Version 5.2



Teamscale

The Official User Reference



CQSE

Disclaimer

CQSE does not warrant that *Teamscale* will meet your requirements, that *Teamscale* will operate in combinations with equipment, devices, software or systems provided by persons other than CQSE, that the operation of *Teamscale* will be error-free or uninterrupted, or that *Teamscale* will discover all errors and vulnerabilities that may reside in the code base. Errors may be caused by misconfiguration, limitations of the applied analysis techniques, or bugs in their implementation. To ensure correct results, both *Teamscale*'s configuration and results obtained from *Teamscale* must be manually inspected and verified.

Copyright 2019 CQSE GmbH

Published by CQSE GmbH

http://www.teamscale.com/

The style of this book is based on http://www.latextemplates.com/template/the-legrand-orange-book. The original layout has been modified.

ABAP®, SAP®, and SAP NetWeaver® are trademarks or registered trademarks of SAP SE in Germany and in several other countries.

Image Copyright D. Steidl (CQSE GmbH)

Version 5.2 – July 2019



Contents

1 Introduction 13
1.1 Software Intelligence Vision 14
1.2 How (Not) To Control Quality 16
1.2.1 Why Many Fail To Improve 17
1.2.2 Improving Code Quality 18
1.2.3 Ensuring Test Quality 20
1.2.4 Managing Architecture Quality 23
1.2.5 Analyzing Feature Usage 24
1.3 Architectural Overview 25
1.4 Target Audience and Outline 26

1.5 Further Resources 27

2 Quality Analyses 29
2.1 Supported Technologies and Infrastructure 30
2.2 Internal Analyses 30
2.3 Integrated External Analyses 31

3 Common Use Cases 35
3.1 Configuring Projects and Analyses 36
3.1.1 Best Practices for Your First *Teamscale* Project 36
3.1.2 Creating an Analysis Profile 38
3.1.3 Creating a Project Configuration 43
3.2 Getting Personalized Developer Feedback 49
3.2.1 Inspecting the Impact of a Single Change 49
3.2.2 Being Informed Without Leaving Your IDE 52
3.2.3 Push Notifications: Receiving Emails about Changes 52
3.3 Getting an Overview of the Quality Status 53
3.3.1 Creating a Single Project Overview 54
3.3.2 Creating a Multi Project Overview 57
3.4 Monitoring and Assessing Quality Evolution 58
3.4.1 Inspecting Recent Metric Trends 59
3.4.2 Finding the Root Cause for a Trend 61
3.4.3 Analyzing the Findings Churn 64
3.4.4 Performing a Baseline-Delta-Analysis 67
3.5 Identifying Untested Changes 70
3.5.1 Creating a *Test Gap analysis* Dashboard 71
3.5.2 Working with Test Gap Treemaps in Practice 76
3.6 Improving test execution efficiency 77
3.6.1 Determining tests covering a method 79
3.6.2 Viewing detailed test information 80

3.6.3 Determining impacted tests over large time intervals 80

4 User Reference 83
4.1 Web Interface 84
4.1.1 General UI Features 85
4.1.2 Dashboard Perspective 88
4.1.3 Activity Perspective 98
4.1.4 Findings Perspective 104
4.1.5 Metrics Perspective 113
4.1.6 Tests Perspective 126
4.1.7 Issue Perspective 136
4.1.8 Task Perspective 145
4.1.9 Architecture Perspective 151
4.1.10 Delta Perspective 167
4.1.11 Project Perspective 177
4.2 IDE Integration 206
4.2.1 Eclipse 207
4.2.2 Visual Studio 210
4.2.3 IntelliJ IDEA 213
4.3 Integration and Extension APIs 218
4.3.1 REST API 219
4.3.2 Post-Commit Hooks 221
4.3.3 External Analysis Results Upload (REST) 226
4.3.4 External Analysis Tools 228
4.3.5 Test Gap Analysis Integration 239
4.3.6 Test Impact Analysis Integration 242
4.3.7 Custom Check API 245
4.3.8 Python API 251

5 Administration 253

5.1 System Requirements 254
5.1.1 Analysis Server 254
5.1.2 Clients 255
5.2 Installation 255
5.2.1 Configuring *Teamscale* 256
5.2.2 Providing a License 256
5.2.3 Configuring Encryption 257
5.2.4 Starting *Teamscale* 257
5.2.5 Installing *Teamscale* As a Service 259
5.2.6 Configuring HTTPS 263
5.2.7 Integrating With Apache Webserver 265
5.2.8 Integrating With nginx 266
5.2.9 Configuring HTTP(S) Proxy 267
5.2.10 Using Certificates from the Windows Certificate store 267
5.2.11 Connecting To External Servers Via HTTPS 268
5.3 Permission Management 269
5.3.1 Users, Groups & Roles 269
5.3.2 Managing Access to a Project 272
5.3.3 Managing Access to an Analysis Profile 275
5.3.4 Managing Access to Global *Teamscale* Features 277
5.3.5 Managing Roles 279
5.3.6 LDAP Server Integration 282
5.3.7 Single Sign-On using SAML 2.0 283
5.3.8 Automatic URL Log-In 284
5.4 Settings 284
5.4.1 Teamscale instance base URL 285
5.4.2 Teamscale Working Directory 285
5.4.3 Server Limits 285
5.4.4 Git 286
5.4.5 GitHub Integration 287
5.4.6 Bitbucket Cloud Integration 290

5.4.7 Bitbucket Server (On-Premise) Integration 291
5.4.8 GitLab Integration 292
5.4.9 Usage Data Reporting 293
5.5 Backup 293
5.5.1 Exporting backups 294
5.5.2 Importing backups 294
5.6 Monitoring 295
5.6.1 Nagios 296
5.6.2 Prometheus 296
5.7 Updates 296
5.7.1 Minor Version Upgrades 297
5.7.2 Major Version Upgrades 298
5.8 Support 300

6 SAP® Integration 311
6.1 *Teamscale Connector for SAP NetWeaver® AS ABAP®* 312
6.1.1 Installation of the Connector 313
6.1.2 User and Role for RFC Access 314
6.2 SAP Connection Configuration in *Teamscale* 316
6.2.1 Configuring Working Directory 316
6.2.2 Configuring SAP ABAP System Connection 318
6.2.3 Configuring Account Credentials 322
6.2.4 Configuring a *Teamscale* Project 323
6.3 Asynchronous Full Export 327
6.3.1 *bgRFC* Configuration in the SAP System 328
6.3.2 Asynchronous Export Configuration in Teamscale 329
6.4 Secure RFC Communication (optional) 330
6.4.1 SNC Configuration of the SAP System 330
6.4.2 SNC Configuration on the *Teamscale* Server 333
6.4.3 SNC Configuration in *Teamscale* 333

6.5 Additional Analysis Configuration 334
6.5.1 Integrating SAP Code Inspector 334
6.5.2 Test Gap Analysis Set Up 338
6.6 Integration of *Teamscale* in SAP Development Environment 343
6.6.1 ABAP Development Tools for Eclipse 344
6.6.2 SAP Code Inspector and ABAP Workbench Integration . . . 344
6.6.3 Integration with SAP Transport Management System 348

Appendix 351

A Dashboard Widgets 353
A.1 Common Widget Configuration Options 354
A.2 Assessment Pie Chart Widget 356
A.3 Assessment Trend Chart Widget 357
A.4 Code City Widget 358
A.5 Commit Chart Widget 359
A.6 External Content Widget 360
A.7 Findings Churn Widget 361
A.8 Findings Summary Bar Chart Widget 362
A.9 Findings Summary Table Widget 363
A.10 Folder Distribution Pie Chart Widget 364
A.11 Issue Table Widget 365
A.12 Label Widget 366
A.13 Metric Bar Chart Widget 367
A.14 Metric Change Table Widget 368
A.15 Metric Distribution Pie Chart Widget 369
A.16 Metric Distribution Table Widget 371
A.17 Metric File Distribution Widget 372
A.18 Metric Hotspot Table Widget 373
A.19 Metrics Table Widget 374
A.20 Metrics Trend Chart Widget 376

A.21 Numeric Metric Value Widget 378
A.22 Ownership Treemap Widget 379
A.23 Single Metric Trend Widget 380
A.24 Task Summary Pie Chart Widget 381
A.25 Test Gap Overview Widget 382
A.26 Test Gap Treemap Widget 383
A.27 Test Gap Trend Chart Widget 384
A.28 Treemap Widget 385
B Repository Connector Options 387
B.1 Subversion (SVN) branch analysis 391
B.2 Team Foundation Server Credentials 391
C Issue Tracker Connector Options 395
C.1 Team Foundation Server Work Items 396
D Entity Selection Predicates 397
E Custom Checks 401
E.1 Custom Checks for ABAP 401
E.2 Custom Checks for C# 409
E.3 Custom Checks for C/C++ 413
E.4 Custom Checks for IEC 61131-3 ST 415
E.5 Custom Checks for Java 415
E.6 Custom Checks for JavaScript 424
E.7 Custom Checks for PL/SQL 425
E.8 Custom Checks for Python 426
E.9 Custom Checks for Xtend 427

Glossary 428

Bibliography 439
Books 439
Online 441



1. Introduction

Teamscale is your platform to ensure high-quality software. It is based on a revolutionary approach to static code analysis: *Teamscale* is a platform not only to collect some metrics, but to combine them with essential data from your development, to aggregate them and to provide solid decision criteria for the fundamental questions during your software development. *Teamscale* addresses your code quality, but also your test quality, your architecture and your usage scenarios. It, thus, supports you throughout the entire life-cycle of your software development.

1.1 Software Intelligence Vision

Ultimately, any software strives to satisfy their users to maximize profit. However, user needs evolve – they change, they expand. Additionally, not only the market situation changes, but also the underlying technology and hardware evolves. Developing software in such agile environments is a challenge: Being able to keep up is not trivial. And it heavily depends on the quality of the software.

To lead a software project to long-term success, software engineers are facing various decisions. From choosing the right underlying hardware and technology to prioritizing which features to implement, decisions have to be made daily. While facilitating continuous software change, managers and developers also have to consider the following: How long will our code base still be maintainable? Is our software sufficiently tested? Does our architecture satisfy current and future requirements? Those decisions are fundamental and have potentially long-term impact on the success (or failure) of the project.

For all these questions, intuition and often prominent gut-feelings are bad decision criteria. Instead, decisions should much rather be made based on solid data from the specific project. Yet, even though the data is technically available, it often remains unused because the data is implicit, hard to extract, hard to visualize or not trivial to interpret. For a profound decision making in software engineering, *Software Intelligence* is required: Just as business intelligence provides a "set of techniques and tools for the transformation of raw data into meaningful and useful information for business analysis purposes" [Wik], software intelligence targets the same purpose: software intelligence uses data mining to provide a solid foundation for decision making in software development.

Today, *Teamscale* is the only Software Intelligence Platform on the market. With *Teamscale*, you can steer the success of your project based on solid and profound data from your software: Integrating various sources such as code repositories, architecture documentations, test executions, issue trackers and runtime execution in production, *Teamscale* provides you transparent answers to the following questions:

- **Code Quality**: Is my code currently more spaghetti-code than clean code?
- **Test Quality**: Did we actually test what we changed or did we miss something?
- **Architecture Quality**: Does my code still match my architecture? Is the architecture suitable?
- **Feature Management**: Is all of my code in fact used by the user?

Being able to control these aspects makes your project more likely to succeed – and also more fun to work in. *Teamscale* will signal red flags early enough when tempting feature pressure deteriorates your software quality. Small quality deviations can be often redeemed quickly when developers discover them early. But at some point, consequences of failing to do so become unbearable. This is not something we just make up. We all have seen software that needed to be rewritten. Also research has shown that software quality – wrt. to code, test, and architecture – gradually decays over time if no effective counter measures are taken [Ede+13; Eic+01; FRJ09; Par94; Ste+14]. So do something before it is too late! And don't waste money for maintaining code that is not even executed by the user. Even though it seems as this code just sits there at no further cost,

do not underestimate the effort which this unused code creates for maintenance [Ede+12].

To make your project a success, *Teamscale* offers a broad range of prospects. It adequately addresses your needs and provides appropriate views of your software's quality regardless of whether you are a developer, a tester, an architect, a product owner or a manager. It adapts flexibly to your chosen development process and also supports feature-branch driven development. In contrast to many existing tools, it does not only analyze the main development trunk, but can be tailored to every single branch.

Further, *Teamscale* can handle common programming languages such as Java, Javascript, C or C++, but also knows how to deal with a bit more exotic languages such as ABAP, Ada, Delphi, Python or PL/SQL – just to mention a few. It also integrates with most common infrastructures such as Subversion, Git or TFS on the version control side or Jira, TFS and RTC/Jazz on the issue tracking side. For an up-to-date and complete list of *Teamscale*'s capabilities, please refer to our website www.teamscale.com.

1.2 How (Not) To Control Quality

Every developer knows the feeling when he looks at code and stunningly asks himself: "Who the hell wrote this code?" Or when he is getting desperate because he wonders: "Yesterday it still worked, didn't it"? It is a common phenomenon in software engineering that lots of code is written under time pressure and, hence, turns into quick 'n dirty written and badly tested code. Code that is very hard to understand and to maintain. In this section, we show you how to get in control of deteriorating quality and what helps to actually improve (as well was what does not).

1.2.1 Why Many Fail To Improve

At some point, many companies realize when gradually decreasing quality becomes an issue. In an attempt to prevent this decay, they install common static analysis tools – tools, which reveal, for example, bad code smells or measure test coverage. However, this is not enough. It is like signing up for the nearest gym and never going there to exercise. Static analyses are very useful to reveal problems in your code. Nevertheless, it is challenging to apply them successfully and to actually improve.

For long-lived software, static analysis tools commonly reveal thousands of problems [SE14]. With respect to code quality, these comprise, for example, badly structured pieces of code or redundant implementations. With respect to test quality, these might be files with very little test coverage. Often, developers look at these findings once or twice, try to fix a few and, ultimately, give up. The number of problems in grown software systems is usually too large to be fixed at once because quality assurance budget is almost always limited. (Dedicated large *clean-up* sessions would not be a good idea anyway because introducing more bugs is far too likely.)

Many existing static analysis tools operate in batch mode, i. e. they often only run once during a (nightly) build. When a developer performs a change, they do not provide feedback till the next build which often takes quite a while. This feedback cycle is simply too long. Who wants to go back to a previous task that seemed already finished? In addition, many existing analysis tools provide feedback only on the *entire* system. However, the developer needs personal feedback. In the moment, he cares only about the code he actually changed himself. Often, instead, he is flooded with all findings from the system and has no easy means to filter the ones relevant for him.

Improving software quality requires more than just measuring some metrics. It requires software intelligence. In the following, we show how to use *Teamscale* to improve code quality (Section 1.2.2), how to ensure test quality (Section 1.2.3) and how to keep your architecture quality (Section 1.2.4). Section 1.2.5 further indicates how *Teamscale* can be used for feature profiling and usage data analysis.

1.2.2 Improving Code Quality

When measuring code quality, there are many metrics out there pretending to provide you some insights. Many of them are useless. Choosing appropriate metrics for your domain requires quite some expertise, see Chapter 2. The default configuration of *Teamscale* offers you commonly used and actionable metrics with respect to the structure of your code (file sizes, method lengths and nesting depths), redundancy caused by copy&paste programming, documentation in term of source code comments, violations of coding conventions and bug patterns derived from data flow analysis. But it can be also easily extended using custom checks specifically tailored for your needs (see Section 4.3.7 and Appendix E).

After choosing appropriate metrics, improving code quality requires a realistic goal, a quality goal. Otherwise, your developers are more frustrated than before. Depending on your project's history, the goal can vary. If you just started to develop a new system on the greenfield, the overall goal of *zero findings* might still be feasible. However, for grown-software, we strongly recommend to accept the current quality status as it is and just start from there. Either choose the goal to keep the status quo and at least not make it worse (i. e. not introducing *new findings*). Or take it one step further and try to improve code that you are working on anyway (i. e. remove findings in code that is currently modified due to a feature change request).

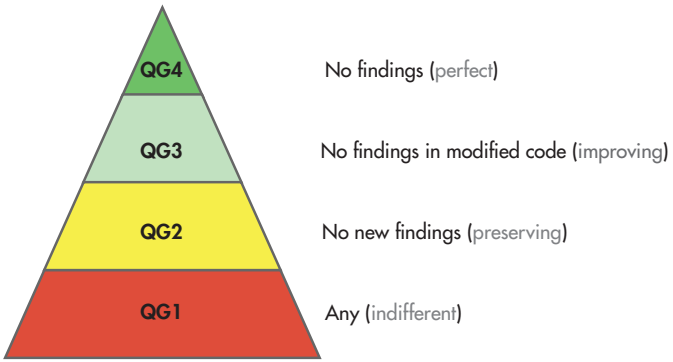


Figure 1.1. Quality Goals

Teamscale helps you to reinforce your quality goal. As many existing tools operate in batch mode, they analyze the entire system at once. While this is lengthy in time, it also omits the ability to provide accurate information which finding is *new* or in *modified* code. *Teamscale*, however, analyzes every single commit to your repository [HHS14]. Hence, it knows exactly when a specific finding was newly introduced to the system. Consequently, *Teamscale* can directly reveal violations of your quality goal, for example, if a developer just created a new finding or if a developer missed the chance to remove a finding in code you touched recently.

Most importantly, developers do not have to wait a day but can fix the problems immediately while they are still working on the subject: *Teamscale* provides the developer feedback within seconds after a commit. It specifically informs the developer about the impact of his change with respect to quality. Even personal notification services can be configured. Personal feedback is crucial to reduce the information flood and to focus on the essentials. It helps to focus only on new problems regardless of the past and to visualize small first steps of success.

In almost all software projects, feature pressure and time constraints are dominating every-day decisions. As a response, many teams have created the role of a *quality engineer* to ensure that quality improvements do not fall off the table. *Teamscale* also addresses the quality engineer's need to monitor the quality trend, to understand outliers of a specific metric and to report to management. Improving quality without management support is like Icarus striving for the sun. Hence, when management is providing resources, creating transparency about its usage is beneficial for both sides. Often, a clearly defined quality control process as exemplary depicted in Figure 1.2 increases satisfaction of all stakeholders and leads to long-term success [Ste+14].

In this process, software intelligence creates transparency and positive quality-awareness. It supports the developer in his daily work and makes the project manager aware of the current quality status and the usage of provided resources. Based on this level of information, he can steer his project more confidently without risking that feature pressure imposes uncontrollable impact on the code quality.

1.2.3 Ensuring Test Quality

In contrast to code quality, test quality is often directly visible to the user – namely, if the software does not work as it should. For long-lived software, most bugs are introduced by changes to existing code or by writing new code. Hence, test managers focus on testing what has changed recently. Nevertheless, untested changes are often released, causing field bugs and potential hot-fixes. Our study has shown that approximately half of the changes are not tested once before being released [Ede+13]. Why is that?

Software errors can have very different root causes. That's why there is usually no single test level, but a pyramid of tests ranging from low-

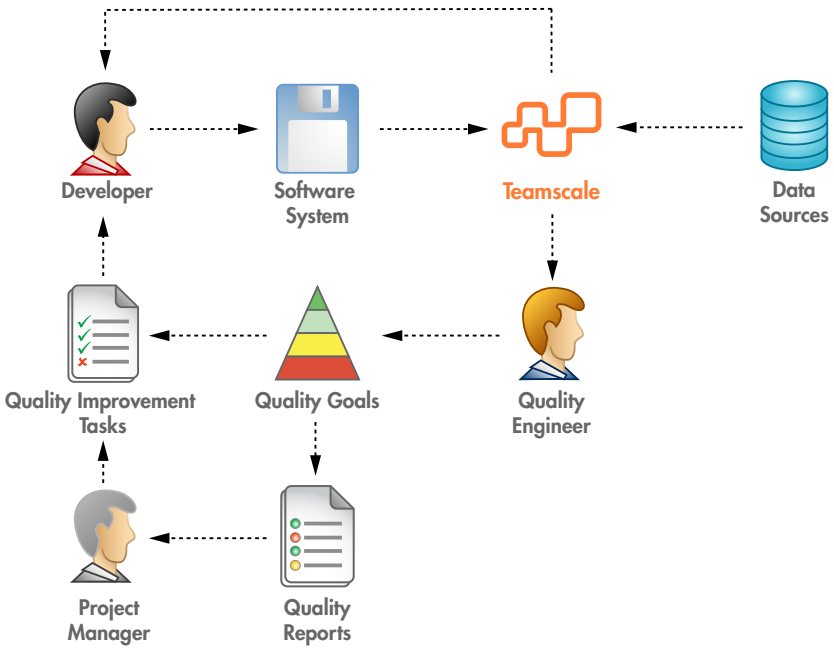


Figure 1.2. Code Quality Control

level unit tests to integration tests and high-level system tests. Each type of test targets to find different types of bugs. Unit tests are often written by the developers themselves. They usually know what changed and what unit tests cover. But testers responsible for integration or system testing often do not know if their test runs cover the actual code changes: Testers are usually driven by change requests resulting from adapting requirements. But the information which code regions were in fact affected by a designated change request is usually not available for them.

Teamscale closes this gap by combining information that is already there: Being connected to the version control system, *Teamscale* knows exactly which parts of the code changed since the last test run. As it also integrates with test coverage tools, *Teamscale* also knows which code was executed during the last test. Combining both sources,

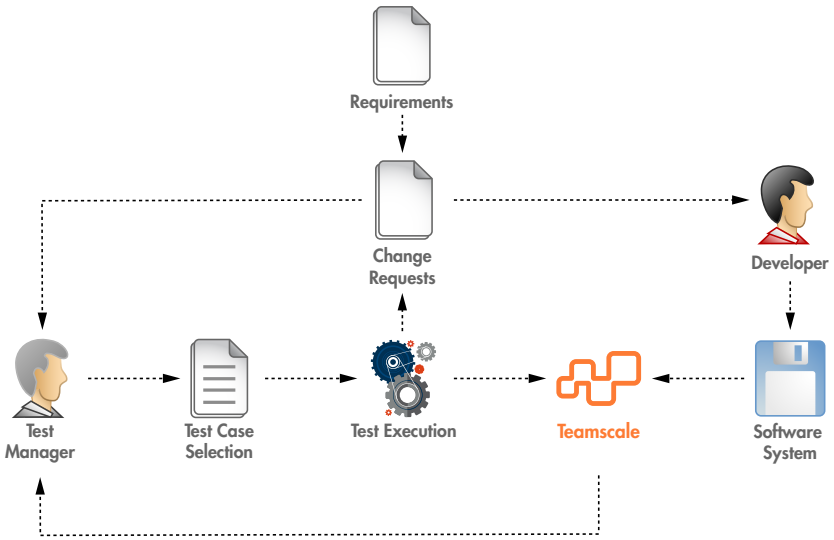


Figure 1.3. Test Quality Control

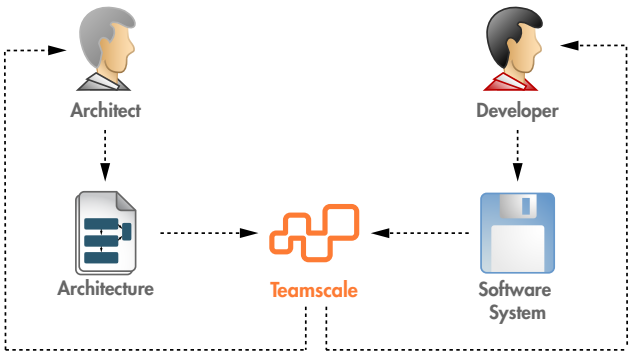


Figure 1.4. Managing Architecture Quality

Teamscale reveals recent changes that have not been tested so far and can provide feedback to the test manager (see Figure 1.3). Based on these existing gaps in the test execution, the test manager can decide which test cases to select for the next test run.

This form of software intelligence combines information from the development and testing process that is already there. But many companies have not been able to combine the information due to missing tool support. With *Teamscale*, the information can be easily combined.

1.2.4 Managing Architecture Quality

In addition to code and test quality, *Teamscale* also helps to manage your architecture quality. Many teams have at least an informal understanding what the architecture of their system – in terms of component diagrams – roughly looks like. But only few have it documented and even fewer have a conforming code base.

Teamscale helps you first to model the architecture. With an easy graphical user interface, you can specify components and wanted dependencies. In a second step, *Teamscale* continuously checks whether your current code base conforms to your specifications. It will notify the developer of unwanted dependencies immediately after a commit. After inspection, it is then up to the developer to clean up the code (if the dependency is, in fact, unwanted) or to adapt your specification (if you can live with the new dependency), see Figure 1.4. *Teamscale* supports you to avoid a commonly seen loss of architecture knowledge [FRJ09] and to keep your implementation conform to the intended architecture.

1.2.5 Analyzing Feature Usage

During any quality assurance process, managers and developers prefer focusing on those parts of the system that users often use. Nobody wants to waste money on code that is not even executed by the user. Even though this code has already been written, it still creates overhead costs for maintenance that should not be underestimated [Ede+12].

On the one hand, *Teamscale* integrates the information from a usage profiler in the system's production environment. On the other hand, it analyzes the development process. Combining both information sources provides valuable input for managing the costs of software maintenance (see Figure 1.5): It supports developers and managers in their decision to delete code that is never executed. It also helps them to make sure only those features are extended that are in fact used.

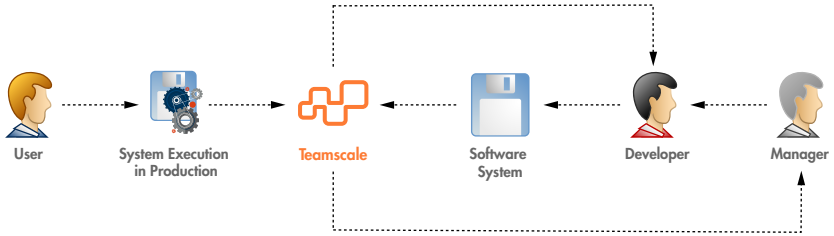


Figure 1.5. Analysis of Usage Data

1.3 Architectural Overview

In this section, we provide a rough overview of *Teamscale*'s internal architecture, see Figure 1.6. *Teamscale*'s core is the incremental analysis engine. It is directly connected to the version control system and, hence, analyzes each commit *incrementally*. This enables *Teamscale* to provide rapid feedback and reveal the root causes on commit-based for emerging problems or deteriorating trends.

With its analysis engine, *Teamscale* provides its own set of code quality analyses, such as clone detection, structural metrics or comment analyses for example (see Chapter 2). It also has its own architecture analysis. In addition, the analysis engine integrates various kinds of data from your software development – from test coverage over usage data to external static code analysis tools that you might wish to use specifically. It also connects to your bug and issue tracking system and can, thus, provide links between change requests, commits and changes to the system's quality status. Integrating all these data, *Teamscale* helps to manage your code, test and architecture quality as well as monitor your feature usages (see Section 1.2). In other words, *Teamscale* becomes the core platform for software intelligence.

The results of the analysis engine are kept in a NoSQL store and made accessible with a REST Service API. The web client, for example, uses

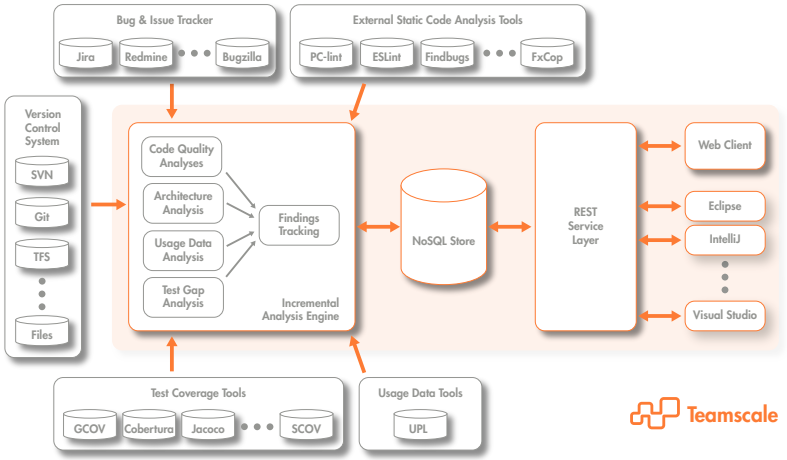


Figure 1.6. *Teamscale* Architecture

this API for the user interface in the browser. In addition, *Teamscale* also provides several IDE clients, for example, plugins for Eclipse, IntelliJ and Visual Studio.

1.4 Target Audience and Outline

This document is targeted at developers, quality engineers and project managers who want to use *Teamscale* in their development projects as well as at administrators responsible for the installation and configuration of a *Teamscale* server instance.

The remainder of this document is structured as follows. Chapter 2 introduces the quality analyses provided by *Teamscale*. This chapter is suitable if you want to find out which metrics and findings are computed by *Teamscale*. Chapter 3 describes common use cases for *Teamscale*. This chapter is ideal if you want to learn about the capa-

bilities of *Teamscale* and find out how to achieve a specific goal. In contrast, Chapter 4 provides a comprehensive overview of the features of *Teamscale*. This chapter for instance helps you to understand how specific parts of *Teamscale* work in detail. Chapter 5 contains an administration guide for installing and configuring a *Teamscale* server instance. Chapter 6 describes the configuration to integrate *Teamscale* with SAP® systems. This is especially relevant if you want to analyze ABAP® code. At the end, the glossary explains the underlying terminology used throughout this book.

1.5 Further Resources

Besides this user guide, the *Teamscale* Knowledge Base[1] provides FAQs, Getting Started Guides and solutions to recurring problems and common pitfalls. In case your question is not answered anywhere, please don't hesitate to contact the *Teamscale* support (https://www.cqse.eu/en/products/teamscale/support/).

[1] https://cqse.freshdesk.com/solution/categories



2. Quality Analyses

As a Software Intelligence Platform, *Teamscale* monitors your software's quality and combines it with essential data from your development. It supports various technologies and provides many different quality analyses. This chapter gives you an overview of the supported programming languages and development infrastructures (Section 2.1), *Teamscale*'s own analysis engine (Section 2.2) as well integrated external analysis tools (Section 2.3). In addition to the integrated analyses, *Teamscale* comes with a framework for simple, user-defined analyses based on token streams or simple parse trees. This »custom-check« framework is documented in Section 4.3.7 (Appendix E contains a list of custom checks delivered with *Teamscale*).

2.1 Supported Technologies and Infrastructure

Table 2.1 shows the list of supported programming languages and development Infrastructure tools.

Table 2.1. Supported Technologies and Infrastructure

Programming Languages	
Languages	ABAP, Ada, C#, C/C++, Cobol, Delphi, Fortran, Gosu, Groovy, HANA SQLScript, HANA View, IEC 61131-3 ST, Java, JavaScript, Kotlin, Magik, Matlab, Objective-C, OpenCL, OScript, PHP, PL/SQL, Python, Rust, Simulink, Swift, XML, Visual Basic, Xtend
Integrated Development Infrastructure	
Version Control	Git, Subversion, TFS, File System
IDE Integration	Eclipse, Visual Studio, IntelliJ
Issue Trackers	Jira, TFS, RTC/Jazz
User Management	LDAP, Active Directory, Integrated User Management

2.2 Internal Analyses

Teamscale's real-time analysis engine enables providing immediate feedback – just seconds after a commit to the version control system. The engine comprises many different analyses, ranging from code

quality analyses (see Table 2.2) to test and architecture quality analyses (see Table 2.3). Please note that not all analyses might work for every language (even though most do).

2.3 Integrated External Analyses

Table 2.4 shows the list of external tools which can be integrated in *Teamscale*. These tools comprise static code analyses, test coverage measures as well as usage monitoring.

Table 2.2. Internal Code Quality Analyses

Code Structure	
File Size	Measures the length of a file.
Method Length	Measures the length of a method.
Nesting Depth	Measures the nesting depth of a method.
Cyclomatic Complexity	Indicates overly complex code.
Redundancy	
Clone Coverage	Measures how much code is affected by code clones.
Code Documentation	
Comment Completeness	Measures how many entities of the public interface are missing a comment. Entities can be configured using entity selection predicates (see Appendix D)
Comment Quality	Indicates comments with insufficient quality.
Task Tags	Indicates ToDo/Hack/FixMe comments.
Code Anomalies	
Bad Practice	Indicates bad programming practices.
Correctness	Indicates potential null pointer dereferences.
General Checks	Indicates commented-out-code and empty blocks.
Unused Code	Indicates unused methods and private fields (Language-dependent.)

Table 2.3. Internal Code, Test and Architecture Quality Analyses

Naming	
Naming Convention Violations	Indicates if identifiers violate the naming convention.
Formatting	
Code formatting	Indicates mutliple statements in same line, multiple declarations in same statement or missing braces.
Text formatting	Indicates Non-ASCII characters or long lines.
Test Gaps	
Test Gap Analysis	Indicates changed code that has not been tested.
Architecture Conformance	
Conformance	Indicates dependencies which are not conform to the architecture specification.
Missing Documentation	Indicates types which are not included in the architecture specification.

Table 2.4. Integrated External Tools

Static Code Analysis	
ABAP	SAP Code Inspector
C#	StyleCop, FxCop
C/C++	Clang, PC-Lint
Java	PMD, FindBugs, SpotBugs
JavaScript	ESLint, TSLint
Python	PyLint
XML	SpCop
...	This list is extendable! As *Teamscale* provides an interface, any external analysis tool can be integrated.
Test Coverage Analysis	
ABAP	SAP SCOV
C#	XR.Baboon, Visual Studio Test Coverage
C/C++	lcov, Visual Studio Test Coverage, gcov, CTC
Java	JaCoCo, Cobertura
JavaScript	lcov, Istanbul JSON report
Python	coverage.py
Usage Analysis	
ABAP	UPL



3. Common Use Cases

This chapter describes five common use cases of *Teamscale*. The first use case explains how to configure a project with a corresponding analysis profile (Section 3.1). The second one explains how to receive feedback from *Teamscale* (Section 3.2). Section 3.3 and 3.4 describe how to get an overview of the current quality status and how to monitor it continuously. The last use case describes how to identify untested changes (Section 3.5).

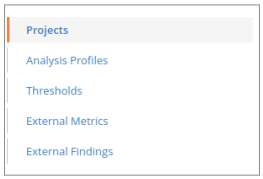


Figure 3.1. Sidebar of the Projects Perspective

3.1 Configuring Projects and Analyses

This use case demonstrates how to configure your project in *Teamscale*. To run *Teamscale* on your code, two steps are required: First, you have to create an analysis profile which configures the analyses that should be run and all their options. Second, you need a project configuration which configures the scope of the analyzed code. Both is done in the **Projects** perspective. The perspective features a sidebar that can be used to navigate to three different configuration pages, Figure 3.1:

- **Projects:** This page allows you to add projects to *Teamscale* that should be analyzed.
- **Analysis Profiles:** This page allows you to configure the analyses that are run for your projects.
- **Metric Thresholds** This page allows you to define a target threshold for all metrics that your projects should abide by.

3.1.1 Best Practices for Your First *Teamscale* Project

Before you jump into configuring your first *Teamscale* project, here are some best practices for first-time users:

3.1.1.1 Best Practice: Start with the Default Analysis Profile

Configuring an analysis profile in all its details can be a daunting task. Teamscale offers hundreds of checks and options that let you customize it fully to your needs. However, when you're setting up your very first project, it makes more sense to start with some sane defaults and then later refine the settings step-by-step. To this end, *Teamscale* provides a default analysis profile for each supported language.

We recommend you start with the default analysis profile for your language and later, step-by-step, refine it to your needs.

3.1.1.2 Best Practice: Limit the Amount of History you Analyze

Teamscale's incremental analysis engine (Section 1.3) allows it to give you instant feedback on each of your commits. Analyzing the changes in a single commit usually only takes a few seconds. If your project has a long history, analyzing that entire history will, of course, take some time. The longer the history, the longer it takes. When you're setting up your first *Teamscale* project, you will usually want to make several smaller changes in short order, e.g. to the analysis profile or the included/excluded files.

Thus, it makes sense to start with a very short history (e.g the last few days) while you're still working on fine-tuning the configuration. Once you're satisfied with the results, reanalyze the project with a longer history.

To limit the amount of history that *Teamscale* will analyze, change the **Start revision** parameter of your repository connector.

3.1.1.3 Best Practice: Exclude Generated Code

Many software projects make use of code generators, e.g. to generate UI components or data classes. Generated code is often repetitive and usually doesn't conform to your company's coding guidelines. Furthermore, you should never make changes to the generated code directly. Thus, for most analyses, it makes no sense to check your generated code with *Teamscale*.

We recommend that you use the include and exclude patterns of the connector configuration to exclude all generated code from the analysis.

If you really want to run certain checks on your generated code (e.g. security analyses), we recommend you create a separate Teamscale *project for it plus a special analysis profile with only those checks enabled.*

To exclude code, add a pattern to the **Excluded file names** parameter of your repository connector.

3.1.2 Creating an Analysis Profile

Teamscale comes with pre-configured analysis profiles for all supported languages (Figure 3.2). These provide sane default values for all analyses. Not all available analyses are enabled by default, though. When starting out, it is usually a good idea to start with the default profile for a language and customize it later to fit your needs. Each project uses exactly one such profile.

To edit an analysis profile, click on next to its name. The other buttons in this row can be used to duplicate a configuration , delete it or download it as an XML file .

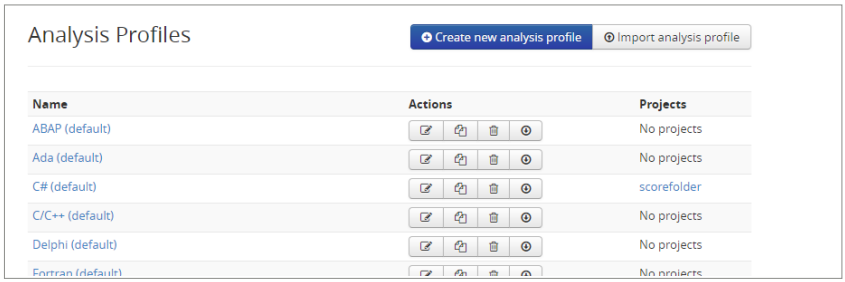
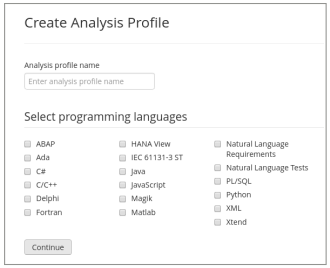
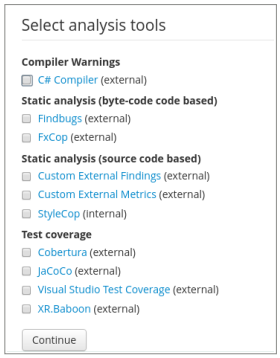


Figure 3.2. The **Analysis Profiles** page

You can also create a new analysis profile from scratch by using Create new analysis profile or import an existing (previously exported) XML file by using Import analysis profile. Regardless of whether you choose to create a completely new profile or edit an existing one, you will be presented with the analysis profile edit screen (Figure 3.3a).



(a) Step 1



(b) Step 2

Figure 3.3. Editing an Analysis Profile

The first section of this screen allows you to specify a name for the profile and select all the programming languages that should be analyzed. Each project you configure in *Teamscale* can only have a single analysis profile. Thus, if your project contains code written in more than one language, you must select all of these languages in the analysis profile to have them analyzed.

If you click on Continue , the next section will appear (Figure 3.3b). In this section, you can select additional analysis tools that should be integrated with *Teamscale*. These include internal tools, like StyleCop for C#, which are run by *Teamscale* itself and external tools, like FindBugs and SpotBugs for Java, which must be run by your nightly build, for example. Internal tools are run incrementally (i.e. with each commit), while the results of the external tools are only updated, when you run them yourself (e.g. nightly). You may also select the tools for which you would like to upload test coverage. If you select none, test coverage metrics will be disabled for this project.

After another click on Continue , you will be taken to the quality indicator configuration screen (Figure 3.4). All analyses in *Teamscale* belong to so-called analysis groups which are in turn aggregated into quality indicators, allowing you to combine related groups. A common use case is to combine several formatting-related groups (e.g. spacing and layouting analyses) into one quality indicator called *Formatting*. All findings generated by these analyses will then show up under that name in *Teamscale* and can easily be filtered or grouped together.

You may create any number of such quality indicators by using the button Add Quality Indicator . To add or remove analysis groups to/from them, simply drag the group's name between the indicators and the section **Inactive Analysis Groups**.

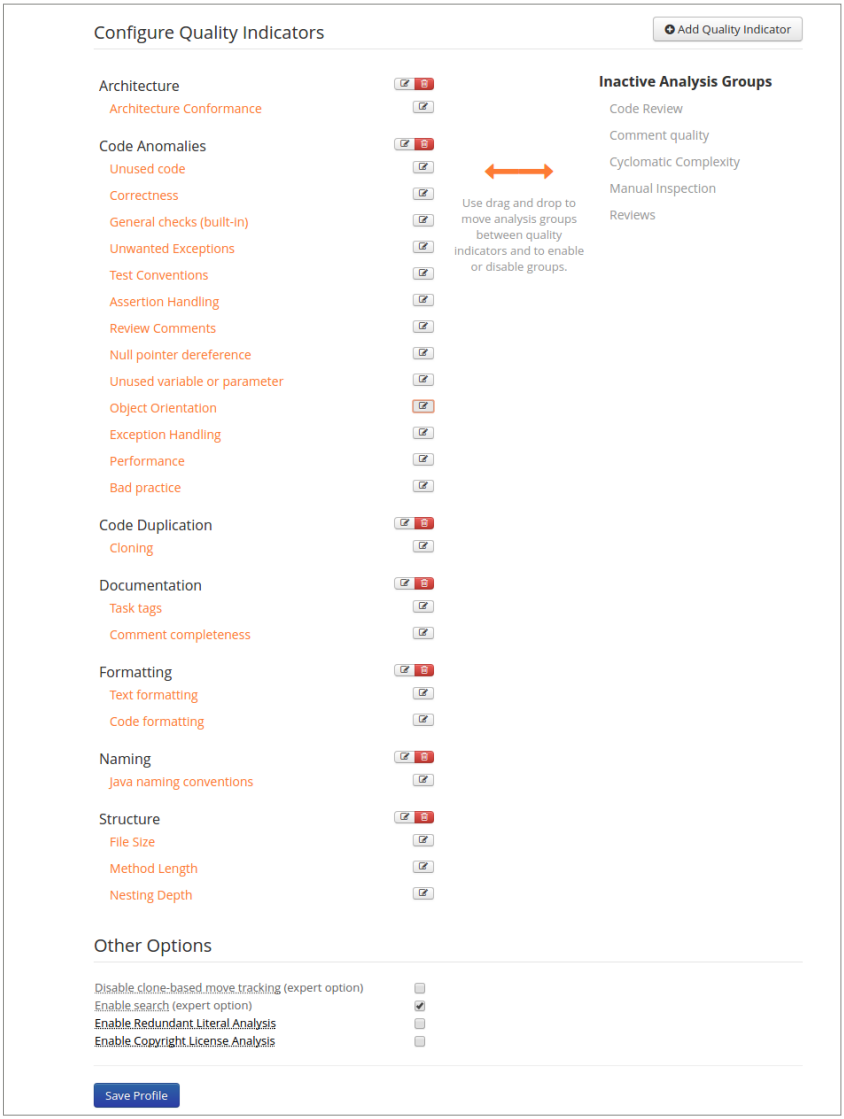


Figure 3.4. Editing an Analysis Profile (Step 3)

Groups and indicators all have their own set of options, which can be accessed via the small ✎ next to their name. For a quality indicator, you can configure its name and which metrics should be calculated for this indicator. For analysis groups, you can configure options that determine the behavior of the analyses as well as the severity of the findings produced by the analyses. To match your individual analysis needs, the analysis groups can also contain custom checks. In which analysis group a custom check is placed, depends on the declaration of the programmer within the implementation of the custom check. *Teamscale* comes with a variety of pre-written custom checks for various languages, see Appendix E.

Since these dialogs often contain many options, you can use the filter text box at the top of the dialog to quickly locate certain options. Be careful with options that are marked as expert options. We recommend to not change these unless you have a deep understanding of the implications. Details for configuration options can be found in Appendix D. Finally, the screen contains a section for **Other Options**. These global options do not affect a single quality indicator but rather the entire analyzed project, e.g. whether to enable searching.

Once you are satisfied with your analysis profile, click Save Profile. You will be directed to a page that shows a summary of the configuration you made. You can now create a project with that profile. Please note that editing an existing analysis profile will not change any settings for existing projects that reference this profile. If you want to not only update the profile but also all projects using it, please select to reanalyze those projects when prompted. Note, though, that the reanalysis might take a while, depending on the length of your project's code history!

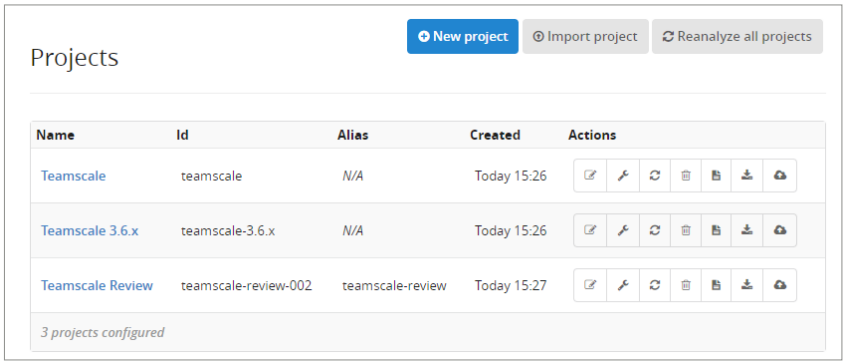


Figure 3.5. Overview of All Configured Projects in *Teamscale*

3.1.3 Creating a Project Configuration

On the Projects page (Figure 3.5), you can see a list of all projects configured in *Teamscale*. The following action box (Figure 3.6) shows you which actions can be applied on an existing project.

To create a new project, select New project button at the top. You will be taken to the project creation screen (Figure 3.7). Here you can enter the name of your project and select the analysis profile (see Section 3.1.2). Next, you can add any number of source code repositories and issue trackers. To do so, select the correct repository or issue tracker type from the corresponding drop-down. A new section (Figure 3.8) will appear where you can configure all options needed to connect to the repository or issue tracker, e.g. credentials, starting revision, files to exclude, etc. Figures 3.9 and 3.10 explain these options in detail.

Most connectors require you to specify an account that holds shared credentials (user name, password). While any user with sufficient rights to create new projects may also create new accounts, only the

Project Management Actions

- Opens the edit dialog for the project. The dialog is the same as for project creation except that the project name can not be changed.
- Opens a dialog for advanced configuration options such as a connection to SAP systems and the threshold configuration to use for the project. *Teamscale* defines a default threshold configuration but users can create custom threshold configurations (also discussed in this section).
- Triggers a complete re-analysis of the project.
- Pauses analysis of the project.
- Deletes the project.
- Exports the project configuration. The exported file can be imported with the "Import project" button in Figure 4.91.
- Exports a complete backup of the project (including analysis results). The exported file can be imported with the "Import project" button in Figure 4.91.
- Opens a view for management of external uploads.
- Opens a view for providing access to the project via users or groups.

Figure 3.6. Actions Available in the Project Overview

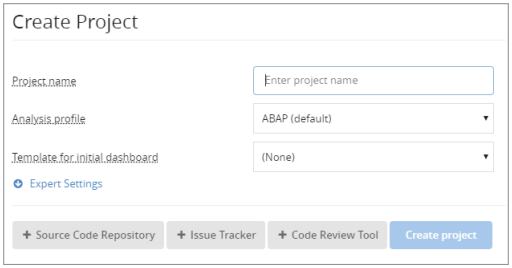


Figure 3.7. Creating a New Project

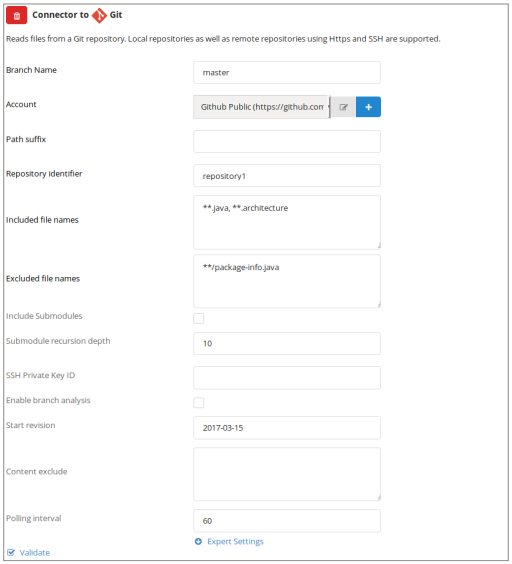


Figure 3.8. Configuring the Connection to a GIT Repository

system administrator may edit or delete existing external credentials in the **Admin** perspective.

You may use ☑ Validate to make sure your settings are correct before clicking Create Project to start the analysis. To monitor analysis progress you can have a look at the **System** or **Activity** perspective (Section 4.1.3).

GIT Configuration Options

Branch Name	The name of the branch to import.
Account	Account credentials that *Teamscale* will use to access the repository. Clicking on the + button opens a credentials creation dialog. The **Account Name** setting in the dialog is *Teamscale*-internal (does not need to match anything in git). When using a file URL in the account, make sure that the corresponding repository is *bare*, i. e. has been cloned with the `--bare` option.
Path suffix	Should *Teamscale* analyze a only a sub directory of the repository?
Repository identifier	*Teamscale* will use this identifier to distinguish different repositories in the same project. (*Teamscale*-internal only)
Included file names	One ore more ant patterns separated by comma or newline to restrict the analysis on a subset of files.
Excluded file names	One ore more ant patterns separated by comma or newline to exclude from the analysis.
Include submodules	Whether to also analyze code in submodules.
Submodule recursion depth	The maximum depths of nested submodules that will be analyzed. (Has no effect if **Include submodules** is not selected.)

Figure 3.9. Configuring a GIT Repository Connector

GIT Configuration Options (continued)

SSH Private Key ID	The ID of a SSH private key stored in *Teamscale* which will be used when connecting to a GIT **Account** over SSH. You can manage all your GIT private keys on the **Settings** page of the **Admin** perspective, in the **Git** section. When using a GIT private key, the **Account** used by the connector has to be configured as follows: The **URL** has to contain the user name (e.g. *ssh://username@gitserver/path/to/repo.git*), the password entered must be the password for the private key and the username may be left empty.
Enable branch analysis	This checkbox activates our new branch support feature. If you select it, *Teamscale* will analyze all branches of the repository. If branch analysis is enabled, the **Branch Name** option specifies the main branch (usually »master«).
Included branches	Explicitly control which branches will be analyzed. (Has no effect if **Enable branch analysis** is not selected.)
Excluded branches	Explicitly state which branches to exclude. Note that *Teamscale* will create so called *anonymous branches* for all historic branches whose name could not be reconstructed. By default these are filtered using the exclude pattern **_anon.***
Start revision	The first revision that will be analyzed. You can enter a date (which selects the first revision after the date) or a revision hash.
Content exclude	A comma-separated list of regular expressions. *Teamscale* will exclude all files that match one of these filters.
Polling interval	How often should we query the repository and update our findings? (Unit is seconds)

Figure 3.10. Configuring a GIT Repository Connector

3.2 Getting Personalized Developer Feedback

As opposed to many existing static analyses tools, *Teamscale* strives to provide *immediate feedback* for developers. Personal and quick feedback is essential when trying to improve quality (see Section 1.2): We have often seen that the absence of it causes quality improvement campaigns to fail (Section 1.2.1). *Teamscale* employs its analyses commit-based and is, thus, able to provide fine-grained feedback. Feedback which does not overflood the developer but instead focuses on his specific need. And feedback which comes quick enough while he is still working on his task.

Developers can use *Teamscale* in several different ways to get feedback: In the activity perspective of the Web UI, they can inform themselves about the impact of their recent commits (Section 3.2.1). If developers want to avoid context switches, they can stay in their IDEs and use *Teamscale* as a plug-in (Section 3.2.2). Or, they can turn on the email service and receive notifications when something happens (Section 3.2.3).

3.2.1 Inspecting the Impact of a Single Change

In the activity perspective, the developer can browse the commit history of the underlying version control system (see Figure 3.11). He can directly glance at the most recent commits at the top of the stream or use the filter mechanism in the sidebar to view only those changes that were committed by himself, for example. The commit stream provides the technical information about the commit itself (author, message, date and related issue) but also its impact on the system's quality: It shows how many findings were added by this commit (red),

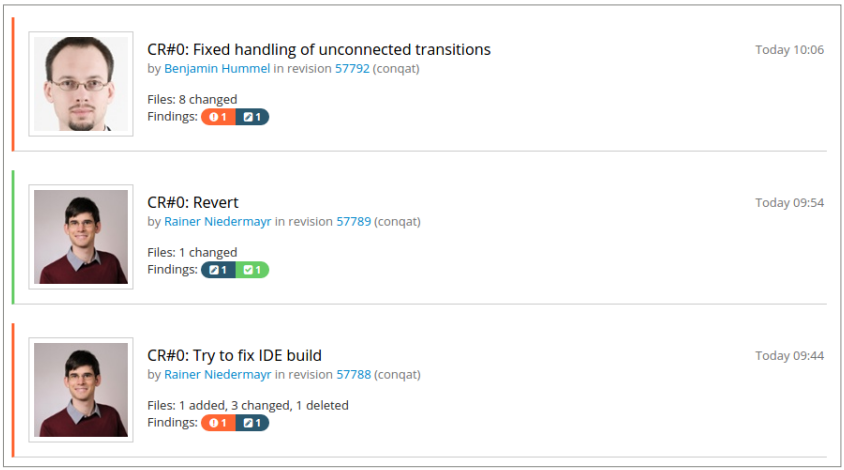


Figure 3.11. Developer Feedback in the Activity Perspective

how many were removed (green) and how many still remain in code that was modified by this commit.

By clicking on the commit, the developer gets more detailed feedback (see Figure 3.12). He can inspect which findings in which file were added or removed and also which impact his commit had on various metrics. He can then either click on the findings message to navigate to the finding's detail page providing more information about the type of the finding and why it is relevant or click on the finding location to directly jump into the source code.

Inspecting the impact of your change in the browser, however, requires the developer to actively pull this information. *Teamscale* also provides push-notification services in your IDE (Section 3.2.2) and via email (Section 3.2.3).

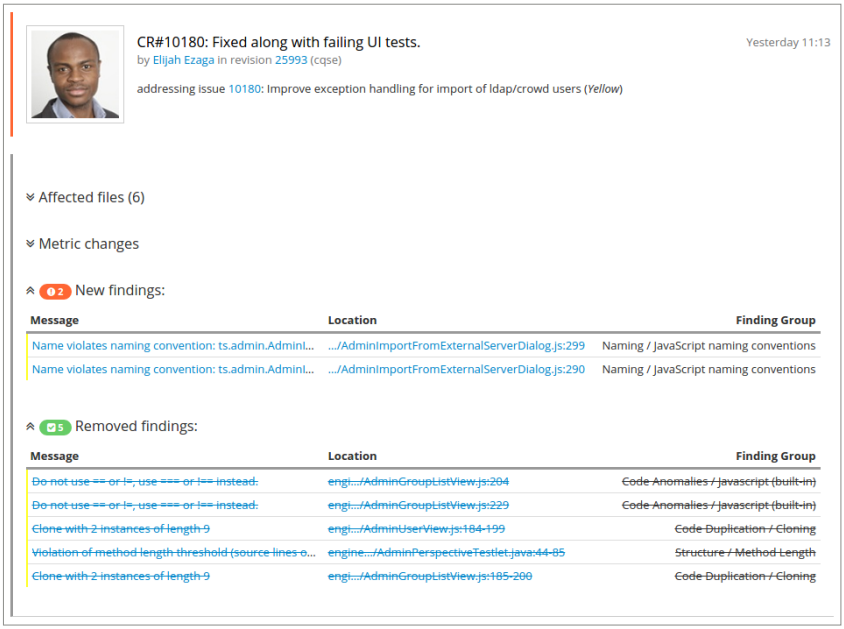


Figure 3.12. Detailed Information about the Impact of a Commit

3.2.2 Being Informed Without Leaving Your IDE

Rather than switching to the browser, developers often prefer to gather all necessary information within their IDE. *Teamscale* integrates in common development environments such as Eclipse, Visual Studio and IntelliJ. With the plug-in, you can inspect the findings directly as markers in the code. To clean up on-the-fly or to inspect newly introduced findings, developers do not have to leave their IDE. Thus, they can avoid distracting context switches. Section 4.2 provides more details about how the IDE integration can be set up.

3.2.3 Push Notifications: Receiving Emails about Changes

Furthermore, you can configure *Teamscale* to notify you via e-mail or on your desktop. It remains up to you, about what kind of changes you want to be alerted.

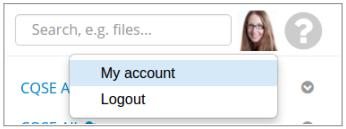


Figure 3.13. Navigating to the Personal Account

To configure your notifications, click on your user symbol in the upper right corner of the browser and select **My Profile** in the drop down menu. Then you can select the **Notification Rules** page in the right side bar menu (see Figure 3.13). You can activate or deactivate desktop notifications by using the check box **Enable desktop notification**, see Figure 3.14. In terms of email notifications, you can add rules to be notified when a certain type of finding is added, when a metric threshold is violated or when you are related to a task.

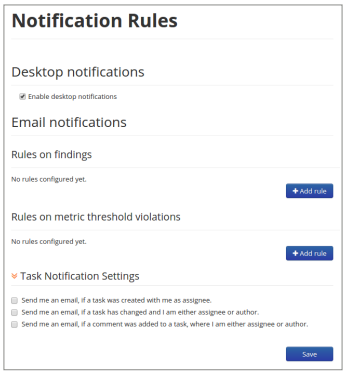


Figure 3.14. Configuring Personal Email Notifications

Figure 3.15 shows exemplary the configuration dialog to add a notification rule for findings. You can configure the projects on which the rules applies as well as committers and events triggering the rule. You can also explicitly sate to not trigger the rule if, e. g. the finding category or the finding group equal your specified string. To look up how your categories are named and which groups you are using, please refer to your analysis profile.

3.3 Getting an Overview of the Quality Status

Often, you as a developer or as a project manager would like to get an overview of the current quality status of your system – or maybe even of your portfolio of systems. In this section, you will learn how to create a single project overview (Section 3.3.1) and a multiple project overview (Section 3.3.2).

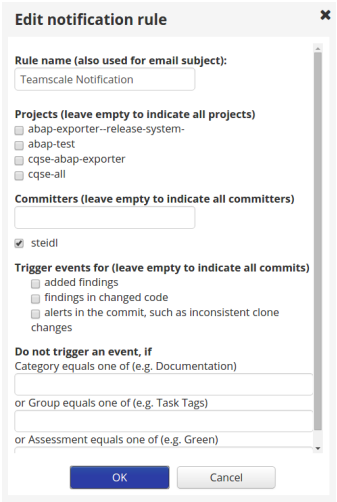


Figure 3.15. Editing a Notification Rule for Findings

3.3.1 Creating a Single Project Overview

To get an overview of the quality status of one single project, there are two different ways – either with the **Metrics** perspective or the **Dashboard** perspective. We will start with the latter.

To configure your personal overview dashboard, click on the **Dashboard** perspective. The **Dashboard** perspective will initially be empty, if no dashboards have been configured yet. It provides three buttons in the right sidebar which are explained in Figure 3.16. Click on  icon to add a new dashboard and select **Create from template** (T) from the drop-down menu. The following dialog will give you the opportunity to configure the creation of the new dashboard, e. g. give it a name and select a template, e. g. the default overview template. For more details, please refer to the user reference in Section 4.1.2.

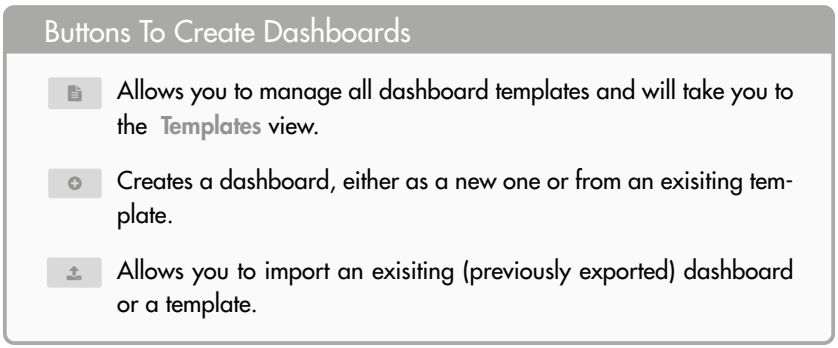


Figure 3.16. Buttons in the Sidebar of the **Dashboard** Perspective

The dashboard now gives you an overview of the quality status of your system, see Figure 3.17. You can modify the dashboard by clicking on next to its name in the right side bar and selecting **Edit dashboard**. Editing allows you to rearrange the widgets and create or delete widgets, Figure 3.18. For more details, please refer to the user reference in Section 4.1.2 or the configuration help for widgets in Appendix A.

The information from the dashboard can be adapted based on (sub-) project and time. To change the project, edit the dashboard as above and configure each widget using the symbol in the top left corner of a widget. By setting the **Path** you can either switch to a different project or select a component or subcomponent of your system.

If you would like to change the time for the dashboard, leave the edit mode and go back to viewing the dashboard. Then click on your projects name in the top right corner (see Figure 3.19). A new row will expand underneath the perspective header, allowing you to click

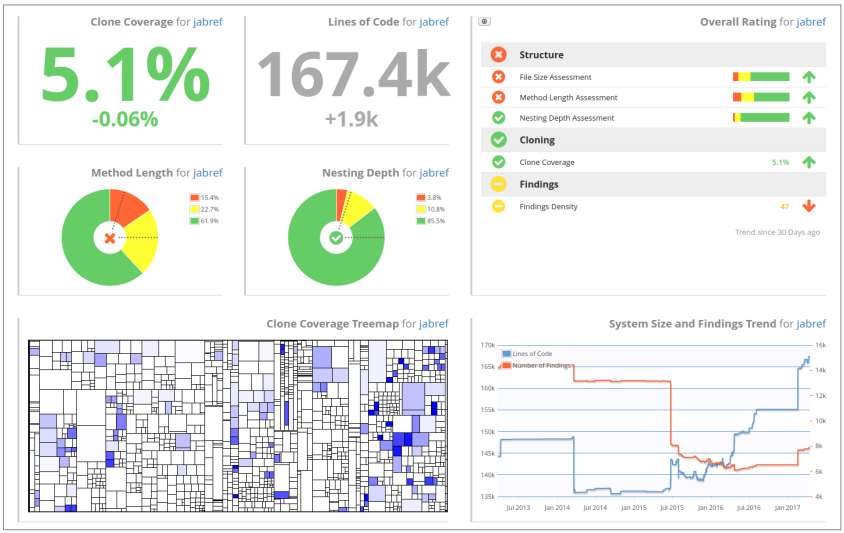


Figure 3.17. Overview Dashboard

on **Timetravel**. The timetravel allows you to select a calendar time, a revision or a baseline. Then, the dashboard will display the system's quality status at the selected point in time.

As an alternative to the **Dashboard** perspective, you can also navigate to the **Metrics** perspective. Here, you will find an overview of all metrics in the first row of the table of the perspective, see Figure 3.20. The overview contains all numeric metrics and assessment metrics and can also be configured. However, it does not provide all information available in the **Dashboard** perspective. Hence, it is useful for a first glance but not for a personally tailored overview.

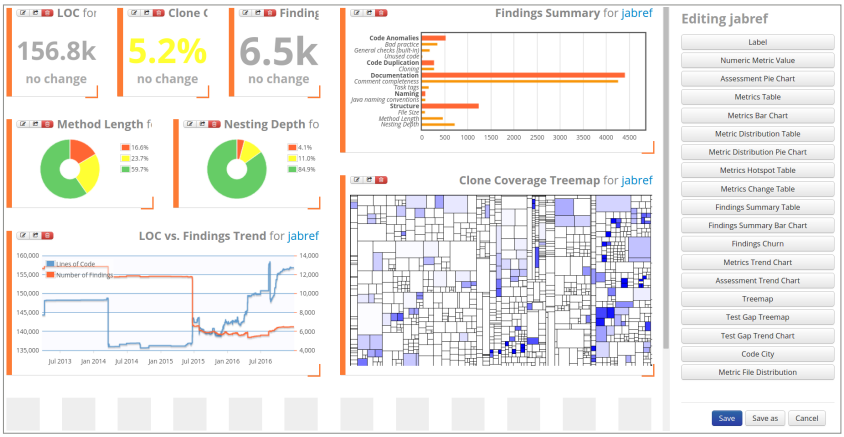


Figure 3.18. Editing a Dashboard



Figure 3.19. Using the Timetravel for a Dashboard

3.3.2 Creating a Multi Project Overview

If your role is being a project manager rather than a developer, you might be more interested in getting an overview of a portfolio of systems. *Teamscale*'s dashboards support combining information from multiple projects. Figure 3.21 shows a very simple example dashboard displaying the lines of code information for six different projects as well as their trend over the last 30 days. You can combine information from all projects. They do not need to be written in the same programming language and they also do not need to be in the same version control system. As long as you access rights, you can combine any of them in one single dashboard.

