QualityGate SourceAudit: A Tool for Assessing the Technical Quality of Software

Tibor Bakota
FrontEndART Software Ltd.
Zászló u. 3 I./5. H-6722 Szeged, Hungary
bakotat@frontendart.com

Péter Hegedűs, István Siket
MTA-SZTE Research Group on
Artificial Intelligence, Szeged, Hungary
{hpeter,siket}@inf.u-szeged.hu

Gergely Ladányi, Rudolf Ferenc
University of Szeged
Department of Software Engineering
Árpád tér 2. H-6720 Szeged, Hungary
{lgergely,ferenc}@inf.u-szeged.hu

***Abstract*—Software systems are evolving continuously in order to fulfill the ever-changing business needs. This endless modification, however, decreases the internal quality of the system over time. This phenomena is called software erosion, which results in higher development, testing, and operational costs.**

The SourceAudit tool presented in this paper helps managing the technical risks of software deterioration by allowing immediate, automatic, and objective assessment of software quality. By monitoring the high-level technical quality of systems it is possible to immediately perform the necessary steps needed to reduce the effects of software erosion, thus reaching higher maintainability and lower costs in the mid and long-term. The tool measures source code maintainability according to the ISO/IEC 25010 based probabilistic software maintainability model called ColumbusQM. It gives a holistic view on software quality and warns on source code maintainability decline.

I. Introduction

Because of continuous changes, software systems are exposed to quality deterioration – software erosion [1] – and as a result, the development and testing expenses increase, as well as the operational risks. Maintenance of custom-made software in the course of its complete life-cycle often exceeds the initial costs of development.

In the case of organizations, operating large number of custom-developed software systems to reduce the effect of erosion is crucial. The risks that are due to quality deterioration of even functionally well operating software systems are taken by the contracting party of the development or the party that will operate the system. Without an objective evaluation of source code maintainability, the quality of the delivered system can be judged only relying on the results of the acceptance tests, which characterize the functionality of the system exclusively, but do not provide information on its maintenance costs and operational risks.

The *SourceAudit* tool, member of the *QualityGate* product family,[1] is a software quality management tool, which allows immediate, automatic, and objective assessment of software quality. The tool measures source code maintainability using our ColumbusQM [2] maintainability model and provides a holistic view on the change of software quality. It warns on source code maintainability decline and helps in improving source code quality and performance of development teams. The tool supports also software operating companies by automatically monitoring the source code quality of their software systems.

[1] http://www.frontendart.com

SourceAudit includes the important features of other existing tools (e.g. SIG maintainability model, QUAMOCO, Sonar SQALE, or SQUALE) and extends them with trend analysis, maintenance cost estimation, and a drill-down mechanism [3] for assessing the maintainability of individual source code elements (e.g. classes and methods).

II. Background

A. Measuring Source Code Maintainability

The theoretical basis of quantifying source code quality is laid down in the work introducing ColumbusQM [2]. One of the keystones of the measurement is the so-called reference database (benchmark), which contains source code characteristics of numerous software systems. The reference database is the basis for comparison of the software system to be evaluated. Using the same reference database, maintainability of different systems or the maintainability of many versions of the same system have become comparable, i.e. the change of the maintainability may be observed in time.

