

<b>Title:</b>
Title: A10.4-D1 Report on End-User or Developer software components Evaluation
<b>Author(s)/Organisation(s):</b>
Dániel Kristóf / FÖMI, Dénes Magyarai / FÖMI, Santiago Cáceres/ETRA, Dominique Laurent/IGN, José Ignacio Gisbert/ETRA, Astrid Fichtinger/TUM
<b>Working Group:</b>
WP10 – Evaluation
<b>References:</b>
<ul style="list-style-type: none"> <li>• A10.1-D1 Report on Definition of evaluation methodology</li> <li>• A10.2-D1 Report on Framework Evaluation Planning</li> <li>• A10.3-D1 Report on Evaluation of HUMBOLDT Framework</li> <li>• A6.1-D4 Concept of component validation Final version</li> </ul>

<b>Short Description:</b>
This report deals with the main conclusions of WP6 with regards of the final user.
<b>Keywords:</b>

<b>History:</b>			
Version	Author(s)	Status	Comment
001	Dániel Kristóf, Dénes Magyarai	new	

## Table of contents

1	Introduction.....	3
1.1	Purpose and scope of A10.4-D1 .....	3
1.2	Relations to other HUMBOLDT work packages.....	4
2	Methodology.....	5
2.1	Software component validation methods .....	5
2.2	HUMBOLDT Roles.....	6
2.3	Infrastructure for technical validation .....	8
3	Continuous Integration .....	9
3.1	Concept of Continuous Integration .....	9
4	Code Analysis Server.....	11
4.1	Sonar Overview.....	11
4.1.1.1	Code Coverage .....	12
4.1.2	Rules Compliance .....	13
4.1.3	General Metrics.....	13
5	Server Infrastructure from the User Point of View .....	16
5.1.1	Contributing Developer .....	16
5.1.2	Framework Integrator.....	17
5.1.3	Decision Maker .....	17
6	Conclusions.....	20

# 1 Introduction

## 1.1 Purpose and scope of A10.4-D1

Both WP6 and WP10 work on different aspects of software quality. The distribution of work between WP 6 and WP 10 is:

- WP6: Technical validation: all technical characteristics of software quality.
- WP10: Usability testing: anything involving interaction with users.

This document describes the results of WP6 from the point of view of the end user.

The activities for the evaluation of the HUMBOLDT project are structured in the following tasks.

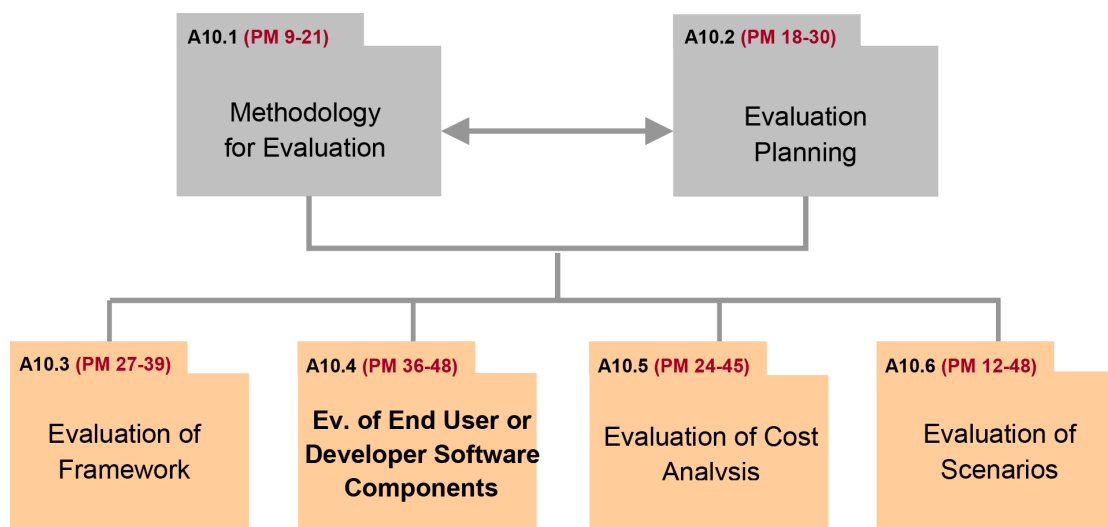


Figure 1: HUMBOLDT evaluation activities within WP 10

- **A10.3 Evaluation of the HUMBOLDT framework** (lead: HSRS). The evaluation will be executed following the guidelines provided in the framework evaluation planning. Once the first versions of the framework are being finalised and integrated, including feedback obtained from the user/applications, the evaluation process will be applied.
- **A10.4 End user or developer software components evaluation** (lead: FOMI). This task will carry out an analysis of the results derived from WP6 that are of interest for evaluation in a user-oriented approach. The leader of this activity is the work package leader of WP6; this assures coordination between both working groups.
- **A10.5 Evaluation of cost analysis** (lead: KTC). A complete evaluation report of the costs analysis performed in WP2 will be carried out within this activity. The task will follow as much as possible the guidelines described in this document and A10.2 Framework evaluation planning.
- **A10.6 Evaluation of scenarios** (lead: UWE). An evaluation of the results developed in the HUMBOLDT Scenarios will be carried out in this activity. The final report will include specific evaluation results for each of the different scenarios.