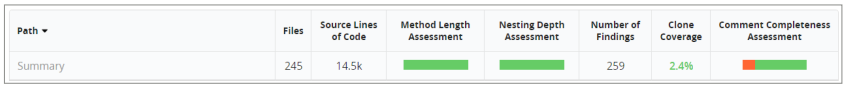


Figure 3.20. Metrics Summary in **Metrics** Perspective Table

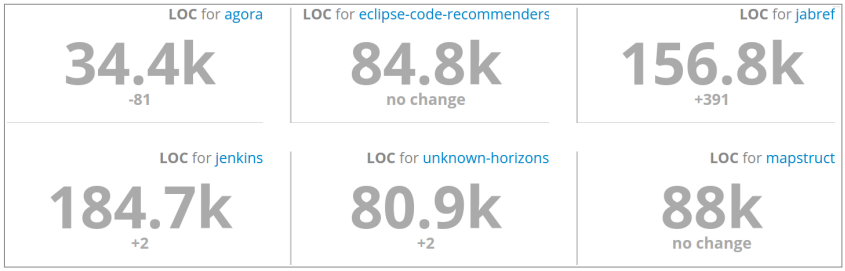


Figure 3.21. Example Portfolio Dashboard

3.4 Monitoring and Assessing Quality Evolution

An essential part of software intelligence comprises monitoring your quality evolution and drawing the right conclusions from it: when something goes wrong, panic and over-the-top decisions are usually the opposite of helpful. Instead, *Teamscale* provides a transparent root-cause analysis. Based on these results, you can derive reasonable and specific counter measures. This section shows how *Teamscale* supports the role of a quality-engineer in a quality-control-like process as depicted in Section 1.2.2. In this context, quality control refers specifically to *code* quality control.

3.4.1 Inspecting Recent Metric Trends

Monitoring quality evolution often starts with the question: In which direction are we heading? Is there a trend to get better or are we doing worse? As *Teamscale* has the entire history available from the underlying repository, it can provide transparent and understandable answers.

For every metric you have configured, *Teamscale* offers you to inspect it as a trend chart. It can compute this trends over any selected period within the history that *Teamscale* has available. To view quality trends for your system, go to the Dashboard perspective and select the button in the right sidebar. If you choose **Create From Template**, you can give the dashboard a name and select the overview dashboard template as well as your specific project. The dashboard gives you an overview of your quality as status as already explained in Section 3.3. But the dashboard also shows you in which direction you are heading: Metric that consist only of one numeric values, such as lines of code, clone coverage and findings count are displayed with a numeric metric value widget. This widget shows both the absolute value as well as the trend over the last 7 days as default, see Figure 3.22. If the trend is positive, the delta in the metric value will be marked as green, see the decreasing clone coverage in Figure 3.22. In contrast, if the trend is negative, the delta will be marked in red.



Figure 3.22. Trend Indication for Numeric Metrics

To change the length of the considered trend interval, you can click on the right arrow next to your dashboard's name in the right sidebar

and select **Edit Dashboard**. Each widget displays an edit button in the top left corner. Click on it and the dialog **Edit Parameters** will open in which you can set the duration of your trend interval (see Figure 3.23).

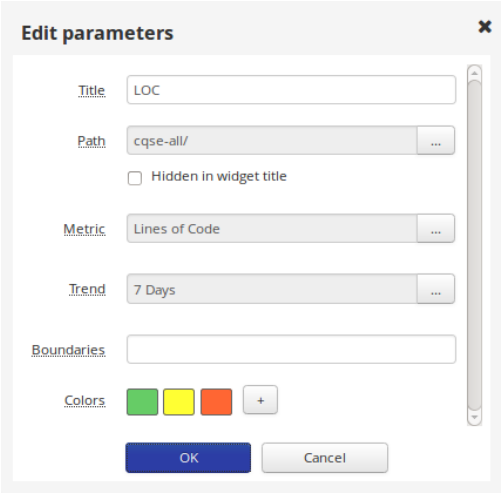


Figure 3.23. Edit Parameters for a Numeric Metric Value Widget

Dashboards can also display trends as graphical trend charts, see Figure 3.24. The template already provides trend chart for the lines of code metric and the number of findings. With the same edit mechanism as above, the widget can be configured to show any numeric metric trend. The default time span of the interval is set to zero days, which means that the entire available history is included. To obtain the graph as depicted in Figure 3.24, we configured the widget to include the clone coverage metric.

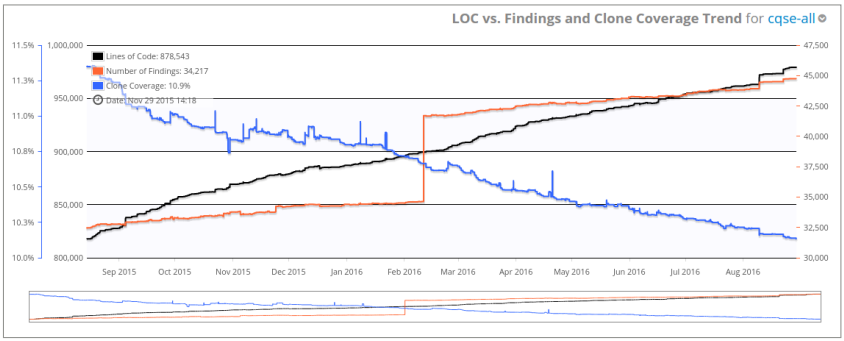


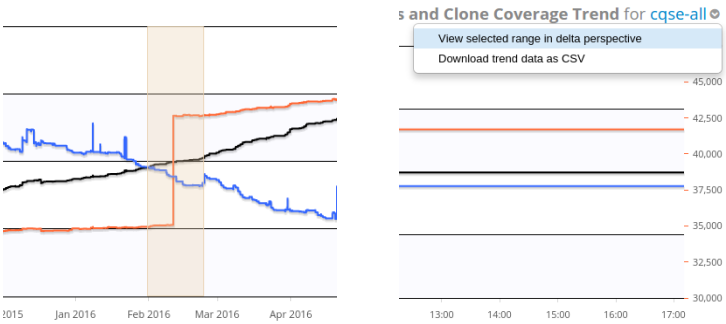
Figure 3.24. Trend Charts for Numeric Metrics

3.4.2 Finding the Root Cause for a Trend

Sometimes, metric trends reveal suspicious drops or increases, for example the sharp increase in the number of findings (red line) in Figure 3.24. This makes us as quality engineers curious to find out what happened. With a specific configuration of the dashboard, you can discover the underlying root cause. Let us show you how: As before, go to the edit mode of our dashboard and open the dialog **Edit Parameters** of this specific metric trend chart widget. Enable the two options **Zooming** and **Action Menu**. Then save the dashboard.

To inspect a suspicious change in a trend, use the mouse to mark the area wich you would like to inspect closer and zoom in, as indicated in Figure 3.25a. Zooming in narrows down the time interval while providing a higher resolution. When releasing the mouse button, the chart will update to represent the zoomed-in area. Then, use the action menu button ⌄ to select **View selected range in delta perspective**, see Figure 3.25b.

While you are redirected to the **Delta** perspective, *Teamscale* computes the so-called delta analysis in the background: It uses the start



(a) Zooming Into a Trend Chart

(b) Selecting the Corresponding Delta

Figure 3.25. Root Cause Analysis For Trend Charts

and the end of the interval that you selected by zooming into the trend chart. It then computes and reconstructs everything that happened in between this start and end date: As a result, it shows you, for example, which files were touched within the time interval or how metrics have changed. In particular, *Teamscale* also shows you the commits to the repository that were made within this time frame. They are often the first starting point to the root-cause analysis.

Select the **Repository Changes** in the left sidebar menu (see Figure 3.26) and scroll through the commits, see Figure 3.27. In our specific example, we want to figure out why the findings trend suddenly increased. Hence, we are looking for commits with a high findings churn. As our selected delta only contains two commits, it is rather obvious which one is responsible.

Click on the commit »inital ABAP architecture« to find out more details about the commit, see Figure 3.28. The commit detail view will show you that an architecture file was added. Resulting from it, more than 7000 findings emerged revealing violations from the newly specified

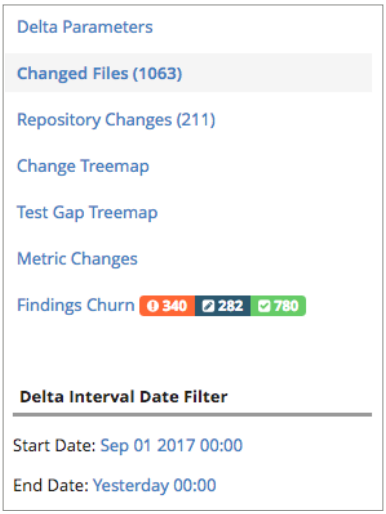


Figure 3.26. The Sidebar of the Delta Perspective.

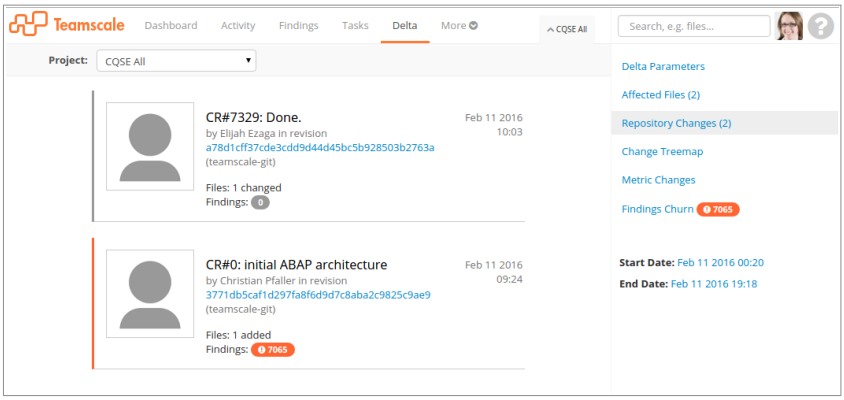


Figure 3.27. The Commits in the Delta

architecture. This is the root-cause for the increase in the findings

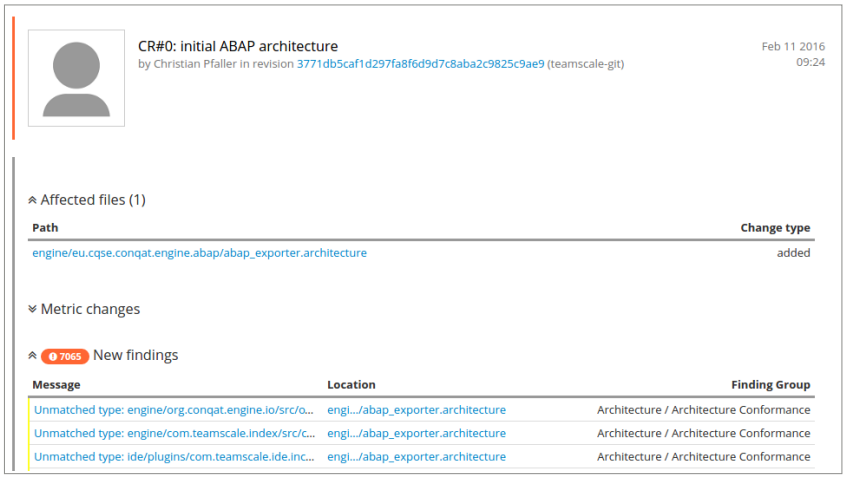


Figure 3.28. The Commit Detail View

trend. For more details about the **Delta** perspective, please refer to Section 4.1.10.

3.4.3 Analyzing the Findings Churn

Besides monitoring the quality status in terms of metric trends, a thorough quality assessment also comprises a deeper inspection of the evolution of the findings in a system. You might want to check if *severe* findings were added. You might be curious how many findings were removed. Or if developers became lazy rather than still cleaning up on the fly. To analyze the findings churn, you can use two distinct perspectives: the **Findings Perspective** or the **Delta Perspective**. This section focuses on the first. To learn about the **Delta Perspective**, refer to Section 3.4.4.

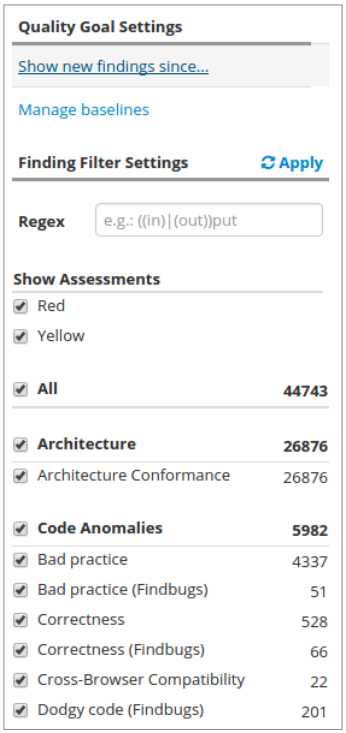


Figure 3.29. Filter Options in Findings Perspective

The **Findings Perspective** in general provides a navigation page through all findings that exist in your system. But it can be also used to just inspect the findings that were added in a specific time frame, for example within the last month. This gives you the chance to assess the recent quality evolution.

To focus on *recent* findings, use the right sidebar to configure which findings are shown, see Figure 3.29. For our use case, click on **Show new findings since...**. With the dialog opening up, you can select a date – findings will then be shown only if they were added *after* this

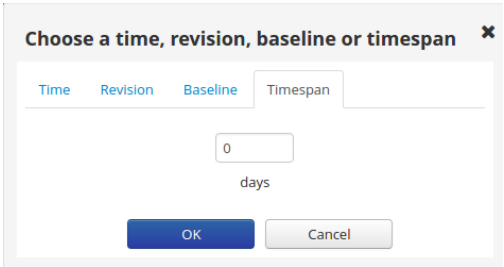


Figure 3.30. Selecting the Baseline

reference date. This dialog has different options, see Figure 3.30. Under **Timespan**, you can indicate, for example, that new findings from *within* the last 40 days should be shown. Under **Time**, you can pick your reference date from the calendar or select it as a revision in your repository under **Revision**. The **Baseline** option additionally provides you the opportunity to use a pre-configured baseline (or create a new one).

After you clicked on the button **OK**, the findings perspective will show you only findings that were added since the selected date. You can see this from the note in the **Quality Goal Settings** in the right sidebar, see Figure 3.31. Caring only about the *new* findings corresponds to the quality goal *preserving* as outlined in Section 1.2.2. If you would like to assess your quality evolution based on the quality goal *improving*, enable the option **Include findings in changed code**. In this case, the findings perspective shows not only the *new* findings but also findings in code that has been *modified* since your reference start date.

To filter findings further, you can also filter them based on their finding group and finding category. Figure 3.29 shows you the selection mechanism. For example, deselect the check box **All**, select only the checkbox **Code Duplication** and scroll to the bottom of the page

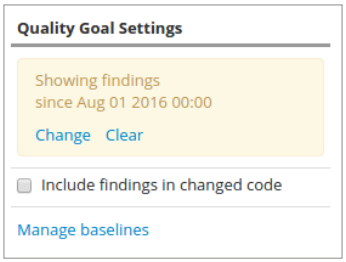


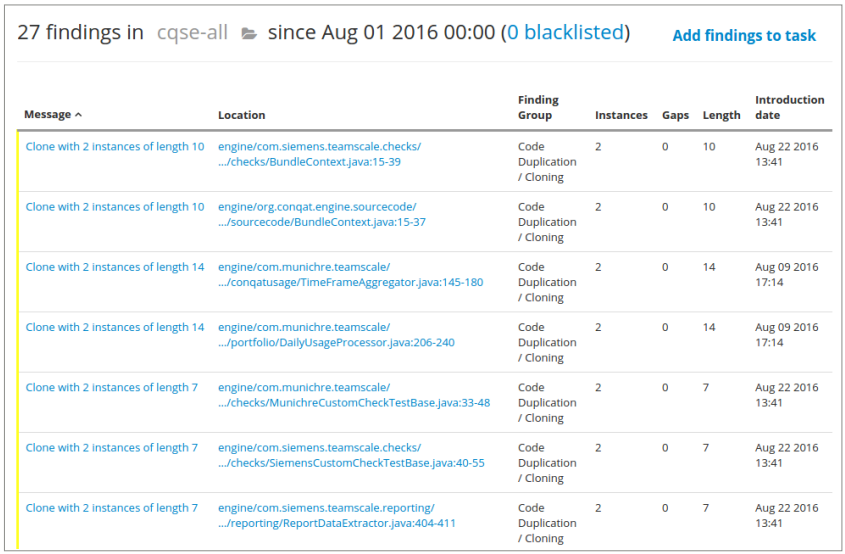
Figure 3.31. Quality Goals Settings in the Findings Perspective

to press the button ⟳ to **Apply**. The main table of the findings perspective now shows you only recent clones, see Figure 3.32. You can use the table header to sort the set of clones, for example based on their number of instances or based on their length. This way, you can inspect, for example the largest clone recently added or the clone with the most instances. This usually should help you assessing what kind of clones were added to the system and how critical they are.

By clicking on a finding, you navigate to the **Findings Detail Page** of a specific clone, see Figure 3.33. This page gives you more detailed information about the finding. You can inspect the **introduction diff**, for example, or jump into the code to see the full surrounding file. In the specific example of code clones, you can also view the siblings, i. e. the other instances of the clone.

3.4.4 Performing a Baseline-Delta-Analysis

Under certain circumstances, you want to compare two snapshots of the system against each other and inspect everything that happened in between. For this use case, *Teamscale* provides the **Delta Perspective**. This perspective allows you to choose an arbitrary start



27 findings in cqse-all since Aug 01 2016 00:00 (0 blacklisted) Add findings to task

Message	Location	Finding Group	Instances	Gaps	Length	Introduction date
Clone with 2 instances of length 10	engine/com.siemens.teamscale.checks/.../checks/BundleContext.java:15-39	Code Duplication / Cloning	2	0	10	Aug 22 2016 13:41
Clone with 2 instances of length 10	engine/org.conqat.engine.sourcecode/.../sourcecode/BundleContext.java:15-37	Code Duplication / Cloning	2	0	10	Aug 22 2016 13:41
Clone with 2 instances of length 14	engine/com.munichre.teamscale/.../conqatusage/TimeFrameAggregator.java:145-180	Code Duplication / Cloning	2	0	14	Aug 09 2016 17:14
Clone with 2 instances of length 14	engine/com.munichre.teamscale/.../portfolio/DailyUsageProcessor.java:206-240	Code Duplication / Cloning	2	0	14	Aug 09 2016 17:14
Clone with 2 instances of length 7	engine/com.munichre.teamscale/.../checks/MunichreCustomCheckTestBase.java:33-48	Code Duplication / Cloning	2	0	7	Aug 22 2016 13:41
Clone with 2 instances of length 7	engine/com.siemens.teamscale.checks/.../checks/SiemensCustomCheckTestBase.java:40-55	Code Duplication / Cloning	2	0	7	Aug 22 2016 13:41
Clone with 2 instances of length 7	engine/com.siemens.teamscale.reporting/.../reporting/ReportDataExtractor.java:404-411	Code Duplication / Cloning	2	0	7	Aug 22 2016 13:41

Figure 3.32. Assessing Clone Findings

date (the baseline) and an end date and performs the delta analysis in between.

To perform a delta analysis, click on **Delta Parameters** in the right side bar and select your start and end date, see Figure 3.34. You can choose both dates in form of a calendar date, a repository revision or a previously defined baseline of your system. In our example, we chose the month of August as underlying time interval for the delta. If branch support is enabled, you can also pick the branch of the underlying version control system. Then, hit the button Compute Delta.

After the delta analysis finished, *Teamscale* will redirect you to the **Affected Files** page within the **Delta Perspective**, see Figure 3.35. It shows you how many and also which files were added, deleted

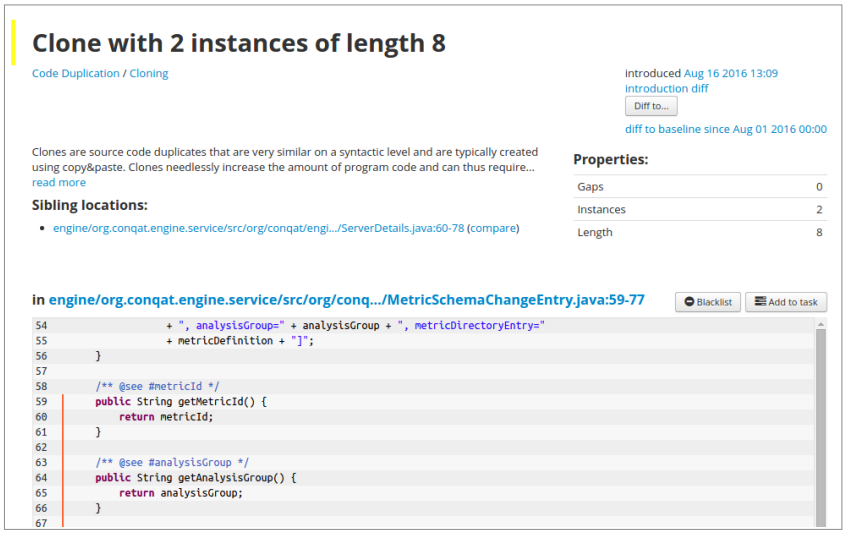


Figure 3.33. Detail Page for a Clone Finding

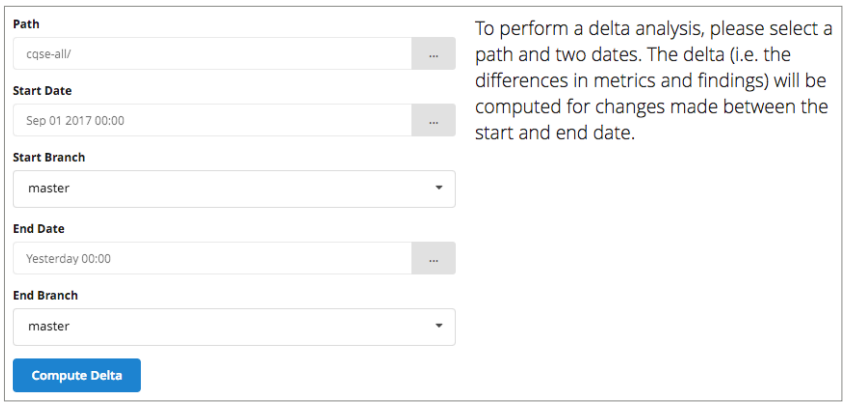


Figure 3.34. Selecting Start and End Date for Delta Analysis

or moved within your project during, in our example, the month of August.

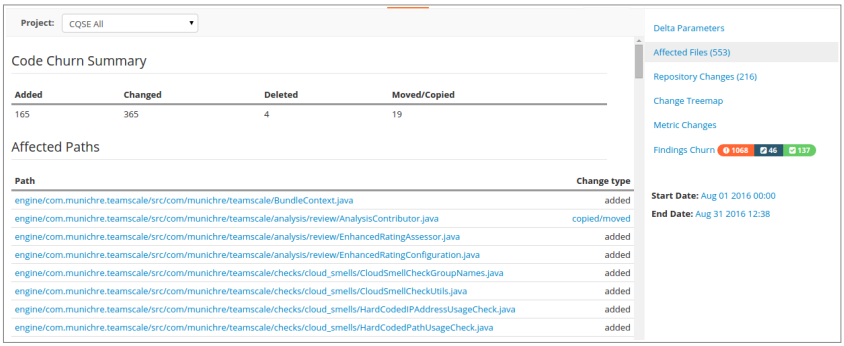


Figure 3.35. Overview of the Affected Files within a Delta

When you click on **Findings Churn** in the left sidebar, you can inspect findings that were either *added* or *removed* within the given time interal. Additionally, this page also shows you findings in code thas was *modified* in this period. Similar as in the **Findings Perspective**, you can navigate and filter these findings (see Section 3.4.3).

3.5 Identifying Untested Changes

Untested changes are typically error prone and often lead to production defects [Ede+13]. *Teamscale*'s *Test Gap analysis* is an automated way of identifying changed code that was never tested since its last modification. Such areas are called test gaps. Knowing a system's test gaps allows you to make a conscious decision, which code has to be tested before shipping and which gaps are of low risk and can be left untested. This section shows how to identify untested changes with the *Test Gap analysis* feature in *Teamscale*.

To use *Test Gap analysis*, your project should be connected to your source code repository, and you should upload test coverage regu-

larly. Please refer to the Section 3.1 for information on how to set up projects in *Teamscale*.

As *Test Gap analysis* works on *changes*, you need to define a suitable time span, i. e. a baseline and an *end date*. *Baseline* refers to the date from which on to investigate code changes, while *end date* marks the other end of the time frame. All changes that were made to the system between baseline and end date are considered in the *Test Gap analysis*.

3.5.1 Creating a *Test Gap analysis* Dashboard

In *Teamscale*, untested changes are visualized using dashboards. Therefore, we first briefly describe how to configure a *Test Gap analysis* dashboard, which is needed to identify test gaps. For more detailed information on how to create dashboards and use dashboard widgets in *Teamscale*, please refer to Section 4.1.2.

Switch to the **Dashboard** perspective and create a new dashboard. On the dashboard canvas, add a new **Test Gap Treemap** widget. You will see a treemap, a graphical representation of the source code of your project. Every rectangle on this treemap corresponds to a method in the source code. The bigger the rectangle, the longer the method. Resize the widget by dragging its lower right corner until its size fits your needs. Then hit ✎ at the upper left corner to configure it.

On the configuration dialog, which is shown in Figure 3.36, there are several settings that determine, which information is shown on the treemap. Depending on your situation at hand, i. e., which specific information you are interested in, you will need to make different settings. These are explaind in Figure 3.38.

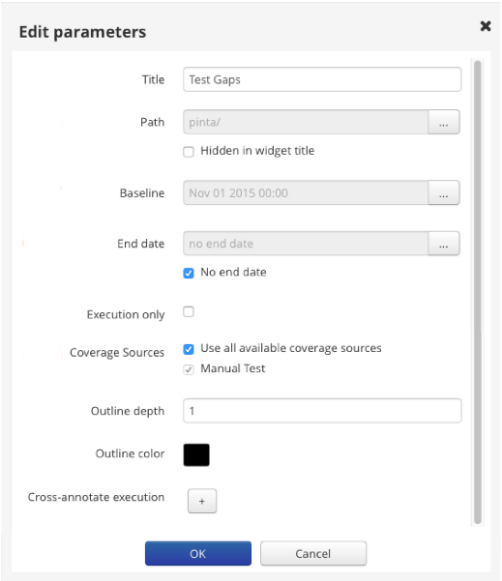


Figure 3.36. Configuration Settings for Test Gap Treemaps

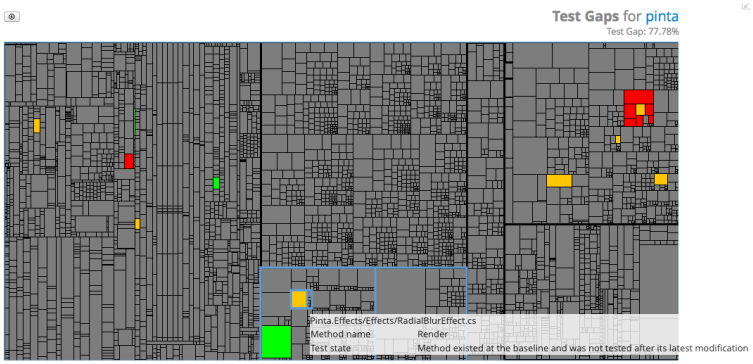


Figure 3.37. Test Gaps on the Test Gap Treemap

TGA Configuration Options

Path	Refers to the part of your project you want to see on the Test Gap treemap. You might want to select your whole project, but you can also choose a smaller subset, say the component you are responsible for. The treemap will then zoom into that area.
Baseline	Refers to the date from which on to consider changes, as described above. Choose a date slightly before the first change that is relevant to you to ensure that no changes are lost.
End date	marks the other end of the time frame. The **No end date** option ensures that also future changes and future test information will be shown, i. e. this option keeps the end date always at the current time.
Coverage Sources	are logical containers for different kinds of coverage, e. g. »Manual Test« or »Unit Test« or »Test Server 3«. Whenever uploading coverage to Teamscale, a coverage source has to be specified. Here you can select the coverage you want to consider in the Test Gap analysis by simply ticking the corresponding check box, or selecting all coverage sources.

Figure 3.38. Configuring a TGA Dashboard Widget

The resulting treemap should look like the one depicted in Figure 3.37. On this treemap, *gray* rectangles correspond to methods that were not changed since the baseline you entered. In contrast, *orange* methods were changed, while *red* methods were added since the baseline. If a method is depicted *green*, it was also executed since its last modification.

Figure 3.39 helps to get a better understanding of how the selected baseline and end date influence what is shown on the Test Gap treemap,

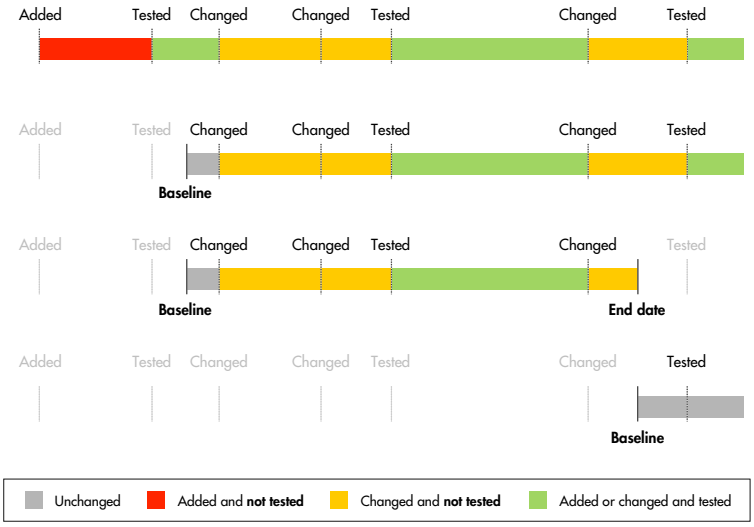


Figure 3.39. Influence of the Selected Baseline on How a Method is Shown on the Test Gap Treemap.

and how methods are color-coded. It shows the history of a single method represented using the previously described colors: The method is first added, then tested, later changed again, and so forth. Because the last version of the method is tested, it will appear green on the treemap.

The second row illustrates, what happens, if a baseline is set at a specific point: The method would still appear green on the treemap.

However, if an end date is set as well, the color might change, as illustrated in row three: Because the end date is set before the last version that was tested, the method would appear orange.

Finally, the last row depicts that a method is shown gray, if the baseline is set after the last change date.

In Figure 3.37, the method `Render` in the class `RadialBlurEffect.cs` is a test gap, because it was changed after the baseline, but not tested since its last modification. If this method is tested in its current version, and the coverage is uploaded to *Teamscale*, the corresponding rectangle will become green, and there will be one test gap less.

Simple `getter` and `setter` methods are not considered by the *Test Gap analysis*. This affects all methods whose method name starts with either "set", "get" or "is" and whose body contains only one statement. Additionally, auto generated properties in C#, denoted by `get` or `set`, are excluded from the Test Gap analysis. Simple methods are too trivial to test and thus cause noise in the Test Gap treemap. Eliminating these methods reduces unnecessary information in the Test Gap treemap and sets the focus on more relevant Test Gaps.

3.5.2 Working with Test Gap Treemaps in Practice

A Test Gap treemap answers the question »*which methods have been changed or added since a specific point in time and which of these methods have not been tested yet?*«. These methods, i. e. the untested changes, are called test gaps.

For each test gap, a conscious decision should be made, if the corresponding source code should be tested, or not – e. g. because there are much more critical parts in the code, and only limited test resources left.

To be able to make this decision, typically two pieces of information are needed: First, which functionality the test gap, i. e. the method, belongs to, and second, which changes were made to this method.

Developers usually get a good understanding of the functionality of a method when they see the source code. As shown in Figure 3.37, hovering over a method in the treemap reveals information about the method under the cursor, such as the class and method name. To see the code behind a method, simply click on the rectangle.

From the source code view, the complete history of the corresponding file can be accessed by pressing the »H« key, as described in more detail in Section 4.1.5. The history of the code shows the corresponding commit messages and, if *Teamscale* is connected to an issue tracker, also points to the change requests from where the changes originate. This information is typically enough to decide how critical a change is.

In our experience, it is most efficient to have a person responsible for development and a person responsible for testing perform this workflow together.

To get just a quick overview of the test status of a system, there is a percentage in the upper right corner of the Test Gap treemap. This denotes the test gap ratio for the code shown on the treemap, i. e. the amount of modified or new methods covered by tests divided by the total amount of modified or new methods. In the example in Figure 3.37, there is a Test Gap ratio of over 77 %, meaning that over 77 % of the modified and added methods shown on the treemap have not been tested in their current version.

It is also possible to download a `CSV` file containing all the methods and their corresponding test state, i. e. if the method is unchanged, was tested, or is a test gap. To this end, click on left above the treemap. The downloaded `CSV` file allows for an in-depth inspection of the remaining test gaps.

In case of very large systems, rectangles in the Test Gap treemap may become very small and hard to distinguish. In these situations it can be easier to identify test gaps when the treemap just shows changed methods, hiding all the gray areas. You can achieve this behavior by enabling the **Focus on changes** option in the treemap's properties, which are explained in more detail in Section A.26. Figure 3.40 shows the effect of that option.

3.6 Improving test execution efficiency

Regression testing of changes is usually done by re-running all automated or manual tests of a system (retest all). As more and more features are added to a software system over its lifetime, the number and overall runtime of tests also increases. With increasing runtime of the testsuite the number of possible retest all runs decreases. Therefore, the time between introducing a new bug with a change and getting the results of a failed regression test increases as well. How-

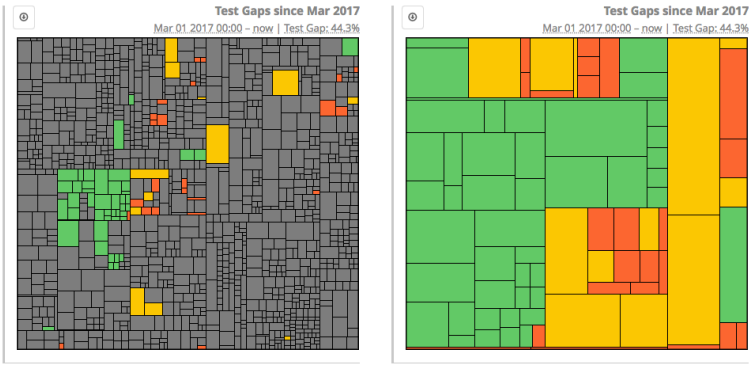


Figure 3.40. Test Gap Treemap with **Focus on changes** disabled (left) and enabled (right).

ever, re-executing all tests is not only expensive, it is also inefficient, since most of the tests in the test suite don't test a given change to the codebase.

test impact analysis provides an automated mechanism to *select* and *prioritize* tests to be executed based on the coverage of previous test runs. For that purpose testwise coverage must be recorded and uploaded to *Teamscale*. In follow-up runs the Test Runner will ask *Teamscale* for the tests that need to be executed to test the changes of the current commit to be tested. *Teamscale* uses the data from the initial run to determine, which of the tests will actually execute the code changes. These impacted tests are then prioritized such that tests with a higher probability of failing are executed before the other tests. The Test Runner can then start executing tests based on the test list.

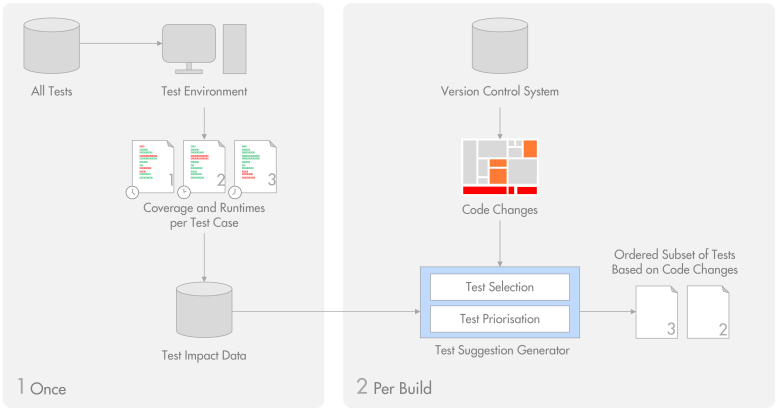


Figure 3.41. *Test Impact analysis* big picture.

```
74          */
75          async retrieveAnalysisProfile() {
76              const analysisProfileAndProject
77
78              this.analysisProfile = analysisP
79              this.projectInfos = analysisPro
80              ts.project.ProjectListView.addC
81          }
```

Figure 3.42. Testwise coverage indicator

3.6.1 Determining tests covering a method

After the initial testwise coverage upload the Code perspective will show an ✅ indicator per method if the the method has been covered as shown in Figure 3.42.

A click on the indicator opens a dialog box as shown in Figure 3.43, which presents a list of all tests that executed the selected method.

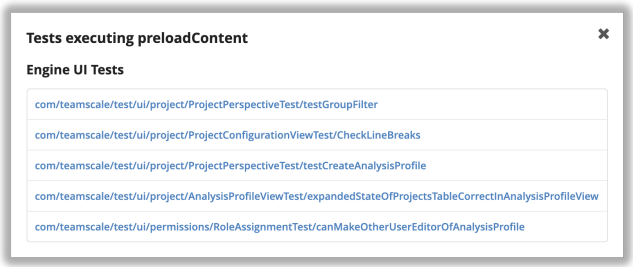


Figure 3.43. Tests executing method dialog

3.6.2 Viewing detailed test information

A single test can either be inspected by following the links in the dialog box or by selecting the test from the Tests perspective in the section Test executions. The detail view depicted in Figure 3.44 shows the most recent test execution result at the top. The treemap below shows coverage of this single test on the whole codebase. By using the Show only executed methods the treemap can be focused to only show the executed code regions. At the bottom the history of the 10 previous test executions are shown with the corresponding durations and test results. A click on one of the previous test executions opens the test details view for this execution e. g. to inspect the error message of a previous failure.

3.6.3 Determining impacted tests over large time intervals

In the Delta perspective the list of impacted tests can be inspected by either selecting a time range or a merge scenario (see Section 3.4.4).

The result is shown in Figure 3.45. The list will show all tests that are impacted by the selected changes. A prerequisite for this is that a

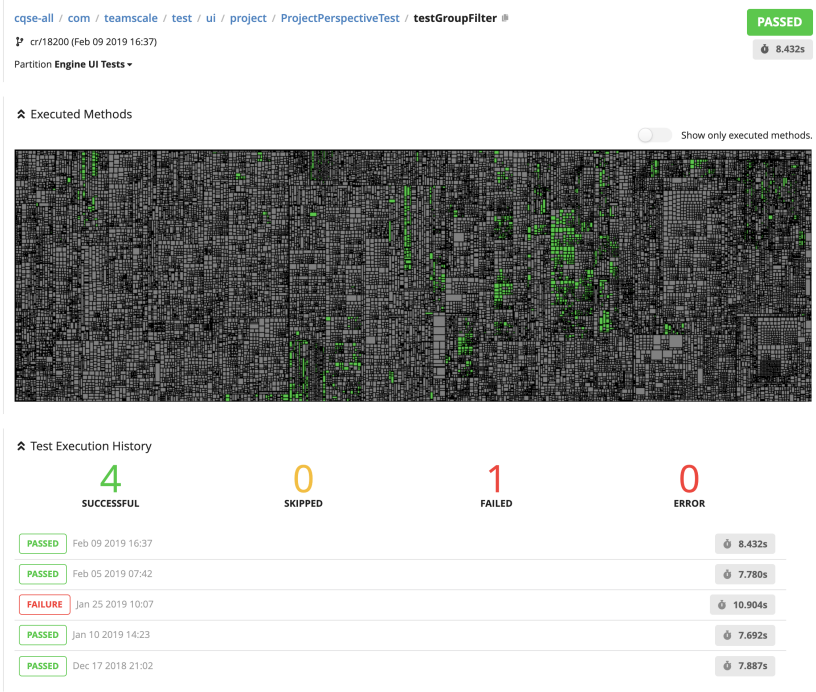


Figure 3.44. Test details view.

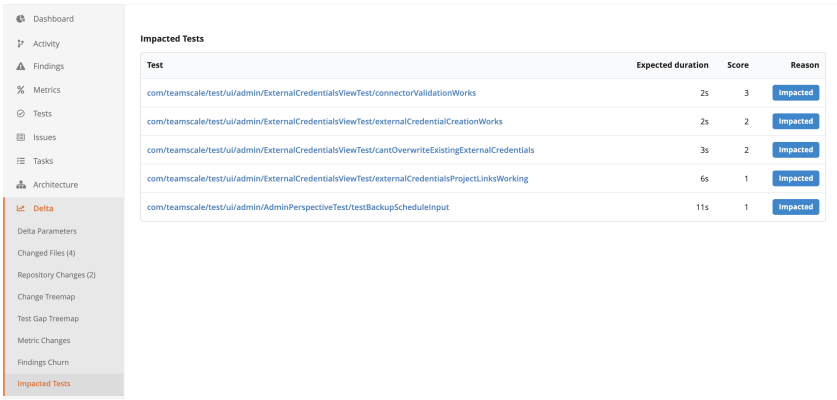


Figure 3.45. Impacted tests in the Delta perspective.

testwise coverage report has been uploaded for a timestamp that lies before the end of the inspected timeframe.

The impacted tests shown in the delta perspective can be used to verify that everying is set up correctly on the server side. It can also be used as an utility to support manual test selection.



4. User Reference

This chapter provides a user reference. It ought to provide a step-by-step guidance to facilitate the usage of *Teamscale*. It describes the different perspectives of the web interface (Section 4.1), provides a guide to set up the IDE integration (Section 4.2) and explains *Teamscale*'s different extension and integration APIs (Section 4.3).

4.1 Web Interface

The web interface is designed with different perspectives, represented in the left navigation bar, see Figure 4.1. There is e. g. a **Dashboard** perspective, a **Findings** perspective or a **Metrics Perspective**. Each perspective consists of one or several pages, views and dialogs. In the top of the perspective is a navigation bar, where the project can be selected as well as a specific branch (if branch support has been configured).

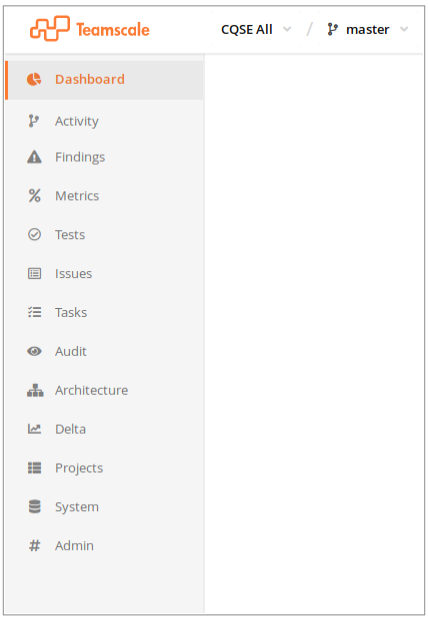


Figure 4.1. Perspective Selector and Settings Bar in the Web UI

4.1.1 General UI Features

Based on *Teamscale*'s general philosophy (see Chapter 1), there are several general features independent of the specific perspective. Most perspectives reveal a gray *perspective settings bar* at the top as shown in Figure 4.1.

Project and Branch In the perspective settings bar, you can choose the underlying project for each perspective. Additionally, if branch support is configured, you can select the corresponding branch, e. g. master or a specific feature branch.

Timetravel The perspective settings bar also contains the **Timetravel** feature. As *Teamscale* is connected to the version control system, it has the entire history of a project available. It does not only provide the historic version control data but also stores the history of corresponding metrics, findings and other quality data. The **Timetravel** feature allows you to travel back in time and view all available information for a specific date in history. You can select a calendar time, a revision or a baseline, see Figure 4.2. Figure 4.3 describes the different options.

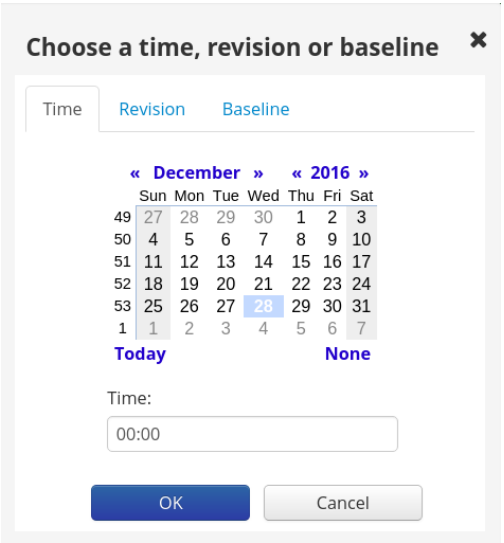


Figure 4.2. Timetravel

Timetravel Options

Time	You can select a time by picking a calendar date and a time, e. g. December 28th, 2016 at 0:00.
Revision	You can enter a specific revision number from your version control system if you have it at hand.
Baseline	You pick a baseline if you have preconfigured one in *Teamscale*.

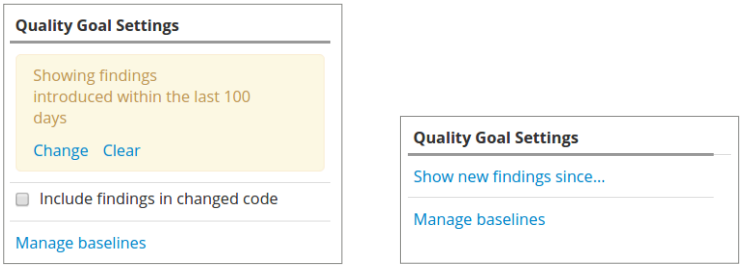
Figure 4.3. Using Timetravel Options

Quality Goals *Teamscale*'s philosophy relies on the concept of quality goals. As the overall set of findings of a grown software system is

usually too large to be managed, *Teamscale* provides quality goals as a filter mechanisms reducing the number of findings which are shown in the underlying view or page.

If no quality goal is set, all findings are shown. For the *preserving* quality goal, only *new* findings since a specific *baseline* are shown. For the *improving* quality goal those *new* findings are shown along with existing findings in code which was *changed* since the baseline. For the *perfect* quality goal, all findings are shown. Please refer to Section 1.2.2 to learn more about the use of quality goals in a quality control process.

In the **Findings** perspective as well as in the *Code Detail View* when viewing individual files, the right sidebar provides a configuration for the quality goal. If you want to configure quality goal *preserving*, click on **Show new findings since...** and choose a baseline. To configure quality goal *improving* additionally mark the checkbox **Include findings in changed code**. Note, that this checkbox only appears after the *preserving* quality goal has already been configured, see Figure 4.4.



(a) Quality Goal On (b) Quality Goal Off

Figure 4.4. Setting a Quality Goal

In the reminder of this section, each perspective is described in detail.

4.1.2 Dashboard Perspective

Use Cases	
	Creating and Viewing Quality Status Dashboards
Views	
	Dashboard View, Dashboard Edit View, Template Management View
Actions	
	Creating Dashboard, Editing Dashboard, Sharing Dashboard, Time Travelling, Managing Templates

After logging in, *Teamscale* puts you into the Dashboard perspective. A dashboard is a way to get a quick overview on a project's current state. Dashboards are configurable collections of widgets, with each widget showing a specific project characteristic, e.g. the system size in lines of code or the distribution of method lengths.

Dashboard View Figure 4.5 shows the **Overview Dashboard** for an example project, depicting various kinds of metrics. The first row in the top left consists of three differently configured numeric metric value widgets, showing the current values for the lines of code, clone coverage, and findings count (Findings Count), respectively, as well as the trend information with regards to the last seven days. Below are two *Assessment Pie Chart* widgets for method length and nesting depth. They show how the code base is distributed into methods that are assessed red, yellow, or green with regards to the respective metric. The *Metrics Trend Chart* widget in the bottom left shows the evolution of metric values over time, namely lines of code and *Number*

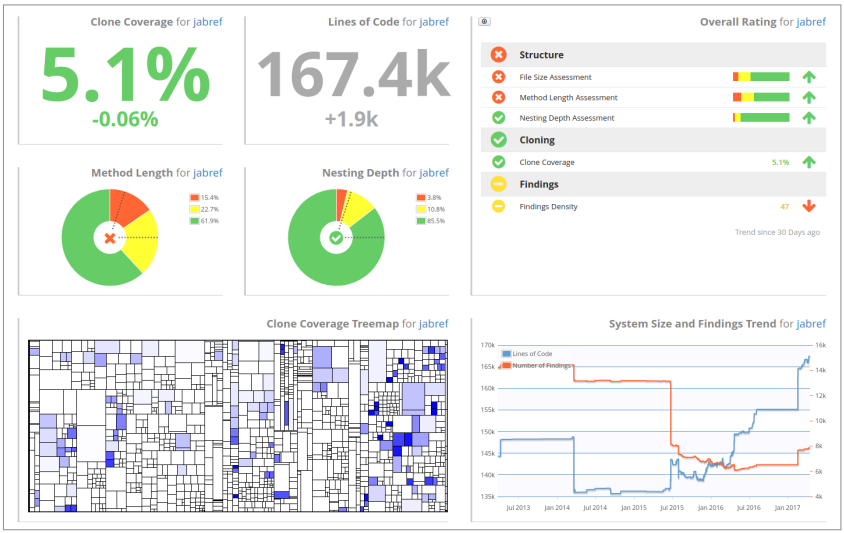


Figure 4.5. Overview Dashboard

of Findings. While there a large variety of Widgets to choose from, *Teamscale* comes with preconfigured templates to provide a starting point for customization.

On the top right, the *Findings Summary Bar Chart* shows how the identified findings are distributed over the available quality indicators and finding groups. Finally, the treemap widget on the bottom right shows how the metric values for clone coverage are distributed over the code base. This makes it very easy to identify hotspots with regard to the selected metric.

4.1.2.1 Action: Creating a Dashboard

To create a new dashboard, you can use the button at the bottom of the dashboard sidebar, which allows you to create a new, empty dashboard, or start with a template (Figure 4.6).

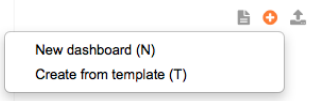


Figure 4.6. Add New Dashboard Menu

You can also import a dashboard, see Figure 4.7. After clicking the button, selecting *New dashboard* will present you with an empty dashboard canvas and a list of widgets that can be added to the dashboard. See the following subsection for more information about how to edit a dashboard. Selecting *Create from template* presents you with a dialog to configure a dashboard from template (see Figure 4.8). Here, you can enter the dashboard's name, select the template to create the dashboard from, and select the project and path that should be used by all widgets to display their data for. After confirming with OK, the newly created dashboard will appear in the dashboard sidebar, and you can share or edit it (Figure 4.9).

4.1.2.2 Action: Editing a Dashboard

To edit an existing dashboard, select **Edit dashboard** in the dropdown menu next to the dashboard's name in the sidebar of the perspective. This will lead you to the *Edit Dashboard View*. You will be in the same view right after creating a new dashboard as described in the previous Subsection.

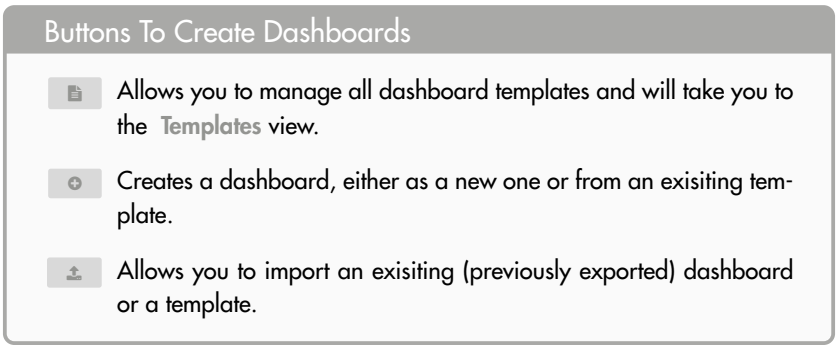


Figure 4.7. Buttons in the Sidebar of the **Dashboard** Perspective

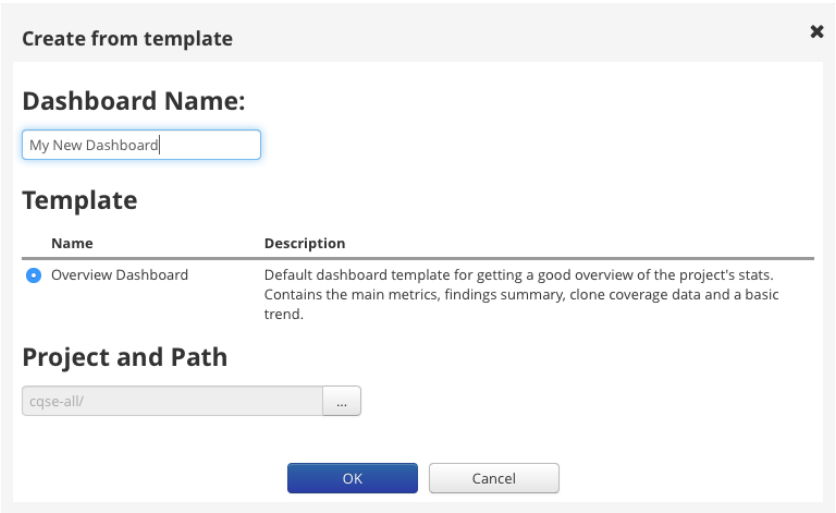


Figure 4.8. Create Dashboard from Template Dialog

Edit Dashboard View Once you are in the dashboard edit mode , the dashboard becomes editable, and the sidebar changes to a list

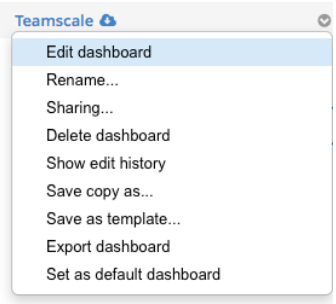


Figure 4.9. Dashboard Sidebar with Context Menu

of buttons. Clicking either of them adds a new widget of the selected kind to the dashboard. As of *Teamscale* 3.0, there are 26 Widgets to choose from, see Figure 4.10. All widgets are also listed in Appendix A. Which widgets to choose strongly depends on the particular use case.

Most widgets are configurable to a large degree, e.g. by selecting which metric(s) to show, which path to consider, the applicable timespan and many more options. Each widget on the canvas is enhanced with a row of buttons to enable configuring, duplicating, or deleting a widget (Figure 4.11 and Figure 4.12). To move a widget across the dashboard, you can drag it via the orange bar on the left. Resizing is possible using the orange corner on the bottom right (see Figure 4.11).

Widgets differ widely in the parameters they provide. Therefore, we illustrate the general concept rather than each individual widget here. Figure 4.13 shows the *Numeric Metric Value* widget's parameters as an example. All widgets share the **Title** and **Path** parameters. These can be used to alter the widget's title and change the project and path for which to display the data. If the path should not be shown on the widget, you can hide it using the check box. Further parameters

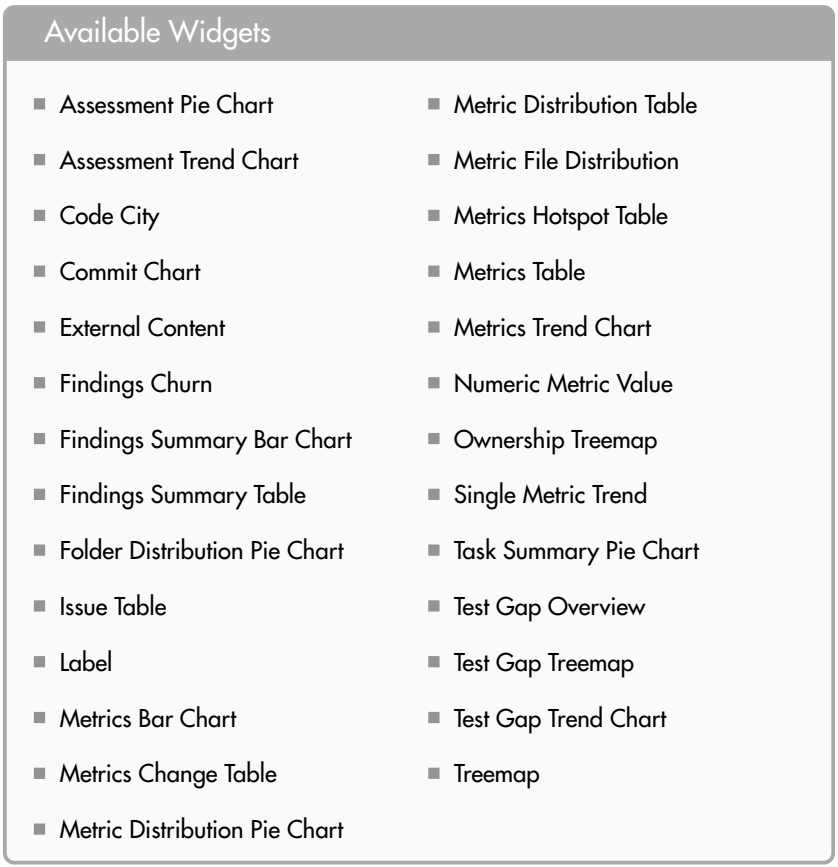


Figure 4.10. List of Widget Types

are provided on a per-widget basis. Usually, they provide tooltips with further information on how to use them when you hover over the parameter name. When you are done editing the widget parameters, OK closes the dialog and applies the changes to the widget.

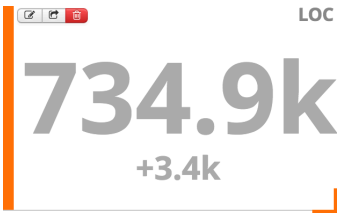


Figure 4.11. Dashboard Widget in Edit Mode

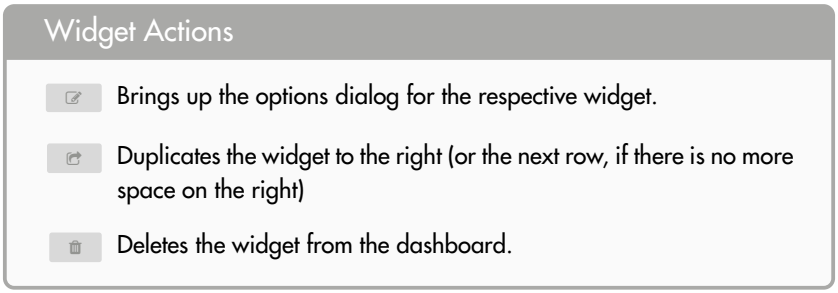


Figure 4.12. Buttons to Edit a Widget

4.1.2.3 Action: Saving and Sharing a Dashboard

When you are done editing a dashboard, you can save it via the Save or Save as buttons on the bottom of the sidebar (Figure 4.14). The former will overwrite the current dashboard, while the latter will create a new dashboard.

Once your dashboard is configured to suit your needs, you may want to share it with you team members. *Teamscale* makes it easy to do just that. A click on Sharing... in the dashboard's context menu in the sidebar (cf. Figure 4.9) brings up the *Sharing View*, where you can configure access rights for individual users or groups. Figure 4.15

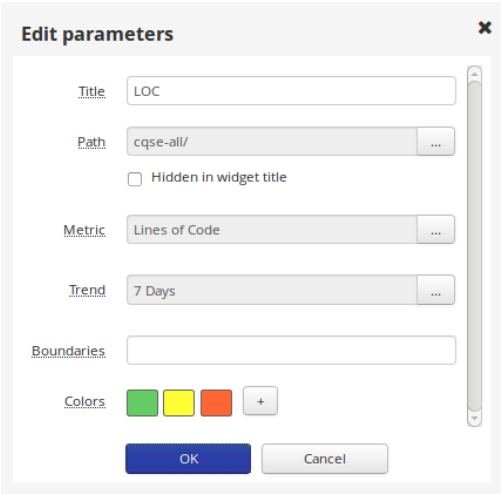


Figure 4.13. Edit Parameters for a Numeric Metric Value Widget

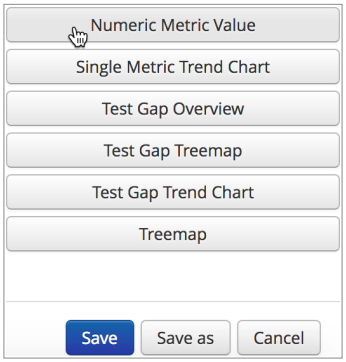


Figure 4.14. Saving a Dashboard

shows the dialog to add a new group. Notice that the input fields for users and groups provide auto-completion as you type, so in order to share your dashboard with a group called »CQSE Developers«, you can just type »Dev« and select the group from the suggestions.

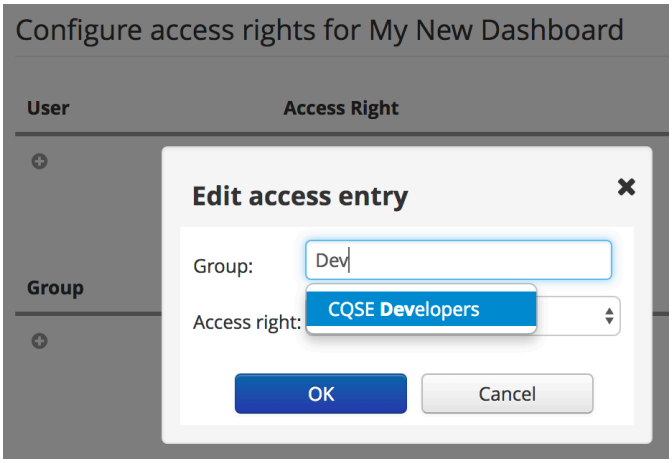


Figure 4.15. Dashboard Sharing

For every user or group you add to your dashboard, you can select whether they should be granted read or write access. When you share your dashboard with colleagues, it will appear in their dashboard sidebar. In your sidebar, private dashboards are shown with , while shared ones have a next to them. Other users' dashboards shared with you appear with a .

The initial creator of a dashboard is set as the *owner* who has full access rights for the dashboard. You can transfer the ownership of the dashboard by selecting Change owner from the context menu.

4.1.2.4 Action: Timetraveling in a Dashboard

In general, dashboards can not only show a system's current state, but can go back to any point in its history. The Dashboard perspective has the timetravel feature available as described in Section 4.1.1. After selecting a date in the Timetravel dialog, the dashboard will

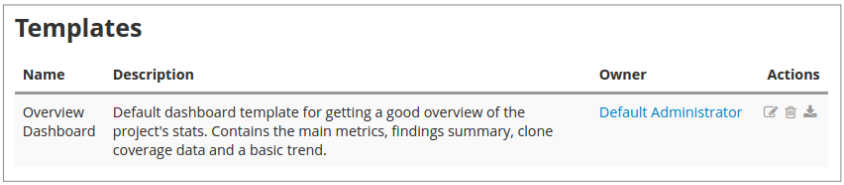


Figure 4.16. Template Management View

display the system's quality status at the selected point in time. Setting the timetravel will update the content of all widgets in the selected dashboard. It will not be transferred when you switch dashboards.

4.1.2.5 Action: Managing Templates

The dashboard perspective also allows you to manage all templates. You can click on at the bottom of the right sidebar in the *Dashboard View*. This will take you to the *Template Management View*.

Template Management View Figure 4.16 shows the *Template Management View*. It currently contains only one dashboard, the default **Overview Dashboard** along with a short description. To create new templates, you can go back to the *Dashboard Overview View* and use the dropdown menu right next to an existing dashboard's name in the right side bar. This drop down menu has already been shown in Figure 4.9. It contains the option **Save as template** which will create a new template based on the layout from the existing dashboards. The *Template Management View* further allows you to edit, delete and export a template with the buttons next to each template.

4.1.3 Activity Perspective

Use Cases	
	Viewing the History and Inspecting Single Commits
Views	
	Activity View, Commit View, Issue View
Actions	
	Browsing History, Inspecting Commit

For projects with a connector to a version control system, the **Activity** perspective provides an overview of the history. When you open the **Activity** perspective, you will first see the *Activity View*.

Activity View Figure 4.17 shows with an example how the history is presented in the *Activity View*. The view visualizes each commit to the repository and offers seamless scrolling through the history. In the given example, branch support was enabled for the underlying GIT repository. Hence, for the given branch, *Teamscale* shows *all* commits of *all* parents. The history of the branch and all branches that were merged into it is visualized on the left side of the commits.

On the right side, you can click on **Repository Details** to inspect the details of the analyzed history, see Figure 4.18. It shows you the number of changes performed on the repository within the last 24 hours or the last 7 days, for example. Note that only changes performed on the scope of your project are counted. The overall repository might, of course, have more changes. Additionally, the

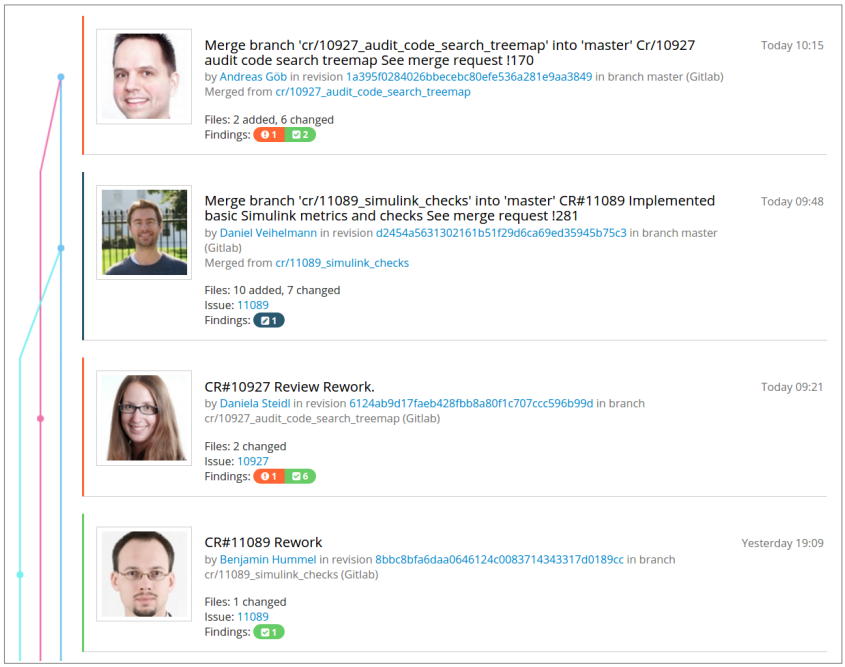


Figure 4.17. History of Commits

time range of the changes is indicated. The **First change** represents the first change within the history time interval as configured in your project configuration. For example, if your project's history dates back to the year 2000 but the project configuration indicates January, 1st 2016 as start revision, then you will see the latter as date of **First change**.

