# Teamscale: Software Quality Control in Real-Time [*]

Lars Heinemann Benjamin Hummel Daniela Steidl
CQSE GmbH, Garching b. München, Germany
{heinemann,hummel,steidl}@cqse.eu

## ABSTRACT

When large software systems evolve, the quality of source code is essential for successful maintenance. Controlling code quality continuously requires adequate tool support. Current quality analysis tools operate in batch-mode and run up to several hours for large systems, which hampers the integration of quality control into daily development. In this paper, we present the incremental quality analysis tool Teamscale, providing feedback to developers within seconds after a commit and thus enabling real-time software quality control. We evaluated the tool within a development team of a German insurance company. A video demonstrates our tool: `http://www.youtube.com/watch?v=nnuqplu75Cg`.

## Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software quality assurance (SQA)*

## General Terms

Management, Measurement

## Keywords

Quality control, static analysis, incremental, real-time

## 1. INTRODUCTION

In many domains, software evolves over time and is often maintained for decades. Without effective counter measures, the software quality gradually decays [4,10], increasing maintenance costs. To avoid long-term maintenance costs, continuous quality control is necessary and comprises different activities: When developers commit changes, they should be aware of the impact on the system's quality and when introducing new quality defects, they should remove them appropriately. Project and quality managers should check the quality status in regular intervals and take actions to improve quality. Research has proposed metrics to assess software quality, including structural metrics (*e. g.*, nesting depth) or redundancy measurements (*e. g.*, code cloning), leading to a massive development of analysis tools.

However, existing tools have three major disadvantages. First, they require up to several hours of processing time for large systems [1]. Consequently, quality analyses are scheduled to off-work hours (e.g. nightly build) and developers are not notified of problems in their code until the next morning. However, by then, they often already switched context and work on a different task. Second, development teams produce hundreds of commits per day. Nightly analyses aggregate them and make it hard to identify the root causes for unexpected quality changes. Instead, developers must reverse engineer the root causes from the versioning system. Third, analysis techniques reveal thousands of quality problems for a long-lived software, making it infeasible to fix all quality issues at once. With existing tools, developers cannot differentiate between old and new quality defects, which often results in frustration and no improvement at all.

**Problem Statement**. *Existing quality analysis tools have long feedback cycles and cannot differentiate between old and new findings, hampering integration into daily development.*

Teamscale[1] overcomes these drawbacks with incremental quality analyses [1]. By processing each commit individually, it provides real-time feedback and reveals the impact of a single commit on the system's quality. With the full history of the code quality at its disposal, Teamscale supports efficient root cause analysis for specific quality problems. Combining incremental analysis with tracking [11] of quality issues, the tool distinguishes between old and new quality defects, encouraging developers to fix recently added findings.

**Contribution**. *Teamscale provides quality analysis in real-time, integrating into the daily development.*

With a web front-end and an IDE integration, Teamscale offers different perspectives on a system's quality. It provides an overview of all commits and their impacts on quality and a file-based view on source code annotated with quality defects. It also offers an aggregated view on the current quality status with dashboards and enables root cause analysis for specific quality problems.

[*]This work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant "EvoCon, 01IS12034A". The responsibility for this article lies with the authors.

[1]`http://www.teamscale.com`

*ICSE Companion'14*, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2768-8/14/05...$15.00
http://dx.doi.org/10.1145/2591062.2591068

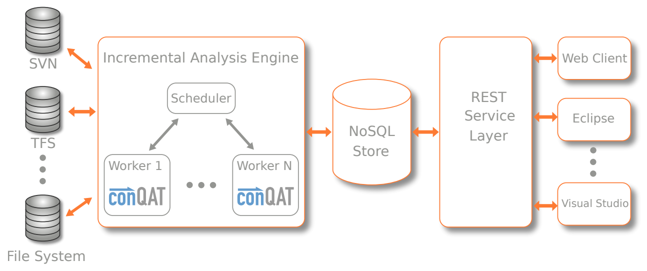

**Figure 1: High-Level Architecture**

## 2. RELATED WORK

In the current state of the art, many quality analysis tools already exist.There are numerous tools, for example, for the analysis of a specific quality metric or defect. While some of them are even incremental [5], Teamscale aims at providing a differentiated, all-encompassing perspective of software quality by using a broad spectrum of metrics and analyses.