## 1.2 Relations to other HUMBOLDT work packages

Evaluation activities are essential for the success of a project such as HUMBOLDT. WP10 together with WP6 comprises all the evaluation tasks within the project and has to maintain close relations to other HUMBOLDT WPs (see figure 4). WP10 deals with the assessment from the user perspective while WP6 focuses on the technical assessment of the pieces (components, modules, libraries, ... ) of the HUMBOLDT Framework. This means that the technical assessment focused on testing the proper and efficient operation of the functions and services of the HUMBOLDT Framework system in terms of technical feasibility and integrity will be carried out in WP6. WP10 will take the WP6 results into consideration and add an evaluation of the performance of the HUMBOLDT system functions from a user perspective point of view in the context of usability and the degree of deviation of the system performance from its objectives. The evaluation results will be fed back into the specification and development process of the HUMBOLDT Framework (WP5 and WP8) as well as into the HUMBOLDT Scenario development. In addition, the assumptions about potential benefits generated by the HUMBOLDT project for the application scenarios made in WP 2 “Cost analysis“ are to be verified in the course of evaluation.

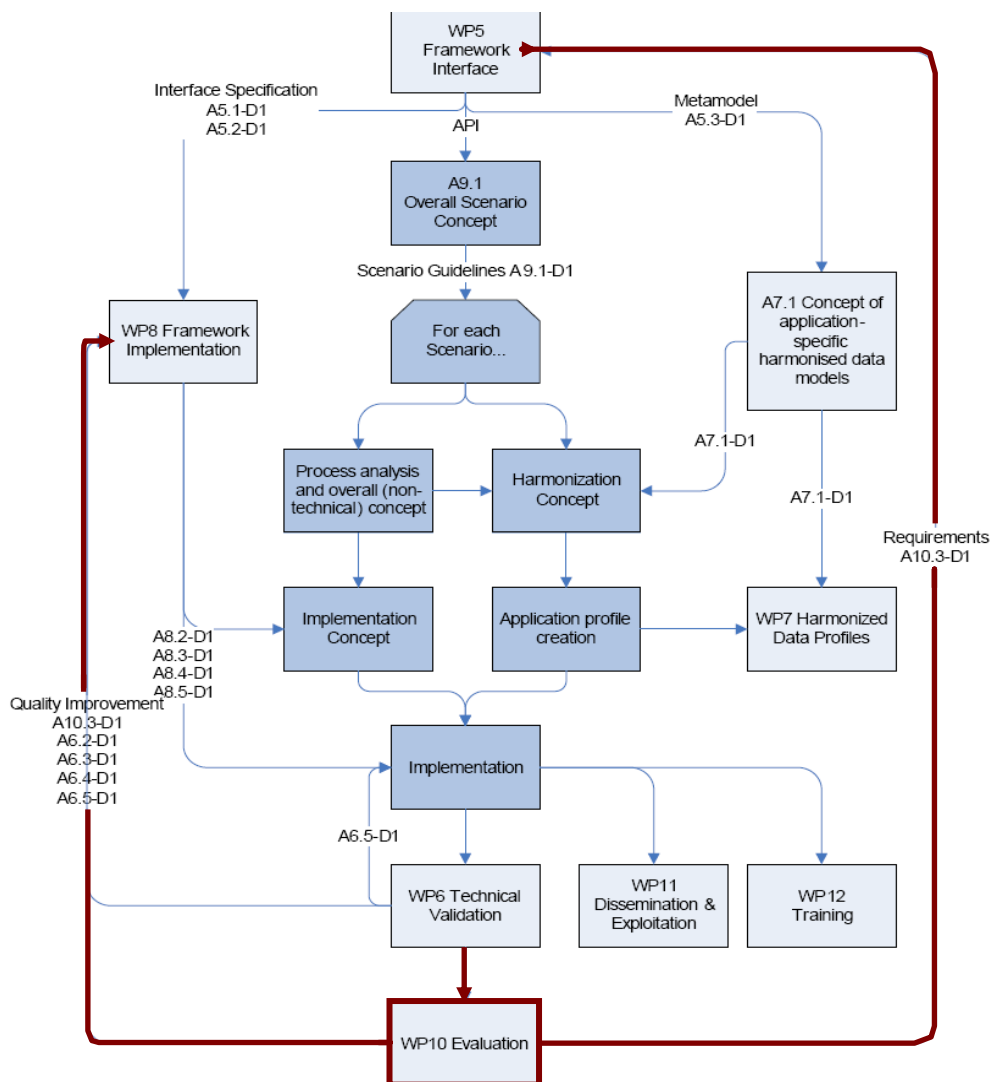


Figure 2: Relations of WP 10 to other HUMBOLDT work packages

## 2 Methodology

### 2.1 Software component validation methods

There are many different software validation methods for components. They could be classified into two classes:

- Black-box validation methods (also known as functional testing methods)
- White-box validation methods (also known as program-based testing methods, or structure-based testing methods)

Black-box method's base idea is a testing of a component as a black box. It focuses only on the external accessible behaviours and functions. It checks a component's outputs for selected component inputs.

Black-box testing methods for components can be classified into three groups:

- Usage-based black-box testing techniques – focusing on user oriented accesses
  - User-operation scenario testing,
  - Random testing,
  - Statistical testing,
- Error-based black-box testing techniques – focusing on error-prone points
  - Equivalence partitioning testing,
  - Category-partitioning testing,
  - Boundary-value analysis,
  - Decision table-based testing,
- Fault-based black-box testing techniques – focusing on targeted fault points
  - Mutation testing,
  - Fault injection method.

White-box testing methods are focusing on the internal structures of a component, on the internal component logics and behaviours of a component. These testing methods can be used in two purposes:

- Component developers use white-box testing methods to discover program-oriented errors in a component development process,
- Component users use white-box testing methods to check program errors in a component based software development process in the following cases:
  - Creating and checking newly created components,
  - Validating the extended and customized parts of adopted components.

From white-box testing methods for components can be classified into three types of testing:

- Path testing (based on program flow graph),
- Data flow testing,
- Object-oriented testing.

Path testing:

- Exercise program control flows of a component to check its outputs based on the given inputs
  - Use program flow graph as a test model,
  - Generate test cases based on the test model,
  - Apply test coverage criteria based on the test model.