4.1.3.1 Action: Browsing the History

To browse the history, you can either just scroll through the list of commits or you can use the **Timetravel** feature to jump back in time.

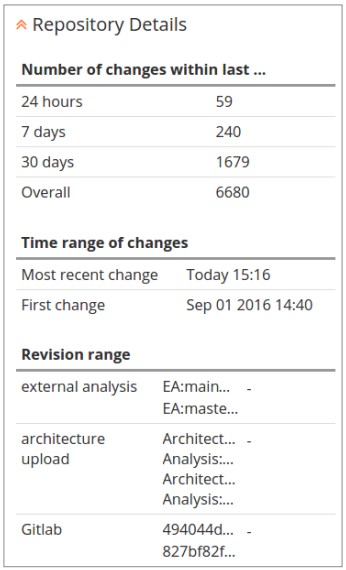


Figure 4.18. Repository Details in the Activity View

If you set your timetravel to January, 1st 2016, then all commits after this date will be hidden and your stream of commits starts with the last commit on January, 1st, 2016. From there on, you can scroll backwards again.

Additionally, several filter options are available to configure which commits are shown or hidden, see Figure 4.19. You can filter commits without findings churn or hide external upload commits. Figure 4.20 shows such external upload commit created by the nightly build to upload test coverage data.

The **User Filter** offers you to enter a user name. After entering the first letter, a drop down menu facilitates selecting the correct user. Afterwards, only commits from the selected user will be displayed.

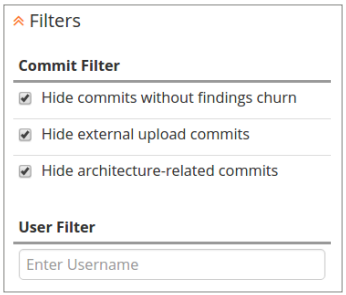


Figure 4.19. Commit Filters



Figure 4.20. External Upload Commit from Nightly Build

4.1.3.2 Action: Inspecting a Single Commit

From the *Activity View*, you can also inspect single commits in more detail. As a short summary, the commit is already annotated with the findings churn, showing the impact of the commit on the system's quality. If you click on the commit message, you will be directed to the *Commit View*.

Commit View The *Commit View* repeats the information about the commit message, the author, the revision number and the branch (if branch support is enabled). Additionally, it links to a connected issue tracking system, if one has been configured. Following the link, you will see the *Issue View* as depicted in Figure 4.21. From there, you can also navigate to the corresponding issue in the issue tracker.

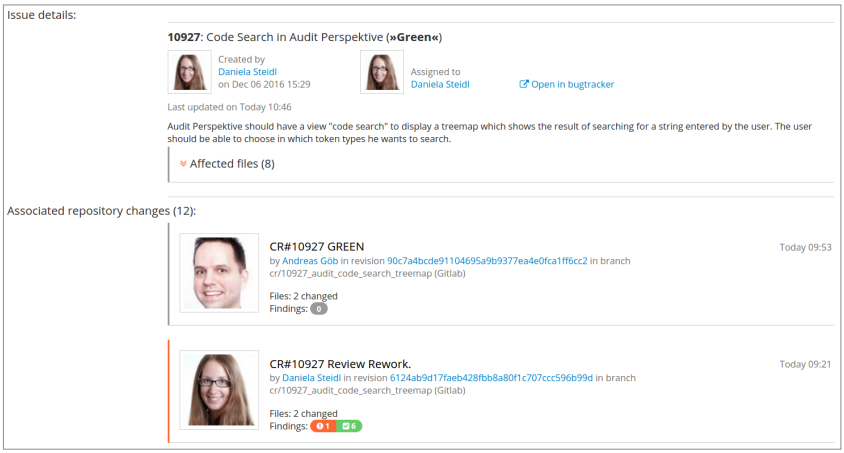


Figure 4.21. Issue View

Affected files (8)

Path	Change type
engine/com.teamscale.index/src/com/teamscale/index/audit/services/CodeSearchTreeMapService.java	added
engine/com.teamscale.index/src/com/teamscale/index/code_cities/CodeCitiesService.java	changed
engine/com.teamscale.index/src/com/teamscale/index/treemap/TreeMapService.java	changed
engine/com.teamscale.index/src/com/teamscale/index/treemap/TreeMapServiceBase.java	changed
engine/com.teamscale.ui/src/com/teamscale/ui/audit/CodeSearchView.js	added
engine/com.teamscale.ui/src/com/teamscale/ui/audit/TeamscaleAuditPerspective.js	changed
engine/com.teamscale.ui/src/com/teamscale/ui/base/TeamscaleBaseJSModule.java	changed
engine/com.teamscale.ui/src/com/teamscale/ui/base/TeamscaleServiceClient.js	changed

Figure 4.22. Affected Files By a Commit

Furthermore, the *Commit View* provides you information which files were affected by the commit (see Figure 4.22). The **Change Type** indicates you whether the file was **added** or **deleted**. If it was **changed**, you can use the link to jump into the *Code Comparison View*. This view will show you a comparison of the current version of the file and the oldest version prior to the commit. The view itself is described in Section 4.1.5.

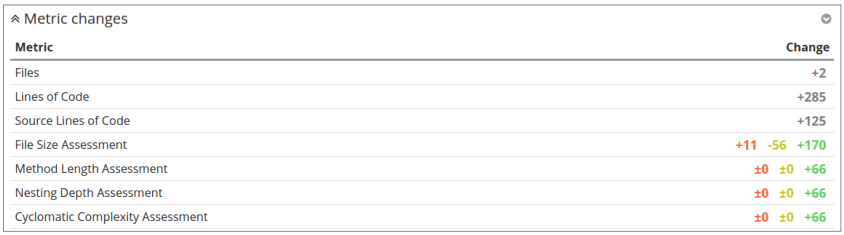


Figure 4.23. Metric Changes for a Commit

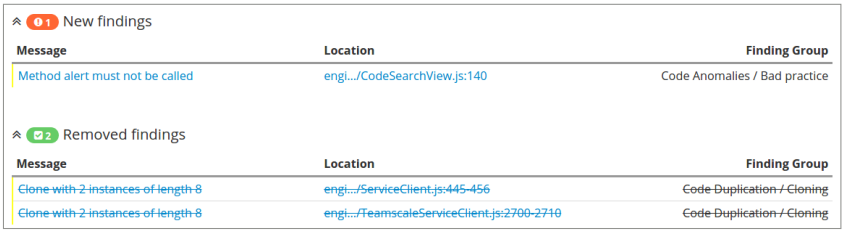


Figure 4.24. Findings Churn of a Commit

Underneath the affected files, you can expand the **Metric Changes** to see the impact of the commit on individual metrics. Figure 4.23 shows the corresponding table. Furthermore, the findings churn of the commit is indicated by a list of new and removed findings, see Figure 4.24. You can either click on the message of a finding to jump into the *Findings Detail View* of the **Findings** perspective or use the link to the source code location to jump into the *Code Detail View*.

4.1.4 Findings Perspective

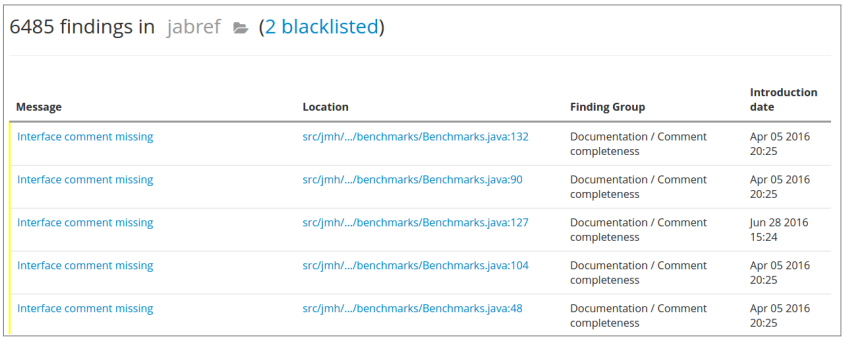
Use Cases	
	Inspecting and Filtering Findings
Views	
	Findings Overview View, Findings Blacklist View, Findings Detail View, Findings Introduction View, Baseline Management View
Actions	
	Navigating All Findings, Inspecting One Finding, Creating a Baseline, Adding a Finding to a Task

The main use case of the **Findings** perspective is to inspect the findings of a system. Several filter mechanisms help to reduce the set of findings to a manageable size. For each finding, detailed information about its introduction (and removal) are available.

Findings Overview View When opening the **Findings** perspective, the *Findings Overview View* shows a table with all findings for a *Teamscale* project, see Figure 4.25. For each finding, the table provides information about the finding's message, the finding location, the finding group as well as the introduction date.

4.1.4.1 Action: Navigating All Findings

For grown software systems, the table of the *Findings Overview View* usually contains a very large number of findings. To better inspect and



6485 findings in jabref (2 blacklisted)

Message	Location	Finding Group	Introduction date
Interface comment missing	src/jmh/.../benchmarks/Benchmarks.java:132	Documentation / Comment completeness	Apr 05 2016 20:25
Interface comment missing	src/jmh/.../benchmarks/Benchmarks.java:90	Documentation / Comment completeness	Apr 05 2016 20:25
Interface comment missing	src/jmh/.../benchmarks/Benchmarks.java:127	Documentation / Comment completeness	Jun 28 2016 15:24
Interface comment missing	src/jmh/.../benchmarks/Benchmarks.java:104	Documentation / Comment completeness	Apr 05 2016 20:25
Interface comment missing	src/jmh/.../benchmarks/Benchmarks.java:48	Documentation / Comment completeness	Apr 05 2016 20:25

Figure 4.25. Findings Overview Table

navigate them, the sidebar (Figure 4.26) provides several different filter mechanisms and navigation helpers.

First, with the **Quality Goal Settings**, you can configure the underlying quality goal of the project. If a quality goal is set, only findings relevant to the quality goal will be shown in the table. The table will then be annotated, e. g. with **since Dec 06 2016 00:00**. Configuring the quality goal as well as its implication was described in Section 4.1.1. The **Quality Goal Settings** also provide the opportunity to manage the baselines which will be explained in a later action.

Second, the **Finding Filter Settings** give you the opportunity to use an arbitrary regular expression. Hitting »Enter« will apply the regular expression to the finding's message as well as to its location. The table then shows only findings matching the regular expression. For example, entering »test« will show findings with the message »Avoid ignoring JUnit tests«.

Third, **Show Assessments** offers to filter findings based on the finding severity. You can either show only red findings, only yellow findings

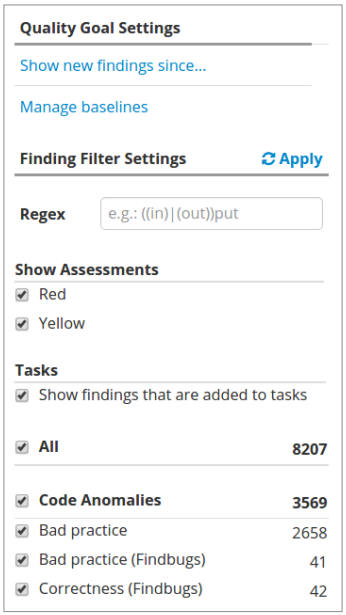


Figure 4.26. Filter Options

or both. Under **Tasks**, you can configure to exclude findings which have been added to a task already.

Fourth, the rest of the sidebar can be used to filter based on finding category and finding group. After configuring the check boxes, you have to use the **Apply** button in order to apply the filters to the table.

Furthermore, the content of the table can filtered based on a path. By clicking on in the table header, you can select subfolders of your project in a dialog opening up . After the selection, the table will show only findings from the specific subfolder. If you are working with an architecture definition, you can also select the architecture and pick a component or subcomponent. Then only findings in files will be shown that map to the selected component.

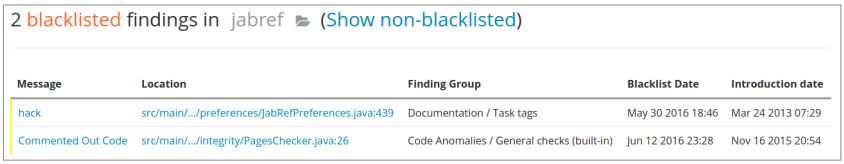


Figure 4.27. Findings Blacklist View

Additionally, the table can be sorted by the values of all four columns. Clicking on the header of one column will sort in descending order, i. e. showing the newest introduction dates or messages starting with »A« first. Clicking the column header again will reverse the order.

Findings Blacklist View Additionally, the table provides a link in its header to the findings blacklist. Clicking on blacklisted will open the *Findings Blacklist View*, Figure 4.27. The view is designed in analogy to the *Findings Overview View*. The table additionally shows the **Blacklist Date** for each finding denoting the date on which the finding was set on the blacklist. In case branch support was enabled for the underlying project, a finding is blacklisted only for the current branch. If the branch which contains the blacklisted finding is merged, the blacklisting is transferred to the other branch.

4.1.4.2 Action: Inspecting a Finding

The table provides several possibilities to obtain more information about a single finding. For example, clicking on the finding location will open the *Code Detail View* and show the source code file containing the finding. Clicking on the finding's message opens up the *Finding Detail View*.

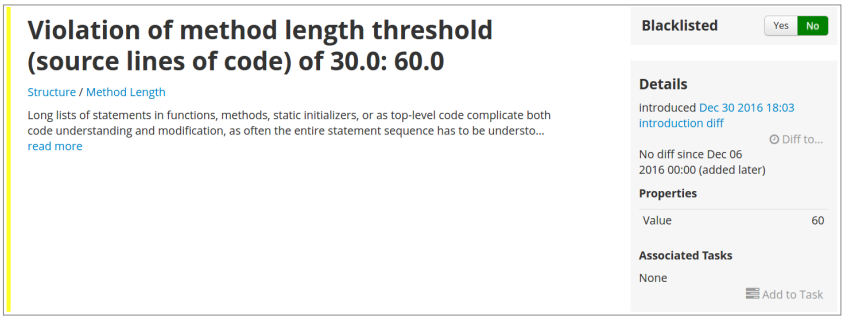


Figure 4.28. Findings Detail View

Findings Detail View This view provides you detailed information about a single finding, see Figure 4.28. It shows you the finding's message again as well as more detailed information about the impact of this type of finding on code quality. On the right, you can set the finding on the blacklist if you consider it to be irrelevant and you wish to no longer see it.

The **Details** provide you more information about the finding. You can view, e. g. its property values or add it to a task which is described in the following in a separate action. Often, you might be curious how a certain finding was introduced. Clicking on the blue date will open up the *Findings Introduction View*.

Findings Introduction View This view shows you how a certain finding was introduced. It comprises a comparison view of the last revision before the finding was introduced on the left an the introducing revision on the right, Figure 4.29. In green, lines are shown which were added. Blue denotes changes to existing lines and red indicates removed lines (not shown in the picture).

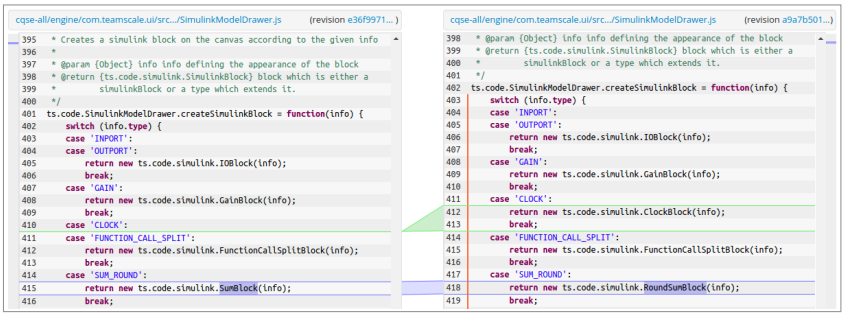


Figure 4.29. Introduction Diff View for a Finding

Back in the *Findings Detail View*, you can also use the Diff to... button and manually choose an arbitrary revision to compare against. This will show you the differences between the revision in which the finding was introduced and the revision you have just chosen.

4.1.4.3 Action: Creating a Baseline

As described in Section 4.1.1, *Teamscale* provides the concept of quality goals to manage potentially large amounts of findings. Quality goals *preserving* and *improving* involve defining a baseline – a point in time after which a finding is considered to be *new*.

To create a new baseline, the sidebar in the *Findings Overview View* provides the link **Manage baselines** under the heading **Quality Goal Settings**. The link opens the *Baseline Management View*.

Baseline Management View This view serves to manage baselines, see Figure 4.30. It contains a table listing all existing baselines. Initially, this table will be empty.

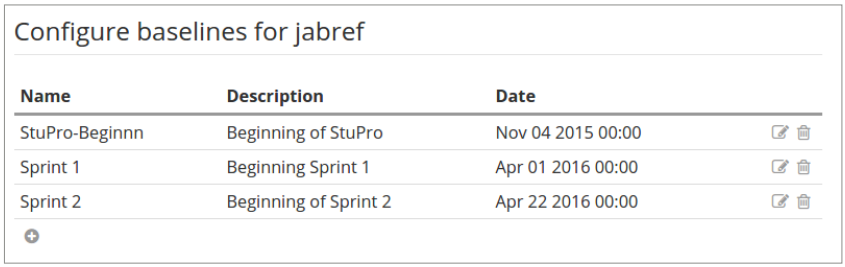


Figure 4.30. Baseline Management View

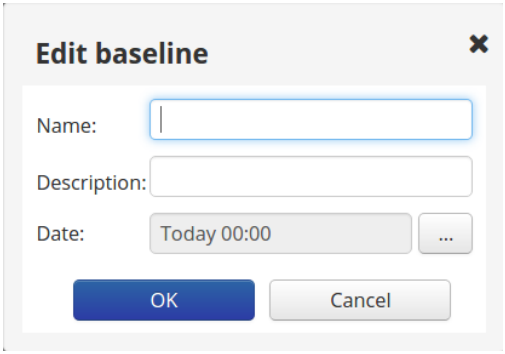


Figure 4.31. Adding A Finding To Task

With the + symbol, you can add a new baseline which will open up the *Edit Baseline Dialog* as depicted in Figure 4.31. You can create a baseline with a name and a date and a description to it. The same dialog will open up when you use the icon in the table to edit an existing baseline.

4.1.4.4 Action: Add Finding to Task

Teamscale comes along with its own task management. The **Findings** perspective provides you the opportunity to add findings to a task to track them[1].

In general, you can either add a single finding or a group of findings to a task. The *Findings Overview View* provides the blue link **Add findings to task** on the top right of the findings table. With this link, you can add all selected findings which are currently displayed in the table to a task. Note, that there is a size limit. Hence, you can add not more than 300 findings to a task. The link will open the dialog as depicted in Figure 4.32.

Alternatively, you can navigate to the *Findings Detail View* and use the button Add to Task to add a single finding to a task. This will open the same dialog. In the dialog, you can either create a new task or add the findings to an existing task.

[1] If you are curious how to use *Teamscale* in your task management in addition to your likely already existing issue tracker, please refer to our blog https://www.cqse.eu/en/blog/teamscale-tasks/

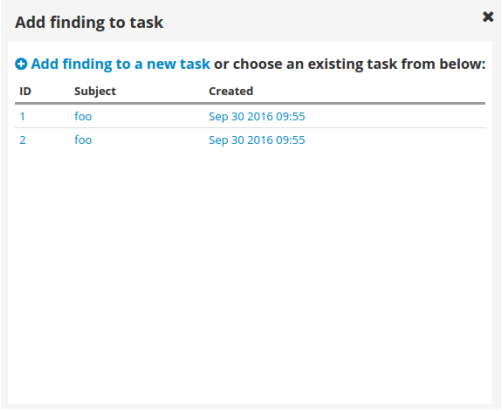


Figure 4.32. Adding A Finding To Task

4.1.5 Metrics Perspective

Use Cases	
	Viewing Metrics of Your System
Views	
	Metrics Overview View, Code Detail View, Code History View, Code Comparison View, Code Dependency View
Actions	
	Navigating the Code Structure, Configuring the Metrics Table, Viewing a Code File, Viewing the History of a Code File, Viewing the Dependencies of a Class

The **Metrics** perspective is the central place for metrics of your system. The perspective provides two different aggregation of the metrics. In the *Code* section, the metrics are aggregated based on the file directory hierarchy. In the *Architecture* section, they are aggregated based on architectural components. Additionally, the perspective also shows *Non-Code* metrics, such as build status or test execution.

Metrics Overview View When activating the **Metrics** Perspective, *Teamscale* shows the *Metrics Overview View* (see Figure 4.33). This view contains up to three sections which present two different forms of aggregation and, additionally, the non-code metrics:

- **Code** In this section, you can view metrics aggregated based on the code directory structure.



Figure 4.33. The Metrics Perspective

- **Architectures** For each architecture file under quality control, this section of the perspective shows metrics aggregated based on components from an architectural definition.

- **Non-Code** You can configure *Teamscale* to show non-code metrics like build execution times (see Section 3.1.2). If you are using this feature, this section of the perspective will provide a quick overview of such metrics. Please note that the section will be hidden if there were no non-code metrics configured.

Each of these three sections can be used as a starting point to navigate deeper into the metrics' hierarchy. From the code table, you descend into folders, whereas the architecture section table allows you to descend into subcomponents of your system. Both ways will, at their lowest level, result in an actual code file in the *Code Detail View*. To see this navigation in action, simply click on a folder name of the **Code** section to load the content of this package. In the following paragraphs, we take a closer look at the views you will encounter when you are browsing through the metrics. All of the three sections mentioned above work in an analogous way. In the **Code** section,

you can navigate and analyze the code in two dimensions – with respect to directory structure and time. By default, this section contains two elements:

- **Metrics Overview Table:** The *Metrics Overview Table* consists of one row for each file or directory in the current directory. Furthermore, it consists of several columns showing metric values for these files and directories. The first line of the table contains an aggregated value for the metric in the respective table column (e.g., the total number of files in the directory and all sub-directories).
- **Configure Table:** The *Configure Table* button opens a dialog that lets you change which metrics are shown in the table, how they are assessed, and in which number format they are presented (see Section 4.1.5.2).

In addition, when you start to browse deeper into the directory structure of your code base, you will encounter two additional elements:

- **Bread crumbs:** The bread crumbs show the current position in the code hierarchy, i.e. the current directory in a file system-like structure. Clicking them navigates to ancestor folders of the hierarchy. The *Metrics Overview Table* shows the content of the current directory.
- **View Findings Button** A click on View Findings will show you all findings stemming from the files you are currently viewing.

4.1.5.1 Action: Navigating the Code Structure

As mentioned before, each of the tables in the **Metrics** perspective – *Code, Architectures, Non-Code*– can be used as a starting point to navigate deeper into the respective directories. Clicking on a direc-

tory will update the *Metrics Overview Table* with the content inside the selected directory. Clicking on a file displays its code in the *Code Detail View*. Navigating back to a top folder can be done by the bread crumbs.

While navigating the code structure, the *Metrics Overview Table* can also be configured. You can click a column head to sort all entries according to the metric values in the corresponding column. For example, you can sort directories and files by their size in lines of code. You can also configure which metrics are shown in the table (see Section 4.1.5.2). Which metrics are available depends on the analysis profile which is configured in the **Project** perspective (see Section 4.1.11).

The table combines the directory structure with change information from its history. The column **Last Change Date** shows you when the respective directory or file was changed last. By clicking the value in column, you jump to the **Activity** Perspective (see Section 4.1.3) and the corresponding activity, i.e. the corresponding commit, is shown.

You can also view each directory at a specific point in time. Use the **Timetravel** feature as displayed in Figure 4.1 to choose a date, baseline or revision. The content of the directory will be updated to the selected timestamp.

When you are browsing through your source code, it is possible to view all findings for the directory currently visible. Simply click View Findings above the table or press »F«. All findings for the current directory will be displayed in the **Findings** perspective (see Section 4.1.4).

4.1.5.2 Action: Configuring the Metrics Tables

For each table visible in the **Metrics** Perspective, you can choose which metrics should be shown and how they should be assessed & presented. You can set these options by clicking on **Configure Table** directly above the corresponding table. This will open up a dialog as depicted in Figure 4.34.

- **Number Formatting:** The checkbox in the *Number Formatting* area allows you to switch whether numeric values should be shown in full (e.g., 1,200) or whether they shall be rounded using appropriate SI prefixes (e.g., 1.2k).
- **Threshold Profile:** The threshold selector lets you choose which of the configured threshold configurations should be used to assess metric values. You can create and edit threshold configurations in the **Project** perspective (see Section 4.1.11).
- **Metrics to show:** Simply check all metrics that should be presented in the table. To quickly find a metric, you can use the search bar above the list. Note that you can also quickly remove table columns by hovering over a column head and clicking the »x« that will appear in the top-right corner.

Which metrics are available depends on the analysis profile which is configured in the Project Perspective (see 4.1.11).

For each available metric, you can view its history trend, as well as a treemap that visualizes the metric distribution across all (code) files. To view them, click on a table cell showing a particular metric. This will display the history and the treemap of the metric, see Figure 4.35. For the history trend, a time range can be selected and viewed in **Delta** Perspective by clicking the arrow ⌄ and choosing option **View selected range in Delta Perspective** from the context menu (see

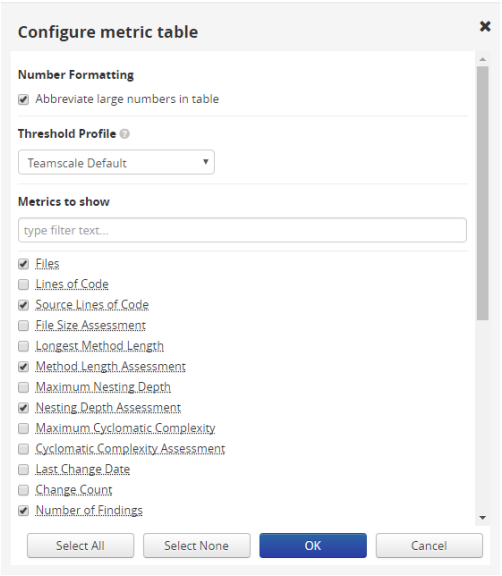


Figure 4.34. Configure Table Dialog for Metrics Table

the use case description in Section 3.4.2). A click on a rectangle in the treemap lets you view the respective file in the *Code Detail View*.

4.1.5.3 Action: Viewing a Code File

Often, browsing a directory or component structure ends when a certain source code file was found. Each code file can be viewed in the corresponding *Code Detail View*.

Code Detail View The *Code Detail View* (see Figure 4.36) appears after clicking on a file in the *Metrics Overview Table* and shows the source code text together with additional information. It consists of the *Bread crumbs* and a More button, the code itself and a sidebar.



Figure 4.35. History and Treemap for Metric SLOC

The sidebar contains the **Quality Goal Settings**, the **Metrics Table** as well as information about **Coverage Highlighting** (not pictured) and a **Findings List** (not pictured). Each element is described in more detail in the following:

- **Timetravel UI:** It shows the date for which the code is displayed.
- **Bread Crumbs:** It display the position of the current file in the code hierarchy.
- **More Button:** It allows you to perform contextual actions (e. g. to download or pretty-print a file) which are described below.

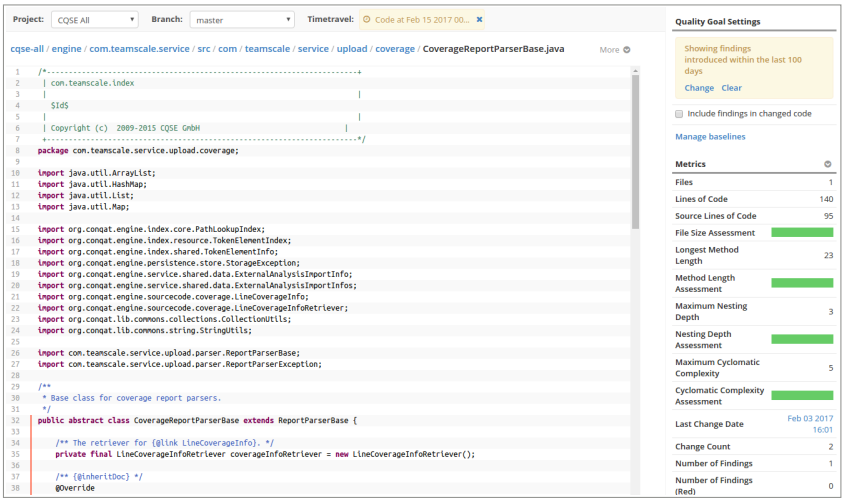


Figure 4.36. Code Detail View

- **Code Text:** This renders the code of the current file with syntax highlighting. The orange lines at the beginning of a code line are finding markers. They indicate that a finding for this line exists. Hovering over the orange line displays a finding description which can be used to jump to the **Findings** perspective (see Section 4.1.4) and see more details.

- **Quality Goal Settings:** The quality goal setting lets you decide which filters to put on findings for code of the current file, i. e. which findings are (not) to be displayed. Please refer to Section 4.1.1 to learn about the impact of quality goal settings on a view.

- **Metrics Table:** The table presents several metrics together with metric values for the current file.

- **Findings List:** This list contains all findings for the current file and can be filtered by clicking on **Show filters**.
- **Coverage Highlighting:** If coverage for the current file exists (i. e. is known to *Teamscale*), covered lines are indicated by a green background color. If available, coverage highlighting can be turned on or off by clicking on the check box **Annotate line coverage**.

Each code file can be viewed in all its revisions available in the history. Clicking the **Timetravel** and choosing a particular date, baseline or revision, shows the code version of this point in time.

To read the source code, developers often mark a specific code element to view all its occurrences. Clicking on a specific code element, e. g. a variable, highlights all occurrences of this variable. A marker appears on the right side of the code text visualizing where in the file the code element occurs (see Figure 4.37).

```
if (UserVisibleHint && LastReportedVisibility != ViewStates.Visible)
{
    OnShowFragment();
    LastReportedVisibility = ViewStates.Visible;
}
else if (!UserVisibleHint && LastReportedVisibility != ViewStates.Gone)
{
    OnHideFragment();
    LastReportedVisibility = ViewStates.Gone;
}
```

Figure 4.37. Highlighted Variable Occurrences in Source Code

The **More** button ⌄ provides several more option. Choosing option **Download** from the context menu, downloads the current file to the local device. Choosing option **Download anonymized** downloads an anonymized version of the current file (e.g., for bug reporting). Choosing option **Pretty print/ format view** from the context menu or pressing key »P«, pretty-prints the code. This does not change the original code but just the way it is formatted and displayed

in the *Teamscale* UI. Clicking **Resource History** (shortcut: »H«) allows you to view the history of the file at hand (see Section 4.1.5.4). Moreover, **Show dependencies** (shortcut: »D«) lets you view the (outgoing and incoming) dependencies of the current file (see Section 4.1.5.5).

4.1.5.4 Action: Viewing the History of a Code File

In addition to a particular version of a source code file, you can also use *Teamscale* to browse the history of a file. Whenever you are viewing an individual file in the *Code Detail View*, clicking **More** and then choosing the option **Browse history** from the menu (or pressing key »H«) will show you the history of the file in the *Code History View*.

Code History View This view (see Figure 4.38) presents the history of activities for a particular file. It consists of a list of commits which is ordered according to time. Each activity consists of a commit message, an author, a revision and a branch as well as a date.

The *Code History View* provides several possibilities. Clicking **Show content** of a commit, opens the *Code Detail View* and shows the code in the version of the selected commit. Clicking the commit message or the revision number displays the commit of the activity in the *Commit View* (Section 4.1.3). Clicking on **Compare with ...** of two commits opens the *Code Comparison View* and allows to compare two versions and identify differences.

Code Comparison View This view (see Figure 4.39) shows two files – usually the same file in different versions – next to each other and highlights differences between both. It displays the **Comparison Mode** at the top of the view, the older version of the code on left side and the newer version of the code on the right side.

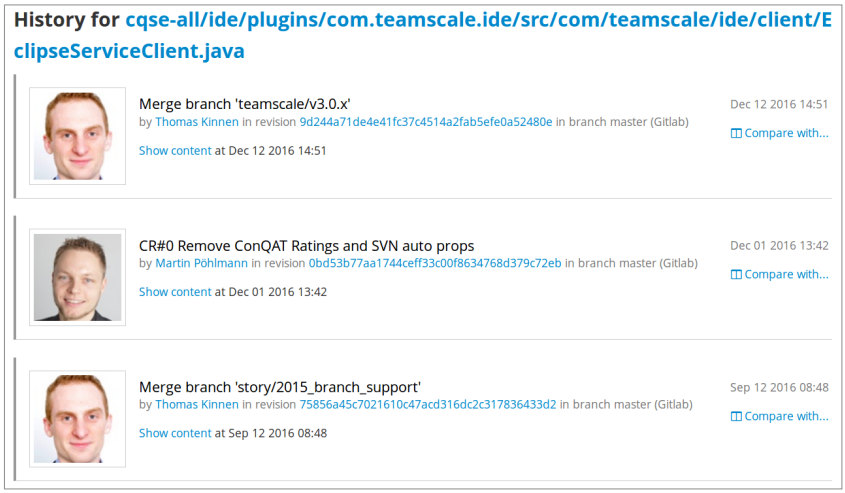


Figure 4.38. Code History View

- **Comparison Mode:** This indicates the current comparison mode. Comparison mode can be line-based, line-based (ignoring white space) and token based. Clicking another radio button changes the comparison mode.
- **Code on the Left:** This presents code of the older file. Additionally, it shows the path and the revision of the older file.
- **Code on the Right:** Similarly, this presents the code, path and revision of the younger file.
- **Visualization of differences:** This shows differences between the code on both sides. Blue differences denote differing code on both sides, green differences denote code only present in the left, older code and red differences denote code only present in the right, newer code.

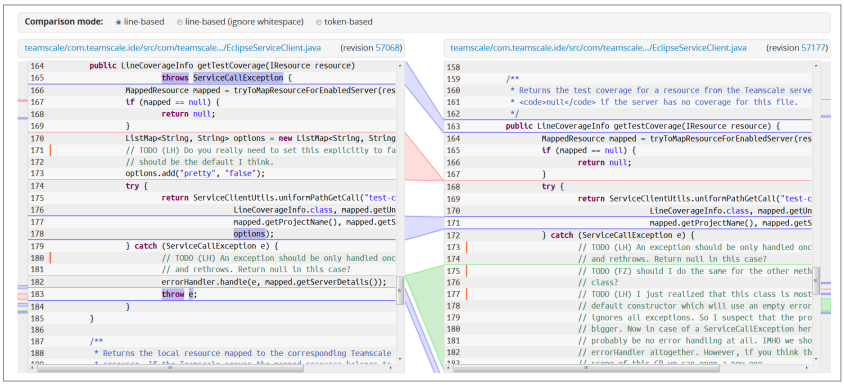


Figure 4.39. Code Comparison View

Scrolling or clicking the visual summary allows to navigate in the code and inspect differences. The shortcut »Alt+N « allows you to jump to the next difference, the shortcut »Alt+Shift+N « allows you to jump to the previous difference. Clicking the path on top of one side shows only one code version and opens the *Code Detail View*. Clicking the revision of one side shows the revision of in the **Activity** Perspective.

4.1.5.5 Action: Viewing the Dependencies of a Class

In the *Code Detail View*, clicking **More** and choosing option **Show dependencies** from the context menu or pressing key »D«, shows the dependencies of the current file. This will open the *Code Dependency View*. Please note that this only works if architecture analysis is activated in the analysis profile.

Code Dependency View This view (see Figure 4.40) shows outgoing and incoming dependencies of a file on other files as well as frameworks and APIs. Please note that this feature is available only



Figure 4.40. Code Dependency View

if architecture analysis is activated. The view consists of two lists, namely outgoing and incoming dependencies. Each dependency has a path and one or more numbers. The path denotes the file on which the current file depends (in case of outgoing dependencies) or the file which depends on the current file (in case of incoming dependencies). The numbers denote the lines of the current file which refer to the depending file (in case of an outgoing dependency) or the lines in the dependent file which refer to the current file (in case of an incoming dependency).

Clicking on the dependency path opens the dependency file and allows to inspect its code. Clicking on the line number jumps to the code location where a reference is made from or to the dependency file. This will open up the *Code Detail View* with the respective file.

4.1.5.6 Performing a Code Review

Teamscale supports a basic file-based review process. Therefor, the analysis group *Code review* contributes an assessment metric which assigns each file with an assessment corresponding to its current review status. As with every other assessment metric, you can view treemaps and trends for the review status of your code base. To enable the Code Review feature, edit the analysis profile of your project and add the analysis group *Code review*.

To set the review status for a file, navigate to the file in the metrics perspective and select a status from the *Code Review Status* dropdown menu. You can assign one of *In review*=yellow, *Needs rework*=red and *Done*=green). Besides the explicit status change, a change of the file's content will change its assessment to red indicating the need for a review of the changes. Please note that in multi-branch projects, the review status will not be transferred between branches upon merge operations.

Changes to the review status will create a *Code Review Commit* which is integrated in the history of the Teamscale project and thus also visible in the activity perspective.

The review feature also supports a baselining concept. To use it, you can define a review baseline date in the project options. As a consequence, files that were not changed after the baseline are assessed with a blue color for the code review assessment metric.

4.1.6 Tests Perspective

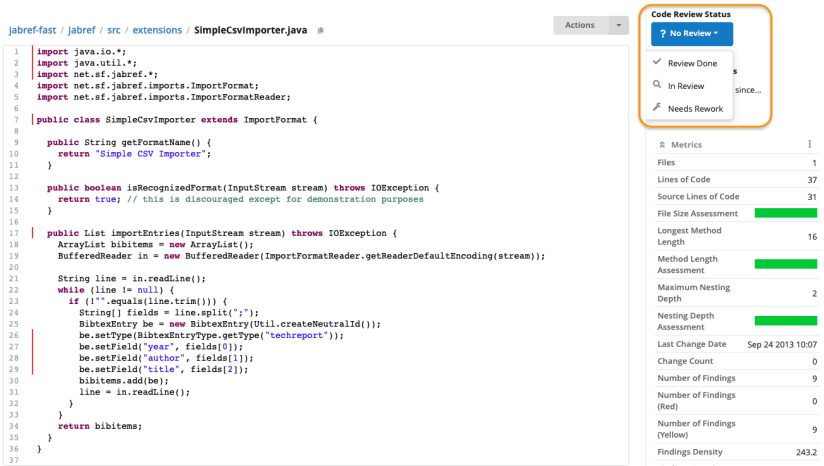


Figure 4.41. Setting a Review Status

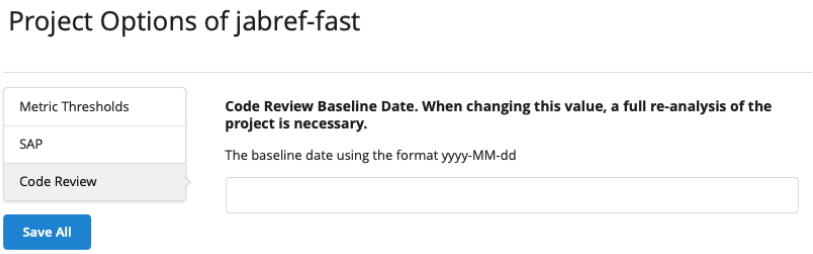


Figure 4.42. Code Review Baseline Date

Use Cases	
	Uncovering Test Holes, Viewing Test Execution Results
Views	
	Test Gap Overview, Code View, Issues View
Actions	
	Configuring *Teamscale* for Test Gap Analysis, Finding Test Gaps in Code, Tracking Test Failures

The **Tests** perspective assists you in assessing the quality of your tests. In particular, it is possible to detect potential test holes using the test gap analysis, which focuses on code that has been untested after its latest modification. You can also view the results of test executions to check for failing tests.

4.1.6.1 Action: Configuring Teamscale for Test Gap Analysis

Test Gap Analysis Generally, the Test Gap Analysis combines static information (from the code repository) and dynamic information (code coverage from test executions measured by a profiler such as JaCoCo). Whereas the static part comes out-of-the-box for every *Teamscale* project, the test coverage has to be uploaded to *Teamscale* as a separate step. For testing purposes, this can also be done manually via the **Upload Report** feature, as described in Section 4.1.11.3.

Actual changes vs. refactorings For detecting actual test holes, only non-trivial changes of the code are considered as »changes«. Otherwise, simple refactorings like variable renamings will falsely mark

a code fragment as »untested«, even though the changes preserved the original semantics of the code.

4.1.6.2 Action: Finding Test Gaps in Code

A test gap is a code region that has remained untested after its latest modification.

Test Gap Overview Once test coverage was uploaded to *Teamscale*, the **Test Gap Overview** serves as main entry point of the **Tests** perspective. It shows an overview of the testing-related metrics of the root level of the repository, as well as for each architecture configured in *Teamscale*. These metrics are:

- **Test Gap** Shows which shares of the code are unchanged (gray), changed & tested (green), changed & untested (yellow), and new & untested (red).
- **# Test Gaps** the total number of test gaps.
- **Execution** Visualizes which percentages of the code have been executed (green) and which have *not* been executed (gray) by the tests.
- **Churn** Shows the parts of the system that were changed or added, while disregarding all coverage information.

The metric descriptions are also available as tooltip of the corresponding header table cell.

If the *Code Structure* tab is selected, you can see the following UI components in addition to the table:

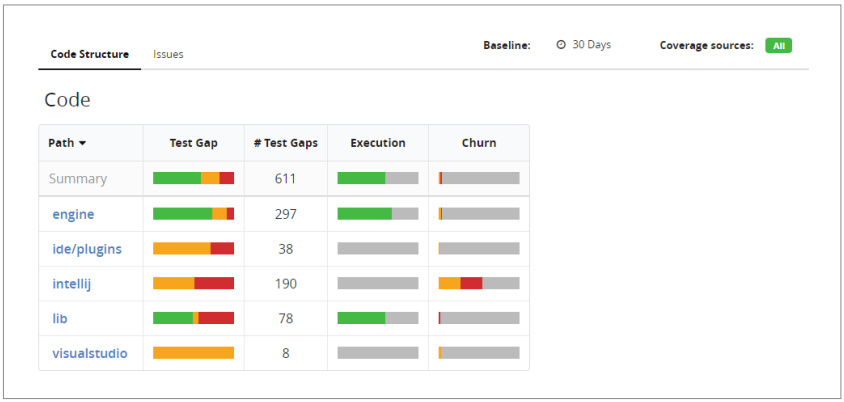


Figure 4.43. Tests Overview

- **Baseline** Defines the beginning of the time frame for which code changes should be counted (e. g. since the last release). The default is 30 days in past.

- **Coverage sources** Lets you choose which test partitions to consider for the coverage. As default, all partitions are selected. You can also select other Teamscale projects as coverage sources (»cross-annotation«). In that case, the coverage for all methods found both in the current project and the selected project are considered if the methods do not differ across the projects.

4.1.6.3 Action: Uncover test gaps in your code base

Code View The code view lets you navigate into individual folders and files while showing aggregated test-gap metrics. The **summary** row of the table contains the aggregated values for all of the shown folders and files. A click on one of the table cells will open an overview of the contents of the corresponding path (see Figure 4.44).

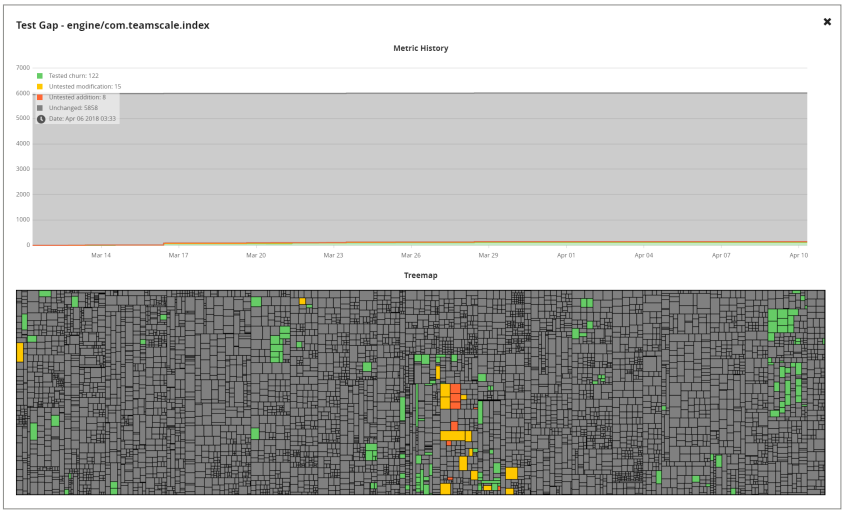


Figure 4.44. Dialog of Test Gap Table

The dialog shows two kinds of data:

- **Test Gap Trend** Shows the distribution of the selected test-gap metric over history.
- **Test Gap Treemap** Displays all methods within the given folder or file, and their corresponding test gap status. The size of a rectangle corresponds to the length of the method. In case an entry from the »Execution« column was selected, every method will either be marked as green (executed) or gray (not executed). Hover over the treemap rectangles with your mouse for details.

Once you click on one of the rectangles (i. e. methods) in the treemap, the file containing the method will be opened in *Teamscale*. To learn more, please refer to Section 3.5.2

4.1.6.4 Action: Tracking Test Failures

In addition to test coverage, you can also upload test execution results to *Teamscale* (see Section 4.1.11.3). These contain information about success or failure of a test case as well as meta-information such as execution time.

Teamscale will structure and display these results in a manner similar to the Metrics perspective (Section 4.1.5). This means that tests from the same package or file will be grouped together, and their results will be aggregated and displayed the same as with other code (or non-code) metrics. Navigating between different test results also works exactly the same as navigating between other metrics (see Figure 4.45).

You can see the trend of a test execution result by clicking on the relevant metric in the table. This allows for example checking the execution result history of a specific test case to find out when it failed in the past.

Issues View The Issues tab of the **Tests** perspective shows test gap information for each issue of a selected issue metric (see Section 4.1.7.3). For meaningful insights, the following prerequisites have to be fulfilled:

- An issue tracker has to be configured for the current project. Please refer to Section C for configuration details.
- Commits to the code base of the project specify the corresponding issue ID of the target issue in the commit message. For example, a commit message like »CR#1234 Added feature« could be mapped to an issue with ID 1234. For details, please check C.

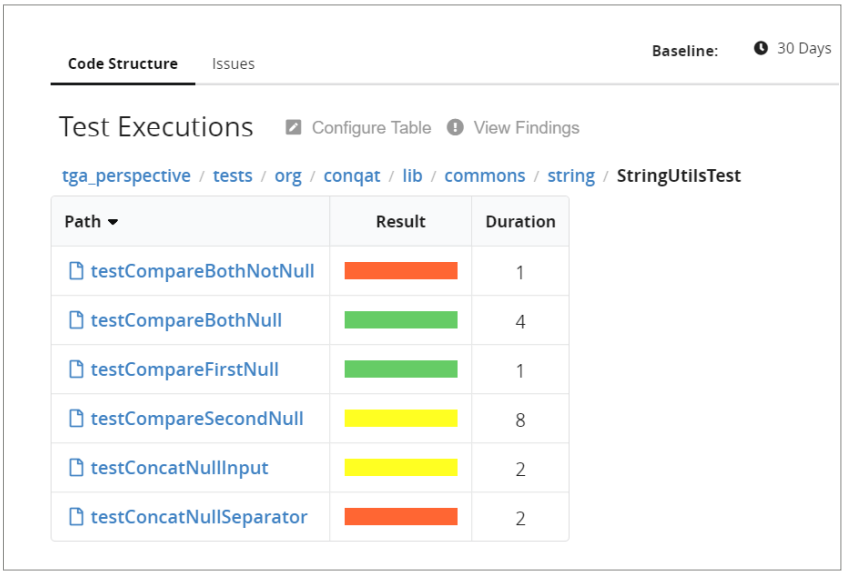


Figure 4.45. Viewing Test Execution Results

- Test coverage has been uploaded to *Teamscale*
- At least one *issue metric* was configured (see Section 4.1.7.3)

Once all of these prerequisites are met, selecting an issue metric from the dropdown within the **Issues** tab will show results similar to the screenshot in Figure 4.46. The bar in the right column visualized which percentage of the code edited for the corresponding issue was untested after its most recent (significant) change. A gray bar with the »na« label usually means to no files known to *Teamscale* were affected by the commits for the issue. A click on a table entry will show detailed information about the issue, including the test gap treemap, which represents a more fine-grained visualization of the test gap data (see Figure 4.47. You can learn more about this view in Section 4.1.7.4.

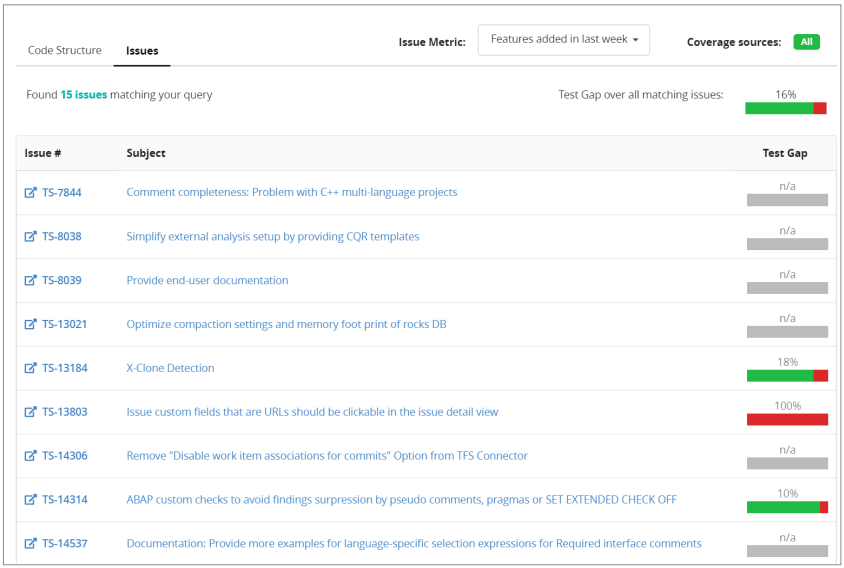


Figure 4.46. Issue overview in Tests perspective

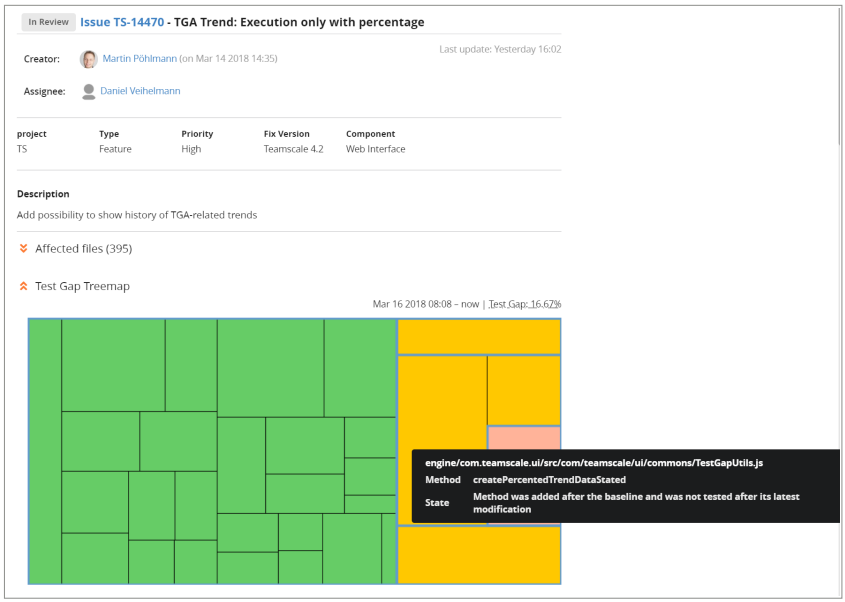


Figure 4.47. Test Gap Treemap in Issue Details

4.1.7 Issue Perspective

Use Cases	
	Evaluating Queries over Issues, Managing Issue Metrics
Views	
	Issue Query View
Actions	
	Query Issues, Display Issue Trend, Manage Issue Metrics

The main purpose of the **Issue** perspective is to calculate metrics over issues stored in an issue tracker. These metrics can reveal weaknesses in your development process, such as long pending critical bugs, and help you to assess any steps toward process improvement over time. To use this perspective, your project must define at least one issue tracker connector when configuring your project (see Section 4.1.11.2 and Appendix C).

4.1.7.1 Action: Query Issues

The **Issue** perspective allows you to perform queries against all issues imported into the Teamscale project to obtain lists of matching issues. Hence, you can, e. g. obtain the number of issues matching certain criteria, such as overdue bugs. For this, you navigate to the **Issue** perspective, which opens the *Issue Query View*.

Issue Query View In this view, you can enter queries (see Figure 4.53). For example, to display all open issues, enter *closed = false* into the query box. After pressing *Enter*, the list of matching issues is displayed below the box. The supported language for these queries is described in Figure 4.48. The input box also provides autocompletion to simplify writing queries.

Each issue in the returned list provides a link to additional details, as well as to the ticket in the originating issue tracker. You should keep in mind, that the default is to query all issues, including closed one. To limit a query to open tickets, add the phrase *closed=false*.

Issue Query Language

Attributes	For each ticket, the default attributes `subject`, `id`, `assignee`, `author`, `created`, `updated`, `status`, and `closed` are defined. Additionally, all custom fields defined in the connector configuration are made available as attributes. For connectors that support it, the `parent` field is filled with the ID of the parent ticket.
Comparison operators	Attributes can be compared to values using the usual comparison operators =, !=, >, <, >=, <=. The order is always attribute, operator, value. You can not compare attributes to each other.
Set query	You can use the keyword `in` together with square brackets to create set queries. An example would be `priority in [high, urgent]`
Logical operators	Comparisons can be connected using the logical operators && (and), || (or), and ! (not). Additionally, parentheses can be used to group subexpressions.
Like operator	The `like` operator ~ can be used to match a value against a regular expression.
membersOf	The special function `membersOf` can be used together with set queries to check for membership in a Teamscale group (that might again come from LDAP). An example would be `assignee in membersOf('UI Team')`.
currentUser	The special function `currentUser` can be used to select tickets where `author` or `assignee` is current Teamscale user. An example would be `assignee = 'currentUser()'`.
baseline	The special function `baseline` can be used to replace dates with a baseline. Note that the time of the baseline is ignored and is set to midnight i.e. just the date is used. An example would be `updated > baseline('Release 1')`.

Figure 4.48. Supported Language for Issue Queries

Issue Query Language (continued)

State queries	The operator `inState` allows to formulate that a ticket has to be in a given state for a longer time. The operator can contain any subquery and is followed by a comparison (>, <, >=, <=) with a time value. An example would be `inState(closed=false) > 7d` to display all tickets that are open for at least 7 days. The number can be followed by suffixes for days (d), hours (h), minutes (m) and seconds (s). The default when no suffix is given is days. Dates can also be used, for example: `inState(closed=false) > 2018-1-12` returns all issues, which have been closed after 2018-1-12. This is especially useful if you want to create a static query, e.g. all closed issues in January. Additionally, the special value `@startOfMonth` can be used to get the duration of time elapsed since the start of the current month, e.g. `inState(closed=false) < @startOfMonth` to display all tickets closed since the start of the current month.
Parent queries	The operator `hasParent` allows to select tickets based on the attributes of their parent ticket. It selects tickets that have a parent and whose parent matches a given sub query. An example would be `parent(closed = true)` to display all tickets whose parent ticket has been closed. Tickets that have no parent or whose parent was not imported into Teamscale are never returned by a `hasParent` query.

Figure 4.49. Supported Language for Issue Queries

An example returning all urgent tickets that have not been closed within two weeks would be

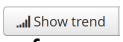
```
inState(closed=false && tracker=Bug && priority in [
    urgent, blocker]) > 14d
```

To limit this to tickets within the UI team, you can change this to

```
assignee in membersOf('UI team') && inState(closed=false
    && tracker=Bug && priority in [urgent, blocker]) > 14d
```

Note that the exact attributes will depend on your issue tracker and the fields that are configured. Also note the subtle difference between including the assignee check in the *inState* part or not. If we would include the assignee check in the inState part, this would cause tickets to not be shown if the assignee changed in the 2 weeks.

4.1.7.2 Action: Display Issue Trend

When an issue query has been executed, you can click on 

in the *Issue Query View*, to display a trend of the number of issues matching the query over time. An example output is shown in Figure 4.50.

4.1.7.3 Action: Manage Issue Metrics

There are certain queries which you might want to execute continuously and see the result regularly without re-formulating the query every time. Therfore, the Issue perspective also allows you to define issue metrics. An issue metric is the numeric result, i. e. the number of tickets matching a query. By defining a metric, the underlying query is saved in *Teamscale* and, thus, can be used continuously.

These issue metrics are displayed in the sidebar, as shown in Figure 4.51. To create such a metric, execute an issue query and click



Figure 4.50. Issue Trend

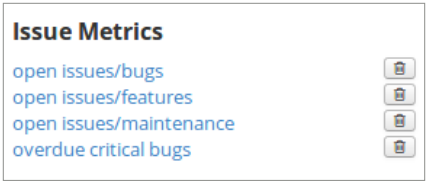


Figure 4.51. Issue Metrics Listed in the Sidebar

+ Add as metric in the *Issue Query View*. After entering a name, the metric will appear in the sidebar. The list of metrics in the sidebar allows you to quickly return to the corresponding query by clicking on its name, or to delete the metric by clicking . Editing a metric works by recreating a metric with an existing name.

The defined issue metrics are also available in the **Dashboard** perspective (Section 4.1.2) upon configuring dashboard widgets. The configuration of a widget offers you to define the scope of a widget by setting the corresponding path, which usually points to a (sub-) folder or (sub-) component of the system. To use issue metrics instead, use the path *-issues-* within a project. When the metric names

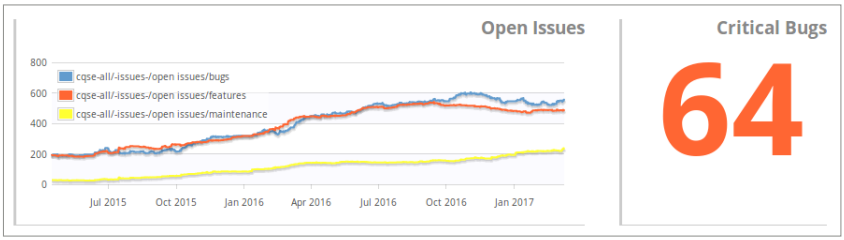


Figure 4.52. Issue Metrics in Dashboard Widgets

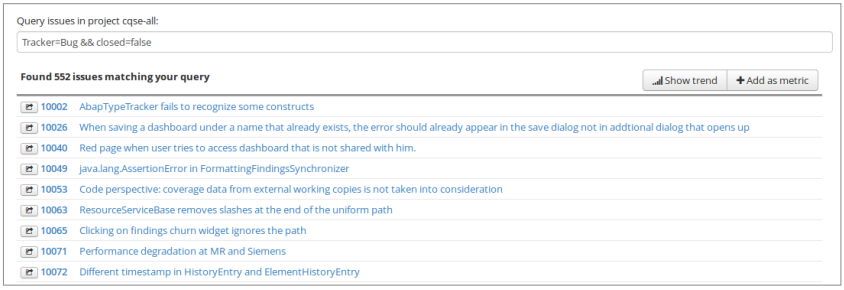


Figure 4.53. Query Area of the Issue Perspective

include slashes, these are used to defined directories for grouping similar issue metrics. Good widgets for using the issue metrics are the *Numeric Metric Value* and the *Single Metric Trend Chart* widget as shown in Figure 4.52. This allows you to track key indicators for your issue management process and react timely to any changes in the observed trend.

4.1.7.4 Action: View Issue Details

A click on a issue from the issue list will open the *Issue Details View* as shown in Figure 4.54. The view subdivides into to following parts:

- **Issue meta data**: This includes, among other information, the subject and the description of the issue, as well as the issue author and the current assignee.
- **Affected files**: Lists all files known to *Teamscale* which have been edited during the implementation of the issue. This is determined based on the issue IDs in the commit messages (see Section C).
- **Test Gap Treemap**: Shows the test gaps in your code base introduced with this issue. See Section 4.1.6.3 for details.
- **Associated repository changes** Display all commits targeting the current issue. Note that only commits which changed files known to *Teamscale* are considered.

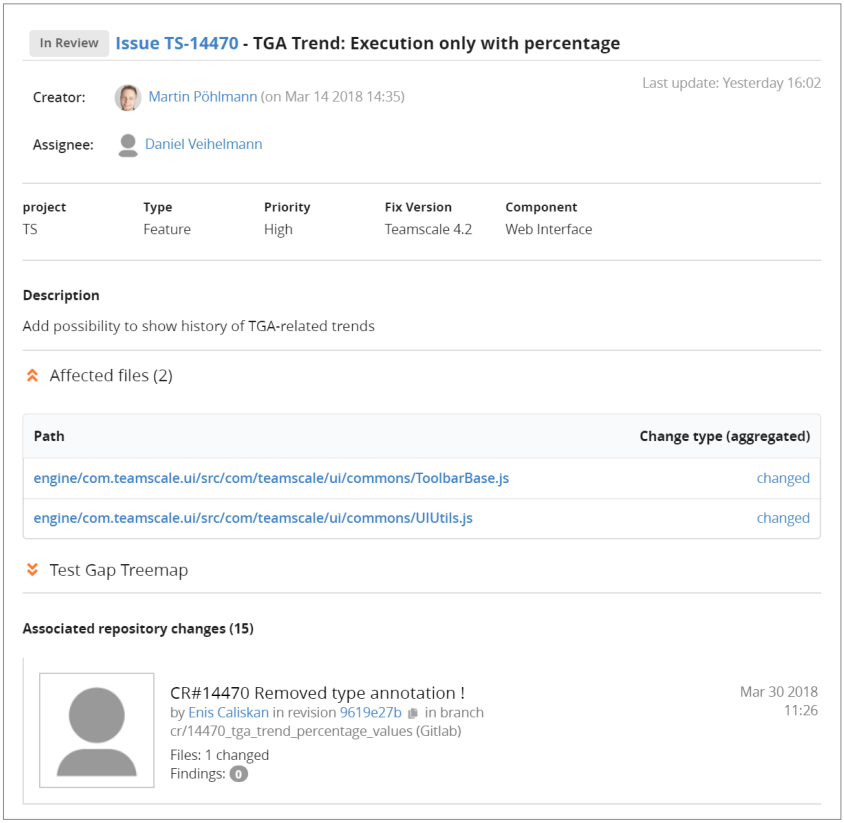


Figure 4.54. Details of single Issue

4.1.8 Task Perspective

Use Cases	
	Creating and Organizing Tasks
Views	
	Tasks Overview View, Create Task View, Edit Task View, Task Details View
Actions	
	Navigating All Tasks, Create Tasks, Edit Tasks, Work On Tasks

In the **Task** perspective, you can create tasks to keep track of the progress of the removal of findings and quality deficits. It is possible to assign tasks to a user and annotate them with tags for filtering. Tasks can be in different states to reflect the progress of the work.

Tasks Overview View As default the **Task** perspective shows a table of all open tasks, see Figure 4.55. The table shows the most important information of the tasks and is sortable by clicking onto the header.

4.1.8.1 Action: Navigating All Tasks

Large software systems can have many tasks. To better navigate them, the sidebar (Figure 4.56) provides several filter mechanisms. The tasks can be filtered by their task states or by different tags. Additionally, it is possible to filter them by their author or assignee.

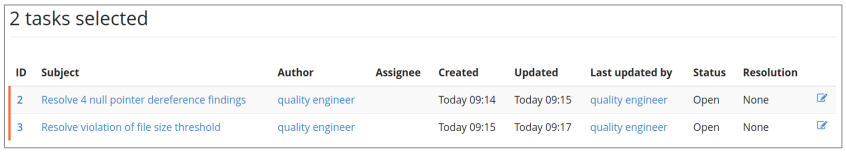


Figure 4.55. Task Overview

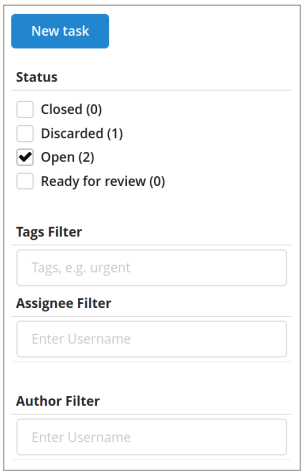


Figure 4.56. Task Overview Sidebar

Create Task View There are two ways to create a new task, either by attaching a finding to a new task (see Section 4.1.4.4) or by clicking on the New task button above the filters, see Figure 4.56. If you create a task via the button New task the *Create Task View* as seen in Figure 4.57 is opened.

4.1.8.2 Action: Create Tasks

A task must have a subject that briefly describes the purpose of the task. Additional you can set an assignee and annotate it with some

tags, e.g. »urgent«. The description can be used to clarify the intent of the task and give hints to resolve it. It is possible to format the description with markdown, for help you can click on **Formatting** below of the label **Description**. It is possible to attach findings after the creation of the task via the **Findings** or **Code** perspective as described in Section 4.1.4.4.

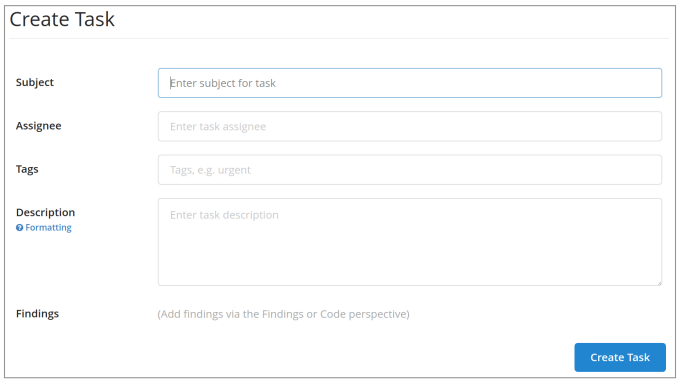


Figure 4.57. Task Create View

Edit Task View The *Edit Task View* allows you to edit the fields of an existing task. The changes are stored in the change history of the task an in Figure 4.58 depicted.