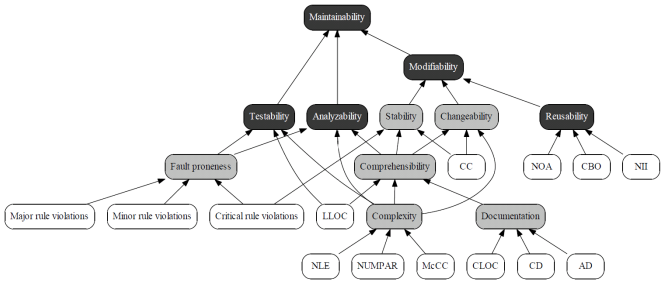

Figure 1. An example of a source code maintainability model

The other important element of measuring source code maintainability is the maintainability model, which is a directed acyclic graph, whose nodes represent low- and high-level characteristics (see Figure 1). The low-level characteristics – the nodes without input edges (sensor nodes) – can be directly calculated from the source code with the help of static analysis (e.g.: the complexity of source code elements, the ratio of serious coding errors, the ratio of code duplications). The high-level characteristics – the nodes with input edges (aggregated nodes) – represent the quality attributes with subjective meaning that cannot be computed directly from the source code (maintainability, stability etc.). The edges among the nodes of the graph indicate a dependency. The model provides the possibility of weighting the edges of the graph by software developer experts, so the subjectivity of the notions during the aggregation becomes manageable. Weighting given

978-1-4799-3752-3/14 © 2014 IEEE

by each expert is considered as one vote, and the probability distributions of the votes on the edges represent the subjective opinions of the experts.

The last building block for maintainability measurement is the method of aggregation itself. In the case of a software system to be assessed, the computation of low-level source code characteristics happens first. Next, they are compared to the software systems in the reference database. The result of the comparison is a probability distribution function, defined for each low-level characteristic of the model. Then, the already calculated probability distributions along the edges in the model are being aggregated – according to the votes' distribution assigned to the edges – with the help of statistical methods, which yield an objective measure for high-level characteristics. Repeating the process several times the Maintainability node – the root of the graph – also gets a measure.

B. Relationship between Source Code Maintainability and Development Costs

The importance of source code maintainability arises from its obvious relationship to software development costs. Maintainability is defined most commonly, as the effort required to modify the behavior of the software. The formal relationship between development costs and source code maintainability is described in a previous paper by Bakota et al. [4].

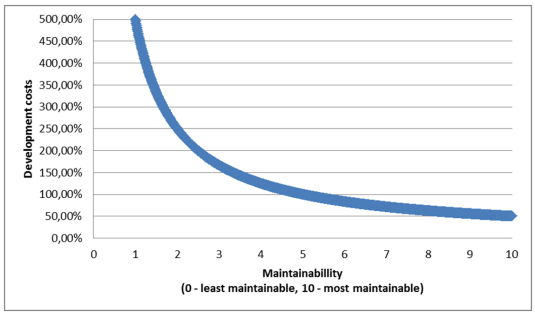

Figure 2. The connection of maintainability and development cost

According to the introduced cost model, source code maintainability is an exponential function of the effort put in the development, i.e. the development effort degrades source code maintainability in an exponential rate. The model makes it possible to express the development costs of a software system relative to another one having an average maintainability. Figure 2 represents this ratio of development costs for different maintainability values. The SourceAudit tool is able to present the maintenance costs of a software system according to this cost model.

III. Using QualityGate SourceAudit

Continuous integration. The QualityGate product family comes with continuous integration support in form of a Jenkins plug-in. The plug-in is capable of managing the whole analysis process, by performing the following steps:

- Regularly checks for changed source code in the version control system.
- Performs static source code analysis of the source code by using the *QualityGate CodeAnalyzer* tool, which computes source code metrics, and detects coding rule violations and code duplications.
- Uploads the analysis results to the central QualityGate repository.
- Computes the source code maintainability based on the models and benchmarks in the repository.
- Visualizes the results on the SourceAudit web-based graphical user interface.

User interface. The user interface of SourceAudit provides a holistic view of the quality of software systems. After logging in, the user can see the three main function groups of the system, appearing as tabs on the top of the page: *Certification*, *Quality Model*, and *Benchmark* (see e.g. Figure 3).

In the following, the main use-cases of the user interface will be described in detail.

A. Benchmark Management

For measuring software maintainability, a reference database (so-called benchmark) is needed. The quality of a system can be quantified relative to the systems in this reference database. SourceAudit provides a default benchmark database, which contains hundred open- and closed-source software systems and their analysis results.

Viewing benchmark details. By clicking on the information box of a benchmark listed on the *Benchmark tab* (see Figure 3), statistical data of the particular benchmark can be seen. The page lists the name, description, the number of systems in the benchmark, as well as the contained systems and the date of last modification. The mean and range values of some important source code metrics (number of lines of code, packages, classes, methods, etc.) of systems in the benchmark can be seen as well. The related quality models are also listed, as well as the metric values, which are shown on pie charts.