Data flow testing:

- Focuses on program errors relating to incorrect data definition and usage
  - Program flow graphs are used as test model,
  - Use program flow graph to identify various data flow paths, such as data define and use path,
  - Generate test cases to exercise various data flow paths to achieve the selected data flow test coverage criteria.

Object-oriented testing:

- Testing object-oriented program by focusing on object-oriented program structures and features,
- Generate test cases based on selected object-oriented test models to achieve pre-defined test coverage criteria,
- Basic approaches:
  - Object state-based testing,
  - Class relation-based testing, inheritance testing,
  - Class function-based dependency testing,
- Test models:
  - Object state chart,
  - Class relation graph,
  - Class function dependency graph.

## 2.2 HUMBOLDT Roles

Within the HUMBOLDT project, user roles have been defined that follow a thematic domain categorization, instead of for example an organizational grouping. This decision was taken in order to

support the application-oriented approach of the HUMBOLDT R&D and contribute to the long-term sustainability of the project results.

The HUMBOLDT user roles are divided on five groups:

***HUMBOLDT Developers***

HUMBOLDT Developers are software developers who work on the HUMBOLDT framework directly or implement software based on the framework.

***HUMBOLDT Data Custodians***

HUMBOLDT Data Custodians are people or institutions offering data that have been adapted to given standards (harmonised) because of legal or market requirements. They provide data and have the responsibility to do so in a format that may be different from the one they normally use. They are mainly data providers. Examples of this group are the national INSPIRE-responsible bodies.

***HUMBOLDT Data Integrators***

HUMBOLDT Data Integrators are people who have to use heterogeneous geodata to meet the daily requirements of their job (e.g. integration of data for complex analysis). They need data themselves and access this from different facilities - potentially in different formats. They have to combine various data sources and harmonise them to make use of them for their own purposes. They are mainly service providers.

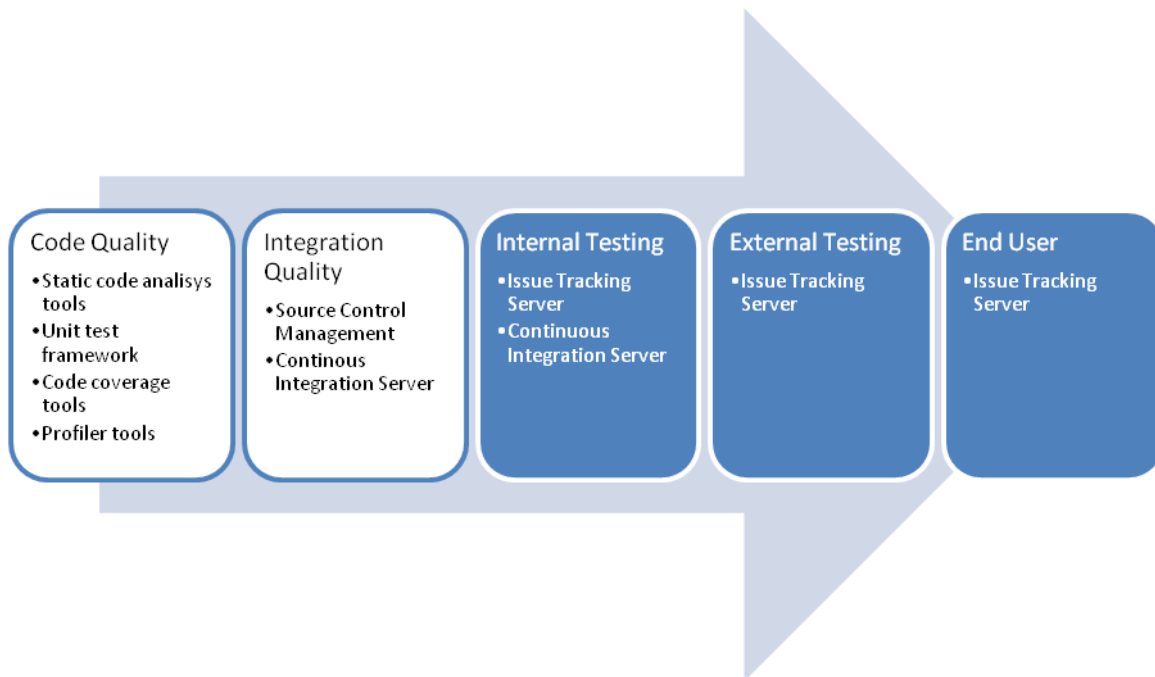
***HUMBOLDT End-Users of Geo-Data***

End users of geodata are users who are working with geographical data directly. They either use geodata in a harmonised form or geodata that doesn't need harmonisation or integration at all. They are interested in the toolset to support their generic data harmonisation needs, or in some of the scenario applications depending on their activity.

***HUMBOLDT End-Users of Spatial Information***

End-users of spatial information do not use geographical data directly, they only use information arising from it (indirect use of geodata). Most commonly, they are users at laymen level, e.g. people using navigation systems, online routing services, etc.

## 2.3 Infrastructure for technical validation



• *Figure 3: Tools supporting each phase of quality assurance*

The infrastructure for white box testing is mainly aimed for developers but may have interest for end users also. The quality assurance server infrastructure for developer support is accessible via public web interfaces. These interfaces carry useful information for users with multiple levels of technical skill sets.

### 3 Continuous Integration

As lead of WP6, FÖMI operates the Continuous Integration Server for HUMBOLDT (<http://humboldt.fomi.hu>). Hudson was chosen for its flexibility, configurability and simplicity.