Figure 4.58. Task Change History

4.1.8.3 Action: Edit Tasks

Within the *Edit Task View* it is possible to change the field values you have set during the creation of a task. In addition, it is possible to set the state and resolution of a task and remove the findings from the task by clicking on the symbol 🗑.

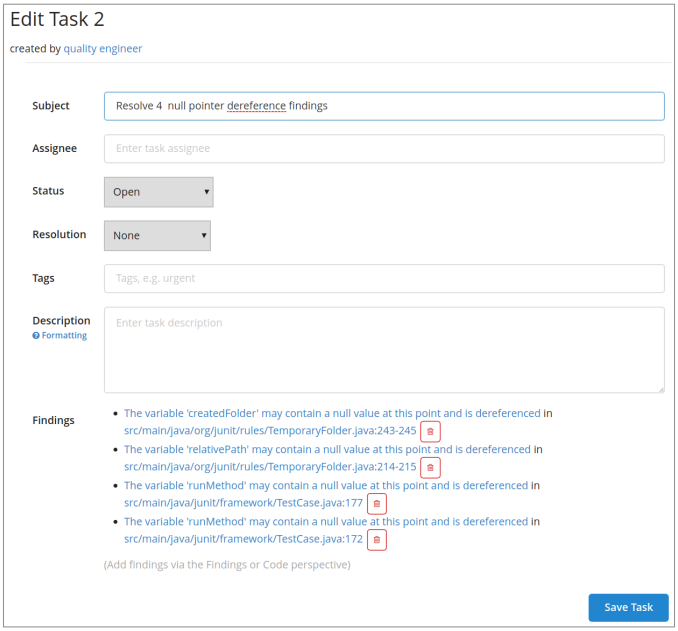


Figure 4.59. Task Edit View

Task Details View The *Task Detail View* shows all details of a task and allows you to work on it. This means you can change the state of a task or you can discuss a solution to resolve it.

4.1.8.4 Action: Work On Tasks

A task can be in one of the following four different states:

- **Open** An unresolved task.
- **Ready for review** The task has been resolved by the assignee.
- **Closed** A quality engineer has confirmed the resolution.
- **Discarded** The task has been discarded.

Throughout the normal workflow a task starts with the status **Open**. The assignee addresses the problem in the code and changes the status to **Ready for review**. This gives the quality engineer the opportunity to verify the solution. He can then either set the status to **Closed**, if the solution is valid or again to **Open**, if there are still problems.

The resolution gives a hint to the quality engineer how the task was tackled: either it was fixed or it was not possible to fix it for specific reasons.

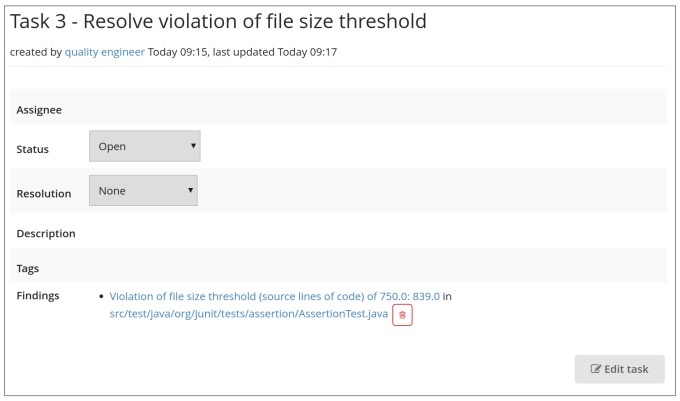


Figure 4.60. Task Detail View

If you want to reference a specific *Teamscale* task from a commit, you can add TS#<ID> to the commit message. The <ID> denotes the ID of a specific task, e.g. TS#3. This commit is then referenced in the *Task Detail View* as seen in Figure 4.61.



Figure 4.61. Task Commit

It is possible to discuss a task within the *Task Detail View* this can be used to clarify the description or a solution, see Figure 4.62.

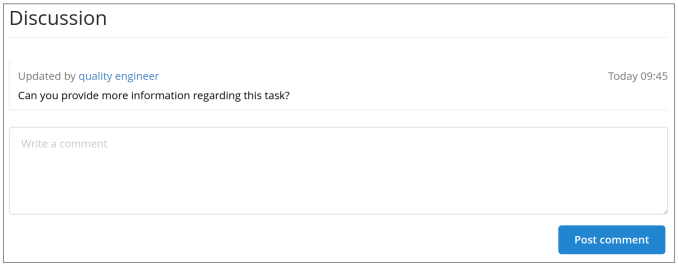


Figure 4.62. Task Discussion

4.1.9 Architecture Perspective

Use Cases	
	Modeling and Analyzing the Architecture of Your System
Views	
	Architecture Overview, Architecture Inspect View, Architecture Edit View
Actions	
	Model Architecture, Save Architecture, View Architecture, Inspect Metrics For Architecture

In the **Architecture** perspective, you can model and analyze the architecture of your system. *Teamscale* provides an architecture conformance analysis to check if the implementation follows the architecture specifications. Therefore, the architecture specifications (.architecture-files) can be modeled using nested components. The specifications can then be committed to the version control system or saved in *Teamscale* itself to be used for continuous conformance checking. To enable the architecture analysis, the architecture quality indicator has to be enabled in the analysis profile of the underlying *Teamscale* project. In the analysis profile, you can also configure if third-party dependencies should be considered in the architecture analysis or excluded from it, see Figure 4.77.

Architecture Overview View When you open the **Architecture** perspective, you will see an overview of existing architecture specifications, see Figure 4.64. This overview will be empty initially if no

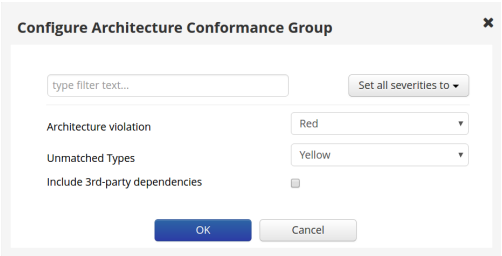


Figure 4.63. Configuring the Architecture Quality Indicator

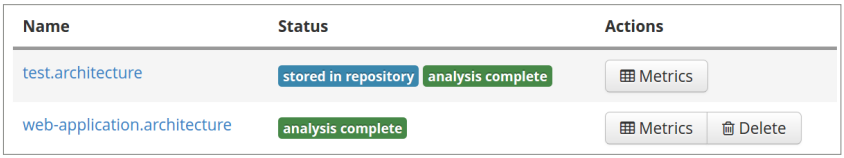


Figure 4.64. Architecture Overview

architecture has been modeled yet. For each existing architecture model, the table shows its status and the available options. For an explanation of all possible status as well as the available action buttons, please see Figure 4.65 and Figure 4.66.

The sidebar provides you four buttons, see Figure 4.67. You can download the dependencies in XML format either as file dependencies or as type dependencies. And you can create a new architecture specification or import an existing one. Before we go into details about how to create a new architecture, we briefly introduce the concept of »architecture« in *Teamscale*, see Section 4.1.9.1.

4.1.9.1 *Teamscale* Architecture Model

The intended architecture can be specified in a simple yet powerful hierarchical component model. The model has few basic concepts:

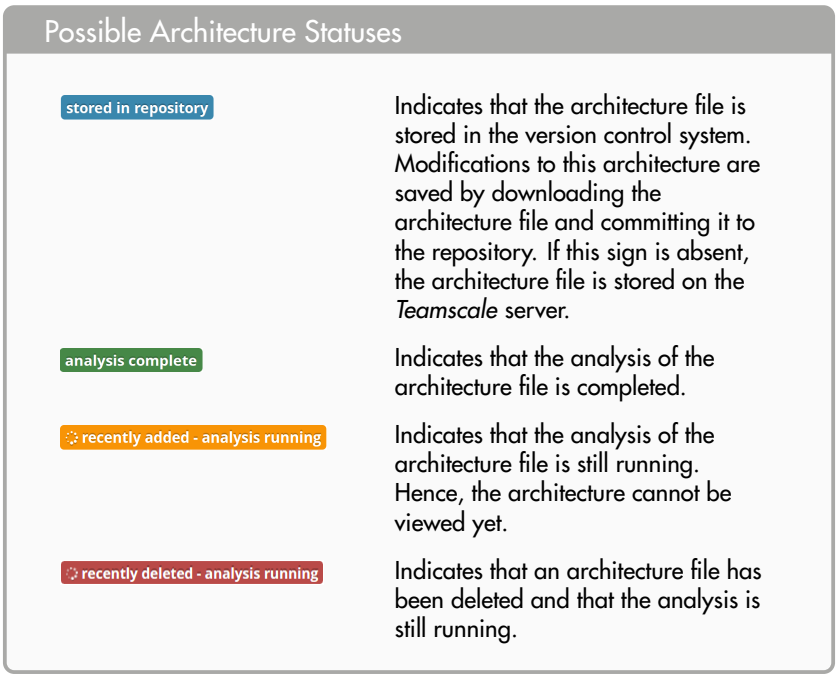


Figure 4.65. Buttons Available in the Architecture Table

Buttons For an Architecture File


Shows the metrics for the given architecture similar as in the **Metrics** perspective.


Deletes the given architecture. This is only an available option if the architecture is stored in *Teamscale*. Architectures stored in the repository can be deleted only by deleting them from the repository.

Figure 4.66. Action Buttons in the Architecture Table

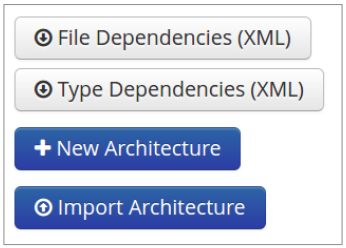


Figure 4.67. Sidebar of the Architecture Overview

- **Components:** The system is modeled as a hierarchy of components. A component has a unique name and mappings and may have subcomponents. Components can have a stereotype PUBLIC indicating that all other components may access this components or a stereotype COMPONENT_PUBLIC specifying that it may be accessed by all other components within the same parent component (e.g. a GUI-library within an UI-layer). This is especially useful for library components that are accessed by many other components.

- **Mappings:** A mapping between code and architecture defines which implementation artifacts belong to which component in the architecture definition. This is specified using Java regular expressions as include and exclude pattern.

- **Architecture Type**: For an architecture, you can choose if the architecture is type- or file-based. For a type-based architecture the full-qualified names of all types (classes, structs, enums etc.) in the code base are used for the mapping. For file-based architectures the path to the source code files are used.

- **Policies:** A policy defines which components may be in a relationship (allow, deny, tolerate) with each other. Each dependency policy has a source and a target component. If components are not connected by a policy-edge, dependencies between them are (implicitly) forbidden. You need to explicitly model allow-policies (green arrows) to allow dependencies. If an allow-policy is specified between two components all of their sub-components may also share dependencies. You can explicitly deny this, by modeling deny-policies (red arrows). Tolerate-policies can be used to accept an explicit set of dependencies between components.

- **Violations**: If additional dependencies are introduced, which contradict the modeled policies, *Teamscale* will mark them as violations of the architecture. Violations can be defined to be treated as regular findings in *Teamscale*. This can be configured while editing the architecture (see Section 4.1.9.2).

- **Orphans**: Types/files that reside within the scope of the architecture, but are not mapped to a component, are regarded as so-called orphans. These orphans can be defined to be findings in *Teamscale*. This is to be configured while editing the architecture (see Section 4.1.9.2).

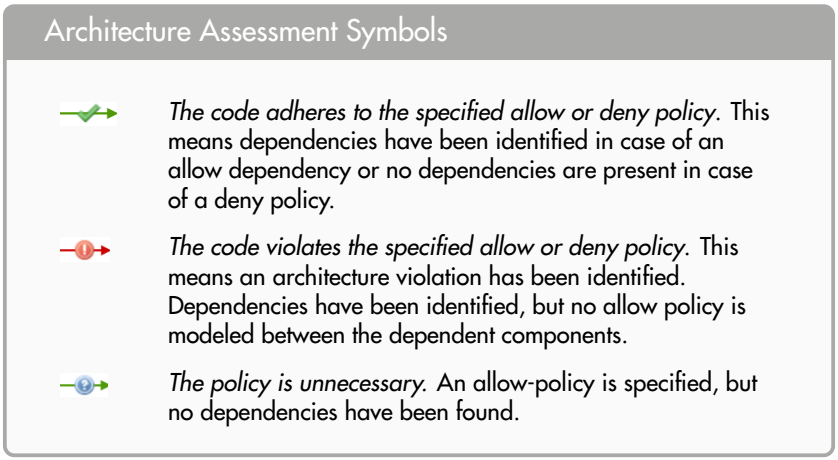


Figure 4.68. Assessment Symbols for Modeled Dependencies

4.1.9.2 Action: Model Architecture

After having gained a basic understanding of the *Teamscale* architecture model in the previous section, we can now start to model our own architecture specification. When you open the **Architecture** perspective and click on New Architecture, the *Architecture Edit View* opens, see Figure 4.69. You will reach the same view upon opening an existing architecture and clicking on Edit Architecture.

Architecture Edit View When creating a new architecture, the editor will be empty. Underneath, a variety of options are displayed on the left and the current orphans are displayed on the right, see Figure 4.69.

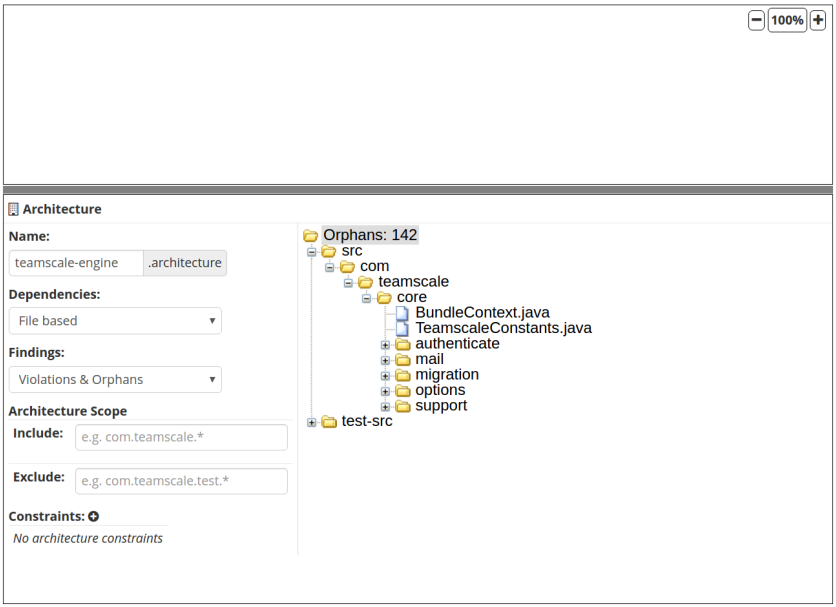
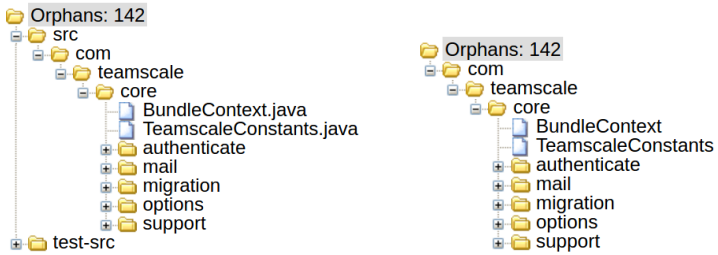


Figure 4.69. Architecture Editor

On the left side, you can edit the name of your architecture specification file and choose, whether the architecture should be based on file dependencies or type dependencies (see our architecture model in Section 4.1.9.1). You can also configure, whether findings should be created for both violations and orphans, or just for violations, or for neither one. Furthermore, the scope of the architecture can be defined. By default, the scope of the architecture matches the scope of the underlying project. It can be narrowed down by defining explicit include or exclude patterns, e. g. to exclude test code.

Additionally, constraints can be configured. A constraint represents a deny policy based on regular expression patterns. For example, in a file-based architecture definition, you may want to add the con-



(a) File-based Orphans (b) Type-based Orphans

Figure 4.70. Differences in Orphan Structure Based On Architecture Type

straint ».$*/src/.* \rightarrow .*/test-src/.*$« to indicate that source code should not depend on test code. In the simplest version, if the regex matches exactly two components, the constraint leads to an identical result as an explicitly defined deny policy between these two components. However, there may be cases, in which policies cannot be matched directly onto components (or it would create overload in creating the specification). Hence, it might be easier to define additional constraints orthogonal to the component definition.

On the right side underneath the editor, the orphans are shown. If no architecture has been created yet, all files or types will be considered orphans. Figure 4.70 shows that in architectures based on file dependencies, orphans are displayed based on the folder structure of the project. For architectures based on type dependencies, in contrast, they are displayed according to, in Java, for example, the package structure.

To start modeling the architecture, the sidebar of the *Architecture Edit View* (Figure 4.71) provides you the necessary tools, which are all described in Figure 4.76. Usually, you start by creating the first component(s). Click on  and then hover over the white (currently still

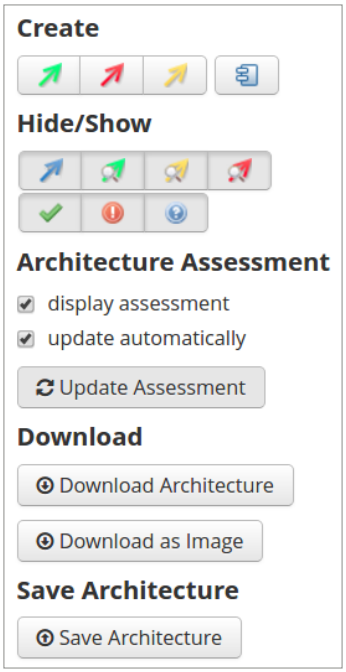


Figure 4.71. Sidebar of the Architecture Editor

empty) editor field to drag & drop your first component. As displayed in Figure 4.72, you can give each component a name.



Figure 4.72. Creating a Component.

Alternatively, from the orphans at buttom-right, you can drag & drop folders or packages onto the editor. A new component will be created for every item dropped while mappings and dependency policies will automatically be created. Within the editor, components

can be dragged into or out of other components and this will adjust the mappings and dependencies accordingly. If the checkboxes **display assessment** and **update automatically** in the right sidebar are marked, you will see how many types are matched to the component. You can also manually trigger the update of the assessment by clicking the **Update Assessment** button. This works if **display assessment** is unchecked.

Each component has a number of properties that are presented for modification when clicked. The properties are shown in a panel with resizable height below the editor. In Figure 4.73, the name of a component as well as some descriptive comment can be changed. If a component has dependencies, they are by default visible as arrows in the editor. Their visibility can be controlled under *Policies Visibility* by clicking the Hide All and Show All toggle buttons for the *Outgoing* and *Incoming* dependencies. The *Stereotype* of a component indicates whether it can be accessed by all components (reflected in the value PUBLIC) or just by only components within the same parent component (value COMPONENT_PUBLIC). Mappings can be added, deleted or modified.

With the buttons in the sidebar on the right (Figure 4.71), you can model policies between components. For example, click on  to create an allow policy and then first click on the source component and – while keeping the mouse button clicked – drag the upcoming policy arrow till you hover over the target component. Then release the mouse button and a green policy should appear, see Figure 4.74. Similarly, you can create tolerate  and deny  policies.

While you are modeling your architecture, *Teamscale* will assess the created policies on the fly, if the checkboxes **display assessment** and **update automatically** in the right sidebar are marked. *Teamscale* assesses whether the created policies are violated by dependencies in the source code, see explanations in Figure 4.68. Figure 4.75 shows,

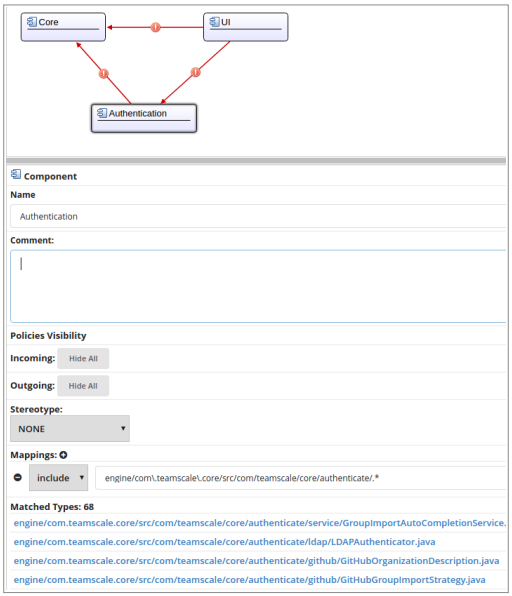


Figure 4.73. Editing a Component.



Figure 4.74. Creating an Allow Policy Between Two Components

for example, that a modeled deny policy was, in fact, violated in the code by one dependency. *Teamscale* also displays source and target of the dependency and allows you to directly jump to the code file by clicking on the respective location link.

4.1.9.3 Action: Save Architecture

When you are done creating your architecture, you need to save your changes. When you are in the *Architecture Edit View*, you can ei-

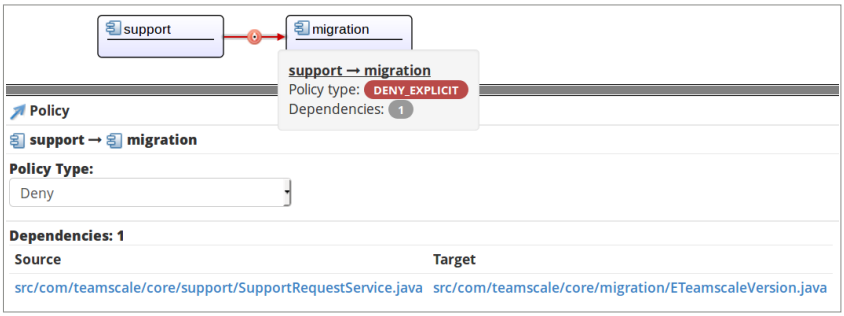


Figure 4.75. Assessment of a Deny Policy.

ther download the architecture file to check it into your repository or you can save the architecture on the *Teamscale* server. Use the corresponding buttons in the sidebar **Download Architecture** or **Save Architecture**. In both cases, the change to the architecture file will show up as a commit in the **Activity** perspective (see Section 4.1.3). When you are storing the architecture in your repository, your regular commit to the repository will be displayed. When storing the architecture in *Teamscale*, an artificial commit will be created, see Figure 4.77. The artificial commit is named »Changed architecture« and also has an artificial revision number.

When saving an architecture on the *Teamscale* server, it will be visible for all users of the underlying project – same as if you would commit it to the repository. Other users can not only read but also modify your architecture.

4.1.9.4 Action: View Architecture

Often, you want to simply view an existing architecture, which was maybe created by your colleagues or by yourself several days ago. When you open the **Architecture** perspective and you already mod-

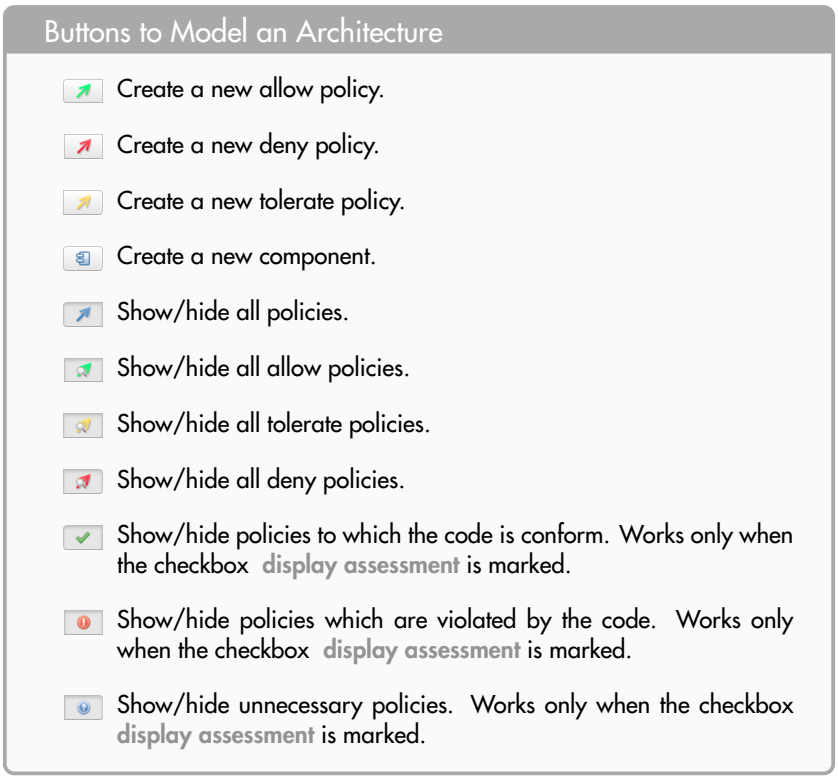


Figure 4.76. Explanations of Buttons in Architecture Edit View

eled an architecture, you can click on its name in the table of the *Architecture Overview View*, see Figure 4.64. This will open the architecture in the *Architecture Inspect View*. A completely modeled architecture may look as depicted in Figure 4.78.



Figure 4.77. Artifical Commit For Saving an Architecture on the *Teamscale* Server

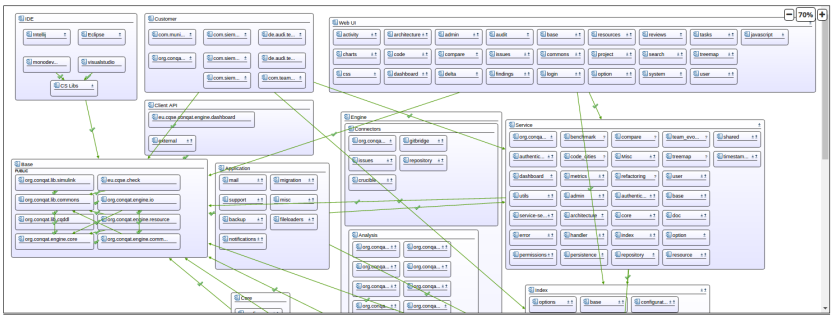


Figure 4.78. Inspecting An Architecture

Architecture Inspect View In this view, you can use the **Hide/Show** buttons in the sidebar, see Figure 4.79 to filter the policies in your architecture view. You can also click on the components to inspect which types were matched or click on the violations to see where they are coming from. Additionally, you can download the architecture either as XML file (**Download Architecture**) or as SVG file (**Download as Image**).

4.1.9.5 Action: Inspect Metrics for Architecture

As *Teamscale* measures a variety of metrics for your system, you can also view them with respect to your architecture. When you open the **Architecture** perspective with the *Architecture Overview View*,



Figure 4.79. Sidebar for Viewing the Architecture



Figure 4.80. Metric Overview Table For an Architecture

you can use the *Metrics* button to inspect the metrics. This will provide you a view with a table, in which the metrics are aggregated based on the components of your architecture, see Figure 4.80. The table is the same as you can also find in the **Metrics** perspective, see Section 4.1.5. When you drill into it, you can navigate the various components and descend into sub-components till you reach a mapped code file. Along the way, the metrics in the table are always aggregated to the (sub-) component in which you are currently located. Hence, you can inspect, for example how large a component is in terms on lines of code, if a specific component is missing documentation or whether the test coverage varies greatly between different components.

In addition to the **Architecture** perspective itself, you can use your architecture specification also in other perspectives. For example, the **Dashboard** perspective (see Section 4.1.2) offers to configure a widget for a specific (sub-) folder of your system. In analogy, a widget can also be configured for a (sub-) component of your system. Generally speaking, in *Teamscale*, whenever you can choose a folder path from your project, you can also choose an architectural component definition instead.

4.1.10 Delta Perspective

Use Cases	
	Comparing two Snapshots of a System
Pages	
	Delta Parameters, Changed Files, Repository Changes, Change Treemap, Test Gap Treemap, Metric Chages, Findings Churn
Actions	
	Calculate Delta, Inspect Changed Files, Inspect Repository Changes, View Change Treemap, View Test Gap Treemap, View Metric Changes, Navigate Findings Churn

The main purpose of the **Delta** perspective is to calculate the delta, i. e. compare two snapshots of a system with each other. A snapshot is defined by a point in time and a branch for which *Teamscale* retrieves the corresponding commit in the version control system as well as all related quality data. The comparison of two snapshots reveals the differences in terms of code changes, metric changes and findings churn as well as information on whether the code changes have been tested.

Teamscale can compute two types of delta: a *commit-range* delta (i.e., »what happened between commit1 and commit2«) and a *merge-request* delta (»what would happen if I merge commit1 on top of commit2«).

The commit-range delta requires two commits as input. These commits should be in a linear commit history (the baseline/start commit should

be an ancestor of the end commit). The input for a commit-range delta can be configured in the *Linear History* tab of the *Delta Parameters Page*.

The merge-request delta also requires two commits as input. They can be configured in the *Merge Request* tab of the *Delta Parameters Page*. You can choose any two different commits from the project repository. Teamscale will compute a preview of the estimated effects that accepting the corresponding merge request would have on the repository (which findings would be added to the target branch, which commits would be merged, etc.).

Since the configuration options and available views for the two types of deltas are similar, we focus on describing the commit-range delta.

4.1.10.1 Action: Calculate Delta

To calculate a delta, you can navigate to the **Delta** perspective which will open up the *Delta Parameters Page*.

Delta Parameters Page On this page you can configure the delta you would like to calculate, see Figure 4.82. The box, see Figure 4.81 describes the available options. If branch support has been enabled for the underlying project, you can also compare snapshots from two different branches with each other. Click Compute Delta to start the delta analysis.

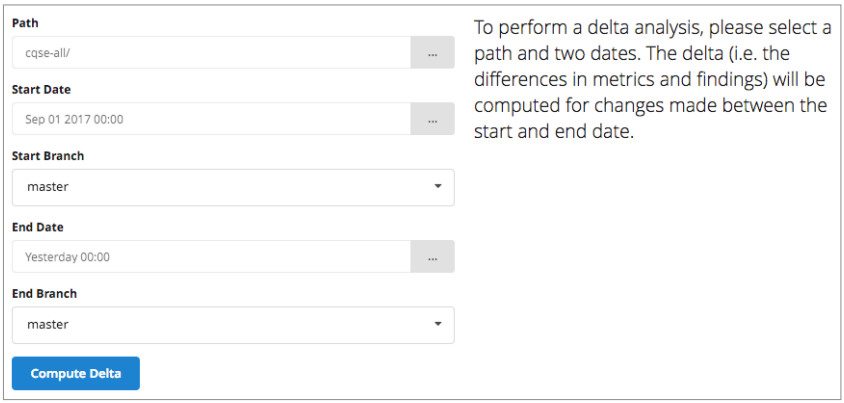


Figure 4.82. Delta Parameters With Branch Support

Delta Parameters

Path	Select the path for which you want to compute the delta. This can be a a top-level code folder or any subfolder of a *Teamscale* project but also a component of an architecture definition.
Start Date	Select the start date as a time, revision or baseline.
Start Branch	If branch support is activated, you can select the branch for the start date.
End Date	Select the end date as a time, revision or baseline.
End Branch	If branch support is activated, you can select the branch for the end date.

Figure 4.81. Parameters for Calculating a Delta

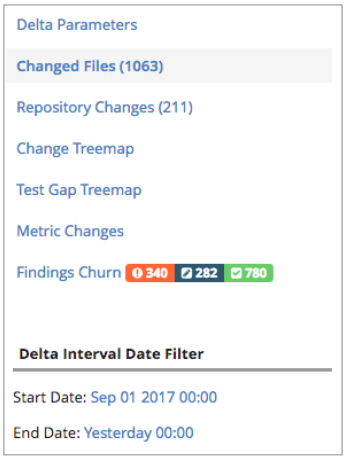
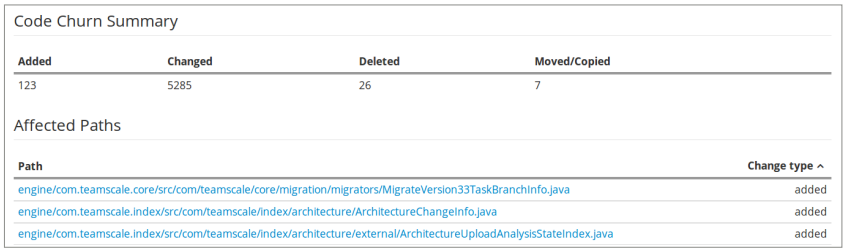


Figure 4.83. Sidebar of the **Delta** Perspective

4.1.10.2 Action: Inspect Changed Files

After calculating the delta, the sidebar of the perspective will appear, see Figure 4.84. It automatically redirects you to the *Changed Files Page*.

Changed Files Page This page gives you an overview which files in the repository were »affected« within the selected interval of the delta. Figure 4.84. The page provides you a **Code Churn Summary** indicating how many files were added, changed, deleted or moved/-copied. The latter is identified with *Teamscale*'s own origin analysis [SHJ14] which reveals if a file was copied from an existing files (either identically or with small changes) or moved between locations. Further more, the page shows you all **Affected Paths**. If you click on a path, it will redirect you to the *Code Detail View* to view the source code (see Section 4.1.5). If a file has been changed, indicated by the


Code Churn Summary

Added	Changed	Deleted	Moved/Copied
123	5285	26	7

Affected Paths

Path	Change type
engine/com.teamscale.core/src/com/teamscale/core/migration/migrators/MigrateVersion33TaskBranchInfo.java	added
engine/com.teamscale.index/src/com/teamscale/index/architecture/ArchitectureChangeInfo.java	added
engine/com.teamscale.index/src/com/teamscale/index/architecture/external/ArchitectureUploadAnalysisStateIndex.java	added

Figure 4.84. List of Affected Files

Change type, you can click on it to open the *Code Comparison View* to view the differences.

4.1.10.3 Action: Inspecting Repository Changes

The sidebar provides a link to the *Repository Change Page*.

Repository Change Page This pages provides an overview of all commits which took place within the time interval of the delta. The page contains the activity stream of commits in the same way as the *Activity View* of the **Activity** perspective (see Section 4.1.3). Browsing the stream of commits is particularly useful if you want to find the root cause of a metric change as described in Section 3.4.2.

4.1.10.4 Action: Viewing the Change Treemap

While the *Changed Files Page* shows the code changes in the delta in a list format, the *Change Treemap Page* provides the same information in treemap form.

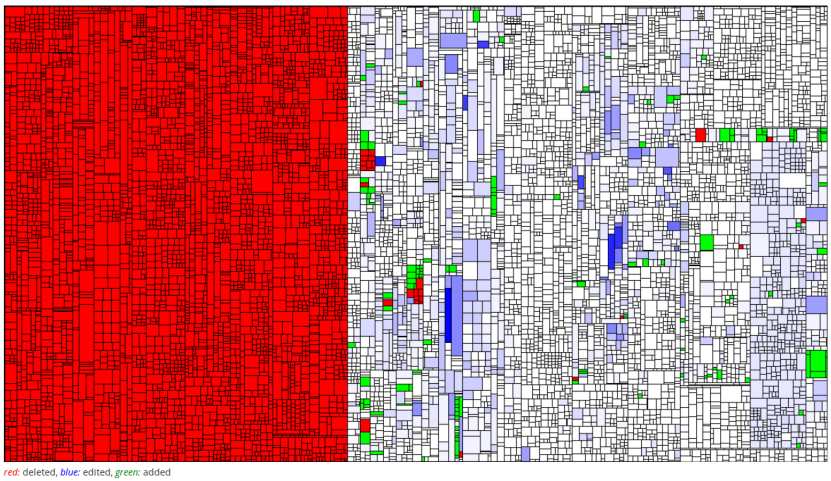


Figure 4.85. Change Treemap for a Delta

Change Treemap Page Figure 4.85 shows such change treemap. Red indicates files which were deleted during the delta, blue indicates changed files and green indicates added files.

4.1.10.5 Action: Viewing the Test Gap Treemap

To get an overview over which changes contained in the delta have been tested and which might have been missed, you can use the *Test Gap Treemap Page*.

Test Gap Treemap Page Figure 4.86 shows a test gap treemap for a given delta. This treemap works just like the Test Gap Treemap Widget in the dashboard (see Sections 3.5.1 and A.26), but is already configured with the start and end date provided in the delta parameters. Every rectangle represents a method in the system. Gray indicates

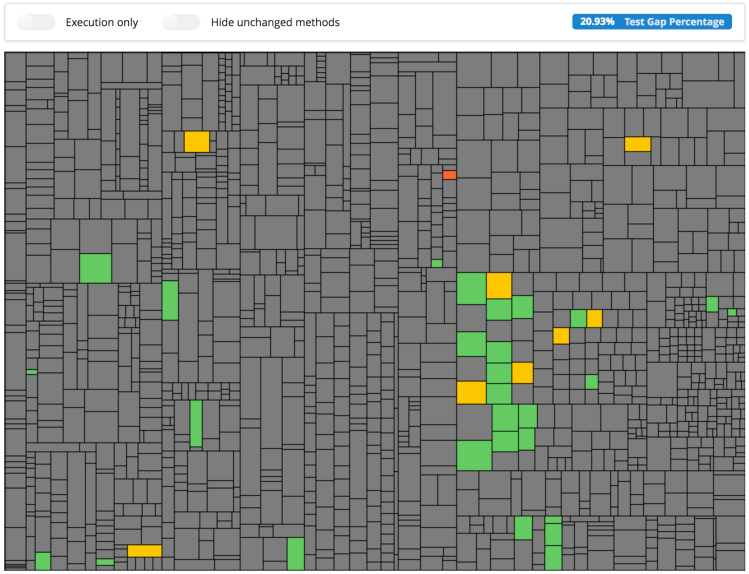


Figure 4.86. Test Gap Treemap for a Delta

methods which were not changed in the selected time frame, green indicates added or changed methods which have been tested (and the test coverage has been uploaded to *Teamscale*), and red and orange indicate untested methods that have been added or changed within the selected time frame, respectively. Also just like in the dashboard, the **Execution only** toggle shows test coverage irrespective of changes, and **Hide unchanged methods** makes small changes in large systems easier to locate by hiding all gray areas.

4.1.10.6 Action: Viewing Metric Changes

The **Delta** perspective does not only provide information about code churn and the corresponding commits, it also contains the delta with

Metric Churn

Metric	Change
Files	+401
Lines of Code	+91k
Source Lines of Code	+44.2k
File Size Assessment	+2155 +10669 +31415
Longest Method Length	+69
Method Length Assessment	±0 +596 +30164
Maximum Nesting Depth	+5
Nesting Depth Assessment	±0 +44 +30716
Last Change Date	Apr 07 2017 16:37
Number of Findings	+239
Number of Findings (Red)	+2
Number of Findings (Yellow)	+237
Findings Density	+2.6
Findings Density (Yellow)	+2.6
Clone Coverage	+1.04%
Code Defect Density Assessment	+34.00%
Change Count, Findings Density (Red), Comment Completeness Assessment, Bug Fix Assessment	*no change*

Figure 4.87. Metrics Table for the Delta

respect to quality data. In particular, it provides the **Metric Churn**. Use the left sidebar to navigate to the corresponding page.

Metric Churn Page This page shows the delta for each metric, comparing the metric value at the end of the selected time interval with the metric value at the selected baseline. Figure 4.87 shows the table indicating the **Metric Churn**. For each metric, it provides the name as well as the delta change.

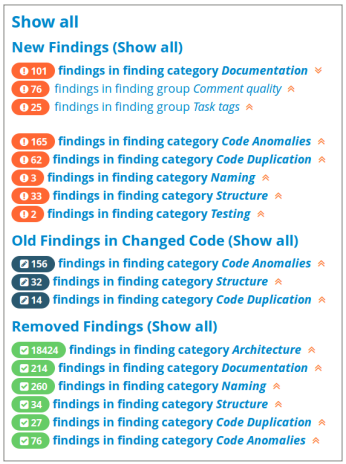


Figure 4.88. Overview Findings Churn

4.1.10.7 Navigating the Findings Churn

In addition to code churn and metric churn, the **Delta** perspective also provides the findings churn. Use the right left to navigate to the *Findings Churn Page*.

Findings Churn Page This page gives you an overview of findings which were added (**New Findings**) and which were removed (**Removed Findings** during the selected time interval, see Figure 4.88. Additionally, it also shows you findings in code which was changed during the time interval. This is relevant if you selected quality goal *improving* for your project. Under this quality goals, findings in changed code should be removed on-the-fly. Hence, this page shows you the opportunities you missed.

The overview of the findings churn can be expanded to view the individual findings in each finding category and finding group. Upon ex-

panding, the findings are shown in a table as depicted in Figure 4.89. The table provides the message, the finding location as well as the finding group. Clicking on the messages opens the *Findings Detail View* of the **Findings** perspective (see Section 4.1.4), clicking on the location redirects you to the *Code Detail View* (see Section 4.1.5).

Additionally, the sidebar offers the same filter mechanisms to navigate the findings as the **Findings** perspective already described in Section 4.1.4: You can filter by entering a regular expression or by selecting specific finding categories or finding groups.

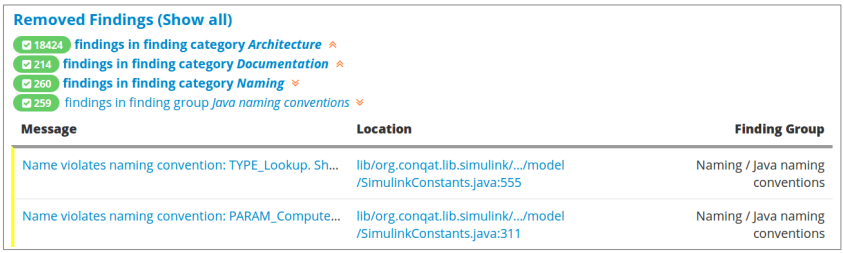


Figure 4.89. Expanding the Findings Churn

4.1.11 Project Perspective

Use Cases	
	Configuring *Teamscale* Projects
Pages	
	Projects, Analysis Profiles, Thresholds, External Metric, External Findings
Views	
	Create Project View, Create Analysis Profile View, Edit Analysis Profile View, Create Threshold Configuration View, Create External Metric Descriptions View, Import External Finding Descriptions View
Actions	
	Managing Projects, Creating a Project, Managing External Uploads, Managing Analysis Profiles, Creating an Analysis Profile, Managing Threshold Configurations, Creating a Threshold Configuration, Creating New External Metric, Import External Metric Descriptions, Export External Metric Descriptions, Creating New External Finding Group, Create New External Finding Description, Import External Finding Descriptions, Export External Finding Descriptions.

The *Teamscale* project perspective enables users to manage projects, analysis profiles, metric threshold configurations, external metrics and external external findings. This section describes the user interfaces for creating and managing each of these entities.

4.1.11.1 Action: Managing Projects

Projects in *Teamscale* contain links to one or more source code repositories and the findings that *Teamscale* generates and collects for the corresponding source code artifacts.

Projects Page Figure 4.91 shows the **Projects** page of the **Projects** perspective. After the project is set up, it appears in the project list. For each existing project, the project list offers different buttons for project management as shown in Figure 4.90.

4.1.11.2 Action: Creating a New Project

To create a new project, the **Projects** page provides above the list of existing projects the button Create new project. A user needs to set a project name and an analysis profile. In Figure 4.92, we chose the *ABAP (default)* profile as an example. Furthermore, the user can choose a dashboard template.

A project can be configured with various connectors for version control systems, issue trackers and code review tools, see Figure 4.93. In terms of version control systems, *Teamscale* offers connectors to GIT, SVN and TFS, as well as the Git management platforms[2] GITHUB, GITLAB and BITBUCKET. However, it can also read from the file system, either with a simple **File System** or a **Multi-Version File System** connector.

[2]While all these platforms can be accessed using the plain Git connector, using a more specific connector enables additional features, such as automatic update on push, and annotation of pull requests.

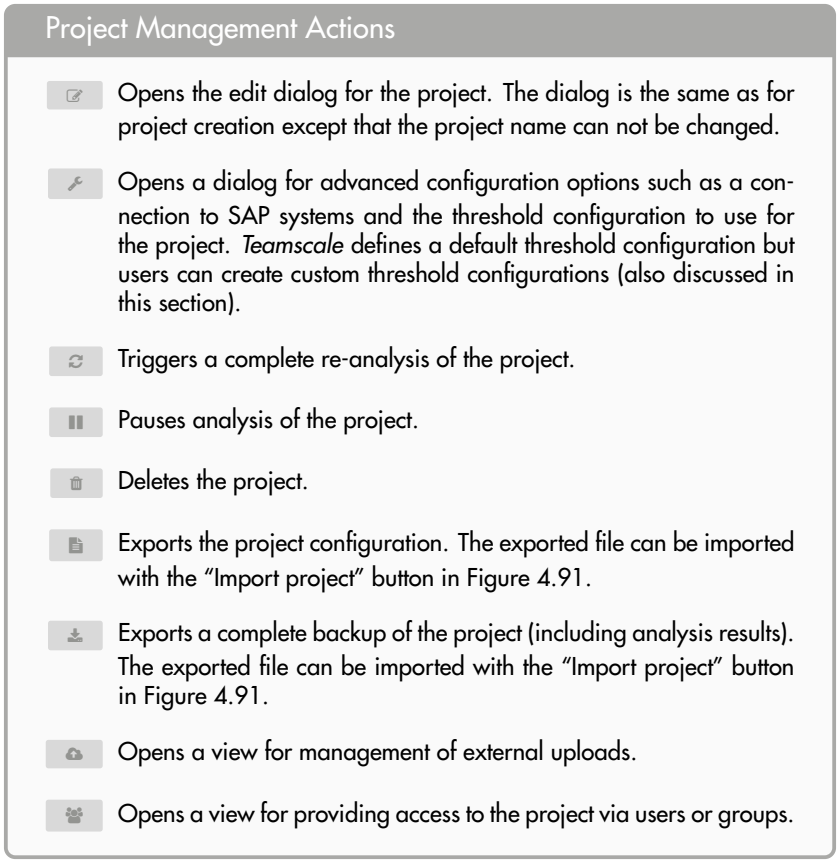


Figure 4.90. Actions Available in the Project Overview

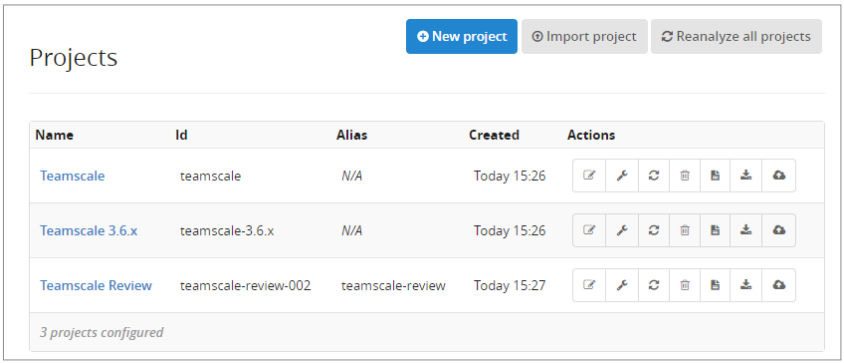


Figure 4.91. List of Configured *Teamscale* Projects

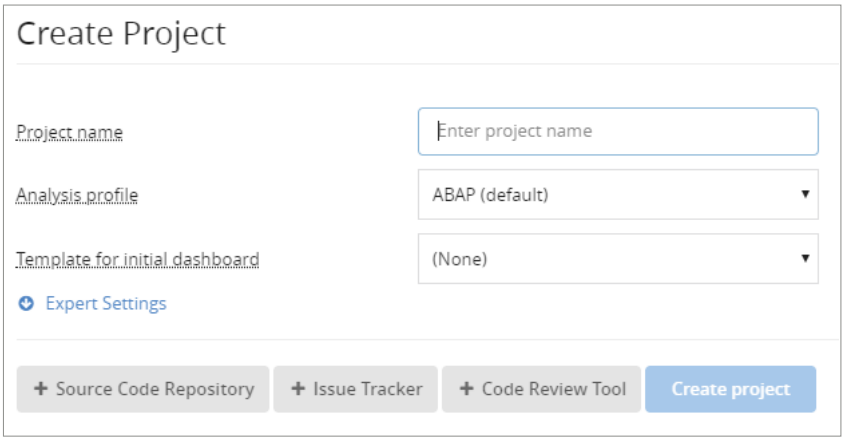
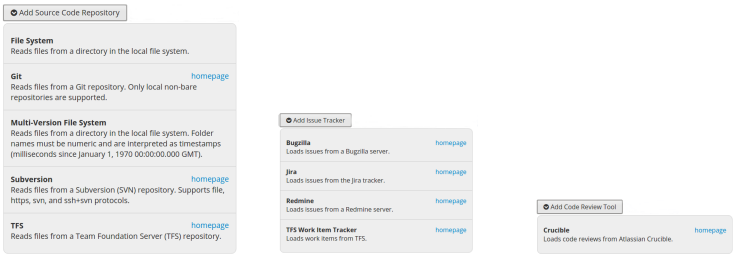


Figure 4.92. Setup View for a *Teamscale* Project

Git Connector To set up a GIT connector, the user has to specify account data (the + button opens an account setting dialog), and a repository identifier (must be unique in the project). Furthermore, the user can specify a start revision to reduce the analysis load and include or exclude files. You might, for example, want to exclude generated code or experimental code from the analysis. Also, you



(a) Repositories (b) Issue Trackers (c) Code Review Tools

Figure 4.93. Connectors for Creating a Project

might want to have separate projects for your application and your test code. Note that *Teamscale* should be connected to either a remote repository (*Teamscale* will clone it locally into the working directory configured in the Server Settings page on the Admin perspective) or a dedicated local clone that is not modified by other tools or users. When using a local repository (file URL), make sure that the corresponding repository is *bare*, i. e. has been cloned with the `--bare` option.

Branch Support Teamscale also offers branch support. As default, branch support is not enabled while configuring the connector for a project. For large projects with hundreds of branches, this would dramatically increase the analysis time. You can manually enable branch support by clicking the corresponding checkbox and configure which branches should be included in the anlaysis by using include/exclude patterns. The patterns are reg-ex based. For example, you can match your release-branches or specific feature branches. Internally, *Teamscale* has a default branch, which maps to »trunk« for SVN and »master« for GIT. Sometimes, this has to be adjusted, however.

Multiple Repository Connectors You can also use multiple repository connectors for one project. For example, you can read both from GIT and SVN or from GIT and the file system or from multiple GIT repositories. If you have two connectors read the same files and/or commits, *Teamscale* will merge these inputs into a unified view. In the **Metrics** perspective and the subsequent views, each file will only appear once. In the history of a file (*Code Detail View*) and in the *Activity View*, each commit for each repository will be shown separately. If there are multiple commits with the same timestamp but with different content (different changes), *Teamscale* will decide which of them to display as the »latest« commit. This choice also affects computation of metrics in the dashboard as the metric values might differ depending on the order of the commits (e.g., added lines of code). A situation where such overlapping commits are loaded can easily be generated by using multiple multi-version file-system connectors.

Branch Support For Multiple Repository Connectors If you use multiple repository connectors and have branch support enabled, *Teamscale* will assume that branches with the same name can be mapped onto each other. Hence, *Teamscale* will only show one branch with the data from all connectors. However, if the branching structure is not consistent between connectors, this will not work, e. g. if you have a SVN connector followed by a GIT connector. For this case, *Teamscale* provides expert options in the connector configurations. The option **Branch transformation** allows you to provide a manual mapping between branches. In our example, you would indicate »trunk → master«, for example.

Multi-Version File-System Connector A multi-version file-system connector reads multiple versions of a repository that are stored in separate directories. The connector expects an input directory that con-

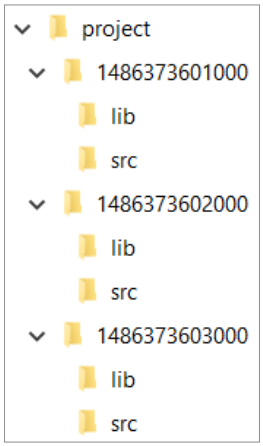


Figure 4.94. Directory layout for a multi-version filesystem connector

tains one subdirectory per repository version. The names of the subdirectories must be numeric and are interpreted as java timestamps. Figure 4.94 shows an example of this directory layout. The multi-version file-system connector expects that each subdirectory contains a full snapshot of the repository at the given timestamp. Each subdirectory is represented by a commit in *Teamscale*. If a file is missing in a subdirectory, it is treated as if it was deleted.

All options for repository, issue tracker or code review tool connectors are described in detail in Appendix B, C and **??**.

4.1.11.3 Action: Managing external uploads

Teamscale integrates reports by external analysis tools such as JACOCO (test coverage) and FINDBUGS (finding reports). Uploading of external-analysis reports is covered in Section 4.3.4.

To list existing external-analysis uploads click on the button. The list linked in this button shows all reports that were uploaded to *Teamscale*. The list is ordered by the target partitions of the uploads (typically, uploads of each external tool such as JACOCO and FINDBUGS are stored on separate configurable *Teamscale* partitions). Each upload partition and each upload commit have buttons. You can use the button of a partition to delete all uploads of the partition and the button of a upload to delete only the specific upload.

When a given upload is deleted, *Teamscale* starts a rollback to the latest commit *before* the upload in question and re-analyzes every following commit except for the deleted upload. Depending on the number and size of commits between the deleted upload and the newest commit, this partial re-analysis can take a significant amount of time.

Manual report upload Assuming you already set up a regular *Teamscale* project (see Section 4.1.11.2) and you have a supported coverage report (see Table 4.1) available, a manual upload can be performed as follows:

- Navigate to the **Projects** perspective
- Click on the button for your target project
- Click the Upload Report button in the top-right corner
- In the dialog (see Figure 4.95), you can configure the following options:
  - The coverage file to upload
  - **Message:** This text will appear in the **Activity** perspective for the current upload (e. g."Coverage upload")

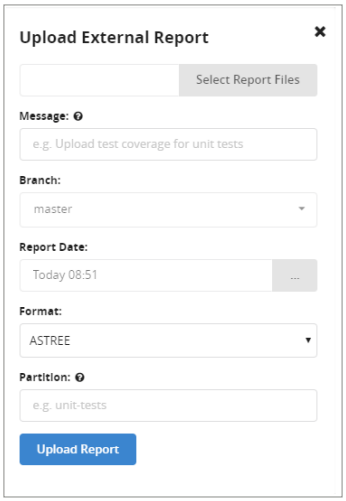


Figure 4.95. The Upload Report dialog

- **Branch:** The repository branch from which the coverage was created.
- **Report Date:** The repository timestamp of the code snapshot that was tested (which may be in the past). This allows *Teamscale* to link the correct state of the code with the test coverage.
- **Format:** The coverage format of a supported test profiler
- **Partition:** The test partition is a label that allows to differentiate between several test sources, e. g. manual tests and unit tests.

After you specified the options above and clicked Upload Report, *Teamscale* will process the coverage report.

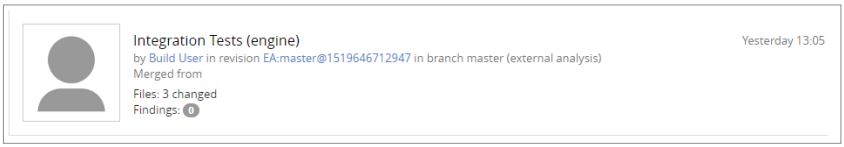


Figure 4.96. Upload commit in the Activity perspective

Verifying that an upload commit was successful To verify that your upload was successful, you can navigate to the **Activity** perspective and check whether you can see an upload commit similar to Figure 4.96. Note: Be sure to (1.) select the correct project and branch, (2.) navigate to the timestamp you selected for the upload, and (3.) deselect the **Hide external uploads commit** checkbox in the sidebar, as your commit will not be visible otherwise.
If the coverage information and code known to *Teamscale* could be matched correctly, the details of the upload commit will show files covered by the upload as "changed". If no files are affected, this can e. g. indicate that the code paths in the coverage report are not equal to the paths in *Teamscale*.

4.1.11.4 Action: Managing Analysis Profiles

In addition to configuring projects, the project perspective enables users to configure analysis profiles that can be used in different projects. An analysis profile configures the programming languages that will be analyzed, external tools which can be included in the analysis results, and which metrics are computed and displayed. *Teamscale* already offers many analysis profiles which are usually tailored to specific programming languages (e.g., **ABAP (default)** or **Java (default)**) but users can create and manage their own ones.

By clicking on **Projects** → **Analysis Profiles** in the left sidebar, the *Analysis Profile Page* will be shown. The page shows all existing

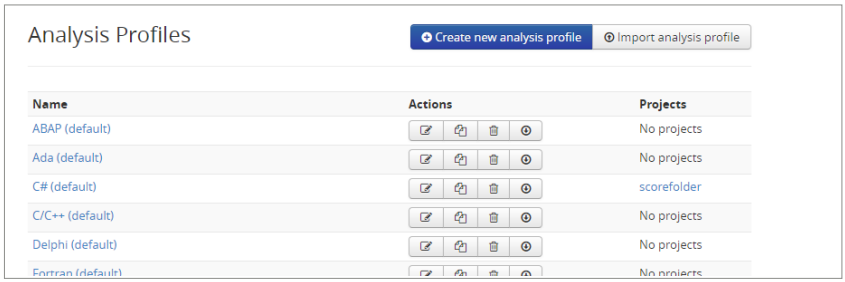


Figure 4.97. Analysis Profile Page

default profiles as well as individually created ones, see Figure 4.97. For each profile, the list offers different buttons for management, see Figure 4.98. In addition, all projects using a profile are stated in a separate column.

Actions for Analysis Profiles

- Opens the edit dialog for the analysis profile. The dialog is the same as for analysis profile creation except that the profile name can not be changed.
- Opens a dialog for copying the analysis profile.
- Deletes the analysis profile.
- Exports the profile as a file. Exported profiles can be imported with the `Import analysis profile` button above the analysis profile list.

Figure 4.98. Actions for an Analysis Profile

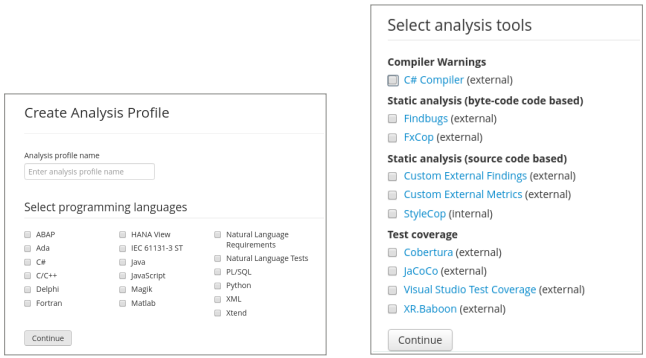
4.1.11.5 Action: Creating an Analysis Profile

In the left sidebar, click on **Projects** → **Analysis Profile**.

Analysis Profile Page On this page is provided the Create New Analysis Profile and the Import Analysis Profile buttons. Additionally, a table showing programming language analysis profiles already created by *Teamscale* as well as user-defined profiles if present are shown. Click the Create New Analysis Profile button to start the process. This leads to the *Create Analysis Profile View*.

Create Analysis Profile View Creating an analysis profile involves three steps illustrated in Figures 4.99a–4.101.

In the first step, the user sets a profile name and selects the analyzed programming languages. The choice of programming languages determines which options are available in the next steps since some metrics are not available for all languages.



(a) Step 1 (b) Step 2

Figure 4.99. Configuration Dialog for Analysis Profiles

Expert Options for Analysis Profiles

Disable clone-based move tracking	You can select this check box to disable tracking of moving or renaming files.
C/C++ predefined macros	Here you can specify macros which are not defined in the project source code. If those macros are not defined, *Teamscale* might issue warnings about undefined macros.
Enable search	Should *Teamscale* enable users to search in the project sourc code, commits, and issues?
Enable Redundant Literal Analysis	This checkbox enables the analysis of redundant literals in the source code. If enabled, teamscale generates findings for often-occurring string literals.

Figure 4.100. Configuring an Analysis Profile

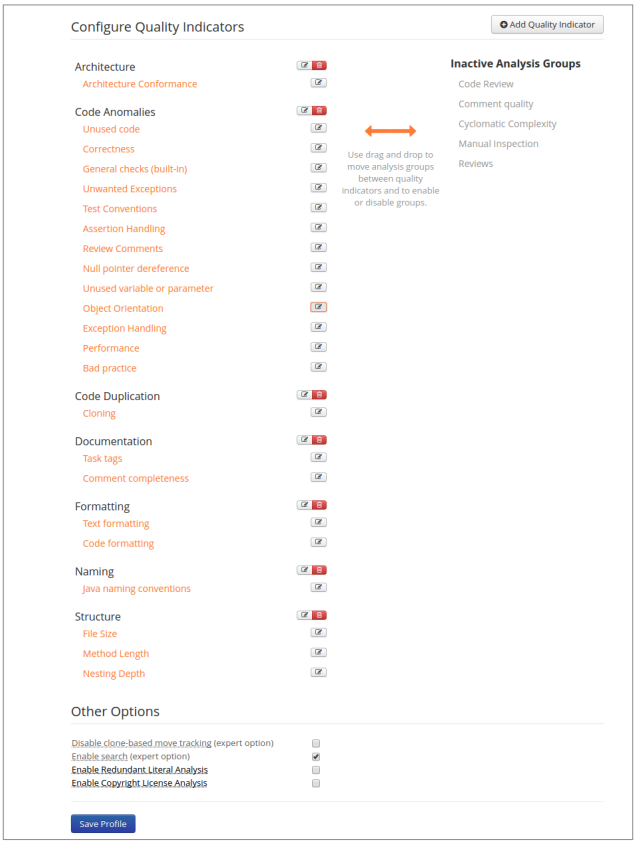


Figure 4.101. Configuration Dialog for Analysis Profiles (Step 3)

In the second step, the user selects which external tools are queried for analysis results. If, for example, the **Findbugs** check box[3] is chosen, *Teamscale* will display findings generated by the Findbugs tool in addition to its own code metrics. The **Custom External Findings** and

[3] Since the FindBugs- and SpotBugs report formats are compatible, the check box also controls SpotBugs findings.

non-code metrics features enable users to upload their own findings using a python API.

In the third step of the analysis profile setup, users can choose which code metrics *Teamscale* will compute and display. *Teamscale* groups code metrics in **analysis groups** and each group is part of a **quality indicator**.

In the analysis profile setup (Figure 4.101), the user can configure which analysis groups should be shown in which indicators, and which analyses groups should not be computed. To match your individual analysis needs, the analysis groups can also contain custom checks. In which analysis group a custom check is placed, depends on the declaration of the programmer within the implementation of the custom check. *Teamscale* comes with a variety of pre-written custom checks for various languages, see Appendix E. In the analysis group options, each custom check can be individually switched off or on by setting its severity to yellow or red.

Below the analysis profile *Teamscale* shows a set of expert options which you can use to fine-tune the analysis behavior. Some of these options depend on the chosen languages, so some options might be missing in your analysis profile dialog. After an analysis profile is set up, it is shown in the analysis profile list.

4.1.11.6 Action: Editing an Analysis Profile

On the **Analysis Profile Page**, click the edit button on the right side of the table along the programming language name. This navigates to the *Edit Analysis Profile View*.



Figure 4.102. **Thresholds** Page

Edit Analysis Profile View This view is illustrated in Figures 4.99a–4.101 similar to the *Create Analysis Profile View*. Under *Select Programming Languages*, you can make changes to the programming languages that a profile uses. When done, click the Continue button to proceed to the *Select Analysis Tools* view. Here, static analysis and code coverage tools, custom external metrics and findings can be activated or deactivated. Continue further to the *Configure Quality Indicators* view where changes to analyses groups and metrics can be effected. Finally, click Save Profile button to save changes.

4.1.11.7 Action: Managing Threshold Configurations

A threshold configuration specifies threshold values for metric. It influences how metric values are displayed, for example, in the dashboard. A threshold configuration is identified by its name and consists of thresholds structured in groups. Threshold configurations are project-independent. The **Thresholds** page gives an overview of all existing threshold configurations, see Figure 4.102. Figure 4.103 shows all available actions for a threshold profile.

4.1.11.8 Action: Creating a Threshold Configuration

To create a new threshold configuration, you have to select **Thresholds** in the projects perspective sidebar. This will lead you to the

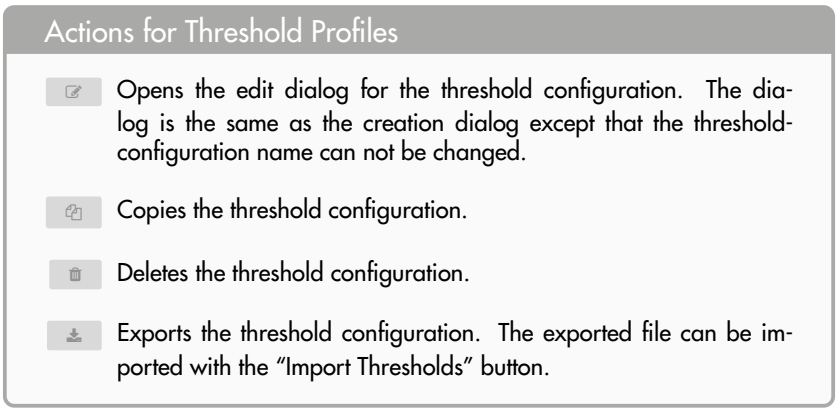


Figure 4.103. Actions Available in the Threshold Configuration Overview

Thresholds page which offers the Create new Thresholds button. This opens the *Create Threshold Profile View* as shown in Figure 4.104.

Create Threshold Profile View In this view, you have to choose a name for the threshold configuration and add metric groups to the threshold configuration. Next, you can add metrics to the metric groups. Each metric is shown in one line. The arrow next to the metric name explains in what direction the metric is assessed. If the arrow points down, low values indicate good values and vice versa.

The field **Custom Name** allows you to use another name than the metric's name when displaying the metric, for example, in the dashboard. Unless two rows with the same metric exist, the custom name is optional and the metric's name will be displayed in case the custom name is not specified.

