PARASOFT

Legacy 9.x Functionality for Concerto and Development Testing Platform

PARASOFT END USER LICENSE AGREEMENT

REDISTRIBUTION NOT PERMITTED

This Agreement has 3 parts. Part I applies if you have not purchased a license to the accompanying software (the "SOFTWARE"). Part II applies if you have purchased a license to the SOFTWARE. Part III applies to all license grants. If you initially acquired a copy of the SOFTWARE without purchasing a license and you wish to purchase a license, contact Parasoft Corporation ("PARASOFT"):

(626) 305-0041

(888) 305-0041 (USA only)

(626) 256-6884 (Fax)

info@parasoft.com

http://www.parasoft.com

PART I -- TERMS APPLICABLE WHEN LICENSE FEES NOT (YET) PAID GRANT.

DISCLAIMER OF WARRANTY.

Free of charge SOFTWARE is provided on an "AS IS" basis, without warranty of any kind, including without limitation the warranties of merchantability, fitness for a particular purpose and non-infringement. The entire risk as to the quality and performance of the SOFTWARE is borne by you. Should the SOFTWARE prove defective, you and not PARASOFT assume the entire cost of any service and repair. This disclaimer of warranty constitutes an essential part of the agreement. SOME JURISDICTIONS DO NOT ALLOW EXCLUSIONS OF AN IMPLIED WARRANTY, SO THIS DISCLAIMER MAY NOT APPLY TO YOU AND YOU MAY HAVE OTHER LEGAL RIGHTS THAT VARY BY JURISDICTION.

NOTE: For BETA versions, it is possible that some functionality, although mentioned in the documentation, is not yet fully implemented.

PART II -- TERMS APPLICABLE WHEN LICENSE FEES PAID

GRANT OF LICENSE.

PARASOFT hereby grants you, and you accept, a limited license to use the enclosed electronic media, user manuals, and any related materials (collectively called the SOFTWARE in this AGREEMENT). You may install the SOFTWARE in only one location on a single disk or in one location on the temporary or permanent replacement of this disk. If you wish to install the SOFTWARE in multiple locations, you must either license an additional copy of the SOFTWARE from PARASOFT or request a multi-user license from PARASOFT. You may not transfer or sub-license, either temporarily or permanently, your right to use the SOFTWARE under this AGREEMENT without the prior written consent of PARASOFT.

LIMITED WARRANTY.

PARASOFT warrants for a period of thirty (30) days from the date of purchase, that under normal use, the material of the electronic media will not prove defective. If, during the thirty (30) day period, the software media shall prove defective, you may return them to PARASOFT for a replacement without charge.

THIS IS A LIMITED WARRANTY AND IT IS THE ONLY WARRANTY MADE BY PARASOFT. PARASOFT MAKES NO OTHER EXPRESS WARRANTY AND NO WARRANTY OF NONINFRINGEMENT OF THIRD PARTIES' RIGHTS. THE DURATION OF IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE, IS LIMITED TO THE ABOVE LIMITED WARRANTY PERIOD; SOME JURISDICTIONS DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO LIMITATIONS MAY NOT APPLY TO YOU. NO PARASOFT DEALER, AGENT, OR

EMPLOYEE IS AUTHORIZED TO MAKE ANY MODIFICATIONS, EXTENSIONS, OR ADDITIONS TO THIS WARRANTY.

If any modifications are made to the SOFTWARE by you during the warranty period; if the media is subjected to accident, abuse, or improper use; or if you violate the terms of this Agreement, then this warranty shall immediately be terminated. This warranty shall not apply if the SOFTWARE is used on or in conjunction with hardware or software other than the unmodified version of hardware and software with which the SOFTWARE was designed to be used as described in the Documentation. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY HAVE OTHER LEGAL RIGHTS THAT VARY BY JURISDICTION.

YOUR ORIGINAL ELECTRONIC MEDIA/ARCHIVAL COPIES.

The electronic media enclosed contain an original PARASOFT label. Use the original electronic media to make "back-up" or "archival" copies for the purpose of running the SOFTWARE program. You should not use the original electronic media in your terminal except to create the archival copy. After recording the archival copies, place the original electronic media in a safe place. Other than these archival copies, you agree that no other copies of the SOFTWARE will be made.

TERM.

This AGREEMENT is effective from the day you install the SOFTWARE and continues until you return the original SOFTWARE to PARASOFT, in which case you must also certify in writing that you have destroyed any archival copies you may have recorded on any memory system or magnetic, electronic, or optical media and likewise any copies of the written materials.

CUSTOMER REGISTRATION.

PARASOFT may from time to time revise or update the SOFTWARE. These revisions will be made generally available at PARASOFT's discretion. Revisions or notification of revisions can only be provided to you if you have registered with a PARASOFT representative or on the Parasoft Web site. PARASOFT's customer services are available only to registered users.

PART III -- TERMS APPLICABLE TO ALL LICENSE GRANTS

SCOPE OF GRANT.

DERIVED PRODUCTS.

Products developed from the use of the SOFTWARE remain your property. No royalty fees or runtime licenses are required on said products.

PARASOFT'S RIGHTS.

You acknowledge that the SOFTWARE is the sole and exclusive property of PARASOFT. By accepting this agreement you do not become the owner of the SOFTWARE, but you do have the right to use the SOFTWARE in accordance with this AGREEMENT. You agree to use your best efforts and all reasonable steps to protect the SOFTWARE from use, reproduction, or distribution, except as authorized by this AGREEMENT. You agree not to disassemble, de-compile or otherwise reverse engineer the SOFTWARE.

SUITABILITY.

PARASOFT has worked hard to make this a quality product, however PARASOFT makes no warranties as to the suitability, accuracy, or operational characteristics of this SOFTWARE. The SOFTWARE is sold on an "as-is" basis.

EXCLUSIONS.

PARASOFT shall have no obligation to support SOFTWARE that is not the then current release.

TERMINATION OF AGREEMENT.

If any of the terms and conditions of this AGREEMENT are broken, this AGREEMENT will terminate automatically. Upon termination, you must return the software to PARASOFT or destroy all copies of the SOFTWARE and Documentation. At that time you must also certify, in writing, that you have not retained any copies of the SOFTWARE.

LIMITATION OF LIABILITY.

You agree that PARASOFT's liability for any damages to you or to any other party shall not exceed the license fee paid for the SOFTWARE.

PARASOFT WILL NOT BE RESPONSIBLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OF THE SOFTWARE ARISING OUT OF ANY BREACH OF THE WARRANTY, EVEN IF PARASOFT HAS BEEN ADVISED OF SUCH DAMAGES. THIS PRODUCT IS SOLD "AS-IS".

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

ENTIRE AGREEMENT.

This Agreement represents the complete agreement concerning this license and may be amended only by a writing executed by both parties. THE ACCEPTANCE OF ANY PURCHASE ORDER PLACED BY YOU IS EXPRESSLY MADE CONDITIONAL ON YOUR ASSENT TO THE TERMS SET FORTH HEREIN, AND NOT THOSE IN YOUR PURCHASE ORDER. If any provision of this Agreement is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This Agreement shall be governed by California law (except for conflict of law provisions).

Parasoft Corporation
101 E. Huntington Drive
Monrovia, CA 91016

Printed in the U.S.A, October 27, 2016

Table of Contents

Introduction

Viewing DTP Tasks in an IDE 4

Widgets

Build Results Widgets 6
Code Widgets 8
Defects Widgets 10
Project Center Widgets 12
Static Analysis (9.x) Widgets 14
Tests (9.x) Widgets 18

Parasoft Test Tool Reports (9.x and older)

Navigating Reports 24
Controlling Schedule, Cost, and Quality 29
Static Analysis 36
Unit Tests 47
Source Code Check-ins 53
Builds 56
Code Review 61
Tools Usage 66
Tests Overview 74
Tests By Date 75
Errors by Category 76
Errors by Severity 84
Recent Test Logs 87
Change-Based Testing 93
Code Metrics Reports 96
Working with the Security Menu 103
Policy Report 108
Project Portfolio Report 109
Code Base Size 111
Build Results 118
Tests (Files) 119
Defects and Enhancements Reports 129
Manual Test Sessions 135
Coverage Report 137

Policy Center (Legacy)

Policy Center Overview 140
Connecting to Policy Center (Standard Edition) 141
Configuring Policy Checking and Reporting 142
Reviewing Policy Check Results 146
Policy Settings 154

Administration

Project Creation and Configuration 164
Report Center Administration Pages 172
Project Center Administration Pages 184
Optional Report Center Configurations 186
Configuring Cache Report Executor 190
Report Center Tools 193
Creating and Locking Down Sandboxes 196

Forwarding DTP Engines 10.x Reports From Data Collector to Team Server 204
Changing Multicast DNS Usage 205
Customizing Exports to Microsoft Word 206
Disabling and Enabling Applications from the Toolbar 207

Integrations

Connecting Report Center and Project Center to Parasoft Test 209
Importing BTS Data from CSV 210
Integrating with Bug Tracking Systems and Requirement Management Systems 215
BTS and RMS Scanner and Updater 221
Integrating with HP Quality Center 235
Integrating with Bugzilla 243
Integrating with IBM Rational ClearQuest 247
Integrating with Atlassian JIRA 253
Integrating with IBM Rational Change and Synergy 261
Integrating with Code Review 267
Integrating Report Center with Emma 271
Integrating Report Center with Source Control Management Systems 275
Integrating with PTC Integrity Source Control Extension 278
Sending Test Results from Third-party Tools to Development Testing Platform 286
Importing Reports to Microsoft Excel 287
Java API 289

Managing Report Center Installation

Managing Report Center Installation in Windows 290
Managing Report Center in Linux 291
Verifying Development Testing Platform Product Availability 294
Migrating Data 295
Upgrading MySQL Server 297
Configuration Manager Internal Details (Linux/Solaris) 298
Creating an Initial Database and Upgrading the Database for Linux (Command-line Menu-driven Method) 300
Registering Automated Startup Manually (Linux) 302

Troubleshooting

Cache Report Executor 305
Team Server 306
Show Source Functionality 307
Switching to Debug Logging Mode 309
Index i

Introduction

This user manual is for organizations using Parasoft Development Testing Platform (DTP) to collect, correlate, and report test and analysis data from 9.x tools, such as Jtest, C++test, and dotTEST 9.x.

For documentation associated with using DTP with the DTP Engines for Java, .NET, and C/C++, use the **help** link in DTP to access the main user manual. The DTP Engines provide richer data, greater flexibility for executing third-party analyzers, and stronger support for IDEs, build systems, and other software development components.

About Parasoft DTP

Parasoft Development Testing Platform (DTP) is an advanced SDLC analytics platform that eliminates the business risk associated with faulty software, accelerates software delivery, and facilitates continuous process improvement. DTP monitors and measures the application of quality practices, such as static analysis, unit testing, coverage analysis, and runtime error detection. Data generated throughout the SDLC is collected, corrlated, and analyized in order to deliver intelligent, actionable findings that enable you to focus on the impact of changec ode and demonstrate full compliance traceability.

For additional information about DTP, see the main user manual by clicking the **help** link in DTP.

Viewing DTP Tasks in an IDE

Development Testing Platform integrates with Eclipse and MS Visual Studio IDEs. Parasoft Test, which is a plugin to Eclipse or MS Visual Studio contains Task Assistant component -- this is a component that brings Development Testing Platform tasks directly to the developer IDEs. You can search through Development Testing Platform tasks, without having to leave development IDE (Eclipse or Visual Studio).

For a detailed description on how to use Task Assistant please see Parasoft Test documentation.

Widgets

The widgets in this documentation show data from the 9.x platform and Project Center.

In this section:

- Build Results Widgets
- Code Widgets
- Defects Widgets
- Project Center Widgets
- Static Analysis (9.x) Widgets
- Tests (9.x) Widgets

Build Results Widgets

Build Results

This widget shows the number of files that failed, contain warnings, and passed during build as reported by 9.x versions of Parasoft tools.

Actions

Click the widget to view the Build Results report. See “Build Results”, page 56, report for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Build Results",
"type": "legacy",
"id": "BuildsHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Jenkins Job Results

This widget shows the status and build results from a Jenkins job.The Jenkins server must allow anonymous read access for the Jenkins-based widgets to function properly.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Jenkins Job Result",
"type": "legacy",
"id": "JenkinsJob"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Jenkins Server:** URL of the Jenkins server.
- **Jenkins Job**: Name of the job.

Code Widgets

Check-ins

This widget shows number of file revisions committed to source control for the associated project by 9.x versions of Parasoft tools.

Actions

Click the widget to view the Source Code Check-ins report. See “Source Code Check-ins”, page 53, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Check-ins",
"type": "legacy",
"id": "RevisionHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter:** Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period:** Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Code Base Size

This widget shows the number of lines of code in the project as reported by 9.x versions of Parasoft tools.

Actions

Click the widget to view the Code Base Size report. See “Code Base Size”, page 111 report for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Code Base Size",
```

```
"type": "legacy",
"id": "CodeBaseSizeHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter:** Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period:** Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Code Review

This widget shows the general status of the code review process as faciliated by 9.x versions of the Parasoft tools.

Actions

Click on a bar in the graph to view the Code Review Activity report. See "Code Review Activity", page 61, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Code Review",
"type": "legacy",
"id": "CodeReviewSummaryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter:** Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period:** Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Defects Widgets

Defect Trend

This widget shows changes in the number of defects over time.

Actions

Click the widget to open the Defects report. See “Defects and Enhancements Reports”, page 129, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Defect Trend",
"type": "legacy",
"id": "PrHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter:** Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Enhancement Trend

This widget shows changes in the number of enhancements over time.

Actions

Click the widget to open the Enhancements report. See “Defects and Enhancements Reports”, page 129, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Enhancement Trend",
"type": "legacy",
"id": "FrHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter:** Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

New Defects by Week

This widget shows the number of defects reported by week in a bar graph. Defects are color-code by state:

- red: new or reopened
- yellow: assigned
- light green: resolved
- dark green: verified

Actions

Mouse over a color-coded section of a bar to see a count of defects for the week. Click on a bar in the widget to open the defects page in Project Center. See “Viewing Defects/Enhancements”, page 115

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "New Defects by Week ",
"type": "legacy",
"id": "DefectsLastWeeks"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter:** Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.

Project Center Widgets

Iteration Burndown

This widget shows the rate of work done versus ideal rate for all open iterations in the selected project.

Actions

Click the widget to open the iteration in Project Center.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Iteration Burndown",
"type": "legacy",
"id": "ProjectIterationsBurndown"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Gate**: Choose a quality gate from the drop-down menu. Project does not have quality gates, then the widget will indicate that no data is available.

Requirements Burndown

This widget shows the change in story points for a requirement over time (burndown).

Actions

Click on the widget to open the requirement in Project Center.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Requirements Burndown",
"type": "legacy",
"id": "RequirementsBurndownChartWidget"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Requirement ID**: Enter the Project Center requirement ID.
- **Start Date**: Enter the start date of the iteration.
- **End Date**: Enter the end date of the iteration (optional).

Tasks

This widget shows all open tasks for an iteration.

Actions

Click on an iteration ID to open the iteration in Project Center.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Tasks",
"type": "legacy",
"id": "TasksInProgress"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Project**: Choose a project.
- **Iteration**: Choose an iteration.
    - Enable the **Show Closed** option to include closed iterations.
- **Status**: Filter widget results by task status.
- **Test Status**: Filter widget results by test status.

Static Analysis (9.x) Widgets

The widgets in this category return static analysis results from version 9.x of Parasoft Test products, i.e., C++test 9.x, Jtest 9.x, dotTEST 9.x, and SOAtest 9.x.

Errors by Category

This widget shows static analysis and unit testing results from 9.x versions of Parasoft Test products grouped by category.

Actions

Click on a graph bar in the widget to open the Errors by Category report. See "Errors by Category", page 76, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Errors by Category",
"type": "legacy",
"id": "ErrorsByCategoryHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Errors by Severity

This widget shows static analysis and unit testing results from 9.x versions of Parasoft Test products grouped by severity.

Actions

Click on a graph bar in the widget to open the Errors by Severity report. See "Errors by Severity", page 84, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Errors by Severity",
"type": "legacy",
"id": "ErrorsBySeverityHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Most Recent Errors by Category

This widget shows details of errors grouped by category detected during the last test run. Results are from 9.x versions of Parasoft Test products.

Actions

Click on a link in the Errors column to open the Errors Detailed Report for the selected category. See "Errors by Category", page 76, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Most Recent Errors by Category",
"type": "legacy",
"id": "ErrorsByCategorySeverityDetailsGrid"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Most Recent Errors by Severity

This widget shows details of errors grouped by severity detected during the last test run. Results are from 9.x versions of Parasoft Test products.

Actions

Click on the widget to open the Errors by Severity report. See “Errors by Severity”, page 84, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Most Recent Errors by Severity",
"type": "legacy",
"id": "ErrorsBySeverityCategoryDetailsGrid"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Static Analysis - Files

This widget shows the number of files that have passed and failed static analysis tests. Results are from 9.x versions of Parasoft Test products.

Actions

Click on the widget to open the Static Analysis - Files report. See “Static Analysis Files”, page 44, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Static Analysis - Files",
"type": "legacy",
"id": "StaticFilesHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.

- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Static Analysis - Violations

This widget shows the number of violations detected in the project. Results are from 9.x versions of Parasoft Test products.

Actions

Click on the widget to open the Static Analysis - Violations report. “Static Analysis Violations”, page 36, for more information.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Static Analysis - Violations",
"type": "legacy",
"id": "StaticViolationsHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Tests (9.x) Widgets

Coverage

This widget shows unit test coverage for the selected project.

Actions

Click on the widget to open the Coverage report. See “Coverage Report”, page 137.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Coverage",
"type": "legacy",
"id": "CoverageHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Functional Tests - Statistics

This widget shows the percentage of tests passed, number of tests, number of failures, number of incomplete tests, and total execution time for the most recent functional test run. The widget also indicates if there has been a change during the selected date range for each statistic.

Actions

Click on the widget to open the Tests (Files) by Type report. See “Tests (Files) By Type”, page 120.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Functional Tests - Statistics",
"type": "legacy",
"id": "functionalTestMetricWidget"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Functional Tests - Summary

This widget shows the percentage of functional tests that are succeeding, as well as the change in the success rate during the selected period.

