11.2.2 Coding style 101
11.2.3 Documentation 101
11.2.4 Declaration and definition issues 102
11.2.5 Imports 103
11.2.6 Size issues 104
11.2.7 Possible coding problems 105
11.2.8 Error handling 107
11.2.9 Design issues 107
11.2.10 Redundant Code 109
11.3 Applying the A5 coding conventions 109
11.3.1 Using Checkstyle 110
11.4 Testing of A5 tools 113
11.5 Quality of the “external” tools 114
12. CONCLUSION AND FUTURE WORK 117
13. REFERENCES 118
APPENDIX A– OVERVIEW ON TESTING 121
Functional Testing 122
The importance of test coverage in functional testing 122
Non-Functional Testing 123
Performance and load testing 124

creative commons

Creating accurate test data 124
Reliability testing 125
Acceptance testing 126
Compatibility, conformance and interoperability testing 126
OSS external functional and non-functional testing 126
Analyzing quantity and quality of test cases 126
APPENDIXB – DESCRIPTION OF SPAGO4Q **128**
APPENDIX C- INFORMATION THAT CAN BE EXTRACTED FROM JABUTI **132**
Testing Coverage Criteria 132
OO Metrics 133
APPENDIXD – QUALITY ASSURANCE **135**
Syd Chapmans Additional Coding Guidelines 135
Using Checkstyle scripts for code analysis 137
Using Ant 137
Using Maven 137

creative commons

1. INTRODUCTION

This document is the user guide for the WP 5.5 tool set. The term “user guide” is to be interpreted very broadly in this context, since the user guide is intended to

- Outline the environment in which the tools will be used
- Collect requirements paying attention on different user roles
- Describe the tools belonging to the tools set and their interaction.

Section 2 of this document outlines a selection process of an OSS product with the intention to identify the context for the evaluation activities. The description is based on the answers in the questionnaires gathered in WP 5.1 [1].

Section 3 focuses on activity “Evaluation” identified in the preceding chapter and gives an outline of the tasks typically related to an evaluation. In a similar way, Section 4 gives an outline of different kinds of techniques used in activity “Testing.”

In Section 5, a set of user stories is collected which will help to identify more detailed requirements for the tools which will be developed or integrated in A5.

Sections 6 though 9 provide short descriptions of the tools which are part of the A5 tool set and give an idea why these tools have been chosen. For some of the tools more detailed information is given in separate user guides. Section 6 focuses on testing tools, Section 7 on code analysis tools, and Section 8 on project analysis tools, and Section 9 on measurement program analysis and assessment tools.

Section 10 presents three ways in which the tools can be and are integrated and shows some of the resulting consequences.

Section 11 describes the measures which are taken to ensure the quality of the tools provide within the A5 toolset.

Section 12 concludes this document and outlines some possible future work. More details on the content of this document can be found in the Appendices.

creative commons

2. SELECTION PROCESS

This section outlines a selection process of an OSS product with the intention to set up the context for the evaluation activities. The description is based on the answers in the questionnaires gathered in WP 5.1 [1].

The questionnaires showed that most interviewees use informal ways to select an OSS product. Nevertheless in some organizations more formalized selection processes are used like the one in Figure 1. The ovals stand for activities which are included in the selection process. The activities depicted here are typical for formal and informal processes though not all activities exist in all environments and their order may be different. For instance, testing may be the first step after having found a potentially fitting product.

In any case, the answers of the interviewees usually do not describe the actors involved in the individual selection activities. This is especially true for evaluation tasks which could be executed e.g. by developers, QA people, clearing houses, project leaders and others.

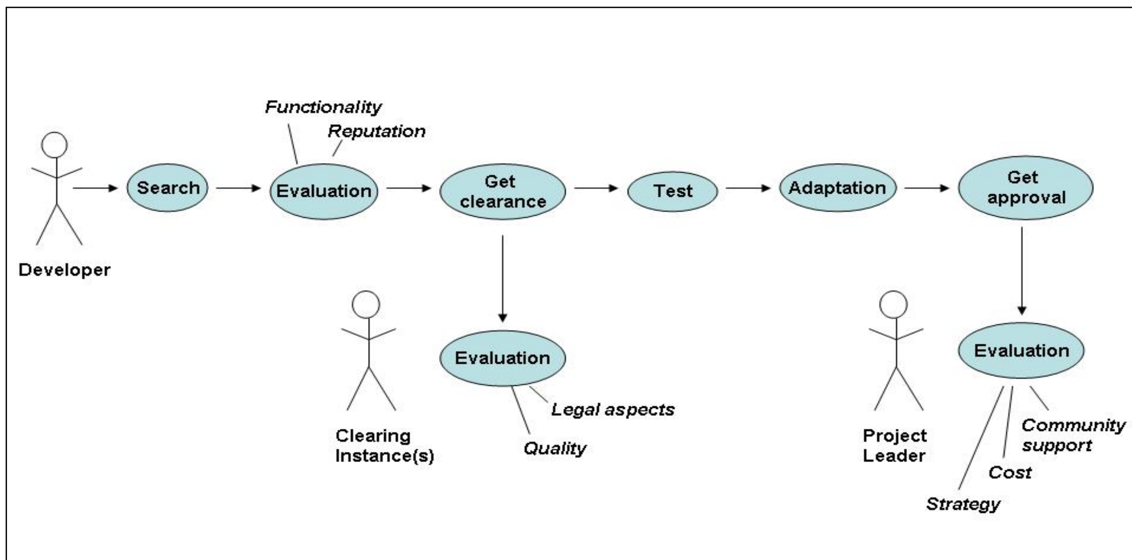


Figure 1: Example of a formalized selection process

The repeated “Evaluation” oval indicates that this activity can be executed several times by different roles and with different focus (see below). The attached terms in italics represent example topics for such a focus.

The selection process includes following activities:

- *Search:* Searching a suitable OSS product is done based on references, search engines and many other means.
- *Evaluation*: Different roles execute different kinds of evaluations in different stages of the selection process. While a developer may be mainly concerned by functionality and the reputation of the OSS provider, the project leader may be more interested in information about the long-term strategy or resulting costs whereas the legal department may concentrate its attention to patent and license issues. As a consequence

creative commons

support for evaluation must enable all stakeholders to answer their specific questions in an efficient way.

Based on the questionnaires we collected in WP 5.1, no systematic "evaluation process" can be identified in the way trustworthiness factors as those identified in wd5.3.1 [3] are assessed in a typical order. Nevertheless in most cases the following aspects are covered, though different kinds of factors are used to explore each of these aspects:

- Functionality
  - Implementation of the requirements (does the product meet the requirements?)
  - Comparison to other products
- Legal issues
  - Is the licensing appropriate for the intended use?
  - IPR – are there any patents infringed by the intended use?
- Non-functional aspects e.g.
  - Reputation of the provider
  - Community support
  - Long-term strategy (e.g. sponsor, roadmap, used technology, standard compliance…)
  - Cost
  - Quality
  - In-house knowledge

- *Get clearance/approval*: In most environments, a second or even third party (apart from the developer) must endorse the use of OSS in a product. Though sometimes a list of acknowledged OSS products exists, which is maintained independently from individual projects, often other people have to be involved, e.g.
  - the project leader who is responsible for the resulting product
  - the legal department
  - an internal clearing centre
  - senior technical experts
- *Test*: Though "testing the OSS product" is mentioned in many answers, few details are given about what is tested and how this is done. Testing mainly seems to be a means to either assess the OSS products functionalities or prove its quality. The last reason has higher importance if the product or its community are rather new. Section 4 contains a more detailed description of this activity.
- *Adaptation*: This activity includes modification and customization of the OSS product

creative commons

3. EVALUATION-RELATED TASKS

This section focuses on the activity "Evaluation" identified in Section 2. It gives an outline of the tasks typically related to OSS evaluation and associated activities.

The description of these tasks has been used to support the identification of the requirements for the tools that will be developed within the QualiPSo project.

These tasks are strongly related to the user stories (see Section 5) which are used to identify concrete requirements for the tools mentioned above from a different point of view.

3.1 Data Acquisition

This task consists of gathering all the facts needed as input for an evaluation. The significant evaluation factors identified in WP 5.3 [38]will be the basis for this task.

The following sub-tasks help to fulfill the task:

- *Collection*: OSS product information that can be used directly is collected from different sources. This can be either raw or pre-processed data like existing evaluation reports.
- *Measurement*: Input from the sources is measured to obtain the required data e.g., measures based on the code like quality measures or based on the mining of community resources like bug repositories.
- *Execution*: Execute the code to gain insights on performance, resource consumption etc.

There is a broad range of resources that can serve as data source, e.g.:

- Code (to be analyzed statically and dynamically)
- Project information (meta info about the product)
- Bug repositories
- Documents (e.g. release notes, FAQs)
- Communities (forums...)
- User reports (like e.g. at the Debian site[1])
- Information on tests (test cases, test software/infrastructure, test results)

Tools supporting data acquisition are developed in WP 5.5.1 and WP 5.5.2.

[1] Debian home: http://www.debian.org/

creative commons

3.2 Data Integration

This task is responsible for integrating the data collected in the previous task. This comprises sub-tasks like

- *Storage*: The data collected in the previous task need to be stored adequately. The structure of the repository must meet the needs of a broad range of requirements. The data can be, e.g., raw data, aggregated data or results of evaluations.
- *Aggregation*, *Combination*: For certain purposes, it is useful to aggregate and combine the collected data to obtain additional or more meaningful data. This applies to more complex trustworthiness factors [38], for instance.
- *Versioning*: Data on different versions of an OSS product can be maintained.

3.3 OSS evaluation

The actual evaluation is done in this task. The requirements of the users and their projects have to be matched with the available data as e.g. defined in [4]. The following sub-tasks help to do this:

- *Filtering*: Filters can be defined to enable focusing on aspects which are relevant for the user.
- *Prioritization/Weighting*: There are means which allow considering aspects of different importance for the user (e.g. mandatory and optional aspects).
- *Comparison*: The user is supported in comparing measurement data of a certain product with those of others (benchmark).
- *Assessment*: Different mechanisms are used to calculate a final evaluation result.

Tools supporting evaluation are developed in WP 5.5.2 and 5.5.3.

3.4 Result usage

This task deals with the usage of the results obtained in the previous tasks. This comprises the following sub-tasks:

- *Visualization*: The various kinds of data are presented in an appropriate form: graphs, statistics, numbers, plain text, reports, aggregated data etc.
- *Zooming*: The user can zoom into the presentation of the evaluation results to get more information on the data it was based upon.
- *Reporting*: Evaluation reports can be created and tailored to the specific needs of the project and the role of the user.

creative commons

- *Annotation*: Existing evaluation reports can be supplemented by additional information, e.g. with regard to other versions of the analyzed OSS product or based on project experiences.
- *Export*: Evaluation reports and the data they rely on can be exported for use in other environments.

Tools supporting result usage are developed in WP 5.5.3.

creative commons

4. TESTING

This section outlines the two different aspects of testing which are related to the selection process described in Section 2. The first aspect concerns the activity “Test” (see Figure 1) as a way to assess the OSS products functionalities or to prove its quality. The second aspect considers testing mainly for measurement purposes, to deliver input data for activity “Evaluation”. Both aspects are closely intertwined.

Since the answers of the interviewees collected in WP5.1.1 [1] remain rather fuzzy with regard to the concrete implementation of activity “Testing,” the content of this section is mainly based on references from the literature and the Internet.

In contrast to others, test-related measures can often not be obtained by just running a measurement tool or collecting data directly from a repository. In good cases, running existing test suites is enough to create test results, coverage data, etc. In bad cases, it is necessary to create test suites for all kinds of tests the user is interested in before anything can be measured at all.

The different kinds of possible tests are a key factor for determining the necessary effort to be spent for creating them. This test type has influence on the complexity of the development of the test cases and the level of knowledge a user must have about the system to be able to develop them.

4.1 Test types

When focusing on activity “Test” as a way to assess the OSS products functionalities or to prove its quality, the following types of tests are typically used (see Appendix A for a detailed description). In spite of the focus the test results may be used as well as input for activity “Evaluation”.

4.1.1 Functional Testing

*Functional Testing*checks if the system implements the required functionality. It consists of

- *Unit testing* verifies the functioning in isolation of software pieces which are separately testable. Depending on the context, these could be the individual subprograms or a larger component made of tightly related units.
- *Integration testing* is the process of verifying the interaction between software components.
- *System testing* is concerned with the behaviour of a whole system.

4.1.2 Non-functional Testing

Non-functional testing comprises tests which compare the system to the non-functional requirements, such as performance (speed), security, usability, and reliability.

creative commons

- *Performance testing* is carried out to determine how fast some aspect of a system performs under a particular workload. It can also be used for validating and verifying other quality attributes of the system, such as scalability, reliability and resource usage. Though it can also measure what parts of the system or workload cause the system to perform badly, the focus in the QualiPSo context will be on the formerly mentioned evaluation aspect.
- *Load testing,* which could as well be considered a kind of performance testing, is usually defined as the process of exercising the system under test by feeding it the largest tasks it can operate with. In this context, it is necessary to emphasize the importance of having large datasets available for testing properly.
- *Reliability testing* (including stress testing and robustness testing) tries to break the system under test by overwhelming its resources or by taking resources away from it.
- *Acceptance testing* is a kind of black box testing, which provides the client with the opportunity to verify the system functionality and usability prior to the system being moved to production. So, acceptance testing can cover functional as well as non-functional aspects.

4.1.3 Interoperability Testing

The following testing types deal with the interoperability of OSS products at different levels:

- *Compatibility Testing*: These tests are concerned with verifying how well software performs in a particular hardware/software/operating system/network/etc.
- *Conformance Testing*: These tests are performed to determine whether a system meets a specified standard.
- *Interoperability Testing*: Interoperability testing is the act of determining if end-to-end functionality between (at least) two communicating systems performs as required by those systems' base standards.

These testing types are covered by the work done in work package WP 3.1 and are so far not planned to be included in the A5 tool set.

4.2 Measurement in the testing context

The measurements in the testing context are used as input for the activity “Evaluation” and concern mainly the following aspects:

- *Code coverage* describes the degree to which the source code of a program has been tested. It is a form of testing that looks at the code directly and as such comes under the heading of white or glass box testing. (Appendix A for more detailed information on code coverage, especially on coverage levels.)
- The *quantity of test cases* is the absolute number of all test cases and test procedures respectively test scenarios. This number should be

creative commons

related to the number of test cases needed to reach a defined test coverage level.

- The *quality of the test cases* can be measured based on attributes like impacted functions, reusability, effectiveness and completeness of test cases.

creative commons

5. USER STORIES

The general vision for using the A5 tools is that the user logs into the QualiPSo factory and either uses the available evaluation reports on the identified product or does his or her own evaluation with the aid of the tools provided by the QualiPSo factory (see later sections for the tools available at the moment and for their integration into the factory). If the tools are used inside the factory, the collected and newly created evaluation-related data can be stored in the associated knowledge base (e.g. in the Spago4Q data warehouse or the project repositories of the factory).

To break that down these user stories give an overview on the desired functionalities of the A5 tools without going into detail with regard to the tool necessary to implement it. Similar to the task descriptions in the preceding section, the user stories have been used to identify concrete requirements and potential issues to enable efficient development of the A5 tools.

Though the A5 tool set does not implement all of the user stories presented here, the implementation of the tools has been strongly influenced by the insights gained in this work.

5.1 Legend

Roles

- P: QualiPSo Trustworthiness Platform
- U: user (e.g., End-user, Developer, Manager)

Tools category and responsibility

- M: measurement (WP5.5.2)
- T: tests (WP5.5.1)
- A: analysis (WP5.5.3)
- A7: provided by the platform

5.2 User Story – General context

5.2.1 Evaluation of the trustworthiness of a product

The general context of the user stories is the evaluation of the trustworthiness of a product by a user. This general story needs support from all types of tools of the A5 toolset (function testing and benchmarking tools; measurement tools; tools supporting analysis, model building and assessment as defined in the QualiPSo project proposal).

A typical scenario could be: a user needs to prepare the decision about using an already identified product in one of his or her projects. To achieve this goal, the following steps could be taken:

- define selection criteria
    - select relevant aspects

creative commons

    - prioritize these aspects according to their importance in the context of the specific project
- collect data matching the selection criteria
- analyze data based on priorities
- prepare presentation
- present results to management

Since the above description is too coarse-grained, user stories have been defined which focus on certain aspects of this context scenario.

5.3 List of user stories

If we talk about projects here, the idea is not to be able to evaluate only OSS projects hosted in the QualiPSo factory, but to have a place to collect information on an external OSS project.

5.3.1 Insert new project

Role: User, *Tools:* M A7

Goal: U inserts a new project in P

- U wants to know something about a product or use a product into a comparison, while the product is not present in P (version must be considered too)
- P asks U for all the information necessary to retrieve and evaluate the new product or the new version (forge, repository, ...)
- P verifies the product references and repositories and collects the data
- P notifies U about the results of the insertion and/or of any errors or problems occurred
- U adds the information and measurements that could not be done in an automated way
- U agrees to publish the product
- P makes the product visible to the community

5.3.2 Get OSS Project list

Role: User, *Tools:* A7

Goal: P shows the list of available projects to U

5.3.3 Start the toolset

Role: User, *Tools:* A7

Goal: U accesses P

- P presents a login dialog that offers an optional user identification with user ID and password (required for any write access), and a user role selection.

creative commons

- U inserts its credentials and role.
- P validates information. All further screen contents will depend on the selected role and can present more or less aggregated data.

5.3.4 Change role after login

Role: User, *Tools:* A7

Goal: U changes his/her role, after accessing P

5.3.5 Get OSS overview information if available

Role: user, *Tools:* A

Goal: U accesses summary information about a product: to have a first overview and an overall impression of the product

- U chooses a product from the list of available projects
- P presents basic information about the product (supplier, latest version analyzed, items analyzed, measurements made, general trustworthiness category, license information, ...) which is similar for all projects
- P presents summary results about the product (trustworthiness level, trustworthiness indicators, ...)
    - If no data is available for the product
        - P shows a trustworthiness evaluation dialog to provide the necessary information for the repository
        - see user story 5.3.10
        - see user story 5.3.11

5.3.6 Update OSS information

Role: End-user, developer, *Tools:* MT

Goal: U updates information about a product

- U chooses a product
- P presents currently available information about the product
- U inserts updated information (e.g. new metric-based data, annotations)
- P validates updated information
- P updates the product

5.3.7 Search for relevant information of a product

Role: User, *Tools:* A A4 A7

Goal: U search, filter, select and analyze specific information about a product

creative commons

5.3.8 Analyze presented information

Role: user, *Tools:* A

Goal: U deeply analyzes the quality related measures and indicators of a product

Goal: U aggregate several detailed information to obtain an abstract view

5.3.9 Define a search and analyze profile

Role: user, *Tools:* A7

Goal: U customizes search and analysis preferences through the definition and storage of a user profile

5.3.10 Create trustworthiness metrics information

Role: User, *Tools:* MTA

Goal: product measures are generated and stored in P, based on the choices of U

- U chooses a product
- P presents the available analyzers to generate metrics. P gives additional guidance on what analyzers should be executed and on how to proceed, depending on user role
- U chooses the analyzers
- P executes the analyzers, retrieve and store the analysis results

5.3.11 Create and execute trustworthiness test-based information

Role: User, developer, *Tools:* T

Goal: U create test suites

- U create a new test suite
- P stores the test suite

5.3.12 Execute trustworthiness test-based information

Role: User, developer, *Tools:* T

Goal: U executes test suites

- U selects product and test suites

creative commons

- P executes the tests (functional, performance, interoperability...) in the selected test suites
- P stores the results (including test coverage data)

Results can be analyzed by means of the user stories: Search for relevant information, Analyze presented information

5.3.13 Create and execute trustworthiness dynamic measures based information

Role: User, *Tools:* T

Goal: U access and analyze dynamic measures information

- U select a product
- P checks if the selected product provides a framework (QualiPSo based or not) to collect dynamic information
- P retrieve and analyzes dynamic information from the project repositories
- P asks U what kind of analysis is requested on the data retrieved
- U choose the analysis from the available ones
- P performs the analysis
- P stores the result

Results can be analyzed by means of the user stories: Search for relevant information, Analyze presented information.

5.3.14 Get quality and trustworthiness trend and history information

Role: Developer, *Tools:* A

Goal: U analyzes metrics and test information over time to see how they contribute to the overall OSS quality and trustworthiness.

See user story: “Get recommendations to improve trustworthiness”.

5.3.15 Get recommendations to improve trustworthiness

Role: Developer, *Tools:* A

Goal: P provides U with a list of recommendations on how to improve the trustworthiness of the product. The list of recommendations can be sorted according to a U decided criterion

5.3.16 Quick compare

Role: User, *Tools:* A

creative commons

Goal: U quickly compares 2 or more products

- U access to P, insert the domain of the product, selects the products to compare
- P gives to U a summary report indicating the relative trustworthiness of the products. The report is customized on U role and on product domain

5.3.17 Deep compare

Role: User, *Tools:* MTA

Goal: U deeply compares 2 or more products

- U access to P, and selects the products
- P shows all the measures that are available on the products
- U selects the measures in which is interested
- P executes all the measures that were selected but not yet executed on the products
- P analyzes the products and give a very detailed report of the analysis of the measures considered

creative commons

6. TESTING TOOLS

6.1 JaBUTi

Version: 1.0.3

Project home: http://ccsl.icmc.usp.br/projects/jabuti

Location: http://ccsl.icmc.usp.br/redmine/projects/jabuti

Provider: University of Sao Paulo

License: LGPL

Integration into Spago4Q:

One extractor with two operations: one for uploading new projects into the service and the other to get the analysis' results.

Known issues: JaBUTi in this current version has scalability problems to collect coverage information of large or long running OSS products.

JaBUTi (Java Bytecode Understanding and Testing) is a structural testing tool that implements intra-method control-flow and data-flow testing criteria for the Java bytecode language [20]. JaBUTi implements four intra-method control-flow-based testing criteria and four intra-method data-flow-based testing criteria. It evaluates the coverage of a given test set against these testing criteria, reporting the coverage obtained with respect to each one. The tool works at bytecode level and no source code is required to compute the testing requirements and the coverage. If the source code is available, the tool is able to map back the results computed from the byte code to its corresponding source code.

JaBUTi Service is a web service built upon the original JaBUTi, offering remote access to the tool's features through standard web services technologies (SOAP and Axis2).

In the current version, the integration with Spago4Q is done by one extractor with two operations, one that uploads new projects into the service (when the project is created in Spago4Q) and one that gets the analysis results back from the service (after the user executed the tests).

Integrating the JaBUTi testing tool developed at USP is an obvious choice since it supports the majority of the testing criteria that are investigated in Task 5.4.2 to be used in an incremental testing strategy under definition.

This choice is supported by additional reasons:

- *Control*: First of all, we can easily have access to the knowledge and the manpower of the developers, since they are researchers in the USP group. Furthermore, there is a guarantee that people are going to work in these tools (the tools will not be discontinued) during the QualiPSo project and beyond.

creative commons

- *Functionalities*: The JaBUTi tool has important functionalities to support the application of both control and data-flow based testing criteria while the majority of OSS Java testing tools (like Cobertura, CodeCover, etc) just include control flow testing. Moreover, with JaBUTi it is possible to evaluate the coverage of different combinations of test cases just by enabling and disabling some of them. Once a testing requirement is identified as infeasible, JaBUTi enables the tester to eliminate it from the coverage computation.

JaBUTi can be operated by its graphical interface or by command line. The graphical interface is interesting when you are learning or teaching testing concepts while the command line interface is more adequate for performing experimental studies when a large amount of data should be collected. For a more extensive list of JaBUTi functionalities, the interested reader can read [20].

When using the web service version of the tool, tests are executed with the command-line, but there's also a GUI (Graphical User Interface) to help the user organize his tests. The GUI is available at: http://ccsl.icmc.usp.br/redmine/projects/jabuti-service-gui. It offers a graphical user interface that communicates with the web service and makes it easier to run the necessary operations.

The acquired experience during the integration of the JaBUTi tools can be used later to create a document that guides the integration process of other testing tools into the QualiPSo project. The process of offering the JaBUTi functionalities as service is described in a separate document [21].[20][22][21]

6.1.1 Installation (Desktop GUI)

To use JaBUti in the most common way (Desktop GUI) the user just has to execute its executable jar package. This file can be downloaded in JaBUTi Project Home. The following third-party-libraries, however, are needed to execute it:

Library	Description	Version	Location
bcel-5.2.jar	Byte Code Engineering Library	5.2	http://jakarta.apache.org/bcel/
capi.jar	Java API for mathematical curves	--	http://sourceforge.net/projects/curves/
mucode.jar	Java API for mobile agent systems	--	http://sourceforge.net/projects/mucode
junit-4.4.jar	Framework for writing and running automated tests	<= 4.4	http://www.junit.org/

creative commons

All these libraries have to be added in the classpath when JaBUti is invoked. The figure below shows how to call JaBUTi correctly in the command line.

```
auri@AURIMRV ~/example
$ java -cp ".;..\Tools\jabuti;\
>..\Tools\jabuti\lib\bcel-5.2.jar;\
>..\Tools\jabuti\lib\capi.jar;\
>..\Tools\jabuti\lib\mucode.jar;\
>..\Tools\jabuti\lib\junit-4.4.jar" gui.JabutiGUI
```

Figure 26.1: Example JaBUTi call

Beyond these libraries, the Graphviz (http://www.graphviz.org/) application has to be installed in the host which JaBUti is running. This third-party-application is responsible for graphs visualization.

6.1.2 Installation (Service)

Installation prerequisites:

- Spago4Q 2.1 platform
- MySQL 5.x Database
- Apache Tomcat 6

Database Setup:

1. Issue the following commands in MySQL:
    - `Create database jabutiservice;`
    - `Grant all on jabutiservice.* to 'jabutiservice'@'localhost' identified by 'jabutiservice';`
    - `Flush privileges;`
2. Run the database script jabutiservice.sql (distributed in the package):
    - `mysql -u superuser -p -D jabutiservice < jabutiservice.sql`

Deploying the Web Service:

Deploy jabuti-service.war file into an Apache Tomcat instance. This is normally done by putting the WAR package inside the "webapps" folder and starting Tomcat (if it is not already running).

Configuring the Web Service:

1. Open the file [tomcat_root]/webapps/JabutiWS/WEB-INF/classes /jabuti.properties.
2. Change the properties as follows:

creative commons

JABUTI_PERSISTENCE_HOME:point it to a folder where project's files can be stored. You need write permission on this folder.

- db.*:if you followed this documentation you probably do not need to change database properties. However, if you have a different setup, you can change it here (e.g. different database url port).

Starting the Web Service:

Restart Tomcat after changing the jabuti.properties file. If everything was correctly configured, the following information will be shown in Tomcat's log (usually catalina.out file):

- Database connection: OK
- Persistence directory: OK
- Tomcat project directory: OK

6.1.3 Tool Usage (Desktop)

JaBUTi is especially useful to collect metrics to assess quality factors such as Functionality. This particular quality factor aims to evaluate the degree to which an OSS product covers functional requirements. The question associated with this quality factor is: what is the percentage of the functional requirements that are covered or satisfied by the tests? The metric utilized is absolute and aims to measure the correctness level of the OSS product.

As a structural testing tool, JaBUTi is able to provide information on how thorough are the tests applied during the development of the OSS product. For example, one can evaluate the trustworthiness of a product by assessing the coverage of the statements executed by the test suite available for the OSS. Statement coverage below 50% indicates that many of the functional requirements have not even been assessed once by the test suite. On the other hand, coverage above 70% indicates that the major part of the code has been exercised. If all the tests are successfully executed, and the coverage is adequate, one can have an evaluation of the trustworthiness of the OSS product according to the Functionality quality factor.

This feature of JaBUTi is exemplified in the figures below in which is presented the coverage obtained by testing a Java application that implements a vending machine. They describe the coverage according to the all-node-ei criterion (equivalent to statement coverage) for the Dispenser class. Figure 3 informs about the total coverage of Dispenser class for the evaluated test case. The coverage according to each method is presented in Figure 4. More details on how to operate JaBUTi, can be found in the JaBUTi user guide (see Project home).

creative commons

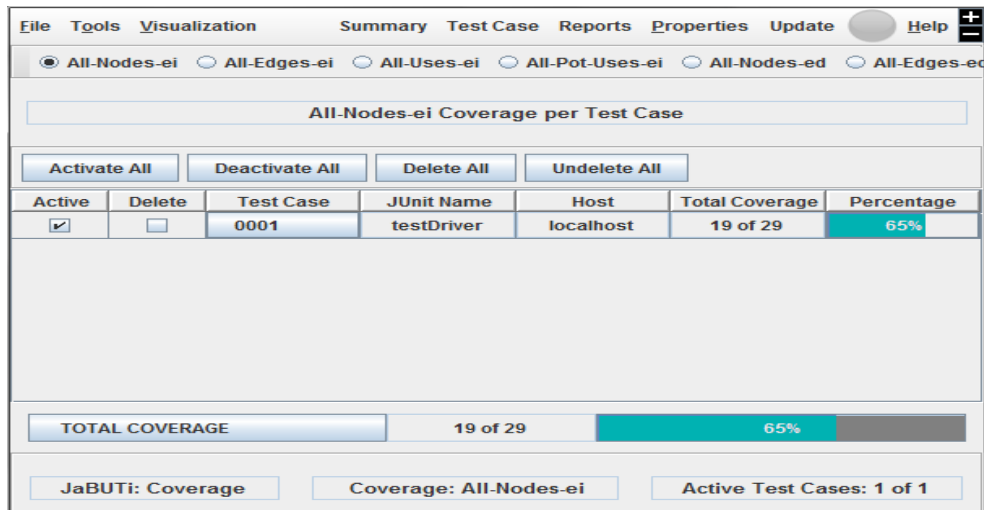


Figure 3: Report by test case

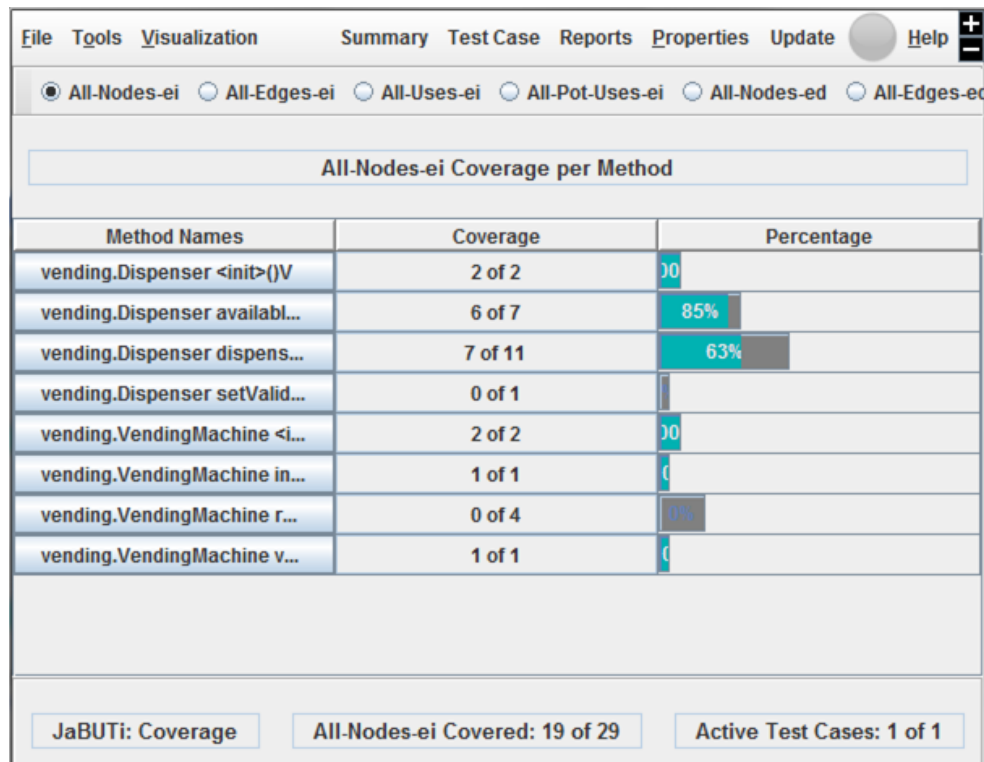


Figure 4: Summary by Method

JaBUTi can create (among others) the following metric values:

- All-Nodes-ei
- All-Nodes-ed
- All-Edges-ei
- All-Edges-ed
- All-Uses-ei

creative commons

- All-Uses-ed
- All-Pot-Uses-ei
- All-Pot-Uses-ed

A more comprehensive overview on the information that can be extracted from JaBUTi can be found in Appendix C.