**Distributed Quality Analysis Tools**. Shang et al. [12] propose to address scalability by adapting tools which originally were designed for a non-distributed environment and deploying them on a distributed platform such as MapReduce. Using enough computing ressources, their approach can reduce the feedback to developers to minutes. However, their approach focusses on individual analyses and does not provide a comprehensive front-end to the quality analysis data. Additionally, our approach of incremental analysis requires significantly less computing resources.

**Quality Dashboards**. Multiple tools provide the user with a dashboard containing various aggregations and visualizations of a system's source code quality. Tools include ConQAT [3], SonarQube[2], or the Bauhaus Suite[3]. All of these tools provide a custom analysis engine, integrate the results of other existing analysis tools, and provide configurable means of displaying the results. All of them, however, require multiple hours on large systems, making real-time feedback and fine-grained root cause analysis impossible.

**Incremental Quality Analysis Tools**. The conceptually most similar tool to ours is SQUANER [7]. It also proceeds incrementally, and updates are triggered by commits. It also aims to provide rapid developer feedback. However, the calculated metrics are file-based and thus limited to local analyses (see [1] for more details). Unfortunately, the corresponding web site is unreachable, but the paper suggests that the data representation does not support root cause analysis and that no tracking of quality defects is provided.

## 3. ARCHITECTURE

Teamscale is a client/server application: the quality analyses are performed on a central server and different user clients present the results. Figure 1 shows an overview of Teamscale's architecture: The incremental analysis engine in the back-end connects to various different data sources, *e. g.*, version control systems. A REST Service Layer provides the interface of the analysis results for different front-end clients.

**Data Sources**. Teamscale directly connects to different version control system (VCS), such as Subversion, GIT and Microsoft's TFS. It constantly monitors the activity in the VCS and directly fetches its changes. Each commit in the VCS triggers the update of all quality analyses. Consequently, in contrast to batch tools, Teamscale does not need to be integrated with continuous integration tools. Teamscale is able to analyze multi-language projects, supporting all widely used languages, such as Java, C#, C/C++, JavaScript, and ABAP. Teamscale further relates quality analysis results with bug or change request data as it connects to bug trackers such as Jira, Redmine and Bugzilla.

**Incremental Analysis Engine**. In the back-end, the incremental analysis engine [1] is built on top of the tool ConQAT [3]. Instead of analyzing a complete software system in batch-mode, it updates quality metrics and findings with each commit to the VCS. Our study in [1] showed that our incremental analyses perform significantly faster than batch analyses. As commits typically consist of only few files, Teamscale computes their impact on the system's source code quality within seconds, providing rapid feedback for the developer. Monitoring the quality changes based on single commits enables a fine-grained root cause analysis.

**Storage System**. Teamscale has a generic data storage interface supporting various noSQL data stores (such as Apache Cassandra[4]). Incremental storage of the analysis results over the system's history [1] allows to store the full analysis data for every single commit as well as the complete history of every source code file within reasonable space. A medium-sized system with 500 kLOC and a history of 5 years typically requires 2–4 GB of disk space.

**REST Service Layer**. A REST service layer provides an interface for retrieving all stored quality analysis results, supporting a variety of clients on multiple platforms.

**Clients**. Multiple user clients present the quality analysis' results. A JavaScript-based web front-end allows platform independent access to Teamscale. For developers, IDE clients integrate quality analysis results directly in their development environment, such as Eclipse or Visual Studio.