Actions

Click on the widget to open the Tests (Files) by Type report. See “Tests (Files) By Type”, page 120.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Functional Tests - Summary",
"type": "legacy",
"id": "FunctionalTestSummary"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Manual Tests Sessions

This widget lists all the manual test sessions that have been run or scheduled for the selected project.

Actions

Click on the widget to open the Manual Test Sessions report. See “Manual Test Sessions”, page 135.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Manual Test Sessions",
"type": "legacy",
"id": "TestingSessionsRunsHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Tests (Files)

This widget shows the number of incomplete, failed, and passed test cases.

Actions

Click on the widget to open the Tests (Files) report. See "Tests (Files)", page 119.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Tests (Files)",
"type": "legacy",
"id": "TestHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Unit Tests

This widget shows the number unit test failures and the total number of test cases over time.

Actions

Click on the widget to open the Test Cases report. See "Test Cases Report", page 152.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Unit Tests",
"type": "legacy",
"id": "UnitTestingHistoryChart"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Unit Tests - Statistics

This widget shows the percentage of tests passed, number of tests, number of failures, number of incomplete tests, and total execution time for the most recent unit test run. The widget also indicates if there has been a change during the selected date range for each statistic.

Actions

Click on the widget to open the Test Cases report. See “Test Cases Report”, page 152.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

```
"name": "Unit Tests - Statistics",
"type": "legacy",
"id": "unitTestMetricWidget"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Unit Tests - Summary

This widget shows the percentage of unit tests that are succeeding, as well as the change in the success rate during the selected period.

Actions

Click on the widget to open the Test Cases report. See “Test Cases Report”, page 152.

Custom Dashboard Properties

Use the following properties when adding this widget to a custom dashboard:

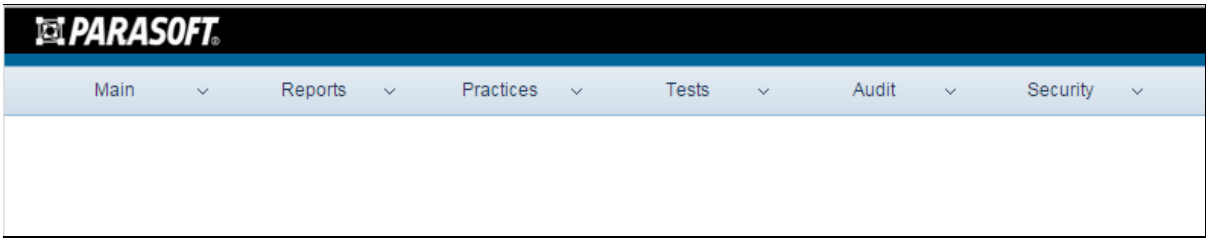
```
"name": "Unit Tests - Summary",
"type": "legacy",
"id": "UnitTestSummary"
```

Settings

- **Title**: Enter a new title to replace the default title that appears on the dashboard.
- **Filter**: Choose Dashboard Settings to use the dashboard filter or choose a filter from the drop-down menu.
- **Period**: Choose Dashboard Settings to use the dashboard date range or choose a period from the drop-down menu.

Parasoft Test Tool Reports (9.x and older)

Widgets associated with Parasoft Test tools (i.e. C/C++test, Jtest, dotTEST, SOAtest) 9.x and older link to nested reports that provide detailed information about software quality activities (see “Static Analysis (9.x) Widgets”, page 14 and “Tests (9.x) Widgets”, page 18). You can also access the reports menu (previously available in Concerto and DTP 5.1 and older) by entering [DTP_DOMAIN] /grs/reports.jsp into your browser address bar:



You can bookmark reports, change the scope of a report, sort tables, and refine the presented data in other ways that enable you to gain insight into your development processes.

In this section:

- Navigating Reports
- Controlling Schedule, Cost, and Quality
- Static Analysis
- Unit Tests
- Source Code Check-ins
- Builds
- Code Review
- Tools Usage
- Tests Overview
- Tests By Date
- Errors by Category
- Errors by Severity
- Recent Test Logs
- Change-Based Testing
- Code Metrics Reports
- Working with the Security Menu
- Policy Report
- Project Portfolio Report
- Code Base Size
- Build Results
- Tests (Files)
- Defects and Enhancements Reports
- Manual Test Sessions

Navigating Reports

Reports present data in readily consumable bits. Most reports also enable you to drill down into more detailed information about a particular segment of the report.

Bookmarking Reports

Click on the **Bookmark** link to modify the URL so that a report can be bookmarked in your browser.

The Bookmark link does not add the link to the report your browser's bookmarks folder. You must add the URL using your browser's bookmark function after clicking the Bookmark link.

You must be logged in to Report Center to view the bookmarked report. If you select the bookmarked report page from your browser and are not logged in to Report Center, you will be prompted to do so. Upon logging in, the bookmarked report will open with current data displayed.

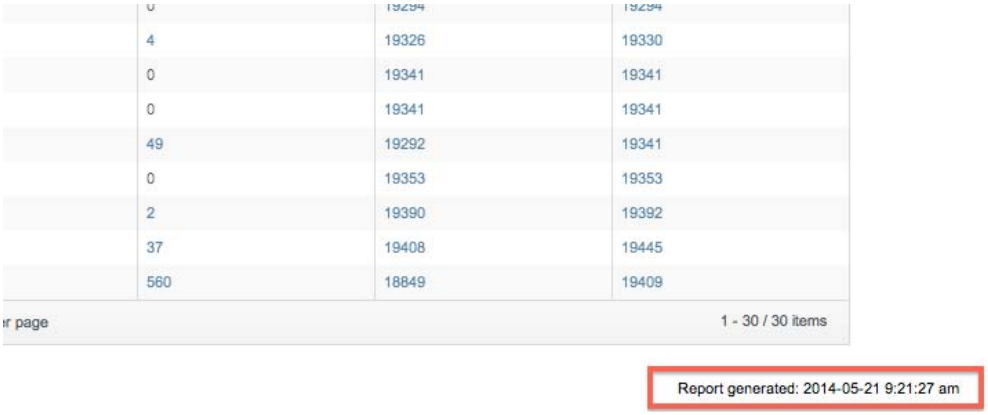
You can also export report data as XML and import it to another program. See "Importing Reports to Microsoft Excel", page 287, for more information.

Refreshing Reports

When you view a report, the data from the Development Testing Platform database is cached on the Development Testing Platform server. If another user uses the same parameters (project, time-span, etc.) to view the same report, the data is retrieved from the cache rather than the database.

By default, the entire report cache on the Development Testing Platform server is cleared once a day at 00:00. However, you can refresh any report with the latest data from the Development Testing Platform database on-demand. To refresh a report, click the **Refresh** link

All DTP reports, including reports that are accessed from the dashboard view, have a timestamp so you know the age of the data you're working with:

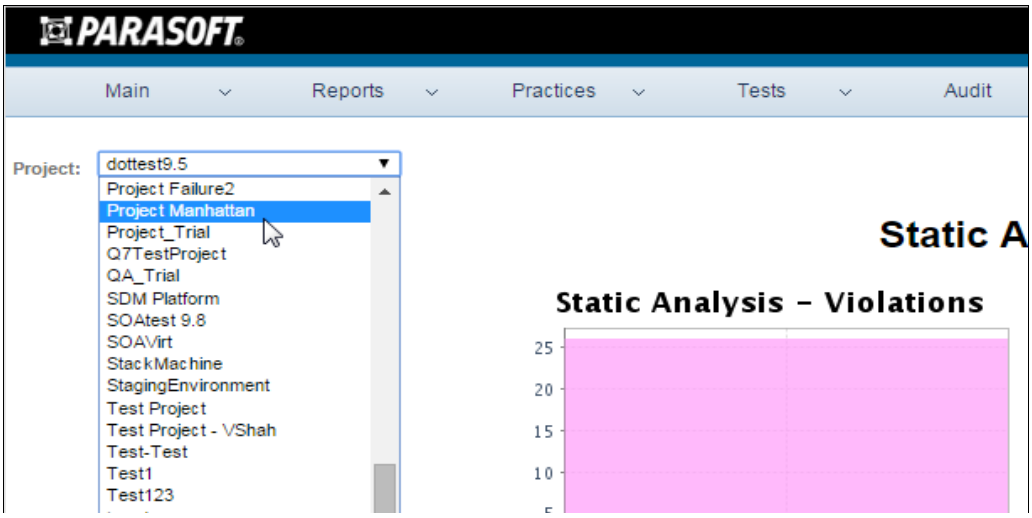


Filtering Reports

Depending on the nature of the data, reports may have options to filter by date, time frame, project, and/or project team members. The following filter options may be available:

Filter by Project

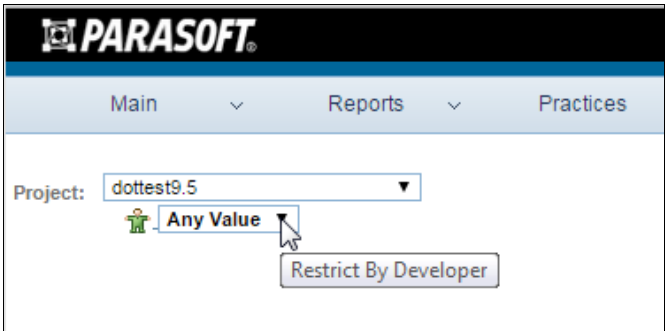
Click the **Project** drop-down menu and choose a project.



Filter by Project and Team Member

1. Choose a project from the **Project** drop-down menu.

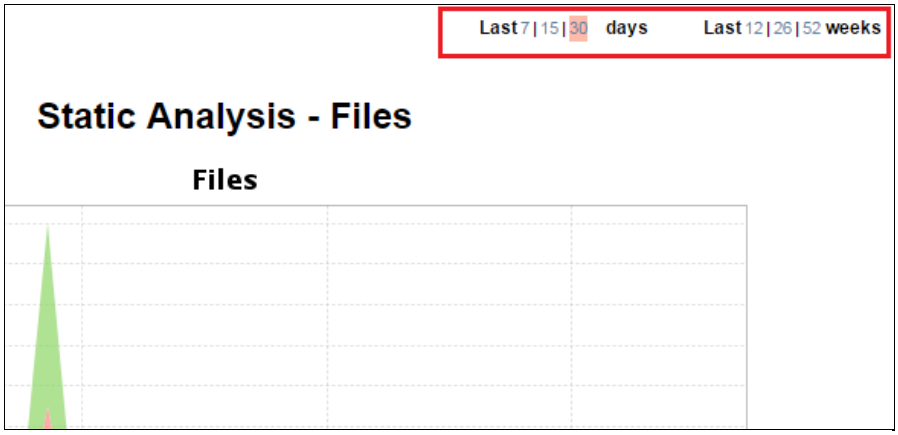
2. Choose a team member from the Restrict by Developer drop-down menu.



Filter by Period

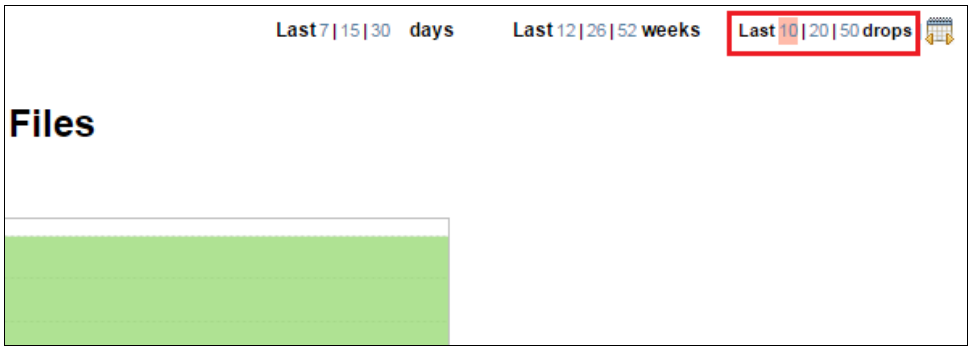
You can filter by the most recent number of days or weeks.

- Click on **7**, **15**, or **30** to filter data by the most recent number of days.
- Click on **12**, **26**, or **52** to filter data by most recent number of weeks.



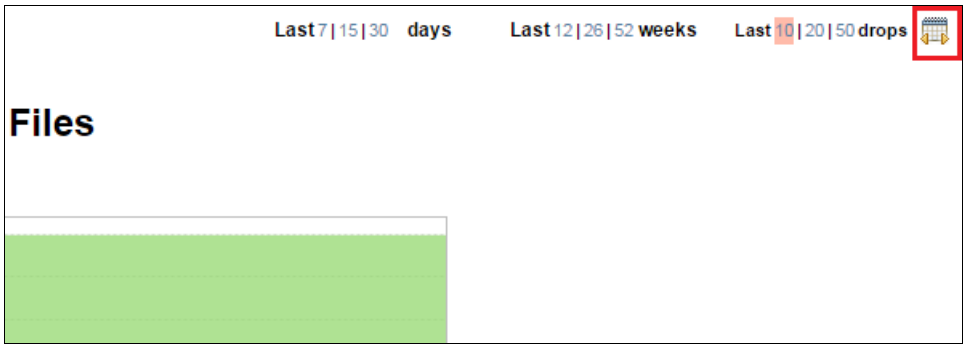
Filter by Most Recent Drops

Click on **10**, **20**, or **50** to filter by the most recent number of drops.

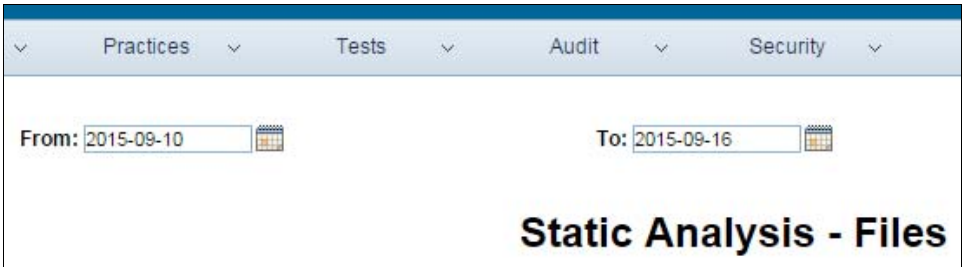


Filter by Custom Date Range

1. Click on the calendar icon to switch to date range mode.



2. Enter dates in the From and To fields to specify a custom date range.



Sorting Tables

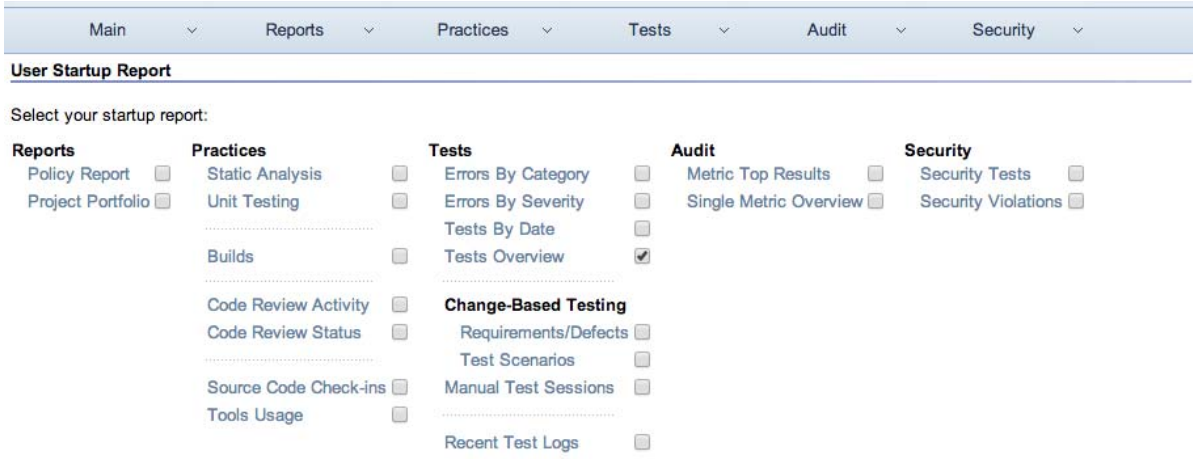
Some tables can be sorted by clicking the column header.

Date	Incomplete Tests	Failed Tests	Passed Tests	Tool	User	Total Tests
Mar 30,06 (419856)	0	0	1	Source Scanner	builds	1
Mar 29,06 (419897)	0	3	0	JUnit	sdtest	3
Mar 29,06 (419896)	0	3	7	JUnit	sdtest	10
Mar 29,06 (419895)	0	2	1	JUnit	sdtest	3
Mar 29,06 (419893)	0	0	54	RTest	devtest	54
Mar 29,06 (419891)	0	15	65	RTest	devtest	80
Mar 29,06 (419890)	0	0	79	RTest	devtest	79
Mar 29,06 (419886)	0	0	79	RTest	devtest	79
Mar 29,06 (419888)	0	5	75	RTest	devtest	80
Mar 29,06 (419884)	0	3	0	JUnit	sdtest	3

Sort by 'Tool'

User Startup Report

You can specify which report displays by default when users access the 9.x and older reports view. Choose **Main> Startup Report** and enable a default report:



Controlling Schedule, Cost, and Quality

The data sent to Parasoft DTP (or Concerto) from Parasoft code analysis and testing tools version 9.x and older follows a report-centric paradigm to provide critical insight about the development processes. This chapter provides guidance on using the report-centric paradigm to help you make decisions about when to release.