6.1.4 Tool Usage (Service GUI)

A Graphical User Interface was developed to interact with JaBUTi Service at the client side. It's used to execute a project, generate results and send them back to JaBUTi Web Service. The following steps explain how to use it after projects were created by using Spago4Q.

1. Download jabuti-service-gui-1.0.zip in the following URL: http://ccsl.icmc.usp.br/redmine/projects/jabuti-service-gui/files
2. Unpack it; the following files are obtained:
    - jabuti-gui_lib (libs needed for execution);
    - scripts (scripts used by jabuti);
    - jabuti-gui.jar (the application);
    - jabuti-gui.properties (configuration file).
3. Open the 'conf/jabuti-gui.properties' file and change the properties:
    - END_POINT: WSDL URL of JaBUTi Service;
    - JABUTI_PERSISTENCE_HOME: a local directory (of user's choice) to locally save the projects tested.
4. Start the application (e.g. java -jar jabuti-service-gui-1.0.jar).
5. Select the tab Project List and click the Update button.
6. In the generated list, select the project which you want to test (the one Spago4Q created). The Project ID field should be auto-filled.
7. Select the tab Configuration and fill the following information, then click Create/Update:
    - Directory with original classes (where are the classes to test);
    - Directory of test script (where is the test script, e.g., build.xml);
    - Directory of test output (where the outputs will be saved);
    - Project ID in Spago4Q (filled automatically);
    - ORIG_JAR (Jar with the classes to test);
    - INSTRUM_JAR (Jar to be created with the classes instrumented).

creative commons

8. Move on to the tab **Execution** and press **Execute** button. JaBUTi CLI will be executed for this project and the execution results will be output in the text area.
9. Finally, go to the **Results** tab and the Spago4Q XML file will be available for the project. The results now have to be sent to JaBUTi Service; to do this, press the **Send** button (make sure the results are ok).

6.1.5 Integration with Spago4Q

Code coverage information obtained from JaBUTi Service are integrated with Spago4Q by using the extractor provided with two operations: Upload and Analysis. How configure the extractors is described in detail in Appendix B.

6.2 JUnit Statistics Tool

Version: JUnit 3.8.1

Project home: http://www.junit.org/

Location: Work Folder/A5/Tools/JUnit-Statistics-Tool

License: Common Public License v1.0 (JUnit)

GPL (web-service and extractor)

Integration into Spago4Q:

The integration into Spago4Q is done, as a proof of concept, with the help of a Spago4Q extractor which requests the JUnit statistics tool web-service (WS).

Known issues: If the project to analyze with the JUnit statistics tool use ant as build tool, the build.xml file must be configured to respect these 2 points :

- have a target "test";
- generate the JUnit report in the folder "target/surefire-reports."

JUnit is a unit testing library for JAVA created by Kent Beck and Erich Gamma. JUnit is largely used in the IT industry for unit testing. Thus, a tool that provides some basic statistics on the unit test cases of a project is needed.

For this purpose we embedded JUnit library together with Maven, Ant and Subversion libraries into a web-service to provide statistics about the number and the successful rate of the unit test cases of a project.

The integration with Spago4Q has been partially carried out as a proof of concept.

6.2.1 Available metrics

The JUnit statistics tool through the use of its web service can retrieve the following metrics :

- The number of test cases;

creative commons

- The successful test case rate.

6.2.2 Installation

Installation prerequisites

The environment requirements to run the JUnit Statistics Tool web-service and extractor are :

- JDK above 1.6;
- Tomcat above 6.0;
- Spago4Q 2.3;
- Maven above 2.2.1;
- Ant 1.7.1;The system variables M2_HOME and ANT_HOME must be set on Windows systems.

Software components

- The web-service :
  - JUnitStatisticsService.war : the web-service to deploy in the Tomcat server.
- The Spago4Q's extractor and its configuration:
  - extractor-junit-1.0.jar: the extractor
  - junit_extractor.zip : an archive containing the Spago4Q configuration metadata of the extractor.

All the software components are available on the Qualipso portal in the following folder “Work Folder/A5/Tools/JUnit-Statistics-Tool/JUnit-Statistics-Tool-Package.tar.bz2”.

Installation of the web-service

To test deeper the web-service you can use a web-service testing tool such as SoapUI. The parameters you will have to give are :

- svnUrl : the svn url where the source code of the project to assess is hosted (mandatory);
- userName : the username that allows to connect to the SVN repository where the project to assess is hosted (optional);
- password : the password that allows to connect to the SVN repository where the project to assess is hosted (optional);
- revision : the svn revision number of the project to checkout. Let it empty if you want to checkout the HEAD revision (optional).
- buildType : the build automated tool used adopted by the project (a string “ant” or “maven” is mandatory).

creative commons

Note: Please be aware that, in case you choose "ant" as build type, the build.xml file must respect 2 things :

1. *Have a target "test";*
2. *Have been configured to generate the JUnit test report in the folder target/surefire-reports.*

Installation of the extractor

To "install" and set up the extractor with Spago4Q, you need to copy the extractor jar file into a dedicated folder in Spago4Q and then load the extractor configuration by using the Spago4Q administration panel. The different steps are :

1. Shutdown the Spago4Q's Tomcat server ([Tomcat directory]/bin/shutdown.sh for a standalone Linux installation of Tomcat);
2. Copy the extractor-junit-1.0.jar in [Tomcat directory]/webapps/SpagoBI/WEB-INF/lib;
3. Start your Tomcat server ([Tomcat directory]/bin/startup.sh);
4. Log into the Spago4Q administration panel;
5. Go to Tools>Import/Export and import the junit_extractor.zip archive;
6. In the operation parameters of the extractor configuration, we defined the the web-service endpoint. You can adapt the value of this parameter to your needs.
7. In the "Interface Type Detail" page, you should create the fact table by clicking on the appropriate button. The fact table will store the values of the metrics of the project you will assess.
8. Finally, you can run the extraction in the "Extraction Process Detail "page. If the the extraction have been run successfully, normally a new record will be inserted into FT_JUNIT table.

creative commons

7. CODE ANALYSIS TOOL

Quality of code is indeed very difficult to be measured; the main difficulties that need to be taken in account, and that are hard, if not outright impossible, to solve, are the following ones:

- *Subjectivity*. Source code quality involves subjectivity, depending on the experience and programming style of the developers and the team.
- *Development environment*. Development tools often influence how portions of code will be written.
- *Application domain*. Application domains often have very different and peculiar requirements and corresponding solutions. The solutions for specific application domains often also define particular coding rules which might be considered inappropriate for other domains.
- *Language*. Source code quality heavily depends on the programming language. Every programming language has its standard coding styles, and these reflects language capabilities and varies a lot from language to language.
- *Development process*. Development processes influence the way a software system is coded; the way the software is coded can be very different, from a cowboy-hack-and-code-style to a very formal and defined style, from a long project waterfall style (a sequential development during even more than 1 year) to an iterative and incremental style with very short (1-2 weeks) iteration cycles, from a localized team in the same room, to a distributed virtual team scattered all around the world. All these different development processes features introduce differences on the coding style preferred and used.
- *Quality processes*. Quality processes may impose some strict rules on source code format and coding style.

Despite the difficulties, the quality of source code is considered an important factor to take in account when evaluating a software tool. This is confirmed by the research on the perceived trustworthiness of OSS software system, carried out in QualiPSo WP 5.1.

In previous versions of the A5 tool set, FindBugs and SISSy were used too. They have now been excluded because the measures they provide are now also provided by the other tools. Fossology was replaced by OSLC for the reasons explained in Section 7.4.

creative commons

7.1 Macxim

Version:	2.0
Project home:	http://qualipso.dscpi.uninsubria.it/macxim
Provider:	University Of Insubria (INS)
License:	LGPL

Integration into Spago4Q:

The integration of Macxim with Spago4Q platform is implemented through the custom Spago4Q extractor which interoperates with Macxim Web Service.

Known issues: None

Macxim is a static code analysis tool for measuring Java source code.

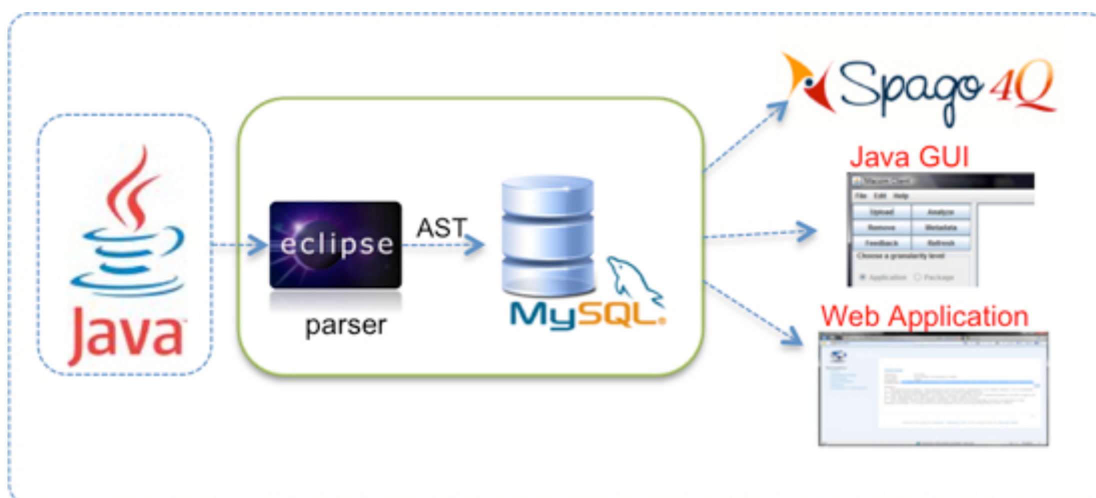
Macxim stores a simplified Abstract Syntax Tree (AST) representation of the source code in a relational database (MySQL).

Several metrics can be extracted simply via SQL queries.

The main advantage of this approach, when compared to similar approaches, consists in the separation of two tasks that are usually found glued together:

- Source code analysis;
- Computation of software measures.

This can be quite an advantage, since while the analysis of the source code is more or less the same in many different tools, the software measures that are calculated can widely vary. The Macxim approach enables the user to add new software measures easily, reusing a full fledged analysis engine.



Macxim have the capability of integrating other measurement tools.

creative commons

In the final release we integrated PMD and Checkstyle.

7.1.1 External Tools integrated into Macxim

PMD

PMD scans Java source code and looks for potential problems. The source code problems that PMD is able to detect are of the following types:

- Possible bugs. Example: empty try/catch/finally/switch statements.
- Dead code. Examples: unused local variables, unused parameters, unused private methods
- Suboptimal code. Example: wasteful String concatenation usage (instead of StringBuffer/StringBuilder).
- Overcomplicated expressions. Examples: unnecessary if statements, for loops that could be while loops.
- Duplicate code. Example: copied/pasted code means copied/pasted bugs

PMD is an automated code review software tool for the Java language. Besides the usual code style and code review oriented features, it also presents some peculiar features, that are usually not found in static analyzers.

One of the most interesting features is its capability to detect cut and paste portions of source code. It can be used against the source code coming from two or more software systems, to detect plagiarism; or it can be used against the source code coming from a single software system, to detect cut-and-pasted portions of code that could lead to maintenance and quality problems.

Some of the shortcuts attributed to the subjectivity and variability of what is considered good quality source code can be solved in PMD since the checked rules can be highly configured and customized.

CHECKSTYLE

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task.

Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration file supporting the Sun Code Conventions is supplied. As well, other sample configuration files are supplied for other well known conventions.

Checkstyle can check many aspects of the source code. It provides checks that find class design problems, duplicate code, or bug patterns like double checked locking.

Checkstyle is written in Java. Since it is developed outside of QualiPSo, it is used “as is.”

Checkstyle has been chosen because it is a widely used automated code review software tool for the Java language. Automated code review software tools check source code for compliance with a predefined set of rules or best practices.

creative commons

Some of the shortcuts attributed to the subjectivity and variability of what is considered good quality source code can be solved with Checkstyle since the checked rules can be highly configured and customized; every project can check its source code with its own defined coding style rules.

7.1.2 The available metrics

Macxim can create (among others) the following metric values:

Code Size	
ELOC (Effective Lines Of Code)	ELOC per Class
ELOC per Interface	Number of Packages
Number of Classes	Number Of Classes Out Of Packages
Number of Abstract Classes	Number of Interfaces
Number of Methods	Number of Public Methods
Number of Private Methods	Number of Protected Methods
Number of Methods per Class	Number of Methods per Interface
Number of Parameters per Method	Number of Attributes per Class
Number of Public Attributes per Class	
Code Documentation	
Comment LOC (Lines Of Code)	Inline Comment LOC
Comment LOC per Class	Inline Comment LOC per Class
Comment LOC per Interface	Inline Comment LOC per Interface
ToDo Comments	Missing Javadoc Comments
Code Quality	
Cyclomatic Complexity (Mc Cabe Index)	CBO (Coupling Between Objects)
DIT (Depth of Inheritance Tree)	LCOM (Lack Of Cohesion Of Methods)
NOC (Number Of Children)	RFC (Response For A Class)
Code Modularity	
Number of Classes With Defined Methods	Number of Classes With Defined Attributes
Number of Not Implemented Interfaces	Number Implemented Interfaces per Class
ECA Rules (Elementary Code Assessment)	
Lines Longer Than 80 Characters	Avoid Catching Throwable
Avoid Nested Blocks	Constructor Calls Overridable Method
Modifier Order	Class Naming Conventions
Upper Ell	Empty Catch Block

creative commons

Final Parameters	Excessive Class Lenght
Parameter Assignment	Excessive Method Lenght
Avoid Star Import	For Loops Must Use Braces
Illegal Import	If Else Statements Must Use Braces
Redundant Import	If Statements Must Use Braces
Unused Imports	Missing Break In Switch Statements
File Too Long	Override Both Equals And Hashcode
Method Too Long	Unused Private Field
Too Many Parameters	Unused Private Method
Double Checked Locking	Switch Statements Should Have Default
Equals Hash Code	Use Equals To Compare Strings
Illegal Instantiation	While Loops Must Use Braces
Inner Assignment	Assignment To Non Final Static
Missing Switch Default	Immutable Field
Fall Through	Final Field Could Be Static
String Literal Equality	Unused Formal Parameter
Redundant Throws	Singular Field
Illegal Catch	Confusing Ternary
Empty Block	Use Collection Is Empty
Final Class	Empty If Statements
Interface Is Type	Unnecessary Local Before Return
HideUtility Class Constructor	Position Literals First In Comparisons
Visibility Modifier	Simplify Boolean Expressions
Hidden Field	Simple Date Format Needs Locale
Parameter Should Be Final	Too Few Branches For A Switch Statement

Table 1: Macxim metrics

For each project is possible to view the results at three different granularity levels:

- Application;
- Package;
- Class (specifying a container package).

The results of the source code analysis and the metrics calculated from Macxim can be viewed through:

- Macxim Web Application

creative commons

- Macxim Personal Client
- Spago4Q platform

7.1.3 Installation

Macxim Server Installation

In order to install Macxim Server, you should:

1. Extract macxim2.0.zip
2. Copy macxim.war to a directory within your Tomcat server's webapps root ($TOMCAT/webapps/)
3. Run the installation script: point your browser to the base URL of your tomcat installation ($TOMCAT/macxim – e.g. http://localhost:8080/macxim)
4. The installation wizard will guide you to set up the database and provide basic settings. Follow the wizard to finalize the installation
5. Check your system prerequisites, as shown in the setup wizard

6. Than click "next" button to go to the next step, and provide all the database information. Remember to check the box "Create Database" if you want to create a new one, otherwise an existing one will be used

creative commons

7. Provide Macxim FileSystem Options and choose your Administrator Account credentials

creative commons

Macxim FileSystem options

To set up your Macxim database, enter the following information.

Remove Sourcese: *

YES, REMOVE SOURCES

Macxim download source code from svn repositories and store sources in TomcatWebapps/Macxim/WEB-INF/project_chechout.
Do you want to remove downloaded source files after the analysis?

Administrator account

The administrator account has complete access to the site; it will automatically be granted all permissions and can perform any administrative activity. This will be the only account that can perform certain activities, so keep its credentials safe.

Username: *

Spaces are allowed; punctuation is not allowed except for periods, hyphens, and underscores.

Password: *

Confirm password: *

Spago4Q connector options

To set up your Macxim database, enter the following information.

Spago4Q endpoint: *

Spago4Q QualiPSo project Manager endpoint

Save and continue

8. If your installation is successfully completed you have to receive confirmation as shown in the following image

MacXim

Navigation
- Home
- Analyzed Projects
- Login

Overview

Macxim has been configured properly

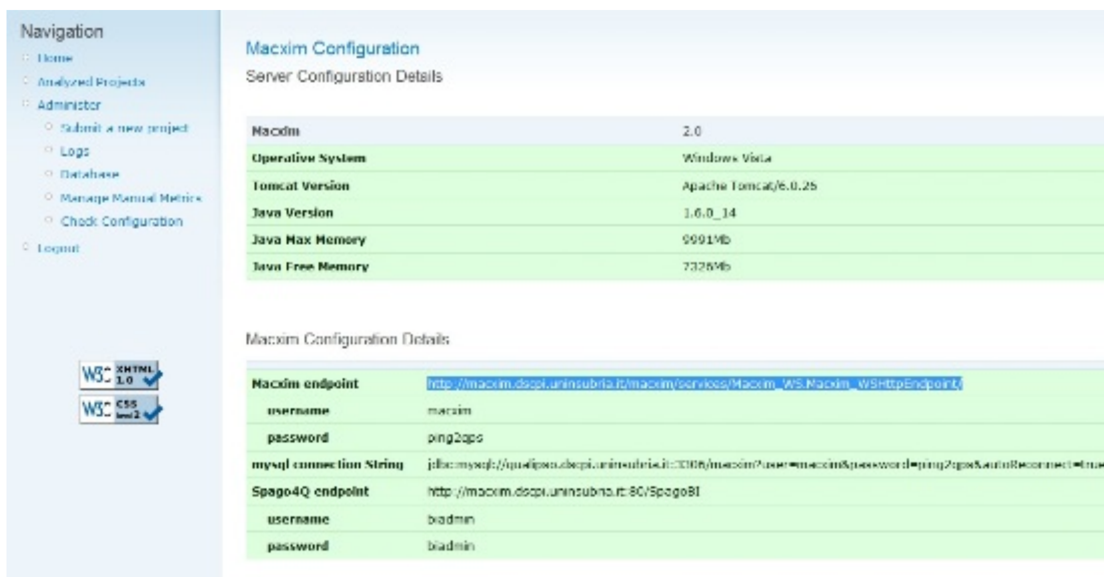
Version: 2.0 RC
Provider: University of Insubria (INS)
License: LGPL
Endpoint:
http://macxim.dscpi.uninsubria.it/macxim/services/Macxim_WS.Macxim_WSHttpEndpoint/

Macxim Personal Client Installation

creative commons

In order to install Macxim Personal Client, you should:

1. Extract MacximGUI.zip
2. Look for your web service endpoint:
    a. Point your browser to the URL of your Macxim installation ($TOMCAT/macxim - e.g. http://localhost:8080/macxim)
    b. Perform Login (on the Navigation Menu) to Macxim Web Application (Username and Password are specified during the installation of Macxim Server)
    c. Go to "Check Configuration" page (on the Navigation Menu)
    d. Copy the endpoint address



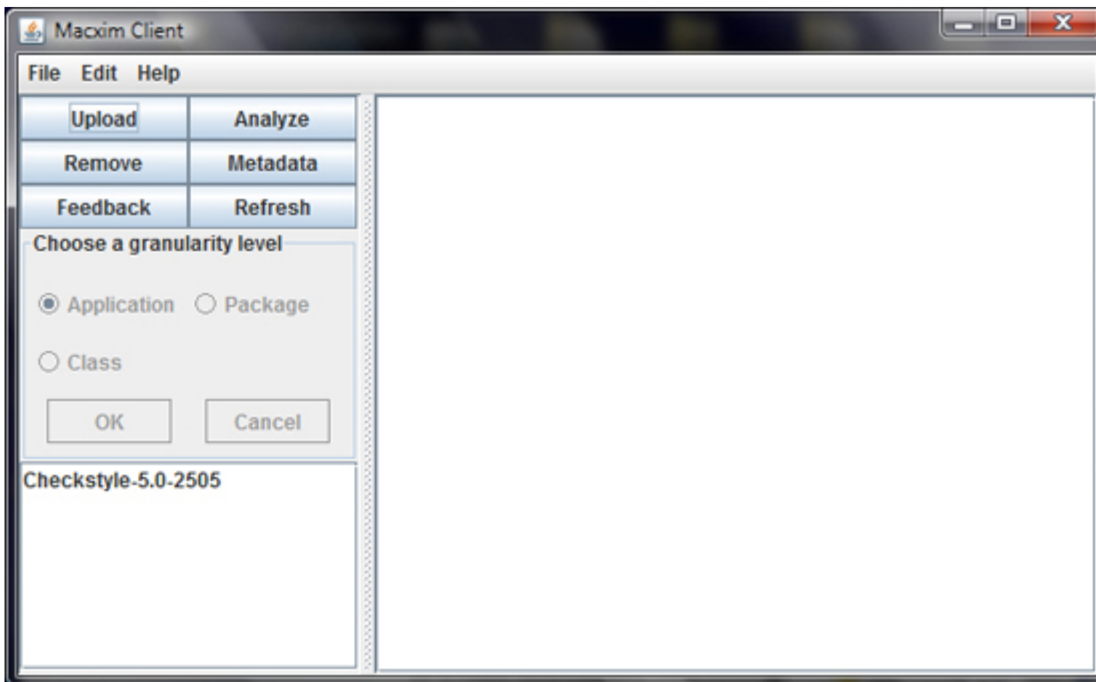
3. Edit your configuration file MacximGUI/conf/macxim.ini:
    a. Edit your web service endpoint
    b. Edit your database connection

```
…

[mySqlApi]

macximdb_jdbcURIString =
jdbc:mysql://localhost?user=myUser&password=MyPassword&autoRe
connect=true

[webService]

endpointAddress =
http://localhost:8080/MacximWS/services/Macxim_WS.Macxim_WSHt
tpEndpoint/

…
```

4. Test your Macxim installation:
    a. Run java –jar MacximClientSwing-1.0-SNAPSHOT

creative commons

b. If macxim has been configured properly, you will see MacximGUI with the example project (Checkstyle-5.0-2505) as shown in the next Figure



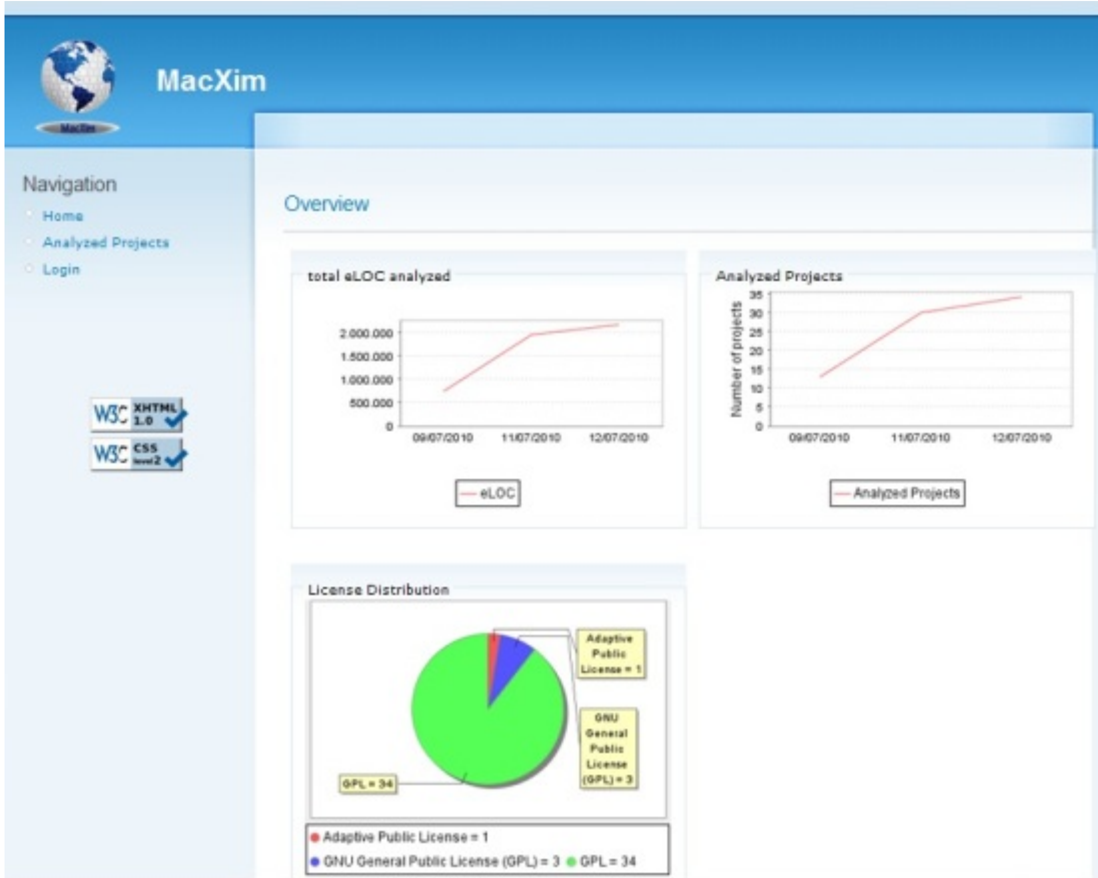
7.1.4 Tool Usage

MacXim Web Application Usage

In order to use MacXim Web Application you have to install MacXim Server on your local machine as specified in Section 7.1.3.

When you point your browser to the base URL of your tomcat installation ($TOMCAT/macxim – e.g. http://localhost:8080/macxim), you are on the Web Application's "**Home**", where are shown some statistics about your MacXim installation's activity.

creative commons



You are able to see the results of the existing projects in your MacXim installation by the link "**Analyzed Projects**" on the left menu.

MacXim

Navigation

- Home
- Analyzed Projects
- Login

Analyzed Projects

Analyze Selected Projects

Select	Project	Version	Revision	Actions
	Ant			
		1.8.1		
☐			961732 (08/07/2010)	
	Checkstyle			
		5.0		
☐			2505 (17/06/2009)	
	eclipse			
		3.5		
☐			21051 (05/07/2009)	
☐			21062 (13/08/2009)	

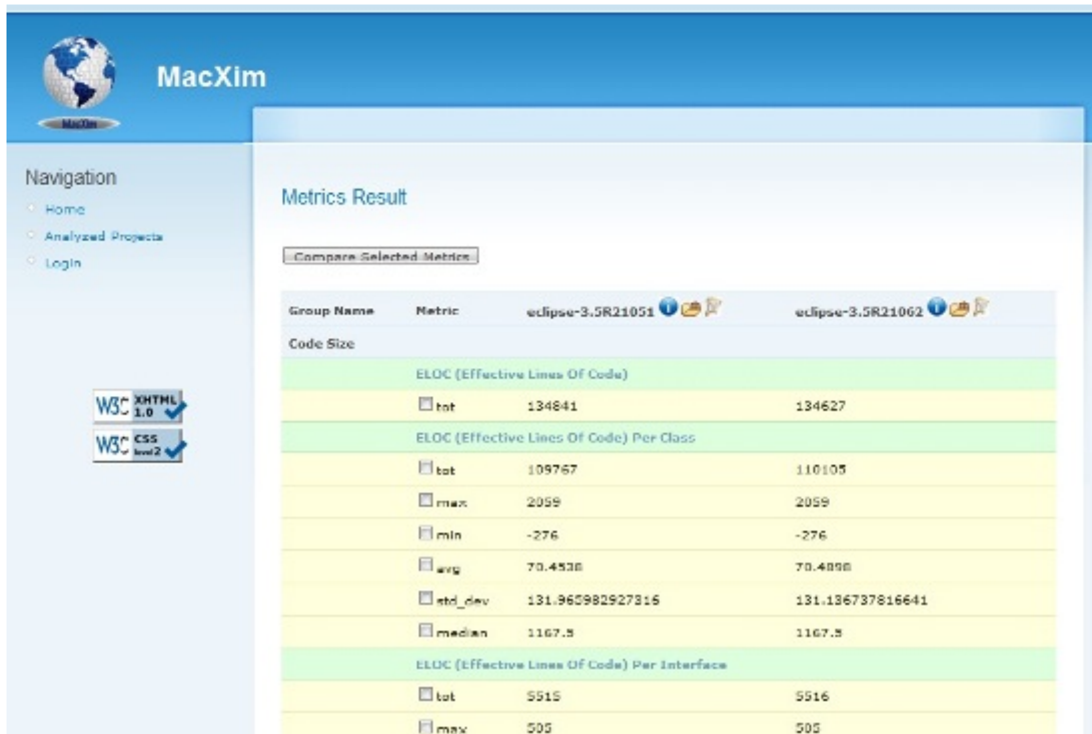
creative commons

The projects are presented in a table showing the different version and revision of each project. For each revision is possible to **see results** in three different ways:

- Select the checkbox on the left column ("Select") of the revision you are interested in, and then click the button "Analyze Selected Projects" placed both on the top and bottom of the page, in order to see "Metrics Results" page
- Click on the "Macxim Report" button (if available) on the right column ("Actions") of the revision you are interested in, in order to see Spago4Q report
- Click on the "CSV" button on the top of the page, in order to export all MacXim analysis results in a CSV file

When "Metrics Results" page is shown (so you have chosen the first way to view the results) MacXim provide more functionalities like:

- Metrics graphical comparison
- Metadata informations for each release selected
- Detailed description of each metric
- Metrics results at different granularity level (starting from application level you can see results for each package, and from a package you can see results for each class)
- Manual metrics results for each release selected



creative commons

To **compare selected metrics** you have to select the checkbox on the left column ("Select") of the metrics you are interested in, and then click the button "Compare Selected Metrics" placed both on the top and bottom of the page, in order to see a graphical comparison.