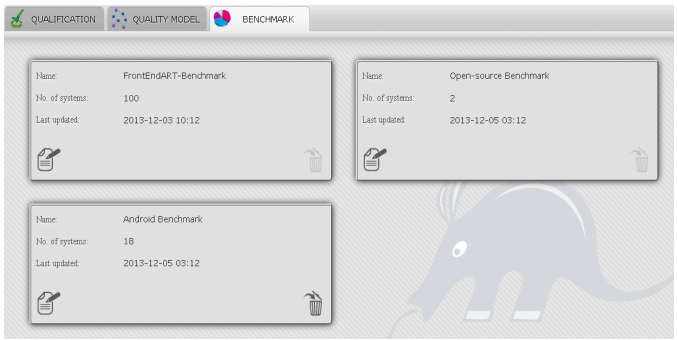

Figure 3. The benchmark tab

Creating new benchmarks. The users are able to create new benchmarks by selecting the systems, which should be included. After creating a benchmark, a new information box representing the newly created reference database appears on the Benchmark site.

Editing and deleting benchmarks. The name, description and the systems contained in the benchmark can be modified at any time. Deleting an existing benchmark is also possible but this operation can be executed only if no quality model is referring to this particular benchmark.

B. Quality Model Management

Software quality assessment is carried out according to a quality model. A model is a directed acyclic graph (see Figure 1), consisting of low- and high-level characteristics. The quality model management tab provides a possibility to the users to create and calibrate their own models.

Viewing quality model details. Details of a quality model appearing on the *Quality model tab* can be viewed by clicking on the information box representing the particular model. Apart from the basic information (e.g. model name, description, benchmark used, number of nodes) the graph of the model can also be seen. By double-clicking on the nodes of the model, supplementary information for the model element can be gained (e.g. name, description, for what type of source code elements it is applicable). In addition, the distribution of expert votes on a given dependency can also be visualized. The elements of the quality model can be rearranged in any way for the sake of better overview.

Creating a quality model. It is possible to create a new quality model at the Quality model tab. After giving a name and a description for the new model, it is necessary to assign a benchmark to it (each model may use only one benchmark). The new model can be created with the help of a graphical editor presented in Figure 4. In order to make the model suitable for quality assessment, weighing the relations among model nodes is needed, which can be done by assigning votes for the edges of the model.

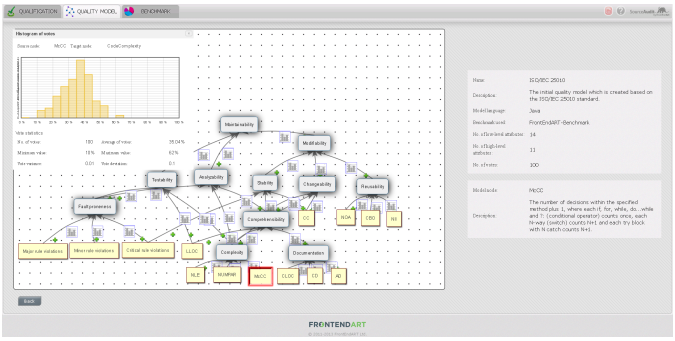

Figure 4. The default Java quality model

Editing and deleting quality models. Editing an existing model is possible only if there are no votes assigned to its edges yet. Users can also delete a quality model.

C. Certification View

Maintainability assessments based on the different models in SourceAudit can be viewed on the *Certification tab* (see Figure 5). Each information box represents an evaluation of a system's maintainability according to a specified model. Apart from the name of the system and the model used for validation, the latest assessment date and the result of the assessment can be seen. The value of the evaluation is defined on a ten-point scale (0 is the worst, 10 the best) and it expresses the quality of the given system compared to the systems in the benchmark used by the model. The most important system-level metrics of the latest assessed source code version can also be displayed (e.g. total lines of code, number of classes, copy-paste ratio).