#### 3.1 Concept of Continuous Integration

Definition by Martin Flower inventor of the concept: *Continuous Integration is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.*

A Continuous Integration Server monitors the source control server (in this case Subversion) for changes. If change occurs checks out or updates the source code and performs full or incremental build. Also automated tests are executed. Parses the result of the build and tests and publishes the results in various ways

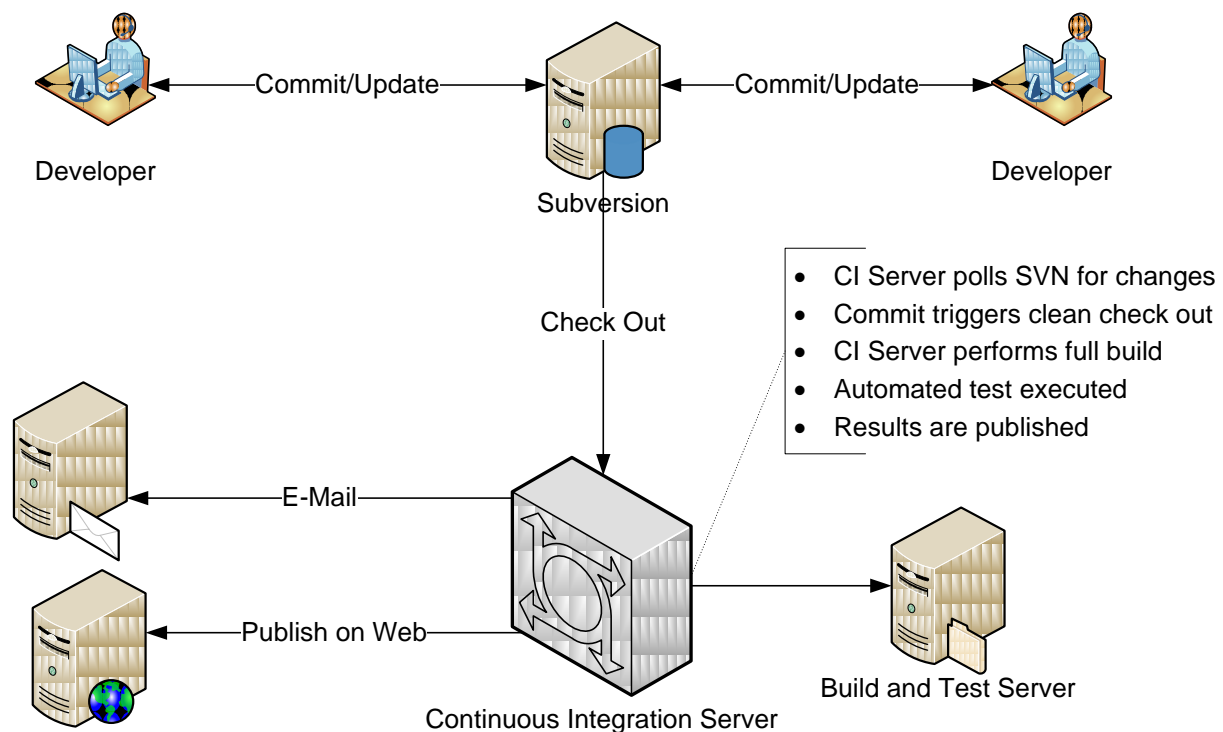


Figure 4: Overview of continuous integration

The CI server connects to the development infrastructure through the following interfaces:

- Access to Source Code Management server. This includes interpretation of tags or branches, possibly monitoring multiple branches of the same project simultaneously.
- Knowledge of the build process, to be able to execute it and parse its results.

- Execution of automated tests and the ability to parse its results. Unit tests might be included in the build script but most CI servers are capable of carrying out tests without it.
- The CI server must have access to the build and if tested, the deployment environment. Depending on the CI server architecture this might be a separate server or the CI server itself must have the necessary infrastructure.

## 4 Code Analysis Server

As the next step of improving the code quality server infrastructure, WP6 operates an instance of the open source Sonar code quality platform (<http://sonarsource.org>). Sonar is a platform for integrated monitoring and tracking of:

- code coverage
- standard metrics
- static code analysis results

FÖMI operates the Code Analysis Server for HUMBOLDT (<http://humboldt.fomi.hu/sonar>).

### 4.1 Sonar Overview

Sonar analysis is performed in the last phase of builds.

```
[INFO] -----
[INFO] Building HUMBOLDT Context Service Component
[INFO]   task-segment: [sonar:sonar] (aggregator-style)
[INFO] -----
[INFO] [sonar:sonar {execution: default-cli}]
[INFO] Sonar host: http://humboldt.fomi.hu/sonar
[INFO] Sonar version: 1.12
[INFO] [sonar-core:internal {execution: default-internal}]
[INFO] Database dialect class org.sonar.api.database.dialect.MySql
[INFO] ----- Analyse HUMBOLDT Context Service Component...
[INFO] Selected quality profile : HUMBOLDT profile, language=java
[INFO] Configure maven plugins...
[INFO] Executing sensor class org.sonar.plugins.core.sensors.JavaSourceImporter
[INFO] Executing sensor class org.sonar.plugins.core.sensors.AsynchronousMeasuresSensor
[INFO] Executing sensor class org.sonar.plugins.squid.SquidSensor
[INFO] Executing sensor class org.sonar.plugins.surefire.SurefireSensor
[INFO] parsing /var/lib/hudson/jobs/HUMBOLDT Context Service Component
(trunk)/workspace/trunk/target/surefire-reports
[INFO] Executing sensor class org.sonar.plugins.cpd.CpdSensor
[INFO] Executing sensor class org.sonar.plugins.core.sensors.ProfileSensor
[INFO] Executing sensor class org.sonar.plugins.core.sensors.ProjectLinksSensor
[INFO] Executing sensor class org.sonar.plugins.core.sensors.VersionEventsSensor
[INFO] Execute decorators...
[INFO] ANALYSIS SUCCESSFUL, you can browse http://humboldt.fomi.hu/sonar
[INFO] Optimizing database...
[INFO] Database purged in 600 ms.
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 29 seconds
[INFO] Finished at: Mon Sep 27 00:17:40 CEST 2010
[INFO] Final Memory: 12M/22M
[INFO] -----
Finished: SUCCESS
```

The static code analysis and the collection of code coverage data is integrated into the Maven build process and the results are pushed to the Sonar database.