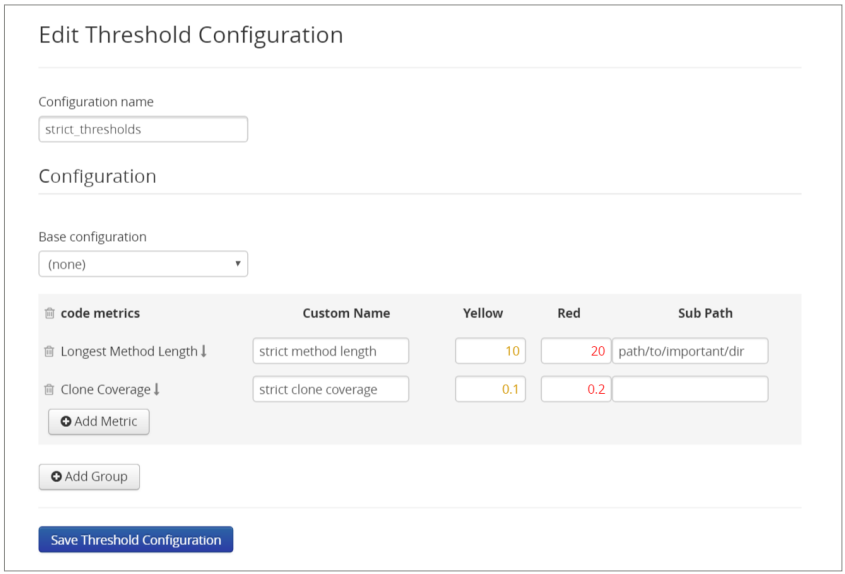


Figure 4.104. Setup View for an Example Threshold Configuration

The fields **Yellow** and **Red** enable you to specify thresholds. Depending on the metric, the fields can be either filled with an arbitrary numeric value or a percentage value (as number between 0 and 1). It is also possible to specify only one of the thresholds or none (in which case no thresholds will be displayed). Note that some metrics are not supposed to be assessed (such as »Lines of Code«); it is not possible to specify thresholds for those.

The field **Sub Path** allows restricting the assessment and validity of the thresholds to a certain part of a project. The sub path can be used to assess different parts of a project differently (e.g. restrict the threshold to the production code by excluding test code). It is also possible to use a path that points to an architecture component.

4.1.11.9 Action: Creating an External Metric

Situations may arise in which you want to observe a quality property of code and *Teamscale* does not provide such metric. For instance, observing code defect density (i.e. the number of code defects per specific number of lines of code) or frequency of bug-fixes on a file level. You can do this by first creating an external metric. Once created, it needs to be activated in the analysis profile that is used by the current project before it becomes visible in the **Metrics** (Figure 4.33) and **Delta** (Figure 4.87) perspectives, and also in the metrics-related dashboard widgets. To create an external metric, select **Project** → **External Metric Descriptions** in the left navigation sidebar. This leads you to the **External Metric Descriptions** page.

External Metrics Descriptions Page On this page, you will find a table of external metric descriptions if they have been configured already. Figure 4.105 shows the two external metrics described in the previous paragraph. Click the button New metric description. This provides the *Create External Metric Description* view, as shown in Figure 4.106.

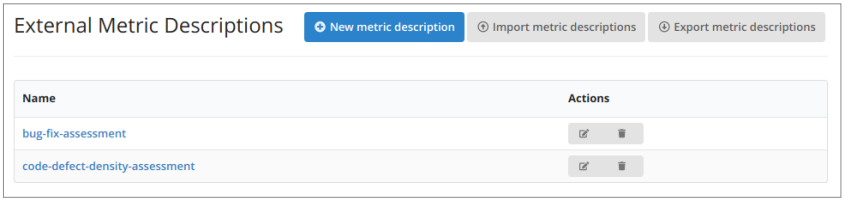


Figure 4.105. Overview Table of External Metrics

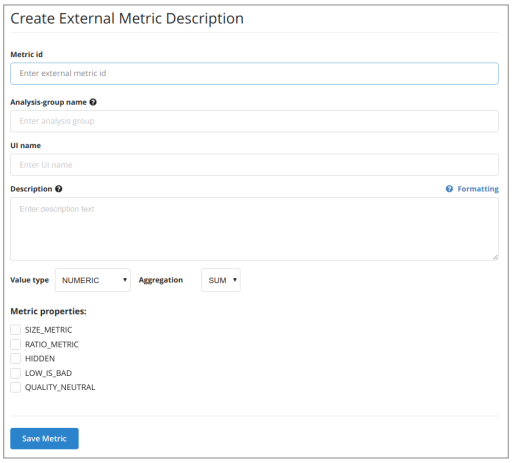


Figure 4.106. Create External Metric View

Create External Metric Page In this view, you have to enter a unique external metric id (that *Teamscale* uses internally to identify metrics), an analysis group name (collects similar metrics under a common group in analysis profile configuration), a UI name (some friendly name to be displayed on the user interface) and a description of the metric. The field aggregation indicates how values of this metric will be aggregated for directories, e. g. by selecting the minimum value or the maximum value or using the sum operator as aggregation.

You can select one or more metric properties under **Select Metrics Properties**. They are explained in Figure 4.107. For instance, the code defect density metric described previously is a SIZE_METRIC and qualifies as RATIO_METRIC if expressed per 100 lines of code.

Metric Properties

SIZE_METRIC	Indicates a size metric describing the amount of source code in some manner, e. g. 800 lines of code or 1000 source lines of code.
RATIO_METRIC	Indicates a ratio metric that only has values between 0 and 1. In the UI, these values are expressed as percentages.
LOW_IS_BAD	Indicates that lower numeric values represent lower quality. The default case is that lower is considered better.
QUALITY_NEUTRAL	Means that values of a metric are not assessed to have negative or positive impact on quality. This value contradicts with LOW_IS_BAD and you should select only one or the other.

Figure 4.107. Metric Properties

The field value type tells the kind of value for a metric. Possible values are explained in Figure 4.108.

Metric Value Types

NUMERIC	A numeric metrics consists of a numeric value e.g. file size.
TIMESTAMP	Indicates that a metric conveys time information e.g. last date modified. It's value is a number in milliseconds since 1.1.1970.
ASSESSMENT	An assessment metric stores an assessment with red, yellow and green.

Figure 4.108. Metric Value Types

On the **External Metric Descriptions** page, in the table of existing metrics under **Actions**, you can edit a metric by clicking the edit button . This leads to the *Edit External Metric View*. Also, you can use the delete button to delete a metric.

All external metrics once created require three additional steps before they are accessible in *Teamscale*. The first step is activating them in the analysis profile of the project. You can do this by editing the analysis profile, making changes only in the **Select Analysis Tools** section of the profile (see Section 4.1.11.6). There, you can select the checkbox besides **Custom External Metrics**. After clicking Continue underneath, a new quality indicator named *Custom Metrics* is visible in the **Configure Quality Indicator** section of the profile, organizing the new metrics under the specified analyses groups and providing more options for configuration.

In the second step, perform a re-analysis of the *Teamscale* project in the *Projects* view as explained in Section 4.1.11.1. Finally metrics-related data need then be uploaded for files through the *Teamscale* Python API.

4.1.11.10 Action: Import External Metric Descriptions

External metric descriptions can be imported into *Teamscale*. On the **External Metric Descriptions** page of the **Project** Perspective, you can click the Import metric descriptions button at the top of the page. This leads to the import view as shown in Figure 4.109. Browse to select a zip file containing metric descriptions and click Upload. Upon successful import, you will be directed back to the **Project** Perspective showing the imported metric descriptions.

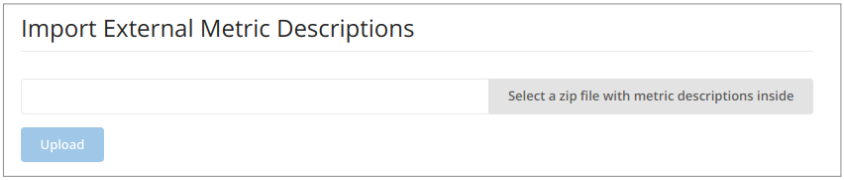


Figure 4.109. Import External Metric Descriptions

4.1.11.11 Action: Export External Metric Descriptions

At the top of the **External Metric Descriptions** page, you can click the Export metric descriptions button to download as a zip archive, the stored external metric descriptions.

4.1.11.12 Action: Create New External Finding Group

Teamscale internal analyses can produce findings in code. However, you can also upload findings from external tools into *Teamscale*. Take for instance an architecture-related issue of dependencies. One concern that could be analyzed for is high coupling, i.e. many dependencies between classes in different packages or components. The starting point to get external findings into *Teamscale* is creating ex-

ternal finding groups and finding descriptions. A finding group describes various related concerns in code, in the example architecture dependencies. Each singular issue is described in a finding description.

In the left sidebar, select **Projects** → **External Finding Descriptions**. This leads to the **External Finding Descriptions** page.

External Finding Descriptions Page On this page is presented a list of existing finding groups or an empty list if none has been created. At the top is the button New finding group.

When there are finding groups, each one is shown with a collapsible orange arrow icon on the left, its name, buttons Add finding description, to modify and to delete the finding group. In addition, underneath these buttons are a mapping string for the group and a table of matching finding descriptions. The mapping string is a regular expression that collects matching finding descriptions (i.e. descriptions whose type identifications are matched by the regular expression) into the table. Figure 4.110 shows the finding group described in the previous paragraph. Finding descriptions that cannot be matched into available groups are listed under the Unmatched group.

To create a new finding group, you have to click the New finding group button. This launches the **Add Finding Group** dialog.

Add Finding Group Dialog Figure 4.111 presents this view. You can type in a name for a findings group and also enter a regular

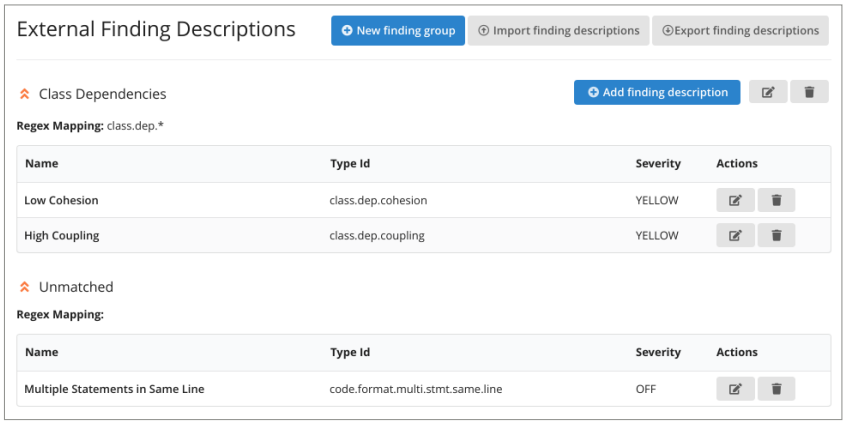


Figure 4.110. Overview Table of External Findings

expression for the mapping. This mapping is used to find matching finding descriptions to be associated with this group. (see Section 4.1.11.13). A sample regular expression for the class dependency finding group could be »class.dep.*«. This will select finding descriptions which have »class.dep« as prefix. Click OK to save.

4.1.11.13 Action: Create New External Finding Description

Finding descriptions are more specific in describing a concern. Take for instance the finding group for class dependencies as shown in Figure 4.110. The description »Low Cohesion« describes a situation where few dependencies exist between classes in the same component or package.

To create a finding description associated with a group, you have to click the Add finding description button beside the group name. It launches the Add finding description dialog.

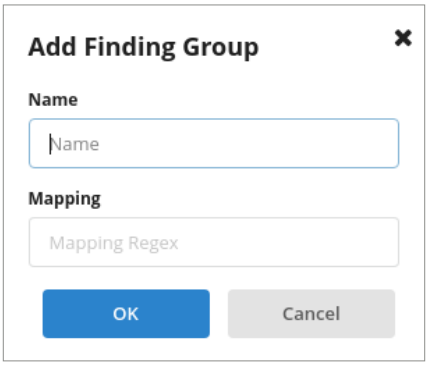


Figure 4.111. Create External Findings Group Dialog

Add Finding Description Dialog Figure 4.112 presents this dialog. It provides four fields: **Type Id**, **UI friendly Name**, **Description** and **Severity**.

The **Type Id** should have a unique value and match the regular expression of an existing finding group to which it will be associated. This value will be used internally by *Teamscale* and cannot be modified later. Afterwards, you can enter a short, **UI friendly Name** for identifying this description in analysis profile configurations. The **Description** field should provide some detailed description as it will explain occurrences of findings in the **Finding Detail View**. Finally, you have to specify the **Severity**. This indicates the level of treatment this finding deserves and how it should be handled. Possible values are described in Figure 4.114. Click OK to save.

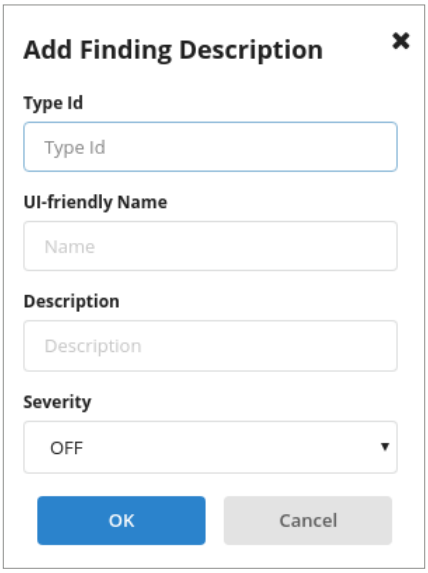


Figure 4.112. Create External Findings Description Dialog

Severity Configuration

OFF	Finding will be hidden.
YELLOW	Finding indicates a warning.
RED	Finding is critical.

Figure 4.114. Options to Set the Severity of a Finding

Two more steps are needed before uploading findings into *Teamscale*: activating the external findings feature in the analysis profile and then re-analyzing the project (see Section 4.1.11.1).

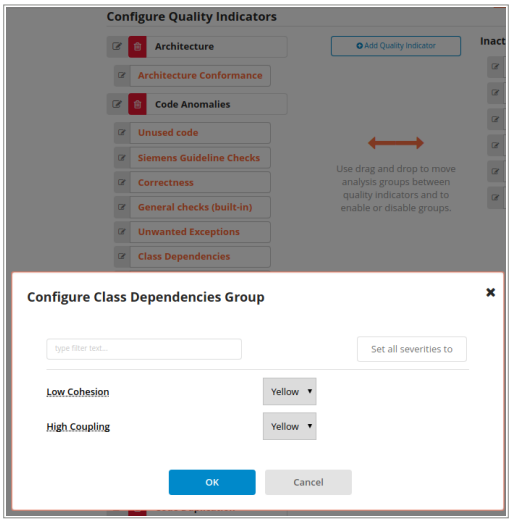


Figure 4.113. Activated Finding Group & Description

In order to activate the findings group in the analysis profile, you need to edit the analysis profile (see Section 4.1.11.6) in the *Edit Analysis Profile View*. Under **Select Analysis Tools**, enable the checkbox **Custom External Findings** and click Continue . The created findings groups will be shown under *Quality Indicators*. Click Save Profile to finalize. If prompted with a dialog-box for reanalysis, click **Expert Settings** and select the **Only schema update (expert option)** checkbox. Also select the **Re-analyze** checkbox and click OK. After re-analyzing the project, findings can be uploaded using *Teamscale* Python API (http://cqse.github.io/teamscale-client-python/how_to_findings.html#upload-the-findings). The new findings are processed and visible in the **Findings** Perspective. On its sidebar the new class dependencies finding group is displayed with other groups for filtering findings, as shown in Figure 4.115. Also the findings detail from

the perspective indicate external findings and their description, see Figure 4.116

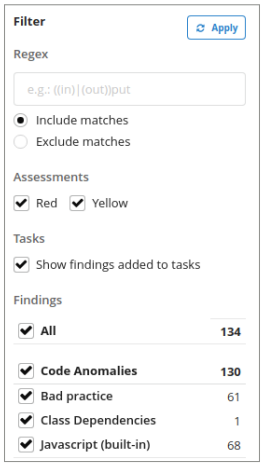


Figure 4.115. New findings Group in Sidebar of Findings Perspective



Figure 4.116. Findings Detail View Showing New finding Group

4.1.11.14 Action: Import External Finding Descriptions

It is possible to import external finding descriptions bundled in a zip file into *Teamscale*. On the **External Finding Descriptions** page of the **Project** Perspective, you can click the Import finding descriptions button at the top of the page. This leads to the import view as shown in Figure 4.117. Browse to select the zip file and click Upload. Upon successful import, you will be directed back to the **Project** Perspective showing the imported findings.



Figure 4.117. Import External Finding Descriptions

4.1.11.15 Action: Export External Finding Descriptions

The **External Finding Descriptions** page as shown in Figure 4.110 presents the Export findings button for downloading stored finding groups and descriptions. Clicking it saves the findings groups and descriptions in a zip file format on to your computer.

4.2 IDE Integration

Teamscale provides an integration with three major IDEs: Eclipse, IntelliJ IDEA and Microsoft Visual Studio. The *Teamscale* IDE plug-ins fetch the analysis results from the *Teamscale* server and annotate the source code with quality-related information (e. g. findings, test cov-

erage) directly within the editor. This avoids a context switch for developers giving them access to quality data right in their accustomed working environment.

In the following we describe the installation and configuration for each of the *Teamscale* IDE plug-ins.

4.2.1 Eclipse

The Eclipse plug-in for *Teamscale* is available in the *Eclipse Marketplace*[4]. The Marketplace page also states the current system requirements (especially supported versions of Eclipse). To install the plug-in, drag the *Install* button to your running Eclipse workspace. Follow the instructions of the installation wizard and restart Eclipse when prompted.

After installing the plug-in, you can configure a *Teamscale* server via the following steps:

1. Open the Eclipse preferences dialog, navigate to the *Teamscale* settings (*Window* → *Preferences...* → Teamscale) and click on *Add*
2. Enter a descriptive name and the URL of the *Teamscale* server
3. Enter your user name and IDE access key (you can generate an access key via the Web UI on your personal profile page accessible via the user icon in the upper right corner)
4. Test the connection by clicking on *Validate*.
5. Close the dialog by pressing *OK*.

[4] https://marketplace.eclipse.org/content/teamscale-eclipse-plugin

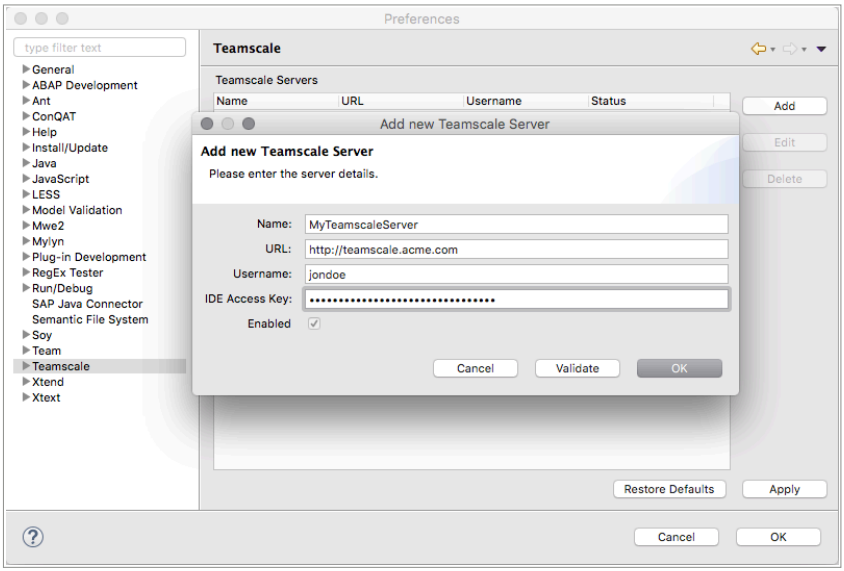


Figure 4.118. *Teamscale* Preferences in Eclipse

To connect an Eclipse project to *Teamscale* and obtain quality data such as findings, perform the following steps:

1. Open the project properties by right-clicking on the project and selecting *Properties*.
2. On the left side, switch to the Teamscale entry.
3. Click on *Enable* Teamscale *for this project* and select the appropriate server.
4. Click on *Automatically configure* Teamscale which will automatically identify the correct mapping to the code in *Teamscale*.

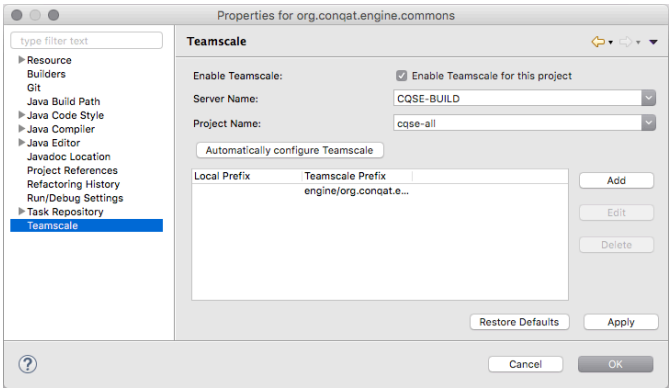


Figure 4.119. *Teamscale* Project Properties in Eclipse

After connecting a project to *Teamscale* you can enable the display of findings in the menu bar via Teamscale → *Load findings for open files*.

4.2.1.1 Pre-commit Analysis

The Eclipse plug-in can be used to analyze the changes prior to committing them to the repository. This functionality is called »pre-commit analysis«. Only the changed files since the last commit that is present on the Teamscale server are uploaded and analyzed. The delta is currently limited to a maximum of 10 local commits. To compute the delta *Teamscale* connects to the version control system configured for the projects. The pre-commit analysis is not available, if no or more than one version control system is configured for the projects in the Eclipse Workspace belonging to the same *Teamscale*-Project. Pre-commit analysis is triggered by selecting Teamscale → *Upload data for pre-commit analysis* from the Eclipse main menu or via the shortcut »Ctrl+9«.

The findings delta is reported back after a few seconds via a popup as shown in screenshot 4.120. Details of the delta are provided via the links in the popup. Note that some limits exist for the precommit analysis in order to protect the *Teamscale* server from being overloaded. More information is available in the administration section (see Section 5.4.3).

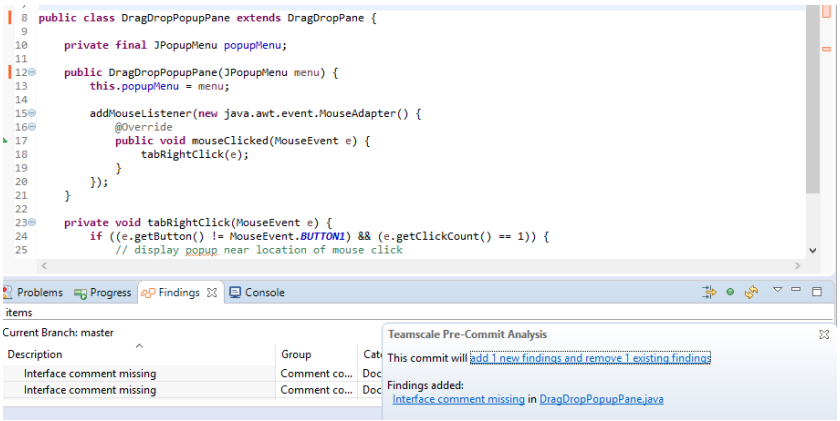


Figure 4.120. Pre-commit Analysis Findings Delta

4.2.2 Visual Studio

In addition to the Eclipse plugin, *Teamscale* also provides an integration within Visual Studio.

4.2.2.1 Download & Installation

You can download the Teamscale Visual Studio plugin by using the integrated extension manager in Visual Studio by searching for Team-

scale or by downloading the Plugin from the Microsoft Visual Studio Gallery[5].

The plugin is compatible with Visual Studio versions 2012 to 2017.

Installation via the extension manager has the added benefit of automatically being notified about new updates, as they get published.

In case you downloaded the extension manually you can install it as follows:

1. Double click on the Teamscale.vsix Extension file.
2. In the dialog of the VSIX Installer, click Install.
3. Restart Visual Studio.

4.2.2.2 Configuring a Teamscale server

Next you have to configure a Teamscale server, so that the plugin knows from where to fetch the findings.

1. Click Tools → Teamscale Options...
2. Click Add.
3. In opened dialog enter the Teamscale Server ID (this name is later references from project mappings), the Teamscale Server URL, your username and password/api key.
4. Click Save. The dialog should close if validation of the connection was successful.

[5] https://visualstudiogallery.msdn.microsoft.com/1b425e5b-014b-449c-a327-b1dc54af733f

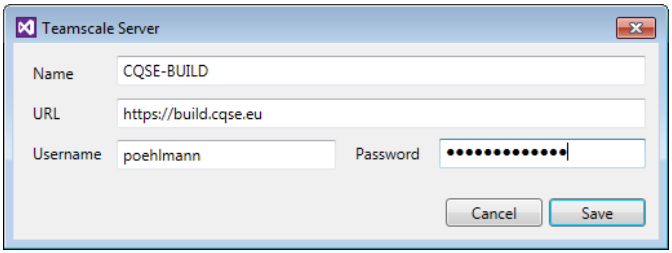


Figure 4.121. *Teamscale* Server Configuration in Visual Studio

4.2.2.3 Configuring the project mapping

Next you have to supply a project mapping, so that the plugin knows which Teamscale Project belongs to which code on the hard drive.

1. Click Tools → Teamscale Options...
2. On the left hand side select Teamscale → Project Mapping.
3. Click Add.
4. In the opened dialog select the Teamscale Server and Project. If branch analysis has been configured on the Teamscale server, select the relevant branch in the combo box. As default value the default branch name specified on the Teamscale server will be used. Choose the local folder you want to map the Teamscale Project to and optionally specify a sub-path of the project on the server (this can be left blank in most cases).
5. Click Save.
6. Reopen the currently opened Solution if it corresponds to the configured project.

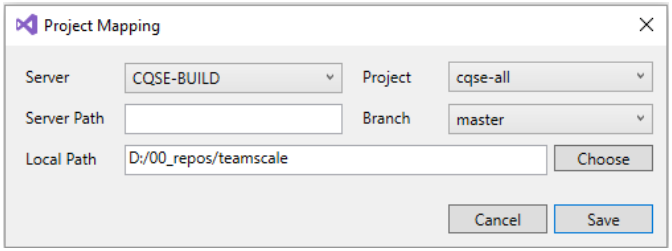


Figure 4.122. *Teamscale* Project Mapping in Visual Studio

4.2.2.4 Inspecting findings

After the previous steps have been completed successfully you should now be able to explore findings in your code.

1. Findings in the opened document should be marked with a red bar on the left margin.
2. Hovering with the mouse over the margin will show a tooltip containing the description of the finding.
3. To show all findings of a file, either right click in the current editor or the Solution Explorer and select Teamscale → Show Findings.
4. In the findings list one can double click to jump to the finding in the code, open it in the browser or blacklist the finding as shown in the screenshot below.

4.2.3 IntelliJ IDEA

Teamscale also integrates with IntelliJ IDEA and many other IntelliJ IDEs, such as WebStorm and PyCharm.

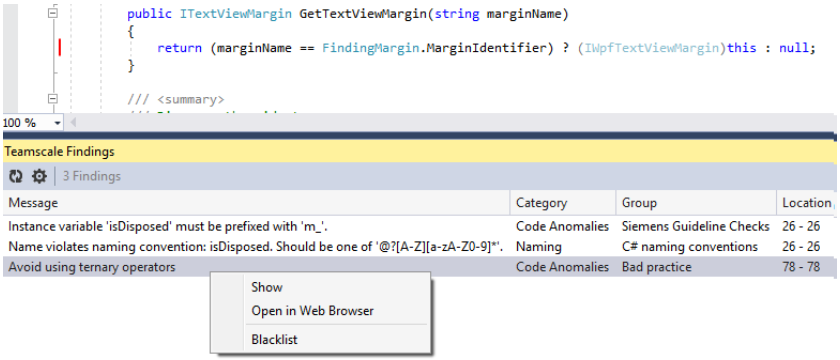


Figure 4.123. *Teamscale* Findings in Visual Studio

4.2.3.1 Download & Installation

The IntelliJ plugin has been tested and used on IntelliJ IDEA, WebStorm and PyCharm IDEs, but should also work on other IntelliJ IDEs. To download and install the *Teamscale* IntelliJ IDEA plug-in, perform the following steps:

1. Inside an IDE browse the Jetbrain Plugin Repositories via *File* → *Settings* → *Plugin* → *Browse Repositories*.
2. In the dialog box, search for »Teamscale IntelliJ Plugin«. On the right view, click the install button.
3. Restart the IDE to conclude installation.
4. If the plug-in installed properly, you should see a Teamscale menu entry in the main menu bar.

4.2.3.2 Configuring a *Teamscale* Server Connection

You need to perform the following actions to configure a *Teamscale* server, see Figure 4.124:

1. Open the IntelliJ IDEA Settings via the menu *File* → *Settings*.
2. Expand *Tools (1)* and select Teamscale *(2)* on the left hand side.
3. Use the + button to add a new *Teamscale* server (3).
4. Fill the dialog (4) with the appropriate values. Note: For security reasons you may not use your *Teamscale* password for authentication. Instead, an IDE Access Key is required. By clicking on the Get Key button, you can navigate to the *Teamscale* web interface and generate a personal access key.
5. Click the *Validate Connection (5)* button to ensure successful connection the *Teamscale* server.
6. Click *Apply* and *OK* to save the settings.

4.2.3.3 Enable Plugin for IDE

The plugin can be enabled for the IDE (i.e. for all opened projects). When turned on, plugin can communicate with Teamscale and fetch quality results. Activate this by clicking *Teamscale menu* → *Fetch Findings from Teamscale Server*. Ensure that this option is checked.

4.2.3.4 Connecting an IDE Project to a *Teamscale* Project

In addition to enabling the *Teamscale* IntelliJ IDEA plug-in, it needs to know which *Teamscale* project the IDE project corresponds to. This

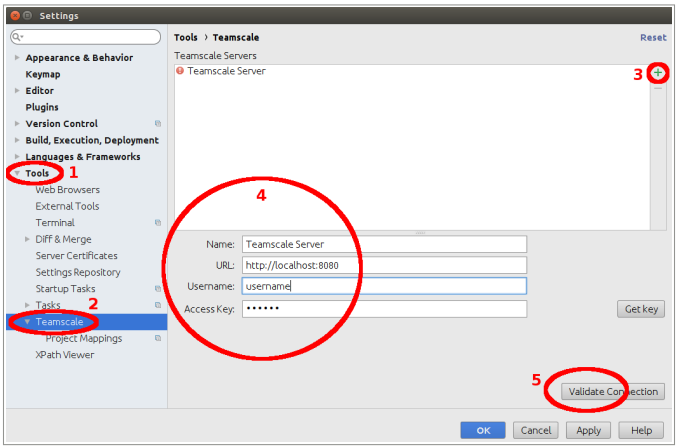


Figure 4.124. Configuring a *Teamscale* Server Connection

mapping is done in the mapping settings page. To access the mapping settings page, you have to perform the following steps:

1. Click *File* menu → *Settings*. Alternatively, you can click *Teamscale* menu → *Configure Plugin ...*
2. Expand *Tools* → Teamscale. Click Project Mappings.

On the mapping settings dialog (see Figure 4.125), you have to carry out the following steps:

1. Check the option »Enable Teamscale quality analysis for project«.
2. Select the appropriate *Teamscale* server and project values.
3. Create prefix mappings by clicking on the Auto-create Prefix Mappings button
4. Click *Apply* and *OK* to save settings.

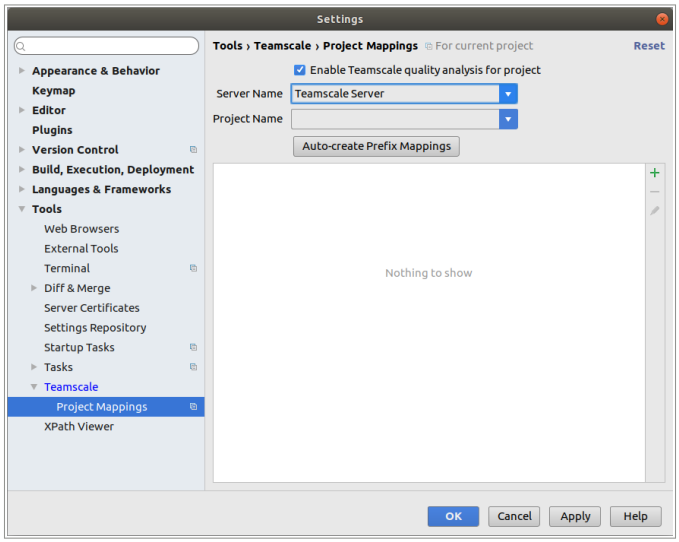


Figure 4.125. Creating Project Mappings

File Mapping Under normal circumstances, the plugin automatically determines how to map the module files to the files in the *Teamscale* project. If the automatic mapping failed, a message is shown. To manually configure mappings between the local and the server's structure, you can add them in the module settings:

1. Open the mapping settings page as mentioned above.
2. Uncheck the option »(Re)create prefix mappings«.
3. Create a mapping by clicking the + sign on the right side. Enter the mappings prefixes in the upcoming dialog. The Local Prefix is a part of your local directory structure. The *Teamscale* Prefix is a part of the project directory structure on the *Teamscale* server.

To check if everything is configured correctly, open the *Teamscale* Findings Tool Window via the menu (*View* → *Tool Windows* → Teamscale *Findings*), see Figure 4.126.

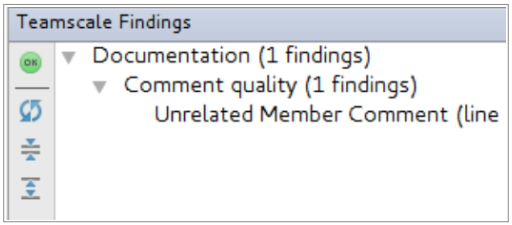


Figure 4.126. *Teamscale* Findings Tool Window

Now you can select any resource that has findings and wait for findings to appear in the *Teamscale* Findings Tool Window. The symbol on the top of the left hand side of the Tool Window will indicate the server's reachability state. Hover the mouse over the icon to get additional information. A click on the icon opens up the *Teamscale* settings.

Refresh Findings In order to get an updated view of the findings of a selected folder or a file after changes in a file system, for instance by switching to a new repository or branch, click the button in the *Teamscale* Findings Tool Window.

4.3 Integration and Extension APIs

This section describes the interfaces provided by *Teamscale* for integration and extension.

4.3.1 REST API

The *Teamscale* server exposes all of its functionality as REST web services. All functionality used in the Web UI and the IDE integrations is built upon these services and can also be used programmatically.

4.3.1.1 Calling Conventions

Service types We differentiate between two types of services:

- *Project services* allow querying data from a single project, for example the list of findings. These services need the project name as a parameter and thus, adhere to the URL form of p/[project-id]/[service-name].
- *Global services* are not specific to a single project and can be used to query all projects of an instance. These services are called directly by appending the service name to the base URL.

Many services take their main argument (e.g. a code path) directly attached to the main URL. Some services also accept additional parameters that are passed using the HTTP convention: Parameters are separated by a *?* from the main URL and separated by *&*.

Authentication Authentication for REST services uses HTTP basic authentication with the user name and the *IDE Access Key* as password. The service call is then executed with the permissions of this user.

Result representation Most services can return either XML (default) or JSON output. Which output is used is determined by HTTP's *Accept* header.

Examples For first experiments, the tool *curl* can be used to query services from the command line. For this we assume the service to be running on http://localhost:8080/ the user name *bob* and an access key of *e434mikm35d*.

To get the list of all projects visible to the user, you can use

```
curl --user bob:e434mikm35d
"http://localhost:8080/projects"
```

To get JSON output instead of XML:

```
curl --user bob:e434mikm35d -H "Accept: application/json"
"http://localhost:8080/projects"
```

To get the information about the configured metrics for a project with id *mon*:

```
curl --user bob:e434mikm35d
"http://localhost:8080/p/mon/metric-schema"
```

And, finally, to get the actual metric values for a branch *rel42* at timestamp *374343221* for the same project:

```
curl --user bob:e434mikm35d
"http://localhost:8080/p/mon/metrics?t=rel42:374343221"
```

Common pitfalls Always make sure that you use the IDE access key as password and not the actual password of the user when making REST calls. Also, the user needs to have sufficient permissions to view or change the corresponding data.

If something goes wrong, you can often find additional information in the service log, available via the Web UI in the System perspective.

4.3.1.2 Available REST Services

The list of REST services is available from a running *Teamscale* instance by appending *servicedoc.html* to the base URL and opening the page in a web browser. For a local test instance, the URL might look like http://localhost:8080/servicedoc.html.

To get an idea which service might be useful, you can observe the calls made from the Web UI using the development tools of your web browser.

4.3.2 Post-Commit Hooks

By default, *Teamscale* uses polling to regularly check for new revisions on the server. Depending on the polling interval you configured, this can lead to a large delay between the actual commit and the update of the data in *Teamscale*, or to a very high load on your version control system server (especially with many projects).

One solution to this problem are post-commit hooks, which are supported by many version control systems. For this, configure your *Teamscale* project with a large polling interval. Polling will still happen from time to time as a fallback if the post-commit hook failed for some reason. Then you have to configure your version control system to call the following URL after each commit:

```
http://teamscale.company.com/post-commit-hook?
... user=USERNAME&access-token=TOKEN&repository=REPOSITORY
```

Here the part `http://teamscale.company.com/` is the URL of your teamscale server. The *USERNAME* is the name of a user that has build permissions, i. e. is in a group with role *Build*. The *TOKEN* is the access token for this user, which is accessible via the user administration page. Finally, REPOSITORY is the URL of the repository as configured in *Teamscale*. For example, if you have a project reading from the Subversion repository `http://svn.company.com/project1/trunk/foo`, the URL may be `http://svn.company.com/project1/trunk/foo` or any prefix, such as `http://svn.company.com/project1/` or even `http://svn.company.com/`. Note that all projects for which the prefix matches will be checked for updates, so a more specific prefix can help to reduce the server load. However, being too specific does not work. If you pass `http://svn.company.com/project1/trunk/foo/some/file` as repository parameter, the prefix match will fail and no update is triggered.

You can use both GET and POST requests to call the URL.

Subversion Subversion supports the execution of arbitrary (shell) scripts for various hooks. The hook that should be used is called *post-commit*. To access the URL you can use the tools *wget* or *curl*. See the SVN book for more details on configuration of commit hooks.

Team Foundation Server In Team Foundation Server users can configure commit hooks with custom alerts and service hooks (TFS 2015+). Both are available via the admin settings for a team project (gear icon in the top loft corner).

For both methods the `REPOSITORY` parameter of the URL to the Teamscale commit hook service (as described above) is `TFS_URL/TEAM_COLLECTION/FOLDER_PATH`, e.g.

`http://yourcompany.com/tfs/DefaultCollection/$/MyProject/Dev/Source`.

Service Hooks *Service Hooks* are the successor of *Alerts* and available in Visual Studio Team Services and Team Foundation Server 2015 onwards. They are easier to set-up and provide debug information if a request fails.

Switch to the *Service Hooks* tab, create a new subscription and select *Web Hooks* as type of service. Change the trigger type to *Code checked in* and enter the TFS repository path you want to monitor for changes (see figure 4.127). If you are hosting a Git repository via Team Foundation Server you have to select *Code pushed* as event type and simply select the repository you want to monitor for changes. Optionally you can restrict the event to only be fired if a specific branch is pushed.

After clicking *Next* you have to enter the URL of the commit hook (see figure 4.128). All other fields can be left empty and it is not required to send any further data to teamscale at all.

The user who creates the subscription needs to have the TFS permission *Edit subscriptions and View subscriptions*. See https://www.visualstudio.com/en-us/get-started/integrate/integrating-with-service-hooks-vs for further information.

Alerts Switch to the *alerts* tab and create a new alert of type *A file is checked in under a specified path* (see figure 4.129). Then alter the following fields as shown in figure 4.130:

- *Name*: Alter the name to something that reminds you this is the commit hook for Teamscale.

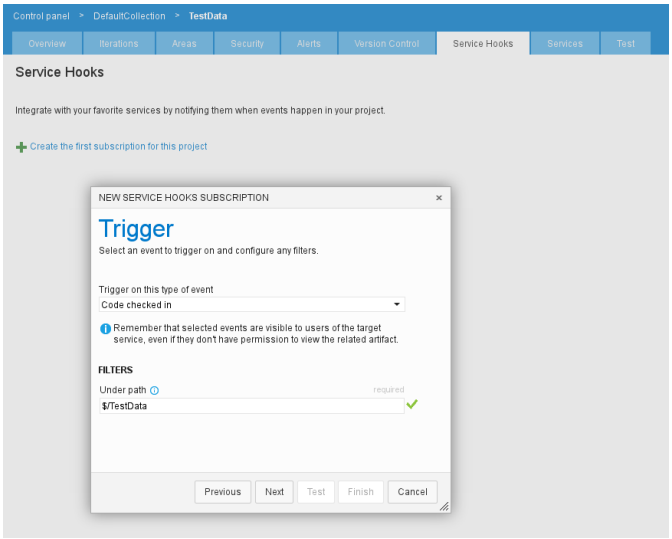


Figure 4.127. Create a trigger that listens on code changes.

- *Send To*: The commit hook url.
- *Format*: Change to SOAP.
- *Filter Value*: Change from `$/` to the folder path used in the commit hook, e.g. `$/MyProject/Dev/Source`.

If you are hosting a Git repository via Team Foundation Server you have to select the alert type *Push > A commit is pushed to a specified repository*. The *Send To* and *Format* fields have to specified in the same way as described above.

If you receive an error on save saying that the user needs the right *Create a SOAP subscription*, an administrator has to issue the following command (e.g. from VisualStudio Command Promt):

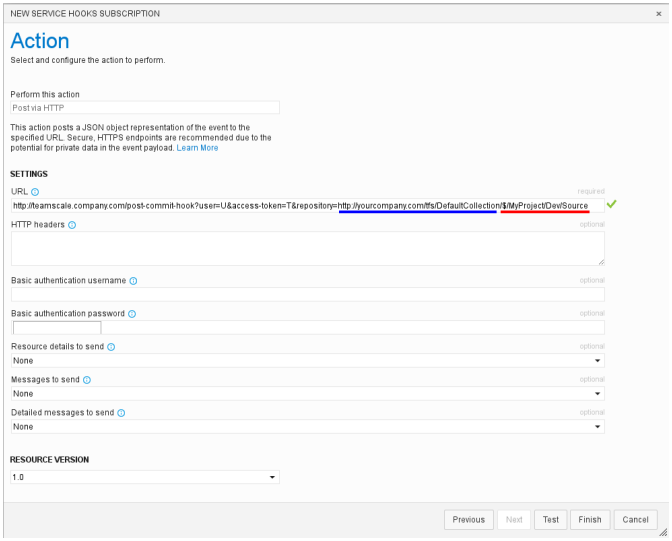


Figure 4.128. Configure the service hook action.

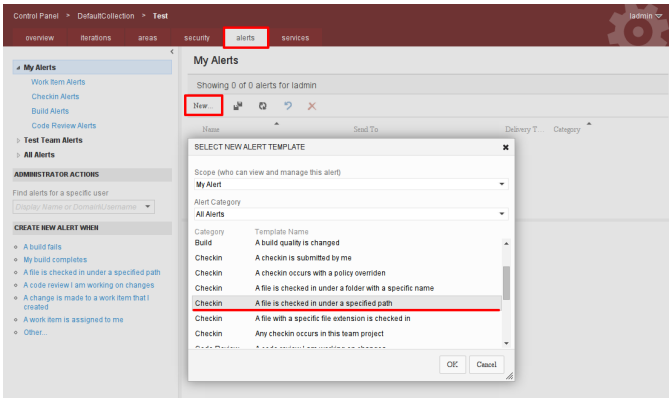


Figure 4.129. Create a new TFS commit alert.

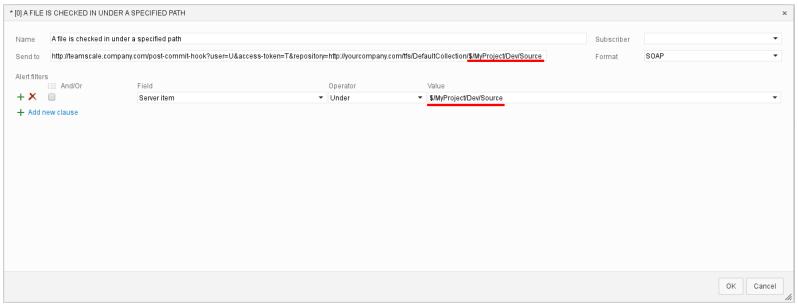


Figure 4.130. Configure the TFS commit alert.

```
tfssecurity.exe /a+ EventSubscription $SUBSCRIPTION:
CREATE_SOAP_SUBSCRIPTION n:DOMAIN\USERNAME ALLOW /collection:
TFS_URL/TEAM_COLLECTION
```

Ensure to substitute `DOMAIN` and `USERNAME` with the login of the account you want to create the alert for and `TFS_URL/TEAM_COLLECTION` with the url to your TFS Team Collection.

4.3.3 External Analysis Results Upload (REST)

The REST service »external-report« allows uploading results from analysis tools like FindBugs/SpotBugs[6] or different coverage tools. The following command line illustrates a file upload using *curl*. You need to adapt the parameter values for your environment:

```
curl --user bob:e434mikm35d -F "report=@./MyReport.xml"
-F "format=FINDBUGS" -F "message=my message"
-F "partition=myPartition"
localhost:8080/p/my-project-id/external-report
```

[6] https://spotbugs.github.io/

Details about the service's parameters can be best retrieved via the online service documentation as described above in section Section 4.3.1.2. Table 4.1 lists the currently supported report formats and the corresponding values to be used for the »format« parameter. Our own custom formats are documented in Table 4.2, Table 4.3 and Table 4.4. In addition to the common pitfalls listed in Section 4.3.1, you need to check if the external tool is activated in the project's analysis profile.

Sessions If you want to upload multiple reports in a single logical commit, you should use *Teamscale*'s session concept. For session handling, the service *external-analysis* is used. The following example uploads two different reports as a single commit using *curl*. For more details, see the service documentation (section Section 4.3.1.2).

```
curl -u bob:e434mikm35d
"http://localhost:8080/p/my-project-id/external-analysis
?message=my+message&partition=myPartition"
```

This command wil return the session id, in this case *Sdfb38cb0-c348-40fb-82f0-144bf0e41beb*. Using this session, we can upload multiple reports.

```
curl -u bob:e434mikm35d -F "report=@./MyReport.xml"
-F "format=FINDBUGS"
-F "session=Sdfb38cb0-c348-40fb-82f0-144bf0e41beb"
http://localhost:8080/p/my-project-id/external-report
```

```
curl -u bob:e434mikm35d -F "report=@./OtherReport.xml"
-F "format=FXCOP"
-F "session=Sdfb38cb0-c348-40fb-82f0-144bf0e41beb"
http://localhost:8080/p/my-project-id/external-report
```

Finally, we commit the session using a POST request:

```
curl -X POST -u bob:e434mikm35d
http://localhost:8080/p/my-project-id/external-analysis/Sdfb38cb0-
c348-40fb-82f0-144bf0e41beb
```

4.3.4 External Analysis Tools

Teamscale also allows to integrate external analysis tools that are not inherently incremental. For this, the analysis tool is executed (for example in a nightly build) and the results are uploaded using *Teamscale*'s service layer. The upload is performed preferably via the REST API as described in Section 4.3.3. As an alternative, the Teamscale Python client can be used which are described in the following sections.

When integrating external tools, it is important that these tools are also activated in the analysis profile, as otherwise findings are imported but not mapped to a quality indicator and, hence, not displayed. Enablement of rules can either be configured in each tool or via the analysis profile.

4.3.4.1 General Approach

Teamscale Python Client For those external tools for which an upload via the *Teamscale* Python client is possible, it is recommended you use this approach.

The Python client is available at https://github.com/cqse/teamscale-client-python and can be installed via Python's package manager pip:

```
pip install teamscale-client
```

Table 4.1. Supported Formats for Upload

Report Format	Parameter Value
Bullseye	BULLSEYE
Clang	CLANG
Cobertura	COBERTURA
Cppcheck	CPPCHECK
FindBugs and SpotBugs	FINDBUGS
FxCop	FXCOP
Gcov	GCOV
Istanbul JSON report	ISTANBUL
JUnit	JUNIT
JaCoCo	JACOCO
JetBrains dotCover	DOT_COVER
Lcov	LCOV
Microsoft Coverage	MS_COVERAGE
PC-Lint	PCLINT
Pylint	PYLINT
Roslyn	ROSLYN
SpCop	SPCOP
Testwell CTC++	CTC
Teamscale Generic Findings	GENERIC_FINDINGS
Teamscale Non-Code Metrics	GENERIC_NON_CODE
Teamscale Simple Coverage	SIMPLE
Visual Studio Coverage	VS_COVERAGE
XR.Baboon	XR_BABOON
XUnit	XUNIT
Xcode Test Coverage	XCODE

Table 4.2. Example for JSON File Format for Teamscale Generic Findings

```
[
        {
                "path": "com/acme/Foo.java",
                "findings": [
                        {
                                "findingTypeId": "my-
                                    finding-1",
                                "message": "Invalid␣
                                    construct",
                                "startLine": 14,
                                "endLine": 14
                        }
                ]
        },
        {
                "path": "com/acme/Foo2.java",
                "findings": [
                        {
                                "findingTypeId": "my-
                                    finding-2",
                                "message": "Range␣not␣
                                    defined",
                                "startLine": 12,
                                "endLine": 44,
                                "assessment": "RED"
                        },
                        {
                                "findingTypeId": "my-
                                    finding-3",
                                "message": "Findin␣with␣
                                    offsets",
                                "startOffset": 1234,
                                "endOffset": 2345
                        }
                ]
        }
]
```

Table 4.3. Example for JSON File Format for Teamscale Non-Code Metrics

```
[
        {
                "path": "metric1/value1",
                "count": 15
        },
        {
                "path": "metric1/value2",
                "count": 42,
                "content": "some␣text",
                "assessment": { "RED": 5, "YELLOW": 3, "
                    GREEN": 1 }
        }
]
```

Table 4.4. Example File for Teamscale Simple Coverage

```
com/acme/Foo.java
5
7
9-12
com/acme/Foo2.java
45
46
47
48-50
51-55
```

You can find examples of how to use the Python client here: https://github.com/cqse/teamscale-client-python/tree/master/examples

4.3.4.2 Clang Static Analyzer

The Open Source LLVM clang compiler includes a static analyzer[7] that performs dataflow analysis on C/C++ code. Analysis is started using the `scan-build` script, which wraps the make process. To enable output in *plist* format (which we require), the option `-plist` must be passed to the `scan-build` command. Additionally, you might want to enable additional checkers using the `-enable-checker` command. To obtain a list of all available checkers, run the following command:

```
clang -cc1 -analyzer-checker-help
```

Note that it is sufficient to provide a package/category to the analyzer to enable more checks. A typical invocation of the static analyzer might look like this:

```
scan-build -plist
-enable-checker alpha
-enable-checker core
-enable-checker cplusplus
-enable-checker deadcode
-enable-checker security
-enable-checker unix
make
```

To upload analysis results, you should use the external-report service as described in section 4.3.3.

[7] http://clang-analyzer.llvm.org/

4.3.4.3 FindBugs and SpotBugs

FindBugs[8] is an Open Source bug pattern search tool working on the byte code of Java programs. This means that in order to use FindBugs, you have to first compile the code.

SpotBugs[9] is the successor of FindBugs, which is no longer maintained. Since SpotBugs reports are compatible with the FindBugs report format, they can be uploaded with the same analysis-tool name (FINDBUGS).

To upload analysis results, you should use the external-report service as described in section 4.3.3.

4.3.4.4 JaCoCo

JaCoCo[10] is an Open Source code coverage tool for Java. Teamscale allows to upload XML coverage reports produced by JaCoCo. To create an XML report with the JaCoCo Ant Tasks use the *xml* sub element in the *jacoco:report* task.

```
<jacoco:report>
    <executiondata>
        <file file="jacoco.exec"/>
    </executiondata>
    <structure name="Example Project">
        <classfiles>
            <fileset dir="classes"/>
        </classfiles>
        <sourcefiles encoding="UTF-8">
            <fileset dir="src"/>
        </sourcefiles>
```

[8] http://findbugs.sourceforge.net/
[9] https://spotbugs.github.io/
[10] http://www.eclemma.org/jacoco/

```
    </structure>
    <xml destdir="report"/>
</jacoco:report>
```

For detailed information on the usage of the JaCoCo Ant tasks, please refer to the JaCoCo user guide to find out how to generate an XML coverage report.

To upload JaCoCo coverage data, you should use the external-report service as described in section 4.3.3.

4.3.4.5 Cobertura

Cobertura[11] is an Open Source code coverage tool for Java. Teamscale allows to upload XML coverage reports produced by Cobertura. You can generate an XML report with the Cobertura Ant task *cobertura-report* by setting the parameter *format* to *xml* as follows:

```
<cobertura-report format="xml"
   destdir="${coveragereport.dir}"
   srcdir="${src.dir}" />
```

For further details on the usage of the Cobertura Ant tasks, please refer to the Cobertura user manual.

To upload Cobertura coverage data, you should use the external-report service as described in section 4.3.3.

[11] http://cobertura.github.io/cobertura/

4.3.4.6 gcov

The tool *gcov*[12] measures line coverage for C/C++ code compiled with the GCC compiler. To determine the code coverage, the code must be compiled using the switch `--coverage`. This creates files with the extension *gcno* during compilation. During execution of the code, the coverage data is stored into files with ending *gcda*. The program *gcov* is then used to convert the coverage data into readable files with the extension *gcov* as follows:

```
gcov -o <OBJ_DIR> -l -p <SOURCE_FILES>
```

Here the `<OBJ_DIR>` is the directory containing the object files. For projects with many subdirectories it should be noted that the parameter must point directly to the directory containing the matching object. The relative path of the source files is not used. The paramters `-l` and `-p` are used to ensure that files with the same name but different directories do not collide.

Alternatively, the python script *gcovr*[13] can be used to execute gcov for complex project layouts. To preserve the gcov files, the parameter `--keep` must be used.

To upload coverage results to *Teamscale*, you should use the external-report service as described in section 4.3.3.

4.3.4.7 ESLint

While ESLint is an internal analysis tool, the ESLint binaries are not shipped with Teamscale. Instead, you will need to install NodeJS[14] on

[12] http://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[13] http://gcovr.com/
[14] https://nodejs.org/

your machine. Once NodeJS is installed, you can run the following command to install the correct ESLint version:

```
npm i -g eslint@0.24.1
```

You will need to have both the NodeJS and ESLint binaries on the PATH and restart Teamscale before you can enable ESLint for your project.

4.3.4.8 TSLint

While TSLint is an internal analysis tool, the TSLint binaries are not shipped with Teamscale. Instead, you will need to install NodeJSChapter 14 on your machine. Once NodeJS is installed, you can run the following command to install the correct TSLint version:

```
npm i -g tslint@5.9.1 typescript@2.7.1
```

You will need to have both the NodeJS and TSLint binaries on the PATH and restart Teamscale before you can enable TSLint for your project.

4.3.4.9 PC-lint

PC-lint[15] is a static analysis tool for C/C++ that covers a wide range of checks. While the tool is delivered as a Windows executable, it can also be used with Linux using wine and a wrapper script[16]. PC-lint is fully integrated into *Teamscale* as an incremental analysis tool, but the tool has to be on the path or the explicit path to the tool must be configured in the analysis profile.

[15] http://www.gimpel.com/html/pcl.htm
[16] http://www.approxion.com/?p=135

Note for running *Teamscale* on Windows: If PC-lint crashes during the execution, the Windows crash dialog will open and block Teamscale until the dialog is closed manually. To prevent this behavior, log in as the user running *Teamscale* and set the following key in the registry to 1:

```
HKEY_CURRENT_USER\Software\ Microsoft\Windows\Windows
Error Reporting\DontShowUI
```

4.3.4.10 Pylint

Pylint[17] is an Open Source code analysis tool for Python working directly on the source code.

To import Pylint analysis results into *Teamscale*, you need to create the generated report using a special format. This can be achieved by using the following command line call:

```
pylint -r n
--msg-template='{{line: {line}, messageId: "{msg_id}",
message: "{msg}", absolutePath: "{abspath}",
messageName: "{symbol}"}}'
> report.json
```

This command generates a JSON file which can be uploaded to *Teamscale* with the *Teamscale* Python client's `upload_report` method.

4.3.4.11 StyleCop

StyleCop[18] is an Open Source style violation checker for C# working directly on the source code. StyleCop is full integrated into *Teamscale*

[17] http://www.pylint.org/
[18] http://stylecop.codeplex.com/

as an incremental analysis step, so to run StyleCop, you only have to enable it when configuring the analysis profile.

When running on Windows, the .NET framework v4.0 or higher must be installed. When running on Linux, Mono 3.0 or higher has to be installed to execute StyleCop.

4.3.4.12 Xcode Test Coverage

Teamscale supports importing line-based test coverage produced by Xcode and the xccov command line tool. As the Xcode tooling cannot produce a comprehensive report with line coverage information for all files in a project, the following shell script must be used to obtain a coverage report that can be imported to Teamscale.

```
#!/bin/bash
if [ $# -eq 0 ]
then
echo "Please provide path to xccovarchive directory"
exit 1
fi
coverage=""
archive=$1
for file in $(xcrun xccov view $archive --file-list); do
coverage+=$file
coverage+="\n"
coverage+="$(xcrun xccov view $archive --file $file)"
coverage+="\n"
done
echo "${coverage}"
```

This command will output the coverage report to standard out. You can pipe the output into a file which can then be uploaded via the REST API as described in Section 4.3.3.

```
./generate-coverage /path/to/xccovarchive > coverage.txt
```

4.3.5 Test Gap Analysis Integration

In order to use the *Test Gap analysis*, you'll need to upload coverage. For most coverage profilers, the resulting coverage files can simply be uploaded as described in Section 4.3.3. For some profilers, however, *Teamscale* needs mapping files to interpret the coverage files.

Teamscale allows you to upload these mapping files ahead of time, e. g. when they are created during the build process. Then, when uploading coverage files, it will use the mapping files to interpret the coverage information. Since every time you build your application, new mapping files are generated, *Teamscale* needs to know, which mapping files to use when you upload coverage. Thus, during the upload of the mapping files, you must provide a version string (any unique string will do, e. g. a revision number). You must again provide the exact same version string when uploading coverage files that belong to those mapping files.

4.3.5.1 TGA for JavaScript/TypeScript

To collect coverage in the browser, currently, only the Istanbul[19] profiler's JSON format is supported by *Teamscale*. Since JavaScript may be minified and TypeScript is always transpiled to JavaScript before execution in the browser, you need to upload the mapping files (com-

[19] https://istanbul.js.org/

monly called *source map files* in this context) generated during the build.

Please note that errors in the source maps can lead to subtle misalignment of the covered lines and thus to wrong coverage results. Please make absolutely sure that source maps are configured correctly for your entire build pipeline and that they correctly map back to your source files. Some minification/bundling tools are known to e.g. produce off-by-one source maps.

The following is an example of how to upload source map files using *curl*:

```
curl --user bob:e434mikm35d -F "file=@./source.map"
-F "version=VERSION" -F "t=TIMESTAMP"
localhost:8080/p/PROJECTID/javascript-source-map-upload
```

The `TIMESTAMP` must be the java timestamp of the commit which was processed by the build to generate the mapping file.

The following is an example of how to upload Istanbul coverage files using *curl*:

```
curl --user bob:e434mikm35d -F "report=@./coverage.json"
-F "message=my message"
-F "partition=myPartition"
localhost:8080/p/PROJECTID/istanbul-tga-coverage-upload
```

4.3.5.2 TGA for C#

There are many C# profilers for which no special mapping files are needed. We do, however, have a custom low-overhead profiler. It

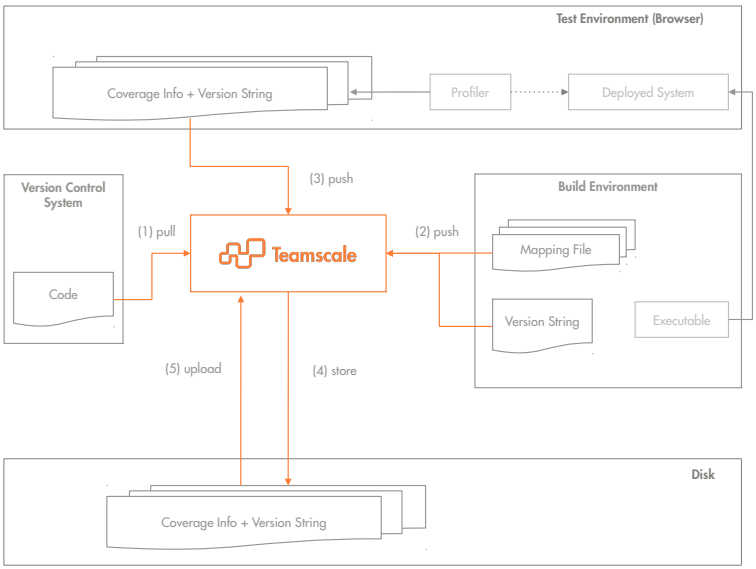


Figure 4.131. *Test Gap analysis* for JavaScript/TypeScript data flow

is specially suited for the *Test Gap analysis* as it only records which methods were executed, not which particular lines. This allows it to have virtually no impact on performance. We have successfully used it to profile in several production environments.

Since only minimal information is recorded at runtime, *Teamscale* needs the PDB files generated during the build to interpret the coverage files.

The following is an example of how to upload PDB files using *curl*:

```
curl --user bob:e434mikm35d -F "file=@./assembly.pdb"
-F "version=VERSION" -F "t=TIMESTAMP"
localhost:8080/p/PROJECTID/dotnet-debug-info-upload
```

The `TIMESTAMP` must be the java timestamp of the commit which was processed by the build to generate the PDB file.

The following is an example of how to upload CQSE profiler coverage files using *curl*:

```
curl --user bob:e434mikm35d -F "report=@./coverage.txt"
-F "version=VERSION" -F "message=my message"
-F "partition=myPartition"
localhost:8080/p/PROJECTID/dotnet-ephemeral-trace-upload
```

4.3.6 Test Impact Analysis Integration

In order to use the *Test Impact analysis*, you'll need to upload testwise coverage.

4.3.6.1 Testwise Coverage report format

Testwise Coverage is a combined JSON-based report format that holds various test related data (see listing 4.1 for an example). It's basis is a list of all tests that are available for execution as one unit (e. g. UI Tests). A test is identified by a unique string (uniformPath). The full list allows *Teamscale* to determine added and deleted tests. Each of the test entries can additionally contain execution information (duration, result, message), coverage information (paths) and additional test metadata (sourcePath, content). The following Figure 4.132 provides more details on each of the available fields.

Test metadata

uniformPath	Unique name of the test case by using a path like hierarchical description, which can be shown in the UI.
content	Identifies the content of the test specification. This can be used to indicate that the specification of a test has changed and therefore should be re-executed. The value can be an arbitrary string value e. g. a hash of the test specification or a revision number of the test.
duration	The time it took to execute the test in seconds.
result	Describes the result of the test execution. Can be one of: ■ PASSED Test execution was successful. ■ IGNORED The test is currently marked as "do not execute" (e. g. JUnit @Ignore or @Disabled). ■ SKIPPED Caused by a failing assumption. ■ FAILURE Caused by a failing assertion. ■ ERROR Caused by an error during test execution (e. g. exception thrown). Tests that did not pass previously are always selected for test execution.
message	Error message and stacktrace or skip reason.
paths	The covered lines grouped by source file and path.

Figure 4.132. Test metadata

Listing 4.1. A simple Testwise Coverage report.

```
{
  "tests": [
    {
      "uniformPath": "com/example/project/JUnit4Test/
          notExecutedTest()"
    },
    {
      "uniformPath": "com/example/project/JUnit4Test/
          systemTest()",
      "duration": 0.025,
      "result": "PASSED",
      "content": "43d743a8ef4389bc5",
      "paths": [
        {
          "path": "com/example/project",
          "files": [
            {
              "fileName": "Calculator.java",
              "coveredLines": "20-24,26,27,29"
            },
            {
              "fileName": "JUnit4Test.java",
              "coveredLines": "26-28"
            }
          ]
        }
      ]
    },
    {
      "uniformPath": "com/example/project/JUnit5Test/
          failedTest()",
      "duration": 0.045,
      "result": "ERROR",
      "message": "Some error"
    },
    {
      "uniformPath": "com/example/project/JUnit5Test/
          skippedTest()",
      "result": "SKIPPED"
```

```
36          }
37      ]
38  }
```

4.3.6.2 TIA for the JVM

For JVM-based languages there exists an extension to the JaCoCo coverage profiler, which can be instructed to generate Testwise Coverage[20].

For Gradle based builds with JUnit Platform based tests there also exists a Gradle plugin[21]. The plugin configures the test execution to use a custom Test Engine which handles the communication with *Teamscale* and delegates the actual execution to existing Test Engines.

4.3.7 Custom Check API

The »*Teamscale* Custom Check API« allows users to extend *Teamscale* by writing custom analyses that create findings. These *custom checks* are executed within *Teamscale* 's incremental analysis engine and, thus, provide real-time feedback to developers on every commit. Consequently, *Teamscale* treats the findings created by custom checks equal to all other findings, such that all finding-related features, like blacklisting, visualizations, filtering etc., can be used just as usual.

A custom check is a *local* analysis in *Teamscale*, i.e. its implementation is given a single source file as an input and the check can create findings for code locations within this file. A check can use the source

[20] https://github.com/cqse/teamscale-jacoco-agent/blob/master/TEST_IMPACT_ANALYSIS_DOC.md#getting-a-testwise-coverage-report

[21] https://github.com/cqse/teamscale-jacoco-agent/blob/master/teamscale-gradle-plugin/README.md

code of the file as plain text, the sequence of tokens or the abstract syntax tree (AST) of the file or a combination of these for its analysis. Each *Teamscale* distribution already contains many custom checks which we list in Appendix E.