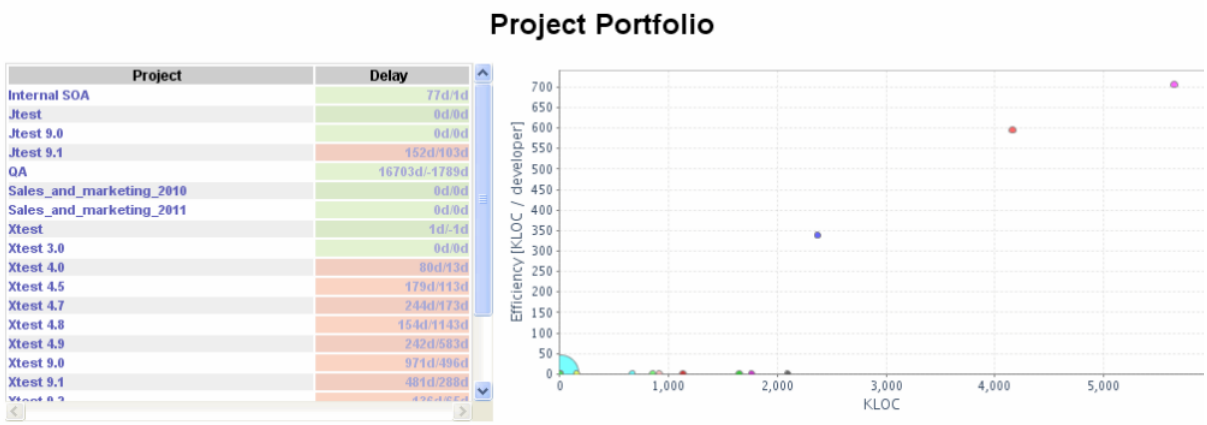
Following are the schedule-, cost- and quality-control topics discussed in this section:

- Verifying Projects Are On-schedule
- Verifying the Amount of Source Code Modified Daily
- Verifying Code was Tested Thoroughly
- Verifying Bugs Are Decreasing and Under Control
- Identifying Weak Requirements and Determining Whether Projects Should Remain in a Release

Verifying Projects Are On-schedule

The Report Center Project Portfolio provides a single view of all projects across the development organization. The dashboard's cross-project view provides quick alert when development metrics indicate potential trouble with a given project.

Choose **Reports> Project Portfolio** in the Reports view to access the report.



In addition to an overall project score, Report Center shows trends, differentials, estimated time to arrival (ETA), estimated days to reach set milestones and separate scores for each of the following project elements:

- PR
- FR
- Tests (all with the exception of manual tests)
- Unit testing (UT)
- Coding standards (CS)
- Efficiency

Drill downs allow for assessment and analysis of project status.

Verifying the Amount of Source Code Modified Daily

As projects progress, you want to know that work is getting done. Integrating with a source control management system allows you to use the Code widgets, which helps you understand team productivity. See “Code Widgets”, page 8.

For more information about Report Center integrating with source control management systems see “Integrating Report Center with Source Control Management Systems”, page 275.

If you are working with a project that has a branch filter (specified using a Source Control Filter in the Administration> Edit Project area), you can restrict source code reports to display code statistics for only a specific branch. To do this, simply choose the appropriate branch name from the **Branch** box.

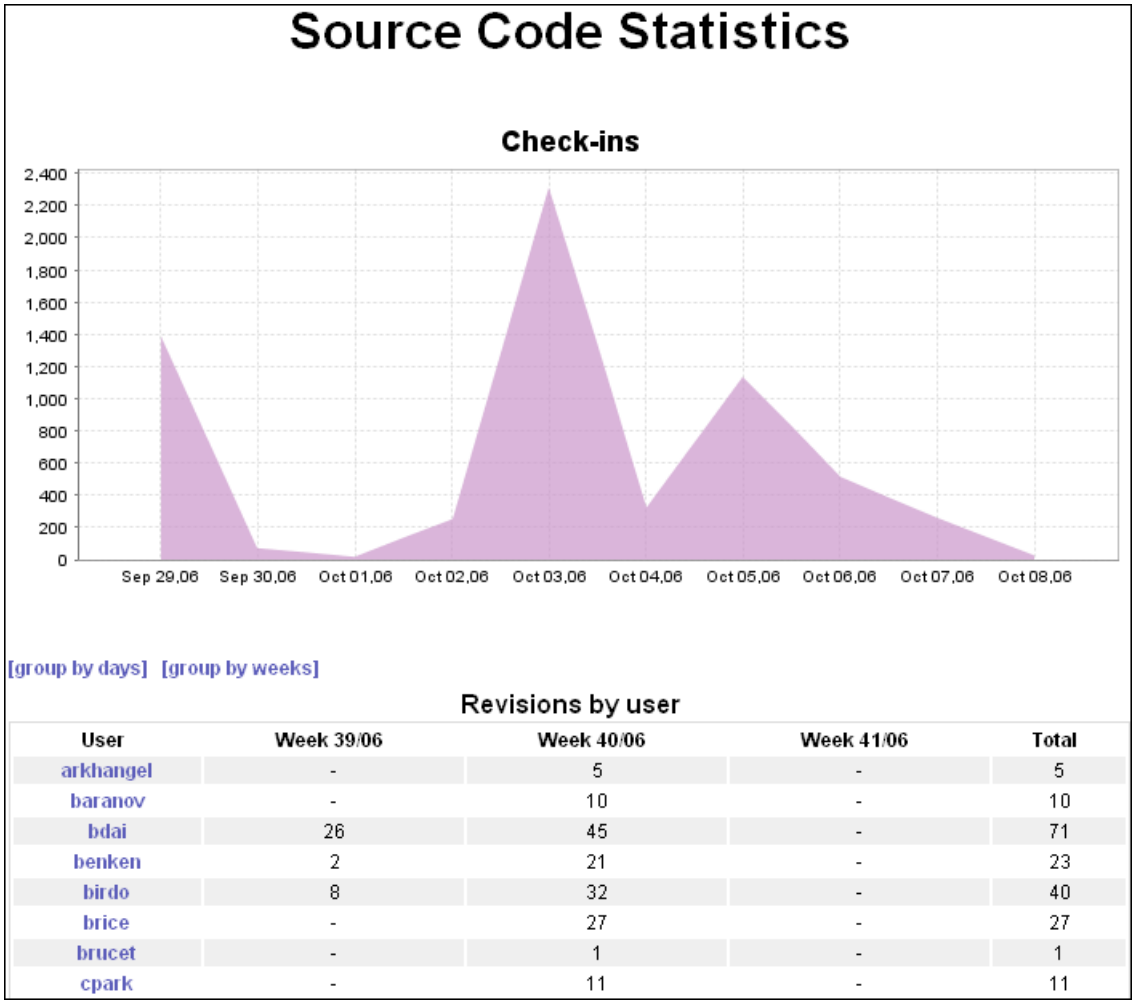
Drill-downs available from the Code Base Size graph provide answers to the following questions:

- Who is working on the code?
- Who is checking in what code?
- What is each team member actually working on?

The graph provides a comprehensive view of the changing nature of your code base, charting overall code growth by project, team and individual team member. Report Center dashboards chart such key code base metrics as number of new file revisions per day or per week, number of lines changed per file, and author of file changes.

The code growth statistics reported provide the means to objectively monitor development efficiencies (or inefficiencies), track development progress, and proactively respond when metrics indicate project goals or deadlines may be at risk. The correlation of code growth with other development metrics

delivered on other Report Center reports, such as Feature Requests, PRs, and Test Results enables regular, on-going evaluation of the impact of code change on code quality and readiness.



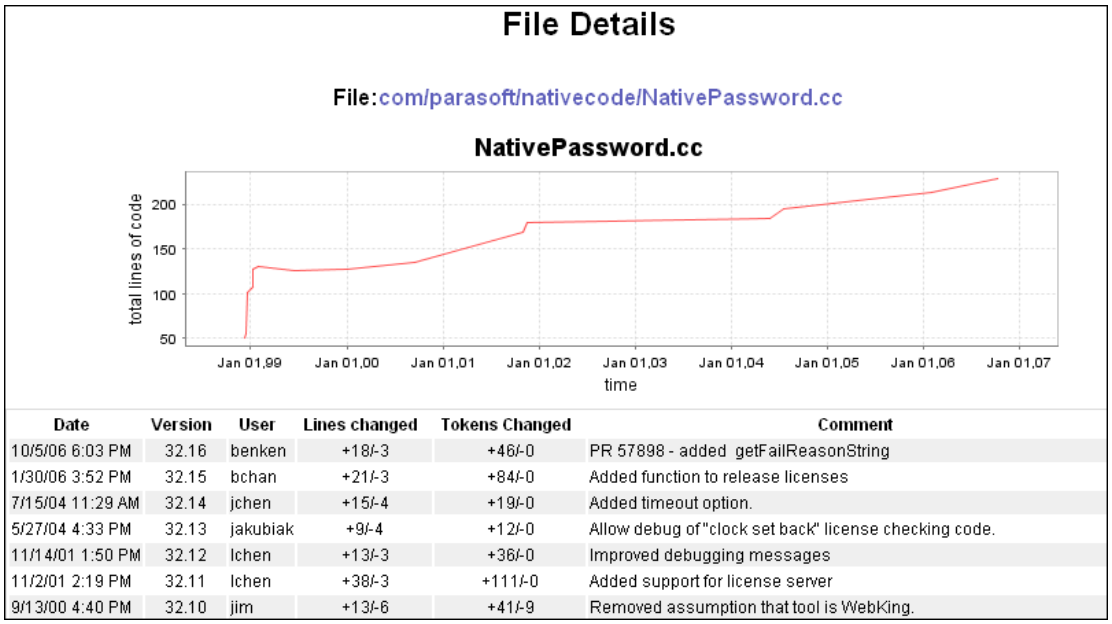
User	Week 39/06	Week 40/06	Week 41/06	Total
arkhangel	-	5	-	5
baranov	-	10	-	10
bdai	26	45	-	71
benken	2	21	-	23
birdo	8	32	-	40
brice	-	27	-	27
brucet	-	1	-	1
cpark	-	11	-	11

Drilling down from the Check-ins graph, you can look at source files and see the entire history of changes within the file. Projects can be organized so that you are able to see which piece of source code each team member is working on. The Source Code Statistics - Check-ins graph provides the architect with development productivity information. You can drill down to the File Details graph to see the growth of a specific file that makes up a specific feature, and then assess whether its growth is appropriate to its priority level.

File Details

1. Click on the Code Base Size graph to open the Source Control Summary report.

2. Click on a link in the File column to open the File Details drill-down report.



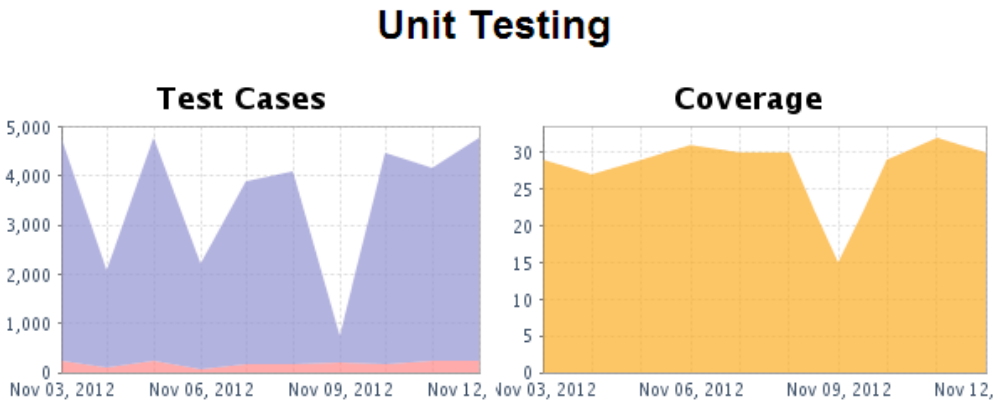
Date	Version	User	Lines changed	Tokens Changed	Comment
10/5/06 6:03 PM	32.16	benken	+18/-3	+46/-0	PR 57898 - added getFailReasonString
1/30/06 3:52 PM	32.15	bchan	+21/-3	+84/-0	Added function to release licenses
7/15/04 11:29 AM	32.14	jchen	+15/-4	+19/-0	Added timeout option.
5/27/04 4:33 PM	32.13	jakubiak	+9/-4	+12/-0	Allow debug of "clock set back" license checking code.
11/14/01 1:50 PM	32.12	lchen	+13/-3	+36/-0	Improved debugging messages
11/2/01 2:19 PM	32.11	lchen	+38/-3	+111/-0	Added support for license server
9/13/00 4:40 PM	32.10	jim	+13/-6	+41/-9	Removed assumption that tool is WebKing.

The Code Base Size graph along with its drill-down graphs and tables provide visibiliity into source control activities. You can see who is working on what and assess the progress of each project, feature and file.

Verifying Code was Tested Thoroughly

To verify whether code was tested thoroughly, you can examine data from unit tests, along with data results displayed in the Tests by Type graph and Manual Tests Efforts graph.

Choose **Practices> Unit Testing** to access the Test Cases anc Coverage graphs and their drill-down reports.



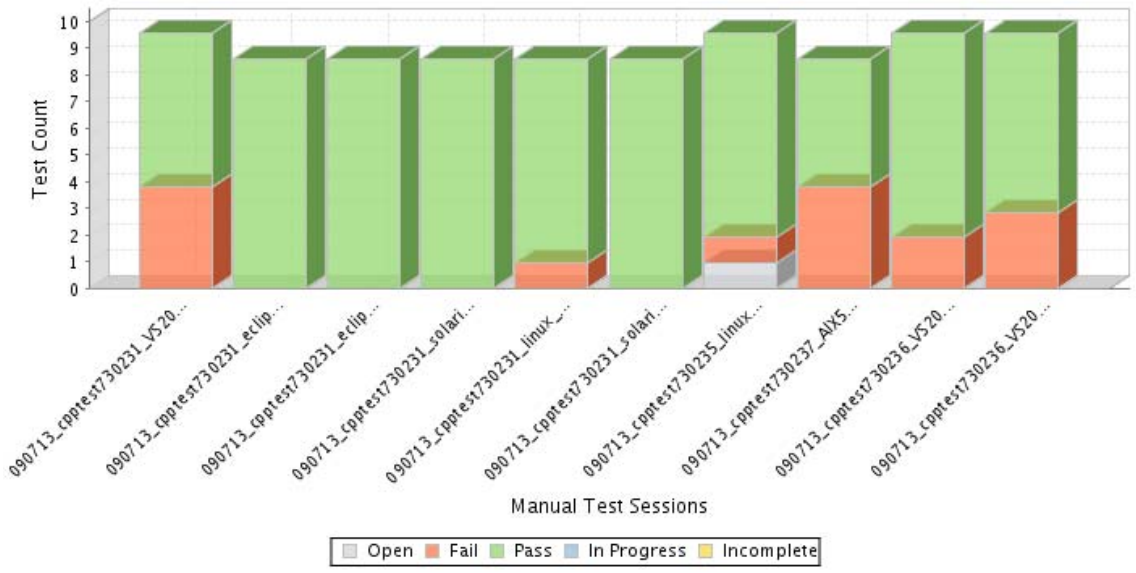
Automated unit tests provide visibility to exaclty how much testing is being performed and the results of those tests. If tests are passing but the test coverage is too low, it may be an indication that the code robust and functionally sound, lacks adequate tests to verify completeness of quality.

The Coverage report and its drill-downs to you which test cases passed, how many passed, and the percentage of code tested that was covered during various stages of the development process.

After reviewing unit testing reports, you can open the Tests graph and drill down to find out whether coding standards (among other testing types) are being adhered to. The Tests by Type report shows the number of incomplete, failed, and passed coding standards tests for a specified date. It also lists the machine and tool on which the tests were run.

To verify thoroughness of manual testing, go to the Manual Test Sessions graph (Figure 1). Drill-downs from the Manual Test Sessions graph and table provide details about code coverage, as well as the number of items, use cases, tests pending, tests executed, and time spent manually testing each module. You can see the testing status (In Progress, Incomplete, Failure, Success) of each module, too.

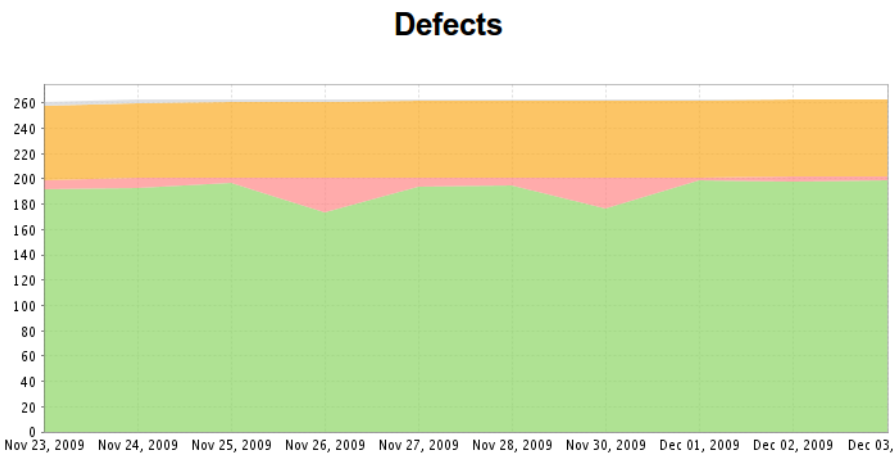
Figure 1: Manual Test Sessions



For more details about manual testing, see “Manual Test Sessions”, page 135.

Verifying Bugs Are Decreasing and Under Control

Click on a the Defects Trend widget to access the Defects report (see “Defects Widgets”, page 10), which shows potential bugs in your application. Defect reporting and remediation follows the same process as features.



Identifying Weak Requirements and Determining Whether Projects Should Remain in a Release

As features/requirements are being implemented, testing occurs, and then bugs are discovered and reported. The following questions regarding bugs need to be answered:

- Is the number of bugs decreasing?
- Are bugs under control?

When bugs are under control, the line that represents the number of fixed bugs in the Bugs History Overview graph will go up and down repeatedly reflecting that bugs were found and then fixed, then more bugs were found and those were fixed, and so on. It is a jagged pattern, similar to a tooth-edged saw. Slight patterns are okay. What is important to see is that bugs are being caught and fixed at a steady rate.

What if you do not see a repeated up and down pattern? Perhaps the line is steadily increasing without any dips. That means that more and more bugs are being found, but few, if any, are being resolved. Further, it indicates a problem that needs some investigating.

Perhaps few bugs are being found, or maybe none at all? Such a case would indicate that either testing is not happening, or developers are writing perfect code. Whatever the case might be, if the tooth-edged saw patter is not reflected in the graph, it is time to do some investigating and take action!