## 4. CODE QUALITY ANALYSES

Teamscale offers many common quality analyses. For the remainder of this paper, we refer to the term *finding* for all quality defects that can be associated with a specific code region. A finding can, *e. g.*, result from a violation of a metric threshold (such as a file which is too long) or be revealed by a specific analysis such as clone detection or bug pattern search. In the following, we will give an overview of the provided analyses:

**Structure Metrics**. As major structural metrics, Teamscale comprises file size, method length, and nesting depth. These metrics are aggregated based on the directory structure and result in findings when thresholds are violated.

**Clone Detection**. Teamscale uses an incremental, index-based clone detection algorithm [8] to reveal code duplication by copy and paste. It can be also configured to use an incremental gapped clone detection to find inconsistent clones which are likely to contain incomplete bug fixes [9].

**Code Anomalies**. Teamscale analyzes many different code anomalies, *e. g.*, naming convention violations, coding guideline violations, and bug patterns: we integrate state-of-the-art bug pattern detection tools such as FindBugs[5],

[2]Formerly called *Sonar*, `http://www.sonarqube.org`
[3]`http://www.axivion.com/products.html`

[4]`http://cassandra.apache.org`
[5]`http://findbugs.sourceforge.net`

PMD[6], and StyleCop[7]. Tools that do not work incrementally (*e. g.*, FindBugs due to non-trivial cross-file analysis) are integrated via a finding import from the nightly build.

**Architecture Conformance.** Architecture conformance assessment compares a high-level architecture specification model against the actual dependencies in the code. Teamscale uses the ConQAT architecture conformance assessment [2] which provides a simple component dependency model. The system is modeled as a hierarchical decomposition into components with dependency policies between them. The assessment result is an annotated version of the component model that rates each dependency as satisfied or violated.

**Source Code Comments.** The code comment analysis verifies if documentation guidelines for the existence of code comments are met, which may for instance prescribe that every public type and public method must have an interface comment. In addition, a machine-learning based algorithm reveals commenting deficits, such as trivial or inconsistent comments which do not promote system understanding [13].

**Test Quality.** From the nightly build, Teamscale uses test coverage data and visualizes which parts of the source code are covered by tests.

## 5. USER CLIENTS

The results of all quality analyses are available in either a web front-end or the IDE.

### 5.1 Web Front End

The web client is organized in *perspectives*, which provide different views of the system's quality: The *code perspective* allows to browse the source code, its quality metrics and findings. The *activity perspective* displays all commits in the system's history, the *finding perspective* presents all current quality findings whereas the *dashboard perspective* offers an aggregated overview of the quality status. The web front-end supports various different use cases (UC):

**UC1: Inspecting quality defects in specific components.** In the code perspective, Teamscale offers a file-system based view on source code, metrics and findings. The user can navigate through the directory structure to find quality defects in specific parts of the system. As all source code and quality analysis results are held in Teamscale for the entire history, the code perspective also functions as a *time machine* to browse previous versions of the code and its findings. On file level, the source code is annotated with findings indicated with colored bars and tool tips next to the code. For each directory level and quality metric, Teamscale provides trend charts showing the evolution of the metric for the selected directory, such as the growth in lines of code or the evolution of the clone coverage.

**UC2: Inspecting the quality impact of the last commit.** The activity perspective shows all commits. Besides the information on the change itself (*i. e.*, revision number, author, message, affected files), Teamscale reveals the impact on the quality status for each change: It indicates how many quality problems were introduced or removed with this change. In addition, the bug tracker integration relates the commits to change requests or bugs.

**UC3: Comparison of two versions of the system.** Teamscale can compute the *change delta* between two different versions of the system, which shows the changes within the time frame between the two versions: The change delta contains information about added, deleted, or changed files, the version control system commits within the time frame, metric value changes, as well as added or removed findings.