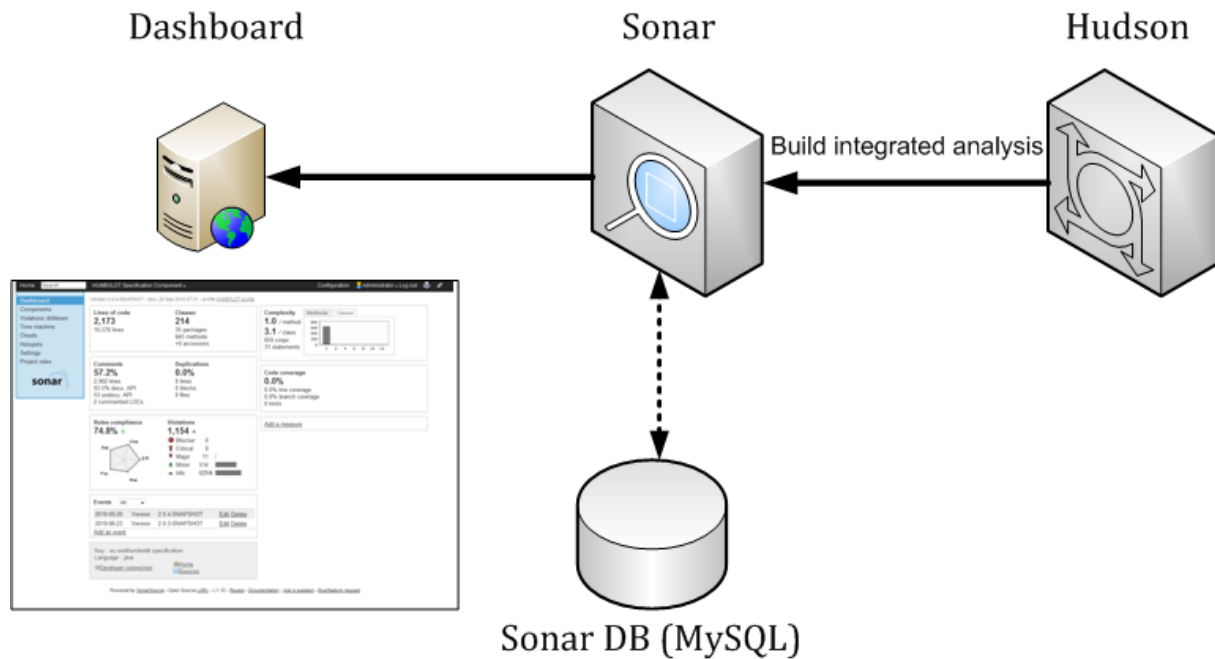


Figure 5: Overview of Sonar operation

Sonar then provides comprehensive analysis of results via its web interface. The web interface allows:

- hierarchical browsing of components, classes and methods
- charting trends of metrics, rules compliance and code coverage
- browsing source code overlaid with coverage and rule violations

#### 4.1.1.1 Code Coverage

Measuring code coverage is the process of detecting elements of source code that are executed during a certain drive of test code. In our case the execution of automated test cases drives the process of code coverage measurement. The measured elements of code can be:

- lines
- branches
- statements
- conditions

The measured coverage can be expressed by the percentage of code element covered against all elements.

There are no exact figures to be defined as acceptance criteria for coverage results. Code coverage rather indicates the strategy of development. Low coverage indicates a development process that is not driven by exact definitions of requirements. On the other hand, a very high coverage might still result software that does not satisfies the needs of the end user.

Trends of coverage percentage should stay steady as the code grows during development indicating that new code introduced to the product also comes with its tests.

## 4.1.2 Rules Compliance

Sonar integrates the most common static code analysis tools into a common platform. These include:

- Checkstyle
- PMD
- Findbugs

Static code analysis is a white box method of testing. These tools evaluate either the source code or the byte code without executing it, looking for common patterns of problems or errors. These are all semantic errors since the compiler will not allow any syntax error. Still a very large number of issues are commonly repeated during software development. Each rule by its nature is categorized into one of the following categories:

- Efficiency
- Maintainability
- Portability
- Reliability
- Usability

Rules are also categorized by severity into the following categories:

- Blocker
- Critical
- Major
- Minor
- Info

The collection of applied rules together with the each rule's severity attribute is called quality profile.

## 4.1.3 General Metrics

General metrics are mainly statistical counts of various code elements and some derived metrics calculated against results of code coverage and rule compliance. These include:

- Complexity
  - Complexity /class
  - Complexity /method
  - Complexity
  - Uncovered complexity
- Documentation
  - Commented LOCs

- Comments (%)
  - Comment lines
  - Public documented API (%)
  - Public undocumented API
- Duplication
  - Duplicated blocks
  - Duplicated files
  - Duplicated lines
  - Duplicated lines (%)
- Rule categories
  - Efficiency
  - Maintainability
  - Portability
  - Reliability
  - Usability
- Rules
  - Blocker violations
  - Critical violations
  - Info violations
  - Major violations
  - Minor violations
  - Rules compliance
  - Violations
  - Weighted violations
- Size
  - Accessors
  - Classes
  - Lines
  - Methods
  - Lines of code
  - Files

- Statements
- Packages
- Public API
- Tests
  - Branch coverage
  - Conditions to cover
  - Coverage
  - Unit tests duration
  - Line coverage
  - Lines to cover
  - Skipped unit tests
  - Unit test errors
  - Unit test failures
  - Unit tests
- Unit test success (%)
- Uncovered conditions
- Uncovered lines

## 5 Server Infrastructure from the User Point of View

The publicly available web interface for code quality assurance has different uses for different types of users.