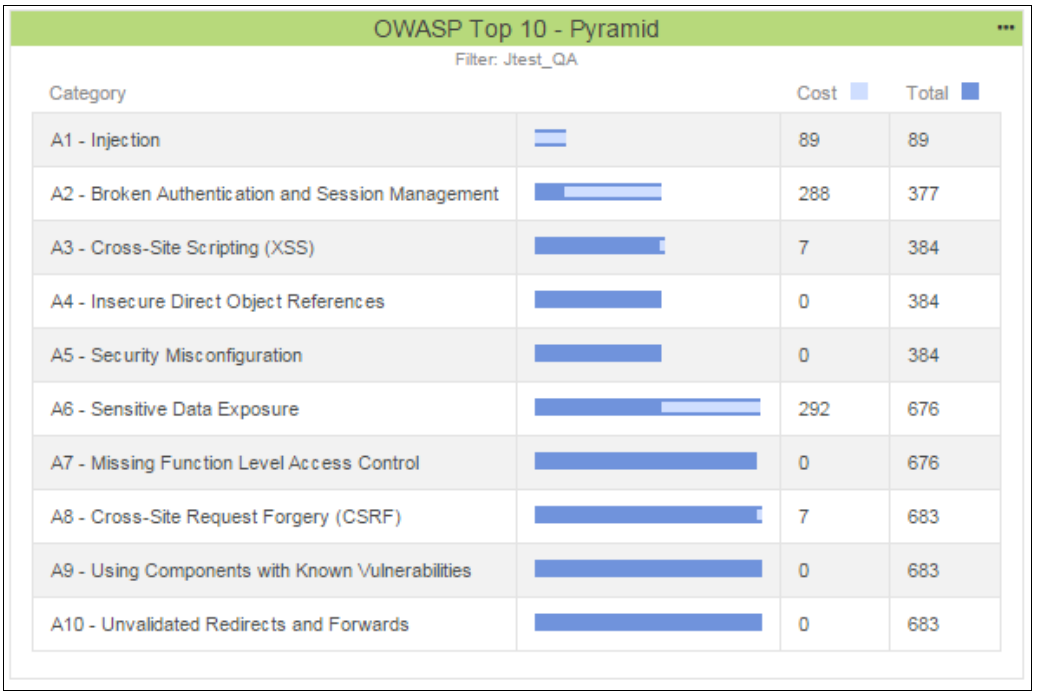
To find the core of the problem, it is best to have a system that can correlate bug fixes with the source code. This system can be implemented as an IDE plugin that can be used to track PR's and FR's information with the relation to the code. This will help to differentiate which files were fixed or modified for a specific requirement. For instance, if manual testing was performed on a recent project, the results of manual tests can be viewed and answer questions such as the following so that the source of the problem can be discovered:

- Who performed the testing?
- How much time was spent testing?
- What were the tests?
- What were the results?

When the problem is found, it needs to be determined whether the feature involved should be included in the release or if it should be waived to the next release so that the projected release date can be made. A good determinant is to figure out how much code needs to be added to fix the bug. The more code needed, the higher the cost.

Determining Cost of Violations

Report Center includes the OWASP Top 10 - Pyramid widget, which helps you understand the cost associated with defects.

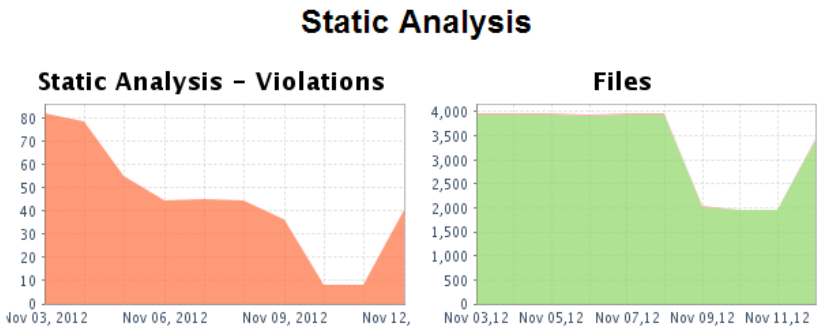


The cost of a violation is calculated by multiplying the number of violations for a rule by a respective weight. By default, the weight is 1. You can update an .xlsx file that contains the rule weights to configure how the cost associated with violations is determined in your organization. The .xlsx file is in the following directory: `DTP_HOME/grs/datasource/JavaOWASPTop10RulesWeight.xlsx`.

Static Analysis

The set of Static Analysis graphs serves as a means to introduce static analysis into your team and ensure that static analysis becomes an enduring part of your software development life cycle. The following two graphs are available from the Static Analysis> Violations menu

- Static Analysis Violations
- Static Analysis Files

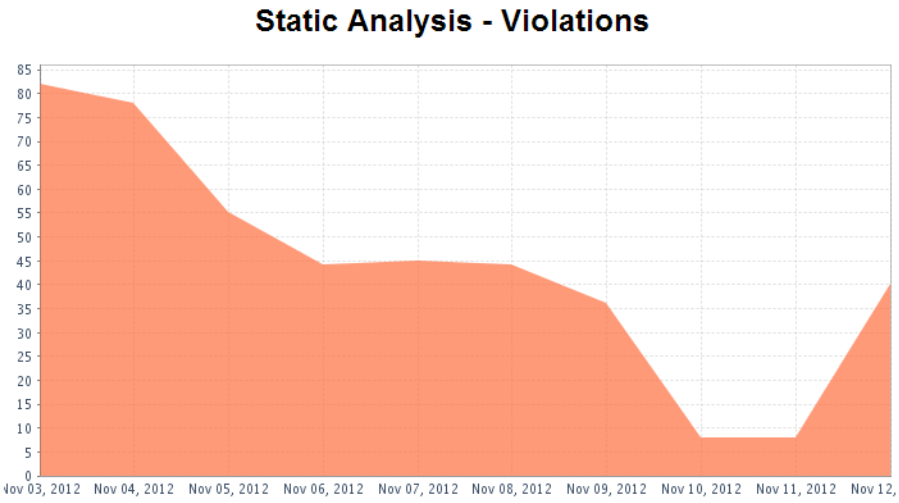


How do I get here? *Practices > Static Analysis*

Static Analysis Violations

The Static Analysis Violations graph provides a quick way for developers to see the number of static analysis violations that have been detected in the selected project's code, and then drill down to view the source of those violations along with the developer who is responsible for that code. The report presents the trend of number of static analysis violations detected in your project source code. Users

can easily see the total number of violations, as well as new violations introduced after the most recent analysis.



New Violations	- violations triggered on a specific date and not triggered on the 'Base Date'
Remaining Violations	- violations triggered on both a specific date and the 'Base Date'

Select how to determine the 'Base Date' for a specific date:

Day-to-day basis: ○ ?

Fixed Base Date: ◉ ?

- Relative to the 'Base Date' [Switch to non delta mode]

Date	New Violations*	Fixed Violations*	Remaining Violations*	Suppressed violations	Violations	Files failed
Nov 12, 2012	0	42	40	2054	40	29
Nov 11, 2012	0	74	8	3007	8	3
Nov 10, 2012	0	74	8	3007	8	3
Nov 09, 2012	0	46	36	721	36	27
Nov 08, 2012	0	38	44	3728	44	30
Nov 07, 2012	0	37	45	3728	45	31
Nov 06, 2012	0	38	44	3713	44	30
Nov 05, 2012	0	27	55	3723	55	36
Nov 04, 2012	0	4	78	3723	78	37
Nov 03, 2012	0	0	82	4028	82	38

How do I get here? *Practices > Static Analysis > Violations*

The following information is displayed in the table below the report:

- New Violations: Violations introduced after previous analysis run. **Note: Switch to "Fixed Base Date" to view new violations relative to a specific date*.
- Fixed Violations: Number of violations fixed since previous analysis.
- Remaining Violations: Number of unfixed violations from previous analysis run.
- Suppressed violations: Number of suppressed violations in your source code.
- Violations: Total number of violations detected in your source code.
- Files failed: Number of files in which some violation(s) were detected

For each drop date, the exact number of static analysis violations that have been detected in the selected project's code is listed. The report can be switched to non-delta mode (see note below).

.

This report is available in two modes:

- Delta mode is the default view described above that distinguishes between new and remaining violations.
- Non-delta mode distinguishes between the number of violations and suppressed violations per date.

Click the **[Switch to . .]** link to toggle between modes.

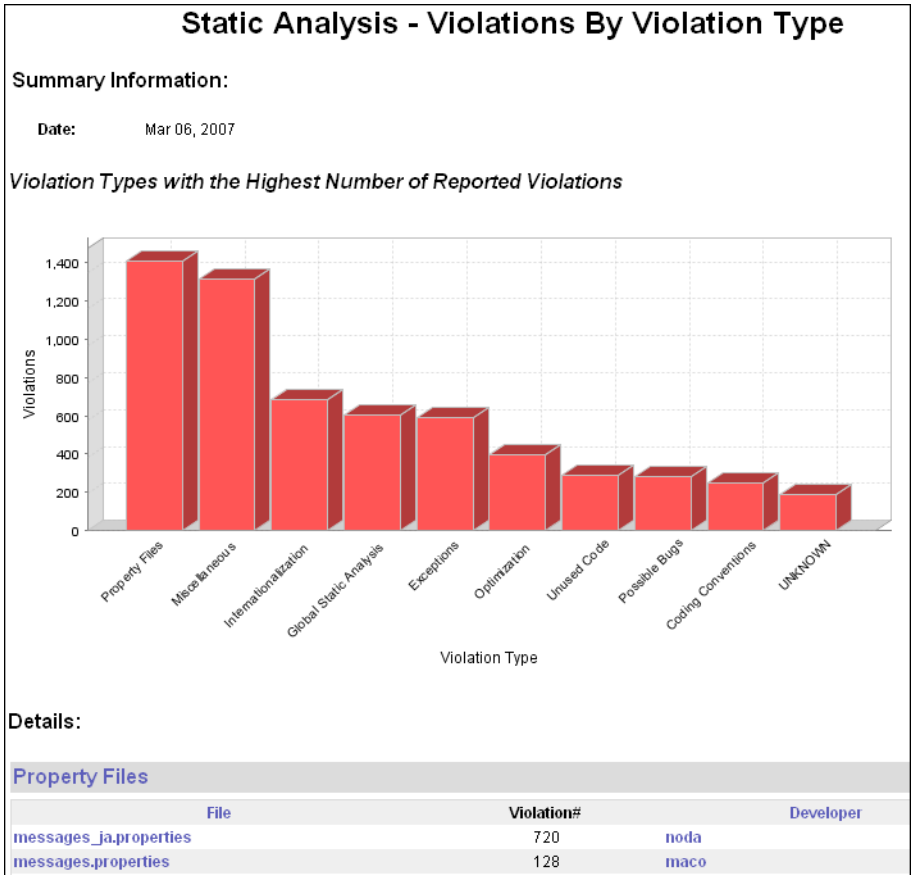
Static Analysis Violations

Static Analysis Violations data can be displayed in three ways:

- By Violation Type: In the table below the Static Analysis Violations chart, click on the number of violations in the *Violations* column to access this view.
- By File: In the table below the Static Analysis Violations chart, click on the number of files in the *Files failed* column to access this view.
- By Developers: Open either the By Violation Type or By File view and click the **Developer(s)** column header to access this view.

By Violation Type

Each bar in the Static Analysis Violations by Type graph represents a violation category that has the highest number of reported violations; up to 10 categories are shown.



How do I get here? *Practices> Static Analysis> Violations> Table> "Violations" column> [number of violations]*

The Details table lists the following information for each violation type:

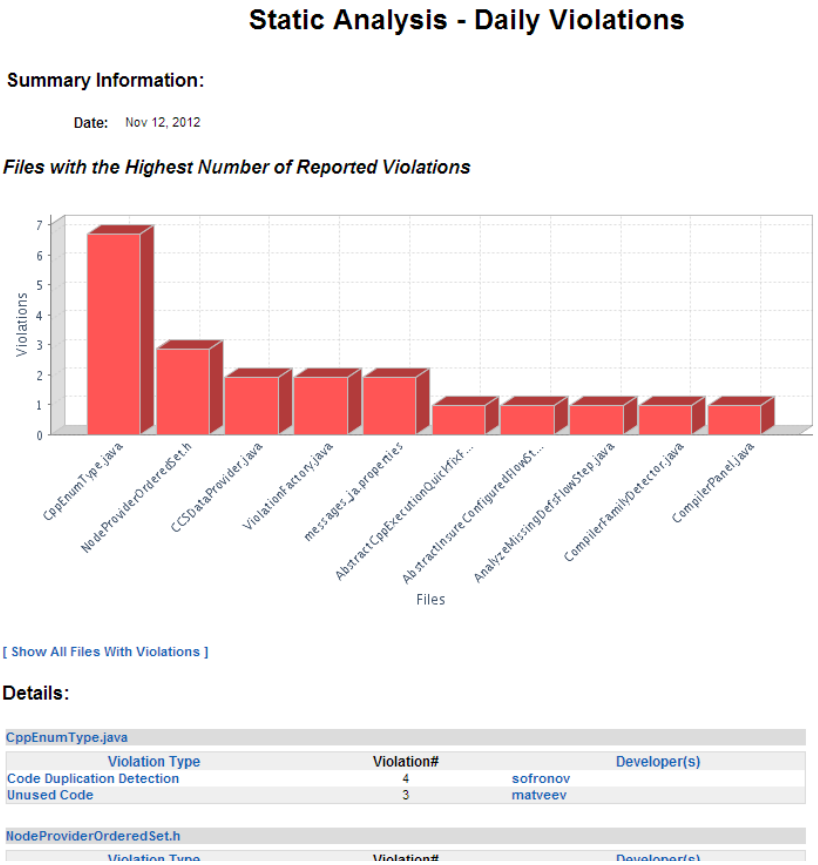
- **File:** Name of the file that contains violations.
- **Violations:** Number of violations detected in the listed file.
- **Developers:** Login ID of the developer responsible for the violation(s) in the listed file.

Click on any link in the table to change views and access additional information:

- **Files** column header: Switch to By File view.
- **Developer(s)** column header: Switch to By Developer view.
- **[Violation type name]**: Opens the Static Analysis Violations Details by Violation Type page. See "For Violation Type", page 41 for details.
- **[File name]**: Opens the Static Analysis Violations Details by File page. See see "For File" on page 42 for details.
- **[Developer name]**: Opens the Static Analysis Violations Details by Developer page. See see "For Developer" on page 42 for details.

By File

Each bar in the Static Analysis Violations by File graph represents a file that has the highest number of reported violations; up to 10 files are shown.



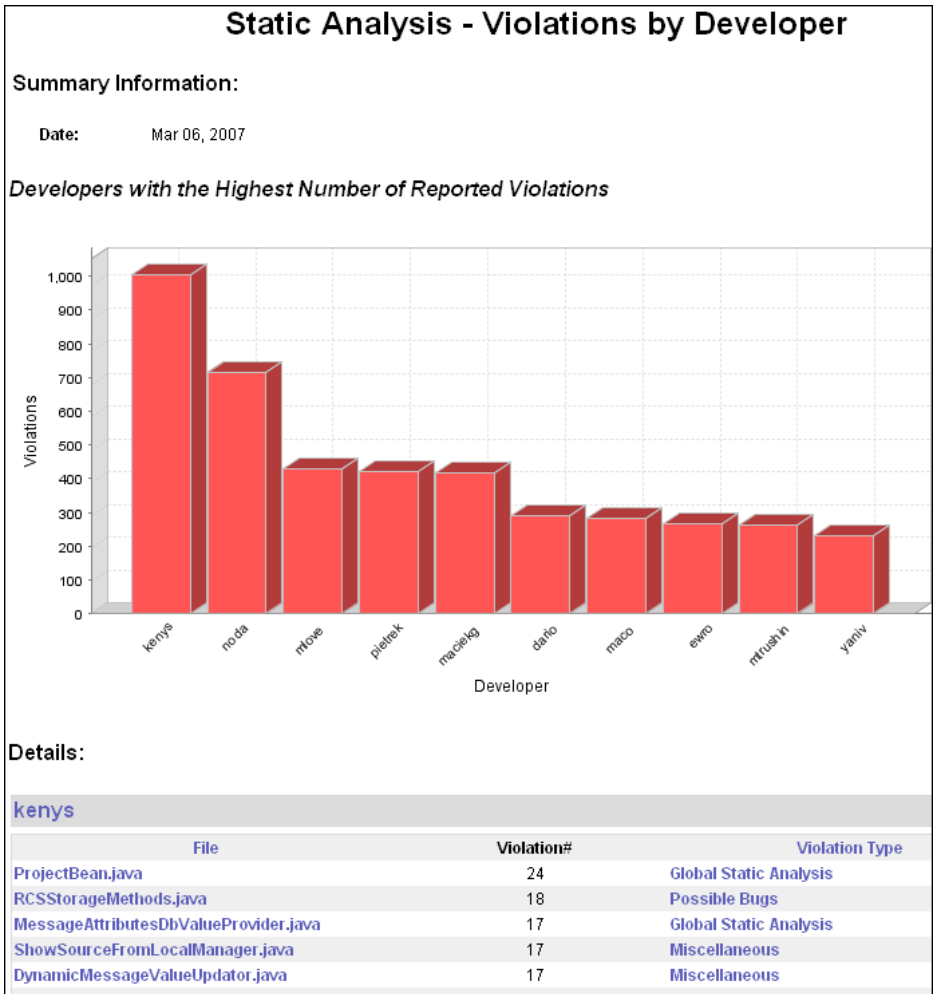
How do I get here? *Practices> Static Analysis> Violations> Table> "Files failed" column> [number of files]*

Click on any link in the table to change views and access additional information:

- **Violation Type** column header: Switch to By Violation Type view.
- **Developer(s)** column header: Switch to By Developer view.
- **[Violation type name]**: Opens the Static Analysis Violations Details by Violation Type page. See "For Violation Type", page 41 for details.
- **[File name]**: Opens the Static Analysis Violations Details by File page. See "For File", page 42 for details.
- **[Developer name]**: Opens the Static Analysis Violations Details by Developer page. See "For Developer", page 42 for details.

By Developers

Each bar in the Static Analysis Violations by Developers graph represents a developer that has the highest number of reported violations; up to 10 developers are shown.



How do I get here? *Practices> Static Analysis> Violations> Table> "Files failed" or "Violations" column> [number of files] or [number of violations]> Developer(s)*

Click on any link in the table to change views and access additional information:

- **Violation Type** column header: Switch to By Violation Type view.
- **Files** column header: Switch to By File view.