[6]http://pmd.sourceforge.net
[7]https://stylecop.codeplex.com

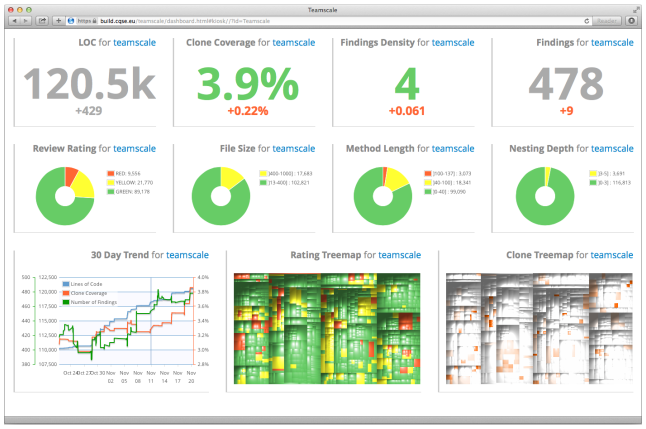

Figure 2: Dashboard Perspective

**UC4: Assessment of architecture conformance.** Teamscale provides a visualization of the architecture specification annotated with its assessment according to the current dependencies in the code. In case a dependency policy is violated, it allows to identify the problematic dependency in the code, i.e. navigate from the model down to individual files that contain the architecture violation.

**UC5: Identification and management of findings.** In the findings perspective, the user can browse quality defects along two dimensions—the directory hierarchy and the finding categories. Teamscale is capable of tracking these findings during the software evolution, in which the location of a finding can change: the finding may move within a file or even across files due to edit operations, moving code snippets, or file renames. Teamscale can track this, using a combination of ideas from [6] and [11]. Based on findings tracking, individual findings, such as false positives, can be *blacklisted*. A blacklisted finding will not be further reported. Teamscale also offers the opportunity to only inspect findings that were added since a configurable revision, *e. g.*, the last release. This way, when developers clean up code, they can focus on newly added findings.

**UC6: The system's quality at a glance.** The dashboard perspective gives an at-a-glance view on the overall quality status. The users can freely build personal dashboards based on a variety of configurable widgets. With his personal dashboard, the user can focus on specific quality aspects that are relevant for him. Teamscale provides widgets for displaying metric values with delta indicators, metric distributions, tree maps, trend charts, and quality deficit hotspots. Figure 2 shows an example dashboard.

### 5.2 IDE Integration

Teamscale provides plug-ins for two widely used IDEs: Eclipse and Visual Studio. The IDE clients annotate the code with findings from the Teamscale server, making developers aware of existing quality issues in their code.

## 6. QUALITY CONTROL SUPPORT

Teamscale is designed to be used in a continuous quality control process during daily development. Continuous control requires constant monitoring of the quality, a transparent communication about the current status as well as specific tasks on how to improve the quality. Teamscale supports various usage scenarios for different stakeholders in this process: The project-manager uses the dashboard to check the quality status of his system in regular intervals. For each commit, a developer receives feedback about findings that were introduced or removed with this commit. A quality manager can monitor the evolution of the quality, detect the root-cause for quality problems and create specific tasks to improve the system's quality.

## 7. EVALUATION

We evaluated our tool with a development team from a German insurance company, LV 1871 which comprises about ten developers, among which six took part in the evaluation. The developers had an average of 18.5 years of programming experience and are hence considered suitable for evaluation.

**Set Up**. We first gave the developers a short introduction to the tool. Afterwards, the developers were asked to solve specific tasks with Teamscale and to fill out an anonymous online survey about their user experience. The tasks comprised nine questions related to the quality of the team's software and were guided by the use cases presented in Section 5.1, except of UC4 (as no architecture specification was available). Developers were also asked to commit live to the tool and evaluate the feedback cycle. Solving the tasks ensured that the developers were familiar with the tool before taking part in the survey and showed whether developers succeeded to use the tool to find the required information.