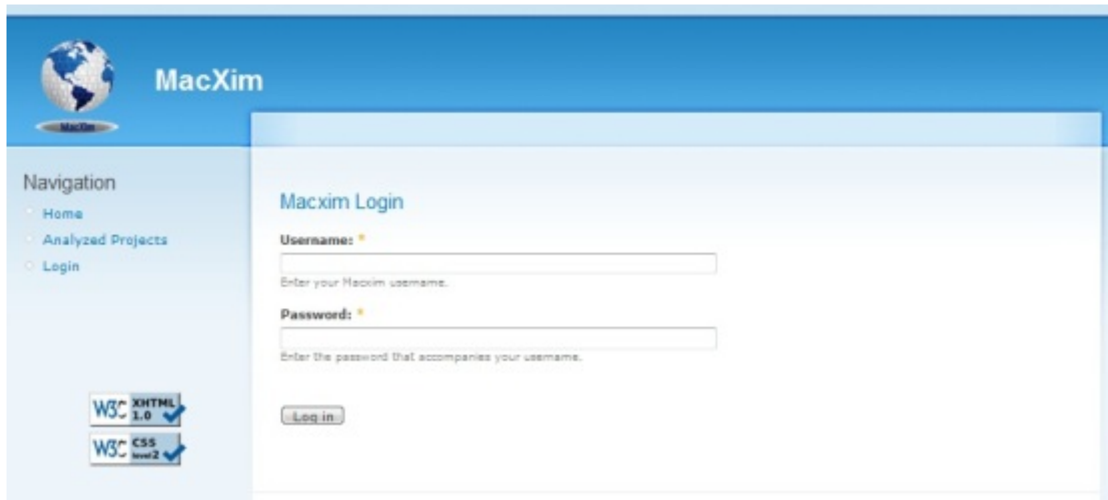
To see **metadata informations** you have to press "Metadata Informations" button near the name of the release you are interested in, in order to see a popup containing these informations.

To see a **detailed metric description** of each metric you have to click the name of the metric you are interested in, in order to see a popup containing these informations.

To see **metrics results at different granularity level** you have to press "Package" or "Class" button near the release or the package you are interested in.

To see **manual metrics results** you have to press "Manual Metrics" button near the name of the release you are interested in, in order to see the page containing these results.

To access other functionalities provided by Macxim Web Application you have to perform the "**Login**" trough the link on the left menu.



The requested username and password are the same you have specified during the installation process. If your login is successful, you can see the message "Welcome To Macxim".

creative commons

Now in the "Analyzed Projects" page you can perform other actions like:

- Delete an existing project
- Edit a project metadata
- Insert manual metrics

To **delete an existing project** you have to press the delete button on the right column ("Actions") of the revision you want to remove from your Macxim installation, and see a message that display the success or insuccess of your request.

When you want to **edit a project metadata** you have to press the edit button on the right column ("Actions") of the revision you are interested in, and you have to insert the new data into the form that contains old values and then press the save button at the bottom of the page.

When you want to **insert manual metrics** you have to press the button on the right column ("Actions") of the revision you want to modify, and you have to fill the form with the values for the metrics you want to add to the current project and than press the insert button at the bottom of the page.

creative commons

In the "Administer" section of the left menu, through dedicated links, you are able to:

- Submit a new Project
- See Logs informations
- Clear cached data in Database SQL tables
- Manage Metrics
- Check your MacXim Configuration

To **Submit a new project** use the link on the left menu and provide all the requested information.

creative commons

To successfully upload a new project you must provide all the metadata that uniquely identify your project:

- Project Name: the name of your project
- Version: the version of the project that you want to analyze
- Release: the release version of the project
- License: the license associated to this project
- Model: the model associated to this analysis
- Prog. Language: the programming language used in the project
- Type of Repository: specify where is located the source code of the project (SVN repository or ZIP file)

and informations about the repository where source code is located:

- Project url: the URL of the SVN repository, or the URL of the ZIP file
- Username: the username to login to the svn server (if required)
- Password: the password to login to the svn server (if required)
- Revision n: to specify a specific SVN revision to be retrieved (HEAD is the default value)
- Commit Date: to specify, by date, a specific SVN revision to be retrieved (if required)

creative commons

When the form is filled and when you are ready to upload the project, click on the "Insert" button at the bottom of the page. If upload request has been correctly sent, you will see a confirm message.

To see the **Logs** informations use the link on the left menu.

You can choose from different Log Types:

- Webapp
- System
- Upload

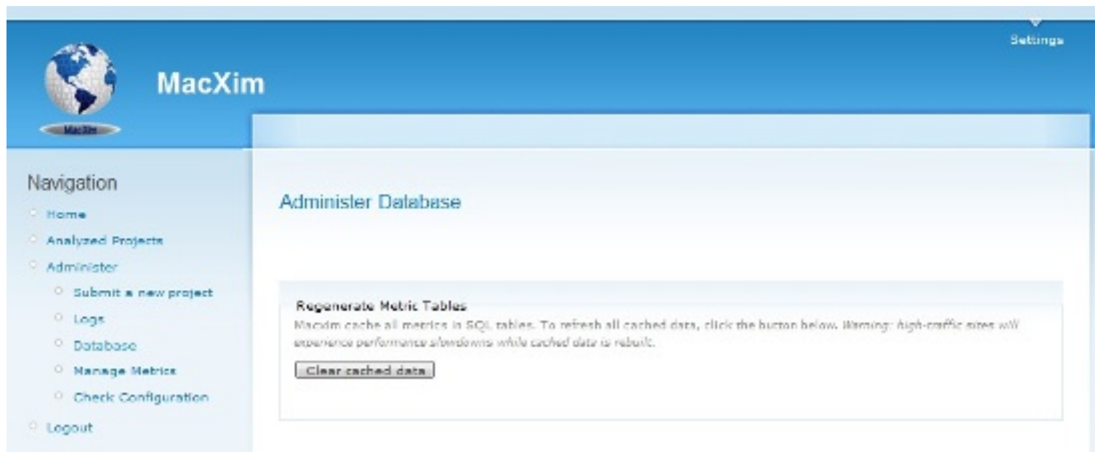
or different Log Levels:

- Trace
- Debug
- Info
- Error
- Fatal

and than click on the "Change" button to refresh the page.

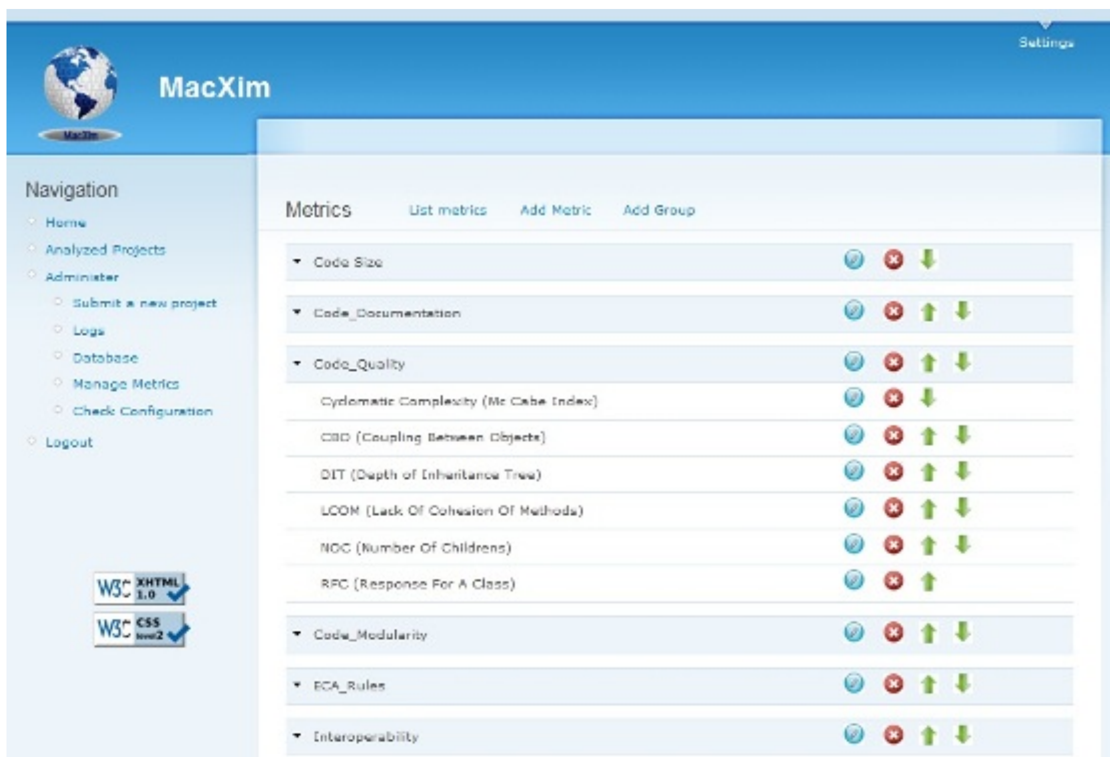
To regenerate metric tables, use the **Database** link on the left menu.

creative commons



MacXim caches all metrics in SQL tables. To refresh all cached data, click on the "Clear cached data" button. Warning: high-traffic sites will experience performance slowdowns while cached data is rebuilt.

To **Manage Metrics**, use the link on the left menu.



At the top of the page there are three buttons to:

- see a list of available metrics
- add a new metric
- add a new metric group

creative commons

When the **List of available Metrics** is shown, you can edit, delete or sort each metric or metric group, using the buttons on the right column, near the metric you are interested in.

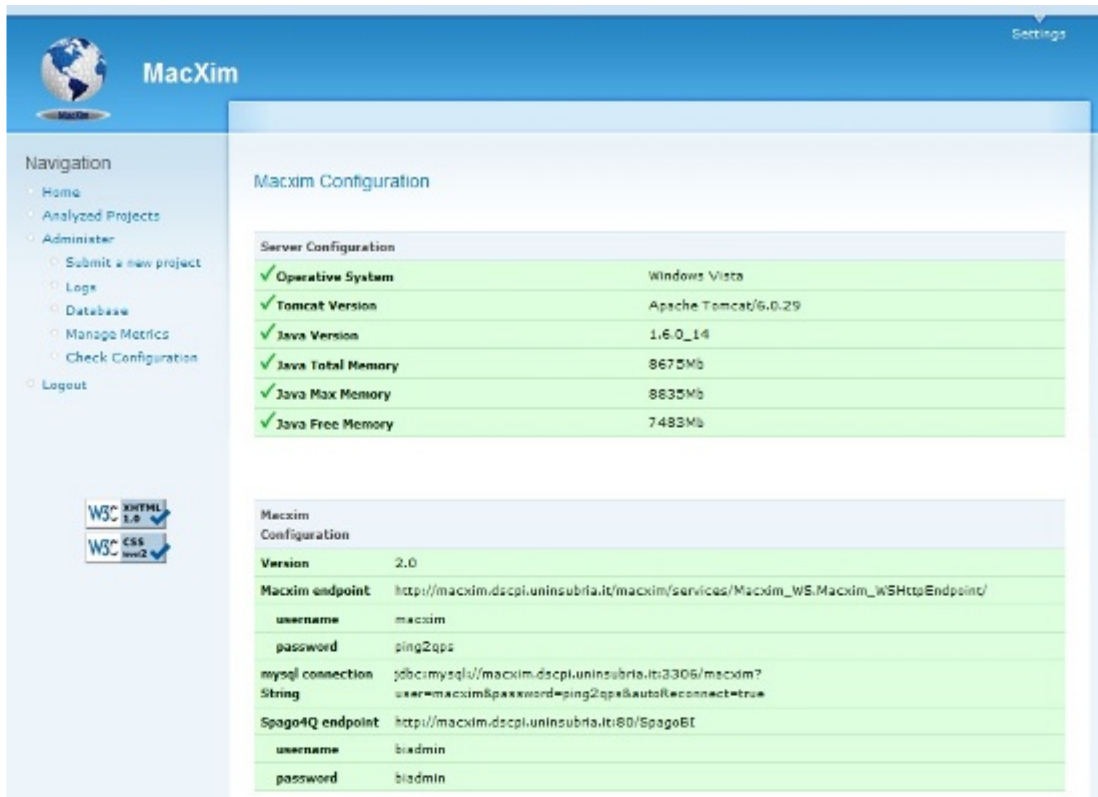
When you **Add a New Metric** you have to specify in a dedicated page the requested informations, like:

- Metric Type (automatic or manual: automatic metrics are calculated by macxim, while manual metrics must be collected by hand)
- Name (metric code saved in the database)
- Title (name shown in MacXim Web Application)
- Group (one of the metric groups saved in MacXim Web Application)
- Data Type
- Range (Range of values used by the metric)
- Description (detailed description of the metric),

and press the "Save" button.

When you **Add a New Metric Group** you have to specify in a dedicated page the name of the group you want to create, and press the "Save" button.

To **Check configuration** of your MacXim installation, use the link on the left menu.



Here you can see all the actual values for your Server Configuration and MacXim Configuration.

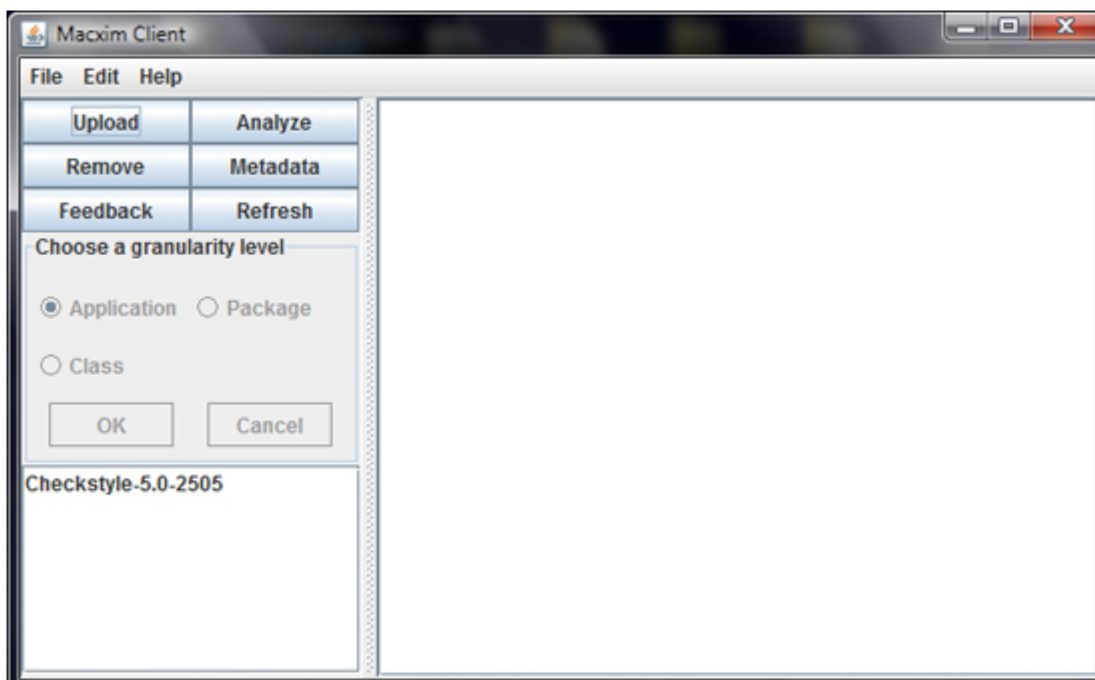
creative commons

MacXim Personal Client Usage

Macxim is released with a GUI client that provides three main functionalities:

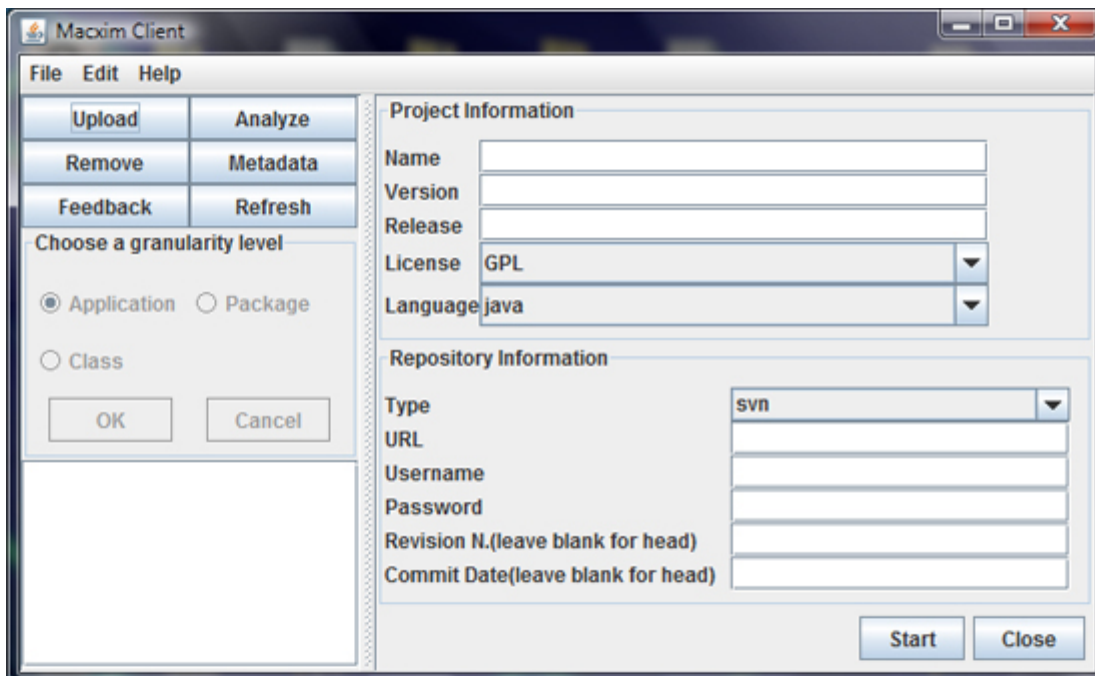
1. Analyze project: to obtain all the application level metrics that refer to the target project
2. Remove project: to remove a project (its abstract syntax tree and its computed metrics) from the MacXim database.
3. Upload project: you can define and customize all the metadata (i.e., project name, svn url, version number) useful to download and analyze the source code of the target project;

The GUI looks as shown in the following figure:



This main window provides an easy access to all the MacXim functionalities.To retrieve the results of a set of metrics that refer to an existing project (e.g., the project has been uploaded into the MacXim database), you can scroll the list of available projects from the scroll menu, then select the target project, and finally click on the “**Analyze project**” button. The client starts a connection with the MacXim Server to retrieve the desired set of metrics and in a few seconds, the values of the metrics are shown into the text box.
To remove an existing project from the MacXim database, you can select a project from the project list and then click on the “**Remove project**” button. When you do this operation ALL the metrics stored in the MacXim database that refers to this project will be deleted.
To upload a new project, you can click on the “**Upload project**” button. A new window appears, as shown in the following figure:

creative commons



To successfully upload a new project you must provide all the metadata that uniquely identify your project. Four types of metadata must be provided:

- Project Name: the name of your project
- Version: the version of the project that you want to analyze
- Release: the release version of the project
- Project url: the svn url of the project.

If needed, you can specify additional metadata such as:

- Username: the username to login to the svn server
- Password: the password to login to the svn server
- Revision: to specify a specific SVN revision to be retrieved
- Commit: to specify, by date, a specific SVN revision to be retrieved

When the form is filled and when you are ready to upload the project, click on the “Upload” project.

7.1.5 MacXim Spago4Q Integration.

MacXim is integrated with Spago4Q by means of macxim-spago4q-extractor. The extractor provides two operations: Upload and Analysis. How configure the extractors is described in detail in Appendix B .

creative commons

7.2 Kalibro Metrics

Version:	0.2
Project home:	ccsl.ime.usp.br/kalibro
Sources/Binaries:	qualipso.org/kalibro-tool
Provider:	University of São Paulo (USP)
License:	LGPL

Integration into Spago4Q:

The integration of Kalibro with Spago4Q platform is implemented through the custom Spago4Q extractor which interoperates with Kalibro Web Service.

Known issues: It is needed to install Analizo, Doxyparse and theirs dependencies (in particular, perl modules). Kalibro, Analizo and Doxyparse was tested on GNU-Linux (Debian, Ubuntu, Mandriva and Geento) and Windows by Cygwin.

Kalibro Metrics aims to improve the use of source code metrics. It is designed for easy integration with a metric calculator tool and show the results in a friendly way. For that, it allows a metric specialist to create a configuration of thresholds associated with qualitative evaluation, including comments and recommendations. These configurations may, then, be used to enhance metric results interpretation.

Kalibro Metrics features include:

- Creation of configurations, i.e. a set of metrics to be used in the evaluation of a project
- Creation of new metrics (via Javascript) based on the ones provided by the metric collector tool
- Creation of ranges associated with a metric and a qualitative evaluation (may be plain text or HTML)
- Calculation of statistics results for higher granularity modules (e.g., average LOC of classes inside a package)
- Possibility of exporting results to a CSV file
- Indication of a grade for the software source code, based on given weights for metrics and grades for ranges (allows project comparisons)
- Possibility of making interpretation more user-friendly associating colors with ranges
- Download of source code from remote zip file, tarball or subversion repository

creative commons

Kalibro is delivered with the proper configuration to run using Analizo - a free and multi-language toolkit - as the source code analysis tool. Analizo supports the extraction and calculation of a fair number of source code metrics, generation of dependency graphs, and software evolution analysis. It efficiently parses source code written in C, C++ and Java.

Analizo, that is called from Kalibro, provides a simple YAML output containing global metrics concerning the whole project and module metrics for each module analyzed. The global metrics are divided in two subsets:

- Statistical metrics take into account the metric values of every module to calculate average, standard-deviation, variance, maximum, minimum, median, skewness and kurtosis.
- The Non-Statistical Metrics are project-wide metrics meaning that it is calculated just considering the source code information of the entire project and not the values of module metrics. They are:

- Total Number of Abstract Classes
- Total Coupling Factor
- Total Effective Lines of Code
- Total Methods Per Abstract Class
- Total Modules/Classes
- Total Modules with Defined Attributes
- Total Modules with Defined Methods
- Total Number of Methods

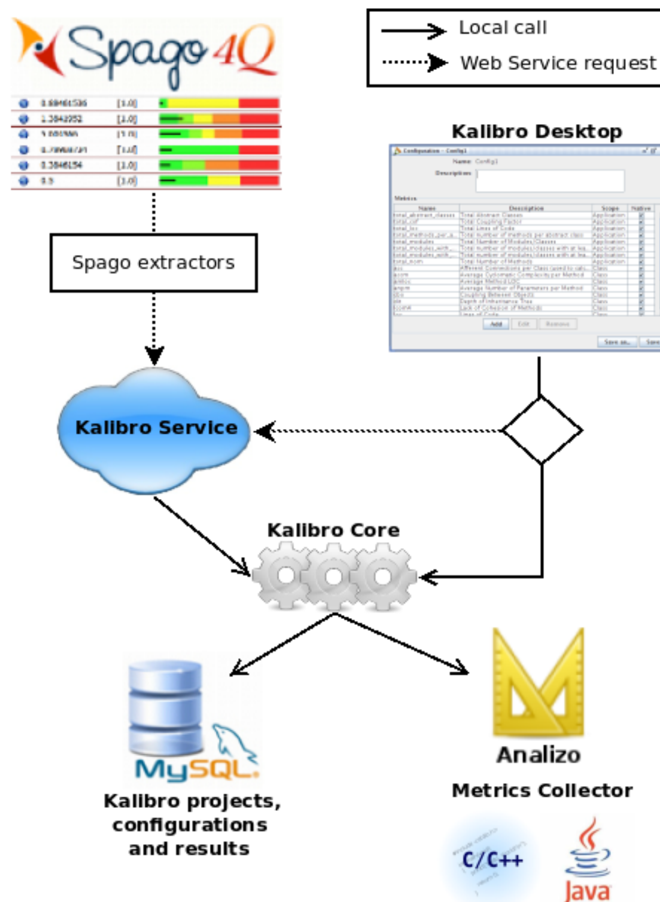
As for the module metrics, Analizo computes:

- Afferent Conexions per Class
- Average Cyclomatic Complexity per Method
- Average Method Lines of Code
- Average Number of Parameters per Method
- Coupling Between Objects
- Depth of Inheritance Tree
- Lack of Cohesion of Methods
- Lines of Code
- Maximum Method Lines of Code
- Number of Attributes
- Number of Children

creative commons

- Number of Methods
- Number of Public Attributes
- Number of Public Methods
- Response For a Class

Analizo was architectured to support the use of external tools as extractors. Doxyparse is the main extractor, it is capable of parsing C, C++, and Java (and possibly more languages) with a great speed. Doxyparse is based on Doxygen, a widely used open source tool for the generation of documentation through the code comments made in different programming languages. In summary, Doxyparse/Doxygen is what provides the multi-language and efficient source code analysis platform for the Kalibro.



Kalibro can be used in off-line mode (Kalibro Desktop) or as a web service client (Kalibro Web Service). Kalibro Desktop is the simplest way of using Kalibro Metrics. It provides all Kalibro Metrics features with a swing user interface.

Kalibro Desktop can also be used as a graphical user interface for Kalibro Service. In this case, database and metric collector installation is required only on the server side. So, Kalibro Service provides Kalibro Metrics features through a Web Service interface. It was created for integration with Spago4Q (detailed in the WD 5.5.4). Installing it has basically the same steps as Kalibro

creative commons

Desktop, but instead of the JAR file, a WAR file is provided for being loaded on a servlet container (tested only with Apache Tomcat).

Kalibro Service is integrated to Spago4Q through the KPI model, extractor and extraction process configuration. A simple Java class acts as a client for the Kalibro Web Service, simplifying the request-response interaction. Extractors use this class to provide an easy integration with Spago4Q. The upload extractor reads projects requests from the Spago4Q queue and sends a request to the Kalibro Web Service to load the projects. The analysis extractor queries the Kalibro database and provides metric values for Spago4Q.

7.2.1 Installation

Installation prerequisites

For using Kalibro Desktop as a client of Kalibro Service, the only prerequisite is the Java Runtime Environment 1.6 and a connection to the machine where Kalibro Service was installed.

In order to run Kalibro locally (either by desktop application or web service installed on your machine) you will also need access to a MySQL Server, besides installation and implementation of a collector to a metric results provider tool. Kalibro package already contains the implementation of a collector for the Analizo metric tool, so you will probably want to install it.

If you are installing Kalibro Service at your machine, you will also need the Apache Tomcat 6 servlet container.

Kalibro Service

1. Install Analizo toolkit: `http://analizo.org/Site/Download`
2. Download KalibroService.war and put it at the webapps directory of your Tomcat installation
3. Create an empty MySQL database for Kalibro, and a MySQL user with all privileges at this database
4. Download kalibro_ini_setup.sh script and run it in order to generate the ~/.kalibro/kalibro.ini configuration file for the service. As alternative, download kalibro.ini file and put it on a directory named .kalibro inside your home directory. Edit the settings as explained below:
    - metric-collector is the class used to obtain the metric values. Put org.software.analizo.AnalizoMetricCollector if you want Analizo as the metric collector tool. This class runs Analizo and parses its output. You can implement the MetricCollector interface and use it to run Kalibro with another metric collector tool.
    - loader-directory is the path to the directory where the submitted projects are going to be checked out. If the specified directory does not exist, it will be created.

creative commons

- maintain-sources speficies if the downloaded projects are going to be deleted after analysis. Change to no if you want the projects source code to remain at the downloads directory.
- Database section: here you have to specify the parameters of the database you configured at the previous step.

5. Start tomcat

Spago4Q integration

1. Download kalibro-spago-extractors.jar and put it at SpagoBI/WEB-INF/lib
2. Restart Spago tomcat
3. Download KalibroExtractorsConfiguration.zip and import it as extractors on SpagoBI. Edit the extractors operation parameter relative to the service endpoint

1. Download KalibroKpiDocument.zip and import it on SpagoBI *"Tools"->"Import/export"*

Kalibro Desktop – Quick install and usage

- Running Kalibro - As Service Client

Dependencies:

Install Java
```
$ sudo apt-get install sun-java6-jre sun-java6-fonts
```

Check if Java is correctly installed
```
$ java -version
The output should be something similar to:
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04)
Java HotSpot(TM) Server VM (build 17.1-b03, mixed mode)
```

Go to Kalibro web site: `http://ccsl.ime.usp.br/kalibro`

At 'Downloads' section, click on 'Kalibro Desktop (.jar)' and save the file.

Open a console, go to download destination folder and run:
```
$ java -jar KalibroDesktop.jar
```

If it is the first time you run Kalibro or by any other reason there is not a .kalibro/kalibro.ini file in your home directory, Edit Settings window will show and you will be able to set up Kalibro as a service client or to run locally.

If you just want to try Kalibro quickly to check it's features, it is possible to use the testing server as shown bellow. However, remember that it is just a testing server and there is no guarantee that data stored there will be safe. Because

creative commons

that, we discourage the usage of this server for any purpose greater than testing.

Having that in mind, you can configure Kalibro to use the testing following this few steps:

On Edit Settings window,
- Check 'Use as service client' option;
- Fill 'Service endpoint' with:
`http://quality-demo.qualipso.org/KalibroService/`
- Let 'Listener latency' as it is
- Click on 'Ok' button

That is it. Now, you can check all Kalibro features.