- **[Developer name]**: Opens the Static Analysis Violations Details by Developer page. See "For Developer", page 42 for details.
- **[Violation type name]**: Opens the Static Analysis Violations Details by Violation Type page. See "For Violation Type", page 41 for details.
- **[File name]**: Opens the Static Analysis Violations Details by File page. See "For File", page 42 for details.

Static Analysis Violations Details

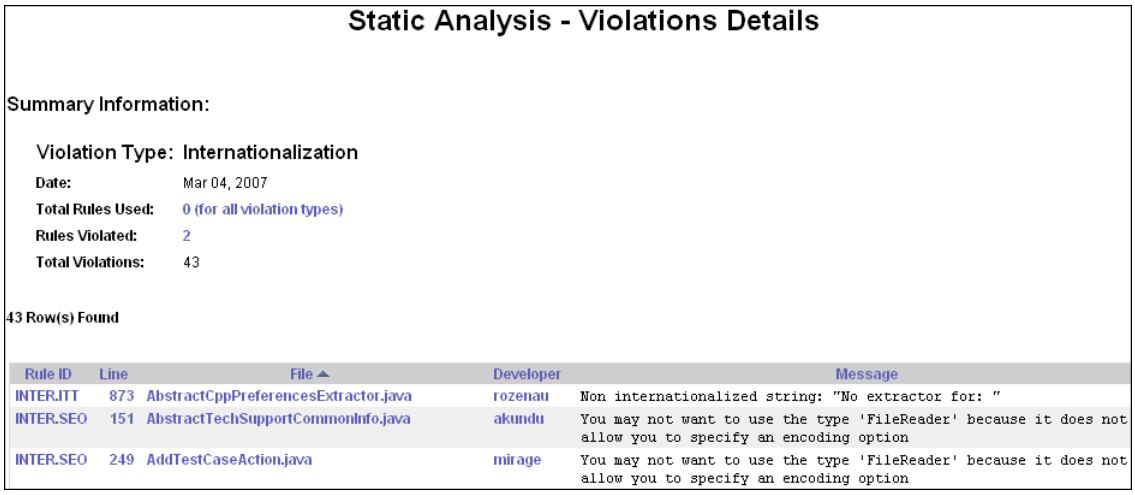
The Coding Standard Violations Details page has three different views and is displayed in one of the following ways based on the link that you click on the Static Analysis Violations by Type page:

- For Violation Type: Shows the number of rules used, violated and total number of violations for the selected violation type on the selected date.
- For File: Shows the number of rules used, violated and total number of violations for the selected file on the selected date.
- For Developer: Shows the number of rules used, violated and total number of violations for the selected developer on the selected date.

For Violation Type

Coding Standard Violations Details for violation type is displayed when you click the name of violation type on the Static Analysis Violations by Type page.

Figure 2: Static Analysis - Violations Details for Violation Type



Static Analysis - Violations Details

Summary Information:

Violation Type: Internationalization

Date: Mar 04, 2007

Total Rules Used: 0 (for all violation types)

Rules Violated: 2

Total Violations: 43

43 Row(s) Found

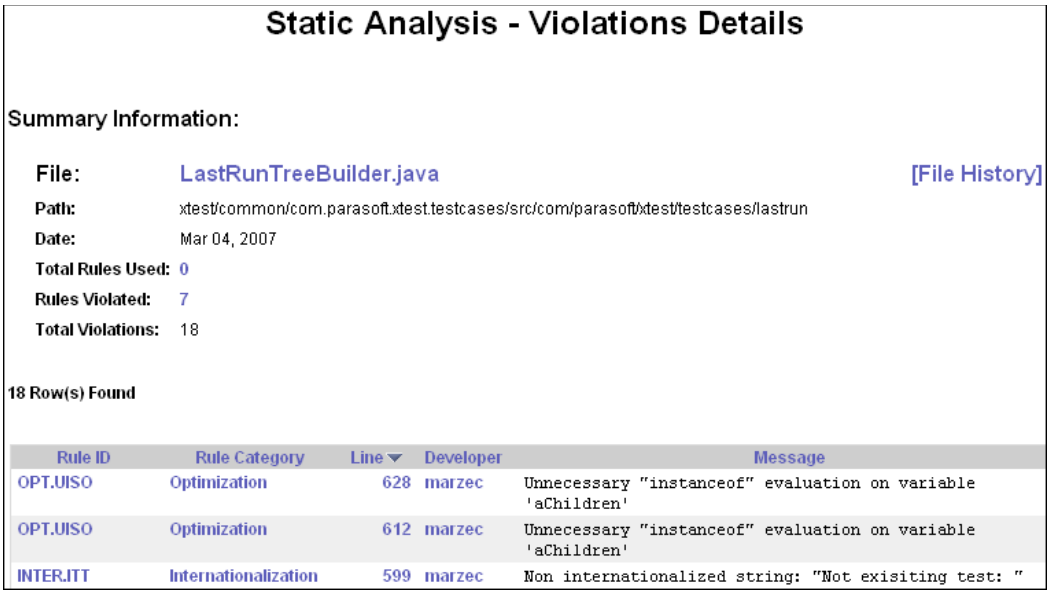
Rule ID	Line	File	Developer	Message
INTER.ITT	873	AbstractCppPreferencesExtractor.java	rozenau	Non internationalized string: "No extractor for: "
INTER.SEO	151	AbstractTechSupportCommonInfo.java	akundu	You may not want to use the type 'FileReader' because it does not allow you to specify an encoding option
INTER.SEO	249	AddTestCaseAction.java	mirage	You may not want to use the type 'FileReader' because it does not allow you to specify an encoding option

How do I get here? *Practices > Static Analysis > Violations > [number of violations] (in table) > [violation type name] (section headings in table, for example: Maintainability, Global Static Analysis, Code Duplication Detection, and the like)*

For File

Coding Standard Violations Details for file is displayed when you click the name of file on the Static Analysis Violations by Type page.

Figure 3: Static Analysis - Violations Details for File



Static Analysis - Violations Details

Summary Information:

File: LastRunTreeBuilder.java [File History]
Path: xtest/common/com.parasoft.xtest.testcases/src/com/parasoft/xtest/testcases/lastrun
Date: Mar 04, 2007
Total Rules Used: 0
Rules Violated: 7
Total Violations: 18

18 Row(s) Found

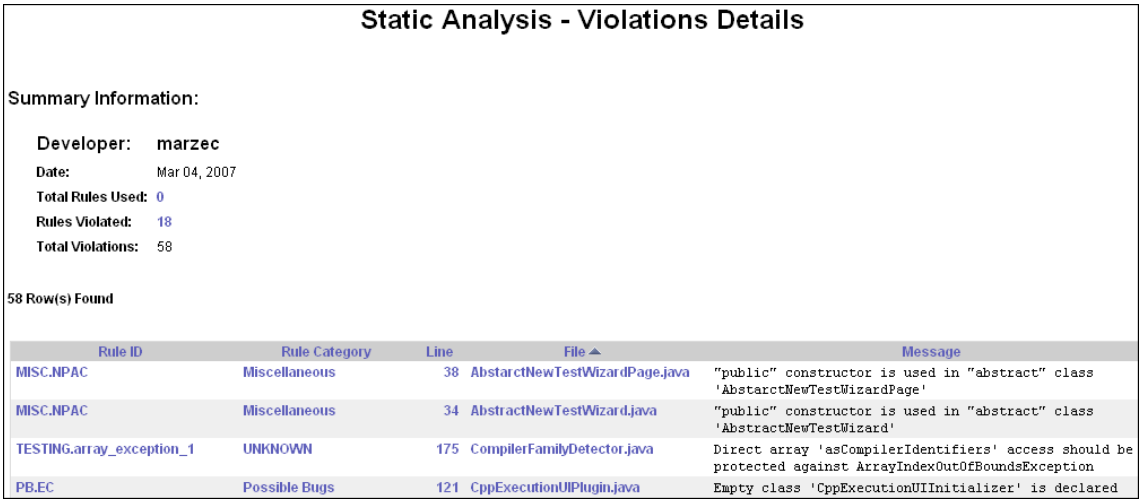
Rule ID	Rule Category	Line	Developer	Message
OPT.UISO	Optimization	628	marzec	Unnecessary "instanceof" evaluation on variable 'aChildren'
OPT.UISO	Optimization	612	marzec	Unnecessary "instanceof" evaluation on variable 'aChildren'
INTER.ITT	Internationalization	599	marzec	Non internationalized string: "Not exisiting test: "

How do I get here? *Practices > Static Analysis > Violations > Violations > [number of violations] (in table) > [file name]*

For Developer

Coding Standard Violations Details for developer (Figure 4) is displayed when you click the name of the developer on the Static Analysis Violations by Type page.

Figure 4: Static Analysis - Violations Details for Developer



Static Analysis - Violations Details

Summary Information:

Developer: marzec
Date: Mar 04, 2007
Total Rules Used: 0
Rules Violated: 18
Total Violations: 58

58 Row(s) Found

Rule ID	Rule Category	Line	File	Message
MISC.NPAC	Miscellaneous	38	AbstarctNewTestWizardPage.java	"public" constructor is used in "abstract" class 'AbstarctNewTestWizardPage'
MISC.NPAC	Miscellaneous	34	AbstractNewTestWizard.java	"public" constructor is used in "abstract" class 'AbstractNewTestWizard'
TESTING.array_exception_1	UNKNOWN	175	CompilerFamilyDetector.java	Direct array 'asCompilerIdentifiers' access should be protected against ArrayIndexOutOfBoundsException
PB.EC	Possible Bugs	121	CppExecutionUIPlugin.java	Empty class 'CppExecutionUIInitializer' is declared

How do I get here? *Practices > Static Analysis > Violations > Violations > [number of violations] (in table) > [developer name]*

The Static Analysis Violations Details reports show the following information:

Summary Section:

- **Violation Type:** (Applicable to Violation Type view.) Type of rule that was violated.
- **File:** (Applicable to File view.) Name of the file that the report concerns. Click the file name link to view the source code of the file.
- **Path:** (Applicable to File view.) Folder location of the file that the report concerns.
- **Developer:** (Applicable to Developer view.) Name of the developer who is responsible for the listed violation.
- **Date:** Date on which tests were run and the listed violations were found.
- **Total rules used:** Total number of rules used in the tests that were run. Click to open the Static Analysis Rules Used report, which lists all rule IDs. See “Static Analysis Rules Used”, page 43.
- **Rules violated:** Number of rules that were violated. Click to drill down to the Static Analysis Rules Used report that lists violated rules ID.
- **Total violations:** Total number of violations detected by Static Analysis tests.
- **File History:** (Applicable to File view.) Click to open the File Details page.
- **Message:** Error message of each violation.

Table Section:

- **Rule ID:** Name of the rule. Click to view a rule description displayed in the browser window.
- **Line:** (Applicable to Violation Type and File views.) Line of the code where the violation was detected. Click to view the source of the file with the violated code line highlighted.
- **File:** (Applicable to Violation Type and Developer views.) Name of the file that contain violations. Click to open the source code
- **Developer:** (Applicable to Violation Type and File views.) Name of the developer responsible for the error. Click the name to view the statistics of that developer.
- **Rule Category:** (Applicable to Developer and File views.) Click to see the Rule Category violations statistic.

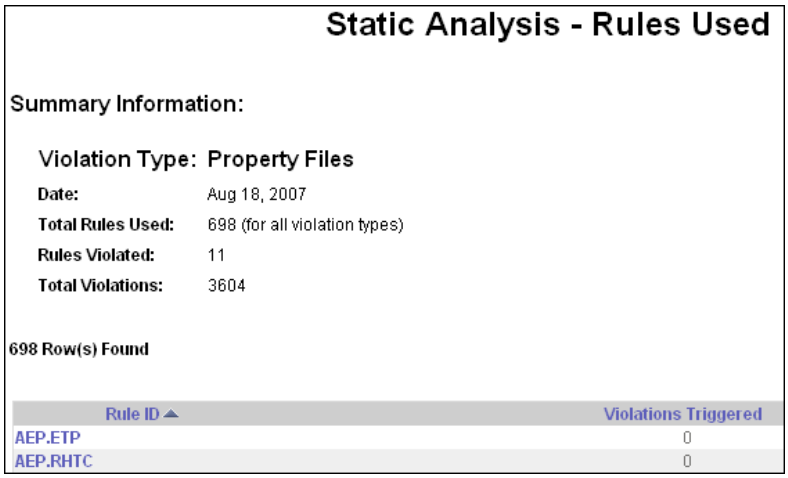
To change the order by which the statistics in the table are displayed, click any of the table headers.

Note: *Source code of a specific file, as well as file history, becomes available to view after SourceScanner is run.*

Static Analysis Rules Used

The Static Analysis Rules Used page (Figure 5) lists all of the rules that were tested on the specified date. The total number of rules used is shown in the report header.

Figure 5: Static Analysis Rules Used



How do I get here? *Static Analysis - Violations Details (any view) > [Total Rules Used]*

Click any of the listed rule IDs to open a popup page that lists information such as note, security relevance, parameters, and the like, for that specific coding standard.

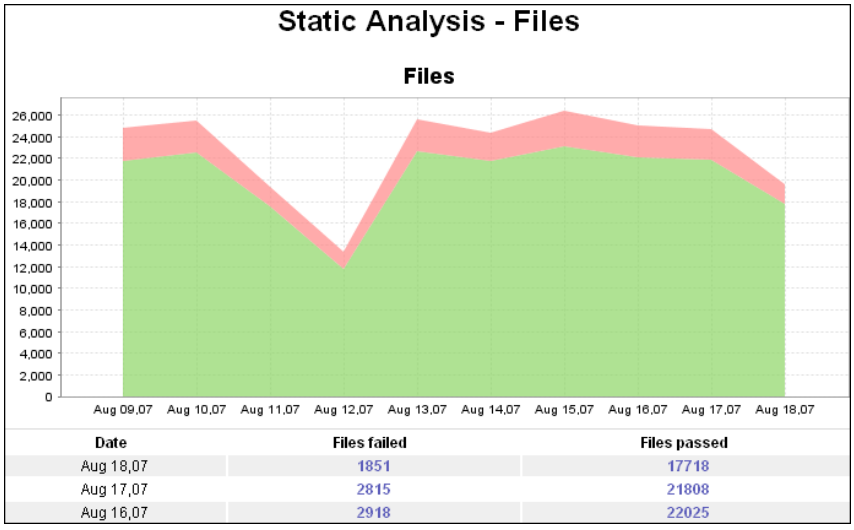
Suppressions for Static Analysis

The Suppressions for Static Analysis page displays information such as the number of rules that are suppressed for the selected project and the number of suppressed rule violations occurred during testing. For details see “Suppressions for Static Analysis”, page 150.

Static Analysis Files

The Static Analysis Files report (Figure 6) shows the number of files that failed static analysis tests and the number that passed during the specified period of time. As any given project progresses, its tendency to adhere to static analysis or not is evident over time in the Static Analysis Files report.

Figure 6: Static Analysis Files



How do I get here? *Practices > Static Analysis > Violations > Files*

The Static Analysis Files table is located beneath the graph. For each drop date within the specified date range, it lists the number of files that contain at least one violation (files failed) and the number of files that passed without any violations detected (files passed).