### 5.1.1 Contributing Developer

As HUMBOLT is an open source software framework, after the lifetime of the project itself a formation of community of contributing developers might arise. These developers extend and enhance the framework with their own work. The drive for their development is to form the framework to their specific needs for a given application.

This community might produce solutions for commonly arising problems or even enhance the overall quality of the framework with fixes of issues. These additional work items might be integrated into the framework though a strict procedure so they become available for all users.

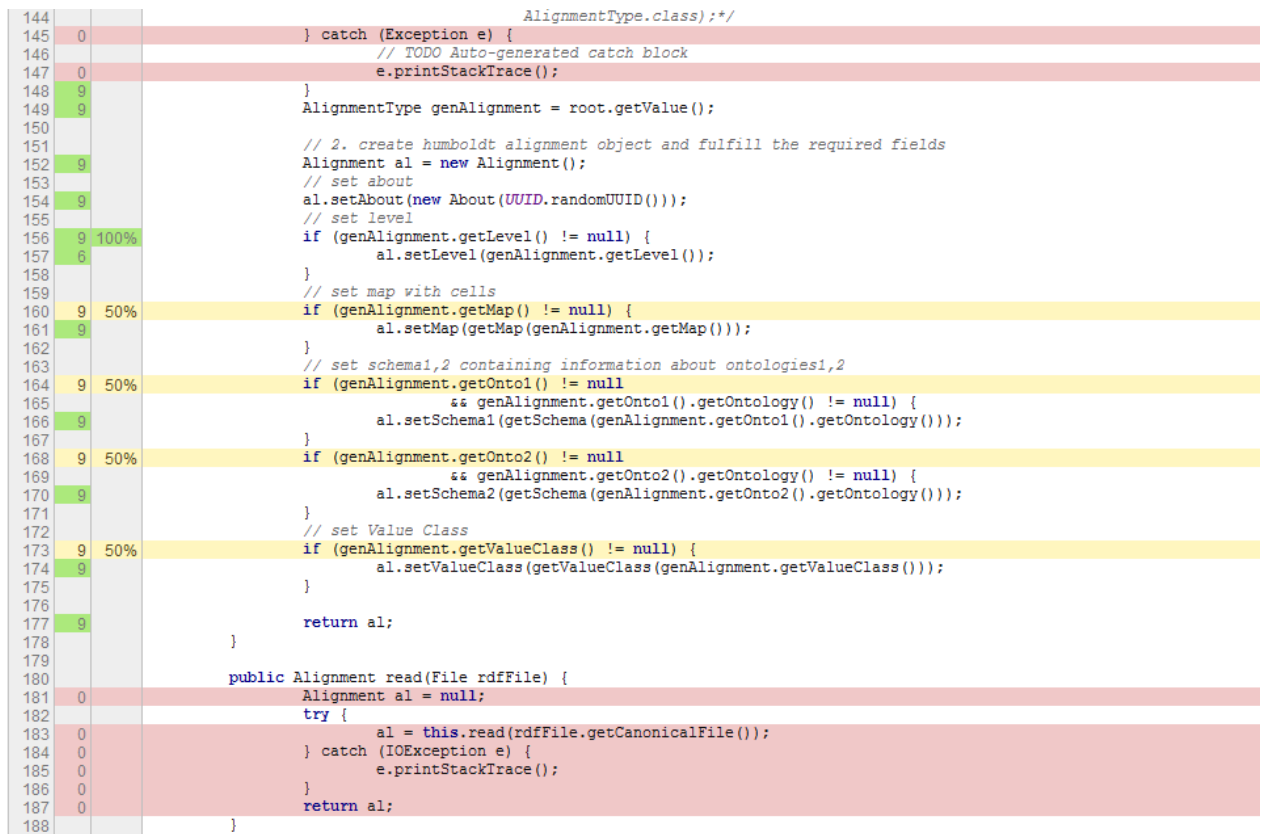


Figure 6 Drilldown into source code for coverage

External contributors of the main framework components might use the server infrastructure the same way as internal project developers:

- Checking CI server if the committed work complies and unit tests are successful
- Browsing the results of code analysis of their work, not having to implement their own static code analysis infrastructure

- Looking for hotspots in code for possible improvements and fixes
- Checking test coverage of new code

### 5.1.2 Framework Integrator

Framework integrator is a developer who will integrate framework components into his own product development. This role has a very high technical skill set, but does not contribute to the framework itself and might be interested in the latest, even unstable developments for producing the most advanced product.