Observations:

- Notice that when Kalibro is configured to run as a client, you must be connected somehow to the server (trough the internet or in LAN). Otherwise, an exception will be thrown and the application will not start.

- It is highly recommended that you find a server other than our testing one or configure Kalibro to run locally. It is always possible to change this settings by clicking on 'Settings' and 'Edit'.

- If by any chance you change wrongly the settings and Kalibro stop working, do not panic. Just delete $HOME/.kalibro/kalibro.ini file, and next time you run Kalibro, edit settings window and you will be able to reconfigure the application.

- Running Kalibro - Local Desktop (off-line)

Dependencies

Analizo:

1) Create a file `/etc/apt/sources.list.d/analizo.list` file with the following contents:

```
deb http://analizo.org/download/ ./
deb-src http://analizo.org/download/ ./
```

2) Add the signing key you your list of trusted keys:

```
$ wget -O - http://analizo.org/download/signing-key.asc |
apt-key add -
```

3) Update your package lists:

```
$ apt-get update
```


creative commons

4) Install analizo:

```
$  apt-get install analizo
```

Java

Install Java:
```
$ sudo apt-get install sun-java6-jre sun-java6-fonts
```

Check if Java is correctly installed:
```
$ java –version

The output should be something similar to:
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04)
Java HotSpot(TM) Server VM (build 17.1-b03, mixed mode)
```

MySQL:

Install MySQL

```
$ sudo apt-get install mysql-server mysql-client
```

Change MySQL root password. In the following example, replace 'new_password' by the password you wish.

```
$ sudo mysqladmin -u root -h localhost password
'new_password'
```

Now login as root user and insert your MySQL root password you have just changed.

```
$ mysql -u root -p
```

The next step is to create a database. You can give any name you want to the new database. For convenience, we will use kalibro.

```
mysql> create database kalibro;
```

Do not forget the semicolon ";" at the end of each command line.
Once we have created the database, we only need to create a user and give him access permission. It is possible to do that with the commands below:

```
mysql> create user 'kalibro'@'localhost' identified by
'kalibro';
```

Notice that we have created a user with 'kalibro' as login and password, but you are free to create the login and password you wish.

creative commons

```
mysql> grant all on kalibro.* to 'kalibro'@'localhost';
mysql> flush privileges;
```

The commands above give to kalibro user all permissions on all tables in kalibro database and then flush MySQL privileges table to ensure that all changes will be applied.

Finally, you can quit MySQL shell:

```
mysql> exit;
```

Kalibro:

Go to Kalibro web site: `http://ccsl.ime.usp.br/kalibro`

At 'Downloads' section, click on 'Kalibro Desktop (.jar)' and save the file.

Running Kalibro:

Locally

Open a console, go to download destination folder and run:

```
$ java -jar KalibroDesktop.jar
```

If it is the first time you run Kalibro or by any other reason there is not a.kalibro/kalibro.ini file in your home directory, Edit Settings window will pop and you will be able to set up Kalibro as a service client or to run locally.

Here we will explain how to configure Kalibro to run locally. So, the first thing to do in Edit Settings window is to make sure 'Use as service client' box is not checked.

Then, you can set the metric collector class you want to Kalibro use. If you do not have a metric collector, it is possible to use Analizo, which is the default metric collector. In order to do that, fill 'Metric collector' text box with:

```
org.softwarelivre.analizo.AnalizoMetricCollector
```

After that, fill database information according the way you have configured your MySQL. Following the settings we have suggested so far, the four fields in database area should be filled as described below.

```
Host: localhost:3306
Database name: kalibro
Username: kalibro
Password: kalibro
```

Finally, click 'Ok'.
That is it. Now, you can check all Kalibro features.

creative commons

Observations:

- If by any chance you change wrongly the settings and Kalibro stop working, do not panic. Just delete $HOME/.kalibro/kalibro.ini file, and next time you run Kalibro, edit settings window pop and you will be able to reconfigure the application.

7.2.2 Tool Usage

Kalibro Desktop

At first Kalibro run, the blank main screen is shown. From this screen, it is possible to access all features.

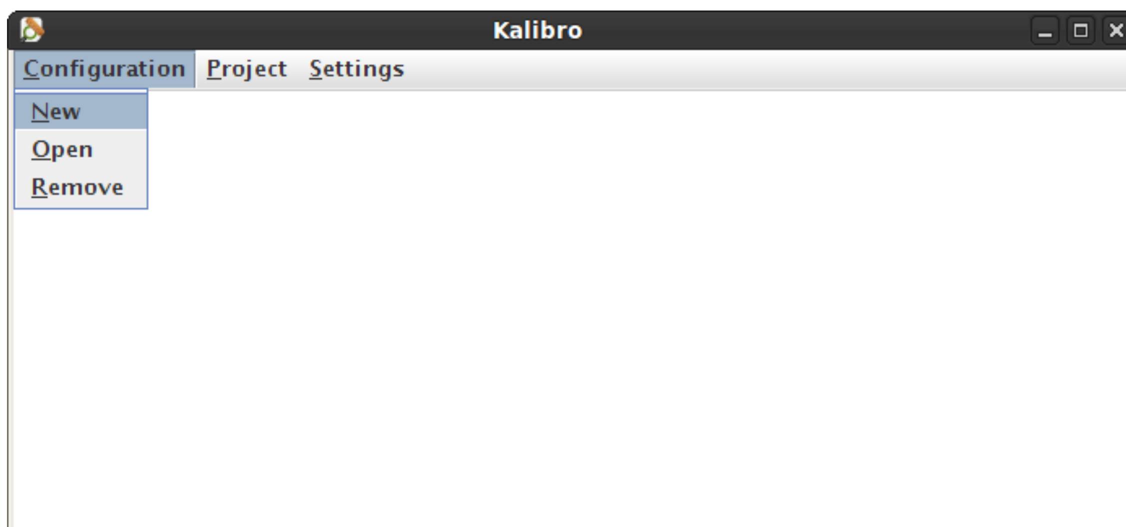


Figure 6: Configuration menu

From Configuration menu, it is possible to create a new configuration, open or remove one created previously. A configuration is a set of thresholds associated with qualitative evaluation, including comments and recommendations. These configurations are used to enhance metric results interpretation.

By clicking on *New* option, a new window will pop where a name to the new configuration must be given.

creative commons

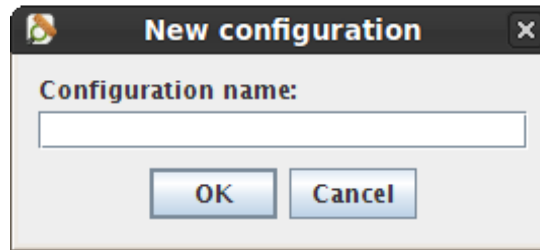


Figure 7: New configuration window

Once it is done, a new window with configuration characteristics will be displayed.

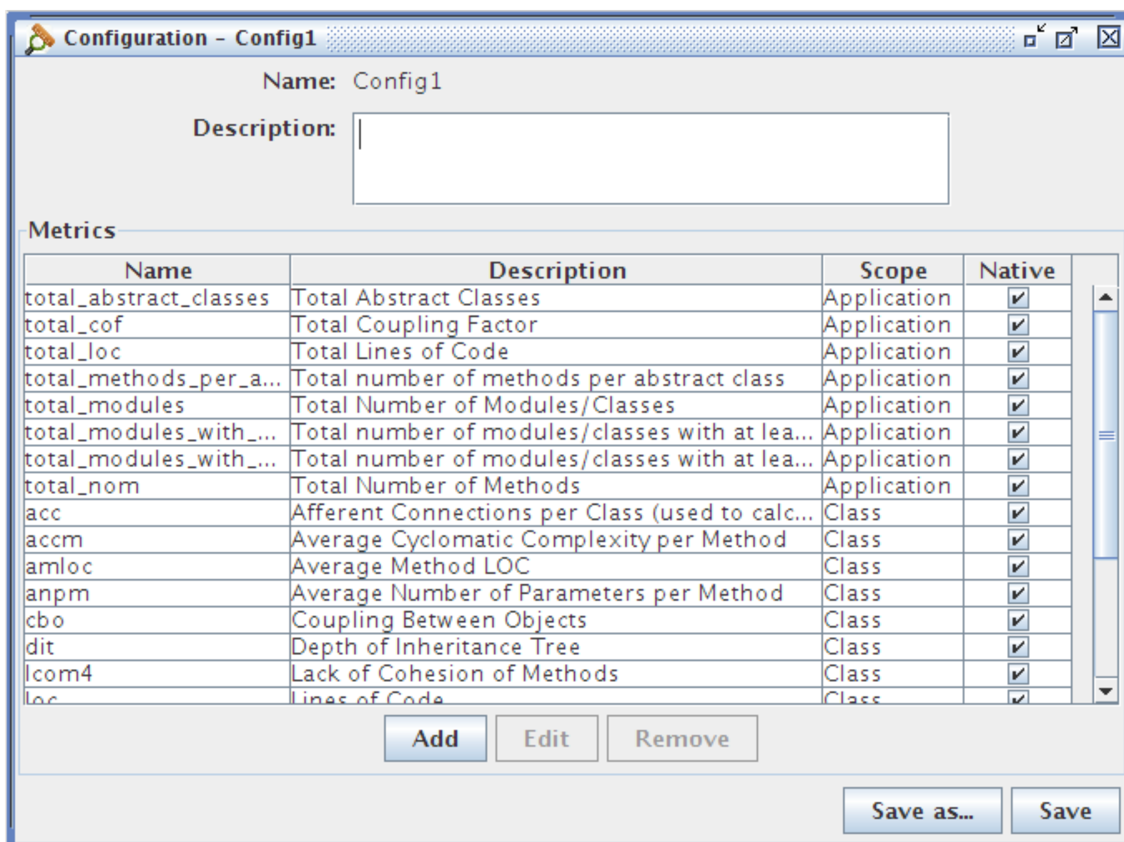


Figure 8:Configuration description window

In this window it is possible to set a description to the configuration and to verify the metric list. The metric list shows all metrics associated with the configuration, their description, the scope (or granularity, e.g. Class or Method) and a flag indicating if the metric is native from metric extractor, Analizo by default, or not.

Besides that, from this window it is possible to manage configuration's metrics by three basic operations: *Add*, *Edit* and *Remove*.

- **Add**: By default, all native Analizo metrics are loaded when a new metric is created. But, if a native metric were previously removed or not loaded,

creative commons

when *Add* button is clicked, a window will pop where the kind of metric must be select. Otherwise, compound type will automatically be selected.

If the chosen metric extractor does not supply metrics which meet your specific needs, you are able to create new metrics based on native ones.

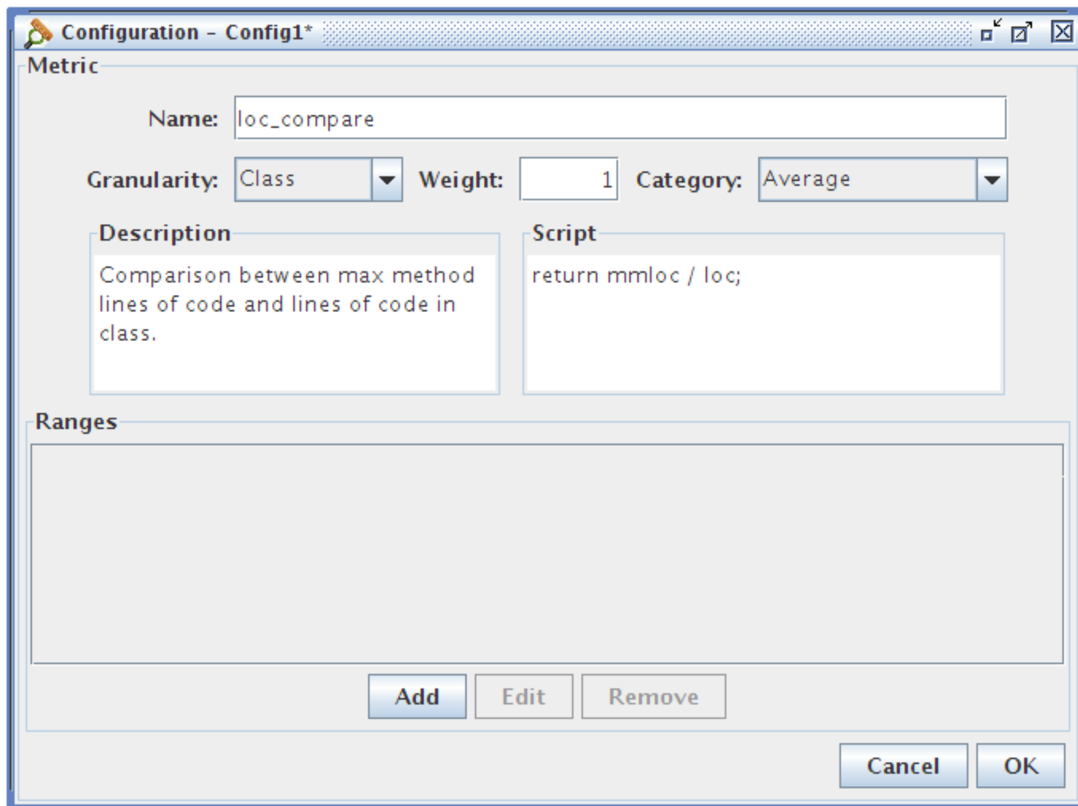


Figure 9:: New metric example

Besides *Name* and *Description*, in this window it is possible to set metric *Granularity*, a *Weight* this metric will have when calculating result grade, the *Category* it best fits and, finally, the *Script*.

The script used to calculate the metric must be written in Java Script syntax and it is possible to get any other metric result just by using the metric name as variable. Figure 5 shows a simple example of compound metric.

Below the *Description* and *Script* boxes, there is the *Ranges* area. The ranges are value intervals used to make metric results friendlier giving them an understandable meaning.

By clicking on *Add* button a window as shown on Figure 6 will appear.

creative commons

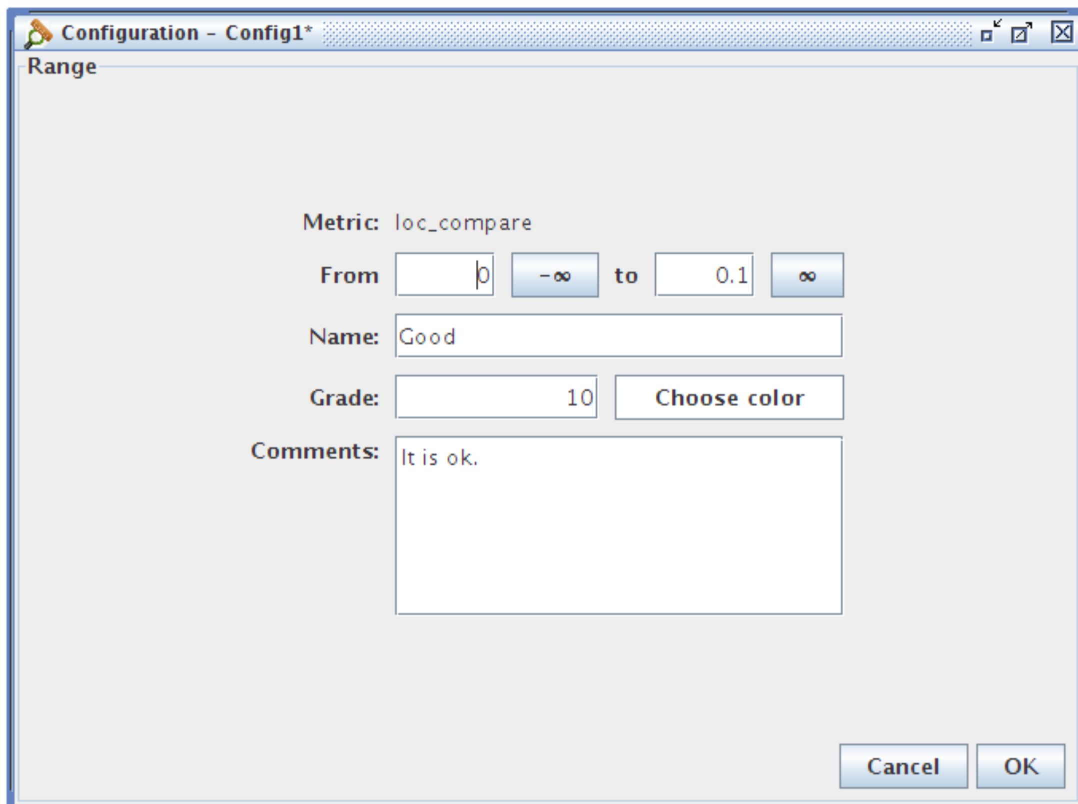


Figure 10:: New range

In this window, it is possible to set the boundaries of the range, a *Name*, a *Grade*, a *Color* and write a *Comment* which will be displayed when a result fits in that range.

IMPORTANT: the numeric interval in ranges are left-closed and right-opened. For example, in mathematical notation, the interval on Figure 6 is [0, 0.1[

Once some ranges are set, the metric screen will be like Figure 7:

creative commons

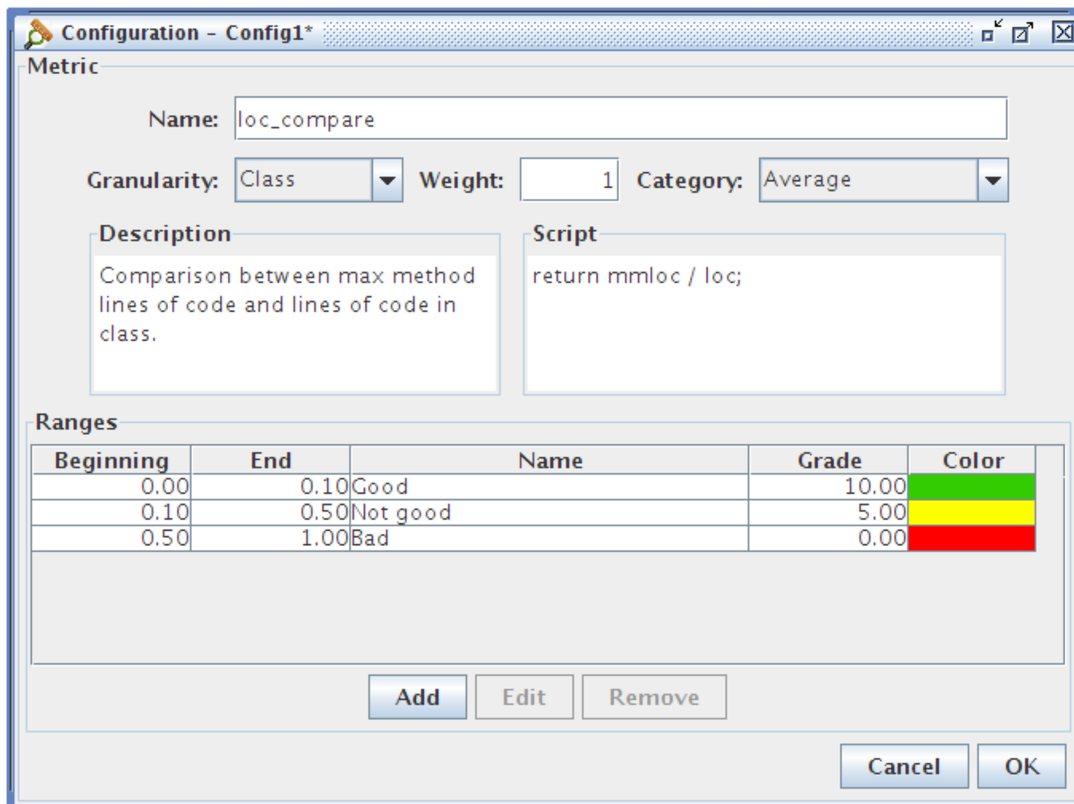


Figure 11: Metric with ranges

- **Edit**: By clicking on edit button, the same window shown on Figure 5 will pop. But all fields will only be editable if the chosen metric is compound. Otherwise, *Name, Granularity* and *Script* will be locked.
- **Remove:** It is possible to remove any metric, compound or native.

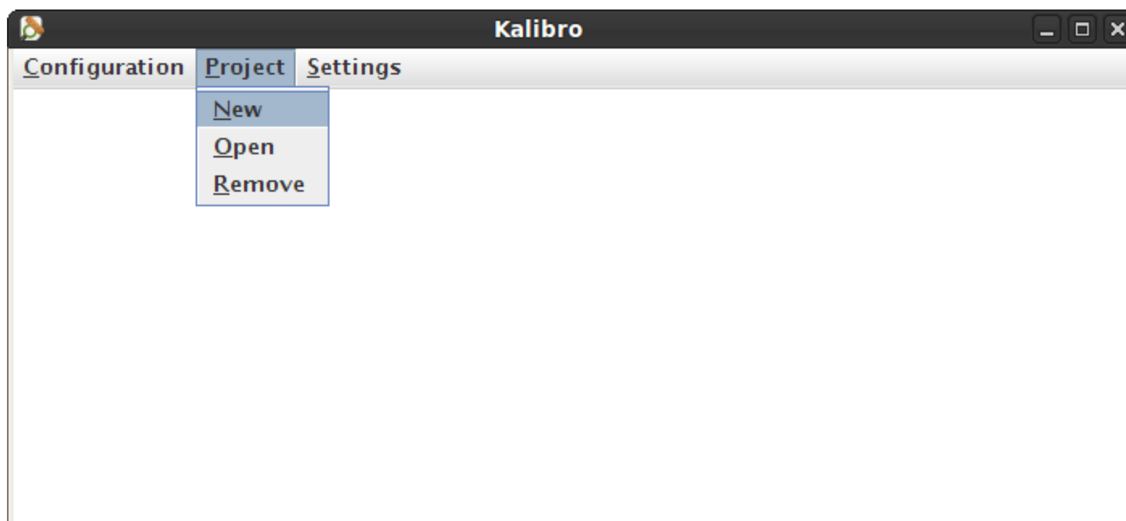


Figure 12: Project menu

Project

In *Project* menu, you can create a new project, open or remove one. Opening and removing a project are basically selecting the name on a list. So only *New* project option will be explained in detail.

creative commons

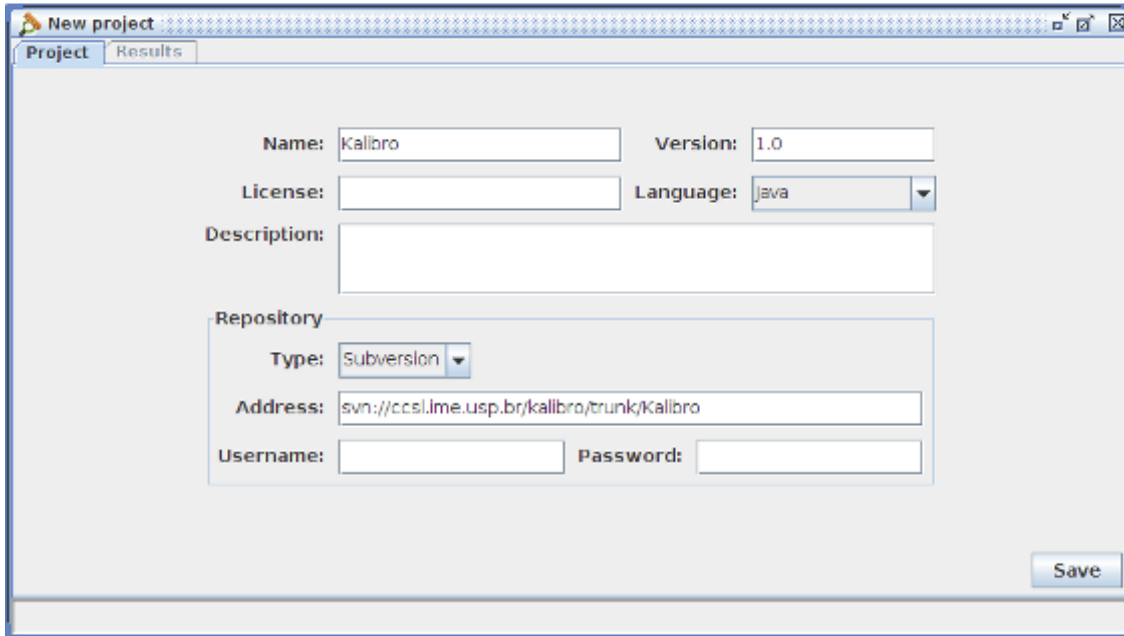


Figure 13: New project window

Once *New* option in *Project* menu is clicked, the window shown on Figure 9 will pop. There, *Name*, *Version* and *Language* fields must be filled.

In *Repository* section, the path to source code must be specified. If the code is stored in a local folder, then select *Directory* as *Type* and fill *Address* field with the full path to the code. When the code is stored in a subversion repository, change *Type* to *Subversion*, fill *Address* with repository URL and, if necessary, set *Username* and *Password* used to checkout the project. Besides these options, it is also possible to give an *Address* pointing to an URL of a .tar.gz or .zip file. In that case, *Type* must be selected as *Tarball* or *Zip*, respectively.

After that, the project will be loaded and metrics will be calculated. When this task is finished, the window switches automatically to *Results* tab as shown bellow.

creative commons

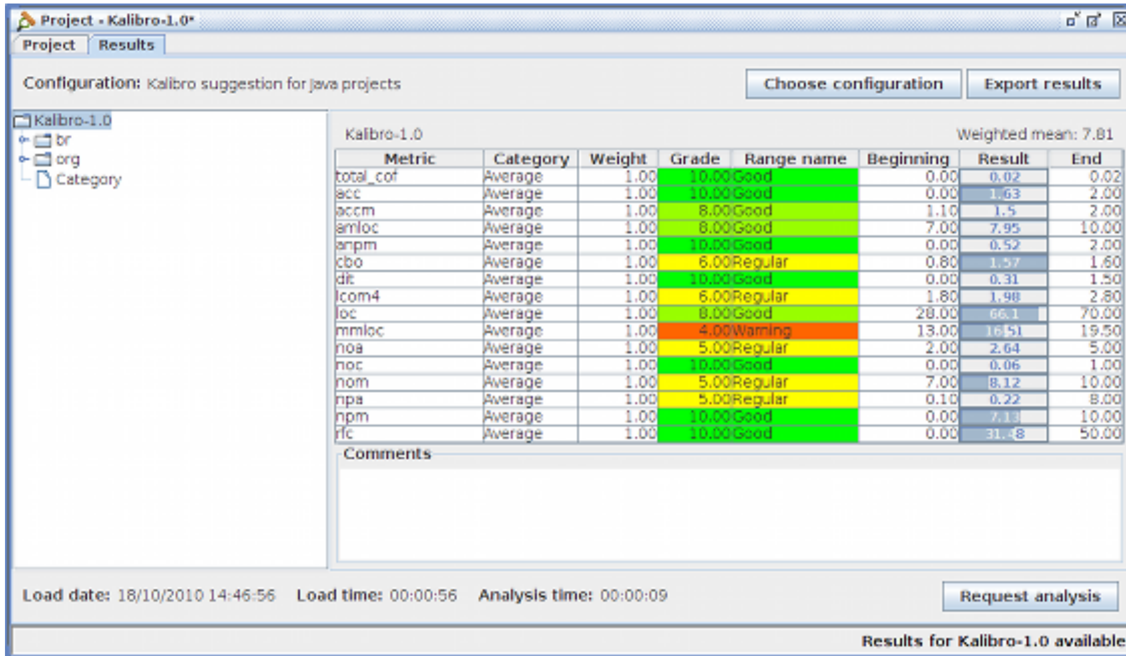


Figure 14: Results tab

On the left, there is a file tree where it is possible to browse through packages and classes. When the entire application or a package is selected, the result table will display statistical values for each metric, such as average and median. But, if a class is selected, its specific results are shown.

By clicking on a metric result in table, the *Comments* box will show the comment associated with the range in which the metric result fits. So, if the range has any hint about what could be done to improve that result, it will be easy to see the message. Figure 11 shows an example.

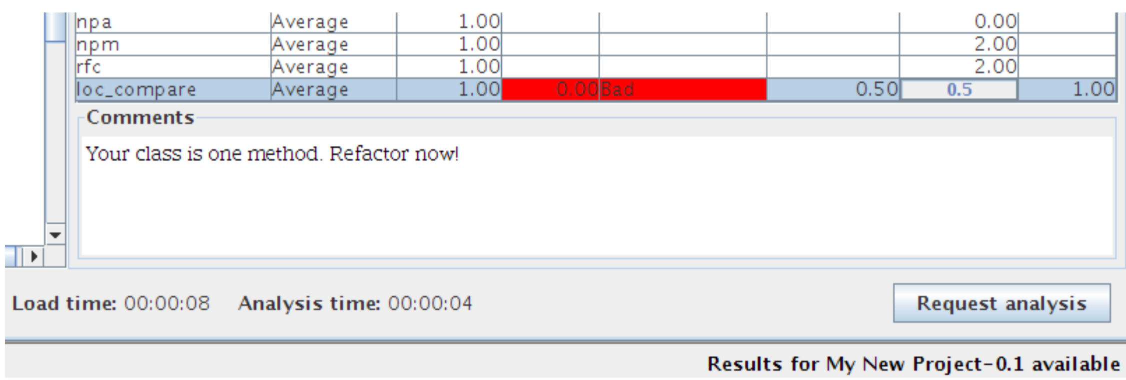


Figure 15: Zoom at comment about a bad result.

When the project has changed, all metrics must be recalculated. It is possible to do so just by clicking on *Request analysis* button.

If you have more than one configuration, it is possible to choose which one will be used clicking on *Choose configuration* button. Notice that there is no

creative commons

need to rerun the metrics calculation once the results will not change. What is going to change is only the ranges used to compare these results.

Other Kalibro feature is that when a configuration is chosen, this association with the project is stored and, next time that same project is open, the chosen configuration will be loaded automatically.

Finally, by clicking on *Export results* button, it is possible to save a csv file with the results shown in the result table.