Click a statistic listed beneath either of the following columns:

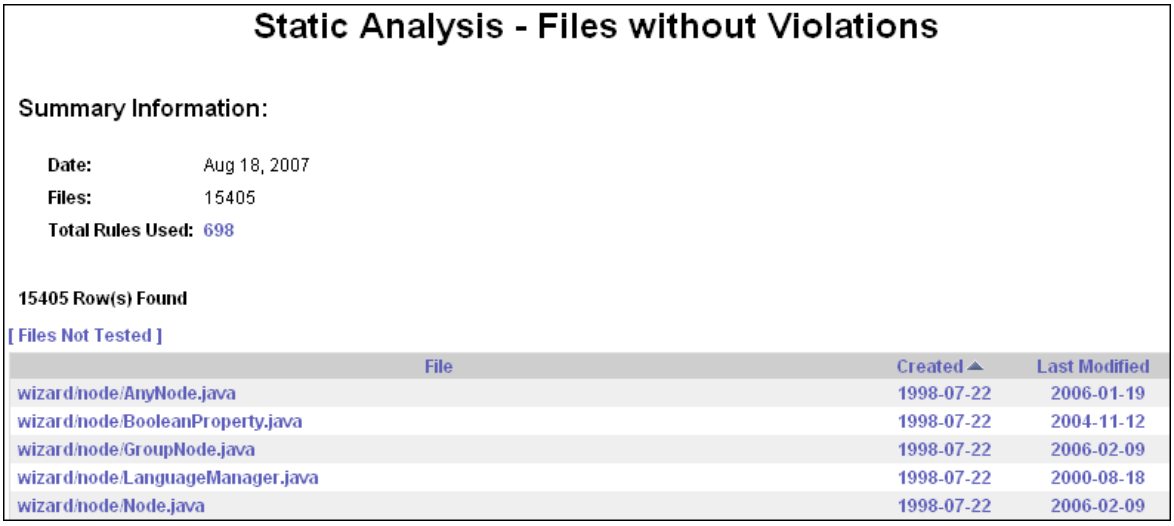
- **Files failed:** Opens the Static Analysis Violations by File page. See "By File", page 39 for details.
- **Files passed:** Opens the Files without Violations page. See "Files without Violations", page 45 for details.

Files without Violations

The Files without Violations report (Figure 7) lists all files that passed static analysis tests. With this report you can answer the following questions:

- What rules were involved in tests?
- What file passed tests without any violations?
- Who/When was the file created?
- Who/When was the file modified?

Figure 7: Files without Violations



How do I get here? *Practices > Static Analysis > Violations > Files > [number of files passed] (in table)*

The Files without Violations page consists of two parts:

Summary Information

Date: Date on which tests were run and all listed files passed.

Files: Number of files that passed the tests run on the listed date.

Total rules used: Total number of rules used in the tests that were run. Click to open the Static Analysis Rules Used report, which lists all rule IDs. See "Static Analysis Rules Used", page 43.

Files Not Tested: Click to open the Files Not Tested page, which lists all files that were not tested during the listed date.

Detailed Information

File: Lists path location and name of each file that passed the tests run on the listed date. Click to open the Static Analysis Violations Details page. See “Static Analysis Violations Details”, page 41 for details.

Created: Date on which the listed file was created.

Last Modified: Date on which the listed file was last changed.

To change the order by which the statistics in the table are displayed, click any of the table headers.

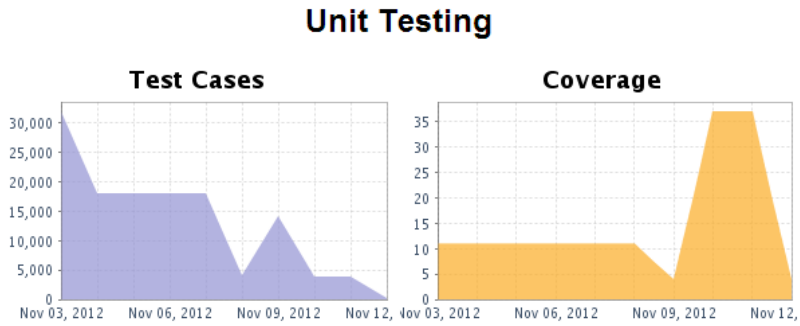
Note: *Source code of a specific file, as well as file history, becomes available to view after SourceScanner is run.*

Unit Tests

Unit Testing reports combine the statistics of unit testing results and display them in three reports:

- Unit Testing
- Unit Testing Test Cases
- Unit Testing Coverage

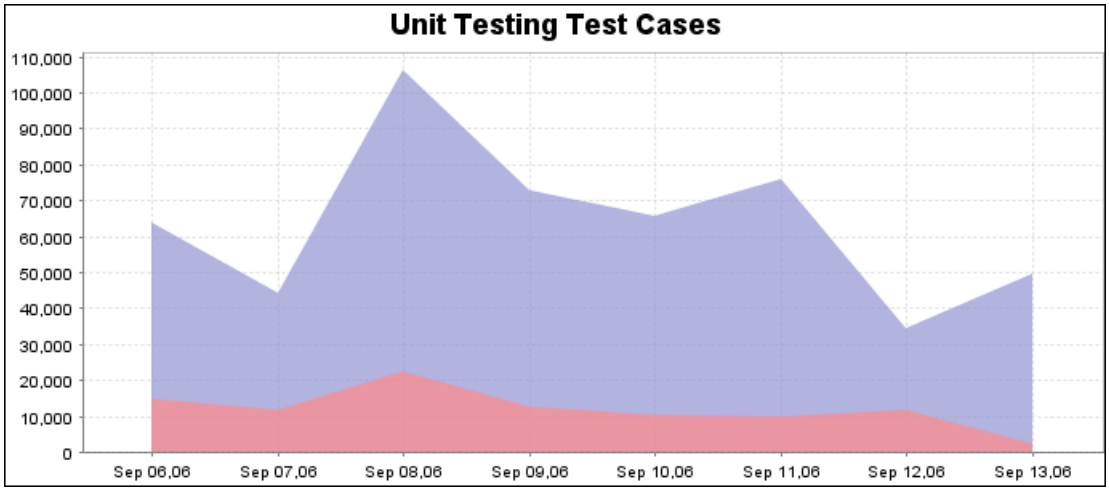
Figure 8: Unit Testing



How do I get here? *Practices > Unit Testing*

Unit Testing Test Cases

The Unit Testing Test Cases graph shows the total number of test cases and the total number of test cases that failed for all unit tests. Over time, you want to see that the number of test cases that run are increasing, while the failures are decreasing. There should be zero failures in the test cases.



How do I get here? *Practices > Unit Testing > Test Cases*

The Unit Testing Test Cases table is displayed beneath the Unit Testing Test Cases graph and shows the total number of test cases and the number of failed test cases for each listed drop date.

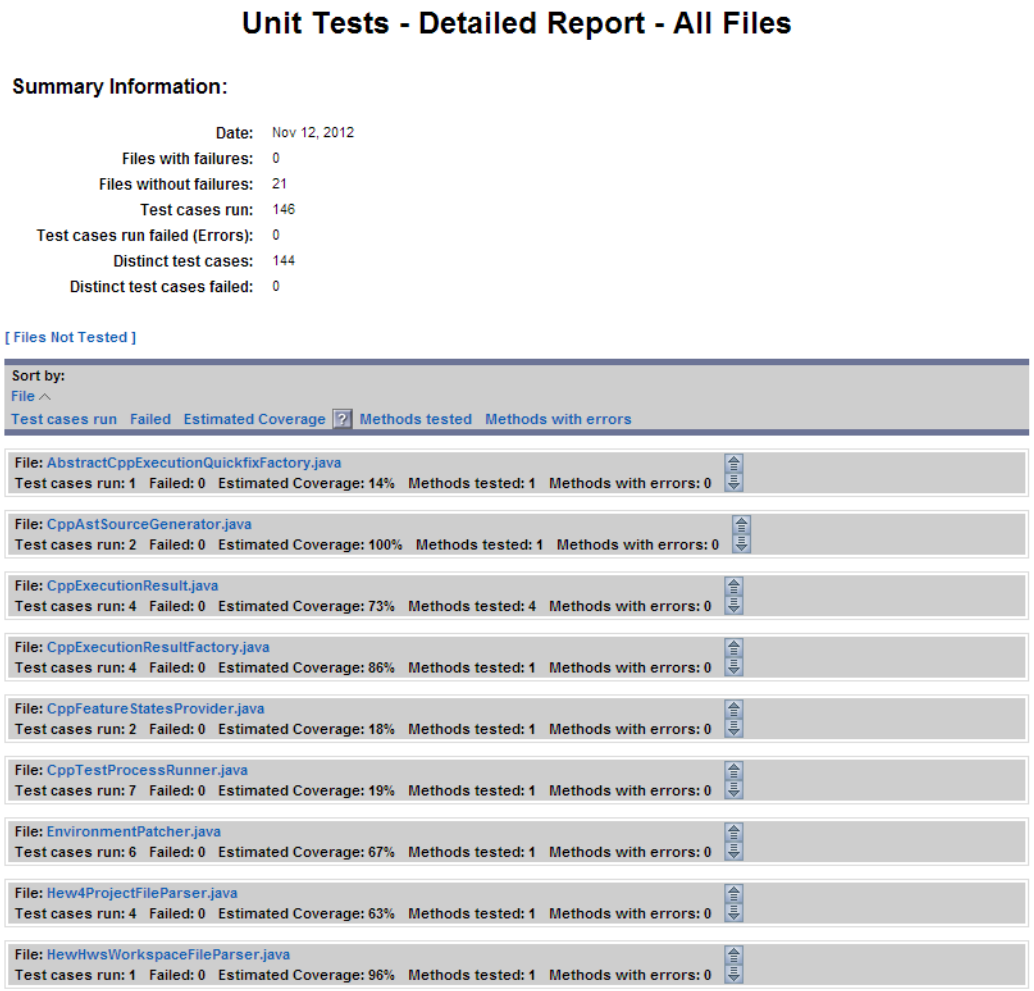
Date	Errors	Total
Sep 13,06	2017	49679
Sep 12,06	11598	34454
Sep 11,06	9706	75915
Sep 10,06	10082	65754
Sep 09,06	12439	72840
Sep 08,06	22278	106091
Sep 07,06	11583	44335
Sep 06,06	14632	63829

Click on a value in the Total column to open the Unit Tests - Detailed Report - All Files report.

Click on a value in the Errors column to open the Unit Tests - Detailed Report - Files Failed report.

Unit Tests - Detailed Report - All Files

The Unit Tests - Detailed Report - All Files page provides a quick overview of unit test results and presents a per file list of test cases ran, failures, estimated coverage, methods tested, methods with errors..

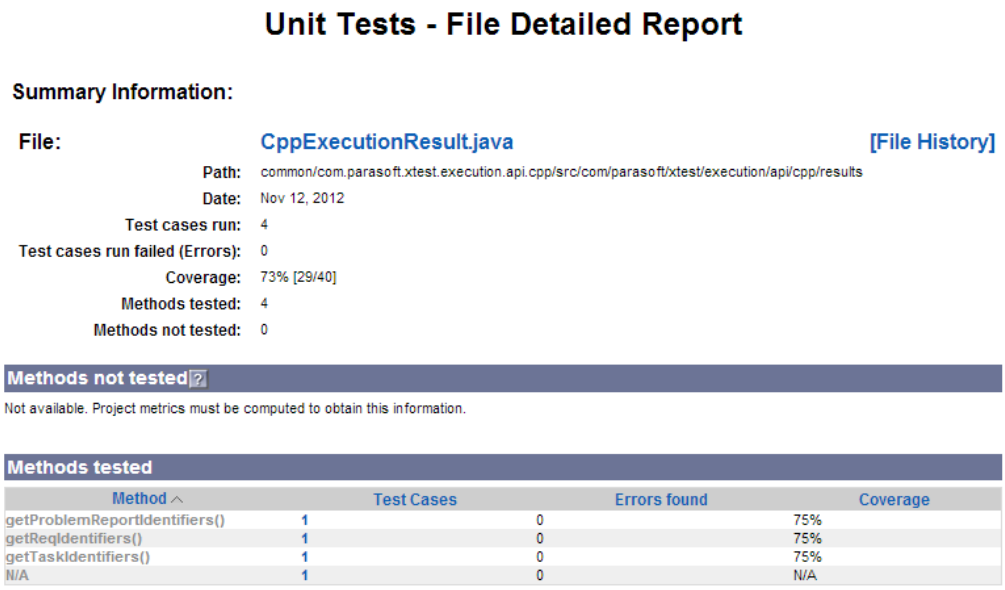


How do I get here? *Practices > Unit Testing > Test Cases> [value in the Total column of chart]*

Click on a file to open the This report provides detailed information about a specific file.

Unit Tests - File Detailed Report

This report provides detailed information about a specific file.



Unit Tests - File Detailed Report

Summary Information:

File:	**CppExecutionResult.java** **[File History]**
Path:	common/com.parasoft.xtest.execution.api.cpp/src/com/parasoft/xtest/execution/api/cpp/results
Date:	Nov 12, 2012
Test cases run:	4
Test cases run failed (Errors):	0
Coverage:	73% [29/40]
Methods tested:	4
Methods not tested:	0

Methods not tested

Not available. Project metrics must be computed to obtain this information.

Methods tested

Method	Test Cases	Errors found	Coverage
getProblemReportIdentifiers()	1	0	75%
getReqIdentifiers()	1	0	75%
getTaskIdentifiers()	1	0	75%
N/A	1	0	N/A

How do I get here?

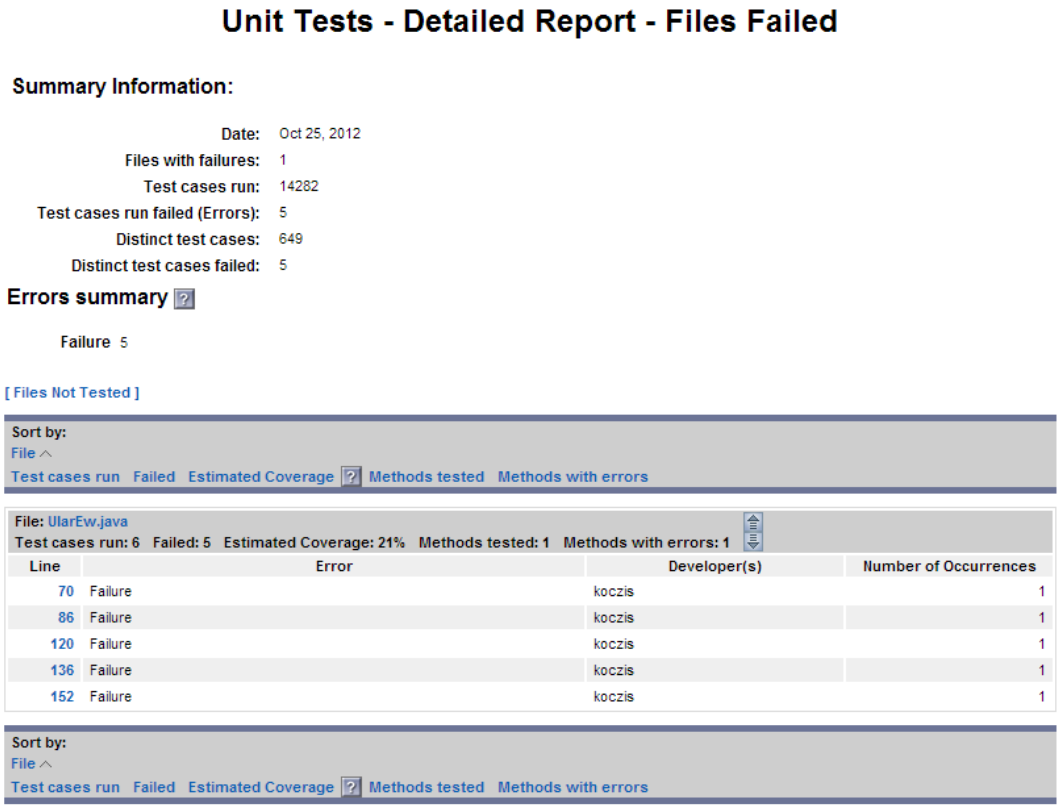
Practices > Unit Testing > Test Cases> [value in the Total column of chart]> [unit test file name]

Click on on the file name to view the source code

Click on value in the **Method** column to open test case detail.

Unit Tests - Detailed Report - Files Failed

This report provids an overview of files that have failed under unit testing.



Unit Tests - Detailed Report - Files Failed

Summary Information:

Date: Oct 25, 2012
Files with failures: 1
Test cases run: 14282
Test cases run failed (Errors): 5
Distinct test cases: 649
Distinct test cases failed: 5

Errors summary

Failure 5

[Files Not Tested]

Sort by:
File
Test cases run Failed Estimated Coverage Methods tested Methods with errors

File: UlarEw.java
Test cases run: 6 Failed: 5 Estimated Coverage: 21% Methods tested: 1 Methods with errors: 1

Line	Error	Developer(s)	Number of Occurrences
70	Failure	koczis	1
86	Failure	koczis	1
120	Failure	koczis	1
136	Failure	koczis	1
152	Failure	koczis	1

Sort by:
File
Test cases run Failed Estimated Coverage Methods tested Methods with errors

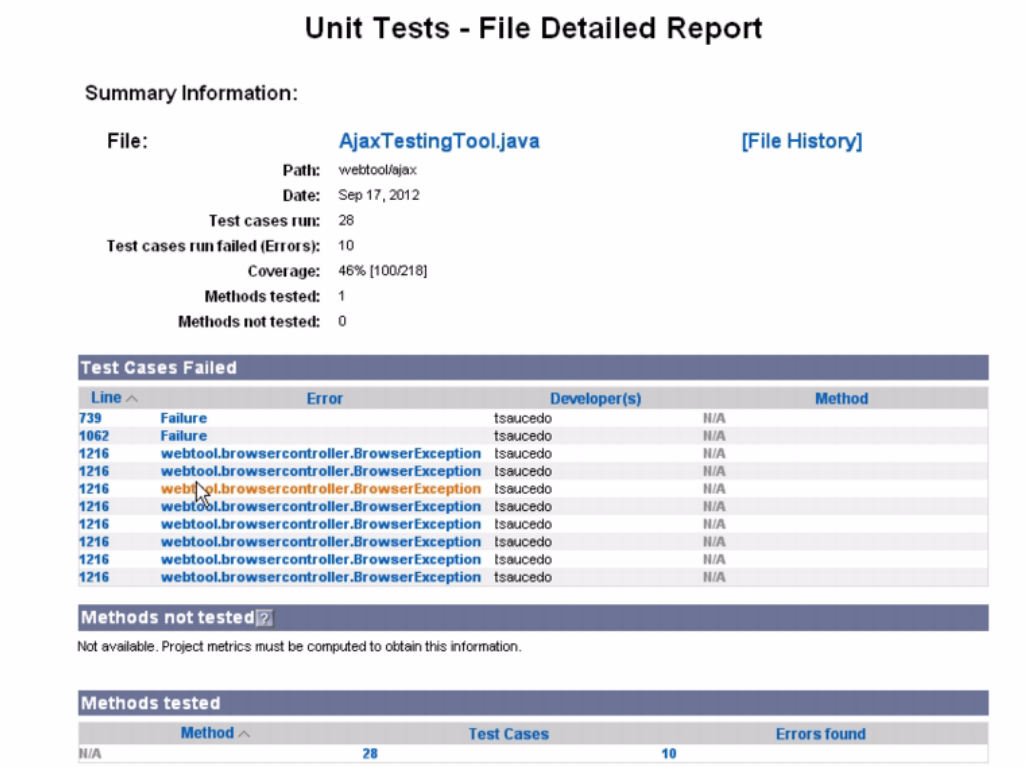
How do I get here? *Practices > Unit Testing > Test Cases > [number of errors] (from graph)*

Click on a file name to open its Unit Tests - File Detailed Report [Failed].

Click on a vaule in the Line column to view the source code.

Unit Tests - File Detailed Report *[Failed]*

This report provides detailed information about a specific file with failed unit tests. The information about files in this report is identical to the Unit Tests - File Detailed Report.



Unit Tests - File Detailed Report

Summary Information:

File: AjaxTestingTool.java [File History]

Path: webtool/ajax
Date: Sep 17, 2012
Test cases run: 28
Test cases run failed (Errors): 10
Coverage: 46% [100/218]
Methods tested: 1
Methods not tested: 0

Test Cases Failed

Line	Error	Developer(s)	Method
739	Failure	tsaucedo	N/A
1062	Failure	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A
1216	webtool.browsercontroller.BrowserException	tsaucedo	N/A

Methods not tested

Not available. Project metrics must be computed to obtain this information.

Methods tested

Method	Test Cases	Errors found
N/A	28	10

How do I get here?

Option 1: Practices > Unit Testing > Test Cases > [total number of tests] (from graph or table) > [file name] (from tables displayed at bottom of Unit Tests - Detailed Report - Files Failed page)

Option 2: Practices > Unit Testing > Test Cases > [number of errors] (from table) > [file name] (from tables displayed at bottom of Unit Tests - Detailed Report - Files Failed page)

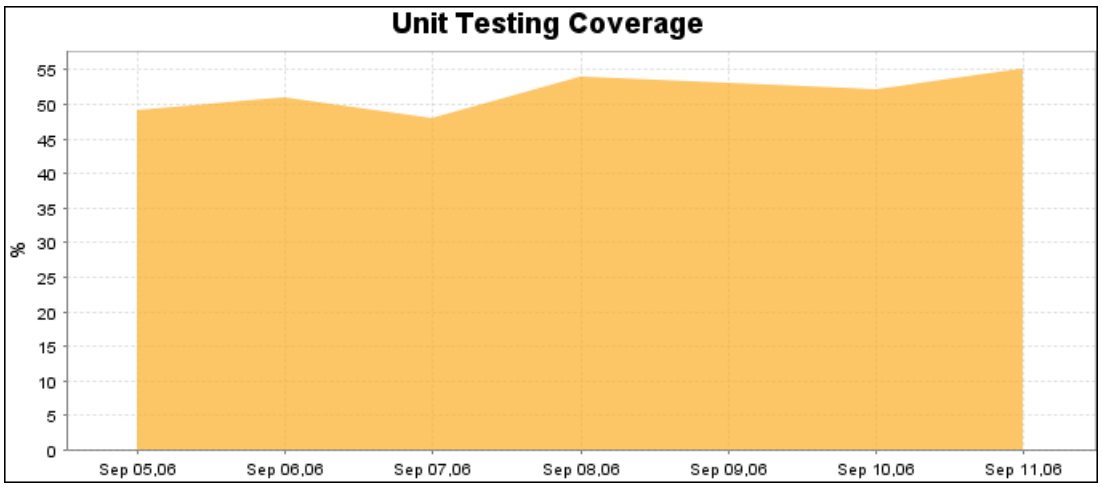
Unit Testing Coverage

The coverage report helps monitor the progress of unit testing covereage over time. Ideally, the percentage should increase; no change is an indicator that test are not sufficiently being created. The Unit Testing Coverage graph shows the following information:

- How much code was actually tested.
- Unit testing coverage of the selected project files.
- A count of tested units and units that still need to be tested

Only line coverage is represented in the graph. MC/DC, branch coverage, and other forms are not shown. Coverage is represented as percentage of succesfully tested lines of code to the total lines of selected source code.

.



How do I get here? *Practices > Unit Testing> Coverage*