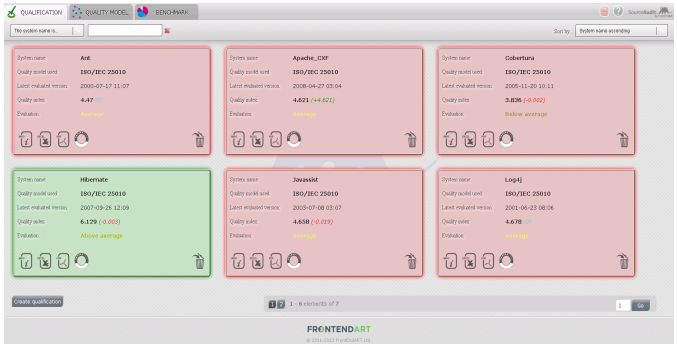

Figure 5. Certification view

Viewing certification details. By choosing the appropriate information box on the Certification tab the evaluation results of the system according to the given model can be seen (see Figure 6). The stars at the top of the page indicate the quality of the system.

The timeline, located in the middle of the page, shows the system's quality change in time. By clicking on the points on the timeline, details regarding the root causes of the change appear (green color indicates quality improvement, red means quality decrease). On the appearing window, it is possible to generate PDF or Excel reports for the selected version. By clicking on the *cost* icon, the user can toggle between the quality and the relative maintenance cost computed for the system, which is shown in percentages and indicates the cost of maintenance compared to an averagely maintainable system (100% stands for the maintenance cost of a system with a quality value of 5, see Section II for more details).

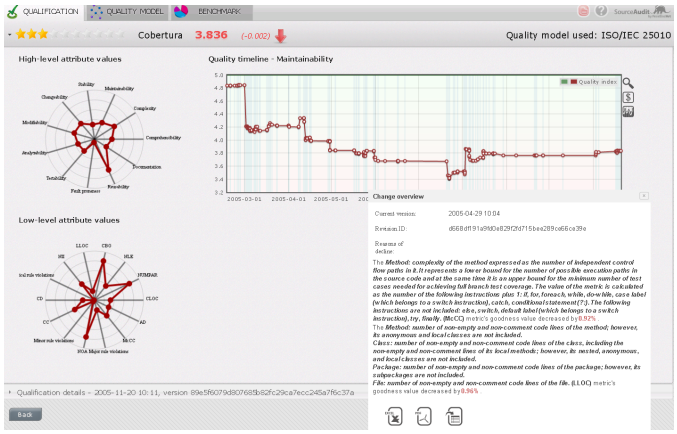

Figure 6. The certification details of a system.

Certification breakdown. It is possible not only to view the high-level quality changes over time, but to select a particular version of the source code and breakdown to the root causes of the certification results. The main benefit of the function is that low-level technical information can be gained without having to generate a report as the concrete source code parts of the affected elements can also be displayed.

The tool provides an overview table with the following information: coding rule violations introduced in the current version, changes in source code metrics causing the change in the system level maintainability, the newly introduced copy-paste code parts, etc. Besides this overview, the actual source