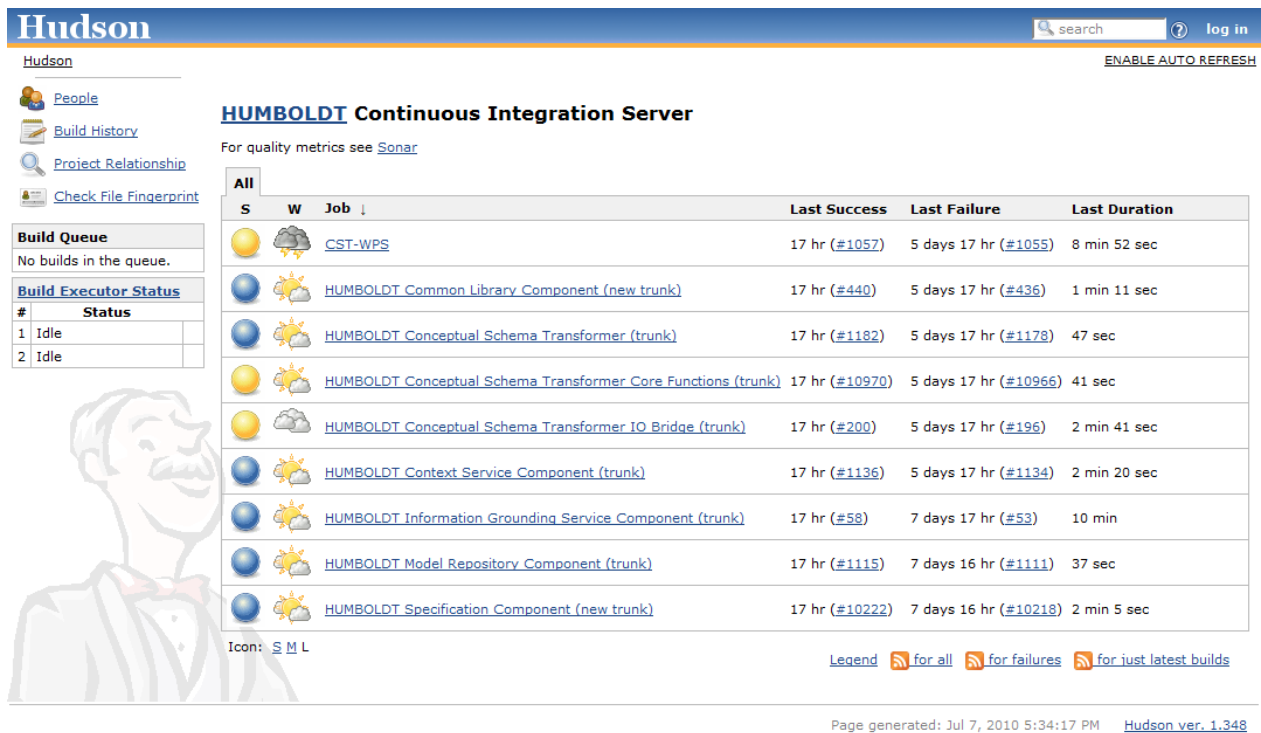


Figure 7: Overall state of development branches

The framework integrator might use the server infrastructure for the following tasks:

- Checking weather current development braches are stable
- Overview of the overall quality of the framework development
- Browsing source code for hotspots that might affect his development strategy
- Adapting quality profiles for his own development

### 5.1.3 Decision Maker

A decision maker is a responsible person for making strategic decisions on projects, weather HUMBOLDT should be used for their needs. The following aspects might take into count on evaluation:

- Overall activity of the project
- Responsibility to arising problems
- Concept and methodology of development
- General measures and metrics of the project
- Trends reflecting quality

The decision maker might not necessary has deep technical knowledge, rather general concepts of development. The public interface for code quality has a nice interface that supports a transparent overview for the state of the code.