Settings

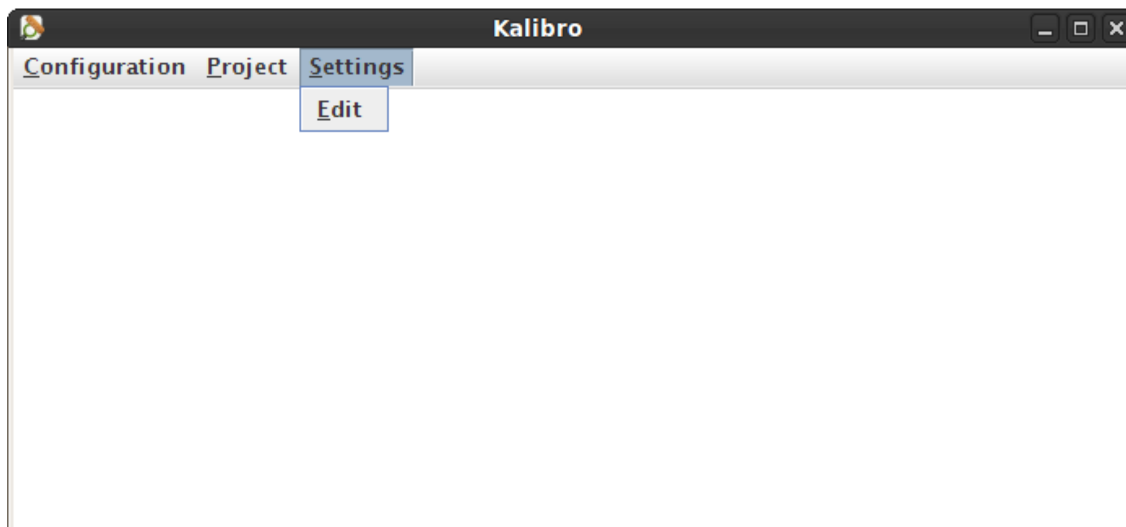


Figure 16: Settings menu

Before trying to change Kalibro settings, all configuration and project windows must be closed. Once it is done, *Edit* menu is unlocked.

By clicking on *Edit* menu, the window shown on Figure 13 will pop and there some Kalibro settings may be changed.

creative commons

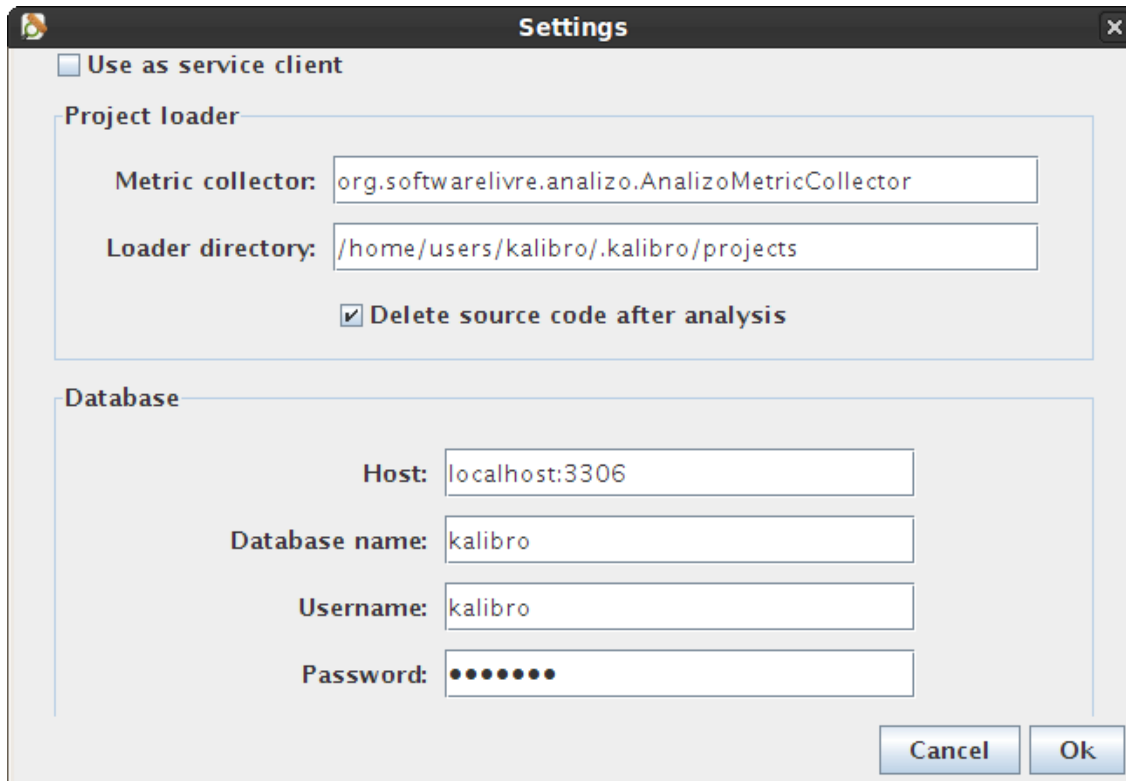


Figure 17: Settings window

The first option available is *Use as service client* check box. If it is checked, the window will show different options.

In service client mode, Kalibro will try to connect to a server which will run the analysis and only send back the results. The server address is set in *Service endpoint* field. The frequency that Kalibro will request results from server can be set in *Listener latency* field.

If Kalibro is not in service client mode, it is needed to configure how projects will be loaded and analysed and the database where results will be stored.

In *Project loader* box, *Metric collector* indicates the class used as metric extractor. *Loader directory* indicates the full path to a directory which will store all source code loaded. If there is no need to keep the code after having the analysis result, *Delete source code after analysis* check box may be unchecked.

In *Database* box, it is possible to specify database host address in *Host*, the *Database name*, and authentication information in *Username* and *Password* when it is required.

7.2.3 Developer Guide

Download Kalibro source code from http://gitorious.org/kalibro:

```
$ git clone git://gitorious.org/kalibro/kalibro.git
```

- Eclipse IDE Imports

creative commons

On Eclipse IDE import this directories (projects) in the following order:

- Libraries
- Resources
- KalibroCore
- KalibroServiceSupport
- KalibroService
- KalibroServiceClient
- KalibroSpago
- KalibroDesktop
- KalibroTest

- Update Kalibro Library

  WHEN: Before running or exporting KalibroService, if the projects KalibroCore or KalibroService were modified
  STEPS:
  1) Righ-click on project "KalibroCore"->"Export..."->"Java"->"JAR file"
  2) On "Select resources to export" select only KalibroCore/src
  3) Select "Export generated class files and resources"
  4) On export destination put "Libraries/Kalibro/kalibro-core.jar"
  5) Click "Finish"
  6) Execute the previous steps for KalibroServiceSupport (kalibro-service-support.jar)
  7) Refresh the workspace

- Update Service Client

  WHEN: After modifying the KalibroServiceProxy.java or any of its direct dependencies
  STEPS:
  1) Right-click on project "KalibroService"->"Run as"->"Run on server". The console should not show exception stack traces
  2) The "Web Services" page is shown. At the column "Information", copy the WSDL, something like:
  http://localhost:8080/KalibroService/?wsdl
  3) At KalibroServiceClient project, delete everything in the source folder except the package
  br.usp.ime.ccsl.kalibro.service.client
  4) At KalibroServiceClient folder, execute the following command, where <WSDL> comes from step 2:
  wsimport -d bin -s src <WSDL>
  5) Refresh the contents of KalibroServiceClient project.
  6) At KalibroServiceClient, delete everything from the the following package, expect ObjectFactory.java:
  br.usp.ime.ccsl.kalibro.service.beans

creative commons

7) Add "extends KalibroServicePort" at the interface KalibroService.java on KalibroServiceClient project.

This should not cause compilation problems

8) Resolve problems at KalibroServiceClient.java according to the new service definition

9) At the "Servers" view, shut down the KalibroService

10) Annotate with @SuppressWarnings("all") every generated class with warnings

- Export Kalibro Desktop

STEPS:

1) Right-click on project "KalibroDesktop"->"Export..."->"Java"->"Runnable JAR file"

2) On "Launch configuration" choose "KalibroController - KalibroDesktop"

3) Choose the destination .../KalibroDesktop.jar

4) On "Library handling" choose "Package required libraries into generated JAR"

5) Click "Finish"

- Export Spago Extractor

1) Right-click on project "KalibroSpago"->"Export..."->"Java"->"JAR file"

2) On "Select resources to export" select the source folder of KalibroSpago, KalibroServiceClient, KalibroServiceSupport and KalibroCore

3) Select "Export generated class files and resources"

4) Choose the destination .../kalibro-spago-extractor.jar

5) Click "Finish"

creative commons

7.3 OSLC

Version: 0.2

Project home: http://forge.ow2.org/projects/oslcv3

Web-service and extractor for Spago4Q : Work Folder/A6/WP6.4/Tools/OSLC-Tool

License: GPLv2

Integration into Spago4Q: OSLC is partially integrated with Spago4Q with a specific extractor which interoperates with the OSLC web-service specially implemented for the Activity 5

Issues: OSLC is integrated as proof-of-concept

Open Source License Checker (OSLC) is a risk management tool for analysing open source software licenses. Compared to Fossology, the previous Intellectual Property Rights (IPR) management tool we chose earlier during the project, the pros for OSLC are that OSLC is faster, lighter, easier to install and multi-platform (it is a Java program). However, OSLC can only analyse source code written in Java, Javascript, PHP, Python and C/C++. This set of programming language is large enough for the purposes of the MOSST model developed in A5. Like for Fossology, the goal of using OSLC is to retrieve some metrics about the licenses of a project to assess with the MOSST model. These metrics are for example the number of unlicensed files, the number of distinct licenses incorporated, the number of copyright holders, etc. Then, to automate the use of OSLC and avoid the intervention of a third party, we have decided to enclose the core of OSLC inside a web-service deployable into Spago4Q. A Spago4Q extractor does the link between the SpagoBI/Spago4Q engine with the OSLC web-service (WS).

The integration with Spago4Q has been partially carried out as a proof of concept.

7.3.1 Available metrics

The OSLC tool through the use of its web service can retrieve the following metrics :

- Number of files,
- Number of sources files,
- Number of licensed files,
- Number of unlicensed files,
- Number of files with uncertain licenses,
- Number of distinct licenses,
- Number of "reference" conflicts,
- Number of "global" conflicts,
- Number of copyrighted files,

creative commons

- Number of copyright holders.

7.3.2 Installation

Installation prerequisites

The environment requirements to run the OSLC web-service and extractor are :

- JDK above 1.6;
- Tomcat above 6.0;
- Spago4Q 2.3;

Software components

- The web-service :
  - OSLCService.war : the web-service to deploy in the Tomcat server.
- The Spago4Q's extractor and its configuration :
  - extractor-oslc-1.0.jar : the extractor;
  - oslc_extractor.zip : an archive containing the Spago4Q configuration metadata of the extractor.

All the software components are available on the Qualipso portal in the following folder “Work Folder/A6/WP6.4/Tools/OSLC-Tool”.

Installation of the web-service

To “install” the OSLC web-service, you need to deploy the OSLCService.war into Tomcat. For this, please respect the following steps :

1. Shutdown the Spago4Q's Tomcat server ([Tomcat directory]/bin/shutdown.sh for a standalone Linux installation of Tomcat);
2. Copy the OSLCService.war into the [Tomcat directory/webapps] folder;
3. Start your Tomcat server ([Tomcat directory]/bin/startup.sh).
4. Test if the web-service has been successfully deployed by reaching the wsdl description file of the web-service at http://<host>:<port>/OSLCService/services/OSLCService?wsdl
5. If you need it, you can configure the folder where the OSLC web-service will store temporarily the source code of the project to assess and the folder containing the licenses patterns. You can edit these paths in the [Tomcat directory]/webapps/OSLCService/WEB-INF/classes/oslcws.properties file.

To test deeper the web-service you can use a web-service testing tool such as SoapUI. The parameters you will have to give are :

- svnUrl : the svn url where the source code of the project to assess is hosted (mandatory);
- username : the username that allows to connect to the SVN repository where the project to assess is hosted (optional);

creative commons

- password : the password that allows to connect to the SVN repository where the project to assess is hosted (optional);
- revision : the svn revision number of the project to checkout. Let it empty if you want to checkout the HEAD revision (optional).

Installation of the extractor

To “install” and set up the extractor with Spago4Q, you need to copy the extractor jar file into a dedicated folder in Spago4Q and then load the extractor configuration by using the Spago4Q administration panel. The different steps are :

1. Shutdown the Spago4Q's Tomcat server ([Tomcat directory]/bin/shutdown.sh for a standalone Linux installation of Tomcat);
2. Copy the extractor-oslc-1.0.jar in [Tomcat directory]/webapps/SpagoBI/WEB-INF/lib;
3. Start your Tomcat server ([Tomcat directory]/bin/startup.sh);
4. Log into the Spago4Q administration panel;
5. Go to Tools>Import/Export and import the oslc_extractor.zip archive;
6. In the operation parameters of the extractor configuration, we defined the the web-service endpoint. You can adapt the value of this parameter to your needs.
7. In the “Interface Type Detail” page, you should create the fact table by clicking on the appropriate button. The fact table will store the values of the metrics of the project you will assess.
8. Finally, you can run the extraction in the “Extraction Process Detail “page. If the the extraction have been run successfully, normally a new record will be inserted into FT_OSLC table.

creative commons

8. ANALYSIS OF PROJECT DEVELOPMENT

8.1 StatCVS and StatSVN

Version: StatCVS: 0.5.0,

StatSVN: 0.5.0

Project home: http://statcvs.sourceforge.net/

http://www.statsvn.org/

Location: http://abelia.man.poznan.pl/stattools/statcvs-statsvn-package-0.8.zip

(Software components and documentation for StatCVS/StatSVN integration with Spago4Q platform).

License: LGPL (StatCVS)

LGPL (StatSVN)

LGPL (Integration of StatCVS-StatSVN with Spago4Q)

Integration into Spago4Q:

Both StatCVS and StatSVN tool are integrated with Spago4Q 2.0 platform as a custom extractor. StatCVS and StatSVN are managed as Stat Tools Web Service, which is directly invoked by the custom extractor.

Known issues: The execution of the integrated tools is a time-consuming process (depending on the project size). It is related to the server capabilities which hosts the CVS/SVN repository and the Internet connection.

StatCVS and StatSVN are used to generate statistics on a basis of information retrieved from CVS[2] or SVN[3] repositories and create reports regarding the process of project development. Such data is provided both for overall project characteristics/features and in respect to individual authors, giving insight into their activity in project. Besides the history of commits, these tools collect also data regarding the size of projects in both number of files and lines of code. The generated statistics take into consideration a number of other project information which are thoroughly assembled and described at the projects' web sites.

StatCVS and StatSVN were chosen due to their usefulness in gathering repository statistics. With these applications it is possible to collect data from two most popular version control systems. In addition, these tools are closely related as StatSVN is based on StatCVS application, hence both have similar architecture, capabilities and ease of use. Moreover they use the same data

2 CVS home, http://www.nongnu.org/cvs/

3 Subversion home, http://subversion.tigris.org/

creative commons

model and report generators. Thanks to their similarity the process of integration of StatCVS and StatSVN tools can be carried out almost simultaneously, saving work and time needed in case of combining two quite different solutions.

8.1.1 Installation

Both StatCVS and StatSVN are command line tools, which can be downloaded from project's home pages[4,5]. The statistics tools for repository analysis need the CVS and SVN command line clients in order to create CVS and SVN log files respectively. They also require Java Runtime Environment in version 1.5.x (1.6 recommended) or above. StatCVS supports both CVS for Unix and CVSNT for Windows. The repository statistics tools require CVS and SVN client to access CVS and SVN repositories respectively. Both tools are provided as zip archives and can be installed by decompressing the downloaded archive. For detailed description of installation and quick start please refer to the StatCVS online manual[6] and StatSVN wiki pages[7] respectively.

8.1.2 Tool Usage

StatCVS and StatSVN tools are command line applications, which generate HTML or XML reports. These reports cover a number of qualities, which can be divided into the following groups:

- File and directory related metrics,
- Developer activity related metrics.

Taking advantage of generated reports and the statistics they provide, the following metrics, which are considered in GQM model can be calculated:

- Average number of changed lines per year,
- Number of major releases per year,
- Number of minor releases per year.

In the GQM model, these measures are used to assess the GQM quality focus defined as “Actual exploitability in development: maintainability”. However, the aforementioned metrics cannot be obtained directly from reports, but there is a possibility to calculate them manually when using the following statistics as a foundation:

- lines of code added,
- lines of code removed,
- lines of code changed,

---

4 StatCVS homepage: http://statcvs.sourceforge.net/

5 StatSVN homempage: http://www.statsvn.org/

6 StatCVS manual: http://statcvs.sourceforge.net/manual.html

7 StatSVN wiki pages: http://wiki.statsvn.org/User%20Manual.ashx

creative commons

- major and minor release rate based on project tag list.

The manual data extraction and collection is a tedious task. Therefore StatCVS and StatSVN integration wraps these tools and combines them with Spago4Q through the Spago4Q extractor interface. It also calculates the values of metrics, providing the final results as a part of Sapgo4Q reports. The description of the integration is available in section 9.3.

creative commons

9. Analysis and Assessment Tools

9.1 Spago4Q

Version:	Spago4Q V2.3.1
Project home:	http://www.spago4q.org/ Sources and Binaries (Internet): http://forge.ow2.org/project/showfiles.php?group_id=301
Documentation:	http://wiki.spago4q.org/xwiki
Provider:	Engineering Ingegneria Informatica SpA
License:	LGPL
Known issues:	Information about known issues can be found on the project site[8].

Spago4Q – SpagoBI for Quality – is a Free Open Source Software platform which intends to support the maturity assessment, the efficiency of the software development process and the quality inspection of the software which is ready to be released: these goals are achieved by evaluating data and measures collected from various project management and development tools with non-invasive techniques. A more detailed description of Spago4Q can be found in Appendix B.

In the QualiPSo project Spago4Q is used to represent metrics for product evaluation. These metrics are part of the GQM defined by A5 in WP 5.3.2[3] and implemented inside this tool. In addition Spago4Q is used as well to represent the metrics for process evaluation defined by A6, so that product and process related data can be presented in a common view.

Since Spago4Q is developed by one of the QualiPSo partners it can be more easily adapted to QualiPSo-specific requirements.

9.2 Spago4Q Extension for QualiPSo

Version:	1.2
Sources, Binaries and Documentation:	http://www.qualipso.org/spago4Q-tool
Provider:	Engineering Ingegneria Informatica SpA
License:	LGPL

This extension has been developed especially for the QualiPSo project to send via Web Service all the information needed to execute the product and process analysis, by the analysis tools, for every specific product.

This extension also performs some operations inside the Spago4Q platform, automatically, to ease the configuration steps such as, the creation of the

8 https://wiki.spago4q.org/xwiki/bin/view/Content/KnownIssues

creative commons

resource (product) and its association to one or more quality models (for the analysis and evaluation,of a project) according to the information sent.

Other utilities, developed as Web Services are used to list the current projects, retrieve the details of a selected one, as well as update a project set of information or delete the project from the list.

A special abstract java class, for the development of the extractor components, provides a couple of utility methods to access the list of products and retrieve their detail information. This to support the extraction processes that are executed automatically once a product is loaded into the Spago4Q platform using the specific Web Service.

Spago4Q-QualiPSo bundle 1.2

A special bundle is available as a package containing the:

1. **platform binaries** (SpagoBI, Spago4Q, Spago4Q extension for QualiSPo)
2. **Tomcat configuration file**
3. **database dump** for a clean installation
4. **models**: the export packages by model for the analysis tools and the assessment or evaluation
5. **extraction configuration**: the export packages containing the configuration of the extraction processes used by analysis tools to load into the Spago4Q platform their analysis results.

9.2.1 Installation

Spago4Q can be installed using the QualiPSo bundle or over an existing SpagoBI environment. Here is described the first method while the second one is available from the Spago4Q wiki. The requirements for the installation are listed below:

Spago4q-qualipso-1.2	http://www.qualipso.org/spago4Q-toolhttp://www.spagobi.org/
Java SE JDK (>= 1.6)	http://java.sun.com/javase/downloads
Apache Tomcat	http://tomcat.apache.org
MySQL	http://www.mysql.org

Note: The versions used during the development are: JDK 1.6 - MySQL 5.0.51b community, 5.0.58, 5.0.67 community - Apache Tomcat 6.0.18.

Create an account on the database for Spago4Q metadata. Create a schema with the same name and give to the user all the grants on this schema.

MySQL	localhost:3306

creative commons

User	spago4q
Password	spago4q
Schema	spago4q

The steps to install the Spago4Q platform are:

1. **Web applications deployment**: from the bundle package copy the four web application folders
    a. SpagoBI
    b. SpagoBIBirtReportEngine
    c. SpagoBIJasperReportEngine
    d. SpagoBISDK

   into the tomcat *webapps* folder. The *SpagoBI* and *SpagoBISDK* folders already contains the Spago4Q platform and the extension developed for the QualiPSo project.
2. **database driver deployment**: copy the MySQL driver

   mysql-connector-java-3.1.13-bin.jar

   from the bundle package to the tomcat lib folder for the common libraries
3. **tomcat configuration**: update the *server.xml* configuration file of the tomcat installation adding
    a. four environment values
    b. one jndi resource

   use the *server.xml* copy from bundle package as example for the configuration (from line 47 to 59)
4. **database import**: import into the spago4q schema the database dump to start with a clean platform
    a. s4q-qps clean [*date time*].sql

The installation is completed and the Spago4Q Platform is ready to be extended with the integration of the analysys tools.

For more detailed and updated information refer to the Spago4Q documentation on the wiki pages.

9.2.2 Tool Usage

Due to the number of features, and their complexity, the tool usage description is forwarded to the wiki pages. Here are briefly reported the most relevant ones.

creative commons

- **Data Extraction**: thanks to the Extractors Components it's possible to gather information from different environments and store them in a centralized data warehouse.
- **Model Definition/Instance**: with this component it's possible to define an n-level tree that represent an Assessment Model (MOSST, OMM, CMMi, ISO …) and create as many instances are needed.
- **KPI**: the platform allows to define KPIs in terms of descriptive information, computational rules, and thresholds. After the definition of a set of KPI it's possible to associate them to the Model Definition/Instance nodes.
- **Analytical Document**: starting from a Model Instance it's possible to define an analytical document that present to the platform user the Model with its KPI computed.
- **Product Management**: this feature comes from the extension developed for the QualiPSo project. It provides the product information Web Services and the abstract java class for the extractor component development.

9.3 Quality Platform

Version: 1

Project home: http://www.qualipso.org/node/538

Location: www.qualipso.org

Provider: Università dell'Insubria

License: LGPL

Known issues: None.

Quality Platform is an integration tool to provide a GUI to the people that will carry out a software assessment with the A5 toolset. It manage the complete process of project analysis and result visualization in a completely automated way.

Quality Platform provide an easy interface to retrieve the results of a project analysis.

Quality Platform also allows Spago4Q authorized users to analyze a new project simply by inserting a set of metadata for the project to be analyzed (project name, version, repository…).

The user management is delegated to Spago4Q and quality platform share the same permission of Spago4Q itself.

9.3.1 Installation

1) Download and unzip the file http://qualipso.org/sites/default/files/quality-platform_0.zip

creative commons

2) Deploy the war file into your Tomcat/Webapps folder

3) Edit the configuration file with spago4q local installation parameters:

$TOMCAT_HOME/webapps/qualipso-platform/WEB-INF/classes/conf.properties

4) Run the platform eg. http://localhost:8080/qualipso-platform

9.3.2 Tool Usage

Quality Platform allows users to get the available quality reports from Spago4Q.

Qualipso Quality Platform for Open Source Software

Trust and Quality in Open Source Systems

Home | Qualipso Community | Download Tools

Actions
- Quality Report

Models
- OMM
- MOSST
- OP2A

Detailed Results
- Spago4Q
- Macxim

Login

Username: *

Password: *

Login

PROJECTS	REPORTS										
	Macxim	Bicho	Fossology	JUnit	StatSVN	Kalibro	JaBUTi	MOSST	CVSAnalY	OSLC	OM Adv Lev
APR-anonsvnRHEAD					StatSVN	Kalibro		MOSST			
Analizo											
Ant-1.8.1R961732	Macxim										
Axis-2.0R961362								MOSST			
Axis-2.0R961362RHEAD						Kalibro		MOSST			
Axis-2CRHEAD								MOSST			
Axis-2RHEAD								MOSST			
Axis2C-2010.12.8RHEAD								MOSST			
Checkstyle-5.0R2505	Macxim										
DDD-3.3.12R7209RHEAD						Kalibro		MOSST			
DDD-3.3.12RHEAD								MOSST			
Findbugs-1.3.9R11397	Macxim				StatSVN						
GCC-4.6R167452								MOSST			
Gnu GCC-						Kalibro		MOSST			

Figure 18The available quality reports

Fig 18 Shows the available quality reports in Spago4q while 19 show an example of Spago4Q report within the quality platform itself.

creative commons

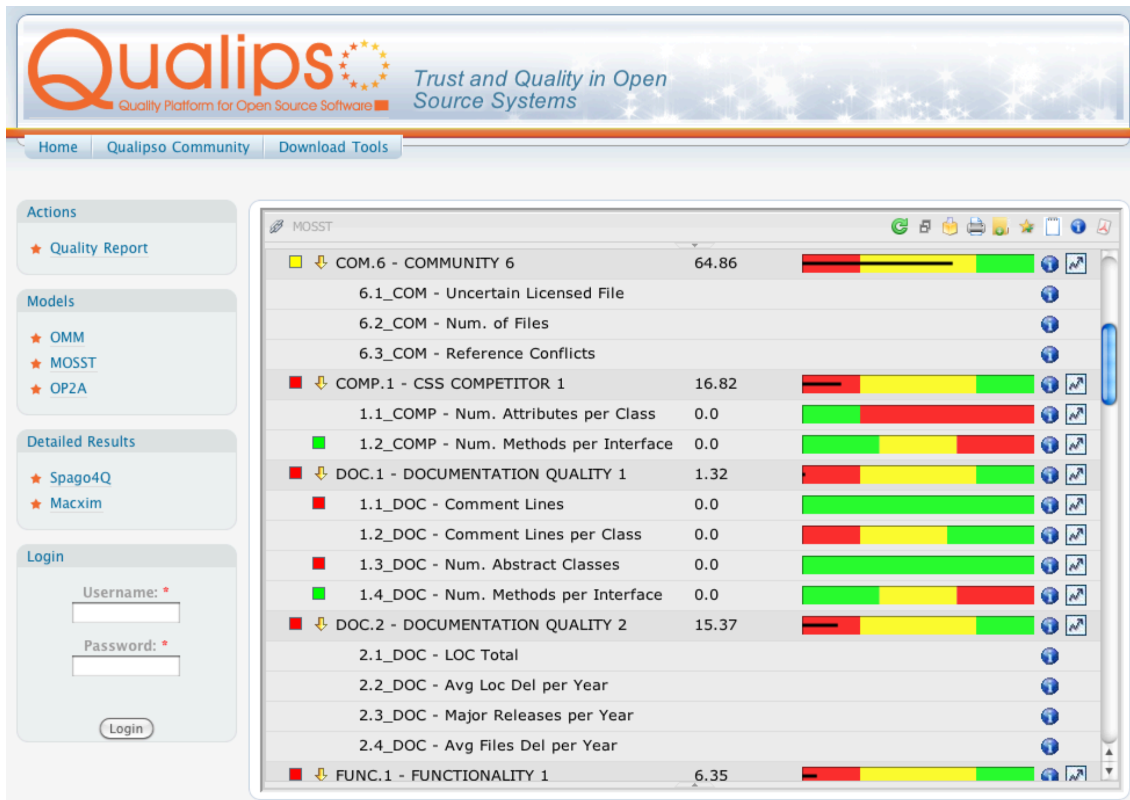


Figure 19An example of quality report

Quality platform also allows to start the analysis of new projects simply by login with a valid Spago4Q user (e.g. biadmin/biadmin).

Once a user is logged a new link "insert project" will appear on the left bar.

Fig 20 shows the form for the analysis of a new project.

Actions
Quality Report
Insert Project
Models
OMM
MOSST
OP2A
Detailed Results
Spago4Q
Macxim
Login
Logged in as:
biadmin
Logout

Insert Project
Project Information
Project Name: *
Version: *
Language: java
The programming language.
License: GNU General Public License (GPL)
From www.opensource.org.
Repository Information
Type of repository: * svn
Model Information
Model: *
MOSST
OMM
Tools Information
Tools:
Macxim OSLC JaBUTi CVSAnalY
StatSVN/CVS Bicho JUnit Kalibro
Repository URL: *
Username:
Password
Revision number: * HEAD
Start: dd/mm/yyyy
End: dd/mm/yyyy

Figure 20 Insert project form

creative commons

9.4 GQMTool

Version: 1

Location: WORKFOLDERS/A5/TOOLS/GQMTOOL.TAR.BZ2

Provider: Università dell'Insubria

License: [to be defined]

Known issues: None.

GQM Tool is focused on the homonymous methodology; the Goal/Question/Metrics paradigm GQM is an approach to using software metrics rationally. GQM defines a measurement model on three levels: a conceptual level (goal), an operational level (questions about the goal) and a quantitative level (metrics associated with each question in order to answer it). The GQM approach is used as the basis of the definition of the trustworthiness factors in WP5.3 [2][3].

GQM Tool makes it easier to define and implement a GQM plan (see Figure 21). Definition and implementation are two distinct phases of the GQM methodology. In the definition phase the GQM plan is modeled and defined: the associations between goals, quality foci, variation factors, hypotheses, impacts, questions and metrics are used. In the implementation phase the GQM plan is applied to a concrete case, and the values are collected through the metrics defined to answer one or more questions.

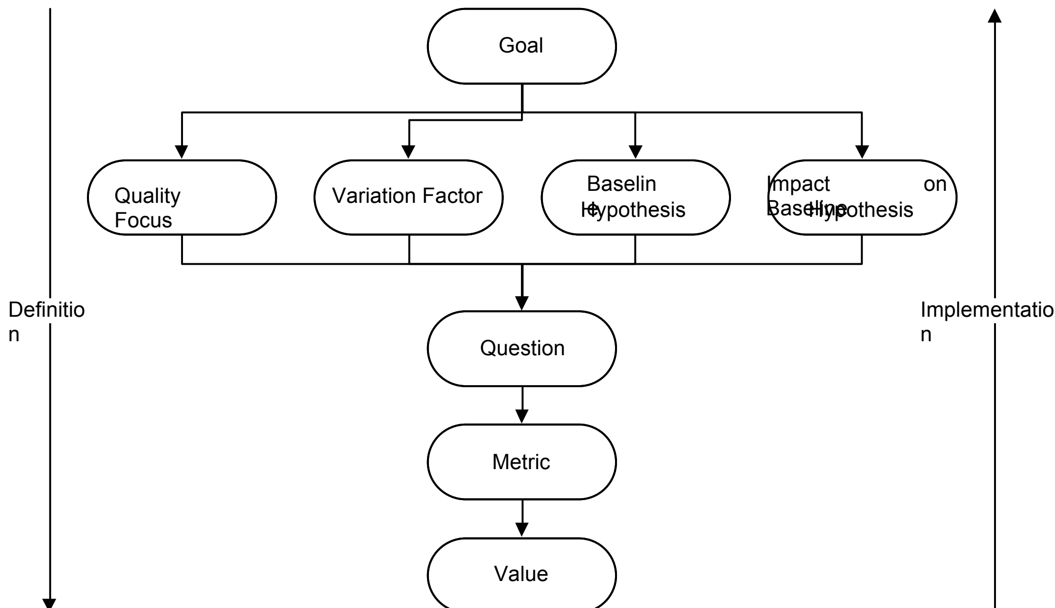


Figure 21 Definition and implementation of a GQM plan

GQM Tool saves the GQM plans in XML format: a specific XML Schema has been defined to support this. A GQM plan can be reused in multiple projects. GQM Tool connects to a RDBMS to extract the values needed in the implementation phase. The extraction is straightforward, since GQM Tool

creative commons

associates an SQL query with every metric in a GQM plan. The queries are executed during the implementation phase, on the project's repository.

creative commons

10. INTEGRATION

This chapter describes in which ways the tools in the A5 tool set may be integrated.