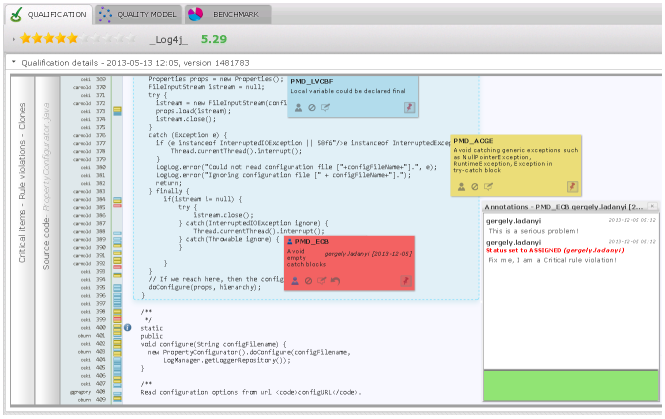

Figure 7. Annotations and source view

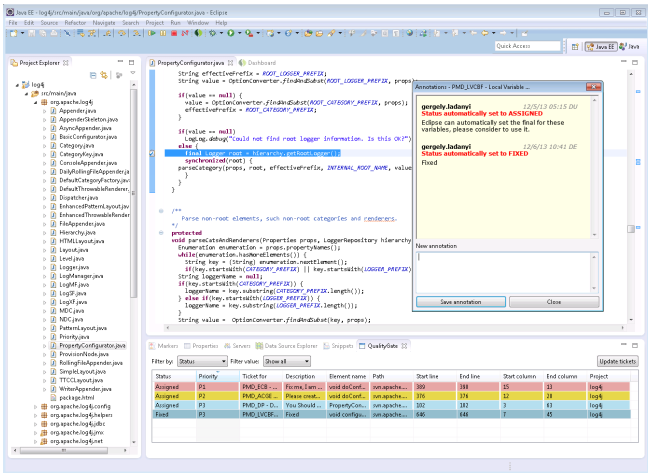

Figure 8. SourceAudit Eclipse plug-in

code of the affected source code elements can be downloaded from the version control system and displayed decorated with the rule violations and copy-paste parts existing in them (see Figure 7). What is more, users can directly annotate code parts or create tickets for a particular issue. This information gets uploaded into the central database which is also read by an Eclipse plug-in that fetches the created issues and annotations and displays the information to the developers right in the IDE (see Figure 8). This is an especially useful feature as there is no need for extra communication between managers, architects, and lead developers (who are more likely to use SourceAudit for getting a quality overview of a system) and programmers (who typically use IDEs).

Executing quality assessments. In general, quality assessments are executed automatically by Jenkins, as described above. However, it is also possible to manually start assessments of systems that have earlier been uploaded to SourceAudit. After selecting a quality model, the versions of the system for which the validation is to be performed should be given. Subsequently, the user gets back to the Certification tab showing the new information box corresponding to this certification. It is also possible to delete any assessment of a system. By deleting a certification, the source code characteristics of the software are not deleted, thus, the validation can be restarted.

Generating reports. It is possible to generate a *PDF* report of the quality of a given system for stakeholders and managers, or an *Excel* report, which provides a technical-level overview of the quality for a given version of the system.

Generating a widget code. The widget code is an HTML code that can be embedded in a website, and which keeps track of the source code quality in form of a stamp logo. The name of the validated system, the latest validation date and the actual quality value appear on the stamp (see Figure 9).

Figure 9. The qualification stamp logo

IV. Conclusions

In this paper we introduced the SourceAudit software-quality management tool, member of the QualityGate product family. It aims to provide an easy-to-understand, holistic view of the high-level quality of software. The tool implements a state-of-the-art ISO/IEC 25010 based probabilistic quality model – called ColumbusQM – for calculating technical quality and integrates the cost model derived from it. SourceAudit includes all the important functionalities provided by similar tools and extends them in several ways (e.g. with trend analysis, source code element level maintainability).

On one hand, our tool is useful for managers at companies operating software systems for monitoring the quality of their software at high level. On the other hand, SourceAudit also helps software developers to gain low-level technical information of the quality of their code and on how to improve it. For them, we also provide an Eclipse IDE plug-in to make it easier to get the information from SourceAudit (e.g. tickets and comments added by project leaders).

Acknowledgment

This research and development was supported by the Hungarian national grant GOP-1.1.1-11-2011-0006, and the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TÁMOP 4.2.4. A/2-11-1-2012-0001 “National Excellence Program”.

References

[1] D. L. Parnas, “Software Aging,” in *Proceedings of the 16th International Conference on Software Engineering*, ser. ICSE ’94. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 279–287.

[2] T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, “A Probabilistic Software Quality Model,” in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM 2011)*. Williamsburg, VA, USA: IEEE Computer Society, 2011, pp. 368–377.