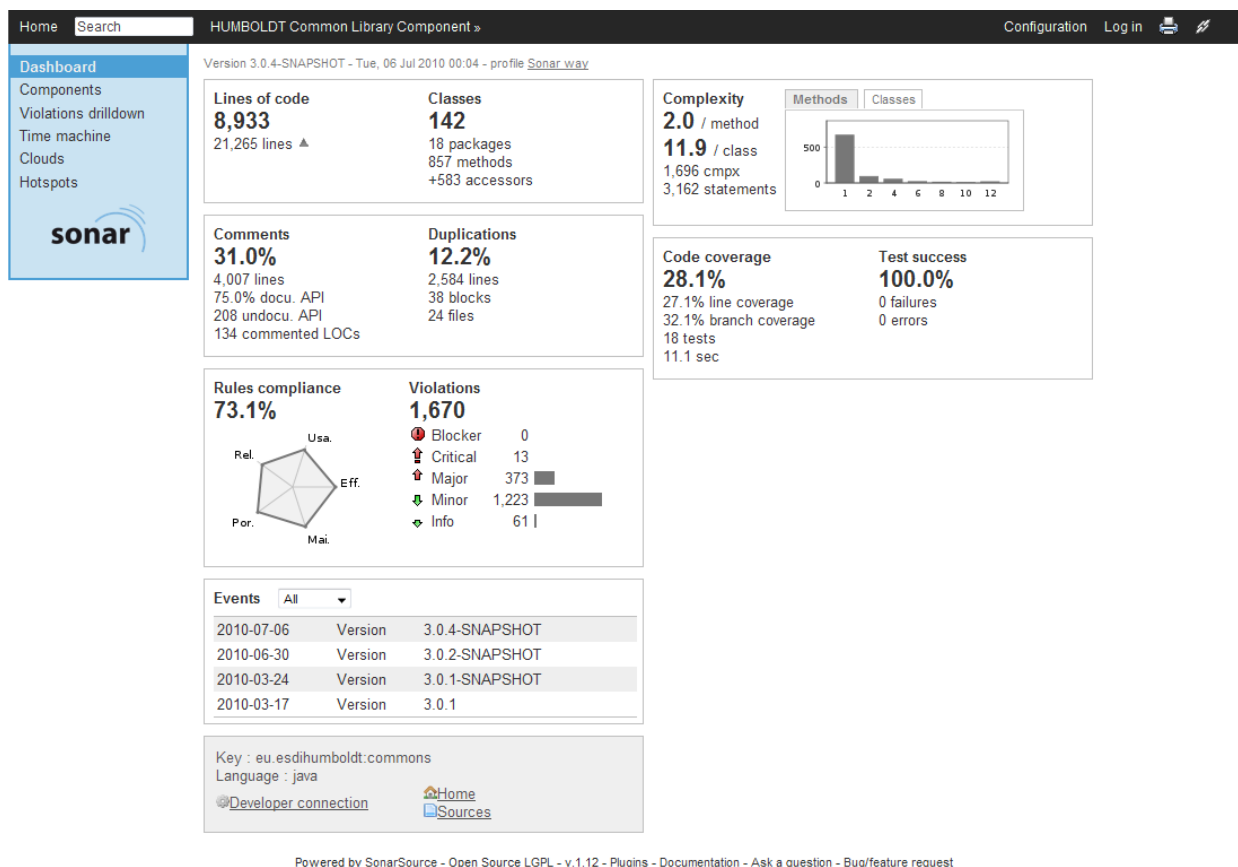


Figure 8: Sonar dashboard

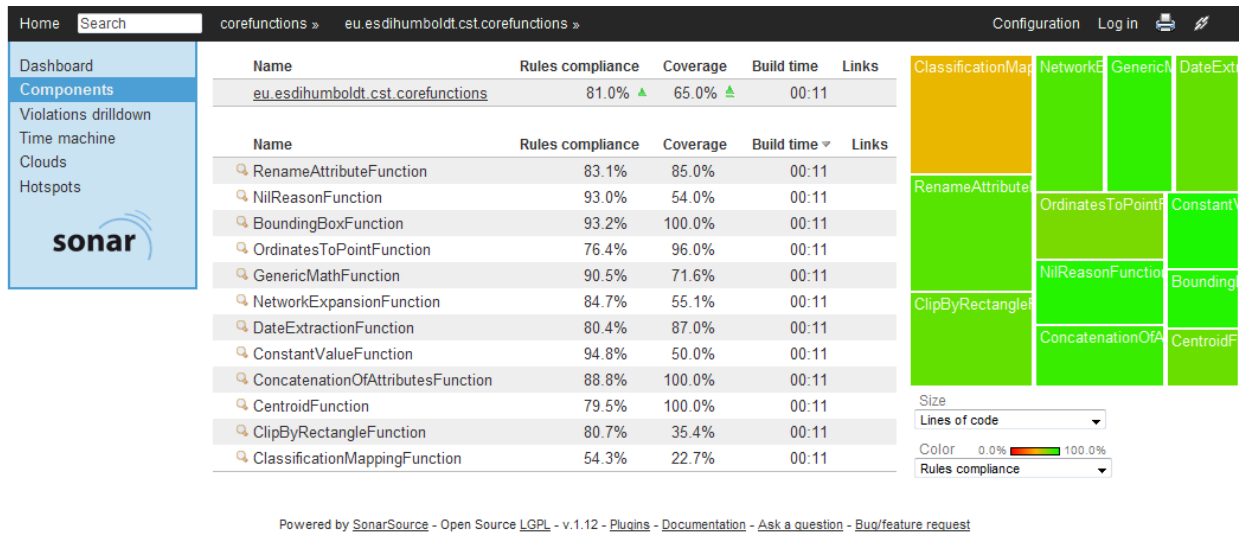


Figure 9: Overview of components

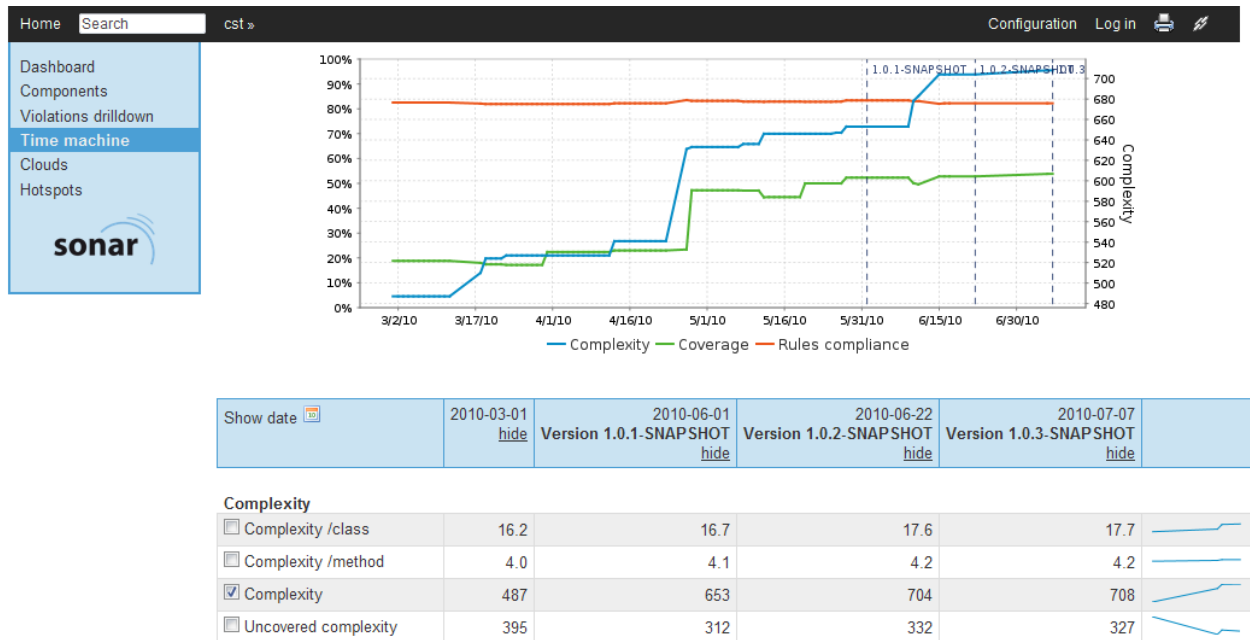


Figure 10: Trends of measures and metrics

## 6 Conclusions

The infrastructure for code quality monitoring is a publicly available for view in (<http://humboldt.fomi.hu/hudson> and <http://humboldt.fomi.hu/sonar>). The main interest for these tools are supporting developers as the full understating of the information provided need technical high technical knowledge, but these interfaces provide useful information also on general overall state and aggregated statistics that are understandable with only general understanding of concepts.

The public availability of these servers might be useful for:

- decision makers
- framework integrators
- framework contributors