Figure 22 shows the A5 toolset integration approach:.

- Quality Platform and QualiPSo Factory are integrated to Spago4Q via web services that allow users to submit a new project to be analyzed and get the results of the analysis
- Measurement tools are integrated at low level through Spago4Q extractors and they communicate with Spago4Q transparently (users do not see the interactions between Spago4Q and the tools)

Figure 22: A5 toolset Integration Overview

10.1.1 Integration mechanism with Spago4Q 2.x

This section describes the mechanisms Spago4Q provides for tool integration and explains how to integrate tools step by step.

The Spago4Q modular architecture and meta-model design guarantees extensibility towards other infrastructure tools and to further sets of activity measure areas (see Appendix B for more information about Spago4Q).

In order to integrate with Spago4Q, three important terms of Spago4Q need to be explained: Extractor, Interface and ETL.

Extractor: Extractors are specialized components for collecting data from different data sources. Extractors have the following characteristics:

- collect data from infrastructure tools and load “Interfaces” components;
- can load one or more “Interfaces” components;
- can be developed with different technologies ;
- XML is adopted to exchange data with “Interfaces” components;
- more extractors mayexist to collect data from a specific tools;
- can apply rules to filter or transform data .

Interface: Interfaces are components defining the format input data for ETL (Extract Transform Load) procedures to load data warehouse. Interfaces have the following characteristics:

creative commons

- for every area of measure (i.e. requirements, bugs, test) only one interface is defined in order to standardize format input data collected from different projects or tools

ETL: ETL procedures load data into the Spago4Q data warehouse. A load procedure is developed for each “Interface”, it applies rules to filter or transform data. In Spago4Q, ETL procedures are executed by the extractor components.

10.1.2 Toolset and Spago4Q 2.x Integration process

This section explains how to integrate toolsets with Spago4Q step by step.

1. *Definition of the tools*: The tools to be integrated with Spago4Q have to be identified. Some examples of the tools are Jira, SVN, bugzilla etc.
2. *Definition of the metadata model*: The metadata model (metrics) to be assessed and monitored has to be defined. The meta data model are specific KPI (Key Process Indicator), KPA (Key Process Area) metrics that are examined or monitored as part of quality control and quality management processes of a company or open source community.
3. *Implementation of the extractor module*: For each tool the extractor module has to be implemented. The extractor collects data from a source tool like database, Jira, SVN, and arranges the information so that the data can be imported into Spago4Q data warehouse repository. In Spago4Q an extractor is implemented as a Java Interface module. The following code snippet illustrates how to create an extractor module:

```
package it.eng.extractors.jira:

public class JiraExtractor extends AbstractExtractor
implement ExtractorInterface {

    void init (ExtractionProcess process){

    …

    };

}
```

4. *Implementation of the interface module*: For each metric the interface module has to be implemented so that the measurement data of a specific tool can be integrated into Spago4Q data warehouse for ETL process. In Spago4Q every area of measure can have only one interface; hence an interface may manipulate multiple extractors. In Spago4Q an interface is implemented as a Java Interface module. The following code snippet illustrates how to create an interface module:

creative commons

```
package it.eng.extractors:
public interface ExtractorInterface {
    void init (ExtractionProcess process);

    GenericItemInterface nextItem();
    Boolean hasNext ( );
```

5. *Implementation of the ETL procedures*: The ETL procedures extract data from the Analysis tools and load them into the DWH-Spago4Q data warehouse module, based on a meta-model definition. In order to complete the assessment process, data inserted in the data warehouse have to be analyzed by the Spago4Q analytical component. Using the SpagoBI platform to implement the Spago4Q analytical components makes it easy to represent every KPIs, metrics and the related thresholds as an instance of a particular analytical document type offered by SpagoBI itself (report, OLAP, dashboard, data mining, free enquiry, geo-referenced analysis).

Spago4Q includes a configurable set of dashboards, reports and OLAP (Online Analytical Processing) documents to monitor process like: requirements management, bugs and issue tracking, test, risk management, version control. Furthermore, Spago4Q can be customized and extended by developing on top of SpagoBI platform in order to cover and satisfy the whole range of BI requirements, both in terms of analysis and data management, administration and security.

10.2 Example of tool integration in Spago4Q 2.3

All A5 tools are integrated in Spago4Q with the same process. Here, we show the integration details for one of the tools.

StatCVS and StatSVN tools are integrated with Spago4Q 2.x platform through the Spago4Q programming interface. The integration makes use of the extractor API to allow Spago4Q to handle the process of executing the integrated tools. The integration is also carried out on data level. The repository statistic tools provide data through Spago4Q *GenericItem* interface to cover metrics included in the GQM based MOSST model. The calculation of metrics is handled by Spago4Q platform mechanisms as well as building reports based on MOSST model.

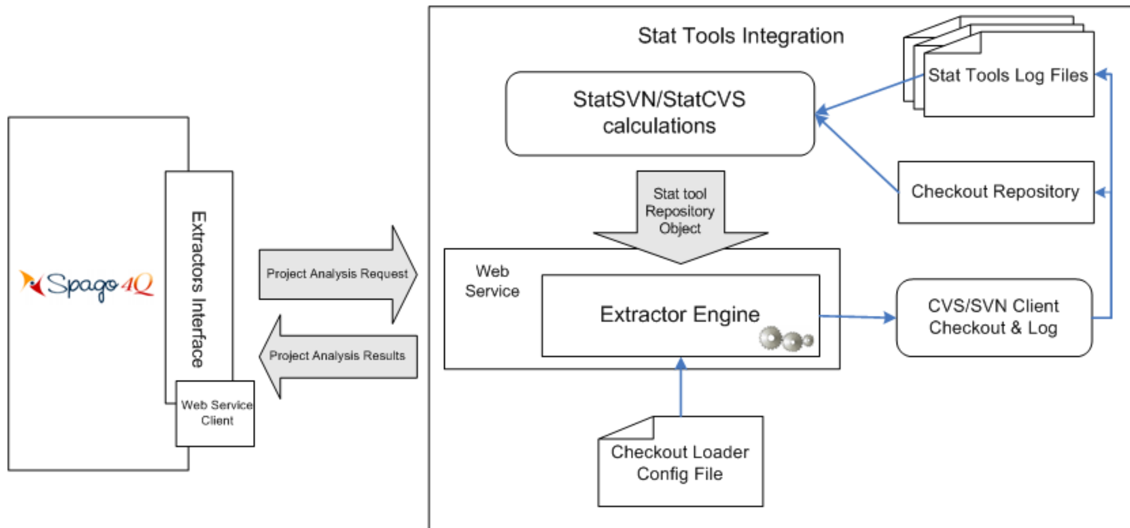
StatCVS and StatSVN integration is divided into three main components:

- Custom Spago4Q extractor,
- Stat Tools Web Service
- XRepoStats library.

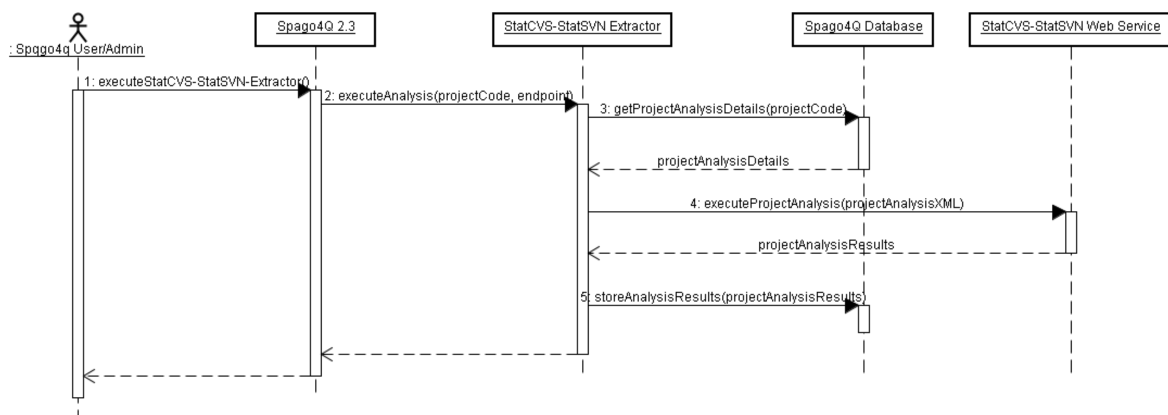
The custom Spago4Q extractor is implemented as an extension of the abstract Java class provided by the Spago4Q programming interface. It is a lightweight implementation which calls Stat Tools Web Service and transforms the results

creative commons

into Spago4Q data format i.e. *GenericItem Interface*. The actual process of execution of the integrated tools is delegated to the logic of Stat Tools Web Service. It manages all phases of StatCVS and StatSVN execution, including project checkout, log file processing and statistics calculation. The third component, XRepoStats library, is used as wrapper for managing aforementioned phases. Additionally, the original StatCVS-StatSVN implementation is instrumented with AspectJ based Aspects, to enable the capture of StatCVS/StatSVN statistics. The following diagram presents the overall architecture of StatTools Web Service.



The resulting statistics are further processed by the custom extractor (first component) to be eventually loaded into Spago4Q warehouse. Subsequently this data is calculated by Spago4Q mechanisms in order to obtain the metric values required by GQM based MOSST model. The extractor itself can be executed from Spago4Q 2.X platform by running the extractor process defined by metadata. In the figure , the sequence of extractor execution is depicted.



creative commons

The Stat Tools Web Service is configured through the xml file *loaderConfig.xml*. The workspace path parameter is used to define where the analyzed projects are to be stored. The sample file can be found below:

```
<?xml version="1.0" encoding="UTF-8"?>
<LoaderConfig>
      <LoaderFactories>
      </LoaderFactories>

<WorkspacePath>../webapps/StatToolsService/RepoHolder</Workspace
Path>
      <Loaders>
      </Loaders>
</LoaderConfig>
```

Additional configuration may be required for StatSVN tool, which establishes many connections to the SVN host server during analysis. Those connections are done in parallel, and can be seen by the host server as a Denial of Service attack. Therefore appropriate number of concurrent connection should be set for analysis. It may be done in *svnToolsConfig.xml* file:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<SVNToolsConfig>
      <LogToolConfig>
      </LogToolConfig>
      <StatToolConfig>
            <DiffThreads>
                  <Default>15</Default>
            </DiffThreads>
      </StatToolConfig>
</SVNToolsConfig>
```

The extractor itself is configurable through Spago4Q user interface and has the following parameters:

- StatTools Web Service endpoint,
- Project code: the code name of a project to be analyzed.

The remaining parameters required for the analysis are stored in Spago4Q database. Currently, these analysis parameters can be loaded using MacXim project details form and are as follows:

- name of a project to be analyzed,
- project revision,
- a type of a repository the project is stored (SVN, CVS),
- CVS/SVN repository link,
- the timeframe of the analysis(currently it needs to be set at least to default values),
- optionally the user credentials for the process of checking out the project,
- regular expressions for defining patterns of names for major and minor project releases.

creative commons

In order to get more insight into the integration of StatCVS-StatSVN with Spago4Q platform, please refer to the documentation provided along with the installation package[9].

10.3 Integration of the A5 tools into the QualiPSo Factory

The integration with QualiPSo Factory has been carried out as a proof of concept. We suggest to use the integrated platform as tool to analyze new projects. For a detailed description of QualiPSo Factory see Activity 7 reports.

The integration of the A5 tools into the factory has the goal of it making possible to assess the quality of the projects hosted by the QualiPSo factory directly from the factory portal. This integration has the same mechanism of quality platform allowing users to:

- Send the project information to Spago4Q required by the A5/A6 set of tools to perform an analysis;
- Retrieve the analysis results from Spago4Q to show them to the factory users.

Consequently, the QualiPSo factory needs to be able to communicate with Spago4Q in two ways.

The goal of this integration is to allow users to get quality reports for projects within the QualiPSo factory and request for a new analysis avoiding to insert all parameters of the quality platform.

We developed a set of components within the QualiPSo Factory:

- A User Interface component (UI)which provides a form to interact with the Factory web-service coded for the integration;
- A web-service which can send the tools' parameters to Spago4Q and retrieve the analysis assessment report from it;
- A Document web-service client containing the classes to call the Document web-service of Spago4Q.

To insert a project from the QualiPSo Factory to Spago4Q, we have to send the project information to the Spago4Q JSP page qpsinsertexecution.jsp. The different steps to reach this page are :

1) From the Factory, the user creates an Evaluation resource for the project to analyze by using the dedicated UI module developed for the integration;

[9] StatCVS-StatSVN integration package, http://abelia.man.poznan.pl/stattools/statcvs-statsvn-package-0.8.zip

creative commons

2) The user is invited to fill a form with all the parameters needed by the tools to conduct the analysis on the project;

   1. Actually, not all parameters are automatically retrieved.

3) The UI calls the web-service of the Factory developed for the integration with the project information;

4) The web-service calls the Spago4Q JSP page with the project information previously received;

5) The JSP page inserts the project information into the QPS_PROJECT_DETAIL table.

Once the project has been inserted and the analysis performed, it is possible to retrieve a PDF report of the project assessed directly from the Factory:

1) From the evaluation resource created before, the user requests the analysis report;

2) The UI calls the dedicated method of the web-service;

3) Then the web-service uses the Document WS client which calls the Documents web service located on Spago4Q;

4) The Documents web-service returns the PDF reports to Documents WS client located on the Factory. This one returns it to the web-service and create a file child resource for the evalution resource. The file Factory resource contains the PDF report;

5) From the UI, the user can browse the children resources of the evalution and display the generated report.

creative commons

11. QUALITY ASSURANCE

This chapter describes the means used in A5 to ensure the trustworthiness of the provided tools. These means intend to make the quality of the A5 tool set more transparent and support the creation of high-quality code.

Two types of tools are used in the A5 toolset:

- “Internal tools,” which are developed or adapted in the context of QualiPSo: JaBUTi, Macxim, Kalibro, Spago4Q, and GQMTool
- “External tools,” which have been developed by a different project and are used as they are: PMD, Checkstyle, OSLC, StatCVS, and StatSVN.

This chapter proposes different quality assurance techniques for both tool types. Concerning “internal tools”:

- Java coding guidelines are defined. The application of the guidelines is supported by using common code checkers, customized with a specific rule set.
- A light-weight testing approach is suggested. Testing all tools systematically is not feasible because of budget limitations.

With regard to “external tools” a “level of trustworthiness” is proposed which is intended to serve as a replacement until the trustworthiness model developed in WP5.3 and WP5.6 can be used for this purpose.

11.1 Software quality overview

Classic software quality characteristics cover functional and non-functional aspects as defined, for example, in the ISO 9126 standard [30]. Each of these characteristics is further refined by quality attributes (see Figure 23).

creative commons

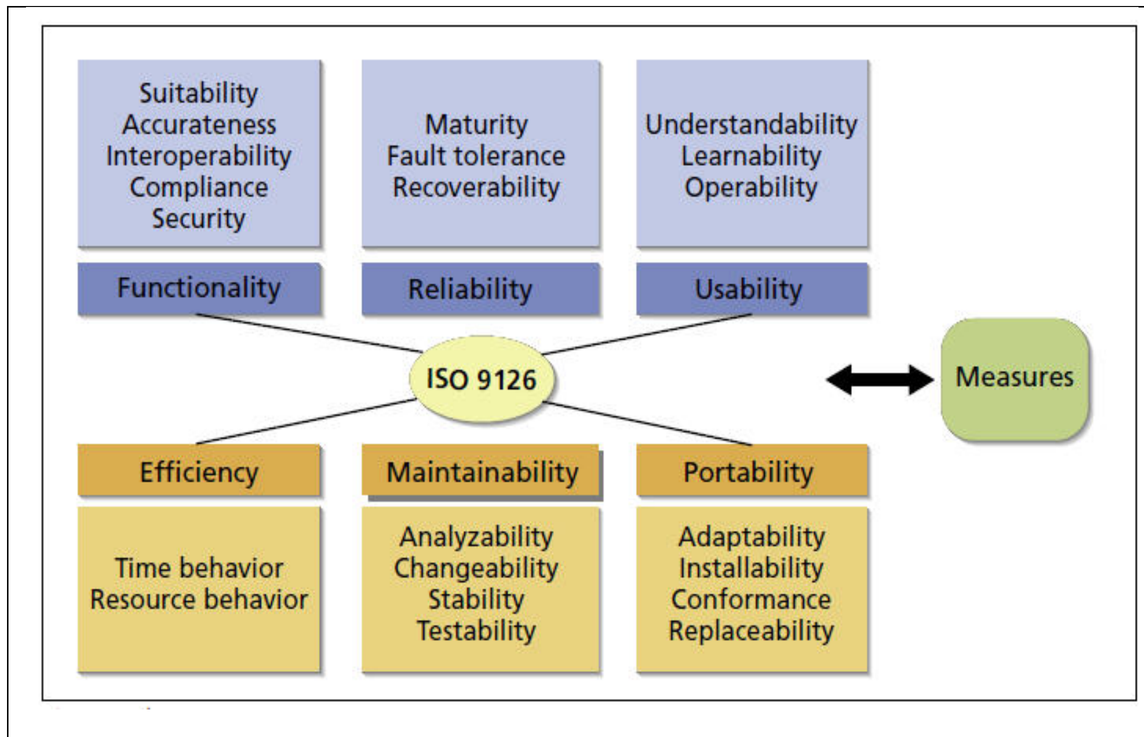


Figure 23: ISO 9126 Quality Model Overview

We use these quality attributes defined by the ISO model as the background of our quality goals (see WD 5.3.1 [2] for a detailed discussion on quality models). High quality products are based on good design and code, and they are robust, reliable, and work reasonably bug free with sufficient performance. They have error and exception handling, use a (status) logging capability, and have been thoroughly tested.

Though we know that the term quality can always be interpreted rather subjectively, we propose here a set of rules and techniques that are adapted to the A5 context. These techniques target the internal quality of the tools as well as their external quality.

As Martijn de Vrieze states in his Internet blog "QA Rocks"[10]:"External quality is that which can be seen by customers and which is traditionally tested. Bad external quality is what can be *seen*: system crashes, unexpected behavior, data corruption, slow performance. Internal quality is the hidden part of the iceberg, i.e. program structure, coding practices, maintainability, and domain expertise. Bad internal quality will result in lost development time; fixes are likely to introduce new problems and therefore require lengthy retesting. From a business point of view, this will invariably result in loss of competitiveness and reputation. *External quality is a symptom whereas the root problem is internal quality.*"

10 http://www.qa-rocks.com/index.php?option=com_content&task=view&id=14&Itemid=28

creative commons

11.1.1 Coding Conventions

Because most of the A5 tools are written in Java we restrict these considerations for now to Java coding guidelines. If similar support for other programming languages is needed, this will be handled in a later version of this document.

The first question that may arise in this context is: “Why do we need coding conventions at all”? Sun’s famous “Java Code Conventions” paper [31] gave us the answer about 12 years ago: “Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

For such conventions to work, every person writing software should confirm to the conventions; everyone. “[31]

Sun’s conventions are often cited as so called “coding guidelines”. They give the developer a template that can be used to configure a code formatter in his/her IDE. That’s because they contain rules that deal with layout and formatting issues, but their rules cover more. They address especially the quality characteristics “*Maintainability*”, “*Usability*”, “*Reliability”*”, and “*Portability*”. You will get advice for the following topics: File Names, Source File Organization, Indentation, Comments, Declarations, Statements, White Space, Naming Conventions and Programming Practices.

A good supplement to these conventions is given by Syd Chapman in his document “Supplementary of Sun Java Coding Conventions” [32]. His document draws attention on topics like appropriate documentation, naming, declarations, cleanliness, memory handling, design aspects, exception handling and logging. Most of the questions concerning these topics can be found in the Appendix D.

A high number of more or less extensive other Java coding guidelines can be found in the literature with highly overlapping and partially contradicting guidelines. We found the presented two documents to be rather typical examples and used them, extended with our own experience, as basis for the coding guidelines presented in Appendix D.

To verify if the code complies with such coding conventions, there are plenty of free and commercial tools available. They present reports about what’s wrong with your code, thus helping you to improve its quality. Examples of open source tools are FindBugs[11], QStudio for Java[12], PMD[13] (with a copy/paste

---

[11] http://findbugs.sourceforge.net/

[12] http://www.qa-systems.com/products/qstudioforjava/

[13] http://pmd.sourceforge.net/

creative commons

utility to detect duplicated code), Checkstyle[14], JDepend[15], or the code control features and analysis features provided by Eclipse and its extensions (plugins). The coding conventions proposed for the A5 tools can be verified by the help of Checkstyle and Findbugs.

Appendix D describes the A5 coding conventions in detail and how the checking can be done comfortably.

11.1.2 Testing Activities

As we saw in the beginning of this section, Software Quality Assurance can be achieved by internal and external quality assurance activities. While the preceding paragraphs focus on internal quality aspects, this section deals with external quality aspects. Methods aiming at external quality are performed during the test phases in the software development process. Chapter 4 “Testing” contains a short description of test types and test measurements. Another path to follow when considering the test topic is looking at its position in the software development process. E.g., the original software engineering process model called V-Model 97[16] (see Figure 24), which was aligned to the waterfall model for software development, already defined test phases for every step in the integration path (right side in the figure).

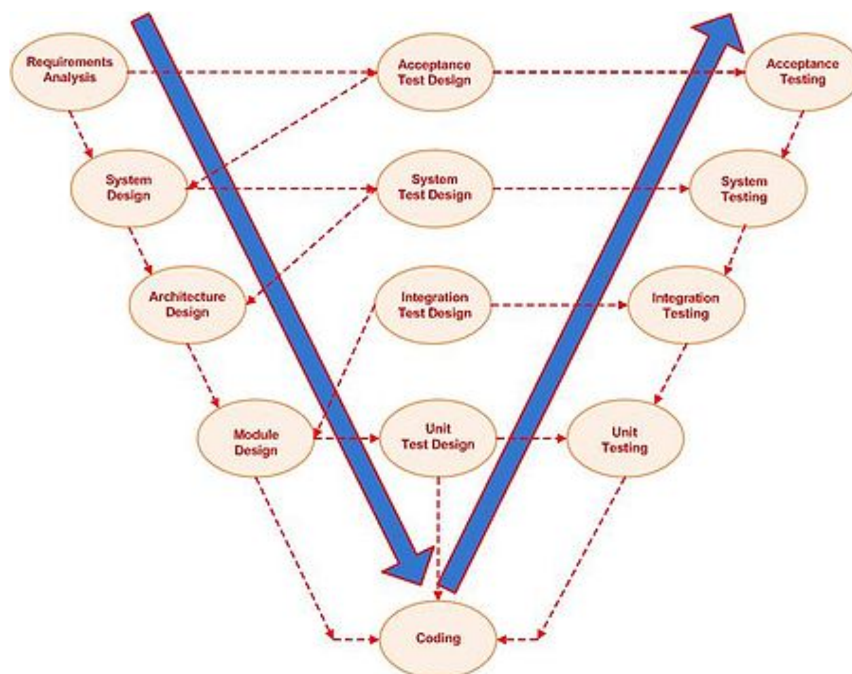


Figure 24: V-Model 97

These test phases (including functional *and* non-functional tests) are:

- Unit Testing

---

14 http://checkstyle.sourceforge.net/

15 http://clarkware.com/software/JDepend.html

16 http://v-modell.iabg.de/v-modell-xt-html-english/index.html

creative commons

- Integration Testing
- System Testing
- Acceptance Testing

The V-Model 97 is of course outdated and replaced by the V-Model XT[17] which better reflects today's usual practice in the software industry and agile software development processes. According to the changes in development methods, test methods have changed too. Regression Testing for example gets more importance along with Test Driven Development and Continuous Integration, where components are integrated frequently which allows earlier identification of failures and support of cycles in software development. But the core test methods introduced with the V-Model 97 still exist.

Therefore we have the test as a demonstration, which can be repeated any time, that a software product has the required functions and performance and that it complies with the agreed interfaces. Test tools are used to support process module, integration and system testing. They are to support both black and white box tests and have (more or less) the following characteristics:

- Test planning,
- Test design,
- Test case definition,
- Test case archiving,
- Test execution,
- Test reports,
- Test management.

This includes the execution of the following types of tests:

- Functional test,
- Interface test,
- Performance test,
- GUI tests,
- Safety and security tests,
- Regression test.For the A5 toolset we restrict the external (test) quality assurance activities to *functional unit testing* and a light-weight acceptance test. However, all unit test activities – either performed with existing tests provided by the A5 tools or with newly developed tests – should be measured for their efficiency by using a *test coverage tool* (for more information on test coverage see Appendix A).

[17] http://v-modell.iabg.de/dmdocuments/V-Modell-XT-Gesamt-Englisch-V1.3.pdf

creative commons

11.2 Coding conventions for the A5 toolset

We had a few requirements for this collection of coding guidelines for the A5 tools. We wanted them to be

- based on widely accepted best practices (like the Sun coding guidelines)
- undisputable to achieve a broad acceptance among the partners
- not too many to avoid overwhelmed developers
- useful to avoid typical bugs
- supported by code checkers to make following them as easy as possible.

This collection shall not be cast in stone but it can be adapted according to the needs of the project. The covered topics are naming conventions, coding style, documentation, declaration and definition issues, import statements, size issues, possible coding problems, error handling, design issues, and redundant code.

The collection contains rules and recommendations. Rule violations should be avoided in any case. If this is not possible, it should be commented. Rules with especially high importance are marked with an asterisk (*) to reflect this. For each code convention a reference to the corresponding Checkstyle check is given.

11.2.1 Naming conventions

Naming conventions play an important role in code readability — these rules are a subset of the Sun conventions.

Rule*: Class and interface naming
Class and interface names must start with an uppercase letter. Only alphabetic characters and digits are allowed. Valid class/interface names are, e.g., “DeviceManager” or “Layout2”. Invalid are names like “protocolHandler” or “_Objectpool. As you can see, it is recommended to use the so called “Camel Case” notation to form compound names. Wikipedia[18] even states that “CamelCase is the official convention for file names in Java”.

Tool reference: Checkstyle – TypeName

Rule*: Method naming
Method names must start with a lowercase letter. Only alphabetic characters and digits are allowed.

Tool reference: Checkstyle – MethodName

Rule*: Package naming
Package names must start with a lowercase letter. Only alphabetic characters, digits, underscores and separating dots are allowed.

Tool reference: Checkstyle – PackageName

18 http://en.wikipedia.org/wiki/CamelCase

creative commons

Rule*: Constants naming
Names of constants (static final variables) must start with an uppercase letter. Only alphabetic characters and digits are allowed.

Tool reference: Checkstyle – ConstantName

11.2.2 Coding style

Recommendation: Usage of braces
Code blocks belonging to control flow (if…else, while…do, for) have to be enclosed in braces. This helps to avoid bugs when the code is extended later.

Tool reference: Checkstyle – NeedBraces

Rule: Nesting blocks
Blocks not belonging to control structures must not be nested. Nested blocks are often leftovers from the debugging process, they confuse the reader.

Tool reference: Checkstyle – AvoidNestedBlocks

11.2.3 Documentation

Documentation is a crucial factor when it comes to understanding code. The following rules intend to encourage better quality technical documentation.

Rule: Documentation language
All documentation (inline and supplemental) must be written in English.

Rule: Class and interface documentation
Each class and interface definition must have a comment describing its purpose. This is required for any class, independent from its visibility. Author or version tags are not required so far.

Tool reference: Checkstyle – JavadocType

Rule: Member documentation
Each public variable must have a comment describing its purpose. Though the best approach would be to require documentation for all members, this mandatory rule is restricted to the public interface of the class to reduce the number of violations (pragmatic approach).

Tool reference: Checkstyle – JavadocVariable

Rule: Method and constructor documentation
Each method or constructor must have a comment describing its purpose. The comment should also describe the input parameters, the returned object and the exceptions declared to be thrown. This is required for public and protected constructors and methods. We allow missing Javadoc on accessor methods for properties (setters and getters).

Tool reference: Checkstyle – JavadocMethod (allowMissingPropertyJavadoc = true)

creative commons

Recommendation: Documentation style
Class, interface and method comments must be well formed. The first sentence of the comment must end with proper punctuation (that is a period, question mark, or exclamation mark). Javadoc automatically places the first sentence in the method summary table and index. Without proper punctuation the generated documentation may be malformed.

Javadoc statements must be followed by a description otherwise they would not be needed anyway. This means completely empty Javadoc and Javadoc with only tags such as @param and @return must be avoided. If HTML tags are used they have to be complete.

Tool reference: Checkstyle – JavadocStyle

Recommendation: Usage of TODOs
Mark open issues in the code with “TODO:” comments. A homogeneous TODO style makes it possible to easily assemble all TODOs in a code base. This list of TODO “styles” can be extended if needed by the A5 team (e.g. allow as well “@todo”).

Tool reference: Checkstyle – TodoComments

11.2.4 Declaration and definition issues

Recommendation: Modifier order
The order of the modifiers should follow the recommendation of Java Language specification [33].The correct order is:

- public | protected | private
- abstract
- static
- final
- transient
- volatile
- synchronized
- native
- strictfp[19]

Tool reference: Checkstyle – ModifierOrder

[19] “stricptfp (floating point precision): Floating point hardware calculates with more precision, and with a greater range of values than the Java specification requires. It would be confusing if some platforms gave more precision than others. When you use the strictfp modifier on a method or class, the compiler generates code that adheres strictly to the Java spec for identical results on all platforms. Without strictfp, is it is slightly laxer.

creative commons

Recommendation: Long constant definition
Long constants should be defined with an upper ell. That is ' L' and not 'l'. This is in accordance to the Java Language Specification. A small capital “l” looks a lot like 1.

Tool reference: Checkstyle – UpperEll

Recommendation: Final parameters
Method/constructor/catch block parameters should be final. Changing the value of parameters during the execution of the method's algorithm can be confusing and should be avoided. A great way to let the Java compiler prevent this coding style is to declare parameters final.

Interface and abstract methods are not concerned - the final keyword does not make sense for interface and abstract method parameters as there is no code that could modify the parameter.[20]

Tool reference: Checkstyle – FinalParameters

Rule*: Parameter assignment
Avoid assigning anything to a parameter. Parameter assignment is often considered poor programming practice because this often leads to confusion especially if the concept of pass-by-value is not completely clear. Forcing developers to declare parameters as final is often onerous. Having a check ensure that parameters are never assigned would give the best of both worlds.

Tool reference: Checkstyle – ParameterAssignment

11.2.5 Imports

The conventions around imports are beneficial for code readability. They are also supported by modern IDEs such as Eclipse and NetBeans with little extra effort.

Rule: Wildcards in import statements
Import statements must not use wildcards (*).Importing all classes from a package leads to tight coupling between packages and might lead to problems when a new version of a library introduces name clashes.

Tool reference: Checkstyle – AvoidStarImport

Rule: Import of illegal packages
Avoid import from illegal packages like sun.* packages. This is not allowed since programs that contain direct calls to the sun.* packages are not 100% Pure Java and may create problems regarding to portability.

Tool reference: Checkstyle – IllegalImport

[20] Note: If you are using Eclipse 3.2+ there is a CleanUp feature available, that can be configured to automatically change modifiers to “final” where possible (see http://www.eclipsezone.com/eclipse/forums/t86864.rhtml).

creative commons

Recommendation: Unnecessary import statements
Avoid unnecessary import statements. An import statement is considered unused if:

- It is not referenced in the file. The algorithm does not support wild-card imports like import java.io.*. Most IDE's provide very sophisticated checks for imports that handle wild-card imports.
- It is a duplicate of another import. This is when a class is imported more than once.
- The class imported is from the java.lang package. For example importing java.lang.String.
- The class imported is from the same package.

Tool reference: Checkstyle – UnusedImports

11.2.6 Size issues

Rule: File size
Avoid creating files with too much content. If a source file becomes very long it is hard to understand. Therefore long classes should usually be refactored into several individual classes that focus on a specific task. We restrict them to maximum 2000 lines.

Tool reference: Checkstyle – FileLength

Recommendation: Line length
Code lines must not have more than 120 characters. Long lines are hard to read in printouts or if developers have limited screen space for the source code, e.g. if the IDE displays additional information like project tree, class hierarchy, etc. However, modern IDEs gives us a lot of formatting capabilities as well as small font settings to cope with longer line lengths than the traditional 80 characters.

Tool reference: Checkstyle – LineLength

Rule: Method length
Avoid methods with too much content. If a method becomes very long it is hard to understand. Therefore long methods should usually be refactored into several individual methods that focus on a specific task. We restrict them to maximum 150 lines.

Tool reference: Checkstyle – MethodLength

Rule: Parameter number
Avoid methods with more than 7 parameters. Methods with lot of parameters are hard to understand. The better alternative is to use an encapsulating object and pass the object as a parameter.

Tool reference: Checkstyle – ParameterNumber

creative commons

11.2.7 Possible coding problems

Rule*: Double-checked locking

Avoid the “double-checked locking” idiom. The "double-checked locking" idiom (DCL) tries to avoid the runtime cost of synchronization. An example that uses the DCL idiom is this:

```
public class MySingleton {
    private static theInstance = null;

    private MySingleton() {}

    public MySingleton getInstance() {
        if ( theInstance == null ) {
            // synchronize only if necessary
            synchronized(MySingleton.class) {
                if (theInstance == null) {
                    theInstance = new MySingleton();
                }
            }
        }
    }
}
```

The problem with the DCL idiom in Java is that it just does not work correctly. Using it introduces bugs that are extremely hard to track down and reproduce. The "Double-Checked Locking is Broken Declaration”[21] has an in depth explanation of the exact problem which has to do with the semantics of the Java memory model. As of J2SE 5.0, this problem has been fixed. The *volatile* keyword now ensures that multiple threads handle the singleton instance correctly. This new idiom is described in [22].

Tool reference: Checkstyle – DoubleCheckedLocking

Rule*: Definition of equals()/hashCode()

Classes that override equals() must also override hashCode(). The contract of equals() and hashCode() requires that equal objects have the same hashCode. Hence, whenever you override equals() you must override hashCode() to ensure that your class can be used in collections that are hash based.

Tool reference: Checkstyle – EqualsHashCode

Rule: Instantiation of String and Boolean

Avoid unnecessary instantiations of classes String and Boolean. Creating a new java.lang.String object using the constructor wastes memory because the object so created will be functionally indistinguishable from the corresponding string constant. Java guarantees that identical string constants will be represented by

---

21 http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html

22 See Footnote 20.

creative commons

the same String object. Therefore, you should just use the string constant directly. The same is true for Boolean, i.e., you should use the factory method Boolean.valueOf() and Boolean.TRUE/Boolean.FALSE constants instead of using new Boolean(true)/new Boolean(false).

Tool reference: Checkstyle – IllegalInstantiation

Rule: Assignment in expressions

Do not use the assignment operator "=" in expressions. With the exception of iterators, all assignments should occur in their own top-level statement to increase readability. With inner assignments as the one below is difficult to see all places where a variable is set.

```
String s = Integer.toString(i = 2);
```

Tool reference: Checkstyle – InnerAssignment

Rule*: Default in switch statements

Switch statements must contain a "default" clause. Even if the developer is sure that all currently possible cases are covered, this should be expressed in the default branch, e.g. by using an assertion. This way the code is protected against later changes, e.g. introduction of new types in an enumeration type.

Tool reference: Checkstyle – MissingSwitchDefault

Rule*: Leaving case statements

Avoid switch statements where a case contains Java code but lacks a break, return, throw or continue statement since omitting this usually is a bug. Special comments to mark intended violations are allowed.

Tool reference: Checkstyle – FallThrough[23]

Rule*: Dubious String comparison

Avoid comparing Strings using "==" or "!=" operators. The comparison of String objects for equality using the == or != operators does usually not what the developer intended to do. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by two different String objects. Consider using the equals(Object) method instead.

Tool reference: Checkstyle – StringLiteralEquality,

Rule*: Dubious floating point comparison

Avoid comparing floating point values for equality. Because floating point calculations may involve rounding, calculated float and double values may not be accurate. For values that must be precise, such as monetary values, consider using a fixed-precision type such as BigDecimal. For values that need

---

[23] Special comment for Checkstyle: By default the text "fallthru", "fall through", "fallthrough", "falls through" and "fallsthrough" are recognized (case sensitive). We have added "FALLTHROUGH" too. The comment containing these words must be a one-liner and must be on the last none-empty line before the case triggering the warning or on the same line before the case (ugly, but possible).

creative commons

not be precise, consider comparing for equality within some range, for example: if (Math.abs(x - y) < .0000001) (see the Java Language Specification, Appendix D).

Rule*: Potential null pointer dereference
Avoid null pointer dereference. This will lead to a NullPointerException when the code is executed.

Tool reference: FindBugs – NP_ALWAYS_NULL

11.2.8 Error handling

Rule: Redundant Exception declarations
Avoid redundant exceptions declared in throws clause such as duplicates, unchecked exceptions or subclasses of another declared exception. Redundant throws may confuse a reader, especially if their presence leads to inadequate javadoc information.

Tool reference: Checkstyle – RedundantThrows

Rule: Catching Throwable
A catch statement must not catch Throwable. This would have the unwanted side effect that as well Errors would be caught.

Tool reference: Checkstyle – IllegalCatch

Rule*: Dealing with Exceptions
Catch blocks must not be empty. If an exception has been caught, the code must either do something sensible to deal with the detected problem or at least create a logging message.

Tool reference: Checkstyle – EmptyBlock (using LITERAL_CATCH)

11.2.9 Design issues

Rule: Final classes
Classes which have only private constructors must be declared as final.

You cannot extend a class with only private constructors defined - you'll get a compiler error if you try to. The reason for this is that, even if you don't declare a constructor in the subclass, a default one is created. As every constructor begins with an invocation of the parent class' constructor (either explicitly or implicitly), you're bound to get an access error because the constructor you're trying to reach is marked as private in the parent class.

That means that a class with only private constructors defined *isimplicitly* final. The error message you get from trying to extend a class with only private constructors isn't the same as the message you'd get from trying to extend a final class, but it's an error, nonetheless. A class that is final, however, is not required to have only private constructors defined. It is perfectly legal to have constructors with any access modifiers within a final class.

creative commons

Tool reference: Checkstyle – FinalClass

Recommendation: Interfaces define types

Use Interfaces only to define types. As J. Bloch states in "Effective Java", Item 19 [34], and an interface should describe a type. It is therefore inappropriate to define an interface that does not contain any methods but only constants. Having Java 5 with static imports on board there is even more no need to misapply interfaces to define constants to avoid writing an additional class name in front of the constant name.

Tool reference: Checkstyle – InterfaceIsType

Recommendation: Constructors in utility classes

Make sure that utility classes do not have a public constructor. Utility classes are classes that contain only static methods or fields in their API. Instantiating utility classes do not make sense. Hence the constructors should either be private or (if you want to allow subclassing) protected. A common mistake is forgetting to hide the default constructor.

If you make the constructor protected you may want to consider the following constructor implementation technique to disallow instantiating subclasses:

```
// not final to allow subclassing
public class StringUtils {
protected StringUtils() {
throw new UnsupportedOperationException();
        // prevents calls from subclasses
    }

    public static int count(char c, String s) {
        // ...
    }
}
```

Tool reference: Checkstyle – HideUtilityClassConstructor

Rule: Restrict the visibility of class members.

Only static final members may be public, but remember that public constants must contain either primitive values or references to immutable objects (see footnote 24). Other class members must be private or protected. The general rules are: "Minimize the accessibility of classes and members" (Effective Java, Item 13) and "Make Each Class or Member as inaccessible as possible".[24]

Tool reference: Checkstyle – VisibilityModifier

Rule*: Hiding class members

A member must not be hidden by a member of the same name defined in a sub class, a local variable or parameter.

[24] See cs.gmu.edu/~pammann/619/ppt/Chapter4.ppt

creative commons

A class must not define a field with the same name as a visible instance field in a superclass. This is confusing, and may lead to an error if methods update or access one of the fields when they wanted the other.

In a similar way, a local variable or a parameter must not shadow a field that is defined in the same class. In most cases, this is a sensible rule, since it can cause considerable ambiguity in the code. However, there are situations where it is not appropriate. IDEs like Eclipse and NetBeans provide convenient functionalities for generating getters and setters, such as the one shown here:

```
public void setName(String name) {
    this.name = name;
}
```

Although this code is fine, some tools like e.g. Checkstyle would indicate an error here. To get around this we will exclude setter and constructor parameters from this rule.

Tool reference: FindBugs – MF_CLASS_MASKS_FIELD, Checkstyle – HiddenField

11.2.10 Redundant Code

Recommendation: Unused members
Avoid unused members. This recommendation applies only to private members.

Tool reference: FindBugs – UUF_UNUSED_FIELD

Recommendation: Unused methods
Avoid methods that are never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed. This recommendation applies only to private methods.

Tool reference: FindBugs – UPM_UNCALLED_PRIVATE_METHOD

Recommendation: Duplicate code
Avoid duplicate code. Similar or identical functionality is often used at different locations of the source code. Duplicated code corrupts the design, increases the size of the code unnecessarily and creates consistency problems when it comes to bug fixing and maintenance. To avoid a huge number of findings the code checker searches for code chunks with more than about 15 lines.

Tool reference: CPD (minimumTokenCount=100), Checkstyle – StrictDuplicateCode, Simian

11.3 Applying the A5 coding conventions

Coding conventions can be most useful if they are applied as early and as consistently as possible without restricting the creativity of the developer or adding an excessive overhead to development efforts. Rules that do not fit the project environment should be removed. To support this, we establish the following approach:

creative commons

1. The partners agree on a set of coding guidelines and aim at delivering compliant code as input for A5. The rules of the set can be adapted if necessary in coordination among the partners.
2. The code is checked regularly by the developing partners for its compliance to the rules, e.g. by the help of the tools described later in this section. Of course, other tools can be used as well. It is recommended to keep the questions of Chapman's checklist in mind (see appendix D).
3. In some cases, it may be necessary to break the rules, for instance based on restrictions from external components. If this is the case, the deliberate violation is justified and must be documented in the code (it is possible to disable most code checkers, e.g., Checkstyle, by using inline comments).
4. Each release of a tool is supplemented by a report on its compliance to the coding guidelines.

To ease the application of the A5 coding conventions, Checkstyle has been chosen to support the A5 internal quality assurance activities for static code analysis ("internal quality"). Both tools are Java based open source projects that are under active development. Both tools support developers by providing configuration possibilities, Ant and Maven integration as well as Eclipse plugins. The following sections describe how this can be done.

11.3.1 Using Checkstyle

Checkstyle, which is one of the A5 tools, it analyzes the source code providing an *Abstract Syntax Tree (AST)*. This AST can be used by custom add-ons that allow the user community to extend Checkstyle. Siemens, e.g., has written a custom code checker (module) that analyzes Log4J/SL4J best practices and looks for potential errors and problems in this area. Checkstyle itself is quality-checked by Checkstyle ("bootstrapping") and comes with a good JUnit 3.8.x based test set and test framework. For our analysis we propose to use Checkstyle version 4.4.

Running the tool

Checkstyle can be used as a command line tool or with Ant or Maven. Plugins for various IDEs are available. The utilities provided in A5_QA_utilities.zip help to set up an environment for checking the coding guidelines described before .

Rule set

The complete Checkstyle checker-module list can be found at http://checkstyle.sourceforge.net/availablechecks.html. Each violation of an audit check is assigned one of the severity levels 'ERROR', 'WARNING', 'INFO' or 'IGNORE'. We recommend using the provided configuration file which is tailored according to the A5 coding guidelines.

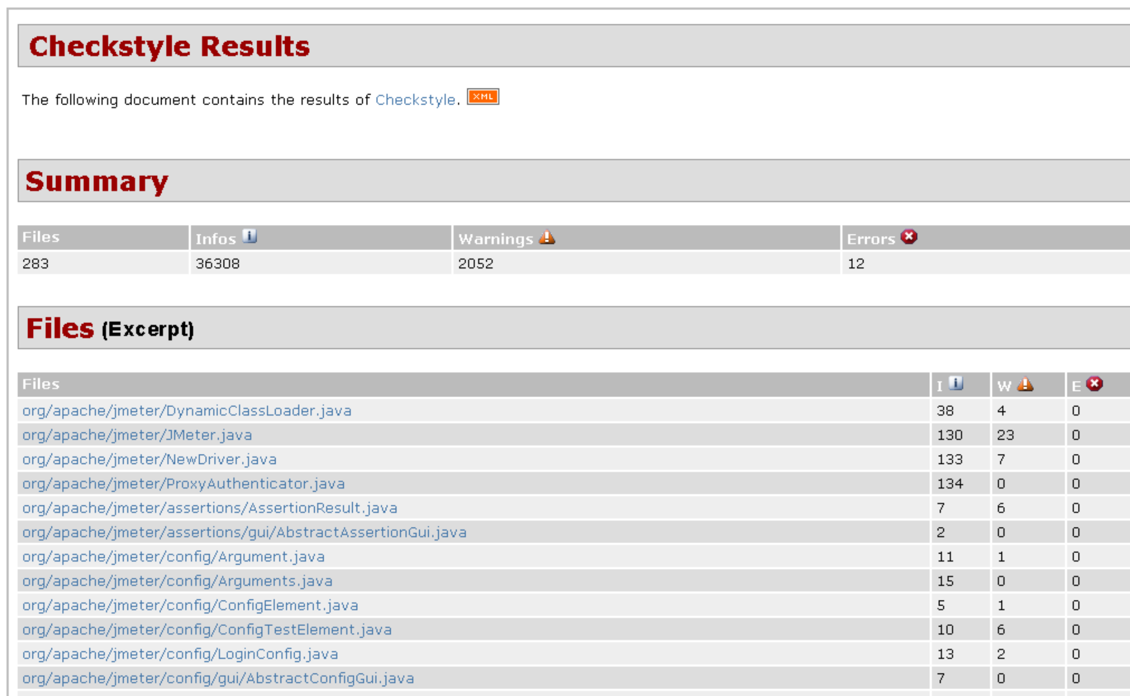
Violations to be ignored

creative commons

Violations can be ignored by either providing a specific suppressions XML document as a filter or by using specific inline comments in your source code.

Reporting

Appendix B contains detailed information on how to create an easy-to-use HTML analysis report. The ANT version of this report is a little bit more “pure” but with Maven2 we can generate a really expressive document containing even cross references to the source code affected. The following screenshots show a sample Checkstyle HTML report generated with Maven2, applied to the open source project JMeter[25].



Checkstyle Results

The following document contains the results of Checkstyle. XML

Summary

Files	Infos	Warnings	Errors
283	36308	2052	12

Files (Excerpt)

Files	I	W	E
org/apache/jmeter/DynamicClassLoader.java	38	4	0
org/apache/jmeter/JMeter.java	130	23	0
org/apache/jmeter/NewDriver.java	133	7	0
org/apache/jmeter/ProxyAuthenticator.java	134	0	0
org/apache/jmeter/assertions/AssertionResult.java	7	6	0
org/apache/jmeter/assertions/gui/AbstractAssertionGui.java	2	0	0
org/apache/jmeter/config/Argument.java	11	1	0
org/apache/jmeter/config/Arguments.java	15	0	0
org/apache/jmeter/config/ConfigElement.java	5	1	0
org/apache/jmeter/config/ConfigTestElement.java	10	6	0
org/apache/jmeter/config/LoginConfig.java	13	2	0
org/apache/jmeter/config/gui/AbstractConfigGui.java	7	0	0

Figure 25: Overview on Violations per File

25 JMeter 2.3.2, see http://jakarta.apache.org/site/downloads/downloads_jmeter.cgi

creative commons

Rules (Excerpt)

<table>
<tr><th>Rules</th><th>Violations</th><th>Severity</th></tr>
<tr><td>JavadocType</td><td>70</td><td>Warning</td></tr>
<tr><td>JavadocVariable
• scope: "public"</td><td>168</td><td>Warning</td></tr>
<tr><td>JavadocMethod
• allowMissingPropertyJavadoc: "true"
• scope: "protected"
• allowUndeclaredRTE: "true"</td><td>1340</td><td>Warning</td></tr>
<tr><td>JavadocStyle</td><td>226</td><td>Warning</td></tr>
<tr><td>ConstantName</td><td>206</td><td>Info</td></tr>
<tr><td>MethodName</td><td>19</td><td>Info</td></tr>
<tr><td>PackageName</td><td>0</td><td>Info</td></tr>
<tr><td>TypeName</td><td>0</td><td>Info</td></tr>
<tr><td>AvoidStarImport</td><td>0</td><td>Warning</td></tr>
<tr><td>IllegalImport</td><td>0</td><td>Warning</td></tr>
<tr><td>RedundantImport</td><td>3</td><td>Warning</td></tr>
<tr><td>UnusedImports</td><td>1</td><td>Warning</td></tr>
<tr><td>FileLength
• max: "2000"</td><td>0</td><td>Info</td></tr>
<tr><td>LineLength
• max: "122"
• ignorePattern: "^\s**.*@(see|throws|link|return) .*$|^import"</td><td>161</td><td>Warning</td></tr>
<tr><td>MethodLength
• max: "150"</td><td>2</td><td>Info</td></tr>
</table>

Figure 26: Overview on violations per rule

Details (Excerpt)

org/apache/jmeter/DynamicClassLoader.java

Violation	Message	Line
Warning	Missing a Javadoc comment.	37
Info	Parameter urls should be final.	37
Warning	Missing a Javadoc comment.	41
Info	Parameter urls should be final.	41
Info	Parameter parent should be final.	41
Warning	Missing a Javadoc comment.	45
Info	Parameter urls should be final.	45
Info	Parameter parent should be final.	45
Info	Parameter factory should be final.	46
Warning	Missing a Javadoc comment.	51
Info	Parameter url should be final.	51
Info	Parameter urls should be final.	59
Info	Found duplicate of 17 lines in D:\Projects\Checkstyle-4.4\JMeter\src\main\java\org\apache\jmeter\assertions\Assertion.java, starting from line 1	1
Info	Found duplicate of 17 lines in D:\Projects\Checkstyle-4.4 \JMeter\src\main\java\org\apache\jmeter\assertions\AssertionResult.java, starting from line 1	1
Info	Found duplicate of 17 lines in D:\Projects\Checkstyle-4.4 \JMeter\src\main\java\org\apache\jmeter\assertions\gui\AbstractAssertionGui.java, starting from line 1	1

Figure 27: Detailed description of violations per file

creative commons

11.4 Testing of A5 tools

For Java based tools we can use the well known JUnit[26] test framework or the test framework TestNG[27] which has more features. Both frameworks allow for test automation with ANT or Maven2, so we are prepared for doing regression testing. Depending on the kind of unit tests provided by the A5 tools (single tests, suites) we will execute these tests in a batch scenario (e.g., using ANT) and measure the effectiveness of these tests (test code coverage) using the open source tool EMMA. The output is a HTML test result report and a HTML test coverage report.

The test scenario for the A5 toolset comprises the following steps:

1. Treat the tool as a standalone application and check its installation procedure: Can the tool easily be installed, e.g., using an existing installation guide? Has the tool an operating system dependency? These questions should be answered by at least one QualiPSo partner different from the developer itself.
2. Are the functional requirements fulfilled? Is the tool under test able to deliver the metric values needed in the context of a trustworthiness evaluation as defined in WP5.3?
3. Which kind of tests does the tool itself already provide? Because the focus in this document version is on Java tools we will look especially for existing JUnit or TestNG unit tests that can be executed automatically using Ant. The result of this step is an overview on the number of available test cases and their location.
4. Try to run existing tests (test suites) and evaluate the (HTML) test report. Check if test reports are available for comparison.
5. Check if there is a test coverage report available that gives us information about the quality of the existing tests. If no report is available, we try to generate a test coverage report using the open source tool EMMA and Ant.
6. The results of the executed tests as well as the result of the test coverage report will give us an overall impression about the quality of the A5 tool with respect to “testing”. For instance, EMMA supported coverage types include: class, method, line, basic block. EMMA can even detect when a single source code line is covered only partially. However, because there are no “overall numbers” for good test coverage available it is up to the A5 team to define the thresholds.

   In general, test coverage data helps developers identify missing test cases. It also helps us to get a clearer picture of the functional quality of the code. However, the test code coverage metric can also create an illusion. The problem is that this number is always related to what’s in the

---

[26] http://www.junit.org/

[27] http://en.wikipedia.org/wiki/TestNG

creative commons

code. *It can never tell us what is missing from the code.* Unfortunately, many bugs are the result of unwritten code. So the coverage topic is sometimes problematic to interpret.

11.5 Quality of the “external” tools

The best way to evaluate the trustworthiness of the “external” tools would be to use the trustworthiness model created in WP5.3. Unfortunately this is not feasible for several reasons. Most important: the trustworthiness model is not yet in a status that allows a fast and cost-effective evaluation of all tools to consider. Nevertheless one of the tools, PMD, will be object of the first set of experiments in WP5.6. Using the same quality assurance mechanisms as for the tools developed in A5 is not feasible for budget reasons as well.

As a pragmatic solution we propose to use a “level of trustworthiness” (until the trustworthiness model is really usable for this purpose) which is not based on an objectively evaluated, well-defined list of criteria, but on a *subjective* rating of a very limited set of questions. These questions are inspired by the high-priority factors identified in WP5.1 and the questions of the questionnaire used in WP5.6 [38]:

- How much does the tool satisfy our functional requirements??
- How reliable is the product?
- How fast is the product?
- Does the code look maintainable?[28]
- How well documented is the product?
- Is the project community lively?
- Does the license of the tool allow using it in the A5 context?

The answers should be a ranking on a 1 to 6 scale where 1=absolutely not; 2=little, 3=just enough; 4=more than enough; 5= very/a lot; 6=completely.

JUnit - Overall rating	5
How much does the tool satisfy our functional requirements?	3
How reliable is the product?	5
How fast is the product?	6
Does the code look maintainable?	5
How well documented is the product?	4
Is the project community lively?	4
Does the license of the tool allow using it in the A5 context?	6

[28] This includes quality characteristics as modularity, analyzability, craftsmanship, structuredness, and standard compliance.

creative commons

PMD - Overall rating	6
How much does the tool satisfy our functional requirements?	6
How reliable is the product?	5
How fast is the product?	6
Does the code look maintainable?	5
How well documented is the product?	6
Is the project community lively?	6
Does the license of the tool allow using it in the A5 context?	6

A short description of PMD can be found in Macxim Section

Checkstyle - Overall rating	6
How much does the tool satisfy our functional requirements?	6
How reliable is the product?	5
How fast is the product?	6
Does the code look maintainable?	5
How well documented is the product?	6
Is the project community lively?	6
Does the license of the tool allow using it in the A5 context?	6

A short description of Checkstyle can be found in Macxim section

OSLC - Overall rating	5
How much does the tool satisfy our functional requirements?	5
How reliable is the product?	5
How fast is the product?	5
Does the code look maintainable?	4
How well documented is the product?	4
Is the project community lively?	3
Does the license of the tool allow using it in the A5 context?	6

A short description of OSLC can be found in OSLC section.

StatCVS - Overall rating	4
How much does the tool satisfy our functional requirements?	5
How reliable is the product?	5

creative commons

How fast is the product?	3
Does the code look maintainable?	3
How well documented is the product?	4
Is the project community lively?	3
Does the license of the tool allow using it in the A5 context?	6

A short description of StatCVS can be found in section 8.1.

StatSVN - Overall rating	3
How much does the tool satisfy our functional requirements?	5
How reliable is the product?	3
How fast is the product?	2
Does the code look maintainable?	3
How well documented is the product?	4
Is the project community lively?	3
Does the license of the tool allow using it in the A5 context?	6

A short description of StatSVN can be found in section 8.1.

creative commons

12. CONCLUSION AND FUTURE WORK

The A5 tool set includes tools from external providers as well as tools developed by one of the partners. The choice of the testing tools is inspired by the concepts investigated in Task 5.4.2 and the choice of the measurement tools is based on the trustworthiness factors identified in WD 5.3.2 [3].

- JaBUTi provides information about testing
- PMD, Checkstyle, MacXim AND Kalibro are code analysis tools that deliver information related to maintainability and reliability
- StatCVS and StatSVN provide data on the development process of the OSS product
- GQMTool supports the definition of a Goal-Question-Metric model
- The platform Spago4Q is used to collect the data coming from these sources and present it according to the GQM defined in WP5.3.
- Quality platform provides an easy way to access to the quality report and to analyze new projects.

All tools have been fully integrated into Spago4Q while OSLC and JUnit were partially integrated.

Spago4Q has been partially integrated into Qualipso Factory as proof of concept.

To ensure the quality of the A5 tools, a quality concept has been introduced and used.

creative commons

13. References

[1] Vieri del Bianco et al, "How European software industry perceives OSS trustworthiness and what are the specific criteria to establish trust in OSS", Deliverable A5.D1.5.1, QualiPSo project, 2007

[2] Luigi Lavazza et al, "The observed characteristics and relevant factors used for assessing the trustworthiness of OSS products and artefacts", Deliverable D5.3.1, QualiPSo project, 2008

[3] Vieri del Bianco et al, "Identification of factors that influence the trustworthiness of software products and artefacts", Working document wd5.3.2, QualiPSo project, 2008

[4] ISO/IEC 14598-1, "Information technology - Software product evaluation - Part 1: General overview", 1999

[5] http://www.aptest.com

[6] Wei-Tek Tsai, "Test Coverage", Department of Computer Science and Engineering, Arizona State University, http://asusrl.eas.asu.edu/cse565/content/coverage/coverage.pdf

[7] Andrew Glover, "In pursuit of code quality: Don't be fooled by the coverage report", January 2006, http://www-128.ibm.com/developerworks/java/library/j-cq01316/?ca=dnw-704

[8] Wikipedia on software performance testing: http://en.wikipedia.org/wiki/Software_performance_testing

[9] EDOS (Environment for the Development and Distribution of Open Source Software) project, "Deliverable WP3-D3.2.1: Quality Assurance Portal Prototype", http://www.edos-project.org/xwiki/bin/download/Main/Deliverables/edos/wp3d2.1.pdf

[10] Grig Gheorghiu: Internet blog "Performance vs. load vs. stress testing": http://agiletesting.blogspot.com/2005/02/performance-vs-load-vs-stresstesting.html

[11] Institute of Electrical and Electronics Engineers (IEEE). IEEE Std 829: *Standard for Software Test Documentation*. New York, NY: IEEE, 1998

[12] Software testing basic definitions: http://www.vocw.edu.vn/content/m10074/latest/

[13] ETSI testing specifications at http://portal.etsi.org/mbs/Testing/testing.htm

[14] Software QA and Testing Frequently-Asked-Questions: http://www.softwareqatest.com/qatfaq2.html#FAQ2_7

[15] Harry M. Sneed, Stefan Jungmayr: Produkt- und Prozessmetriken für den Softwaretest, Informatik-Spektrum, 29_1_2006 (only available in German). See also: http://www.gm.fh-koeln.de/~winter/tav/html/tav20/P3Sneed1TAV20.pdf

[16] Applicability of modified condition/decision coverage to softwaretesting
Chilenski, J.J.; Miller, S.P. Software Engineering Journal, Volume 9, Issue 5, Sep 1994 Page(s):193 – 200

[17] Software considerations in airborne systems and equipment certification, RTCA/DO-178B, 1992

creative commons

[18] Guide to software quality assurance, in ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS0511.pdf

[19] Arnaud Laprévote et al, “Test suites and benchmarks for the chosen set of Open Source projects and artifacts. Methodology for creating test suites and benchmarks for arbitrary systems”, Working document wd 5.4.2, QualipSo project, 2009

[20] A.M.R. Vincenzi, M.E. Delamaro, J.C. Maldonado, W.E. Wong, “Establishing structural testing criteria for Java byte code”, Software, Practice & Experience, v. 36, p. 1513-1541, 2006

[21] André Takeshi Endo, Marcelo Medeiros Eler, Rodrigo Pinto Gondim, Paulo Cesar Masiero, Marcio Eduardo Delamaro, “Documentation of the JaBUTi Service, version 1.0”, 2008

[22] A.M.R. Vincenzi, J.C.Maldonado, W.E. Wong, M.E. Delamaro, “Coverage Testing of Java Programs and Components”, Journal of Science of Computer Programming, Elsevier,v. 56, 211-230, 2005

[23] J.C. Maldonado, “Potential-Uses Criteria: A Contribution to the Structural Testing of Software”, DCA/FEE/UNICAMP, 1991

[24] S. Rapps, E.J. Weyuker, “Selecting Software Test Data Using Data Flow Information”, IEEE Transactions on Software Engineering, v.11, 367-375, 1985

[25] M. Roper, “Software Testing”, McGrall Hill, 1994

[26] Etiel Petrinja, Alberto Sillitti, Sergio Oltolina, Gabriele Ruffati, Felipe Ortega, Renata Fortes, “Specification of the tools to support CMM-like model for OSS, v1”, Working document 6.4.1, QualiPSo project, 2008

[27] Panagiotis Rentzepopoulos et al, “Technical pre-requisites from the factory to implement all the QualiPSo results – first version”, Deliverable WD7.3.2.1, QualiPSo project, 2008

[28] George Milis et al, “Constraints (functional and technical) from all the QualiPSo Work Packages – first revision”, Working document wd 7.3.1.1, QualiPSo project, 2008

[29] Abbas Tahir et al, “Common approaches and techniques for testing Open Source Software.”, Working document wd 5.4.1, QualiPSo project, 2008

[30] ISO/IEC 9126-1:2001 “Software Engineering—Product Quality—Part 1: Quality model”, June 2001

[31] Sun Microsystems, “Java Code Conventions”, 1997, http://java.sun.com/docs/codeconv/

[32] Syd Chapman, “Supplementary of Sun Java Coding Conventions”, 2004, http://www.omii.ac.uk/dissemination/JavaCodingStandards.pdf

[33] James Gosling et al, “Java Language Specification”, Second Edition, 1996, http://www.omii.ac.uk/dissemination/JavaCodingStandards.pdf

[34] Joshua Bloch, “Effective Java”, Second Edition, Prentice Hall, 2008, http://www.omii.ac.uk/dissemination/JavaCodingStandards.pdf

[35] http://findbugs.sourceforge.net/factSheet.html

[36] PostgreSQL user manual; http://www.postgresql.org/docs/8.4/static/index.html

[37] Frank Simon et al, “Code-Quality-Management”, dpunkt.verlag, 2006 (only available in German)

creative commons

[38] Luigi Lavazza et al, “Experimentation on the trustworthiness of Open Source Software”, Working Document WD5.6.1, QualiPSo project, 2009

creative commons

APPENDIX A– OVERVIEW ON TESTING

Though wd5.4.1 [29] already gives “an overview on the state-of-the-art of software testing techniques”, we resume some of the aspects here to make the document self-contained and get into more detail on aspects which are of greater importance in the context of this document (e.g. coverage measures).

Software that has not been tested is very unlikely to work. That is true for OSS as well as for commercial software. Therefore in QualiPSo we have to convince the potential OSS users that testing has been done properly. To do this, the QualiPSo toolset should check that testing activities have been performed (or can be performed) in two areas:

1. Inside the OSS itself with the OSS’s own test set (referred to as OSS "internal" tests)
2. With a suite of tests developed in the QualiPSo environment (referred to as "external" tests).

Furthermore we should check if the tests

- comply with the testing requirements[29] (if there are any requirements at all)
- are properly documented
- are appropriate for the degree of criticality of the software.

It is the job of the testing activity to uncover bugs and ensure the system under test (SUT), i.e., the OSS, performs as expected. By doing so testing should increase the trustworthiness of the SUT.

We can define testing as all or some of the following activities: [5], [11]:

Testing is:

- the process of exercising software to verify that it satisfies specified requirements and to detect errors
- the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs), and to evaluate the features of the software item
- the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.

On this note, testing contributes to analyzing OSS quality.

[29] It is generally accepted that requirements are the foundation upon which an entire system is built. Testing requirements can be derived from these general system requirements by means of refinement techniques (=>requirements tracking) or the can be created by developers and/or testers as a first step in the test process. They describe exactly what should be tested (which functional/non-functional aspects of a system/component) and with what kind of test data. Ambiguous or otherwise poorly worded requirements could be subject to testing problems later on.

creative commons

In general we can distinguish between *functional* and *non-functional* testing.

Functional Testing

Functional testing consists of unit, integration, system testing (all performed by the software producer).

According to [12] these terms are defined as follows:

- *Unit Testing* verifies the functioning in isolation of software pieces which are separately testable. Depending on the context, these could be the individual subprograms or a larger component made of tightly related units. A test unit is defined more precisely in the IEEE Standard for Software Unit Testing (IEEE1008-87), which also describes an integrated approach to systematic and documented unit testing. Typically, unit testing occurs with access to the code being tested and with the support of debugging tools, and might involve the programmers who wrote the code.
- *Integration testing* is the process of verifying the interaction between software components. Classical integration testing strategies, such as top-down or bottom-up, are used with traditional, hierarchically structured software. Integration testing is a continuous activity, at each stage of which software engineers must abstract away lower-level perspectives and concentrate on the perspectives of the level they are integrating.
- *System testing* is concerned with the behaviour of a whole system. The majority of functional failures should already have been identified during unit and integration testing. External interfaces to other applications, utilities, hardware devices, or the operating environment are also evaluated at this level.

The importance of test coverage in functional testing

First let us look at the functional testing:

A number of OSS comes with a set of unit, integration and/or system tests. In general it is not the task of the QualiPSo toolset to run these internal OSS tests again though potential users might run the existing tests to verify if the promises of the tool are kept. In any case it is important to identify missing test or code coverage data. Especially the coverage data is important because it relates the tests performed to the tested source code and therefore helps to figure out which parts of the OSS are covered by tests.

According to Wikipedia, *code coverage* describes the degree to which the source code of a program has been tested. It is a form of testing that looks at the code directly and as such comes under the heading of white or glass box testing.

We can identify different levels of test coverage [6]. According to the needs of the stakeholders that want to use the OSS, a certain level may be required. The

creative commons

following levels are ordered from low to high. Note that not every coverage tool is capable of reporting every level listed below.

- *Function coverage*: Reports whether each function or method of a program has been executed.
- *Statement Coverage (C0)*: Reports whether each executable statement is encountered. Also known as: line coverage. This metric is quite easy to check, but statement coverage is not a good indicator for the quality of the software because branches and conditions are the most critical areas of a program.
- *Decision Coverage (C1)*: Reports whether boolean expressions tested in control structures evaluated to both true and false.
- *Condition Coverage(C2/C3)*: Reports the true or false outcome of each boolean sub-expression, separated by logical 'and' and logical 'or' if they occur. Similar to Decision Coverage but has better sensitivity to the control flow.
- *Multiple Condition Coverage*: Reports whether every possible combination of boolean sub-expressions occurs.
- *Condition/Decision Coverage*: A hybrid measure composed by the union of condition coverage and decision coverage.
- *Modified Condition/Decision Coverage*: This is a structural coverage criterion requiring that each condition within a decision is shown by execution to independently and correctly affect the outcome of the decision. This criterion was developed to help meet the need for extensive testing of complex Boolean expressions in safety-critical applications. It is used e.g., to test complex software in the area of safety-critical computing. For example, the RTCA/DO178b, which is a document describing software engineering for commercial airborne systems, requires that structural code coverage analysis has to be used as an additional measure to ensure adequate testing [16], [17].
- *Path Coverage (C4)*: Reports whether each of the possible paths in each function have been followed.

Test coverage tools bring valuable depth to unit testing, but they're often misused. So it is wise to follow Andrew Glover's hints [7] to take a closer look at what the numbers on the coverage report really mean, as well as what they don't. Everyone faced with code coverage numbers in the context of software quality should beware of taking them as absolute numbers to judge code quality.

Non-Functional Testing

The second major area of testing is *non-functional testing*. In this area the system is compared to the non-functional requirements, such as performance (speed), security, accuracy, and reliability.

creative commons

Performance and load testing

- *Performance testing* is testing that is performed, from one perspective, to determine how fast some aspect of a system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system, such as scalability, reliability and resource usage. Performance testing can serve different purposes. It can demonstrate that the system meets performance criteria and it can compare two systems to find which performs better. Alternatively, it can measure what parts of the system or which workload causes the system to perform badly [8].
- *Load testing* is usually defined as the process of exercising the system under test by feeding it the largest tasks it can operate with. Load testing has the following goals [9]:
  - Expose bugs that do not surface in cursory testing, such as memory management bugs, memory leaks, buffer overflows, etc.
  - Ensure that the application meets the performance baseline established during performance testing. This is done by running regression tests against the application at a specified maximum load.
  - Verify that resource utilization increases linearly with the offered load, and that the response time increases slowly until the utilization of the bottleneck resource exceeds 70-80%.

Although performance testing and load testing can seem similar, their goals are different. On one hand, performance testing uses load testing techniques and tools for measurement and benchmarking purposes and uses various load levels. On the other hand, load testing operates at a predefined load level, usually the highest load that the system can accept while still functioning properly. Note that load testing does not aim to break the system by overwhelming it, but instead tries to keep the system constantly humming like a well-oiled machine [10].

The QualiPSo toolset should offer performance as well as load tests to be executed by the user. When we look at the non-functional tests it may be more the exception than the rule that an OSS itself comes with relevant tests in this area. However, we need tools to identify the means of an OSS under test to provide adequate information on its non-functional tests -- if there are any tests available at all. Even if we can find such tests the general problem remains on how to execute them and analyze the relevant information they will give us to judge about the OSS quality.

Creating accurate test data

In the context of load testing, it is necessary to emphasize the importance of having large datasets available for testing. The datasets should be populated in a manner similar to that which is expected in production. Many disruptive bugs simply do not surface unless you deal with very large entities such thousands of users in repositories such as RDBMS/LDAP/Active Directory, thousands of mail

creative commons

server mailboxes; multi-gigabyte tables in databases, deep file/directory hierarchies on file systems, etc.

This issue is of cause only relevant for certain kind of OSS products, which have to handle higher load like e.g. web servers or database systems, but for these it would be important if we want to perform load and stress testing within QualiPSo toolset. In this case we obviously will need automated tools to generate these large data sets.

Reliability testing

ISO 9126 defines six quality characteristics, one of which is reliability. Therefore it is important to look at the reliability testing aspect, too.

- *Reliability testing* (including stress testing and robustness testing) tries to break the system under test by overwhelming its resources or by taking resources away from it (in which case it is sometimes called negative testing). The main purpose is to make sure that the system fails and recovers gracefully -- this quality is known as recoverability. Where performance testing demands a controlled environment and repeatable measurements, reliability or stress testing induces chaos and unpredictability. To take the example of a Web application, here are some ways in which stress can be applied to the system:
    - double the baseline number for concurrent users/HTTP connections
    - randomly shut down and restart ports on the network switches/routers that connect the servers (via SNMP commands for example)
    - take the database offline, then restart it
    - run processes that consume resources (CPU, memory, disk, network) on the Web and database servers.

  According to [12] there is agreement on the intuitive meaning of reliable or dependable software: it does not fail in unexpected or catastrophic ways. Robustness testing and stress testing are variances of reliability testing based on this simple criterion.

  Robustness of a software component is the degree to which it can function correctly in the presence of exceptional inputs or stressful environmental conditions. Robustness testing differs with correctness testing in the sense that the functional correctness of the software is not of concern. It only watches for robustness problems such as machine crashes, process hangs or abnormal termination. Robustness testing does not break the system purely for the pleasure of breaking it, but instead it allows us to observe how the system reacts to failure:

    - Does it save its state or does it crash suddenly?
    - Does it just hang and freeze or does it fail gracefully?
    - On restart, is it able to recover from the last good state?
    - Does it print out meaningful error messages to the user, or does it merely display incomprehensible hex codes?

creative commons

- Is the security of the system compromised because of unexpected failures?

Acceptance testing

Acceptance testing is kind of black box testing, which provides the client the opportunity to verify the system functionality and usability prior to the system being moved to production. So, acceptance testing can cover functional as well as non-functional aspects.

Compatibility, conformance and interoperability testing

A distinct part of the QualiPSo project (Activity 3) deals with interoperability of OSS products at difference levels. In this area we have

- *Compatibility Testing*: these tests are concerned with verifying how well software performs in a particular hardware/software/operating system/network/etc.
- *Conformance Testing*: these tests are performed to determine whether a system meets a specified standard, i.e., the act of determining to what extent a single implementation conforms to the individual requirements of its base standard [13].
- *Interoperability Testing*: Interoperability testing is the act of determining if end-to-end functionality between (at least) two communicating systems is as required by those systems' base standards. Interoperability is often confused with the term "Interworking". But the latter relates only to the ability of dissimilar protocols (such as ISDN and GSM) to exchange service information [13].

OSS external functional and non-functional testing

Up to now we have only looked at possibly available tests provided by the OSS itself. The other important part of testing is providing tests developed inside the QualiPSo project (see wd 5.4.2 [19] for more information about this topic). These QualiPSo tests should either complement the information gained from evaluating the OSS internal tests or – in absence of such tests – the QualiPSo developed test suites and benchmarks are the only criteria to judge about the OSS quality from a testing perspective. In either case, based on the information provided in this chapter, chapter 5.2 will look at the resulting requirements for the testing and benchmarking tools to be developed in the QualiPSo project.

Analyzing quantity and quality of test cases

According to [14] we can state that

- A *test case* is a document that describes an input, action, or event and an expected response, to determine if a feature of an application is

creative commons

working correctly. A test case should contain particulars such as test case identifier, test case name, objective, test conditions/setup, input data requirements, steps, and expected results.

The process of developing test cases can help find problems in the requirements or design of an application, since it requires completely thinking through the operation of the application. For this reason, it's useful to prepare test cases early in the development cycle if possible.

Test cases should be a major contribution to every software product. Therefore an important quality check is to confirm that enough test cases have been specified. Up to now we have either assumed that an OSS SUT comes with its own set of functional and non-functional tests or the QualiPSo project itself has to develop and provide a suitable set of test suites. In either case, it is important to judge about the quality and quantity of the test cases, too.

The *quantity of test cases* is the absolute number of all test cases and test procedures respectively test scenarios. This number should be related to the number of test cases needed to reach defined test coverage. When full branch coverage has been specified, the QualiPSo testing toolset should check that the:

- Number of test cases for each module is greater than or equal to the Cyclomatic Complexity of the module [18]
- Paths traversed in the test cases actually result in every branch being traversed [18].

The *quality of the test cases* can be measured by calculating the average value of four attributes [15]:

- Test case impact = 1 –(#TestCases / #Impacted_Functions)
- Testcase reusability = #Automated_Test_Cases / #TestCases
- Test case effectiveness = (#TestCases / #Methods_Triggered_or_Affected_by_this_TestCase)
- Test case completeness = 1- (#Missing_or_Incorrect_TestCase_Attributes / #TestCase_Attributes)

Test case attributes are, e.g., test case name and ID, test case usage, name/type of input and output parameters, values of input and output parameters.

According to [15] it should be possible to measure the test case quality with a tool (e.g., CTFAnal as developed by Harry M. Sneed).

creative commons

APPENDIXB – DESCRIPTION OF SPAGO4Q

Since Spago4Q is used by several QualiPSo activities, it is described already in other documents. Nevertheless we give an overview here to make the document self-contained.

Spago4Q architecture, obtained as a verticalization of SpagoBI[30] (the Business Intelligence Free Platform) is designed in order to be easily adapted to complex organizational contexts. It integrates an advanced meta-model which makes Spago4Q fully independent from the adopted software development processes, infrastructure tools, measurement and assessment frameworks.

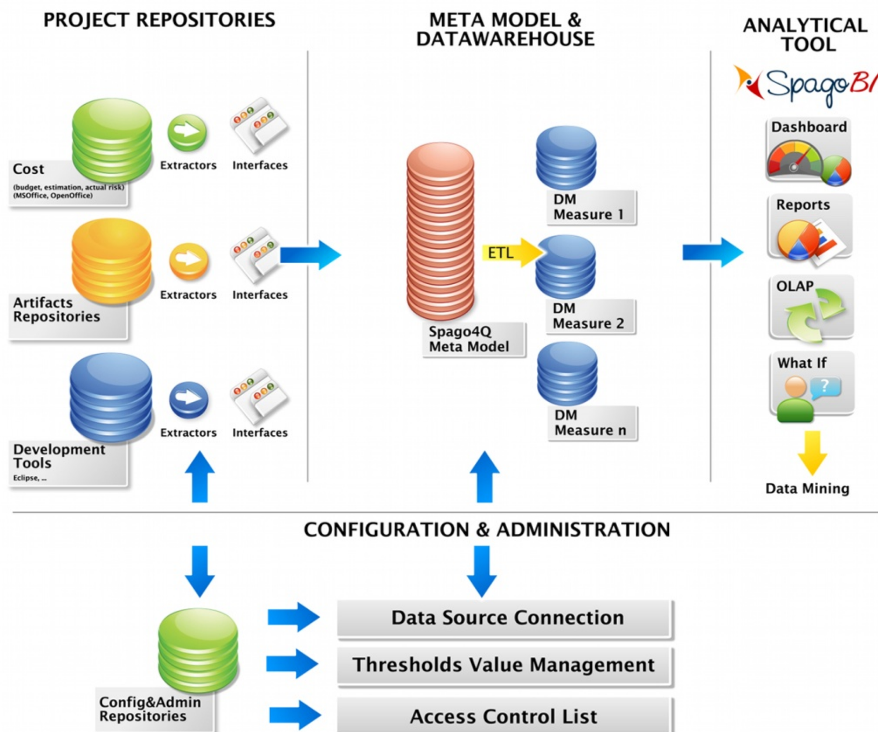


Figure 28: Spago4Q - Architectural overview

The main components are described in the following paragraphs.

Specialized extractors are used to collect measure attributes from the external sources.

Extractors are specialized components to collect measure attributes from different data sources. They have the following characteristics:

- collect measure attributes directly from infrastructure tools (i.e. bug tracking, source configuration management, ..)

[30] SpagoBI home: http://www.spagobi.org/

creative commons

- collect measure attributes or metrics parsing the output produced from tools (as example, Checkstyle)
- can load one or more “Interfaces” components
- XML is adopted to exchange data with “Interfaces” components;
- new extractors can be created to collect measure attributes from a specific tools;
- can apply rules to filter or transform data .

Interfaces are components that define the format for the input data to load into the warehouse. They create a decoupling layer between extractors and data analysis. They have the characteristics that for every area of measure (i.e. requirements, bugs, tests) only one interface is defined in order to standardize format input data collected from different projects or tools.

Extraction Process loads measure attributes into the *DWH-Spago4Q* data warehouse using one or more **Extraction Operation**.

An Extraction Operation is developed for each “Interface”, and it can apply rules to filter or transform data.

DWH-Spago4Q is the repository of all measure attributes collected from projects developed by an Organization.

SpagoBI analytical tools are used to analyze measure attributes and represent KPIs. The measure attributes inserted in the data warehouse has to be analyzed by the Spago4Q analytical component. This module has been implemented as a verticalization of SpagoBI, in order to cover and satisfy the whole range of BI requirements, in terms of analysis, data management, administration and security.

Configuration & Administration modules allow system configuration.

All the components described above are properly configured through the *Configuration & Administration modules* that provide the following characteristics:

- extractors configuration allows the definition of connections to repositories and tools;
- access control list;
- thresholds values management;
- GQM management.

Spago4Q integrates an advanced meta-model which makes Spago4Q fully independent from the adopted software development processes, infrastructure tools, measurement and assessment frameworks. Spato4Q enables you to collect data easier and provides analytical components for data analysis and project assessment as well.

creative commons

Spago4Q meta-model follows the Meta-Object Facility (MOF) approach proposed by Object Management Group (OMG). In Spago4Q, three major meta-model components have been identified: Process, Measurement, and Assessment meta-models.

The **Process Meta-model** has been defined starting from a simplified version of the OMG's SPEM (Software Process Engineering Meta-model) specifications, which describes a concrete software development process.

The **Assessment Meta-model** fully supports CMMI (Capability Maturity Model Integration) as assessment framework. Therefore, the keeping of CMMI practices is monitored by specific KPIs and metrics, examining their values with respect to predefined thresholds. Spago4Q gives at a glance a reliable snapshot of company state and allows managers and assessors to monitor the adherence of CMMI goals.

By means of the meta-model it is possible to represent:

- specific development processes (e.g.: UP, SCRUM, waterfall, evolutionary)
- assessment framework (CMMI, ISO 9001:2000)
- activity measure areas (e.g.: requirements, bugs and issues, testing, risks)

The **Measurement meta-model** has been defined following the Goal-Question-Metric (GQM) paradigm. Following GQM principles, the meta-model defines three main entities:

- measurableConcept: the goals of the analysis, i.e the concepts to measure
- measurableAttribute: the attributes to be measured for evaluating a specific goal
- KPI (Key Process Indicator) and metric: the operations to apply on measured attributes

The **Measurement module and GQM Management** is based on GQM (Goal-Question-Metric) approach and defines a generic framework exploitable to get measures from any external sources. In this context the Measurement module is a logical module: the group of objects used to manage the measurement area. It is therefore not a part of the architecture, but it is a part of the database used to store a GQM framework.

The Information Need (Goal) node is the container node that defines the need that drives all the measurement actions. For each information need there is a set of Measurable Concept (Question) nodes. These concepts drive the definition of the Measurable Attributes (Metrics), which indicates the attributes to be measured in order to accomplish the analysis goals.

The Measure node defines the structure of data retrieved during a measurement campaign. This node is directly connected to attributes and supplies raw data to KPI and Metric. Each node is specified by the nodes Scale

creative commons

Type and Unit, defining, respectively, the unit of measurement used and the type of scale adopted (nominal, ordinal, and so forth).

The monitoring indicators are defined in terms of KPIs (Key Process Indicator) and Metrics (elementary or aggregated). Finally, the Metric class is in relation with the Threshold entity specifying the threshold values for each metric when needed for qualitative evaluations.

creative commons

APPENDIX C- INFORMATION THAT CAN BE EXTRACTED FROM JABUTI

JaBUTi implements four intra-method control-flow based testing criteria (all-nodes-ei[31], all-nodes-ed, all-edges-ei, and all-edges-ed) and four intra-method data-flow based testing criteria (all-uses-ei, all-uses-ed, all-potential-uses-ei and all-potential-uses-ed).

Observe that the pair all-nodes-ei and all-nodes-ed and the pair all-edges-ei and all-edges-ed compose the traditional all-nodes and all-edges criteria, respectively [25], likewise the all-uses-ei and all-uses-ed compose the all-uses criterion [24] and the all-potential-uses-ei and all-potential-uses-ed compose the all-potential uses criterion [23]. We decided to distinguish between two different kinds of edges to represent the exception-handling mechanism of Java. Exception-independent edges (ei-edges) correspond to the normal control flow of the program, while exception-dependent edges (ed-edges) represent the control flow when an exception is raised. Exception-independent nodes are those reachable through a path that does not include any exception-dependent edge. Exception-independent def-use associations are those that can be covered through a path that does not included any exception-dependent edge. This distinction permits the tester to concentrate on different aspects of the program at a time, performing the testing activity in an incremental way.

JaBUTi evaluates the coverage of a given test set against its eight testing criteria, reporting the coverage obtained with respect to each one. It operates at unit level but provided aggregated testing reports by method, class, packages or testing project (composed by a set of packages and classes under test).

The tool works at byte code level and no source code is required to compute the testing requirements and the coverage. If the source code is available, the tool is able to map back the results computed from the byte code to its corresponding source code. In the current version, JaBUTi is able to import JUnit test sets such that their quality can be evaluated against the different structural testing criteria supported by JaBUTi. More information about JaBUTi can be found elsewhere [22].

Testing Coverage Criteria

All-Nodes-ei	This criterion requires that all statements not related with exception-handling mechanism were executed at least once.
All-Nodes-ed	This criterion requires that all statements related with exception-handling mechanism were executed at least once.

[31] ei and ed stand for exception-independent and exception-dependent, respectively, and they are used to distinguish testing requirements which demand no exception to be raised to be covered from the ones that mandatorily demand an exception to be raised to be covered.

creative commons

All-Edges-ei	This criterion requires that all conditional expressions were evaluated as true and false at least once .
All-Edges-ed	This criterion requires that each exception-handler be executed at least once from each node where an exception might be raised .
All-Uses-ei	This criterion requires that all definition-use associations not related with exception-handling mechanism were executed at least once .
All-Uses-ed	This criterion requires that all definition-use associations related with exception-handling mechanism were executed at least once .
All-Pot-Uses-ei	This criterion requires that all potential-use associations not related with exception-handling mechanism were executed at least once .
All-Pot-Uses-ed	This criterion requires that all potential-use associations related with exception-handling mechanism were executed at least once .

OO Metrics

NPIM	Number of public instance methods in a class
NIV	Number of instance variables in a class
NCM	Number of class methods in a class
NCV	Number of class variables in a class
ANPM	Average number of parameters per method
AMZ	Average method size
UMI	Use of multiple inheritance
NMOS	Number of methods overridden by a subclass
NMIS	Number of methods inherited by a subclass
NMAS	Number of methods added by a subclass
SI	Specialization index
NOC	Number of Children
DIT	Depth of Inheritance Tree

creative commons

WMC	Weighted Methods per Class
LCOM	Lack of Cohesion in Methods
RFC	Response for a Class
CBO	Coupling Between Object
CC	Cyclomatic Complexity Metric

creative commons

APPENDIXD – QUALITY ASSURANCE

Syd Chapmans Additional Coding Guidelines

A good supplement to these conventions is given by Syd Chapman in his document “Supplementary of Sun Java Coding Conventions” [32]. The following topics are based on his questions and recommendations.

- The code should not use any hard coded values (i.e., port numbers, install / classpath, server names etc.).
- The header(s) for the source code files should contain details of any included third party code along with dates of any changes to that included code and the author. This information is necessary to keep track of licensing issues with Open Source software.
- If the code is for general public consumption: are the javadoc comments present, useful, up to date and relevant? Is the javadoc information available to review? Do they follow the recommended guidelines?[32]
- Do variable and method names make the code self-documenting and promote understandability?
- Do variable, class or method names exceed the compiler name length limits?
- Are variables recycled and reused with different semantics?
- Are change flags properly used?
- Is there dead code or old commented out code littered about?
    - Should it be deleted, or remain commented out?
- Is the code clean and finished?
- Ensure there’s no uncalled, unneeded, redundant code or leftover stubs or test routines. Remove any unused variables that are redundant.
- Does the source file contain a copyright statement?
- Does the source file start with appropriate descriptive text?
- Are all parameters clearly identified as such?
- Are complex algorithms and code optimizations adequately commented?
- Complex areas, algorithms, and code optimizations should be sufficiently commented, so other developers can understand the code and walk through it.
- Are all comments consistent with the code?
- Do the comments lie? Are they out of date?
    - Do the comments actually describe the code? Make sure the two will match!

32 http://java.sun.com/j2se/javadoc/writingdoccomments/

creative commons

- Do code comments explain why something is done or do they state the obvious?
- Is allocated memory freed? It is not sufficient to rely on the JVM garbage collection only!
- Are all allocated objects (database connections, sockets, file handles etc.) freed when no longer required? Check especially that error code paths are freeing up objects that are no longer required. Ensure these objects are not freed more than once.
- Are all variable properly defined with meaningful, consistent and clear names?
- Does each class have appropriate constructors and “destructors” (i.e. public methods to free resources, like “close()”, “release()” or “shutdown()”)?
- Do any subclasses have common members that should be in the super-class?
- Can the class inheritance hierarchy be simplified? Do we have too many hierarchy levels?
- Are the right abstractions used?
- Are encapsulation principles violated?
- Do exceptions include enough information to communicate what failed and why?
- Are exceptions handled in the right part of the code?
- Is there any information loss about failure reason when exceptions are mapped and re-thrown?
- Are error-conditions propagated, logged or ignored?
    - Is a meaningful message emitted?
    - Messages should include what went wrong, with relevant clues why it went wrong, and a possible user action, or a useful exception. When exceptions are logged, sufficient information is always required to locate and understand the situation.
- Do entry logs include input parameters?
- Do exit logs include return values?
- Are entry and exit logs properly paired?
- Do major branches or decision points in the code include trace messages to indicate flow of control (i.e. logging with TRACE level)?
- Do empty catch blocks have at least a comment to say why it's OK to ignore them?

creative commons

Using Checkstyle scripts for code analysis

The utility archive A5_QA_utilities.zip contains the following information to support Checkstyle execution:

- "checkstyle_A5_checks.xml" (in the sub-directory "ProjectDirectory\src\main\config"): configuration file which chooses the checks to be executed and parameterizes them where necessary; additionally a filter can be set based on the severity of the findings
- "build_checkstyle.xml": build file to be used with Ant
- "checkstyle-4.4.jar": a modified version of Checkstyle which includes a patch for an error occurring sometimes when customized exceptions are used in the code to be analyzed
- "csRuntimeBugFilter.jar": severity based filter as add-on for Checkstyle
- "pom.xml": a "Project Object Model" which contains the information of a Maven project, in this case just the call of Checkstyle
- "ProjectDirectory": an example directory structure like it is typically used in Maven projects

The configuration file can be used for all kinds of Checkstyle executions. It can be load into Eclipse or other IDEs as well as be used by an Ant or Maven target.

Using Ant

Preconditions:

- Ant must be installed and should be included in the environment variable PATH.
- "checkstyle-all-4.4.jar" and "csRuntimeBugFilter.jar" must be available

How to create reports:

- Adapt the path properties in "build_checkstyle.xml".
- Make sure that the output directory for the reports exists.
- Call "ant –f build_checkstyle.xml" from the project root directory. The findings will be written to a csv, xml and/or html file as defined in the build file.

Using Maven

Preconditions:

- Maven must be installed and should be included in the environment variable PATH. Since Maven gets the needed plugins just-in-time during the execution, the patched Checkstyle version has to be copied to the Maven repository (usually in <user directory>\.m2\repository\checkstyle\checkstyle\4.4) after the first Checkstyle run.
- The structure of the project to be analyzed must follow the Maven conventions (see "ProjectDirectory") for an example.
- "pom.xml" must be copied to the root directory of the project.

creative commons

- Call “mvn jxr:jxr” to create a html representation of the code.
- Call “mvn checkstyle:checkstyle” to create a report. The report will be written to <root directory>\target\site and contain links to the HTML representation of the code which enables easy investigation of the findings (see Figure 25, Figure 26 and Figure 27 for examples).

creative commons