[3] P. Hegedűs, T. Bakota, G. Ladányi, C. Faragó, and R. Ferenc, “A Drill-Down Approach for Measuring Maintainability at Source Code Element Level,” *Electronic Communications of the EASST*, vol. 60, 2013. [Online]. Available: http://journal.ub.tu-berlin.de/eceasst/article/download/852/846

[4] T. Bakota, P. Hegedűs, G. Ladányi, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, “A Cost Model Based on Software Maintainability,” in *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM 2012)*. Riva del Garda, Italy: IEEE Computer Society, 2012, pp. 316–325.

APPENDIX

During the demonstration we use a SourceAudit tool installation publicly available at the following URL: http://columbus.inf.u-szeged.hu:10000/QualityGate. We show the certification details of an open-source project, going through the following steps.

a) Present the applied quality model: The SourceAudit tool is able to evaluate systems according to an arbitrary quality model. However, it comes with a default quality model defined for the Java language. All the presented results are prepared using this default quality model. As a first step, we show how this model is built up using the quality model preview screen depicted in Figure 4. We overview the general structure of the model, the applied source code metrics, and the weight distributions coming from the expert votes.

b) Show the statistics of the used benchmark: Each model uses a benchmark of other systems as the basis of the qualification. After we present the structure of the quality model itself, we show some statistics of the default benchmark assigned to the default quality model. The statistics page of the benchmark is shown in Figure 10. It lists the contained systems together with some descriptive metrics (summed and averaged), e.g. total lines of code, total number of classes, and total number of methods. Besides the overall values the distribution of these basic metrics is presented on pie-charts.

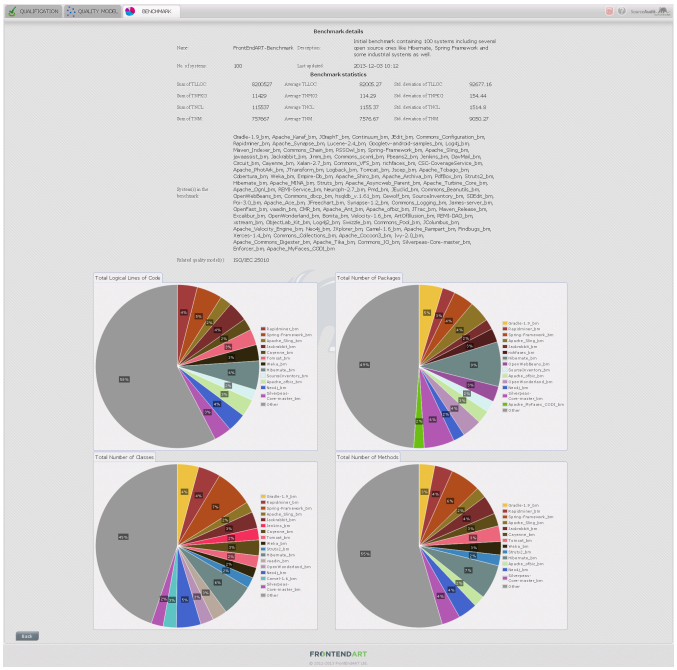

Figure 10. The default benchmark statistics

c) Certification details of the open-source system: As the main part of the demonstration we show the certification history and details of an open-source system (with a large number of analyzed versions). We introduce the information found on the certification details view (see Figure 6):

- The maintainability timeline showing the high-level quality trend of the system.
- The spider charts and timelines representing the values of the low- and high-level quality attributes.
- The trend of maintainability costs of the system (see Figure 11) calculated according to the cost model introduced in Section II.