4.3.7.1 Implementing a Custom Check

Setting up the Development Environment It is recommended to use the Eclipse development environment to implement Custom Checks. To set up a development workspace, please perform the following steps:

1. Download an up-to-date version of the Eclipse IDE and unzip it.
2. Launch Eclipse and create a new workspace.
3. Download the Custom Check Example project from the following URL, create an Eclipse project as described there and import it into your Eclipse workspace.
   https://github.com/cqse/teamscale-custom-check-sample

Writing a custom check To implement a custom check, create a new Java class deriving from `CheckImplementationBase` that is annotated with `@ACheck`. The class CheckImplementationBase contains the analysis context (ICheckContext) which you can use to get information about the analyzed source code file. This includes the textual representation of the code, the stream of tokens in the file (the output of the scanner/lexer), as well as the abstract syntax tree (AST, output of the parser). The only method your Custom Check needs to implement (overwrite) is named execute(). The baseclass offers you helper methods to select certain elements in the AST using XPath expressions and to create findings easily. However, in many cases your Custom

Check class does not need to use this rather general base class but a more specific subclass of CheckImplementationBase. E.g. one of the following:

- EntityCheckBase: for checks that select a set of entities via XPath and inspect each of these.
- EntityFindingCheckBase: for checks that select a set of entities via XPath and create a finding for each selected entity.
- EntityTokenCheckBase: for checks that select entities via XPath and process the tokens of each selected entity.

In the @ACheck-annotation you must specify meta-data about your check. This meta-data is used by *Teamscale* mainly for configuration of the analyses (quality profile) and the presentation of the findings your check will produce. The meta-data includes the following information:

- name: Name of the check
- description: A description of the check that explains developers why the findings of the check are a problem and how they should fix them.
- groupName: The analysis group the check should be contained in (e.g. General Checks)
- languages: The programming languages for which the check can be used.

Furthermore, a Custom Check must specify on which representation of the code it performs the analyses (using the parameters-argument of ACheck). E.g., the check may only access the AST if the option ECheckParameter. ABSTRACT_SYNTAX_TREE is specified or type in-

formation will only be present if type resolution is explicitly switched on by using ECheckParameter.TYPE_RESOLUTION. *Teamscale* needs this information to cluster the checks, so that representations that are not needed are not calculated at all.

The following code listing shows a simple Custom Check:

Listing 4.2. A simple check that selects abstract types with public constructors and emits a finding for every hit.

```
1  @ACheck(name = "Abstract types should not have
       constructors (CA1012)",
2      description = "Constructors on abstract types can
           be called only by derived types. Because public
            constructors create instances of a type, and
           you cannot create instances of an abstract type
           , an abstract type that has a public
           constructor is incorrectly designed.",
3      groupName = "Language misuse",
4      languages = { ELanguage.CS },
5      parameters = { ECheckParameter.
           ABSTRACT_SYNTAX_TREE })
6
7  public class
       AbstractTypesShouldNotHaveConstructorsCheck extends
        EntityFindingCheckBase {
8
9    /**
10    * Returns an XPath selection string that
11    * selects public abstract types with public
12    * constructors.
13    */
14   @Override
15   protected String getXPathSelectionString() {
16     return "//TYPE[modifiers('public', 'abstract')]/
           METHOD[subtype('constructor') and modifiers('
           public')]";
17   }
18
19   /**
20    * Returns the message for the created finding.
```

```
21      */
22     @Override
23     protected String getFindingMessage(ShallowEntity
           entity) {
24       return "An abstract type must not have a public
             constructor.";
25     }
26   }
```

Structure of the AST and selection via XPath The AST consists of so called shallow entities. These entities are the nodes of the AST (e.g. types or methods). They are called shallow because the AST is only shallow in the sense that it is constructed down to the statement level but does not have detailed information on expressions within statements. The enum EShallowEntityType documents the list of possible shallow entity types. These are also used by the XPath notation to make a selection on AST nodes. For example, the XPath expression //METHOD would select all shallow entities that denote a method. Shallow entities also have a subtype, which further specify the entity. For instance a method entity can have the subtype 'constructor' denoting the special case of a constructor. Subtypes may be specific for a programming language. For a list of possible values, refer to the class SubTypeNames. Selection functions can be used to further filter shallow entities. As an example, the XPath expression //METHOD[modifiers('public')] selects all public methods. For a list of possible selection functions, please refer to the package eu.cqse.check.framework.core.xpath.functions.

Testing Your Custom Check The *Teamscale* Custom Check Framework provides a Unit-Testing framework you should use to write tests for your checks. The easiest way to test your check is using data-driven tests. This way you do not have to write any testcode but only

provide test data in terms of code listings that contain the findings your check should identify and a file describing the expected results. To define a data-driven test you need to perform the following steps:

1. Create a new subfolder within the test-data folder using the class name of your custom check class as the folder name (e.g. MyCheck)

2. Place a test code file (e.g. MyTestClass.java) to test your check against in the newly created folder.

3. Create a second file in the same folder and use the same name followed by '.expected', e.g. MyTestClass.java.expected. This file will be used to compare the findings detected by your check with the expected findings. For every finding your check should emit, put a line into this file using the following format: Offsets <fromOffset>-<toOffset> (lines <fromLine>-<toLine>): <findingMessage> The offsets here are character offsets in the source code. Please specify the following information corresponding to your test-data file: fromOffset/toOffset: Offsets to the tokens the finding should be annotated. fromLine/toLine: The line number that should be marked with a finding of your check. findingMessage: The message that should be displayed for your finding.

4. To be able to execute the tests, you need to create a single test class deriving from CheckTestBase that will automatically create a test suite from the test data (this class is already available in the test-src folder of Custom Check Example project):

Listing 4.3. Test class for data-driven custom check tests

```
1 /** Main class for testing checks */
2 @RunWith(Parameterized.class)
3 public class CheckTest extends CheckTestBase {
4
```

```
5     /** Constructor. */
6     public CheckTest(File referenceFile, Map<String
          , CheckInfo> checkInfoBySimpleClassName) {
7       super(referenceFile,
            checkInfoBySimpleClassName);
8     }
9
10    /** Generate Test Parameters. */
11    @Parameters(name = "{0}")
12    public static Collection<Object[]>
          generateParameters() throws IOException {
13      return CheckTestBase.generateParameters(new
            CheckTest(null, null));
14    }
15  }
```

5. To run the tests, launch the test class as a JUnit-Test in Eclipse and inspect the results (information about the actual results of your check are written to the directory test-tmp).

Deploying your Custom Check Package your custom check into a JAR file using the JAR Export Wizard of Eclipse. You do not need to re-package any of the included libraries. The check classes are sufficient. Place the JAR in the custom-checks subfolder of your *Teamscale* installation and restart *Teamscale*. The checks then become available when configuring analysis configurations.

4.3.8 Python API

We maintain a Python API that encapsulates the REST API and simplifies writing simple integrations and automations. The Python API is hosted at https://github.com/cqse/teamscale-client-python and documentation is provided online at http://cqse.github.io/teamscale-client-python/.



5. Administration

Teamscale can be installed on-premise on your own servers. This chapter gives you an overview of all administrative tasks to install and run *Teamscale*, starting with the system requirements, licensing over the actual installation as well as user administration and permissioning.

Table 5.1. System Requirements for Analysis Server

Category	Requirement
Operating System	Linux or Windows with Visual Studio 2015 Redistributables[1]
Java	Java Runtime Environment (e.g. Oracle Server JRE or OpenJDK), Version 8 or later, 64-bit with Unlimited Strength Jurisdiction Policy[2]
RAM	4 GB minimum
CPU	2 GHz, 2 or more cores recommended
Disk Space	Installation: 250 MB, Data: at least 10 GB recommended, SSD recommended

5.1 System Requirements

This section describes the system requirements both for the server on which the analyses are executed as well as on the client side.

5.1.1 Analysis Server

Table 5.1 denotes the requirements for an analysis server.

Note: The disk space requirements for data highly depend on

- the amount of source code of the analyzed software
- the time range of the history being analyzed

[1] https://www.microsoft.com/en-us/download/details.aspx?id=48145

[2] This is the default for Oracle Java 8 Update 161 and higher and recent OpenJDK versions. See http://opensourceforgeeks.blogspot.com/2014/09/how-to-install-java-cryptography.html

Table 5.2. System Requirements for Clients

Category	Requirement
Web Browser	Internet Explorer version 11 or later, Firefox latest version, Chrome latest version
Eclipse	Eclipse SDK 4.4.2 (*Luna* SR2) or later running with Java 8 or later
Visual Studio	Windows Vista or later, .NET Framework 4.5 or later, Visual Studio Version 2010 or later
IntelliJ	2016.3 or later
NetBeans	8.1 or later

- the commit behavior (frequency and number of committed files) within the time range analyzed

As an example, a system of about 500 kLOC and a history of 1.5 years requires about 1 GB of disk space.

5.1.2 Clients

Table 5.2 denotes the requirements for the clients for which an IDE Integration (see Section 4.2) is possible.

5.2 Installation

This chapter gives a brief overview of the installation procedure. Before continuing unzip the archive into the preferred target directory. For the remainder of this document, `<Teamscale-Dir>` is the absolute path to this directory.

5.2.1 Configuring *Teamscale*

Basic configuration of *Teamscale* can be done by editing the file `config/teamscale.properties`. Table 5.3 shows the configuration options. If you change the configured database type to RocksDB on a Windows environment, please be aware that you also need to install the Visual C++ Redistributable for Visual Studio 2015. The redistributable is available under this URL: https://www.microsoft.com/en-us/download/details.aspx?id=48145.

5.2.2 Providing a License

Teamscale needs a valid license to run. *Teamscale* automatically searches several locations for a license file named `teamscale.license` (in this order):

1. The directory specified in the Java system property `teamscale.license.path` (if it is set)
2. The *config* directory in the *Teamscale* installation directory (given by TEAMSCALE_HOME environment variable; this variable is set automatically, if using the start script)
3. The current working directory
4. The home directory of the user running *Teamscale*

Before starting *Teamscale* for the first time, it has to be ensured that a valid license exists in one of these locations.

5.2.3 Configuring Encryption

Teamscale encrypts all tables in the storage system that might contain sensitive information, such as passwords to your SVN server. The encryption algorithm used is AES-256. By default, *Teamscale* uses a default key that is hard-coded. To further improve security, you can provide your own key. For this, write your key or passphrase into the file `teamscale.key` in the `config` directory. *Teamscale* then uses the content of this file for initializing the key that is used to protect your data. The file should be protected using the usual file system permissions.

When creating backups, you can select to use this *local* key instead of *Teamscale*'s default key for encrypting the backup. The benefit of this method is that nobody without your key file can read the encrypted parts of the backup. On the flip-side, you can not import this backup into an instance that does not know the secret key of the instance. To import into an instance that uses a different encryption key, you either have to use a backup that uses *Teamscale*'s key instead of the local key, or you must provide the key as *alternative* decryption key. For this, place the key into the *config* directory using any file name with the extension *.key*. When reading a backup, *Teamscale* will automatically find the correct key to use.

5.2.4 Starting *Teamscale*

As a prerequisite for starting *Teamscale* it has to be ensured that the Java executable is on the path. *Teamscale* can be started with *Teamscale* startup shell script by executing following command:

```
<Teamscale-Dir>/teamscale.sh
```

In a Windows environment please use the corresponding `.bat` file in the same directory.

By default, *Teamscale* uses the configuration file `<Teamscale-Dir>/config/teamscale.properties`. To provide a different configuration file, use the `-c` option of the start script:

```
<Teamscale-Dir>/teamscale.sh -c <path-to-teamscale-config.properties>
```

In the default configuration, *Teamscale* starts a web server on your machine, using port 8080. This web server will be also available from other machines in your network. If you do not want this, remove the comment before the line `server.bind-hostname=localhost` in the file `teamscale.properties`.

Check if *Teamscale* has started successfully by opening

```
http://localhost:8080
```

in a supported web browser. You can log in using the default user »admin« and the password »admin«. *Teamscale* writes a log-file named `teamscale.log` in the execution directory.

5.2.4.1 Logging Settings

If you need to configure the *Teamscale* logging settings, you can do so in the file *config/log4j2.yaml*. This is a Log4j 2 configuration file in the YAML format, which you can adjust according to the guidelines available at the official documentation page. If you need to override the file location for some reason, you can set the Java system property `log4j.configurationFile` to point to the desired path. Alternatively,

the environment variable `TS_LOGGING_CONF` may hold the content of a properties file as documented here: https://logging.apache.org/log4j/2.x/manual/configuration.html#Properties. Prior to the *Teamscale* release version 4.8, the given properties had to use the naming required by Log4j 1. Starting with *Teamscale* 4.8, the Log4j 2 property names are used instead. Please note that the new configuration format is not backwards-compatible.

5.2.4.2 JVM Memory Settings

By default, the *Teamscale* start script will launch a JVM with a maximum Java heap size of 2.500 MB. You can change this by setting the environment variable `TEAMSCALE_MEMORY` or adjusting the file *config/jvm.properties*.

5.2.5 Installing *Teamscale* As a Service

You might want to install *Teamscale* as a service to allow for more consistent management with other services on the same machine and to ensure that *Teamscale* is automatically started after a reboot. Additionally, this allows *Teamscale* to automatically restart in case of a crash.

Before installing *Teamscale* as a service, ensure that the configuration as described in the previous sections was completed and both Cassandra and *Teamscale* can be started manually. Then follow the steps in one of the following subsections depending on your operating system.

5.2.5.1 Windows

The required files are located in the folder `windows` in the *Teamscale* installation directory.

1. Edit the file `teamscale-service.xml`.
   a) Enter or edit the correct path to the *Teamscale* root directory, for example

   ```
   <env name="TEAMSCALE_HOME"
   value="c:\opt\teamscale" />
   ```

   b) Enter or edit the correct path to the Java installation directory, for example

   ```
   <env name="JAVA_HOME"
   value="c:\program files\java\jdk1.8.0_45" />
   ```

   c) Optionally uncomment and fill the section `<serviceaccount>` to run the services as a different user. Alternatively you can set this information after installing using the service console, which avoids storing the password in clear text. In both cases the user needs to have the *Logon as a service* right. This right can be granted in the *Local Security Policy* management console by navigating to *Local Policies > User Rights Assignment* and adding the user to *Logon as a service*.

2. Install the *Teamscale* service by running the following command as an administrator:

   ```
   teamscale-service.exe install
   ```

3. Check that the service is running. Directly after the installation, you might have to start the service manually or alternatively reboot the server.

Uninstall If you later have to uninstall the service run the following command:

```
teamscale-service.exe uninstall
```

5.2.5.2 Linux

The required files are located in the folder `linux` in the *Teamscale* installation directory. You can choose between a `systemd` based installation or an `init.d` based one.

Installing the systemd service To see if your system supports systemd, check for the existance of the directory `/lib/systemd`.

1. Modify the `teamscale.service` file according to the TODO comments inside.
2. Copy the file to `/etc/systemd/system`
3. Enable the service to start at boot time by running
   ```
   sudo systemctl enable teamscale.service
   ```

To start the service manually use

```
sudo systemctl start teamscale.service
```

This procedure was tested on Ubuntu 17.04.

Uninstalling the systemd service

1. Run

```
sudo systemctl stop teamscale.service
sudo systemctl disable teamscale.service
```

2. remove the file you copied to `/etc/systemd/system/teamscale.service`

Installing the init.d service The installation is based on `init.d`, which is supported on most Linux distributions (this guide was tested with *Ubuntu 12.04*).

1. Edit the file `teamscale.init`.
   a) Enter or edit the correct path to the *Teamscale* root directory, for example

   ```
   TEAMSCALE_HOME=/opt/teamscale
   ```

   b) Enter or edit the correct path to the Java installation directory, for example

   ```
   export JAVA_HOME=/opt/jdk
   ```

   c) Optionally uncomment and fill the `EXTRA_START_ARGS` line to run the services as a different user.
2. Install the service by running from the command line:

   ```
   sudo ./install-service.sh teamscale
   ```

3. Start the service by running the following command:

   ```
   sudo service teamscale start
   ```

Uninstalling the init.d service If you later have to uninstall the service run the following command:

```
sudo ./uninstall-service.sh teamscale
```

5.2.6 Configuring HTTPS

Teamscale can optionally provide HTTPS communication, either in addition to HTTP or exclusively. The enablement of both can be controlled via the settings `server.port` and `https.port` in the server configuration (see also Section 5.2.1).

To set up HTTPS communication for *Teamscale*, a pair of private key and certificate is required. Your company may already have a certificate and key available or a new pair has to be generated. Please consult your IT operations team for potential regulations in your company. Technically, you also have the option of generating a self-signed certificate (not recommended for security reasons).

Teamscale requires the private key and certificate to be stored in the *Java Keystore* format. Prior to importing a certificate in a Java keystore, the certificate has to be converted to the PKCS 12 format. In case you are in a Microsoft Windows environment and have the certificate stored in the Certificate Management of the operating system, you can directly export the certificate and private key in the PKCS 12 format using the Certificate Export Wizard. You can then skip the conversion in the next paragraph and continue with the creation of the Java keystore.

In case the certificate and key was created with OpenSSL, the conversion can be done using the `openssl` command line tool[1]. Assuming your certificate is available in a file `myhost.crt` and your private key in a file `myhost.key`, the following command will combine them and

[1] Please refer to the OpenSSL website (https://www.openssl.org) for download and installation instructions.

save them a file `myhost.p12` converting them to the PKCS12 format which is compatible with the Java keystore (you will be asked for an export password):

```
openssl pkcs12 -export -in myhost.crt -inkey myhost.key
-out myhost.p12
```

After this, you can create a new Java keystore and import the certificate/key pair into the newly created store. This can be done with the `keytool` command line tool that is part of Java (located in the `bin` folder of the Java installation). The following command will create a new file `myhost.jks` containing a Java keystore with both the certificate and private key. You will be asked for import and export passwords. Please ensure to use the same password used previously for protecting the private key also for the keystore.

```
keytool -importkeystore -srckeystore myhost.p12
-srcstoretype pkcs12 -destkeystore myhost.jks
```

To enable HTTPS for *Teamscale*, all configuration settings in `teamscale.properties` starting with *https.* have to be properly configured (see also Section 5.2.1). Make sure to properly configure the path to the newly generated Java keystore, its password as well as the certificate alias. If you do not know the alias of your certificate, you can look it up with a keytool command:

```
keytool -list -keystore myhost.jks
```

Under Windows, please use forward slashes instead of backslashes for paths configured in `teamscale.properties`. Backslash is interpreted as an escape character.

If everything was properly configured, *Teamscale* will accept HTTPS connections on the HTTPS port specified in the *Teamscale* settings (`https.port`). All connections to the configured HTTP port (i. e. the value for `server.port` or the default of 8080) will be forwarded to the HTTPS port. If the HTTP port is set to 0, HTTP is disabled and only HTTPS connections are accepted.

5.2.7 Integrating With Apache Webserver

To fulfill certain requirements, such as different authentication mechanisms or HTTPS support, *Teamscale* can be operated behind a webserver, such as Apache. One possible setup has *Teamscale* running on the same machine as the Apache server. For this, the `teamscale.properties` should contain the following two lines.

```
server.urlprefix=teamscale
server.bind-hostname=localhost
```

This sets an explicit URL prefix to use and ensures that the instance is reachable only from the local machine (otherwise Apache could be circumvented). The Apache configuration then contains the following lines:

```
ProxyPass         /teamscale
  http://localhost:8080/teamscale
ProxyPassReverse  /teamscale
  http://localhost:8080/teamscale
ProxyRequests     Off

# This is important, as otherwise some URLs are blocked
# by Apache
AllowEncodedSlashes On
```

Depending on your configuration and goals, slightly different and additional directives are required. Please refer to the Apache manual for more information.

5.2.8 Integrating With nginx

To fulfill certain requirements, such as different authentication mechanisms or HTTPS support, *Teamscale* can be operated behind a nginx reverse proxy.

```
events {
    worker_connections 1024;
}

http {
  server {
    server_name teamscale\_nginx;

    location /teamscale {
      proxy_pass http://localhost:8080;
    }
  }
}
```

When *Teamscale* is served with a prefix like in the above configuration `/teamscale` *Teamscale* must be adjusted accordingly. For this, the `teamscale.properties` should contain the following two lines.

```
server.urlprefix=teamscale
server.bind-hostname=localhost
```

This sets an explicit URL prefix to use and ensures that the instance is reachable only from the local machine.

Depending on your configuration and goals, slightly different and additional directives are required. Please refer to the nginx documentation for more information.

5.2.9 Configuring HTTP(S) Proxy

If access to source code repositories, issue trackers etc. has to go through an HTTP proxy, the proxy host and port must be configured via system properties. You can achieve this by adding the following to the JVM_EXTRA_ARGS entry in *config/jvm.properties*:

```
-Dhttp.proxyHost=<Proxy-Host>
-Dhttp.proxyPort=<Proxy-Port>
```

If the proxy should also be used for HTTPS connections, use the following arguments instead:

```
-Dhttps.proxyHost=<Proxy-Host>
-Dhttps.proxyPort=<Proxy-Port>
```

Furthermore, for the HTTPS proxy, the proxy's SSL signing certificate needs to be added to *Teamscale*'s certificate store. Follow the procedure in 5.2.11 for that.

5.2.10 Using Certificates from the Windows Certificate store

On Windows, you can try to use the certificates from the build-in certificate store by adding the following options to the entry JVM_EXTRA_ARGS in the file *config/jvm.properties*:

```
-Djavax.net.ssl.trustStoreType=Windows-ROOT -Djavax.net.ssl.
trustStore=NONE
```

5.2.11 Connecting To External Servers Via HTTPS

If access to source code repositories, issue trackers etc. is through SSL-secured connections and the certificates for those connections are not signed by one of the certificate authorities known to a default Java installation, *Teamscale* needs to be made aware of these certificates. Start by downloading the certificate in question, e.g. through your web browser, and store it in DER format (`.cer` file extension).

Next, create a keystore that contains that certificate using the `keytool` command line tool (located in the `bin` folder of the Java installation) as follows, where `certificate.cer` is the certificate you downloaded, `keystore.jks` is the keystore to be created and `Alias` is the alias under which to store the certificate in the keystore:

```
keytool -importcert -file certificate.cer -keystore
keystore.jks -alias Alias
```

You will be prompted for a password for the keystore. Multiple certificates can be imported into the same keystore under different aliases.

Edit the file *config/jvm.properties* and add the following options to the entry JVM_EXTRA_ARGS:

```
-Djavax.net.ssl.trustStore=<path-to-keystore-file>
-Djavax.net.ssl.trustStorePassword=<password>
```

In case you also depend on the default certificates present on the system, you can import these into your keystore from the default instal-

lation (usually `$JAVA_HOME/jre/lib/security/cacerts`) using the `-importkeystore` command.

In case the self-signed certificates are not valid (e.g. the hostname is incorrect), Java will still reject the certificates and refuse to connect to these servers. To fix this problem, you can either install a correct certificate on the server or instruct *Teamscale* to not validate any SSL certificates.

Disabling SSL validation is strongly discouraged. It is not a secure practice.

To disable validation of SSL certificates, add the following to the entry JVM_EXTRA_ARGS in the file *config/jvm.properties*:

```
-Dcom.teamscale.disable-ssl-certificate-validation=true
```

5.3 Permission Management

In the following, the user management and access right management in *Teamscale* is explained.

5.3.1 Users, Groups & Roles

Access right management in *Teamscale* is based on *users*, *groups* and *roles* assigned to them. Each developer using *Teamscale* needs to have a user account. Users can be tied to multiple groups. In order to grant permissions to a user or group members, *roles* are assigned. A role represents a set of permissions granted to the user or group members and may have an optional context in which it applies. One such context would be a specific project within which the permissions of a

role apply. There are three different kinds of roles. Firstly, there are *global roles* which grant access to administrative features of *Teamscale* and creation of new objects like projects or analysis profiles. Table 5.5 lists the pre-configured default global roles of *Teamscale*. Secondly, there are *project roles* which a user or group members may have within a project. They grant access to different actions available within the project. See Table 5.6 for the pre-configured project roles of *Teamscale*. Lastly, there are *basic roles* which allow simple view, edit and delete access to access controlled objects like analysis profiles or external credentials. In the following the three different role types and corresponding permissions are explained in detail.

5.3.1.1 Global Roles

Global roles represent a set of permissions which enable features of *Teamscale* which are unrelated to a specific object like a project or analysis profile. They can be created, edited and deleted by a user with the global permission to edit roles. See Table 5.7 and Table 5.8 for global permissions and their meaning. A global role can be assigned to a user or group and grants the global permissions of the role. The role *Instance Admin* cannot be edited and always includes all global permissions.

5.3.1.2 Project Roles

Project roles represent a customizable set of permissions which are specific to projects. They can be created, edited and deleted by a user with the global permission to edit roles. See Table 5.9 project-specific permissions and their meaning. A project role can be assigned to a user or a group for a single project and grants the permissions of the role within the project. Furthermore, a project role can be assigned globally for all projects. Such a global assignment will grant the per-

missions to the user or group for all projects that exist in *Teamscale*. The role *Project Administrator* can't be edited and always includes all project permissions.

5.3.1.3 Basic Roles

Basic roles can be assigned to users and groups for objects in the following categories:

- Analysis Profiles
- Metric Threshold Configurations
- External Credentials
- Groups
- Users

The rest of this section will use the analysis profiles category as an example. However, the same also applies to objects in the other categories. Each basic role grants a fixed set of permissions to the user or group members. See Table 5.10 for more details about each basic role. A basic role can be assigned for a single analysis profile to a user or a group and grants the permissions of the role for the analysis profile. Furthermore, a basic role can be assigned globally for all analysis profiles. Such a global assignment will grant the permissions of the role to the user or group for all analysis profiles that exist in *Teamscale*.

5.3.2 Managing Access to a Project

In *Teamscale* access to a project is controlled by assigning a project role to a user or group. See section Section 5.3.1.2 for more information on project roles. The role is assigned for a specific project and grants the permissions of the role to the user or groups members.

5.3.2.1 Role Assignment Page

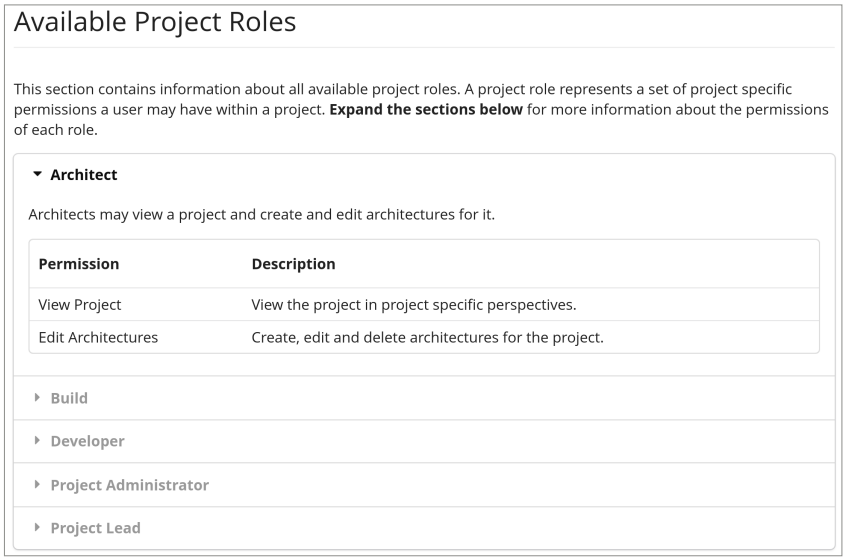
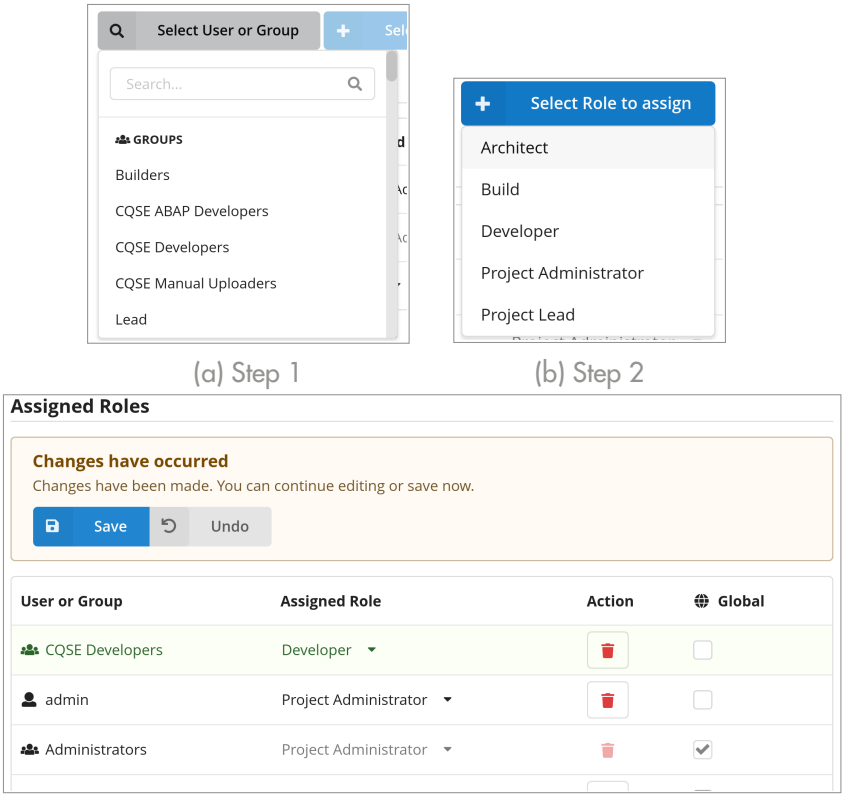


Figure 5.1. Available Project Roles Section

Roles can be assigned to users or groups on the role assignment page of the project. First, navigate to the **Projects** perspective. Here, we assume that you already created a project. In the list of projects click the edit role assignments button for the project you would like to assign roles for. The top section of the role assignment page shows the currently configured project roles as depicted in Figure 5.1. Click

on the role to see detailed information about the permissions granted by each role. Below the available roles the section for assigning roles to users or groups for the current project follows.



(a) Step 1 (b) Step 2

(c) Step 3

Figure 5.2. Assigning a Project Role

Example: Granting Project Access to Developers Assume that a group with all users you want to have developer access for a project already exists. In our example we use the role *Developer*, which is

included in *Teamscale* by default. It just contains the project permission *View Project* which allows users or group members to view the project in project-specific perspectives like the **Activity** perspective. First, navigate to the role assignment page for the project as described in Section 5.3.2.1. To assign the role to your group click the button Select User or Group. Select a user or group you would like to assign a role for from the dropdown menu from Figure 5.2a. In our example we use the group *CQSE Developers* which contains all developers of the project. Next, we choose the project role by clicking using the button + Select Role to assign. We pick the role *Developer* from the dropdown menu shown in Figure 5.2b. After selection the UI shows that changes to role assignments have occurred as depicted in Figure 5.2c. To apply the changes we press the Save button. After saving the changes all users in the group *CQSE Developers* now have the permissions of the *Developer* role for the project.

Example: Granting a Build User with limited Access to all Projects

This section describes how to grant simple build access to all projects configured in Teamscale to a build user. We assume you already created a build user to which you would like to grant access. First, navigate to the role assignment page of any project you created as described in Section 5.3.2.1. For providing build level access to the project we use the *Build* role included in the pre-configured project roles of Teamscale. It allows external uploads and trigger of commits hooks for a project. Add the role *Build* for the build user as described in the previous example. Before saving the changes check the checkbox for the *Global* column in the row of the added role assignment. Then press the Save button. This global role assignment now applies to all projects stored in Teamscale. Note that the build user can't view any project via the Web UI. Hence access to code visible in the Teamscale UI is not shown if you login as the build user.

5.3.3 Managing Access to an Analysis Profile

As outlined in Section 5.3.1.3, access to objects in the categories analysis profiles, metric threshold configurations, external credentials, groups and users is controlled by assigning *basic roles* for the object. The process for assigning these roles is analogous to managing access to projects from Section 5.3.2. In the following access management for analysis profiles is presented. However managing access to the objects in the other categories works in the same fashion by performing the same steps from the respective pages for listing the objects in on of the categories.

5.3.3.1 Role Assignment Page

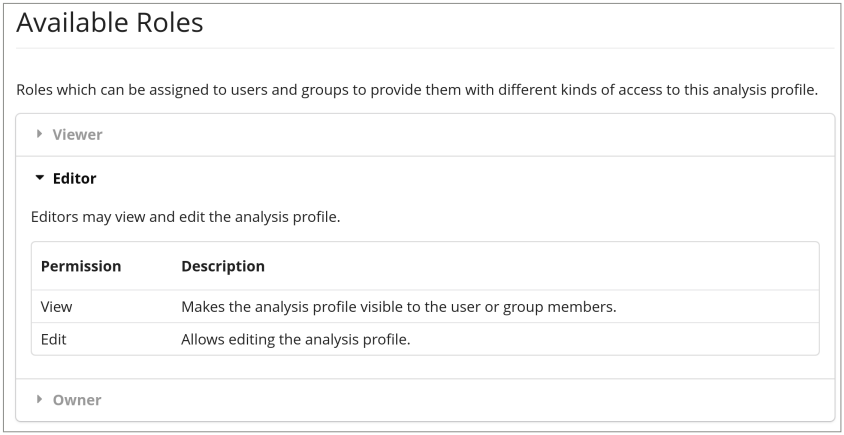
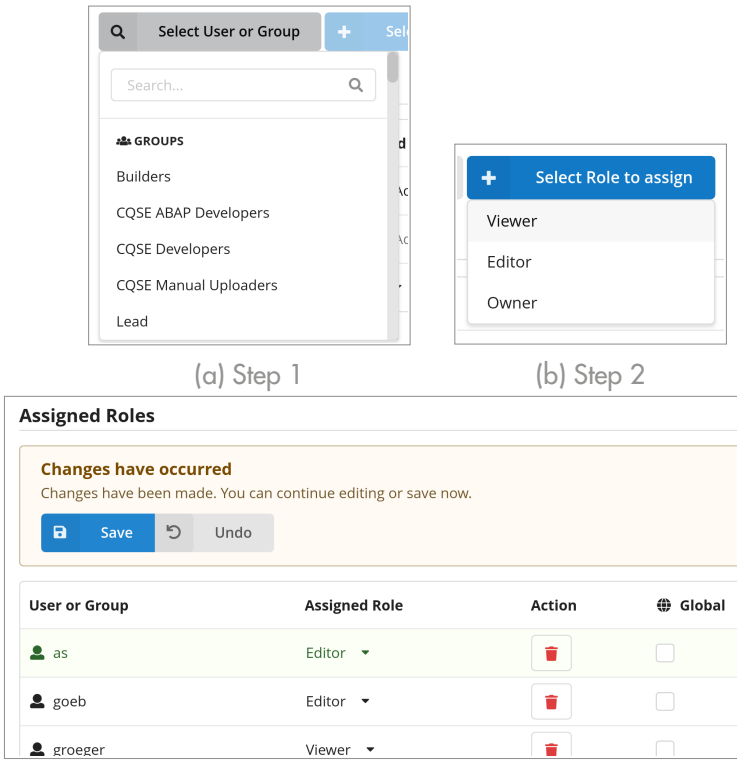


Figure 5.3. Available Project Roles Section

Roles can be assigned to users or groups on the role assignment page of the analysis profile. First, navigate to the **Analysis Profiles** page of the **Projects** perspective. Here, we assume that you already created an analysis profile for which you want to control access for. In the list of analysis profiles click the edit role assignments button for

the analysis profile you would like to assign roles for. The top section of the role assignment page for the analysis shows the basic roles as depicted in Figure 5.3. Click on the role to see detailed information about the permissions granted by each role. Below the available roles the section for assigning roles to users or groups for the current analysis profile follows on the role assignment page.



(a) Step 1 (b) Step 2

(c) Step 3

Figure 5.4. Assigning a Basic Role for Analysis Profiles

Example: Allowing a User to Edit an Analysis Profile This section describes how to allow another user to edit an analysis profile. Assume that a user to which you would like to grant access for an analysis profile already exists. First, navigate to the role assignment page for the analysis profile as described in Section 5.3.3.1. In our example, we want to allow a single user to view and edit an analysis profile. For that purpose we assign the basic role *Editor* to the user for the analysis profile. To assign the role to your group click the button Select User or Group. Select a user or group you would like to assign a role for from the dropdown menu from Figure 5.4a. In our example we select the user *Andi Scharfstein*. Next, we choose the role by clicking using the button + Select Role to assign. We pick the role *Editor* from the dropdown menu shown in Figure 5.4b. After selection the UI shows that changes to role assignments have occurred as depicted in Figure 5.4c. To apply the changes we press the Save button. After saving the changes the user *Andi Scharfstein* has the permissions of the *Editor* role for the analysis profile.

5.3.4 Managing Access to Global *Teamscale* Features

Teamscale has several global features to which access can be controlled. Examples include the creation of projects or editing the global e-mail notification settings. To grant access to these features global roles are assigned to users or groups. Refer to Section 5.3.1.1 for more information on global roles and global permissions which grant access to global *Teamscale* features.

5.3.4.1 Role Assignment Page

Roles can be assigned to users or groups on the role assignment of the user or group respectively. First, navigate to the Admin perspective

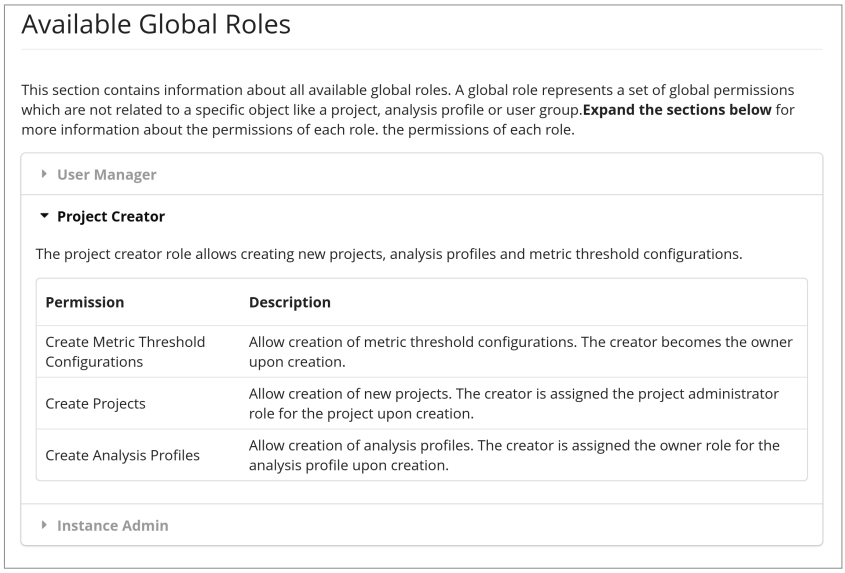
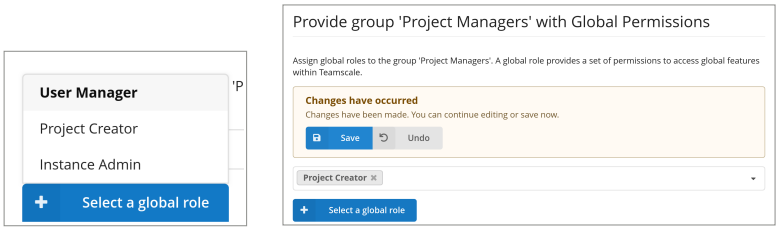


Figure 5.5. Available Global Roles Section

and open either the **Users** or **Groups** page. Here, we assume that you already created a user or group to which you would like to assign a global role. In the list of users or groups click the edit role assignments button for the user or group to which you want to assign a global role. The sections for assigning global roles are at the bottom of the page. Section *Available Global Roles* of the page shows the currently configured global roles as depicted in Figure 5.5. Click on the role to see detailed information about the permissions granted by each role. Global roles are assigned in the section below *Available Global Roles* on the role assignment page.

Example: Allowing Users to Create a Project This section describes how to allow a group of users to create projects, analysis profiles



(a) Step 1 (b) Step 2

Figure 5.6. Assigning a Global Role

and metric threshold configurations. For this example a group called *Project Managers* was created. It contains all users which should be able to create projects in *Teamscale*. First, navigate to the role assignment page for the group *Project Managers* as outlined in Section 5.3.4.1. Next, click the button + Select a global role. From the dropdown menu shown in Figure 5.6a, choose the role *Project Creator*. The *Project Creator* role is one of the default global roles included in *Teamscale* and grants the global permissions to create projects, analysis profiles and metric threshold configurations. After selecting the role the UI informs you that changes to role assignments have occurred as depicted in Figure 5.6b. To apply the role assignment press the Save button. After saving the assignment the members of group *Project Managers* have the permissions of the *Project Creator* role.

5.3.5 Managing Roles

Teamscale allows creating new roles with the set of permissions you would like to grant to users within projects or for administrative duties. Therefore new project- and global roles can be created with a custom set of permissions. Furthermore, existing roles can be edited

to change the permissions for all assignments of that role or deleted if they are no longer needed. To edit roles the global permission *Edit Roles* is required. Note that this is a powerful permission which should only be granted to *Teamscale* administrators.

5.3.5.1 Role Configuration Page

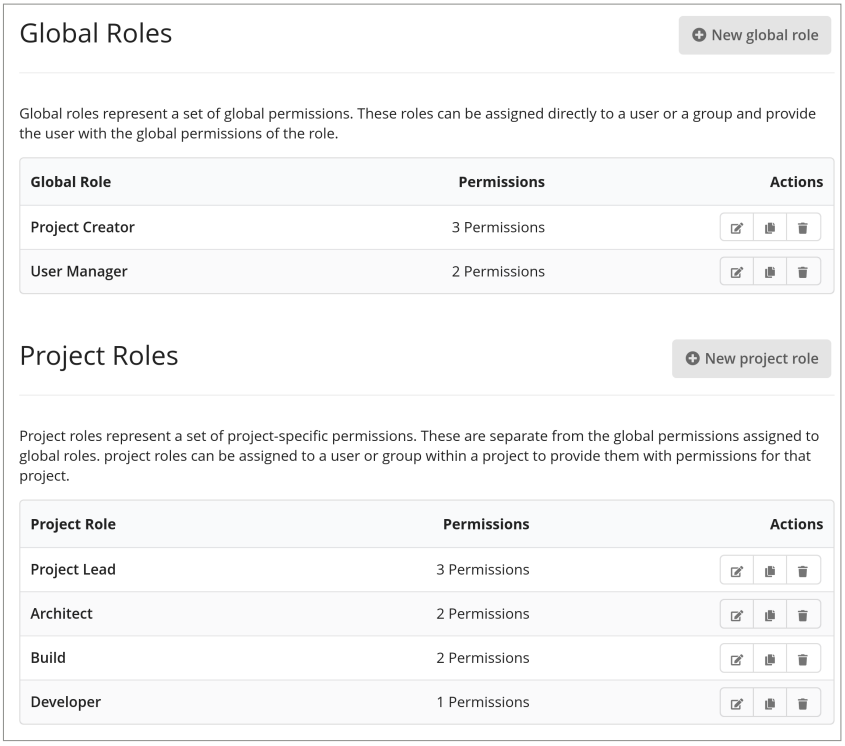
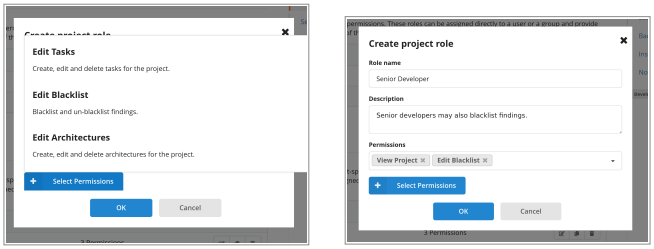


Figure 5.7. Roles Page

Project- and global roles can be created, edited and deleted on the **Roles** page of the **Admin** perspective. There exits a section for global

roles and projects roles as shown in Figure 5.7. Existing roles can be edited using the button and deleted using the button.



(a) Step 1 (b) Step 2

Figure 5.8. Creating a Project Role

Example: Creating a Project Role This section describes the creation of a new project role. First, navigate to the role configuration page as described in Section 5.3.5.1. Next, click the + New project role button which will show a dialog for creating a new role. In this example the role *Senior Developer* is created which should allow blacklisting findings in addition to viewing the project. Click the button + Select Permissions to add the permissions. From the dropdown menu pick the *View Project* permission. Repeat the last previous two actions for the *Edit Blacklist* permission as shown in Figure 5.8a. After adding the roles fill the role name field with *Senior Developer* and optionally add a short description. The result of these steps is shown in Figure 5.8b. Lastly, click the OK button to create the role. The role is now shown in the table for project roles and can be assigned to users or groups as outlined in Section 5.3.2.1.

5.3.6 LDAP Server Integration

Teamscale provides LDAP integration to synchronize your LDAP users and groups with the teamscale server. *Teamscale* can import groups or individual users from a configured LDAP server.

To configure LDAP go to the **Admin** perspective and open the **Settings** page. Press *Add* under the **LDAP Server** section and enter a name for the server. Fill in the settings for the LDAP server. The different options are described in Table 5.11.

Add the server by pressing the *Add* button. You can now import entire groups of users using the *Import* button in the *Groups* view. If you wish to import single users, use the *Import* button in the *Users* view. If you want to update user or group information use the synchronize buttons found in both views.

Typical Settings for Active Directory When configuring an *Active Directory* server, typically, the values as shown in Table 5.12 can be used for the general settings.

Typical Settings for Open LDAP The following Table 5.13 shows typical settings for configuring an Open LDAP server.

5.3.6.1 Global Catalog

If users are stored in different domains and are thus not available from a single LDAP server it is recommended to query the global catalog of one of the directory servers using port 3268 (or 3269 with SSL). The global catalog offers read-only access to most data of the whole domain tree and thus reduces the overhead to follow LDAP redirects

to other servers. Not using the global catalog and following referrals will create a new connection for each LDAP request which may lead to *refused connection* errors.

5.3.6.2 Fallback Servers

If the LDAP server has fallback servers which are listed in the DNS record, it is recommended to configure the server host using the domain name instead of a single IP address. *Teamscale* will then use the fallback servers if the primary server is not available.

5.3.7 Single Sign-On using SAML 2.0

This option can be used, if you already have a SAML 2.0 compatible Identity Provider. Note that SAML can only be used for authentication, which means that your users need to be known to Teamscale beforehand. This is typically achieved by connecting Teamscale to your LDAP server and importing corresponding users and groups.

As first step, you must configure your identity provider to allow requests from Teamscale. For this, you need to configure at least the entity id of the Teamscale service provider, which you can choose freely, and the assertion consumer URL. The later is the service *auth-saml*, so if your Teamscale server is configured to serve https://ts.company.com/, then the assertion consumer URL would be https://ts.company.com/auth-saml. The configuration details of your identity provider depend on the product used and are beyond the scope of this guide. As a result you should obtain a metadata XML description of the identity provider as SAML 2.0 entity description.

On the Teamscale side, in the **Admin** perspective navigate to **Settings**. First, ensure that your *Teamscale instance base URL* is set cor-

rectly. Next, create a new *SAML 2.0 Identity Provider* with a meaningful name and fill the fields (for details see Table 5.14). Most of the required information is parsed from the metadata you provide there. For testing, you should log out. On the login screen you should see a button that allows you to sign in using the configured identity provider.

5.3.8 Automatic URL Log-In

Normally, a human user logs on to *Teamscale* via the log-in page by entering username and password. However, there are cases where the log-in should be performed *automatically* when opening the first *Teamscale* page. This can be for instance needed for integrating *Teamscale* dashboards in other websites or showing them on public screens. This is supported by the automatic log-in feature via URL. You can provide the username and password as well as the target page to open within the URL. As an example, the following URL will perform a log-in for the user *john* and redirect to the **Dashboard** perspective.

```
http://teamscale.acme.com/login.html?autologin=john:
PASSWORD&target=dashboard.html
```

5.4 Settings

The **Admin** perspective provides a view for configuring various settings in *Teamscale*. The most important ones are described here.

5.4.1 Teamscale instance base URL

This allows to set the URL *Teamscale* is running in. The value here is important for providing links to *Teamscale* in notification emails and exported files.

5.4.2 Teamscale Working Directory

Allows to configure the full path to the Teamscale working directory where Teamscale stores data such as local Git clones and other cached data.

5.4.3 Server Limits

This section describes the configurations to help protecting the server from being overloaded by too many requests.

5.4.3.1 Pre-Commit Limits

Pre-commit analysis is triggered from the IDEs by developers. If a high number of developers is using this feature, a considerable number of requests is sent to the *Teamscale* server. The maximum number of files and the maximum individual file size per pre-commit as well as the minimum waiting time between pre-commit requests can be configured.

5.4.3.2 Late External Upload Processing

When external analysis data (e. g. coverage) is uploaded to *Teamscale*, a rollback is performed to integrate the external data with code from the repository. Especially for old uploads that require a roll-

back of several days, the following reanalysis can take a considerable amount of time. As a solution to this problem, *Teamscale* can postpone these uploads to a time where such a reanalysis does not disrupt normal operation, e. g. during the night.

The setting requires a crontab expression that defines at which time rollbacks and reanalysis from those late uploads may be performed, and a threshold that defines how many minutes in the past an upload must be to be considered *old*. The setting is active if a non-empty crontab expression is provided.

5.4.4 Git

Teamscale can communicate with Git repositories using a variety of protocols, e.g., HTTPS or SSH. This section allows to configure such connections.

5.4.4.1 Git Private Key

When communicating with a GIT repository over the SECURE SHELL (SSH) protocol, the *Teamscale* server may need to authenticate itself with a private key. Such keys can be managed here. Each key conists of a unique ID and the actual key material. The key can then be referred to in the GIT repository connector's configuration (Figure 3.9, Section B). The key material is a Base64-encoded string enclosed by a header/footer pair like `BEGIN/END RSA PRIVATE KEY`.

While *Teamscale* supports a wide range of SSH private key formats, it does not yet support the "new" OpenSSH private key format, which is enclosed by a header/footer pair of `BEGIN/END OPENSSH PRIVATE KEY`. If your private key is stored in this format, you can convert it to a format that *Teamscale* can handle like this:

OpenSSH: `ssh-keygen -e -m PEM -f $INPUT > $CONVERTED`
PuTTY: `puttygen $INPUT -O private-openssh -o $CONVERTED`

5.4.5 GitHub Integration

Teamscale integrates with GitHub as a *GitHub App*. This applies to both the official github.com site and custom installations of *GitHub Enterprise*. The minimum required GitHub Enterprise version for the integration to work is 2.14.

Prerequisites for the configuration are a Teamscale license with GitHub support enabled, a Teamscale installation that can be reached from the GitHub instance you want to connect to[2], and a GitHub organization used to register the GitHub App.

1. Ensure that the public base URL is configured correctly in Teamscale. For this, open the settings page (**Admin**/ **Settings**) and check the entry *Teamscale instance base URL* in tab **Server Settings**. This should point to the URL of the server that is reachable by the GitHub installation.

2. In GitHub, navigate to the settings page of your organization. Open *Developer Settings*/*GitHub Apps* and click the button *New GitHub App*. Here you can provide a name and details of your app. The following fields are relevant for Teamscale:
   User authorization callback URL: This is the public URL of your Teamscale installation followed by *github-oauth*, so for the Teamscale URL https://teamscale.acme.com this would be https://teamscale.acme.com/github-oauth.

[2]When evaluating Teamscale on a local laptop or during development, a service for exposing local ports on the internet can be helpful. One such service is *ngrok* (https://ngrok.com/).

Setup URL: Should be left empty.

Webhook URL: This is the public URL of your Teamscale installation followed by *github-hook*, so for the Teamscale URL https://teamscale.acme.com this would be https://teamscale.acme.com/github-hook.

Webhook secret: While this is optional, it is a good idea to place a random string in here. Remember this string for later usage in Teamscale.

Permissions: Teamscale requires these permissions:

- Repository metadata: Read-only (the default)
- Checks: Read & write
- Commit statuses: Read & write
- Issues: Read-only
- Pull requests: Read & write
- Organization members: Read-only
- Repository contents: Read-only

Subscribe to events: The following events should be checked:

- Issues
- Pull request
- Push

3. Still in GitHub, you now have to generate a private key for your application. This can be done in the section *Private keys* in the general settings of your GitHub App. Download and save the key for later.

4. Back in Teamscale, go to **GitHub Integration** on the settings page and fill the fields as follows:

GitHub URL: Enter the URL of your GitHub installation; for the public GitHub instance use https://github.com/.

ID of the GitHub app: This value can be found at the top of the settings page of your GitHub App (*App ID*).

Application private key: Enter the private key you just downloaded in the previous step.

Secret used for securing webhook calls: This is the webhook secret you configured in step 2. If you did not use a webhook secret, leave this empty.

OAuth client id : This value can be found at the top of the settings page of your GitHub App (*Client ID*).

OAuth client secret : This value can be found at the top of the settings page of your GitHub App (*Client secret*).

Use GitHub for Single Sign On: Check this box if you want to allow users to log into Teamscale using their GitHub account.

Create a new user on first login: Check this box if you want users that log into Teamscale using GitHub but do not have a Teamscale account yet to be created automatically.

5. Go back to the settings page of your GitHub App and switch to the *Advanced* tab. Check the *Recent Deliveries*. There should be a successful *ping* attempt in the list.

To install this GitHub App for your repository, navigate to the settings page of the GitHub App and select *Install App*. This allows *Teamscale* to access you repositories and vote pull requests. Note that *Teamscale* projects are not automatically created, but rather have to be created as needed using the GitHub connector.

Teamscale ensures that only users that have access to a repository on GitHub can create or modify a GitHub connector for that repository in *Teamscale*. This ensures that only authorized users can import code from a repository into *Teamscale*.

To that end, whenever you modify or create a GitHub connector in *Teamscale*, *Teamscale* checks your *Teamscale* user name against the list of team members and collaborators that have read access to the corresponding repository on GitHub. If your *Teamscale* user name is not in that list, you will not be able to create/modify the *Teamscale* project.

5.4.6 Bitbucket Cloud Integration

Teamscale integrates with Bitbucket Cloud as a *Bitbucket App*. Prerequisite for the configuration is a Teamscale license with Bitbucket support enabled, a Teamscale installation that can be reached from the Bitbucket Cloud instance you want to connect to[2], and a Bitbucket team used to register the Bitbucket App. Please note that this app needs to be installed on the team account, so that Teamscale could have access to all available repositories of your team.

1. Ensure that the public base URL is configured correctly in Teamscale. For this, open the settings page (**Admin**/ **Settings**) and check the entry *Teamscale instance base URL* in tab **Server Settings**. This should point to the URL of the server that is reachable by the Bitbucket installation.

2. In Bitbucket, navigate to the settings page of your team. Open *Installed apps* and click the link *Install app from URL*. Here you can provide the App URL to directly install the app.
   App URL: This is the public URL of your Teamscale installation followed by *bitbucket-install*, so for the Teamscale URL https://teamscale.acme.com this would be https://teamscale.acme.com/bitbucket-install.

3. After clicking *Install*, the app will request access to Bitbucket. Click *Grant access*. You will get a message that the app has

been successfully installed and will appear in the list of installed apps.

Note that *Teamscale* projects are not automatically created, but rather have to be created as needed using the Bitbucket connector after the app has been installed.

5.4.7 Bitbucket Server (On-Premise) Integration

Teamscale integrates with Bitbucket Server (On-Premise) using the available API and web hooks. The minimum required Bitbucket server version for the integration to work is 5.10.0.

To connect a project to code from a Bitbucket project, no global configuration is required. You just need to follow these steps:

1. Ensure that the public base URL is configured correctly in Teamscale. For this, open the settings page (**Admin**/ **Settings**) and check the entry *Teamscale instance base URL* in tab **Server Settings**. This should point to the URL of the Teamscale server.

2. Select the Bitbucket Server (On-Premise) connector during project creation.

3. Select or create an account for the Bitbucket server. The account's fields should be filled out as follows:
   URI : Bitbucket server's root URL (e.g. https://bitbucket.mycompany.com).
   Username : The username of a user who has *REPO_ADMIN* rights on the Bitbucket server.
   Password : The user's password or *personal access token* with *Admin* permissions for both the project and repository. To obtain a personal access token in Bitbucket, navigate to

your user's *Manage account* page and select the *Personal access tokens* page.

4. Complete project creation as usual, by providing the full name of the Bitbucket repository and completing any other required fields. The name should be in this format *projectKey/repoSlug*.

As the first step in project creation, *Teamscale* will register its web hook with the project. After this, *Teamscale* will automatically be informed about events in the project, such as pushes to the repository and changes to any merge requests.

5.4.8 GitLab Integration

Teamscale integrates with GitLab using the available API and web hooks. No global configuration is required. To connect a project to code from a GitLab project, follow these steps:

1. Ensure that the public base URL is configured correctly in Teamscale. For this, open the settings page (**Admin**/ **Settings**) and check the entry *Teamscale instance base URL* in tab **Server Settings**. This should point to the URL of the Teamscale server.

2. Select the GitLab connector during project creation.

3. Select or create an account for the GitLab server. The account's URL should point to the GitLab server's root URL (not project specific). The username can be empty (ignored). The password is a *personal access token* of a GitLab user with permissions to read and clone the repository and create web hooks. To obtain a personal access token in GitLab, navigate to your user's settings page and select the *Access Tokens* page.

4. Complete project creation as usual, by providing the name of the GitLab project (e.g. *my-user/my-project*) and completing any other required fields. As the first step in project creation, *Teamscale* will register its web hook with the project. After this, *Teamscale* will automatically be informed about events in the project, such as pushes to the repository and changes to any merge requests.

5.4.9 Usage Data Reporting

You can help us to improve *Teamscale* by activating *Usage Data Reporting* in the settings. This option will regularly transfer information about the used features and statistics about errors to our servers. You can configure, which information you are willing to share and also see a preview of the shared information. The preview dialog also contains a link to a web form that allows a one-time usage data reporting by copying the displayed preview information there. Please note that the automatic reporting needs out-bound HTTPS connection to our own servers.

5.5 Backup

This section describes how you can back up the data stored in *Teamscale* and how to restore the data if necessary.

The **Admin** perspective offers a dedicated **Backup** view. Here, you can perform both an **Backup Export** (see Section 5.5.1) and an **Backup Import** (see Section 5.5.2).

5.5.1 Exporting backups

Depending on your needs, e.g., during an update (see Section 5.7), you can export only global data (e.g., users, groups, settings, permissions, dashboards), project data (e.g., blacklists, baselines, and external findings), or both. Furthermore, you can choose to only export project data for some rather than all projects.

Once you have chosen what to export, you have two options to initiate the backup:

1. When you click Download Backup, your browser will download a single backup ZIP and you can store it locally.
2. Alternatively, you can provide an URI (see Figure 5.9) and click Store Backup to let *Teamscale* store your backup remotely. This way is often perferrable if the backup ZIP would be too large to comforably download.

5.5.2 Importing backups

As for exporting backups, you can two options to import a backup:

1. When you click Select a Backup File, your browser will let you select a single backup ZIP which you have downloaded earlier. By clicking Upload Backup you initiate the import.
2. Alternatively, you can provide an URI (see Figure 5.9) you earlier exported a backup to. Then click Import Backup to let *Teamscale* fetch the backup and import it.

Backup URI Schemes

none, file	A path to a backup ZIP on the *Teamscale* instance's local disk, e.g., /var/backups/teamscale/1.zip or file:///var/backups/teamscale/1.zip.
s3+http, s3+https	A location in an S3 bucket, using either HTTP or HTTPS are transport, e.g., s3+http://01234567:abcdefgh@s3.amazonaws.com/backups/teamscale/1.zip. The format of the URI is as follows: (s3+http|s3+https)://[<access key>:<secret key>@]<hostname>[:<port>]/<bucket>/<path> To avoid ambiguity, the S3 bucket must be part of the URI's path, not the hostname. If access key or secret key contain a slash (/), it must be percent escaped (%2F). If desired, access key and secret key can be kept in the **Account Credentials** of external credentials rather then encoded in the URI. To do so, create **Account Credentials** with an ID and URL of (http|https)://<hostname>[:<port>]. Username and password are your access key and secret key, respectively.

Figure 5.9. Backup URI schemes supported by *Teamscale*

5.6 Monitoring

Teamscale provides many helpful metrics via the **System Information** view in the **System** perspective. Additionally, it provides end-points for various monitoring solutions.

5.6.1 Nagios

The URLs *health-check* and *health-metrics* provide check results and metrics in Nagios format[3]. This can be used with Nagios or compatible solutions, such as Sensu, to monitor the current health status of Teamscale. To integrate on the command-line, see the *monitoring* directory in your Teamscale distribution.

5.6.2 Prometheus

The URL *prometheus-metrics* exposes various metrics of Teamscale in Prometheus format[4]. To enable this service, the environment variable `TS_PROMETHEUS_ENABLED` must be set to `true`. Additional protection of the metrics can be provided by setting a secret token in the environment variable `TS_PROMETHEUS_TOKEN`. If this is set, the URL becomes *prometheus-metrics/<secret-token>*.

5.7 Updates

This section contains instructions on how to migrate user- and project-specific data from an existing installation to newer versions of *Teamscale*. Please note that the steps in this guide are for a normal installation and do not apply if you are using docker to deploy *Teamscale*.

[3] https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/3/en/pluginapi.html
[4] https://prometheus.io/docs/instrumenting/exposition_formats/

5.7.1 Minor Version Upgrades

Minor version upgrades are upgrades where the first two parts of the version number are the same. For example, an update from version 4.7.2 to the newer hot-fix version 4.7.4 would be a minor version update. In most cases[5], minor version updates are fully compatible, which means that all configuration files and the database do not need to be modified and thus no re-analysis is needed. To perform such an update, follow these steps:

1. Stop *Teamscale*. This depends on how you installed *Teamscale* and may require you to stop a service or kill the *Teamscale* process.

2. It is strongly recommended to create a backup of the *Teamscale* installation and storage directories, so you can return to the current version if anything does not work as expected.

3. Replace the *Teamscale* installation with the new one. For this you should rename/remove the existing files in the *Teamscale* installation directory and extract the new version there. Make sure to preserve the files from the *config* directory and also preserve the contents of the *storage* and *workspace* directories, if they are within the installation directory. After extraction, the overall file layout should look the same as before.

4. Copy the *config* files from the previous version to the *config* folder of the newly extracted version.

5. Start *Teamscale*. Depending on the installation this may require starting the corresponding service or running start script.

[5] Consult the download site of *Teamscale* for details on individual versions and their compatibility.

After this, *Teamscale* should be running with the new version and the existing data. No re-analysis is required, unless the hot-fix version provides fixes for bugs that you want to apply to the entire history of a project.

5.7.2 Major Version Upgrades

This section describes the general procedure for migrating from previous versions of *Teamscale* that is also applicable for major version updates. As the format of the internal database may change and also new checks become available, a full analysis is required after a major version update. The detailed steps are as follows:

1. Export a complete backup (see Section 5.5) of the configuration and external data via the Backup tab of the **Admin** perspective.
2. Stop *Teamscale*. This depends on how you installed *Teamscale* and may require you to stop a service or kill the *Teamscale* process.
3. It is strongly recommended to create a backup of the *Teamscale* installation and storage directories, so you can return to the current version if anything does not work as expected.
4. Replace the *Teamscale* installation with the new one. For this you should rename/remove the existing files in the *Teamscale* installation directory and extract the new version there. Make sure to preserve the files from the *config* directory. After extraction, the overall file layout should look the same as before.
5. Also delete the storage directory, if this is outside of your *Teamscale* installation directory. In case you have configured an external data directory (workspace), make sure to delete this as well.

6. Copy the *config* files from the previous version to the *config* folder of the newly extracted version. You might need to adjust the configuration, if the format changed. This will be described at the download site of *Teamscale* if required.

7. Start *Teamscale*. Depending on the installation this may require starting the corresponding service or running start script.

8. Now you should have an empty *Teamscale* instance. Import the backup ZIP file into this instance via the **Backup** page of the **Admin** perspective.

9. If you were using the default settings during backup import, the instance will now be in *Shadow Mode* which means that no external events, such as notification emails of voting to other systems will happen.

10. You may have to delete the default administrative user (*admin*) after importing the backup as it is always created when starting *Teamscale* on an empty storage directory.

11. Wait for the analysis to complete.

12. If required, disable the *Shadow Mode* now to make *Teamscale* send notifications and votes again.

Note that due to the required re-analysis, there might be a significant time where *Teamscale* does not yet provide the results for the current head revision. To mitigate this, we recommend a setup with two separate instances, one life instance and one shadow instance, for production use. For an update, you update the shadow instance first, and then switch instances after the analysis was completed. For more details see https://www.cqse.eu/en/blog/teamscale-production-setup/.

5.8 Support

If you need support configuring *Teamscale* please contact us using support@teamscale.com.

Table 5.3. Basic Configuration Settings

Option	Description	Default
`server.port`	*Teamscale* HTTP server port	`8080`
`server.urlprefix`	Prefix of *Teamscale* URLs	`<empty>`
`server.bind-hostname`	Bind address of the HTTP server	`<none>`
`database.directory`	Database directory where all data is stored	`<Teamscale-Dir>/storage`
`database.cache-size`	The cache size used by the database in MB	`512`
`engine.workers`	The number of concurrent analysis worker jobs. When working with multiple projects, this can be used to parallelize analyses. Increasing this value requires the *Teamscale* JVM memory settings to be adapted as well (see Section 5.2.4.2). Allocate about 2GB per worker. For best performance use as many workers as cpu cores are available. On larger instances, use one less worker so one core remains free for handling service requests during high-load situations.	`2`
`servicelog.loglevel`	Log level for logging service calls—one of OFF, INFO, WARN, ERROR	`WARN`

Table 5.4. Basic Configuration Settings (continued)

Option	Description	Default
`servicelog.loguser`	Whether to log the user of service calls	`false`
`database.type`	This is an expert setting and should not be changed. Valid options are: `leveldb`, `rocksdb` and `cassandra`	`leveldb`
`https.port`	Port to be used for HTTPS. If this option is not set, HTTPS is disabled (Please refer to Section 5.2.6 for details on how to configure HTTPS for *Teamscale*.)	
`https.keystore-path`	The absolute path to the Java keystore containing the certificate and private key	
`https.keystore-password`	The password for the keystore	
`https.certificate-alias`	The alias of the certificate and private key in the keystore	

Table 5.5. Default Global Roles

Permission	Description
Instance Admin	The instance admin has all global permissions.
Project Creator	The project creator role allows creating new projects, analysis profiles and metric threshold configurations.
User Manager	The user manager allows creating users and groups.