The Unit Testing Coverage table is displayed beneath the Unit Testing Coverage graph and shows the percentage of coverage for each listed drop date. In parenthesis, it shows the total number of tested units as well as the number of units left to test.

Date	%Coverage	Tested Units	Units To Test
Sep 11,06	55	192792	351140
Sep 10,06	52	168555	321344
Sep 09,06	53	177477	335353
Sep 08,06	54	297569	555312
Sep 07,06	48	104454	217292
Sep 06,06	51	114202	225448
Sep 05,06	49	234995	475641

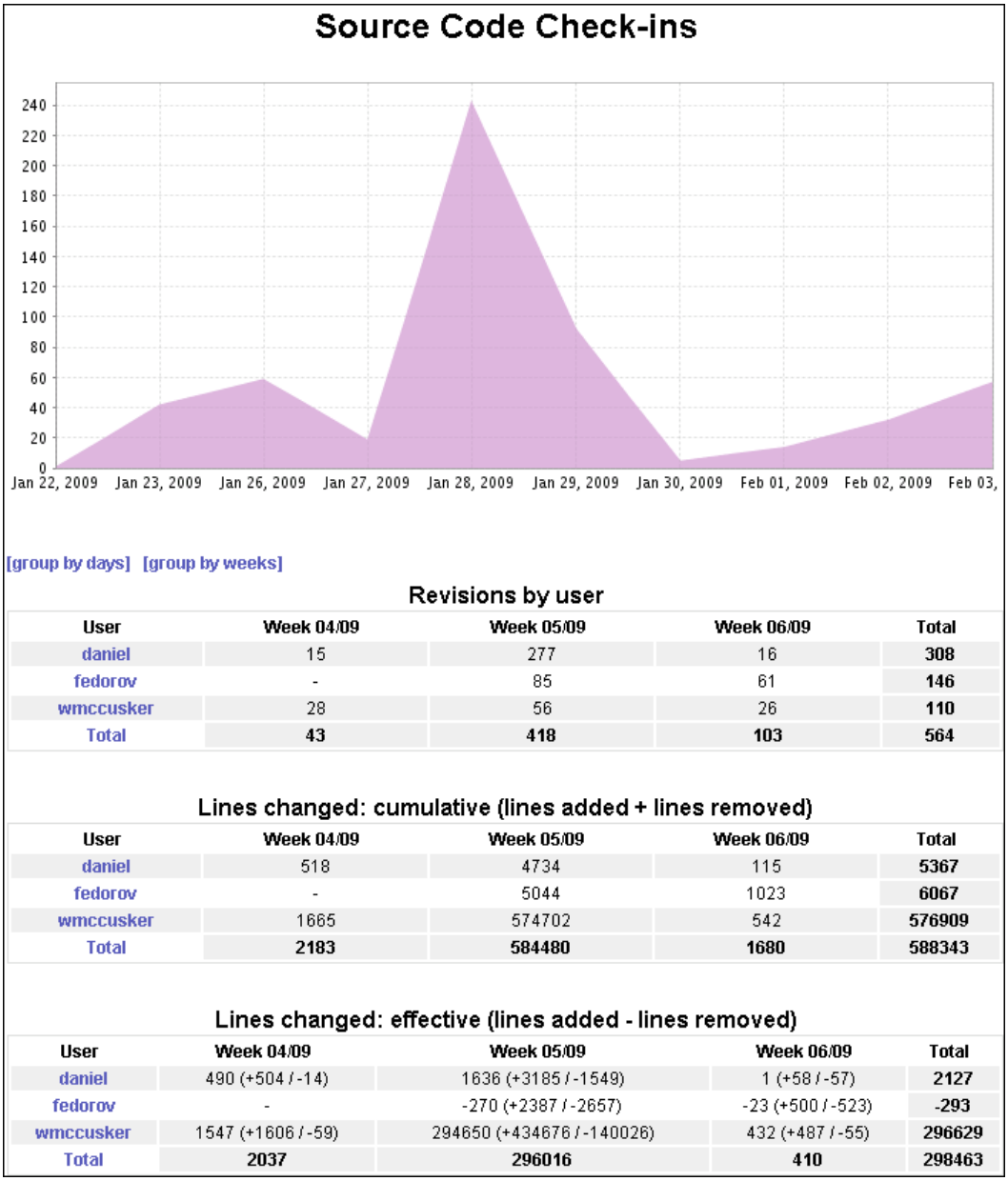
From Unit Testing Coverage, you can click on any date in the graph or table to drill down and view details about files included in unit tests in Unit Tests - Detailed Report - All Files.

Unit Tests - Detailed Report - All Files

You can click on any test case listed in Unit Tests - Detailed Report - All Files to drill down and view details about files included in unit tests. See “Unit Tests - File Detailed Report [Failed]” for details about the data displayed on this page.

Source Code Check-ins

In the Reports view, choose **Practices> Source Code Check-ins** to open the Source Code Check-ins report. This report shows the number of revisions that were made to source control files associated with a specified project during each listed day of the specified date range..



Source Code Check-ins

[group by days] [group by weeks]

Revisions by user

User	Week 04/09	Week 05/09	Week 06/09	Total
daniel	15	277	16	**308**
fedorov	-	85	61	**146**
wmccusker	28	56	26	**110**
Total	**43**	**418**	**103**	**564**

Lines changed: cumulative (lines added + lines removed)

User	Week 04/09	Week 05/09	Week 06/09	Total
daniel	518	4734	115	**5367**
fedorov	-	5044	1023	**6067**
wmccusker	1665	574702	542	**576909**
Total	**2183**	**584480**	**1680**	**588343**

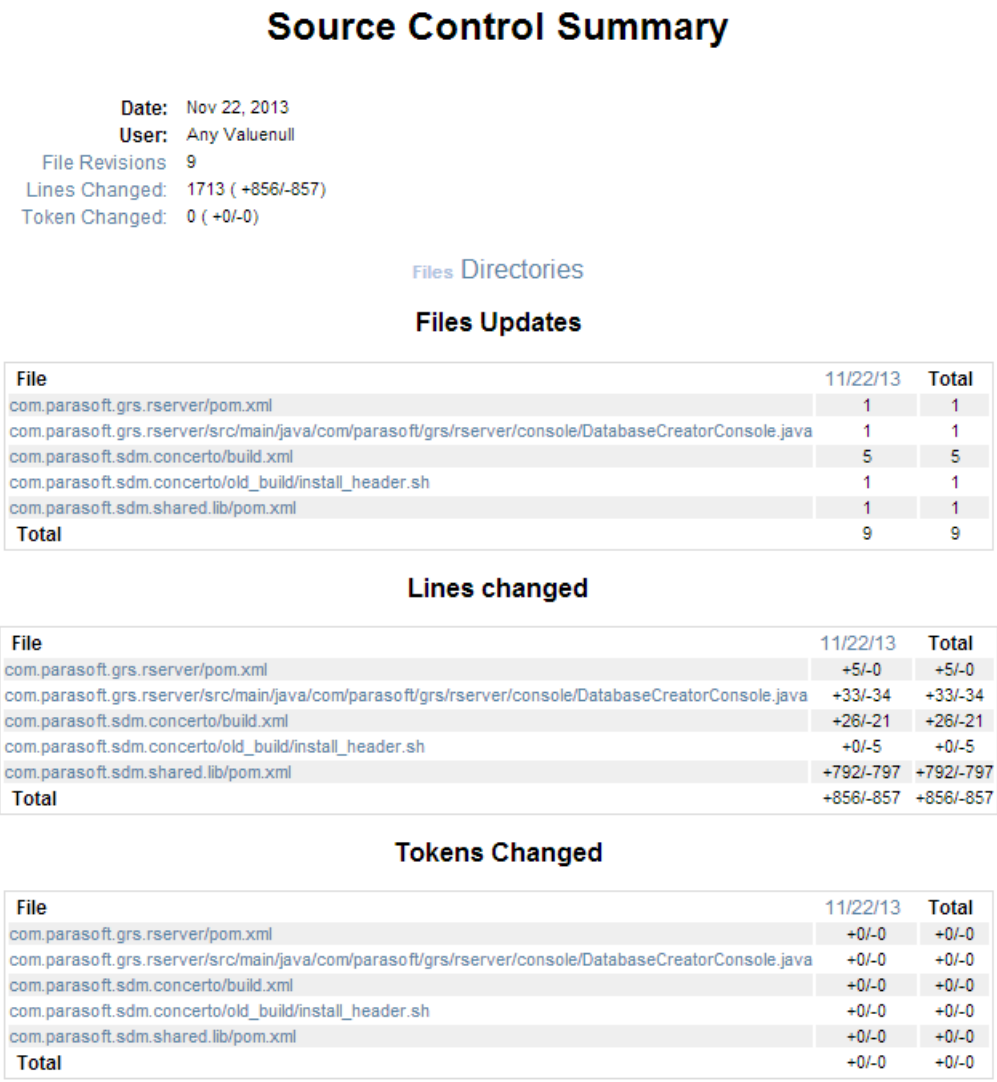
Lines changed: effective (lines added - lines removed)

User	Week 04/09	Week 05/09	Week 06/09	Total
daniel	490 (+504 / -14)	1636 (+3185 / -1549)	1 (+58 / -57)	**2127**
fedorov	-	-270 (+2387 / -2657)	-23 (+500 / -523)	**-293**
wmccusker	1547 (+1606 / -59)	294650 (+434676 / -140026)	432 (+487 / -55)	**296629**
Total	**2037**	**296016**	**410**	**298463**

- Click on an area in the Source Code Check-ins graph for a specific date to open the source control summary for all files updated on that date.
- Click the **[group by days]** link to view revisions for each user sorted and listed by date.
- Click the **[group by weeks]** link to view revisions for each user grouped and listed by week based on the the dates shown in the graph.
- Click on a user name to open the source control summary for all files updated by that user.

Source Control Summary

The source control summary shows the files and directories changed in source control for the selected date.



Source Control Summary

Date: Nov 22, 2013
User: Any Valuenull
File Revisions 9
Lines Changed: 1713 (+856/-857)
Token Changed: 0 (+0/-0)

Files Directories

Files Updates

File	11/22/13	Total
com.parasoft.grs.rserver/pom.xml	1	1
com.parasoft.grs.rserver/src/main/java/com/parasoft/grs/rserver/console/DatabaseCreatorConsole.java	1	1
com.parasoft.sdm.concerto/build.xml	5	5
com.parasoft.sdm.concerto/old_build/install_header.sh	1	1
com.parasoft.sdm.shared.lib/pom.xml	1	1
Total	9	9

Lines changed

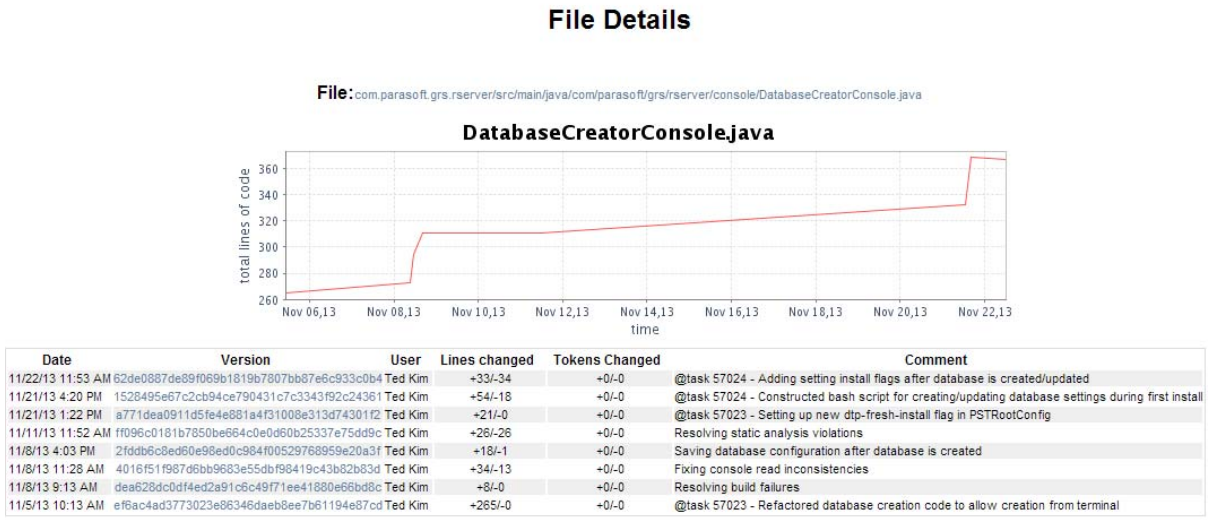
File	11/22/13	Total
com.parasoft.grs.rserver/pom.xml	+5/-0	+5/-0
com.parasoft.grs.rserver/src/main/java/com/parasoft/grs/rserver/console/DatabaseCreatorConsole.java	+33/-34	+33/-34
com.parasoft.sdm.concerto/build.xml	+26/-21	+26/-21
com.parasoft.sdm.concerto/old_build/install_header.sh	+0/-5	+0/-5
com.parasoft.sdm.shared.lib/pom.xml	+792/-797	+792/-797
Total	+856/-857	+856/-857

Tokens Changed

File	11/22/13	Total
com.parasoft.grs.rserver/pom.xml	+0/-0	+0/-0
com.parasoft.grs.rserver/src/main/java/com/parasoft/grs/rserver/console/DatabaseCreatorConsole.java	+0/-0	+0/-0
com.parasoft.sdm.concerto/build.xml	+0/-0	+0/-0
com.parasoft.sdm.concerto/old_build/install_header.sh	+0/-0	+0/-0
com.parasoft.sdm.shared.lib/pom.xml	+0/-0	+0/-0
Total	+0/-0	+0/-0

- Click **Files** or **Directories** to switch between file or directory view.

- In file view, click on a link in the File column to open its detail view, which shows the change in number of lines of code added for each revision.



Date	Version	User	Lines changed	Tokens Changed	Comment
11/22/13 11:53 AM	62de0887de89f069b1819b7807bb87e6c933c0b4	Ted Kim	+33/-34	+0/-0	@task 57024 - Adding setting install flags after database is created/updated
11/21/13 4:20 PM	1528495e67c2cb94ce790431c7c3343f92c24361	Ted Kim	+54/-18	+0/-0	@task 57024 - Constructed bash script for creating/updating database settings during first install
11/21/13 1:22 PM	a771dea0911d5fe4e881a4f31008e313d74301f2	Ted Kim	+21/-0	+0/-0	@task 57023 - Setting up new dtp-fresh-install flag in PSTRootConfig
11/11/13 11:52 AM	ff096c0181b7850be664c0e0d60b25337e75dd9c	Ted Kim	+26/-26	+0/-0	Resolving static analysis violations
11/8/13 4:03 PM	2fddb6c8ed60e98ed0c984f00529768959e20a3f	Ted Kim	+18/-1	+0/-0	Saving database configuration after database is created
11/8/13 11:28 AM	4016f51f987d6bb9683e55dbf98419c43b82b83d	Ted Kim	+34/-13	+0/-0	Fixing console read inconsistencies
11/8/13 9:13 AM	dea628dc0df4ed2a91c6c49f71ee41880e66bd8c	Ted Kim	+8/-0	+0/-0	Resolving build failures
11/5/13 10:13 AM	ef6ac4ad3773023e86346daeb8ee7b61194e87cd	Ted Kim	+265/-0	+0/-0	@task 57023 - Refactored database creation code to allow creation from terminal

- In directory view, click on a link the Directory column to open a source control summary report for the selected directory.

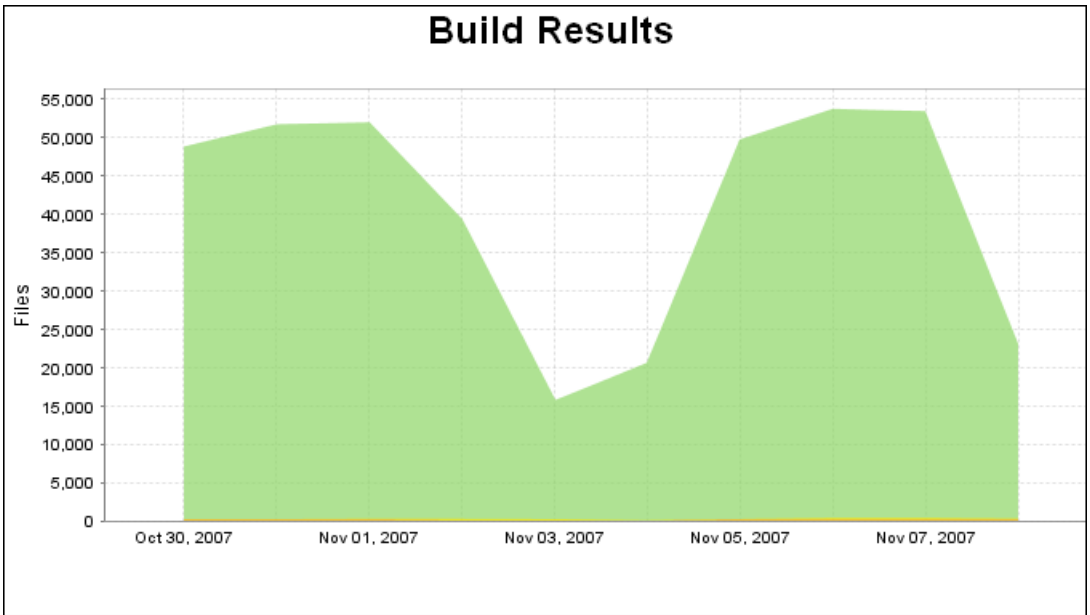
Builds

The Build Monitor tracks compilation and build processes, analyzes compilation output, and reports results to Report Center. Such data is reported by Report Center on demand so that users can verify final results of build processes and uncover exactly where errors occurred.

Build Results

Click on the Build Results widget to open the Build Results report (see “Build Results Widgets”, page 6). This reports shows the following build information for each listed drop date:

- Number of files that failed
- Number of files that contain warnings and are incomplete
- Number of files and modules that passed t

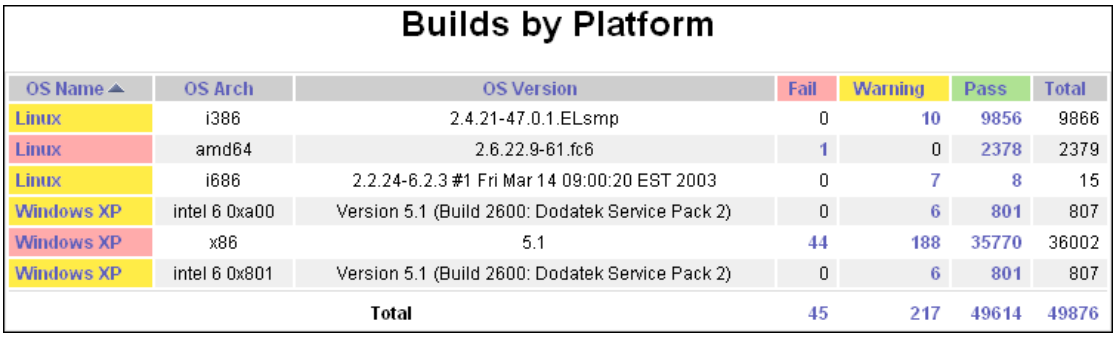


Date	Failed	Warning	Passed
Nov 08, 2007	63	283	23040
Nov 07, 2007	63	358	53291
Nov 06, 2007	63	374	53594
Nov 05, 2007	45	217	49614
Nov 04, 2007	6	62	20560

Click the appropriate date from the graph or table to drill down to the Builds by Platform report.

Builds by Platform

The Builds by Platform page lists the platforms on which the builds were run on the selected date. For each listed platform, the architecture and version number is also listed, along with the number of files that failed, contain warnings, or passed during the build process.



Builds by Platform

OS Name ▲	OS Arch	OS Version	Fail	Warning	Pass	Total
Linux	i386	2.4.21-47.0.1.ELsmp	0	10	9856	9866
Linux	amd64	2.6.22.9-61.fc6	1	0	2378	2379
Linux	i686	2.2.24-6.2.3 #1 Fri Mar 14 09:00:20 EST 2003	0	7	8	15
Windows XP	intel 6 0xa00	Version 5.1 (Build 2600: Dodatek Service Pack 2)	0	6	801	807
Windows XP	x86	5.1	44	188	35770	36002
Windows XP	intel 6 0x801	Version 5.1 (Build 2600: Dodatek Service Pack 2)	0	6	801	807
		Total	45	217	49614	49876

Following is the information displayed on the Builds by Platform page:

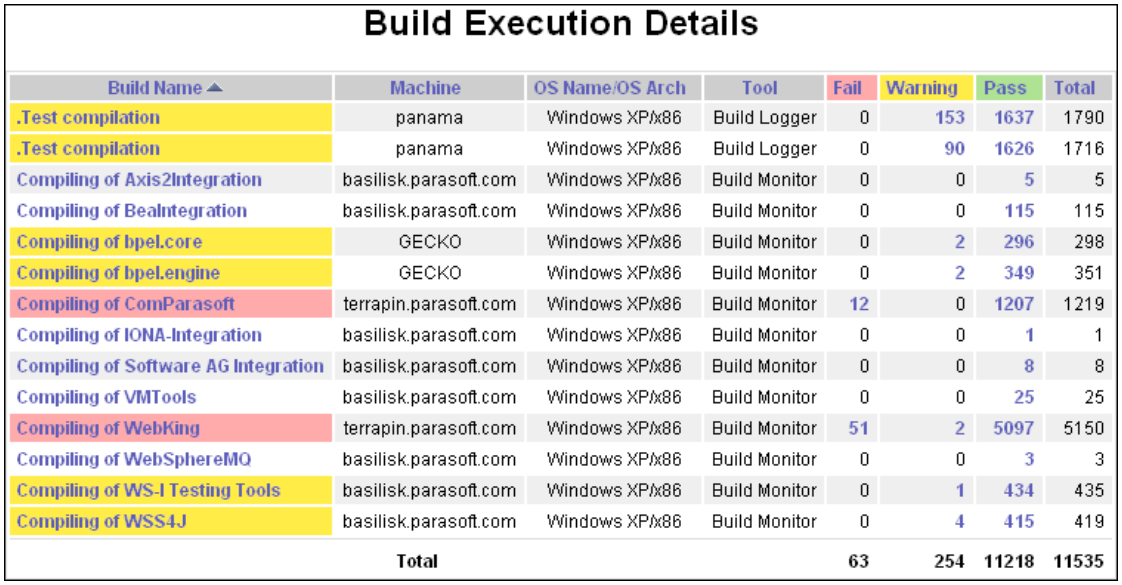
- **Project:** Name of the project, which is selected from the drop-down list, for which builds were run.
- **Date:** Date of the build selected from the Build Results report.
- **OS Name:** Name of the operating system installed on the machine that ran the build. Click to open the Build Execution Details report. See “Build Execution Details”, page 58 for details.
- **OS Arch:** Type of processor on which the build was run.
- **OS Version:** Version number of the operating system on which the build was run.
- **Fail:** Number of files that failed on the listed platform for the selected build. Click to open the Build Execution Details report. See “Build Execution Details”, page 58 for details.