**Results**. Of all 9 tasks, all developers were able to complete 7 tasks correctly. Two tasks regarding the dashboard perspective were not solved correctly by all, two developers each made a mistake. Problems with those two tasks were related to a lacking piece of information about a specific dashboard widget. In the survey, developers rated several statements about Teamscale on a Likert scale from 1 (I totally agree) to 6 (I do not agree at all). Table 1 shows the average developer response per statement ($\mu$) and the standard deviation ($\sigma$). The results show a very positive user experience. All developers agreed that they want to use Teamscale for their own development ($\mu = 1.2$) and that Teamscale provides fast feedback in practice ($\mu = 2$). They also agreed that they were able to differentiate between new and old quality problems ($\mu = 1.5$) and that Teamscale enables root-cause analysis ($\mu = 1.67$). To summarize, the developers showed that they were able to solve specific tasks successfully using the tool and that they would like to be supported by the tool in their daily development.

**Table 1: Average Developers' Agreement**

| Statement | $\mu$ | $\sigma$ |
|---|---|---|
| Teamscale provides a good overview of the quality of my software. | 1.5 | 0.5 |
| Teamscale helps to identify the root cause of quality problems. | 1.67 | 0.47 |
| Teamscale supports me in a quality-driven software development | 1.67 | 0.47 |
| Teamscale provides feedback fast enough in practice. | 2 | 0.57 |
| Teamscale makes it easy to differentiate between new and old quality problems. | 1.5 | 0.5 |
| Teamscale's UI is intuitive. | 2 | 0 |
| I'd like to use Teamscale for my own development. | 1.2 | 0.37 |

## 8. CONCLUSION

In this paper, we presented a comprehensive quality analysis tool which performs in real-time. Giving feedback about the impact of a change on the quality of a system within seconds, the tool smoothly integrates into the daily development process, enabling successful continuous quality control in practice. Providing root cause analysis, developers can identify the cause of a specific quality problem and address it appropriately. With the help of findings tracking, developers can decide which findings should be reported, helping them to manage and remove findings such that quality actually improves. With the tool, developers, quality managers, and project managers get their own perspective on the system's quality, adapted to the specific needs of each role. A survey among developers showed a high user acceptance.

## Acknowledgements

We thank our evaluation partner LV 1871 for their support.

## 9. REFERENCES

[1] V. Bauer, L. Heinemann, B. Hummel, E. Juergens, and M. Conradt. A framework for incremental quality analysis of large software systems. In *ICSM'12*, 2012.

[2] F. Deissenboeck, L. Heinemann, B. Hummel, and E. Juergens. Flexible architecture conformance assessment with ConQAT. In *ICSE'10*, 2010.

[3] F. Deissenboeck, E. Juergens, B. Hummel, S. Wagner, B. M. y Parareda, and M. Pizka. Tool support for continuous quality control. *IEEE Softw.*, 2008.

[4] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus. Does code decay? Assessing the evidence from change management data. *IEEE Softw.*, 2001.

[5] N. Göde and R. Koschke. Incremental clone detection. In *CSMR'09*, 2009.

[6] N. Göde and R. Koschke. Frequency and risks of changes to clones. In *ICSE '11*, 2011.

[7] N. Haderer, F. Khomh, and G. Antoniol. SQUANER: A framework for monitoring the quality of software systems. In *ICSM'10*, 2010.

[8] B. Hummel, E. Juergens, L. Heinemann, and M. Conradt. Index-based code clone detection: Incremental, distributed, scalable. In *ICSM'10*, 2010.

[9] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *ICSE '09*, 2009.

[10] D. L. Parnas. Software aging. In *ICSE '94*, 1994.

[11] S. P. Reiss. Tracking source locations. In *ICSE '08*, 2008.

[12] W. Shang, B. Adams, and A. E. Hassan. An experience report on scaling tools for mining software repositories using mapreduce. In *ASE'10*, 2010.

[13] D. Steidl, B. Hummel, and E. Juergens. Quality analysis of source code comments. In *ICPC'13*, 2013.