Table 5.6. Default Project Roles

Permission	Description
Project Administrator	Project administrators have all available project permissions for a project. This role is not editable.
Developer	Developers can just view the project.
Project Lead	Project leads may view a project and additionally manage baselines and tasks.
Architect	Architects may view a project and create and edit architectures for it.
Build	Build permissions allow uploading external data and trigger information on commit hooks for a project.

Table 5.7. Global Permissions

Permission	Description
Create Projects	Allow creation of new projects. The creator has the project administrator role for the project upon creation.
Create Users	Allow creation of users. The creator has the owner role for the user upon creation.
Create Groups	Allow creation of groups. The creator has the owner role for the group upon creation.
Create Analysis Profiles	Allow creation of analysis profiles. The creator has the owner role for the analysis profile upon creation.
Create Metric Threshold Configurations	Allow creation of metric threshold configurations. The creator has the owner role for the metric threshold configuration upon creation.
Create External Credentials	Allow creation of new external credentials for repository connectors, issue trackers etc.. The creator has the owner role for the account upon creation.

Table 5.8. Global Permissions

Permission	Description
Access Administrative Services	Allow access to administrative services. Mostly affects debug services.
Admin Dashboards	Allow viewing and editing all dashboards including admin dashboards.
Backup Global Data	Allow backup of global data like users, groups and permissions.
Assign Global Roles	Allow assigning global roles to users or groups.
Edit Roles	Allow editing the permissions of global- and project roles. Note that a role edit affects all role assignments of that role.
View System Status	Allow viewing the system status via the system perspective.
Edit External Metrics Schema	Allow editing the external metrics schema.
Edit External Findings Schema	Allow editing external findings and finding groups.
Edit Global Notification Settings	Allow editing global notification settings.
Edit Server Options	Allow editing server options via the settings page of the admin perspective.

Table 5.9. Project Permissions

Permission	Description
View Project	View the project in project-specific perspectives.
Edit Project	Edit the project configuration and pause/start the project analysis.
Delete Project	Delete the project.
Edit Baselines	Create, edit and delete baselines for the project.
Edit Tasks	Create, edit and delete tasks for the project.
Edit Blacklist	Blacklist and un-blacklist findings.
Edit Architectures	Create, edit and delete architectures for the project.
Edit Issue Metrics	Edit issue metrics of the project.
Perform External Uploads	Perform external uploads to the project.
Trigger Commit Hook	Trigger scheduling a repository update trigger on a commit hook for a repository of the project.
Assign Roles	Edit role assignments of the project to grant a set of permissions to other users or group members within the project.
Edit Project Options	Edit project options.
View All User Data	View data from other users in the project in case the data privacy option is enabled.

Table 5.10. Basic Roles

Role	Description
Viewer	Only allows viewing the object.
Editor	Allows viewing and editing the object.
Owner	Allows viewing, editing and deleting the object. Furthermore, allows providing access via assignment of roles to other users or groups for the object.

Table 5.11. LDAP Server Options

Option	Description
Hostname	The server's hostname (URL).
Port	The server's port.
SSL	Select this checkbox if the server is using SSL for connections.
Base DN for users	The DN under which the users are stored on the LDAP tree. For example: `ou=users,dc=example,dc=com`.
Base DN for groups	The DN under which the groups are stored on the LDAP tree. For example: `ou=groups,dc=example,dc=com`.
Group attribute	The attribute groups are saved with. For example: `cn`, if the group DN looks like: `cn=GROUPNAME,ou=groups,dc=example,dc=com`.
DN for the initial bind	Full DN of the user used for the connection. This user should have read access to all users and groups. For example: `uid=readadmin,ou=users,dc=example,dc=com`.
Password for initial bind	The password used by the bind user.
Login attribute	The attribute users are saved with. For example: `uid`, if the user DN looks like: `uid=USERNAME,ou=users,dc=example,dc=com`.
Group member attribute	The attribute under which each group stores its members.
First name attribute	The attribute used to store the first name of users.
Last name attribute	The attribute used to store the last name of users.
Email attribute	The attribute used to store the user's email address.

Table 5.12. Typical Settings for Active Directory

Option	Value
Group attribute	cn
Login attribute	sAMAccountName
Group member attribute	member (in some cases memberOf)
First name attribute	givenName
Last name attribute	sn
Email attribute	mail

Table 5.13. Typical Settings for Open LDAP

Option	Value
Group attribute	cn
Login attribute	uid
Group member attribute	memberUid
First name attribute	givenName
Last name attribute	sn
Email attribute	mail

Table 5.14. SAML 2.0 Identity Provider Options

Option	Description
Display name	An arbitrary string that is displayed on the button on the login screen. Example: *Corporate Login*
Service Provider ID	Id of the service provider, i.e. the Teamscale instance. This must match what you configured in your identity provider for Teamscale.
Metadata XML	The entity descriptor metadata XML. This should also contain the identity provider's public key as X.509 certificate.



6. SAP® Integration

This chapter describes how to integrate *Teamscale* with SAP® systems in order to use the analyses provided by *Teamscale* in the area of ABAP® custom code development. The following is relevant for ABAP developers, quality engineers for ABAP, and SAP NetWeaver® system administrators who want to configure *Teamscale*, or those of them who want to connect to *Teamscale* from an SAP system. It covers the SAP and ABAP specific configuration steps and usage scenarios, especially those tasks, which are done within an SAP system. This chapter is irrelevant for all who have nothing to do with custom code development for SAP.

Regarding questions about non-ABAP-specific installation and configuration, refer to the common administration guide in Chapter 5. For general usage of *Teamscale* see the user reference in Chapter 4.

6.1 Teamscale Connector for SAP NetWeaver® AS ABAP®

The *Teamscale Connector for SAP NetWeaver® AS ABAP®* provides the connection between *Teamscale* and SAP NetWeaver AS ABAP based systems. The connector has to be transported to the SAP NetWeaver system and provides the following main functions:

- An RFC-based interface, which is used by *Teamscale* to incrementally extract source code from the SAP system for code analysis within *Teamscale*.
- Execution of SAP Code Inspector (SCI) and export of the inspection results to *Teamscale*.
- Extraction of test coverage and usage data for *Teamscale* to support test gap analysis (see Section 1.2.3) and usage analysis (see Section 1.2.5).
- A plug-in for the ABAP Workbench and the Change and Transport System to access *Teamscale* results from the SAP system. This is realized by a custom SAP Code Inspector test case which shows messages for findings identified in *Teamscale*.

The *Teamscale Connector for SAP NetWeaver® AS ABAP®* is compatible with SAP NetWeaver AS ABAP 7.0 EHP2 and later versions. It should work on earlier versions, but this has not been tested.

6.1.1 Installation of the Connector

An archive file containing the latest release of the *Teamscale Connector for SAP NetWeaver® AS ABAP®* can be downloaded via https://www.cqse.eu/get-abap. Download either the `.zip` or `.tar.gz` source code file. Each archive contains the `K*` and `R*` transport files as well as the file `README.md` containing release notes.

To install the connector on an SAP system, the transport files must be transported into the system. The archive contains two transports:

- The main component of the *Teamscale Connector for SAP NetWeaver® AS ABAP®*. All objects are in `/CQSE/` namespace and within package `/CQSE/TEAMSCALE_CONNECTOR` and its subpackages. This transport is always required.
- A default role `/CQSE/TEAMSCALE_RFC` for RFC access from the *Teamscale* instance to the *Teamscale Connector for SAP NetWeaver® AS ABAP®*. This transport is optional, alternatively an appropriate role can be defined manually.

For the mapping of transport files to the mentioned transports, see file `README.md` in the archive.

For transport, place the `K*` and `R*` files from the archive in the `cofiles` and `data` directories of the transdir of your host machine (typically `/usr/sap/trans` on Unix/Linux or `C:\usr\sap\trans` on Microsoft Windows) and import the transport (transaction `STMS`). Enable special conditions »Overwrite Originals« and »Ignore Invalid Component Version«, see Figure 6.1.

Basically, no further configuration in the SAP system is required. One exception is the use of *Secure Network Communications (SNC)*, see Section 6.4.1 for details.

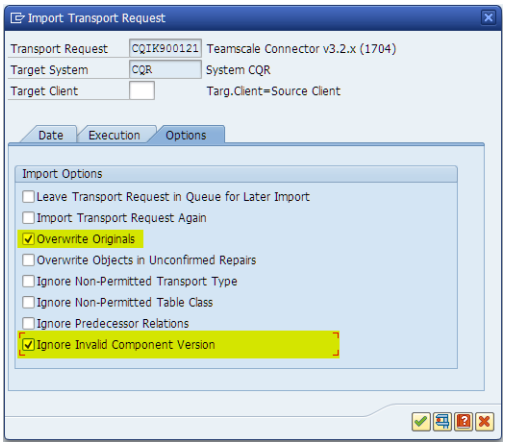


Figure 6.1. Transport Import Options

6.1.2 User and Role for RFC Access

Teamscale connects to the SAP application server via RFC (remote function call) to retrieve the source code, which should be analyzed as well as some metadata (for example test coverage). The called functions are in function group `/CQSE/TEAMSCALE_CONNCT_RFC`. Hence, a (technical) SAP user is required, which will be used by *Teamscale* to connect to the SAP system.

6.1.2.1 Role and Required Access Rights

The SAP user for *Teamscale* must have an appropriate role assigned. The required authorizations are listed in Table 6.1. The authorizations are required for basic RFC access (including background RFC execution), read access to development objects (source code) and for enabling test coverage analysis.

Table 6.1. Authorizations for SAP *Teamscale* User

Authorization Object	Field	Value
S_RFC	RFC_TYPE	FUGR (Function Group)
	RFC_NAME	/CQSE/TEAMSCALE_CONNCT_RFC
S_RFC	RFC_TYPE	FUNC (Function Module)
	RFC_NAME	BGRFC_CHECK_UNIT_STATE_SERVER BGRFC_DEST_CONFIRM BGRFC_DEST_SHIP DDIF_FIELDINFO_GET RFCPING RFC_GET_FUNCTION_INTERFACE
S_COV_ADM	ACTVT	02 (Change)
S_DEVELOP	DEVCLASS	*
	OBJTYPE	*
	OBJNAME	*
	P_GROUP	*
	ACTVT	03 (Display)

The role can be defined manually, e. g. using transaction PFCG, or the predefined role /CQSE/TEAMSCALE_RFC can be used.

6.1.2.2 RFC User for Teamscale

The SAP user for *Teamscale* can be created in the usual way users are crated in the SAP system, e. g. using transaction SU01. You should not use a *dialog* user but a *system* user[1]. The role described before (e.g., /CQSE/TEAMSCALE_RFC) must be assigned to this user.

[1] Using a dialog user might result in connection timeouts as dialog connections are by default terminated after 10 minutes without user interaction. The initial extraction of the source as well as the retrieval of SAP Code Inspector results might take longer.

It is highly recommended that the user time zone of the SAP user for *Teamscale* is the same as the SAP system time zone. If the user time zone differs from the system time zone, the continuous export of code changes might not work correctly and code changes might not be visible or only with a longer delay. The time zone setting of the user can be checked / edited in transaction `SU01`.

Additionally, it is highly recommended that the system time zone in SAP NetWeaver is the same as the time zone of the operating system.

6.2 SAP Connection Configuration in *Teamscale*

This section describes the basic configuration, which must be done within the web interface of *Teamscale*. For configuration, switch to the **Admin** perspective and there to the page **Settings**.

6.2.1 Configuring Working Directory

First, you should check if a proper working directory is specified in section **Server Settings** / **Teamscale Working Directory**. For the SAP connector, it is required that the working directory is explicitly specified. *Teamscale* will save the (historized) code exports in sub-directories of this directory and access them for analysis there.[2] Usually, the working directory will be specified by the system administrator of the *Teamscale* server, you should check with him/her to define the proper location. It can be specified as absolute or relative

[2] Technically, this is realized by a local Git repository which is managed by *Teamscale*.

SAP ABAP System Connection

SAP ABAP application server	Mandatory parameter for **direct connection**: The full qualified host name of the SAP NetWeaver application server, e.g. `saphost.me.com`
Instance number	Mandatory parameter for **direct connection**: The instance number (formerly called *system number*) of the SAP NetWeaver application server, e.g. `00`
System ID	Mandatory parameter for **logon balancing connection**: 3-letter SAP system ID, e.g. `NSP`
SAP message server	Mandatory parameter for **logon balancing connection**: URL of SAP message server
Group	Mandatory parameter for **logon balancing connection**: Group of SAP NetWeaver application servers (e.g. `public`)
SAP message server port	Optional parameter for **logon balancing connection**: Port of message server, typically `36xx` where xx refers to the instance number
SAP router string	Optional parameter for connection to systems behind an SAP Router: SAP router string contains the chain of SAP routers and their port numbers and has the form `(\H\<host>[\S\<port>])+`
SAP client	Mandatory parameter: Defines the client used for logon (e. g. `001`)
Logon user	Mandatory parameter for password-based authentication: The SAP user name which should be used by *Teamscale* (see Section 6.1.2)
Logon password	Mandatory parameter for password-based authentication: Password of the user

Figure 6.2. Configuring System and Logon Parameters for SAP Connection

path. In case of a relative path, the location is relative to the parent of the *Teamscale* storage directory (parameter `database.directory` of `teamscale.properties` configuration file, see Section 5.2.1).

6.2.2 Configuring SAP ABAP System Connection

Under **SAP** / **SAP ABAP System Connection** the parameters to connect to an SAP NetWeaver system must be specified. For every SAP system, a separate connection configuration must be added. To add a new connection, press the Add button. In the opening dialog, a unique connection ID must be specified, for example the three letter SAP system name could be used. After this, a new configuration section is added, in which the parameters must be specified.

The first options are the parameters for the RFC connection. The required parameters mainly depend on the connection type, i. e. if the connection to the SAP system is direct or via logon balancing. Further options specify the logon data (client, user, password). Figure 6.2 describes these parameters.

Further options of the **SAP ABAP System Connection** parameters control the code retrieval of *Teamscale* from the SAP system. These specify which code should be retrieved, how often it should be updated and if other result data should be included (e.g., test coverage data). Figure 6.3 describes these parameters. The most important parameters specify the exported objects and schedule the update intervals.

Additional parameters are required for secure RFC communication using *Secure Network Communications (SNC)*, see Section 6.4.3 for a description of these parameters. Together with the ABAP code retrieval from the SAP system also the execution of SAP Code Inspector

SAP ABAP System Connection

Include objects in default custom namespaces	Specifies if ABAP source code objects within the default custom namespaces `Y... / Z...` should be exported to *Teamscale*.
Further custom namespaces	Comma-separated list of further custom namespaces like `/ABC/, /DEF/` for which ABAP code should be exported.
Schedule incremental code update	Defines in which interval continuous code changes are retrieved from the SAP system.
Schedule full code update	Defines in which interval a full synchronization of the code base is performed.
Execute full update asynchronously	If enabled, full export is scheduled asynchronously using the background RFC (bgRFC) protocol.
Name of the inbound queue	Only required if **Execute full update asynchronously** is enabled, specifies the name of the inbound bgRFC queue in the SAP system.
Enable ABAP exports archive	If enabled, the exported code changes are archived as Zip files on the disk.
Include programs generated from BW objects	If enabled, source code objects generated from SAP BW objects should be included in the code / coverage retrieval. This is used for test gap analysis. Note the version prerequisites at the end of this Section.
Include code coverage data in incremental exports	Specifies if coverage data is retrieved for each incremental export for all objects in the selected namespaces regardless of the code changes.
Include code coverage data in full exports	Specifies if the coverage data is retrieved during a full export for all objects in the selected namespaces.
Clear table COVRES after the export	If enabled, the table `COVRES` which holds coverage data will be cleared after retrieving the coverage data.

Figure 6.3. SAP Code Retrieval Parameters (for SAP Code Inspector Parameters see Figure 6.9)

checks in the SAP system can be performed. See Section 6.5.1, which describes the Code Inspector configuration in more detail.

Namespace Configuration The namespace parameters **Include objects in default custom namespaces** and **Further custom namespaces** specify the code base which should be retrieved from the SAP system. It is recommended to specify all custom namespaces, which are available for development, e.g., enable the retrieval of the default custom namespaces `Y` and `Z` as well as all further custom namespaces e. g. `/ABC/` which are registered for your organization. Note that these parameters only define for which code *Teamscale* keeps track of the change history, but not which code gets analyzed. The actual selection of the analyzed code is specified in the project configuration (see Section 4.1.11.2).

Update Intervals The schedule parameters **Schedule incremental code update** and **Schedule full code update** define when code (and further data, e. g. coverage results) is retrieved from the SAP system. Two different intervals must be defined: The *incremental update interval* and the *full synchronization interval*. The incremental update only retrieves the objects which have changed since the last retrieval, thus the amount of transferred data is low and the execution time is very short (a few seconds only). A change to an object is observed, if an object gets activated. Here, intervals between 10 and 30 minutes are recommended. Longer intervals may result in an unwanted long delay of analysis results after activation of code changes, shorter intervals may lead to a too fine-grained change history.

The full synchronization interval controls when an export of all code objects (within the specified) namespaces is performed. Since in rare cases the code base can change without the incremental export recog-

nizing the change, a regular synchronization export should be scheduled. This ensures that the exported code repository is synchronized with the actual code repository within the SAP system. Usually, no code change is visible during synchronization. Typically, such missed changes which are only visible in synchronization do not result from regular development but may be caused by special transport or upgrade activities. We recommend a daily update during night.

Both intervals are specified as `cron`[3] expressions, for example `*/10 * * * *` specifies an update of every 10 minutes, `0 1 * * *` a nightly update at 1:00am. Typically it will take some minutes until the initial export occurs.

Asynchronous Communication The full code update can be also configured to be executed by asynchronous communication where the exeuction will run in a background job of the SAP system. Use this mode if the run-time of the code retrieval takes too long time. This is configured by the settings **Execute full update asynchronously** and **Name of the inbound queue**. See Section 6.3 for a detailed description.

BW Objects Export of source code objects generated from SAP BW objects is supported from *Teamscale* release version 5.1.6 and *Teamscale Connector for SAP NetWeaver® AS ABAP®* version 1807 onwards.

[3] https://en.wikipedia.org/wiki/Cron

6.2.3 Configuring Account Credentials

After the code retrieval is configured and executed for the first time, *Teamscale* creates an ABAP repository directory inside the working directory. This directory contains a Git repository for every SAP system. As for any repository, account credentials must be configured for accessing it. By default, Teamscale creates a common entry **ABAP Repositories**, which points to `<working-dir>/sap.abap.system`. With these account credentials, all ABAP repositories can be accessed. It is recommended to use these credentials in the project configuration.

If you need to configure the account credentials manually for a particular SAP connection, go to section **Account Credentials** (also at the **Settings** page of the **Admin** perspective) and add new credentials, the unique ID can be arbitrarily chosen, e.g., `abap-Git-<sap-connection-id>`. In the details, the only field which must be filled is the URL field, which is named **The URL of the external server**. The URL must be specified as file URL in the following format:

- For Unix/Linux based systems:
  `file:///<working-dir>/sap.abap.system/<sap-connection-id>/git`
- For Windows based systems:
  `file:///<drive-letter>:/<working-dir>/sap.abap.system/<sap-connection-id>/git`

Where

`<working-dir>` refers to the *Teamscale* working directory (relatively to system root, on Windows backslashes must be replaced by forward slashes), see Section 6.2.1

<sap-connection-id> refers to to the identifier of the SAP System connection, see Section 6.2.2

<drive-letter> refers to the Windows drive letter, e.g., `C`.

6.2.4 Configuring a *Teamscale* Project

After the SAP system connection is specified and the code retrieval was executed for the first time, projects for analyzing the ABAP code of the respective SAP NetWeaver system can be created. This follows the standard project configuration as described in Section 4.1.11.2, where the Git connector and an ABAP-specific analysis profile must be used.

For the Git connector, you should use the default account credentials **ABAP Repositories** together with a path suffix of the form `<sap-connection-id>/git`, see Figure 6.4 (there, `CQI` is the identifier of the SAP System Configuration).

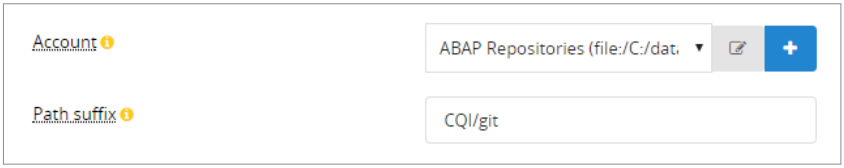


Figure 6.4. Git Connector Configuration for ABAP

Note that the settings to include/exclude source code are specified using the parameters **Included file names** and **Excluded file names**, which refer to the file paths within the internal Git repository of the ABAP source code. The path consists of the the full package hierarchy and source code files suffixed with `.abap`. Furthermore, the slashes at custom namespaces are replaced by an exclamation mark, thus `/XYZ/` is replaced with `!XYZ!`. Table 6.2 states some examples of ant patterns for including and excluding ABAP source code.

Table 6.2. Examples for ABAP Code File Include/Exclude Patterns

Pattern	Matched ABAP Source Code Objects
`**.abap`	All ABAP source code objects in configured namespaces (** matches any path)
`$*/**.abap`	ABAP source code within local packages (starting with `$`, * matches all but the path/package separator slash)
`ZPACKAGE/**.abap`	Objects within the top-level package `ZPACKAGE` and its sub-packages only. Will **not** match if `ZPACKAGE` has a super package
`**/ZPACKAGE/**.abap`	Objects within the package `ZPACKAGE` and its sub-packages only, regardless if `ZPACKAGE` is a top-level package
`**/!XYZ!*/**.abap`	Objects within packages in namespace `/XYZ/` only
`!XYZ!ROOT_PACKAGE/**.abap`	Objects within the top-level package `/XYZ/ROOT_PACKAGE` and its sub-packages only
`**/PROG/*.abap`	Programs and program includes only
`**/FUGR/**.abap`	Function groups only
`**/FUGR/ZSOME_FUGR/*.abap`	Function group `ZSOME_FUGR` only
`**/CLAS/**.abap`	Global classes only
`**/CLAS/!XYZ!CL_BASE_**.abap`	Global classes where name starts with `/XYZ/CL_BASE_` only
`**/CLAS/*/*CCAU.abap`	ABAPUnit test classes only (class addition suffixed with `CCAU`)
`**/INTF/**.abap`	Global interfaces only

Table 6.3. Examples for ABAP Dictionary File Patterns

Pattern	Matched ABAP Dictionary Objects
`**.abap_ddic`	All ABAP Dictionary objects in configured namespaces (** matches any path)
`**/DOMA/*.abap_ddic`	Domains
`**/DTEL/*.abap_ddic`	Data elements
`**/STRU/*.abap_ddic`	Data structures
`**/TABL/*.abap_ddic`	Database tables
`**/TTYP/*.abap_ddic`	Table types
`**/VIEW/*.abap_ddic`	Database views

ABAP Dictionary (DDIC) objects. Besides the ABAP source code in `*.abap` files, also a textual representation of objects defined in the *ABAP Dictionary (DDIC)* is exported from the SAP system. The dictionary objects are stored in `*.abap_ddic` files. Adding these files to the project will allow to identify dependencies from ABAP source code to ABAP Dictionary objects and between dictionary objects when performing architecture conformance analysis. The textual representation of structures and database tables is the same as the representation used in the source code based editors of the ABAP Development Tools for Eclipse (available from SAP NetWeaver 7.51 or 7.52 on, respectively). For older SAP NetWeaver versions and further dictionary objects, a similar notation is used in *Teamscale*. As in *Teamscale* the only purpose of the dictionary objects is to perform architecture conformance analysis, `.abap_ddic` files will only contain the required information to identify dependencies and omit further attributes.

Table 6.3 states typical examples for including and excluding ABAP Dictionary objects. Of course, these path patterns can also be com-

bined with package specifications, similar to the ABAP source code files. The include/exclude pattern for dictionary objects must be added to the list of include/exclude patterns besides the patterns for the ABAP code.

6.3 Asynchronous Full Export

By default, all updates of the ABAP code are retrieved by synchronous communication from the SAP system. The full synchronization update, where the entire code of the configured namespaces is exported, can be alternatively configured to use asynchronous communication. Asynchronous communication is used to avoid a long run-time of the export job what would block other maintenance jobs and/or result in communication timeouts. A long run-time is to expect especially if SAP Code Inspector check execution is configured. The SAP NetWeaver *bgRFC* protocol is used for asynchronous communication. For details refer to https://help.sap.com/. It is required that *bgRFC* is configured accordingly in the SAP system.

In the asynchronous mode, a full synchronization update will only schedule a background job in the SAP system first. *Teamscale* is not waiting for the update job to finish and closes the connection to the SAP system immediately after scheduling the job. The collection of the source code and meta-data and especially the execution of SAP Code Inspector checks is then executed in the background on the SAP system. When the background job finishes, the results will be stored in table `/CQSE/ASYNCTSEXP`. For retrieving the results, *Teamscale* is polling the SAP system regularly if the scheduled jobs are finished and results are available. If so, the result data is retrieved from the SAP system and the internal ABAP repository of *Teamscale* gets updated.

The incremental update, which only contains changed objects, is always in synchronous communication mode. Also the initial code retrieval is done synchronously but will not execute SAP Code Inspector to make the entire code base visible in *Teamscale* without a too long delay.

6.3.1 *bgRFC* Configuration in the SAP System

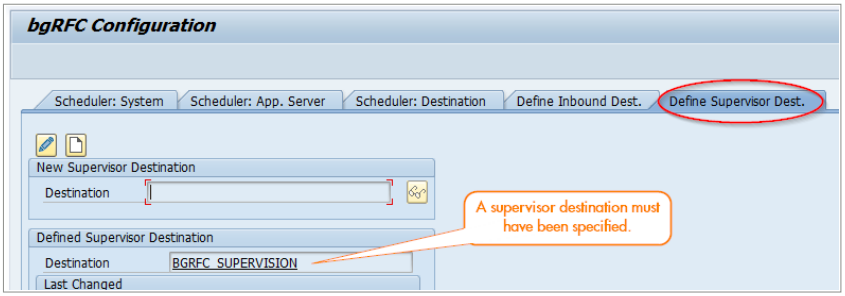


Figure 6.5. Define Supervisor Destination in SBGRFCCONF

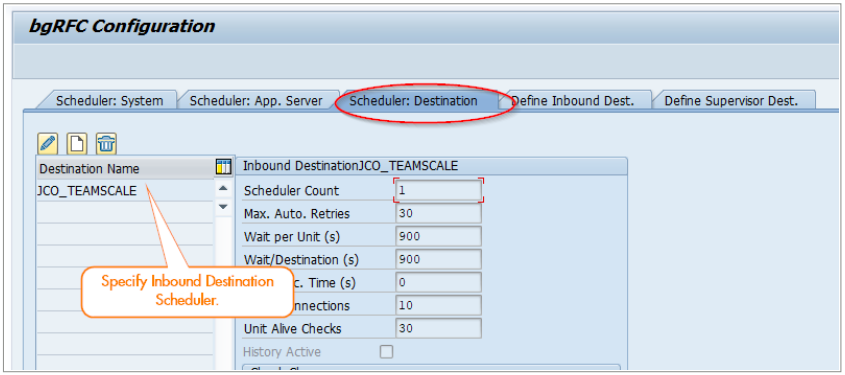


Figure 6.6. Define Destination Scheduler in SBGRFCCONF

To be able to use the asynchronous communication, the *bgRFC* queue used by Teamscale must be configured in the SAP system. This is done in transaction SBGRFCCONF. The following needs to be configured in SBGRFCCONF:

- In tab *Define Supervisor Dest.* (see Figure 6.5) a bgRFC supervisor destination must be defined. If this is not yet the case, a new one can be created using the *create* button. The destina-

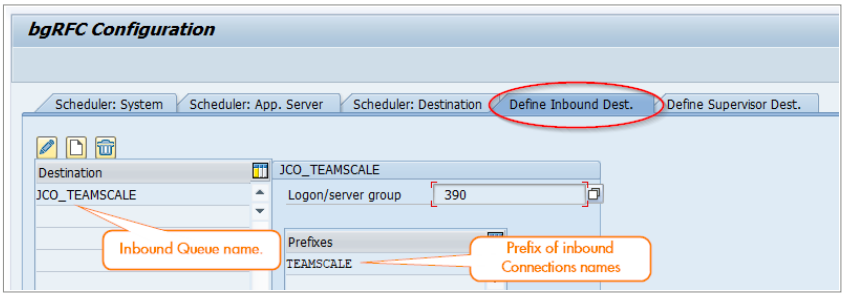


Figure 6.7. Define Inbound Destination in SBGRFCCONF

tion requires a name and to specify a bgRFC supervisor user (the user can be created directly in the opening dialog).

- In tab *Define Inbound Dest.* (see Figure 6.7) a destination must be defined as an inbound destination and an identifying prefix must be assigned. A matching name must be used in the Teamscale connection settings, as described in the next section.
- In tab *Scheduler: Destination* (see Figure 6.6) a scheduler for the beofre defined destination must be defined. A new one can be crated using the *create* button.

The settings must be saved (`Ctrl+S`) before leaving this transaction.

To check the state of queued jobs in the SAP system, transaction SBGRFCMON can be used.

6.3.2 Asynchronous Export Configuration in Teamscale

The main configuration in Teamscale is made by the two options **Execute full update asynchronously** and **Name of the inbound queue** of the **SAP ABAP System Connection** settings.

To enable asynchronous full updates, the check-box **Execute full update asynchronously** must be enabled and a name matching the inbound queue prefix (as specified in the SAP system in `SBGRFCCONF` under *Define Inbound Dest.*, see above) must be specified.

Furthermore, the scheduling interval of the polling job for retrieving the finished results can be specified in the **SAP ABAP System Connections - Global Settings** of the admin settings in field **Schedule for requesting results of asynchronous ABAP exports**. Also it is configurable in the global options how long scheduled jobs are polled from *Teamscale* if no results could be retrieved for a long time, this is configured in parameter **Maximum wait time**.

6.4 Secure RFC Communication (optional)

If the RFC communication between *Teamscale* and SAP NetWeaver should be secured (encrypted), *Teamscale* can also connect via *Secure Network Communications (SNC)* to the SAP system. In order to use it, a respective configuration of the SAP system and the *Teamscale* server is required.

6.4.1 SNC Configuration of the SAP System

To enable SNC, the SAP system must be configured accordingly. In the following a rough outline of the configuration is stated. As the configuration steps vary depending on the SAP version and on the individual configuration needs for a customer SAP system, please refer to https://help.sap.com/ for detailed instructions on how to set-up SNC. Furthermore, there is a very helpful SAP Community blog post[4].

[4] https://blogs.sap.com/2006/09/29/setup-data-encryption-between-rfc-client-and-web-as-abap-with-snc/

The recommended SNC configuration steps for an SAP NetWeaver system are as follows:

1. Ensure that the following instance parameters (transaction `RZ10`) are set:

   ```
   snc/enable = 1
   ```

   ```
   snc/identity/as = snc_name
   ```

   Where *snc_name* is the SNC name of the SAP system, something like `p:CN=SID, O=ACompany, C=EN`.

2. The *SAP Cryptographic Library* must be installed, see SAP Note 1848999.

3. With the command line tool `sapgenpse` of SAP Cryptographic Library, a *Personal Security Environment* (PSE, stored in a `*.pse` file) for *Teamscale* must be created, This can be done by executing `sapgenpse` on the application server (or any other sever where the SAP Cryptographic Library is available). For example, enter the following command:

   ```
   sapgenpse gen_pse -v -p teamscale.pse
   ```

   During creation of the PSE, the tool will ask for the name of the PSE owner, here the SNC name for *Teamscale* should be supplied (something like `p:CN=TEAMSCALE, O=ACompany, C=EN`). Afterwards, the client certificate (`*.crt` file) must be exported, again by using `sapgenpse`:

   ```
   sapgenpse export_own_cert -v teamscale.pse
   -o TEAMSCALE.crt
   ```

   This file contains the X.509 client certificate which can be also used for logon instead of user name and password, see below.

4. In transaction `STRUST` the following configuration is required:

- The PSE *SNC (SAPCryptolib)* must have been created. A respective entry should already exist after the SNC instance parameters were set and the SAP system was restarted. The SNC name should be the same as specified in `snc/identity/as`
- The client certificate created previously (e. g. `TEAMSCALE.crt`) must be imported and added to the PSE *SNC (SAPCryptolib)*. *Base64* file format should be used.
- The own (server) certificate of *SNC (SAPCryptolib)* must be exported to a file, e. g. `SNC.crt` (again, use *Base64* file format).

5. The server certificate (e. g. `SNC.crt`) must be imported in the client PSE file (e. g. `teamscale.pse`) and the credentials file `cred_v2`, which will be used on the *Teamscale* server, must be created from the client PSE file. For both steps, again the command line tool `sapgenpse` is used, see the following examples:

```
sapgenpse maintain_pk -v -a SNC.crt -p teamscale.pse
```

```
sapgenpse seclogin -p teamscale.pse -O root[5]
```

6. The Access Control List (ACL) for SNC restrictions must allow RFC communication with the client: In view `VSNCSYSACL` (transaction `SM30`), an entry for work area *external (E)* with the SCN name of the client (*Teamscale*), e. g. `p:CN=TEAMSCALE, O=ACompany, C=EN`, must be added.

7. If you want to use the client X.509 certificate also for logon instead of user name and password, the certificate must be mapped to the SAP *Teamscale* user. To do so, an entry in

[5] The `-O` option is used to specify other users, who should not be able to read the `cred_v2` file - this should be adopted accordingly.

view VUSREXTID (transaction SM30) must be added, again for the SNC name of the client.

6.4.2 SNC Configuration on the *Teamscale* Server

On the *Teamscale* server, the following configuration steps are required:

1. The library files from *SAP Cryptographic Library* must be saved on the *Teamscale* server, *SAP Cryptographic Library* can be downloaded from SAP Support Launchpad for the required operating system, search for COMMONCRYPTOLIB 8.

2. The cred_v2 file must be moved to a directory on the *Teamscale* server. The file should be only accessible for the operating system user running *Teamscale*.

3. The environment variable SECUDIR must have been defined pointing to the directory containing cred_v2.

4. If the X.509 client certificate should be used for logon (SSO mechanism), the client certificate file (e. g. TEAMSCALE.crt) must be saved on the *Teamscale* server.

6.4.3 SNC Configuration in *Teamscale*

In the Settings page of the Admin perspective, you have to specify the path to the directory containing the library files of *SAP Cryptographic Library* under SAP / SAP ABAP System Connections - Global Settings.

Under SAP ABAP System Connection SNC must be enabled and the SNC options must be specified (see Figure 6.8).

SAP ABAP System Connection

Enable Secure Network Communications (SNC)	Tick to enable SNC
SNC name of the communication partner server	The SNC name of the SAP system, something like `p:CN=SNC, O=ACompany, C=EN`
Own SNC name of the caller	The client SNC name of *Teamscale*, something like `p:CN=TEAMSCALE, O=ACompany, C=EN`
Enable the SSO mechanism of SNC	Enable logon by X.509 certificate instead of user name/password
Path to X.509 certificate	Path to `*.crt` file holding the X.509 client certificate, only required if SSO mechanism is enabled

Figure 6.8. Configuring SNC Parameters for SAP Connection

6.5 Additional Analysis Configuration

This section describes additional configuration steps required for specific analyses in *Teamscale*, namely the upload of SAP Code Inspector findings and the usage of coverage results for test gap analysis.

6.5.1 Integrating SAP Code Inspector

Messages resulting from checks of *SAP Code Inspector (SCI)* can be also shown as findings in *Teamscale*. Thus all findings management features of *Teamscale* (e. g. working with baselines, blacklists, the definition of tasks, ...) are also available for SCI messages. Once con-

figured, no further manual actions are required by developers to get results of the SCI tests.

The integration of SAP Code Inspector findings in *Teamscale* requires that an appropriate Code Inspector variant is defined in the SAP system and that the SAP system connection settings and the project configuration in *Teamscale* is set up accordingly.

6.5.1.1 Selecting/Defining a Code Inspector Check Variant

An appropriate Code Inspector *variant* must be specified, or you can use an existing variant. The variant defines, which checks are executed by Code Inspector. We recommend to enable all tests which might be relevant. A detailed selection of the messages which should lead to findings can be done in the *Teamscale* analysis profile. Tests which will produce a high amount of irrelevant messages should be excluded, otherwise the storage and backup size of *Teamscale* could increase heavily. Further, the execution time of the Code Inspector checks might be quite long, especially if the whole custom code base is checked. If the execution time is too long, you should also switch off irrelevant checks.

To specify a variant, execute transaction `SCI` and create a new check variant. You should consider to use a public variant as public variants can be made transportable, then the variant could be transported to further systems.

You may also ask our *Teamscale* support[6] for a pre-defined check variant.

[6] Get either in touch with your personal contact person at CQSE or contact us at support@teamscale.com.

6.5.1.2 Code Inspector Settings in SAP ABAP System Connection

SAP ABAP System Connection

Name of the SAP Code Inspector variant	Name of an SAP Code Inspector variant which should be executed.
Owner of the SAP Code Inspector variant	Specifies the SAP user name of the variant owner (leave empty for public variants).
Code Inspector include specification	Defines for which code objects Code Inspector should be executed.
Code Inspector exclude specification	Defines exclusions of code objects for Code Inspector execution.

Figure 6.9. Parameters for SAP Code Inspector Execution during Code Retrieval

To execute SAP Code Inspector during code retrieval of *Teamscale*, this must be configured in the admin settings of **SAP ABAP System Connection**, as shown in Figure 6.9. The name of the executed Code Inspector variant must be specified in parameter **Name of the SAP Code Inspector variant**. If this is not a public variant, but a user variant, the SAP user name of the owner must be additionally specified in the parameter **Name of the owner of the SAP Code Inspector variant**.

The parameters **Code Inspector include specification** and **Code Inspector exclude specification** are used to restrict the code base checked by *SAP Code Inspector*. These includes/excludes are stated as a list of Ant patterns matching against the file paths of the ABAP source code objects as visible in *Teamscale*, this is similar to the patterns used for project configuration, see Section 6.2.4 and the exam-

ples in Table 6.2. Note that the the single star wild card (`*`, matching a sequence of chars but not `/`) cannot be used in these patterns.

With this configuration, Code Inspector is executed in the SAP system during every code retrieval. During the incremental update, only the changed ABAP repository objects are checked by SAP Code Inspector, but during the full synchronization update, Code Inspector is executed for all ABAP repository objects in the specified namespaces. As the full synchronization may take a long time (up to a few hours for code bases of many million lines of code), we recommend to schedule the full synchronization update at most once every night.

6.5.1.3 Code Inspector Settings in Project Configuration

To actually include findings from SAP Code Inspector in a *Teamscale* project, the following must be configured:

- In the analysis profile, **Code Inspector** must be enabled as analysis tool, and analysis groups for Code Inspector must be added in the quality indicators.
- **Included file names** of the Git connector must also include files with suffix *.sci*, e.g., by pattern `**.sci`.
- Under the expert options of the Git connector, the **Analysis report mapping** rule `**.sci -> SAP_CODE_INSPECTOR` must be specified.

The latter two are set as defaults for new projects.

If all the above is correctly configured, findings from SAP Code Inspector should be visible in *Teamscale* like any other *Teamscale*-internal finding and are updated simultaneously with every code change. Note that during the incremental code update, only changed repository objects are analyzed by SAP Code Inspector. Findings from

Code Inspector which affect unchanged objects will only be visible after the full synchronization update of the ABAP repository was performed.

6.5.2 Test Gap Analysis Set Up

To perform test gap analysis (see Section 3.5) it is required that the *SAP Coverage Analyzer (SCOV)* is collecting coverage data from (test) executions. Thus, it is necessary to activate the coverage analyzer SCOV (Lite). This has to be done manually for every system.

Depending on the actual system, either SCOV Lite (if available) or the regular SCOV has to be used as coverage analyzer. SCOV Lite is the preferred method, as it has a smaller performance impact. The following two sections describe how to activate SCOV Lite and standard SCOV. Please make sure to check which case applies in your situation. In either case, the activation requires the authorization object `S_COV_ADM`.

Both SCOV modes will schedule the batch job `RSCVR_TRIGGER_COLLECT` to run every 45 minutes. By default, SAP NetWeaver configures this job to be executed by the user that activates SCOV. We suggest to edit this batch job and use a technical user instead in order to be independent of personal account availability and permissions.

Note that background jobs in SAP are always scheduled in the system time zone and do not respect the user time zone. Thus, errors may occur during SCOV activation if the user who activates SCOV has a different user time zone setting. The user time zone can be checked/modified in transaction `SU01`. It is recommended that a user with no specific user time zone setting performs the activation.

6.5.2.1 Activating SCOV Lite

The analysis only needs the lite version of SCOV, which cannot be activated via the transaction `SCOV`, but via the *Usage Procedure Logging (UPL)* functionality of SAP Solution Manager or programmatically. The *Teamscale Connector for SAP NetWeaver® AS ABAP®* includes a utility which allows to toggle the SCOV Lite mode on and off. To execute it, call transaction `/CQSE/SCOV_LITE_TOGG`. Once executed, it should confirm with the message »Toggled SCOV state from [OFF] to [LIT]« (see Figure 6.10) — if the message says ».. from [LIT] to [OFF]«, execute the toggle again.

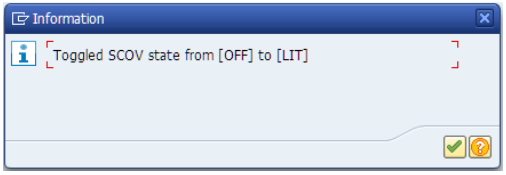


Figure 6.10. SCOV Lite Activation

Should there be problems at this point, the coverage analyzer has to be activated via the transaction `SCOV` (see below).

6.5.2.2 Activating SCOV

The coverage analyzer is configured via the transaction `SCOV`. Here it is important that both the coverage analyzer and the data collection for the server are activated. A detailed description of the activation can be found at http://help.sap.com/. No specific settings (e.g., test groups) need to be configured. Figure 6.11 shows the required configuration, the important parts are marked in yellow.

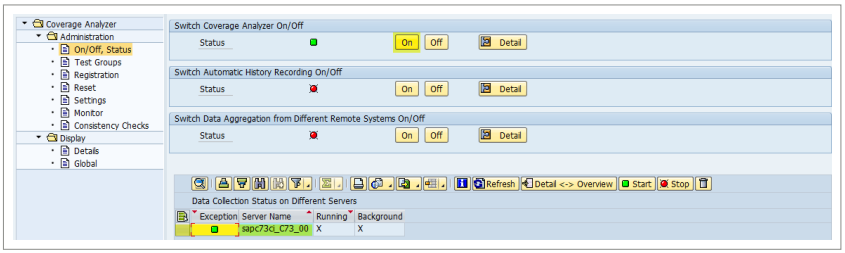


Figure 6.11. SCOV Activation (Normal Mode)

6.5.2.3 Ensuring SCOV is Active

In order to make sure SCOV is functional and records data as expected, the start screen of transaction `SCOV` can be checked. Depending on the mode in which SCOV is used, the expected output is slightly different.

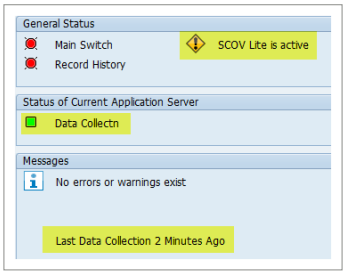


Figure 6.12. Check SCOV Lite

When using using **SCOV Lite**, please ensure the following (see also Figure 6.12):

- The *General Status* area shows *SCOV Lite is active*.
- The *Status of Current Application Server* area has a green indicator next to *Data Collectn*.

- The *Messages* area does not show errors.
- The *Last Data Collection* is less than 45 minutes ago.

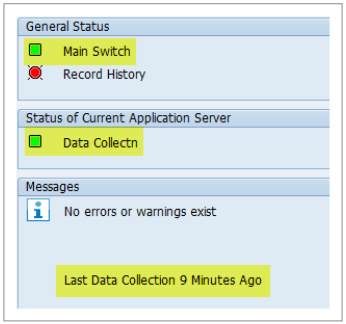


Figure 6.13. Check SCOV (Regular Mode)

When using the **regular SCOV**, please ensure the following (see also Figure 6.13):

- The *General Status* area has a green indicator next to *Main Switch*.
- The *Status of Current Application Server* area has a green indicator next to *Data Collectn*.
- The *Messages* area does not show errors.
- The *Last Data Collection* is less than 45 minutes ago.

6.5.2.4 Troubleshooting: No Coverage Information

If the results do not look as expected, most of the time it is enough to use the repair functionality that is included in SAP Coverage Analyzer. Execute transaction `SCOV` and run the consistency checks with repair option, see Figure 6.14.

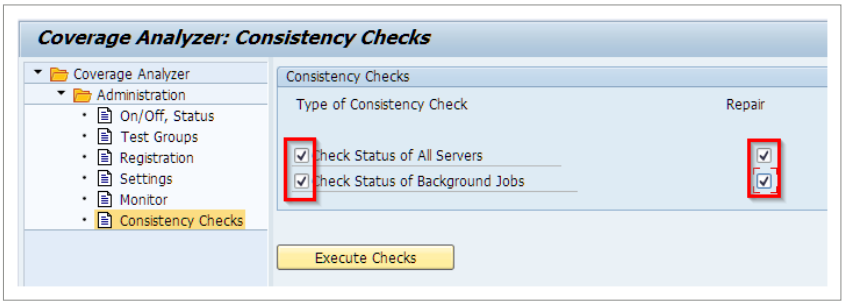


Figure 6.14. SCOV Consistency Checks & Repair Option

Afterwards, you can check the »Monitor« view for errors. If there are old errors, which are obsolete after the repair, these can be cleared by selecting a line and clicking »Completed« as shown in Figure 6.15.

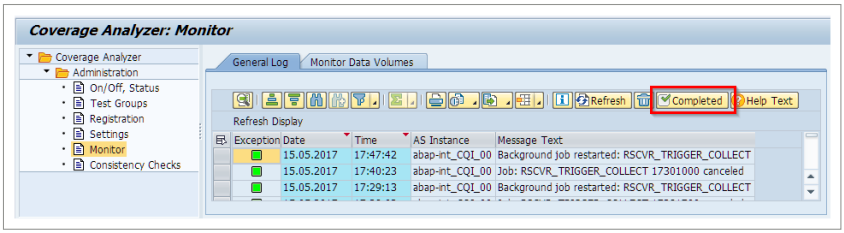


Figure 6.15. SCOV Monitor

In case of problems with the batch job `RSCVR_TRIGGER_COLLECT`, please make sure the user configured for the job does still exist and is not locked.

6.5.2.5 Using SCOV Lite for Teamscale and UPL

If you would like to use Teamscale to analyze coverage information from a system that is already using Usage Procedure Logging (UPL), there are a few things to consider. Otherwise, these tools may inter-

fere with each other and produce incorrect results. In essence, both the coverage-enabled Teamscale export and UPL use the same raw data created by SCOV Lite in the system table `COVRES`. However, when UPL extracts that data via the batch job `/SDF/UPL_PERIODIC_EXT_JOB` every 24 hours, it empties that table, whereas the Teamscale Connector does not.

This means the Teamscale export should run briefly before the UPL extraction job. In order to find out when that job is scheduled, you can either use transaction `SM36` in the SAP system or have a look at the file `job_status.csv` in one of the exported zip files. Afterwards, adjust the scheduling of the *incremental* code update in your SAP System Connection to run a few minutes before that time. You may still have multiple updates per day, as long as one of them always runs shortly before the UPL extraction job. Coverage data is contained in the incremental updates, but not in full code updates.

6.6 Integration of *Teamscale* in SAP Development Environment

The analysis results of *Teamscale* can be integrated in the SAP environment again. This will reduce switches between SAP and the web interface of *Teamscale*. Thus, the user will get most of the relevant information in his usual development environment. An integration is possible with the *ABAP Development Tools for Eclipse (ADT)*, with *SAP Code Inspector* and hence with the *ABAP Workbench*. Finally, the *SAP Transport Management System* can check for *Teamscale* findings on transport releases.

6.6.1 ABAP Development Tools for Eclipse

Teamscale can be integrated into Eclipse. Hence findings are shown directly in the IDE. This also works the *ABAP Development Tools for Eclipse (ADT)* and avoids unnecessary context switches between IDE and the analysis tool. For installation and configuration of the plug-in, see Section 4.2.1.

6.6.2 SAP Code Inspector and ABAP Workbench Integration

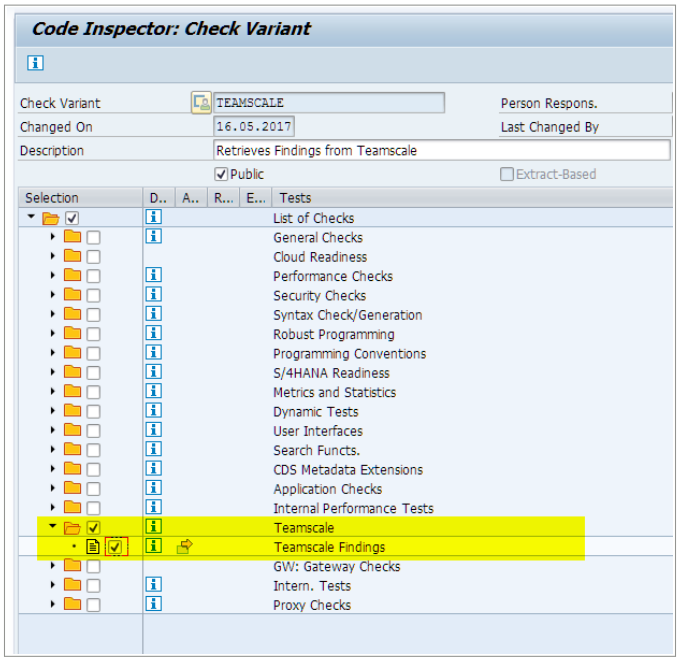


Figure 6.16. *Teamscale* Findings SCI Test

The findings in *Teamscale* can be also shown the *SAP Code Inspector (SCI)*, like the results of other Code Inspector checks. Thus, *Teamscale*

findings can be handled within the ABAP Workbench in the same way as any other Code Inspector messages, e. g. by using the check functions available in `SE80`.

To achieve this, the *Teamscale Connector for SAP NetWeaver® AS ABAP®* contains an implementation of a custom SAP Code Inspector test which allows to import findings *from Teamscale* to the SAP NetWeaver system . To be able to execute the respective Code Inspector test *Teamscale Findings*, the following configuration steps are required:

Test Activation. First, Code Inspector test *Teamscale Findings* must be activated: In transaction `SCI` select the action `Code Inspector` → `Management of` → `Tests` (or `Ctrl+Alt+F5`). There, mark the check boxes at the check classes `/CQSE/CL_CI_CATEGORY_TEAMSCALE` (category for *Teamscale*) and `/CQSE/CL_CI_TEST_TS_FINDINGS` (*Teamscale* findings test). Save your changes.

Definition of a Check Variant. Second, a Code Inspector check variant which contains the *Teamscale* findings test must be defined. Add a new or edit an existing check variant in transaction `SCI` and select the *Teamscale Findings* test, see Figure 6.16. Also the the required parameters of this test must be specified, as described in Figure 6.17.

As of *Teamscale Connector for SAP NetWeaver® AS ABAP®* v19.07, you may leave the Teamscale project id parameter empty. This is especially relevant when executing the test in a central ATC check system, which is typically connected to may development systems, and therefore would have to reference different Teamscale projects. If no project ID is provided, the *Teamscale Findings* test will query *Teamscale* (version 5.2.1 or later) for projects that have the respective

system ID configured in their project options. See Figure 4.90 on page 179 for how to do that.

Teamscale Findings Parameters

Teamscale server URL	URL of the *Teamscale* instance
Teamscale user name	Name of the *Teamscale* user which should be used to retrieve the findings
User's IDE access key	The IDE access key of this user (must be generated in the user preferences of the *Teamscale* Web UI)
Teamscale project id	The identifier of the project in *Teamscale* for which findings should be retrieved
Only new findings since (label)	Optional. If set, only findings which were introduced after the given baseline label are included (the baseline is managed in the **Findings** perspective of the *Teamscale* Web UI)
Only new findings since (date)	Optional. If set, only findings introduced after the specified date are included (this parameter will be ignored if the »Only new findings since (label)« parameter is already set)
Also finds. in changed code	If enabled and also one of the »Only new findings since« parameters is set not only new findings but also findings in code changed since the baseline are included

Figure 6.17. Parameters of the Teamscale Findings Test (Code Inspector)

Execute the Check Variant. After the variant is saved, it can be executed like any other Code Inspector variant, e. g. by transaction

`SCI` or from the ABAP Workbench (`SE80`) by the e. g. `Check` → `Code Inspector` / `ABAP Test Cockpit (ATC)` commands. Note that you must edit the `DEFAULT` Code Inspector variant if the *Teamscale* findings test should be executed when running these `Check` commands, unless you use command `ABAP Test Cockpit (ATC) with ...`.

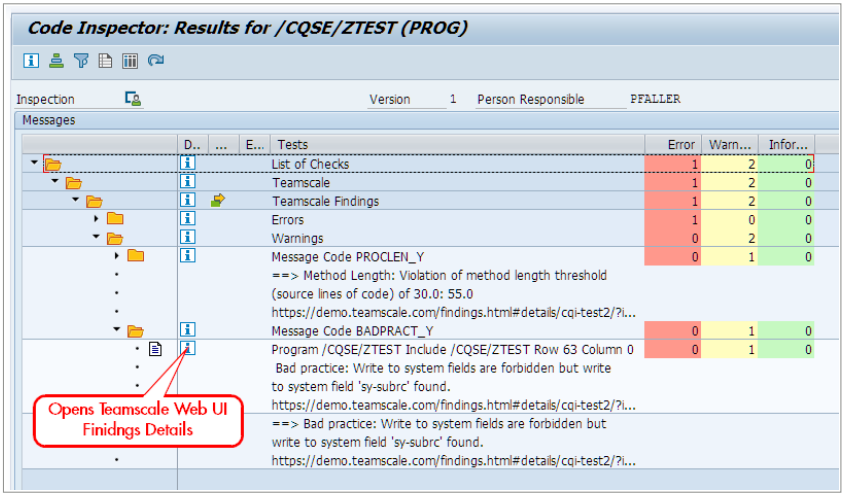


Figure 6.18. *Teamscale* Findings within SCI Result Browser

The results are displayed in the Code Inspector results browser as shown in Figure 6.18. By clicking on the info button at the single messages (leaves in the tree), the *Findings Details* view of the *Teamscale* web UI opens, where e. g. more information of the finding can be viewed, see Section 4.1.4.2. Here, the finding can be assigned to a task or put on the blacklist. (Note that the info buttons at other nodes than the leaves are without any meaningful function.)

6.6.3 Integration with SAP Transport Management System

As described in the section before, the SAP Code Inspector test *Teamscale Findings* allows to retrieve findings from *Teamscale* ans display these in the SAP system. Thus, *Teamscale* findings can be integrated in the SAP Transport Management System (TMS) like any other Code Inspector results. Basically two options for integration exist: Using the function of the *ABAP Test Cockpit (ATC)* for performing checks on transport release, or using a BAdI implementation which is included in the *Teamscale Connector for SAP NetWeaver® AS ABAP®*.

6.6.3.1 Using the ATC Check on Transport Request Release

If ABAP Test Cockpit (ATC) is available on your system, it can be used to perform Code Inspector checks on transport request release (note that is not possible to perform checks at task release via ATC, but this is possible with the BAdI implementation, see following Section 6.6.3.2).

To enable the ATC check on transport release, the following configuration is required:

- Define a Code Inspector variant which executes the test *Teamscale Findings* as described in the previous section.
- Ensure that checks on request release are globally activated: Transaction `SE03` (Transport Organizer Tools), item »Administration / Global Customizing«, setting for »Check Objects when Request Released«.
- In transaction `ATC`, item »ATC Administration / Setup / Basic Settings« configure the respective Code Inspector variant and

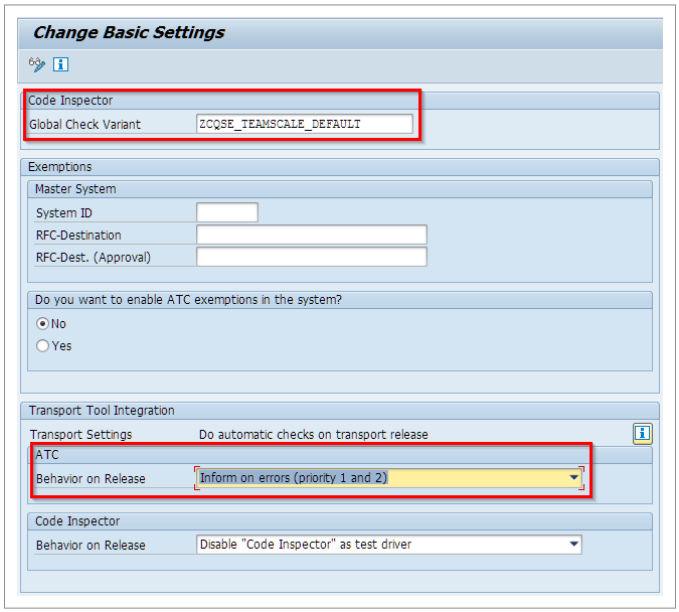


Figure 6.19. ATC Settings for Check on Transport Release

the setting »ATC: Behavior on Release«. Choose »Inform on Errors« or »Block on Errors«, see Figure 6.19. Save the settings.[7]

6.6.3.2 Using the BAdI Implementation for Checks on Transport Request or Task Release

The *Teamscale Connector for SAP NetWeaver® AS ABAP®* also includes an implementation of the BAdI `CTS_REQUEST_CHECK` to check transport requests and/or transport tasks on release. This is an alternative to the before mentioned use of ATC if ATC is not available or check on *task* release should be done.

[7] The setting for »Code Inspector: Behavior on Release « in the ATC basic settings refers to the execution of the `DEFAULT` Code Inspector variant only.

This again uses the SAP Code Inspector test »Teamscale Findings«. If enabled, findings are displayed to the user as in Figure 6.18 and he/she can decide if the transport should be canceled or released anyway. Thus, there is no automatic blocking of the release, it always needs confirmation by the user.

To enable the BAdI-based check, the following configuration must be done:

/CQSE/C_TSTRNCHK Customizing

CONFIG ID	Arbitrary configuration identifier.
CI VARIANT AT REQUEST RELEASE	Specifies the Code Inspector variant which should be executed on request release. This variant typically only includes the »Teamscale Findings« test. Leave empty if no check should be performed on request release.
CI VARIANT AT TASK RELEASE	Specifies the Code Inspector variant which should be executed on task release. This variant typically only includes the »Teamscale Findings« test. Leave empty if no check should be performed on task release.
CANCEL IN RFC ON FINDIGNS	If enabled, the release will be always canceled (blocked) when the release is performed via an RFC call, e. g. from ADT for Eclipse, and *Teamscale* findings exist. **This is currently an experimental feature, we recommend to leave it disabled.** When disabled, transports are always released when called from RFC.

Figure 6.20. Customizing Parameters for Check on Transport Release

- Define a Code Inspector variant which executes the »Teamscale Findings« check as described in the previous section.
- Add an entry to customizing table `/CQSE/C_TSTRNCHK`, e. g. by using transaction `SE16` and command `Create Entries` (`F5`). Figure 6.20 describes the customizing parameters. Save the new customizing entry.

The BAdI-based check is active as long as an SAP Code Inspector variant is set in the customizing table `/CQSE/C_TSTRNCHK`. The check has no effect on code which is not included in the *Teamscale* project which is configured in the specified Code Inspector variants.



Appendix

A Dashboard Widgets

Teamscale offers a large variety of highly configurable widgets to create personalized dashboards. These widgets and their configuration options are displayed in the following. Some options are the same for all widgets. Thus, they are described only once at the beginning.

A.1 Common Widget Configuration Options

The following table denotes configuration options that are commonly shared by many widgets. As an example, Figure App.21 shows the beginning of a configuration dialog containing some of these option.

Title	Each widget can be given its own custom title. Teamscale suggests a default title.
Path	The path can be used to select the scope for which the content of the widget should be displayed. The scope can either be an entire project or you can select any folder or component (if an architecture file has been modeled). You can also configure the branch if branch support is enabled and use the timetravel feature.
Trend	Many widgets can display a trend indicating the evolution of a metric. You can configure the time interval, e. g. if the trend should be considered for the last five days.
Zooming	Allows you to zoom into the chart.
Hide Y-Axes	Removes labeling of the y-axes.
Baselines	Shows the baselines of the underlying project as vertical blue lines in the chart.
Action menu	Adds a drop down menu with option to view a selected zoom range in the Delta perspective and to download the trend as csv.

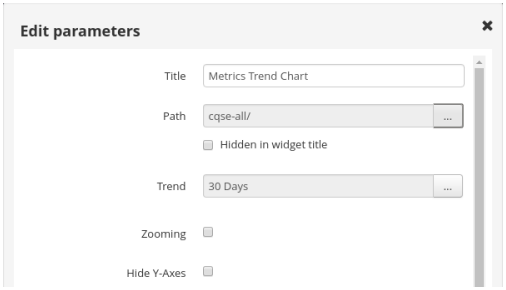


Figure App.21. Excerpt of Widget Configuration Dialog

A.2 Assessment Pie Chart Widget

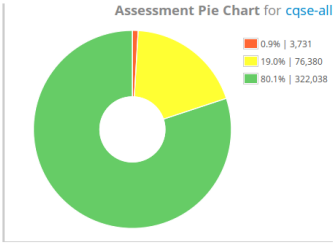


Figure App.22. Assessment Pie Chart Widget

Purpose	Shows an assessment metric as a pie chart.
Metrics	Lets you choose from the available assessment metrics. The thresholds of those metrics are configured in the analysis profile, not in this widget.
Legend	Displays a legend for the widget.
Absolute values	If checked, absolute values are shown. For structure metrics, these absolute values are LOC or SLOC depending on the analysis profile. (For the comment completeness, these values are the number of (un)documented code entities.) If unchecked, only relative percentage values of the assessment metric are shown.
Assessment thresholds	Shows target threshold lines. Should be of format »0.05,0.25«, for example.

A.3 Assessment Trend Chart Widget

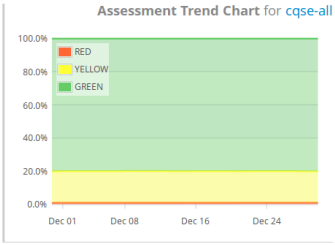


Figure App.23. Assessment Trend Chart Widget

Purpose	Shows the evolution of an assessment metric over time.
Assessment	Lets you choose from the available assessment metrics as configured in your analysis profile.
Show ratios	Displays a legend.

A.4 Code City Widget



Figure App.24. Code City Widget

Purpose	Shows the system as a code city.
Area Metric	Metric used to determine width and depth of a code city skyscraper.
Height Metric	Metric used to determine the height of a code city skyscraper.
Color Metric	Metric used to determine the color of a code city skyscraper; can be different than the height metric.
Color	Color used to color the skyscrapers. When using an assessment metric as color metric, the assessment colors red, yellow and green are always used.
Included files	Only files matching this regular pattern are displayed in the code city. The regular pattern is provided as a Java Regex. To include only Java files, enter »`.*\.java`«. You can add multiple include patterns, too, by clicking on `+`. This will provide you additional input fields.
Excluded files	Files matching this regular pattern will be excluded from the code city.

A.5 Commit Chart Widget

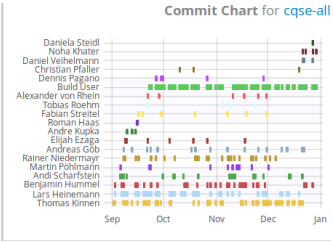


Figure App.25. Commit Chart Widget

Purpose	Shows the commit frequency of the development team over time.
Min. commits per developer	Number of commits a developer must have made at least in order to appear in the chart.
Order	Indicates ordering in which the authors are sorted. You can either select by time, by number of commits or alphabetically by the account name of the authors.

A.6 External Content Widget

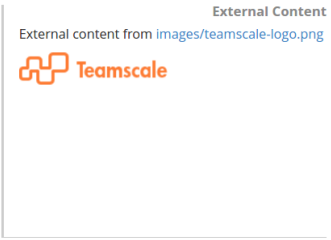


Figure App.26. External Content

Purpose	Provides a link to external content and embeds it as an iframe.
URL	Url for the external content.
Allow scrollbars	Allows scrolling within the widget.

A.7 Findings Churn Widget

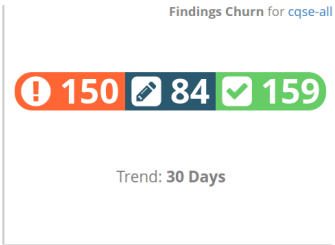


Figure App.27. Findings Churn Widget

Purpose	Shows the findings churn within the configurable time interval. Red indicates added findings, blue indicates findings in modified code (based on quality goal *improving*, see Section 1.2.2), and green indicates removed findings.
Show trend text	Indicates if a label with the trend should be displayed within the widget.
Filters	Allows to select the findings to be shown in this widget based on, for example, finding group and finding category.