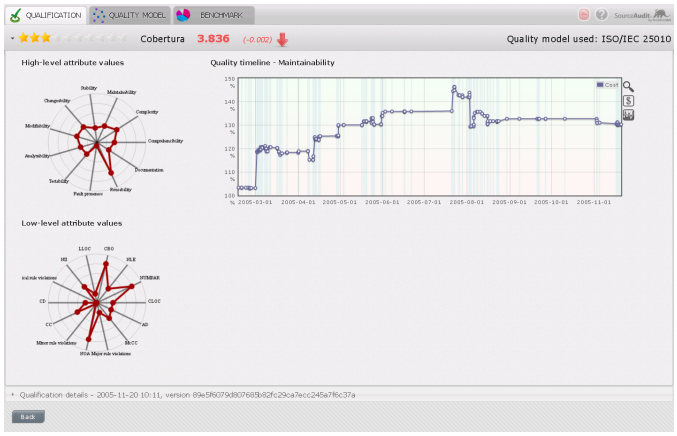

Figure 11. The timeline of the maintainability costs of the system

d) Breaking down to root causes of the changes in a particular version: Selecting a particular version of the analyzed system we drill down to the root causes of the maintainability change. We show the reports that can be used for analyzing the source code level changes in the system as well as the source code view of SourceAudit. The breakdown view of the tool (see Figure 12) helps to identify the new rule violations introduced since the previous version, the changed source code metrics, the most critical source code elements according to our drill-down algorithm, the code clones, etc. We present the different options to get to the source code of the analyzed systems using SourceAudit.

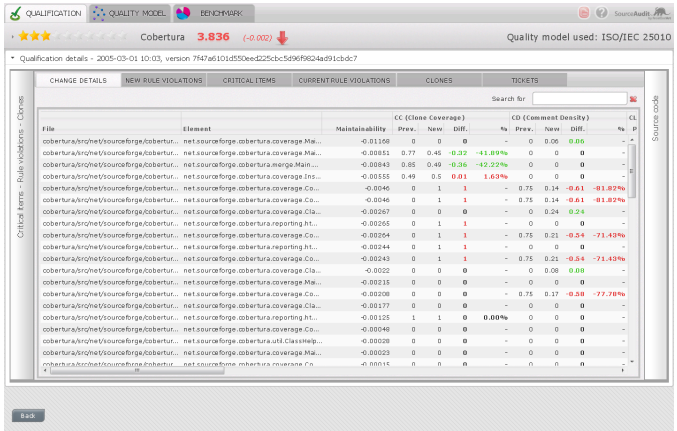

Figure 12. The breakdown table view of SourceAudit

e) Creating tickets and annotations in the source view: The source code view is decorated with the current rule violations and code clones. The view is depicted in Figure 7. We show the feature of SourceAudit with which the user is able to annotate a certain code part as well as to create new

issues (tickets) for the developers. We also create a new issue and assign it to a developer.

f) Finding the issues of a source code element: Not only coding rule violations can be assigned to a developer but it is possible to annotate a whole method or class itself. To help identifying the issues of these source code elements, their metrics are also shown in a table depicted on Figure 13. The metrics in the table are split into several groups e.g., coupling, complexity. The NII (Number of Incoming Invocations) metric for instance helps to determine the importance of a method. If this value is large then an error can be more serious because it affects a large number of other source core parts.

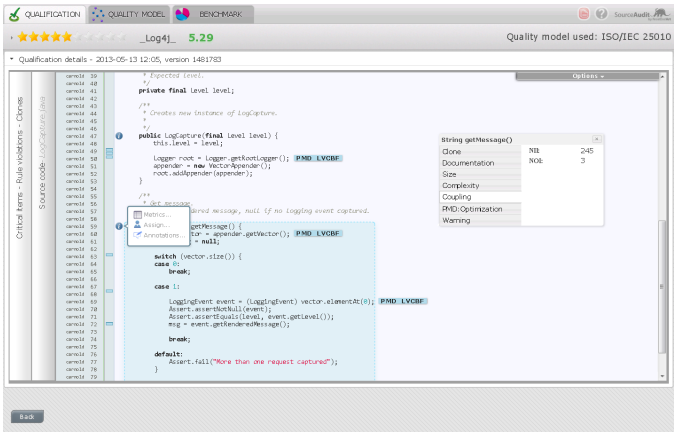

Figure 13. Metrics of a method in the source view.

g) Finding clones with SourceAudit: SourceAudit also provides a table that lists clone instances in the current version of the system (see Figure 14). In the table one can see the most important information about these clone instances e.g., in which clone class they are, or their position in the file.

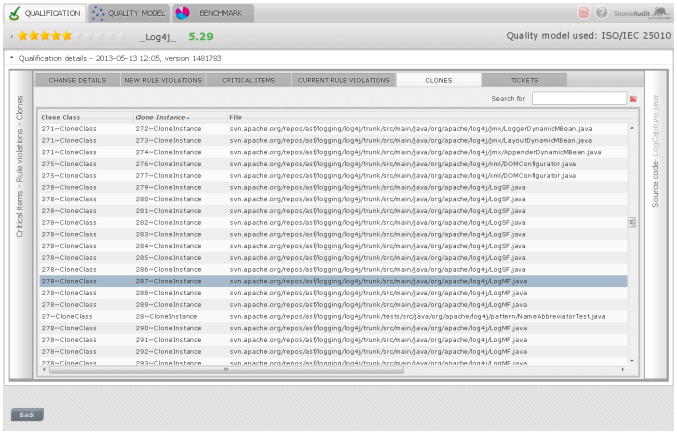

Figure 14. Listing clone instances in SourceAudit.

By selecting one of the instances SourceAudit opens the source view panel highlighting the affected clone as shown in Figure 15. Clones also have different metrics (e.g. risk of the clone) which are displayed in SourceAudit. We present how to find clones and view their source code in SourceAudit.

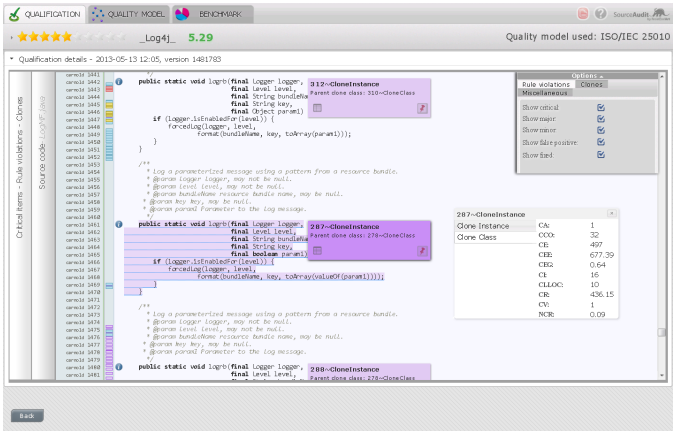

Figure 15. Clone instances in the source view.

h) Loading and fixing tickets in the Eclipse IDE: The created tickets in the SourceAudit tool get uploaded into a central database. We present our Eclipse plug-in which is able to read the information from this central database. The plug-in lists all the tickets assigned to a developer and shows all the annotations and comments made by the project manager and/or other developers (see Figure 8). After navigating to the issue in the source code using the IDE and fixing the bug, the ticket can be closed. We present this scenario and show how this information is propagated back to SourceAudit where the status of the tickets can also be observed (see Figure 16).

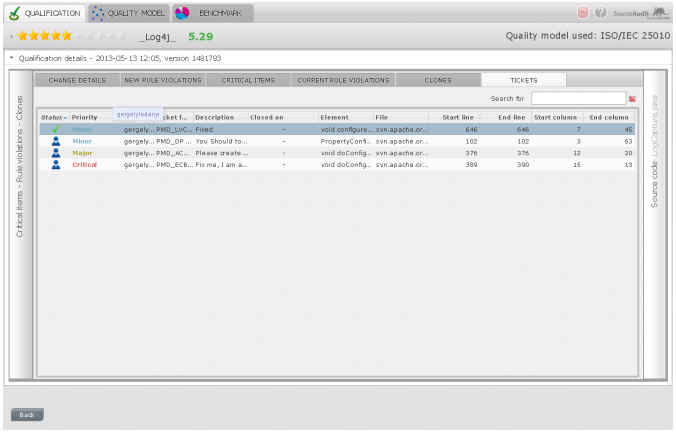

Figure 16. The active tickets in the current version.