A.8 Findings Summary Bar Chart Widget



Figure App.28. Findings Summary Bar Chart Widget

Purpose	Provides a bar chart as summary of all finding categories. Yellow and red indicates the severity of the findings as configured in the analysis profile.
Findings since	Selects a baseline for the findings. Only findings introduced after the baseline are shown.
Assessment filter	Choose if all, only red or only yellow findings should be displayed.
Show timespan text	Choose whether the timespan since the baseline should be displayed.
Selected categories	Select which finding categories should be considered in this widget. The available categories depend on the underlying analysis profile.
Detail level	Choose for which level a bar should be created; you can pick »only categories«, »categories and groups«, »categories, groups, and rules« or »groups and rules«.
Horizontal orientation	Indicates whether bar chart should be oriented horizontally or vertically.
Sort by findings count	Indicates whether chart should be sorted by number of findings or by category names.

A.9 Findings Summary Table Widget

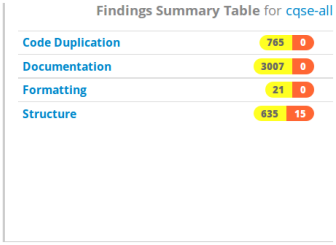


Figure App.29. Findings Summary Table Widget

Purpose	Provides a table with a summary of all finding categories. Yellow and red indicates the severity of the findings as configured in the analysis profile.
...	Please refer to the Findings Summary Bar Chart Widget in Appendix A.8 which has the same configuration options.

A.10 Folder Distribution Pie Chart Widget

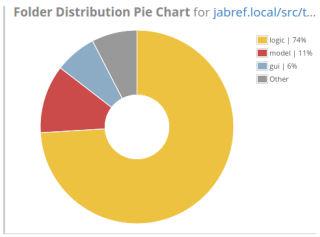


Figure App.30. Folder Distribution Pie Chart Widget

Purpose	Shows metric distribution for subfolders and files (or subcomponents and files) inside a folder (component). For example, you can use this pie chart to see the size distribution in lines of code of a folder's content.
Metric	Select the metric based on which the subfolders or subcomponents are compared.
Legend	Add a legend to the widget.
Labels with % and Values	Creates the legend both with relative as well as with absolute values.
Combine slices below %	Combine subfolders/files into »other« which make up less than the specified amount of % of the overall distribution.

A.11 Issue Table Widget

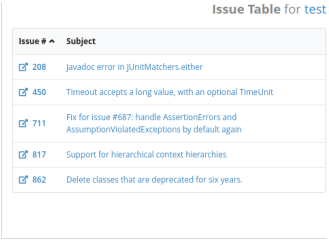


Figure App.31. Issue Table Widget

Purpose	Shows all issues matching a given query.
Enter Query	Choose whether to enter query or select issue metric.
Issue Metric (Saved Query)	Select previously saved issue metric to find matching issues. Selected query will be shown in the field Query.
Query	Enter query to find matching issues.

A.12 Label Widget

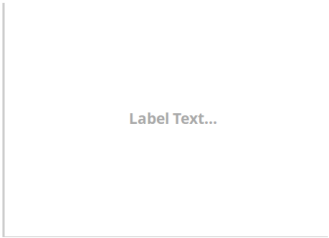
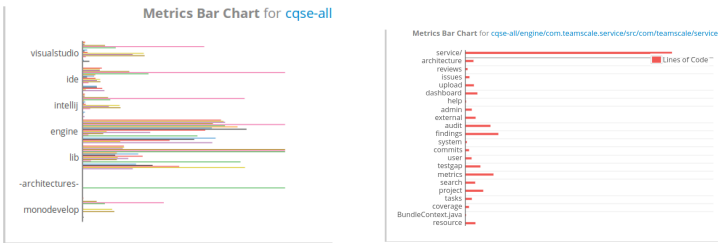


Figure App.32. Label Widget

Purpose	Adds a label to the dashboard.
Text size	Choose between small, medium and large.
Text color	Pick the label color.
Bold text	Uses a bold font instead of regular.
Wrap text	Uses more than one line if necessary.

A.13 Metric Bar Chart Widget



(a) Displaying multiple metrics. (b) Comparing lines of code.

Figure App.33. Metric Bar Chart Widget

Purpose	Displays metrics for folders or components as bar chart. Can be used, for example, to compare subfolders by size (LOC).
Metrics	Select the metrics you want to display as bars. Available are only numeric metrics. As default, all are selected.
Metric Threshold Configuration	Select your threshold configuration profile.
Hide Threshold Configuration Markers	If unchecked, thresholds from the selected threshold configuration will be shown as markers in the bar charts.

Additional Marker Threshold Configuration	For each numeric metric, you can enter an additional threshold. For example, for the clone coverage, enter »0.05« if you want to set a 5% threshold.
Metric bar colors	Choose the colors of the bars. Colors are assigned to the metrics based on the ordering of the metrics.
Horizontal orientation	Indicates whether bar chart should be oriented horizontally or vertically.
Include parent node	Include the folder/component you selected as path for the widget.

A.14 Metric Change Table Widget

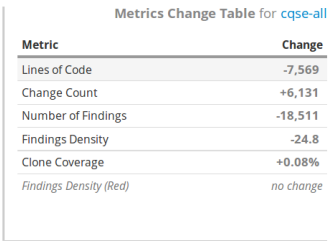


Figure App.34. Metric Change Table Widget

Purpose	Shows metric changes within a configurable time interval in a table.
Metrics	Select the metrics from your analysis profile to be shown in the widget.

A.15 Metric Distribution Pie Chart Widget

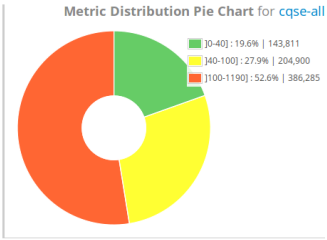


Figure App.35. Metric Distribution Pie Chart Widget

Purpose	Shows how code distributes over user-defined intervals of a metric. Examplary questions that can be answered with this widget are: How much of my code is in overly long files? How much of my code is in files with very high clone coverage? How much of my code is in files with sufficient test coverage?
Primary Metric	Based on the values of the primary metrics, files are classified into different categories. The categories are described by the boundaries. If you select clone coverage as a primary metric, you can display, for example, how much of your code is in files with very high clone coverage.
Boundaries	Sets the thresholds for the primary metric. Based from this thresholds, intervals and corresponding categories are derived. For example, entering two values »0.1 , 0.2« defines three intervals: $[0, 0.1],]0.1, 0.2],]0.2, \infty]$, which could be interpreted as »low«, »medium« and »high« clone coverage.
Colors	For each category defined by the boundaries, you can define a color. Colors are assigned in the same order as the interval/categories are derived.

Aggregation Metric	After each file is classified based on the boundaries of its primary metric, the aggregation metric is used to calculate the distribution. Choosing lines of code, for example, will give you the percentage of your code (in lines of code) that resides within files with a low/medium/high clone coverage.
Legend	Adds a legend.
Boundaries in Legend	Adds a legend and the configured boundaries.
Show absolute values	Shows also the absolute values of the aggregation metric in addition to the relative distribution.

A.16 Metric Distribution Table Widget

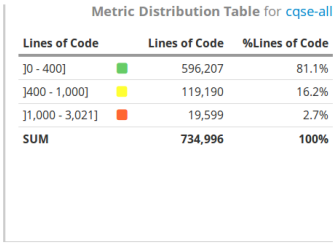


Figure App.36. Metric Distribution Table Widget

Purpose	As the previous *Metric Distribution Pie Chart* widget, this widget shows how code distributes over user-defined intervals of a metric. It only differs by showing the result in a table rather than a pie chart.
...	Please refer to the configuration of the Metric Distribution Pie Chart Widget in Appendix A.15.

A.17 Metric File Distribution Widget

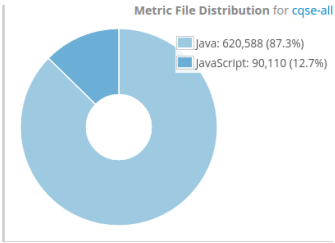
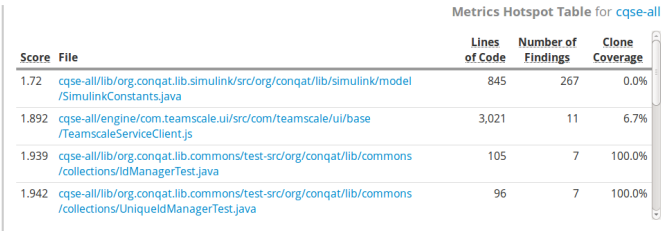


Figure App.37. Metric File Distribution Widget

Purpose	Shows how a size metric distributes over files matching configurable regular expressions. Can be used, for example, to display how the amount of code (in LOC) distributes over Java and JavaScript files.
Metric	Choose between LOC, SLOC and Files (Number of Files) to measure the amount of code matching one of your regular expressions.
File regexes	Enter the regular expression patterns using Java Regex syntax for which you want to know the code distribution. Examples can be »`Java:.*\.java$`« and »`JavaScript:.*\.js$`«, assigning the labels »Java« and »JavaScript « to the two categories of java and javascript files.
Colors	Specify a color for each category of files. Colors are assigned in the same order as the categories are defined.
Legend	Include a legend to the widget.

A.18 Metric Hotspot Table Widget



Metrics Hotspot Table for cqse-all

Score	File	Lines of Code	Number of Findings	Clone Coverage
1.72	cqse-all/lib/org.conqat.lib.simulink/src/org/conqat/lib/simulink/model/SimulinkConstants.java	845	267	0.0%
1.892	cqse-all/engine/com.teamscale.ui/src/com/teamscale/ui/base/TeamscaleServiceClient.js	3,021	11	6.7%
1.939	cqse-all/lib/org.conqat.lib.commons/test-src/org/conqat/lib/commons/collections/IdManagerTest.java	105	7	100.0%
1.942	cqse-all/lib/org.conqat.lib.commons/test-src/org/conqat/lib/commons/collections/UniqueIdManagerTest.java	96	7	100.0%

Figure App.38. Metric Hotspot Table Widget

Purpose	Shows a table with hotspots for the selected metrics. Given the selected metrics, an internal score is calculated to find and rank the hotspots. The smaller the score, the »hotter« the hotspot.
Metrics	Select one ore multiple numeric metrics.
Number of hotspots	Enter how many hotspots (files) should be displayed in the table.
Score cutoff	Files with this score or higher will not be shown in the table.
Show score	Select whether or not to show the score in the table.

A.19 Metrics Table Widget

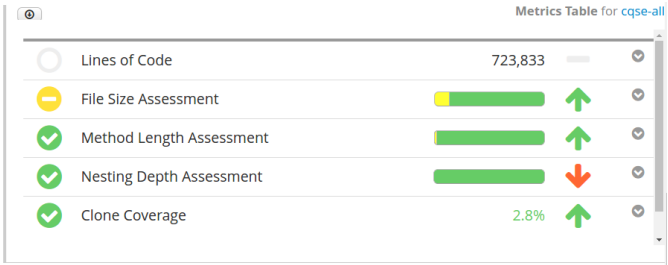
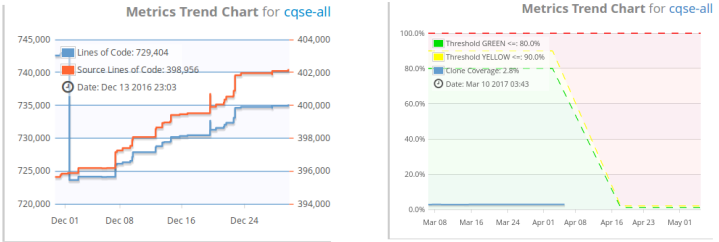


Figure App.39. Metrics Table Widget

Purpose	Shows a table with metrics, along with their trend and assessment based on the underlying threshold configuration.
Metric Threshold Configuration	Select your threshold configuration.
Metrics	Choose the metrics to be shown in the table. Only metrics configured in the threshold configuration are available.

Flatten Single Group	Hides the group title if selected threshold configuration contains only one single group.
Show actions	Shows an action button for each metric to view the history trend of the metric as well as its treemap.
Show values	If checked, shows the values of the metric.
Show icons	If checked, shows the assessment icons on the left (red, yellow or green circle)
Show trend	Shows the trend icons on the right (red or green vertical arrow)
Number of days in future	This option can be used if »Show actions« is checked. The actions provide the opportunity to view the history trend of a metric in a new pop-up dialog. In this dialog, the trend chart can include a certain number of days in future. This can be used for a burn-down chart, when certain target thresholds are set for a metric in the future.

A.20 Metrics Trend Chart Widget



(a) LOC and SLOC Trends. (b) Treshold Configuration Trends.

Figure App.40. Metric Bar Chart Widget

Purpose	Shows the trend chart for one or multiple metrics over time.
Colors	Choose the colors for each metric. Colors are assigned in the order as the metrics are sorted.
Show Thresholds	Shows the thresholds. This works only if you selected exactly one single metric and for this metric, there is a corresponding threshold configured in the selected threshold configuration.
Metrics	Select one ore multiple numeric metrics.
Stacked	Stack all metrics on top of each other. Needs »Include 0« and »One scale« to be enabled.
Include 0	Include 0 in the y-Axis.
One scale	If checked, all metrics are shown on the same scale with just one axis. Otherwise multiple y-Axis with different scales are used.

Number of days in future	Includes future days in the chart with the main purpose to see thresholds configured for the future. Enabling this feature works only, if you selected exactly one single metric and for this metric, there is a corresponding threshold configured in the selected threshold configuration. The right screenshot in Figure App.40 shows an example of a trend that also displays days in the future. This can be seen as the blue line indicating the clone coverage stops on April, 5th. The threshold configurations in the example were set up such that they decline over time.

A.21 Numeric Metric Value Widget

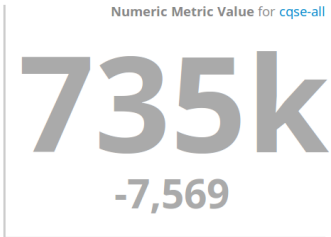


Figure App.41. Numeric Metric Value Widget

Purpose	Shows the value of a numeric metric.
Boundaries	Boundaries are a light-weight version of a threshold configuration. They allow you to configure thresholds to assess a metric. For example for the metric of clone coverage, you can enter the values »0.05,0.10 «. Using the default colors of green, yellow and red, this configuration would display clone coverage values less than 5% with green color, between 5 and 10% with yellow color and above 10% with red color.
Colors	The input for boundaries defines a certain number of intervals. For example, entering two numbers results in three intervals, as explained above. For each interval, you have to pick a color. The color is used as font color of the label when the metric value resides within the corresponding interval.

A.22 Ownership Treemap Widget

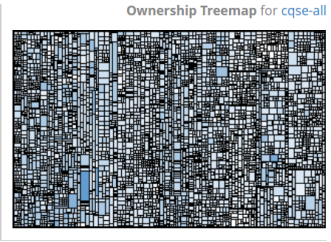


Figure App.42. Ownership Treemap Widget

Purpose	Shows code ownership information for each file in a treemap, i. e. how many developers have changed the corresponding file. The widget reveals, e. g. code areas that only one or few developers are familiar with.
Exclude merge commits	Exclude merge commits to calculate ownership information.
Exclude Teamscale import	Excludes the initial *Teamscale* import commit to calculate ownership information.
Ownership color	Pick the color to display ownership information. The darker the color, the more developers have changed the corresponding file. Areas, that only one developer is familiar with, are displayed in white.

A.23 Single Metric Trend Widget

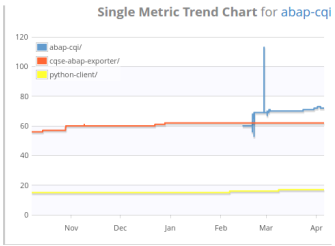


Figure App.43. Single Metric Trend Widget

Purpose	Shows the trend chart for a single metric over time. It can show, however, the same metric for multiple projects.
Additional paths	In addition to the normal path which every widget has, you can add additional paths for this widget. For each configured path, the selected metric will be shown as a separate trend line. Hence, you can, for example, show the lines of code for three different projects in one chart and compare their evolution.
...	For the remaining configuration options, please refer to the very similar *Metrics Trend Chart* widget described in Appendix A.20.

A.24 Task Summary Pie Chart Widget

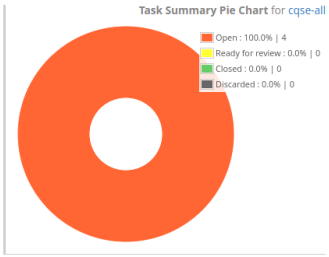


Figure App.44. Task Summary Pie Chart Widget

Purpose	Shows a summary of *Teamscale* tasks by giving a distribution of open, ready for review and closed tasks.
Legend	Include a legend.
Show absolute values	Shows the absolute number of tasks in the legend in addition to the percentage values.
Show discarded tasks	Includes discarded tasks as separate category in the summary.

A.25 Test Gap Overview Widget

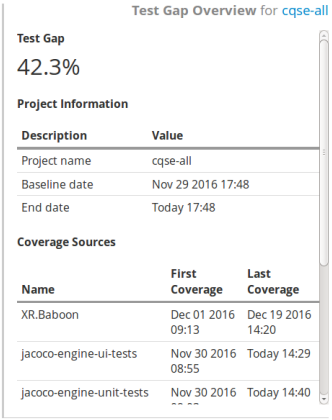


Figure App.45. Test Gap Overview Widget

Purpose	Gives an overview of the results from the test gap analysis.
End date	The point in time until which changes are considered.
Cross-annotate execution	Use additional coverage information from more projects than specified in the given path. This might be useful in setups where you have different testing environments for different branches of the same system.
Coverage sources	Selects which coverage sources should be considered.

A.26 Test Gap Treemap Widget

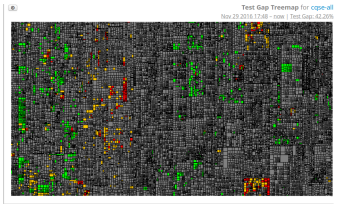
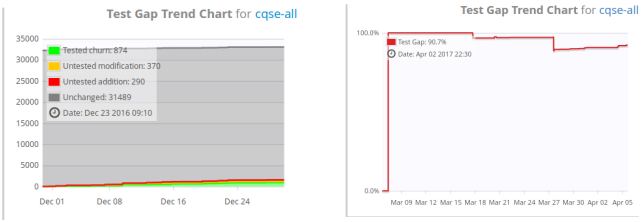


Figure App.46. Test Gap Treemap Widget

Purpose	Shows the results from the test gap analysis in a treemap, in which every rectangle represents a single method.
...	Some configuration options have already been described in the *Test Gap Overview* widget in Appendix A.25.
Execution only	Shows only the execution information from the test runs (without combining it with change information).
Outline depth	Determines how many level of packages should be outlined. Outlined packages will be surrounded by a thicker line.
Outline color	Color used to outline packages.
Issue id	Only methods changed during the implementation of a specific issue are shown in the treemap. If set, baseline and end date parameters will be ignored.
Focus on changes	Shows only changed methods, i. e. hides all gray areas from the treemap. This is particularly helpful in case there are only few changes in a large system, which would otherwise make small changed methods hard to identify.

A.27 Test Gap Trend Chart Widget



(a) Complete trend. (b) Only percent trend.

Figure App.47. Test Gap Trend Chart Widget

Purpose	Shows test gap analysis results as a trend chart.
...	Some configuration options have already been described in the *Test Gap Overview* widget in Appendix A.25.
Focus on changes	Rescale the y-Axis to eliminate the big gray and uninformative areas to focus on where the trend actually changes.
Show only percent trend	Shows the test gap analysis information as one single numeric metric (»test gaps«) and displays its trend. This option cannot be used, if the previous one is enabled and vice-versa.
Execution only	Show the execution information only (without combining it with change information).

A.28 Treemap Widget

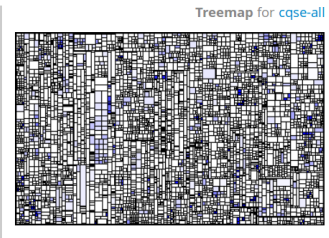


Figure App.48. Treemap Widget

Purpose	Shows a metric with a treemap in which every rectangle represents a file.
Area Metric	Metric used to determine the area size of a rectangle representing a file. Only size metrics are available. If the files metric is chosen, all rectangles will have the same (default) size.
Color Metric	Metric used to color a rectangle. The higher the value for a file, the darker the color for the corresponding rectangle.
Color	Color for the treemap.
Included files	Only files matching the provided regular pattern are displayed in the code city. To include only Java files, use the Java Regex syntax and enter »`.* \.java`«. You can add multiple include patterns, too.

Excluded files	Files matching this regular pattern will be excluded from the treemap.
Enable cushion	Enables a cushion effect on the treemap.
Minimum value	Only for numeric metrics. All values equal or below this value are colored with white. Use -1 to automatically use the lowest metric value as minimum value.
Maximum value	Only for numeric metrics. All values greater or equal to this value are colored with the specified color. Use -1 to automatically use the largest metric value as minimum value. You might want to use this option if you have a single but very large value – an outlier – in your data set which almost dismisses the gradient in the coloring in the treemap.

B Repository Connector Options

To create a project, *Teamscale* needs to be connected to your version control system. To this end, it offers several different repository connectors. We describe the options for each repository connector. Setting up the other connectors to source code repositories, issue trackers, and review tools is very similar to the git connector setup. Many connectors share general options such as a **polling interval** and therefore, we only describe the options (if there are any) for first connector that uses it. If there are no additional options in a connector, we don't document the connector separately.

GIT Configuration Options

Branch Name	The name of the branch to import.
Account	Account credentials that *Teamscale* will use to access the repository. Clicking on the + button opens a credentials creation dialog. The **Account Name** setting in the dialog is *Teamscale*-internal (does not need to match anything in git). When using a file URL in the account, make sure that the corresponding repository is *bare*, i. e. has been cloned with the `--bare` option.
Path suffix	Should *Teamscale* analyze a only a sub directory of the repository?
Repository identifier	*Teamscale* will use this identifier to distinguish different repositories in the same project. (*Teamscale*-internal only)
Included file names	One ore more ant patterns separated by comma or newline to restrict the analysis on a subset of files.
Excluded file names	One ore more ant patterns separated by comma or newline to exclude from the analysis.
Include submodules	Whether to also analyze code in submodules.
Submodule recursion depth	The maximum depths of nested submodules that will be analyzed. (Has no effect if **Include submodules** is not selected.)

GIT Configuration Options (continued)

SSH Private Key ID	The ID of a SSH private key stored in *Teamscale* which will be used when connecting to a GIT **Account** over SSH. You can manage all your GIT private keys on the **Settings** page of the **Admin** perspective, in the **Git** section. When using a GIT private key, the **Account** used by the connector has to be configured as follows: The **URL** has to contain the user name (e.g. *ssh://username@gitserver/path/to/repo.git*), the password entered must be the password for the private key and the username may be left empty.
Enable branch analysis	This checkbox activates our new branch support feature. If you select it, *Teamscale* will analyze all branches of the repository. If branch analysis is enabled, the **Branch Name** option specifies the main branch (usually »master«).
Included branches	Explicitly control which branches will be analyzed. (Has no effect if **Enable branch analysis** is not selected.)
Excluded branches	Explicitly state which branches to exclude. Note that *Teamscale* will create so called *anonymous branches* for all historic branches whose name could not be reconstructed. By default these are filtered using the exclude pattern **_anon.***
Start revision	The first revision that will be analyzed. You can enter a date (which selects the first revision after the date) or a revision hash.
Content exclude	A comma-separated list of regular expressions. *Teamscale* will exclude all files that match one of these filters.
Polling interval	How often should we query the repository and update our findings? (Unit is seconds)

File System Connector Options

Input directory	The path of the source-code directory.

Options for a Connector to a Team Foundation Server

Path suffix	A common sub directory on the TFS Server appended to the account URI. Leave empty if the whole directory structure is relevant. This is the *root directory* when looking for branches if branching is enabled.
Branch path suffix	The sub path (relative to the branches) that should be analyzed. Leave empty to analyze all code within each branch.
Branch lookup paths	The paths in the repository separated by commas where *Teamscale* will search for branches within the root directory defined by the Path suffix and account URI. If set, *Teamscale* only looks for branches on the top level of each branch lookup path. If empty only the root directory is searched for branches on the top level.
Included/Excluded branches	Included and excluded branches must contain the full path to the branch relative to the root directory. This means the Branch lookup paths appended with the contained branch.

B.1 Subversion (SVN) branch analysis

Subversion by itself does not support *real* branches, but rather uses its *cheap copy* operation and folder naming conventions for managing logical branches. For *Teamscale*'s branch support to recognize the branches, we assume the standard layout[8], where a project has top-level folders *trunk* and *branches*. Branches are created by copying the trunk folder (or another branch) as a new folder within *branches*. When using this layout, you should enable branch support and configure the URL of the SVN connector to point to the project folder, i.e., the one containing *trunk* and *branches*.

Note that *Teamscale* assumes all top-level folders within *branches* to be actual branches. Using further folder hierarchies to organize your branches is currently not supported. More specifically, a branch name may never contain a slash. Also, as *Teamscale* analyzes your history, even branches that have been deleted from the *branches* folder will be found and potentially be analyzed, as they are part of the recorded history. If you want to prevent this, either set a *start date* after the time the branch was deleted or exclude the corresponding branches using the *branch exclude patterns*.

B.2 Team Foundation Server Credentials

When connecting to a Team Foundation Server via a repository connector, you may use either the account credentials of a domain user or a personal access token (if these are enabled in your TFS).

When configuring an account credential for an access token, enter the TFS access token in the **Password** field. The **Username** is optional in this case. By convention, you may use the name of the access token.

[8] http://svnbook.red-bean.com/en/1.7/svn.branchmerge.using.html

Example: Configuring a TFS connector with branches The following example shows how to configure a repository connector for TFS. The example TFS project is called *TFS Example* which contains a folder called *Code* with all branches. See Figure App.49 for the full folder and branch structure of the TFS project. For our example we'd like to have all branches from the *Branches* folder except for the branch *HackFix*, the release branches from the *Release* folder and the *Main* branch included. Furthermore, only code located in the *src* directory should be included in the analysis. The connector configuration parameter values are listed in table *Configuration parameter values of the TFS example*. The Branch lookup paths must include the root directory containing the *Main* branch via the folder represented by „.". Note that the Default branch name value is case sensitive and leads to an extra empty branch if it doesn't match an existing branch.

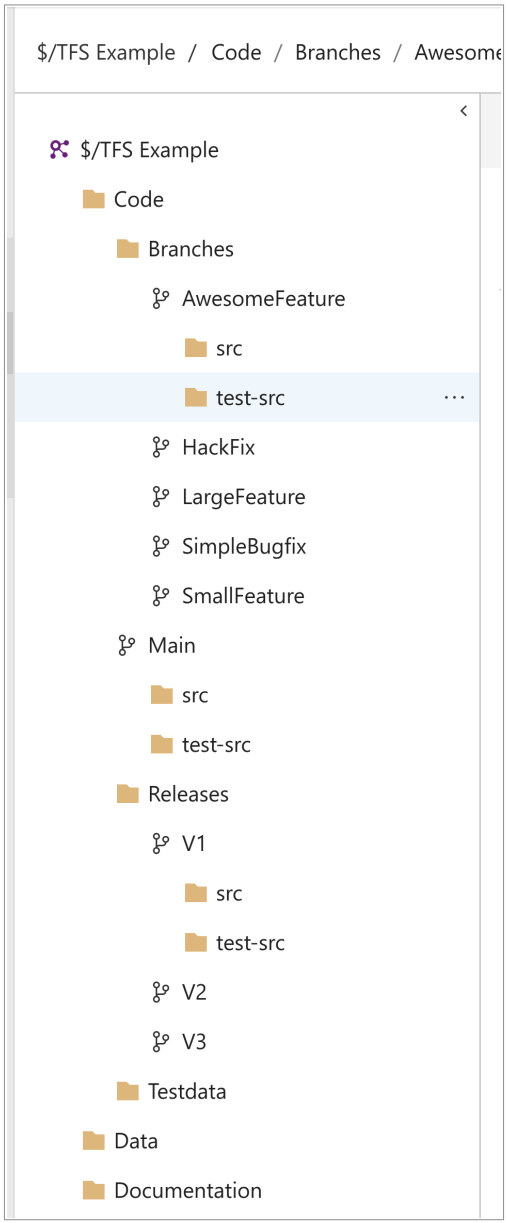


Figure App.49. Example folder structure on TFS server with branches

Configuration parameter values of the TFS example

Path suffix	$TFS Example/Code
Branch path suffix	src
Branch lookup paths	.,Releases,Branches
Included branches	Main
	Releases/.*
	Branches/.*
Excluded branches	Branches/HackFix
Default branch name	Main

C Issue Tracker Connector Options

Options for all Connectors

Issue ID pattern	A regular expression that matches the issue ID inside commit messages. Must contain at least one capturing group. The first capturing group of the regular expression must match the entire issue ID as it is used by the issue tracker. For example, for JIRA a valid pattern might be `(JIRA-\d+)`. If the regular expression can match alternate spellings that the issue tracker does not consider valid issue IDs, then these spellings must be normalized using an **Issue ID transformation** (see below).
Projects	A comma-separated list of project names from the issue tracker. Only issues from these projects will be imported
Issue ID transformation (expert option)	A regular expression matching all spellings of an issue ID that may occur inside commit messages together with a replacement expression that converts the alternate spellings into the normalized spelling required by the issue tracker. For example, `JIRA[-_](\d+) -> JIRA-$1`

Options For Team Foundation Server Work Item Tracker	
Areas	A list of full area paths to extract work items from.
Include Sub-Areas	If checked, issues from sub-areas are retrieved as well, otherwise just issues exactly matching one of the area paths are retrieved.

Options for a Connector to RTC/Jazz	
Field filter expression	An expression used to filter issues that should be imported. Must conform to RTC/Jazz's Reportable REST API specification.

C.1 Team Foundation Server Work Items

When connecting to a Team Foundation Server via an issue connector, please verify whether the *Basic Authentication* option is enabled in your IIS or not. Depending on this option, you may need to use different credentials than you used for your repository connector when connecting to the TFS to retrieve work items.

Basic Authentication enabled: You must connect with the credentials (username and password) of a TFS user account. Connecting with personal access tokens will not work. This may either be a domain user or a TFS user with the necessary rights to read work items.

Basic Authentication disabled: You must connect with a personal access token. Connecting with a user account (username + password) will not work. Put the personal access token in the **Password** field in the Teamscale UI. Configuring a **Username** in Teamscale is optional in this case. By convention, you may use the name of the TFS access token.

D Entity Selection Predicates

Teamscale provides special predicates for the selection of code entities. For example, you can use these predicates to define methods that require a documentation comment in the analysis profile.

An entity selection predicate consists of primitives that can be connected using negation »!«, conjunction »&«, disjunction »|«, and parentheses. The primitives are listed in the following configuration boxes. Obviously, not all predicates are applicable to all programming languages.

An example that selects all public simple getters and setters would be

```
public & (simpleGetter | simpleSetter)
```

A good starting point are the following examples

Entity Selection Expressions for Comment Completeness

Java	`public & (type | method | attribute) & !(simpleGetter | simpleSetter | annotated(Override) | override)`
C#	`public & (type | method | property) & !(simpleGetter | simpleSetter | annotated(Override) | override)`
C++ (Comments in Headers)	`type | (public & (attribute | (method & declaration)))`
C++ (Comments at Implementation)	`type | (public & (attribute | (method & !declaration)))`

The following boxes provide an overview of the available predicates.

Selection Predicate Primitives (Part 1)

language(name)	Selects entities in files with the given language. If the given language name is invalid, no entities are matched. Valid names are for example JAVA and CPP.
module	Selects namespaces and modules.
type	Selects types, such as classes or structs.
method	Selects methods and functions.
declaration	Selects declarations of types and methods.
internal	Selects entities that are internal (C# keyword).
attribute	Selects attributes, fields and global variables.
property	Selects all properties: defined attributes that have only method children.
statement	Selects all statements.
meta	Selects all meta information (defines, annotations, etc.).
public	Matches all public entities as well as methods inside interfaces, which are considered public in most languages.
protected	Matches all protected entities.
override	Matches all entities marked with override keyword.
private	Matches all private entities.

Selection Predicate Primitives (Part 2)

default	Matches all entities with default visibility.
final	Matches all final entities.
static	Matches all static entities.
primitive	Matches all primitive entities (i.e. those without children).
subtype(name)	Matches entities by their subtype. The name parameter could be for example *if* for if statements.
typeSubtype(name)	Matches entities by their parent type's subtype (e.g. to select all members of interfaces).
name(name)	Matches entities by their name. For example a method name.
nameRegex(regex)	Matches entities by their name using regular expression.
fileRegex(regex)	Matches entities by their containing file's unified path using regular expression, e.g. »fileRegex(.*\.ts)« matches all files with the extension ».ts«.
annotated(name)	Matches all entities that are annotated with an annotation of given name (excluding the »@« sign).
simpleGetter	Matches simple getters, i. e. methods starting with »get« or »is« and containing at most one statement.
simpleSetter	Matches simple setters, i. e. methods starting with »set« and containing at most one statement.
simpleMethod(prefix)	Matches simple methods whose name starts with the given prefix. A method is simple, if it contains as most one statement.

Selection Predicate Primitives (Part 3)

export	Matches all entities with ES6's export visibility.
typeexported	Matches all entities (e.g. methods, attributes) within types that have ES6's export visibility.

E Custom Checks

Custom checks can be used to easily define rules checked by *Teamscale*. The *Teamscale* distribution already comes with about 100 custom checks for different languages. Users can write their own custom checks as documented in Section 4.3.7. Each custom check can be enabled or disabled separately for each analysis profile that contains one of the languages supported by the custom check. A check (and its options) can be configured in the configuration dialog of its analysis group in the analysis profile configuration (Figure 4.101).

The following sections list the available custom checks, ordered by their supported languages and by the analysis groups in which they can be configured. The analysis groups are: Unused code, Test Conventions, Exception Handling, Correctness, Performance, Security, Bad practice, Javascript (built-in), and Cross-Browser Compatibility.

E.1 Custom Checks for ABAP

E.1.1 Bad practice

Assert commands should have ID

Assert should always have an ID.

No automated code manipulation

Programs should not create, delete, or change coding.

No static breakpoints in code

Coded breakpoints should not be used.

No call to external form routines

External form routines should not be called.

No complex logic in event block

Complicated statement blocks should not be implemented directly in event blocks but in appropriate modularization units.

No complex logic in macro

A macro should not contain complex logic.

No data declarations in functional code blocks

Data declarations should not be placed in functional code blocks.

No database operation in macro

A macro should not contain database operations.

No non-class-based exceptions via RAISE statement

Non-class-based exceptions should not be thrown using RAISE statement. Use 'MESSAGE Ennn with <…> RAISING <exception>' or class based exceptions via 'RAISE EXCEPTION <…>' instead.

No use of task handler functions

Task handler functions (FuGr THFB, THFB2, THFB_ALV, THSCHED, THTX) should not be used.

No write access to SAP standard tables

Write accesses to SAP standard tables by customized program components should not occur.

No write access to system fields

System fields (Sy-xxxx, fields of structure SYST) should only be read (as they are set by the runtime environment).

Unescaped WHERE ... LIKE

SELECTs with non-literal WHERE …LIKE should always specify ESCAPE to avoid unintended wildcards.

E.1.2 Security

Suspicious call of function modules RFC_REMOTE_EXEC or RFC_REMOTE_PIPE

The function modules RFC_REMOTE_EXEC and RFC_REMOTE_PIPE allow to execute operating system commands. In custom code, these function modules should not be used. The SXPG framework should be used instead.

Suspicious usage of CALL system function

The internal statement CALL *cfunc* executes a system function *cfunc*. Some system functions, e.g. 'SYSTEM' or 'ThWpInfo' allow to execute any command of the operating system and should be avoided. The SXPG framework should be used instead.

Control flow depending on system variable (SY-...)

A system variable, e.g., SY-UNAME (user name of current user) or SY-SYSID (SAP system id) is misused to control the program flow. This should should be avoided, since it may lead to different behavior depending on logon data or platform.

Critical Statement

Critical statements should be avoided since the danger to introduce security flaws is high. Usually these statements should be never used in production code. These statements might be necessary e.g., for code generators or other development related activities, but then the containing objects should be placed in packages which will not be transported to the production system. Usages of critical statements should properly document why the statement is needed.

The statements which are regarded as critical can be specified in the analysis profile.

Statements which are considered as critical by default are:

- EDITOR-CALL - allows to modify ABAP programs regardless if development authorization is granted to the executing user
- GENERATE SUBROUTINE POOL - generates FORM routines at runtime, security critical code could be obfuscated in this way. A severe vulnerability exists if the code is generated from an external input.
- INSERT REPORT - generates programs (REPORTS) at runtime, as with GENERATE SUBROUTINE POOL, security critical code could be obfuscated in this way. A severe vulnerability exists if the code is generated from external input.

Cross-client database access

The addition CLIENT SPECIFIED for the Open SQL statements SELECT, INSERT, UPDATE, MODIFY, DELETE and OPEN CURSOR allows access to data of an other client than the client to which the current user is logged in. Since there is no authority check performed, a user could possibly access data in a client where he has no authorization.

Note that there is no need to add a condition to the WHERE clause for restricting the selection to the current logon client (e.g. MANDT = SY-MANDT) together with CLIENT-SPECIFIED. Without CLIENT SPECIFIED this is ensured by the Open SQL statements.

See also ABAP Security Note on Cross-Client Database Access.

Hard-coded password

Hard-coded passwords should be avoided, since hard-coded credentials like user name or password are a potential security risk. Anybody who has access to the source code (regardless if on the development or production system) would be able to read the password.

Hard-coded user name

Hard-coded user names should be avoided, since hard-coded credentials like user name or password are a potential security risk. In any case it should be avoided that the program flow depends on the name of the currently active user.

Missing authority check at CALL TRANSACTION

CALL TRANSACTION executes an transaction with the given transaction code. Until SAP_BASIS release 7.40 no authority check was performed for CALL TRANSACTION, from SAP_BASIS release 7.40 on, a authority check is only performed, if the addition WITH AUTHORITY CHECK is used.

To solve this issue, the addition WITH AUTHORITY CHECK should be added (if SAP_BASIS version is 7.40 or higher)or the function module AUTHORITY_CHECK_TCODE should be called before.

Missing AUTHORITY-CHECK in reports/programs

Executable programs (reports)—introduced with statement REPORT or PROGRAM—should contain an explicit AUTHORITY-CHECK before executing any statement. This is especially relevant since programs may be executed not only via transaction codes but by other programs using the SUBMIT statement. In these cases no implicit authority check is performed by the system.

The AUTHORITY-CHECK is expected to be placed at the beginning of the INITIALIZATION or START-OF-SELECTION event block, or before the first executable statement if none of these blocks is explicitly specified.

Custom authority check procedures which encapsulate the AUTHORITY-CHECK statement can be specified in the analysis profile as alternative authority check statements to avoid false positives.

Missing check of SY-SUBRC return code after authority check

Improper authorization: The statements AUTHORITY-CHECK and CALL FUNCTION 'AUTHORITY_CHECK_TCODE' require an check of the return code SY-SUBRC to be effective. This check is missing and has to be added.

Suspicious usage of OPEN DATASET

When used with the addtion FILTER, OPEN DATASET allows to execute any operating system command. Thus, the FILTER addtion should be avoided in general. The SXPG framework should be used for OS calls instead.

Suspicious Statements

Suspicious statements are statements which may introduce security risks if not used properly.

Examples of such statements are:

- CALL TRANSACTION - executes the stated SAP transaction code, authorization of the user is only check if stated explicitly.
- EXEC SQL - executes native SQL

The list of relevant statements is specified in the analysis profile.

Usage of system variables (SY-...)

A system variable, e.g., SY-UNAME (user name of current user) or SY-SYSID (SAP system id) is accessed. System variables which contain user, installation or client specific values should be avoided, since these may lead to different behavior depending on logon data or platform.

E.2 Custom Checks for C#

E.2.1 Bad practice

File must have a valid copyright header

Ensures that each file has a valid copyright header.

Avoid using ternary operators

Ternary operators make debugging difficult and represent code that is hard to read.

Usage of goto

The usage of goto is discouraged as it leads to code that is hard to read.

Switch/case without default

Switch/Case statements must have a default case.

Abstract types should not have constructors

Constructors on abstract types can be called only by derived types. Because public constructors create instances of a type, and you cannot create instances of an abstract type. An abstract type that has a public constructor is incorrectly designed.

Avoid partial classes

Partial classes allow the class definitions to be broken up across multiple files. This is considered as a bad practice, because you will not be able to see the bigger picture of the class anymore and may not realize the side-effects you are introducing to the other partial files. By only viewing and working on a part of the class, you will not see all the internal fields that need updating, thus leading to incompatible changes and higher possibility of breaking code or dependencies in the rest of the class.

Methods should not have too many parameters

Methods should not have too many parameters in order to be maintainable.

Do not declare virtual members in sealed types

A public type is sealed and declares a method that is both virtual (Overridable in Visual Basic) and not final. This rule does not report violations for delegate types, which must follow this pattern.

Pointers should not be visible

A public or protected System.IntPtr or System.UIntPtr field is not readonly.

Use params for variable arguments

A public or protected type contains a public or protected method that uses the VarArgs calling convention.

Usage of the var keyword

The usage of the var keyword is discouraged as it makes the code harder to read.

E.2.2 Correctness

Override GetHashCode on overriding Equals

A public type overrides Object.Equals but does not override Object.GetHashCode or vice versa.

Test for NaN correctly

Creates a finding if an expression tests a value against Single.Nan or Double.Nan.

E.2.3 Exception Handling

Exception stacktrace is lost

If you catch an exception and then throw a different exception, you should preserve the original stacktrace by passing the caught exception to the constructor of the thrown exception. Failing to do so will cause the original stacktrace to be lost, which will hinder debugging the root cause.

E.2.4 Performance

Use literals where appropriate

A field is declared static and readonly and is initialized with a value that is computable at compile time. Change the declaration to a const field so that the value is computed at compile time instead of at runtime.

E.2.5 Unused code

Avoid unused private fields

This rule is reported when a private field in your code exists but is not used by any code path.

Avoid unused private methods

This rule is reported when a private method exists but is never called by any code path.

Remove empty finalizers

Creates a finding if a type implements a finalizer that is empty, calls only the base type finalizer, or calls only conditionally emitted methods.

E.3 Custom Checks for C/C++

E.3.1 Bad practice

File must have a valid copyright header

Ensures that each file has a valid copyright header.

Avoid using ternary operators

Ternary operators make debugging difficult and represent code that is hard to read.

Usage of goto

The usage of goto is discouraged as it leads to code that is hard to read.

Methods should not have arrays as arguments.

Having arrays as method argument should be avoided. This check reports an extra finding if 'sizeof' is used on an array argument. Using 'sizeof' on the array argument returns the size of the pointer instead of the byte size of the array as they are treated as pointers by the compiler.

Exception not caught by reference

Exceptions should always be caught by reference to avoid unnecessary copies.

Do not use forbidden types (C++)

Reports findings if a type from a user-defined list is used.

Methods should not have too many parameters (C++)

Methods should not have too many parameters in order to be maintainable.

Multiple classes in one file (C++)

Defining more than one class in a file makes finding the file for a given class very hard. Therefore each file should contain only one top-level class.

Public class attribute (C++)

Classes should not have public attributes as this violates encapsulation. Instead access should be provided using public methods. This rule does not apply to constants and to attributes in structs.

Top-level class name (C++)

Reports if the top-level class name does not match the file name.

E.3.2 Correctness

Missing header protection

Header files must protect from multiple inclusion using the preprocessor. The macro variable used for this must match the filename.

Missing virtual destructor

If a class has no virtual destructor and some other class inherits from it, this can lead to memory not being freed correctly.

E.4 Custom Checks for IEC 61131-3 ST

E.4.1 Bad practice

Assert uppercase keywords and operators

All keywords and operators in Structured Text should be written in uppercase.

E.5 Custom Checks for Java

E.5.1 Bad practice

File must have a valid copyright header

Ensures that each file has a valid copyright header.

Avoid using ternary operators

Ternary operators make debugging difficult and represent code that is hard to read.

Switch/case without default

Switch/Case statements must have a default case.

Avoid inverted conditions

To improve readability, it is desirable to have an assertFalse statement when checking for conditions to be unsatisfied (and vice versa)

Never check for more than one condition in assert statements

Assert statements should never test more than one feature. This way, failed tests can be linked to errors more easily.

Bad assignments

Using assignments in places where comparison operators are frequently used can lead to unreadable code and thus should be avoided.

Transformation of byte sequence into String must consider encoding

When transforming byte sequences into Strings the encoding must be considered. Using the platform encoding may result in interoperability issues.

Avoid leaving deprecated classes / methods / fields

Deprecated classes / methods / fields

Do not compare this and null

This must not be compared with null, because this cannot be null.

finalize() may not be overwritten

The method finalize() may not be overwritten.

hashCode() and equals()

A class implementing one of equals() or hashCode() should also implement the other.

Avoid ignoring JUnit tests

JUnit tests should not be ignored for longer periods of time.

Inner type visiblity

Inner types must have private or protected visibility. If the type must be public, move it to its own file.

Field could be final

Fields that can be made final, should be marked as such to avoid accidental assignment to them.

Import from java.lang

Types from java.lang do not need to be imported explicitly as they are imported by default.

Methods should not have too many parameters

Methods should not have too many parameters in order to be maintainable.

Java unwanted method calls

Certain methods like console or debugging methods should be avoided in code because they could introduce unwanted side effects or reduce performance.

Specify new Logger instances with current class

The logger should be specified with the class it operates in. This way, the log output can be linked to said class.

Maximum type name length

Type name should not be longer than a given number of characters (default=30).

Multiple assignments

Multiple assignments in one statement should be avoided as they hamper readability.

Multiple top-level classes

Multiple top-level classes in a single file are possible in Java but make them hard to find for developers. Better use separate files, inner classes or anonymous classes.

No star imports

Star imports must not be used, because they clutter the local namespace.

Too many constructors

Classes should not contain more than the configured number of constructors (default=5). Too many constructors make the class difficult to use and should thus be avoided.

Too many imports

The number of imports should not be too big. Too many imports are an indication of too many concepts being used in one class.

Number of method parameters

Methods should not take more than 5 parameters. Using a large number of parameters is an indication of high complexity and should be avoided.

Too many methods

Classes should not contain more than then configured number of methods (default=32) as this indicates a high complexity that is difficult to understand.

Modifiers must be in the right order

According to the Java Language Specification the modifiers on methods and fields should be ordered as follows: public/protected/private, abstract, static, final, transient, volatile, synchronized, native, strictfp.

Package name mismatch

Package names must start with one of the configured prefixes.

Call to printStackTrace()

Avoid use of printStackTrace() calls.

Private methods must not be final

Since private methods cannot be overridden, declaring them as final is unnecessary.

Public class attribute

Classes should not have public attributes as this violates encapsulation. Instead access should be provided using public methods.

Method must not be named like constructor

A method should not be named like its containing type's constructor, because this can lead to confusion.

Same package import

Must not import types from own package, as they are accessible without import.

Properly initialize static variable

Static variables that are not protected or private should be initialized directly or in a static initializer.

Suspicious methods

If methods are named similar to ones from Object, this might be a typing error or can lead to confusion. Those methods should be renamed.

Case Blocks Length

Each case block in a switch statement should not be longer than the configured number of statements (default=10).

Unused imports

Unused imports are unnecessary.

Unwanted constructor calls

Constructors of some types, e.g., primitive wrappers and String must not be called, because they only create extra objects which take up more memory and reduce performance, without any other functional effect.

Java unwanted imports

Creates a finding for unwanted imports.

E.5.2 Correctness

Method returns null but return type is Optional

Optional.empty() should be returned instead of null.

E.5.3 Exception Handling

Exception stacktrace is lost

If you catch an exception and then throw a different exception, you should preserve the original stacktrace by passing the caught exception to the constructor of the thrown exception. Failing to do so will cause the original stacktrace to be lost, which will hinder debugging the root cause.

Catch of generic exception

The exceptions java.lang.Throwable, java.lang.Error, java.lang.RuntimeException, and java.lang.Exception must not be handled by a catch block.

Custom exception class

Creates a finding for each custom exception class.

Throw of generic exception

The exceptions java.lang.Throwable, java.lang.Error, java.lang.RuntimeException and java.lang.Exception must not be thrown by methods. Appropriate subclasses should be used instead.

E.5.4 Performance

Calling contains on list

Calling contains() on lists is inefficient. Use hashsets instead.

E.5.5 Test Conventions

Annotation in JUnit 4 test class missing

Public, non-annotated methods in JUnit 4 test classes may be an indicator for a missing @Test annotation.

Test class naming convention

Test classes should be distinguishable from other classes by their name. Reports test classes whose name does not match a configured pattern.

Unit test classes should be located in the same package as the class under test

Unit test classes should be located in the same package as the class under test.

E.5.6 Unused code

Avoid unused private methods

This rule is reported when a private method in your code exists but is not called by any code path.

Avoid unused private fields

This rule is reported when a private field in your code exists but is not used by any code path.

E.6 Custom Checks for JavaScript

E.6.1 Bad practice

File must have a valid copyright header

Ensures that each file has a valid copyright header.

Avoid using ternary operators

Ternary operators make debugging difficult and represent code that is hard to read.

Javascript unwanted method calls

Certain methods like console or debugging methods should be avoided in code because they could introduce unwanted side effects or reduce performance.

Use consistent form for String literals

Use consistent form for String literals, i.e. only one of ″ or ″″

E.6.2 Cross-Browser Compatibility

Internet Explorer compatibility check

Some method calls on global objects are not be supported by all browsers. Please refer to online resources like the Mozilla Developer Network (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference) for details.

Google Closure Method Check

For optimal cross-browser compatibility, certain JavaScript method calls should be replaced with their matching Google Closure methods. Please note that this check uses a heuristic which might lead to false positives.

E.6.3 Javascript (built-in)

Do not use == and !=

The comparison operators == and != should be avoided. Their behavior is inconsistent and the performance of === and !== is better.

E.7 Custom Checks for PL/SQL

E.7.1 Bad practice

Plain SQL SELECT column count

In case you have to select lots of columns, a cursor and FOR loop is the better way as you don't need to add so many local variables.

Unclosed cursors

An opened cursor is not closed within the same procedure.

E.7.2 Performance

No SELECT *

Never use 'SELECT * FROM xxx'—only select the columns you need.

Prefer JOIN over Get_

Do not use Get-functions if it's not absolutely needed—join the tables instead.

E.8 Custom Checks for Python

E.8.1 Bad practice

Do not compare None with == or !=

None should not be compared with == or !=. Instead 'is' or 'is not' should be used.

Do not shadow built-ins

Built-in names should not be used as identifier for variables, parameters, methods and types.

Enclose doc-strings in triple quotes

Documentation strings should be enclosed in triple quotes.

Redundant parentheses check

Reports redundant parentheses within if, while, for and return statements.

E.9 Custom Checks for Xtend

E.9.1 Bad practice

Switch/case without default

Switch/Case statements must have a default case.

Bad assignments

Using assignments in places where comparison operators are frequently used can lead to unreadable code and thus should be avoided.

Do not compare this and null

This must not be compared with null, because this cannot be null.

hashCode() and equals()

A class implementing one of equals() or hashCode() should also implement the other.

Import from java.lang

Types from java.lang do not need to be imported explicitly as they are imported by default.

No star imports

Star imports must not be used, because they clutter the local namespace.

Private methods must not be final

Since private methods cannot be overridden, declaring them as final is unnecessary.

Same package import

Must not import types from own package, as they are accessible without import.

E.9.2 Unused code

Avoid unused private methods

This rule is reported when a private method in your code exists but is not called by any code path.



Glossary

analysis group An analysis group is an aspect of a quality indicator that exposes options for passing data in order to influence analysis of code in *Teamscale*. The options may be for computation of metrics or running of custom checks. 198

analysis profile An analysis profile comprises a set of analyses and their configurations. For each language, there is a default analysis profile, but *Teamscale* users can also configure their individual profiles tailored to suit their needs. 35, 38, 43, 52, 124, 151, 177, 178, 186, 188, 194, 196, 198, 203, 356, 357, 362, 363, 368, 434

ant pattern A pattern, as used by ANT and other tools, to match one or more paths. A question mark (?) matches a single character within a path component but not the path separator. Likewise, a single star (*) matches zero or more characters within a path component but not the path separator. In constrast, a double star (**) matches zero or more characters including the path separator. Examples for Ant patterns are `**/*.java` or `include/*.?`. 46, 323, 387

assessment metric An assessment metric is a metric which divides the system into parts belonging to the green assessment, the yellow and the red one. Example: The metric File Size is an assessment metric. Green files are, for example, those with less than 300 SLOC, yellow are those with less then 750 SLOC and the remaining ones are red. 56, 197, 356, 357, 434

baseline A baseline is a point in time defined by the user. A baseline is often used as reference point to compare the current quality status against, for example in the delta analysis. In Teamscale, a baseline is defined with a calendar date and a corresponding label. It is thereby independent of the underlying branch in case branch support is enabled. 55, 65, 67, 68, 71, 85, 87, 105, 109, 174, 334, 354, 362, 431

blacklist To blacklist a finding means that this finding will be put in the blacklist and not shown directly anymore. In the *Code Detail View* or in the IDE integration, it will be not shown at all. It can, however, still be inspected in the blacklist view. 107, 334, 347

branch support If enabled, the branch support feature allows *Teamscale* to not only analyze the »master« or »trunk« of a repository, but also its branches. 68, 84, 85, 98, 101, 107, 168, 181, 182, 354, 430, 438

clone coverage The clone coverage indicates the percentage of code lines which are covered by at least one clone. It can be interpreted as the probability that a random (evenly distributed) change to a code line needs to be propagated to at least one other clone instance. 59, 88, 89

component In an architecture specification, a system is modeled as a hierarchy of components. A component has a unique name and may have sub-components and mappings. 152, 435

custom check A custom check is an analysis implemented by the user which can be integrated in *Teamscale*. *Teamscale* also comes with a set of pre-written custom checks. 29, 40, 191

delta The delta between two snapshots contains the changes from the repository between a start date and an end date as well as differences in the quality status. It also contains the differences in metric values and assessments as well as the finding churn, for example. Often, the start date is referred to as baseline for the delta. 167, 168, 170, 171, 172, 173, 431

delta analysis The delta analysis computes the delta. 61, 67, 168

dialog A dialog is a UI element of *Teamscale*'s web interface. It dynamically pop ups if the user executes a certain task and closes when the user is done with the task. The user has to press a button in order to close a dialog. 84

external finding An external metric is one created by users.. 177

external metric In contrast to metrics created internally by *Teamscale*, an external metric is a code metric created by users.. 177, 194, 195, 198

external upload commit These commits are not present in the version control system of the project, but artificially added to *Teamscale*.

They provide data from external analysis tools, such as test coverage tools, for example. Often these commits are created by the nightly build system. 100

file dependency In a file-based architecture, the path to the source code files are used for dependencies. 152, 156, 158, 434

finding A finding refers to a specific region in the code, that is likely to hamper software quality. The region in the code is referred to as finding location. Generally, a finding belongs to one finding group. A finding can occur when a metric and its corresponding threshold are violated, for example, when a file exceeds the size limit of 300 SLOC. Then this overly long file becomes a finding. Alternatively, a finding can directly results from a quality analysis. The code clone detection, for example, reveals clones as findings. These clones do not result from a threshold violation of the corresponding clone coverage metric, but are findings themselves. Each finding as a introduction date in *Teamscale*. 17, 18, 26, 40, 59, 64, 85, 104, 145, 155, 175, 199, 436

finding description A finding description provides information about the nature of the finding and its impact on quality. For external findings, finding descriptions are used to associate findings with their findings group. 199, 200

finding group A finding group describes a single analysis or a group of analyses creating findings. For example, there are three different finding groups »File Size«, »Method Length« and »Nesting Depth« which represent one analysis each. The finding group »Comment quality« comprises several analyses, detecting unrelated member comments or empty interface comments, for example. A finding group is always associated with one finding category. 52, 66, 104, 106, 175, 176, 199, 202, 361, 432, 433

finding category A finding category combines several finding groups. For example, the finding category »Structure« contains the three groups »File Size«, »Method Length« and »Nesting Depth«. A finding category represents a quality indicator. 52, 66, 106, 175, 176, 361, 362, 363, 432

finding location The location can span several files (e. g. a code clone), a single file (e. g. a overly long file), a method (e. g. an unused method) or a single line (e. g. a naming convention violation). 50, 104, 107, 175, 432

findings count This metric denotes the number of all findings for a project. Naturally, it depends on which analyses are configured. 88

findings churn A finding churn indicates for a given time internal, how many findings were added to or removed from the system. 62, 64, 100, 102, 167, 174, 361

impacted test An Impacted Test for a given set of changes is a test that executes some of the changed methods.. 78, 80, 82, 436

introduction date The introduction date refers to the revision in which a finding was first detected by *Teamscale*. For internal analysis, this refers to the revision with which the developer introduced the finding in the code base. For external analysis, which are integrated via a nightly build, this refers to the revision with which the finding was first uploaded to *Teamscale*. Hence, the introduction date then refers to the first detection time, not the actual introduction time. 104, 432

issue metric An issue metric is the numeric result of an issue query which can be saved in *Teamscale* and, thus, be used continuously. 140, 365

java timestamp The Unix epoch time (time since January 1, 1970 00:00:00.000 GMT) in milliseconds. 182, 240, 242

lines of code This metric counts all lines of code of a file as displayed to the developer. The count includes empty lines and comments. 56, 59, 88, 116, 164, 196, 364, 367

mapping Mappings between code and architecture define which implementation artifacts belong to which component in the architecture definition. If the architecture is based on file dependencies, a mapping specifies which files belong a component. For type dependencies, it specifies which types map to a component. 154, 431

mapping file Contain mappings from the information in a coverage file to the original source code lines, e.g. line number translation tables or mappings from method IDs to source code lines. 239, 240

method length This metric is an assessment metric and denotes the distribution of code in short (green), long (yellow) and very long (red) methods. The length of each method category is configured by a threshold in the analysis profile. 88

metric A metric captures an automatically measurable aspect of software quality. In can either be a numeric metric or an assessment metric. 26, 85, 174, 191

nesting depth This metric is an assessment metric and denotes the distribution of code over methods with low (green), high (yellow) and very high (red) nesting. Each nesting category is configured by a threshold in the analysis profile. 88

non-code metric A metric that cannot (directly) be derived from source code, e. g. the build status of your system. *Teamscale* offers a

flexible mechanism to integrate show such custom metrics. 113, 114, 189

numeric metric A numeric metric is a metric that consists of a single numeric value. Example: The clone coverage is an example of a numeric metric. Its value could be 20%. Another example is the metric number of files. Its value could be 4000. 56, 59, 60, 88, 197, 367, 373, 376, 378

orphan If a file or a type are not mapped to any component in an architecture specification, they are marked as orphan. 155, 156, 158

page A page is a UI element of *Teamscale*'s web interface. If a perspective has a sidebar on the right hand side, then each entry in this sidebar represents a page. 84, 86, 438

perspective A perspective is the main UI element in *Teamscale*. Toplevel, *Teamscale* consists of serveral different perspective which can be navigated in the top header row of the web UI. 84, 435

policy A policy in an architecture specification determines which components may be in a relationship (allow, deny, tolerate) with each other. 155, 159, 160, 438

pre-commit analysis Pre-commit analysis allows a *Teamscale* client like an IDE plug-in to submit source code for analysis to the *Teamscale* server even though the code hasn't been committed to a version control system yet. The server then analyzes the code changes on-the-fly and informs the client about any change to the number of findings that would occur were the code changes actually committed. 209, 285

quality indicator A quality indicator denotes a certain quality aspect. For example, »Structure« or »Code Duplication« represent quality indicators. 40, 151, 198, 433

quality goal A quality goal represents the target line for the quality status of a system. It can have one of the following four values: perfect (no findings at all), improving (no new findings and no findings in modified code), preserving (no new findings) or none (any number of findings). 18, 66, 86, 105, 109, 120, 175, 361

section A section is a part of a view that is independent of the other content of that view, e. g. one of the main tables of the **Metrics** perspective. 113

severity The severity of a finding is expressed by either a yellow or a red color. It can be configured in the analysis profile and used to filter findings. 40, 105, 191, 362, 363

size metric A size metric indicates the size of a system. It can either count the number of files, lines of code, or source lines of code. 196, 372, 385

task A task in *Teamscale* is a concept to schedule findings for removal and keep track of the process. Multiple findings can be grouped together to one task. 105, 108, 110, 145, 334, 347, 381

test gap analysis The Test Gap Analysis (TGA) can detect holes in the test process by uncovering code that was not tested after its latest modification. The TGA can combine test coverage from different origins (e.g. manual tests and unit tests). The start date after which code modifications are considered is called "Baseline" and is an important parameter of each TGA inspection.. 128

test impact analysis The Test Impact Analysis (TIA) can select and prioritize regression tests given a concrete changeset. The TIA uses testwise coverage for the test selection. Test covering the given changes are called impacted tests.. 78

test gap ratio This metric denouts the amount of modified or new methods covered by tests divided by the total amount of modified or new methods. 76

test gap A test gap is a method that has been changed but not tested ever since. 70, 75, 129, 172, 312, 318, 334, 338, 382, 383, 384

testwise coverage Testwise Coverage is the name of a report format based on JSON. It contains a list of tests contained in the set of all available tests, as well as optional coverage per test and execution results of the test.. 78, 80, 242, 436

threshold A threshold can be used to derive findings from metrics. For example, the metric »File Size« has two thresholds, set to 300 (yellow) and 750 (red) as default. Hence, files which are longer thann 300 SLOC, become a yellow finding. Files longer than 750 SLOC become a red finding. 36, 434

threshold configuration An threshold configuration comprises a set of thresholds for the analyses. 117, 177, 192, 367, 374, 376, 378

timetravel The timetravel is a feature of *Teamscale* which makes it possible to show all of *Teamscale*'s content in a historized form, i. e. to display all information at any point in time of the project's history. 55, 96, 99, 354

treemap Treemaps visualize individual metrics in relation to the source code structure and file sizes. Therefore, directories or source code files of the analyzed system are drawn as rectangles. The area of the rectangle corresponds to the number of lines contained in the file (or for directories all the files included in the directory). The position of the rectangle follows the directory structure, i. e. rectangles for files in the same directory are

drawn next to each other. The shading of the rectangles indicates the hierarchical nesting of the files and directories. 71, 89, 117, 171, 374, 379, 383, 385

trend A trend indicates the evolution of a metric over time. A trend can be calculated both for a numeric metric as well as for an assessment metric. In this first case, the trend is just a simple function in the mathematical sense. In the second case, the trend chart indicates three values per time: the green, the yellow, and the red value and shows how the system distribution over these three categories evolves. If branch support is disabled, the history of the system and, hence, a metric trend, is linear by default. If branch support is enabled, however, the history of a branch is not necessarily linear. To display a linear trend, the »first-parent« stragety is used to pick one parent for each commit. 56, 58, 88, 117, 374

type dependency In a type-based architecture, the full-qualified names of all types (classes, structs, enums etc.) in the codebase are used for dependencies. 152, 156, 158, 434

view A view is a UI element of *Teamscale*'s web interface. In constrast to pages, it cannot be references by the sidebar but appears and disappears as the user navigates and performs certain tasks. 84, 86, 436

violation If a dependency contradicts a modeled policy, it will be treated as an architecture violation. 155, 156



Bibliography

Papers

[Ede+13] Sebastian Eder, Benedikt Hauptmann, Maximilian Junker, Elmar Juergens, Rudolf Vaas, and Karl-Heinz Prommer. "Did We Test Our Changes? Assessing Alignment between Tests and Development in Practice". In: *Proceedings of the Eighth International Workshop on Automation of Software Test (AST'13)*. 2013 (cited on pages 15, 20, 70).

[Ede+12] Sebastian Eder, Maximilian Junker, Elmar Jürgens, Benedikt Hauptmann, Rudolf Vaas, and Karl-Heinz Prommer. "How

much does unused code matter for maintenance?" In: *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*. 2012 (cited on pages 16, 24).

[Eic+01] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J. S. Marron, and Audris Mockus. "Does Code Decay? Assessing the Evidence from Change Management Data". In: *IEEE Software*. 2001 (cited on page 15).

[FRJ09] Martin Feilkas, Daniel Ratiu, and Elmar Jürgens. "The loss of architectural knowledge during system evolution: An industrial case study". In: *Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC'09)*. 2009 (cited on pages 15, 24).

[HHS14] Lars Heinemann, Benjamin Hummel, and Daniela Steidl. "Teamscale: Software Quality Control in Real-Time". In: *Proceedings of the 36th ACM/IEEE International Conference on Software Engineering (ICSE)*. 2014 (cited on page 19).

[Par94] David Lorge Parnas. "Software aging". In: *Proceedings of the 16th International Conference of Software Engineering (ICSE)*. 1994 (cited on page 15).

[Ste+14] Daniela Steidl, Florian Deissenboeck, Martin Poehlmann, Robert Heinke, and Baerbel Uhink-Mergenthaler. "Continuous Software Quality Control in Practice". In: *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2014 (cited on pages 15, 20).

[SE14] Daniela Steidl and Sebastian Eder. "Prioritizing Maintainability Defects by Refactoring Recommendations". In: *Proceedings of the 22nd International Conference on Program Comprehension (ICPC)*. 2014 (cited on page 17).

[SHJ14] Daniela Steidl, Benjamin Hummel, and Elmar Juergens. "Incremental Origin Analysis of Source Code Files". In: *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*. 2014 (cited on page 170).

Online

[Wik] Wikipedia. *Business Intelligence*. https://de.wikipedia.org/wiki/Business_Intelligence (cited on page 14).