- **Warning:** Number of files with warnings on the listed platform for the selected build. Click to open the Build Execution Details report. See “Build Execution Details”, page 58 for details.
- **Pass:** Number of files that passed on the listed platform for the selected build. Click to open the Build Execution Details report. See “Build Execution Details”, page 58 for details.
- **Total (column):** Total number of files included in the builds on the listed platform for the selected build.
- **Total (row):** Combined total of build files that failed, contain warnings, passed and ran on all platforms. Click to open the Build Execution Details report. See “Build Execution Details”, page 58 for details.

Click the appropriate platform (via OS Name, Fail, Warning, Pass or Total links) from the Builds by Platform table to drill down to the Build Execution Details report.

Build Execution Details

The Build Execution Details page (Figure 9) shows builds run on the selected platform on the selected date. It also shows the files included in each build. You can see the number of files in the build that failed, contain warning messages, passed, and the total number of files.

Figure 9: Builds Execution Details



Build Execution Details

Build Name ▲	Machine	OS Name/OS Arch	Tool	Fail	Warning	Pass	Total
.Test compilation	panama	Windows XP/x86	Build Logger	0	153	1637	1790
.Test compilation	panama	Windows XP/x86	Build Logger	0	90	1626	1716
Compiling of Axis2Integration	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	0	5	5
Compiling of BeaIntegration	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	0	115	115
Compiling of bpel.core	GECKO	Windows XP/x86	Build Monitor	0	2	296	298
Compiling of bpel.engine	GECKO	Windows XP/x86	Build Monitor	0	2	349	351
Compiling of ComParasoft	terrapin.parasoft.com	Windows XP/x86	Build Monitor	12	0	1207	1219
Compiling of IONA-Integration	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	0	1	1
Compiling of Software AG Integration	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	0	8	8
Compiling of VMTools	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	0	25	25
Compiling of WebKing	terrapin.parasoft.com	Windows XP/x86	Build Monitor	51	2	5097	5150
Compiling of WebSphereMQ	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	0	3	3
Compiling of WS-I Testing Tools	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	1	434	435
Compiling of WSS4J	basilisk.parasoft.com	Windows XP/x86	Build Monitor	0	4	415	419
Total				63	254	11218	11535

The following information is displayed on the Build Execution Details page:

- **Build Name:** Name of the build that was run on the selected platform on the selected date. Click to open the Build Modules Details page in a view that lists all build modules, along with files included in each module. See “Build Modules Details”, page 58 for details.
- **Machine:** Name of the machine on which the listed build was run.
- **OS Name / OS Arch:** Name of the operating system installed on the machine that ran the build. / Type of processor on which the build was run.
- **Tool:** Name of the tool used to run the build.
- **Fail:** Number of files that failed in the listed build on the selected platform. Click to open the Build Modules Details page in a view that only lists files that failed in selected build for the selected date. See “Build Modules Details”, page 58 for details.
- **Warning:** Number of files with warnings in the listed build on the selected platform. Click to open the Build Modules Details page in a view that only lists files with warnings in the selected build for the selected date. See “Build Modules Details”, page 58 for details.
- **Pass:** Number of passed in the listed build on the selected platform. Click to open the Build Modules Details page in a view that only lists files with warnings in the selected build for the selected date. See “Build Modules Details”, page 58 for details.
- **Total:** Total number of files included in the listed build on the selected platform.

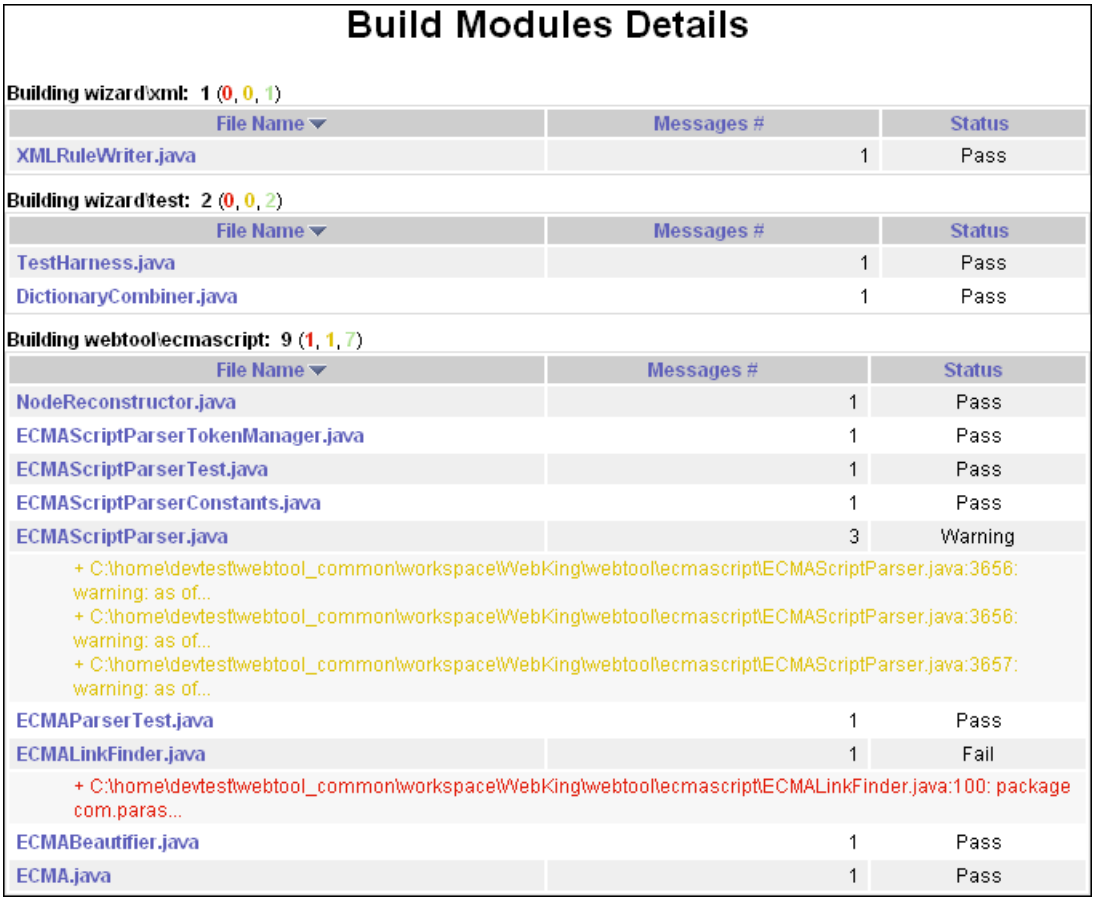
Drill-down from Build Execution Details report:

Click the appropriate execution details (via OS Name, Fail, Warning, Pass or Total links) from the Build Execution Details table to drill down to the Build Modules Details report.

Build Modules Details

The Build Modules Details report (Figure 10) lists each module include in the selected build.

Figure 10: Build Modules Details



Build Modules Details

Building wizard\xml: 1 (0, 0, 1)

File Name	Messages #	Status
XMLRuleWriter.java	1	Pass

Building wizard\test: 2 (0, 0, 2)

File Name	Messages #	Status
TestHarness.java	1	Pass
DictionaryCombiner.java	1	Pass

Building webtool\ecmascript: 9 (1, 1, 7)

File Name	Messages #	Status
NodeReconstructor.java	1	Pass
ECMAScriptParserTokenManager.java	1	Pass
ECMAScriptParserTest.java	1	Pass
ECMAScriptParserConstants.java	1	Pass
ECMAScriptParser.java	3	Warning
+ C:\home\devtest\webtool_common\workspaceWebKing\webtool\ecmascript\ECMAScriptParser.java:3656: warning: as of...		
+ C:\home\devtest\webtool_common\workspaceWebKing\webtool\ecmascript\ECMAScriptParser.java:3656: warning: as of...		
+ C:\home\devtest\webtool_common\workspaceWebKing\webtool\ecmascript\ECMAScriptParser.java:3657: warning: as of...		
ECMAParserTest.java	1	Pass
ECMALinkFinder.java	1	Fail
+ C:\home\devtest\webtool_common\workspaceWebKing\webtool\ecmascript\ECMALinkFinder.java:100: package com.paras...		
ECMABeautifier.java	1	Pass
ECMA.java	1	Pass

Let's examine the first module listed in Figure 10 to get a better understanding of the information available:

Building wizard\xml: 1 (**0**, **0**, **1**)

- **Building wizard** is the name of the directory where the module is located.
- **xml** is the module name.
- **1** represents the number of files contained within the listed module.
- (**0**, **0**, **1**) represent the number of files that failed, finished with warnings, and passed, respectively.

For each listed module, a table is displayed beneath it that lists all of the files created for it. For each listed file, the table also lists the number of messages generated and its status.

In some instances, the actual messages might be displayed right inside the table, as shown in Figure 10. Messages can appear in the following ways:

- **Red font:** For failed file status only. Explains the problem exposed during the build.

- **Yellow font:** For warning status only. Indicates that minor problems were detected during the build.

The following information is displayed on the Build Modules Details report (Figure 10):

- **File Name:** Name of the file included in the build. The link selected in the Build Execution Details report determines the files listed in the Build Modules Details report. Following are the possible views:
  - Clicking a specific **Build Name**, lists all modules.
  - Clicking a number in the **Fail** column, lists only modules for the corresponding build name that contain files with failures.
  - Clicking a number in the **Warning** column, lists only modules for the corresponding build name that contain files with warnings.
  - Clicking a number in the **Pass** column, lists only modules for the corresponding build name that contain files that passed.

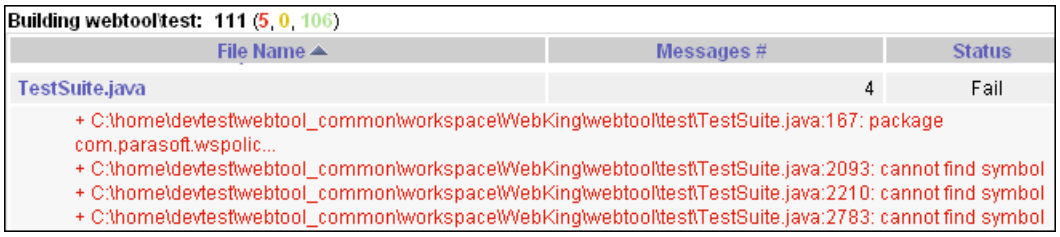
  In the Build Modules Details report—if SourceScanner is properly configured—you can click the name of any file to open the Compiled Files Details pop-up (Figure 11) and view the status, line number of any failures or warnings, and corresponding messages.

Figure 11: Compiled Files Details

- **Messages #:** Lists the number of messages available that pertain to the corresponding file. For messages pertaining to files with a Fail or Warning status, those messages are listed beneath the file name as shown in Figure 12.

  of the build selected from the Build Results graph or table.

Figure 12: Build Modules Details - Messages #



- **Status:** Fail, Warning, or Pass.

Code Review

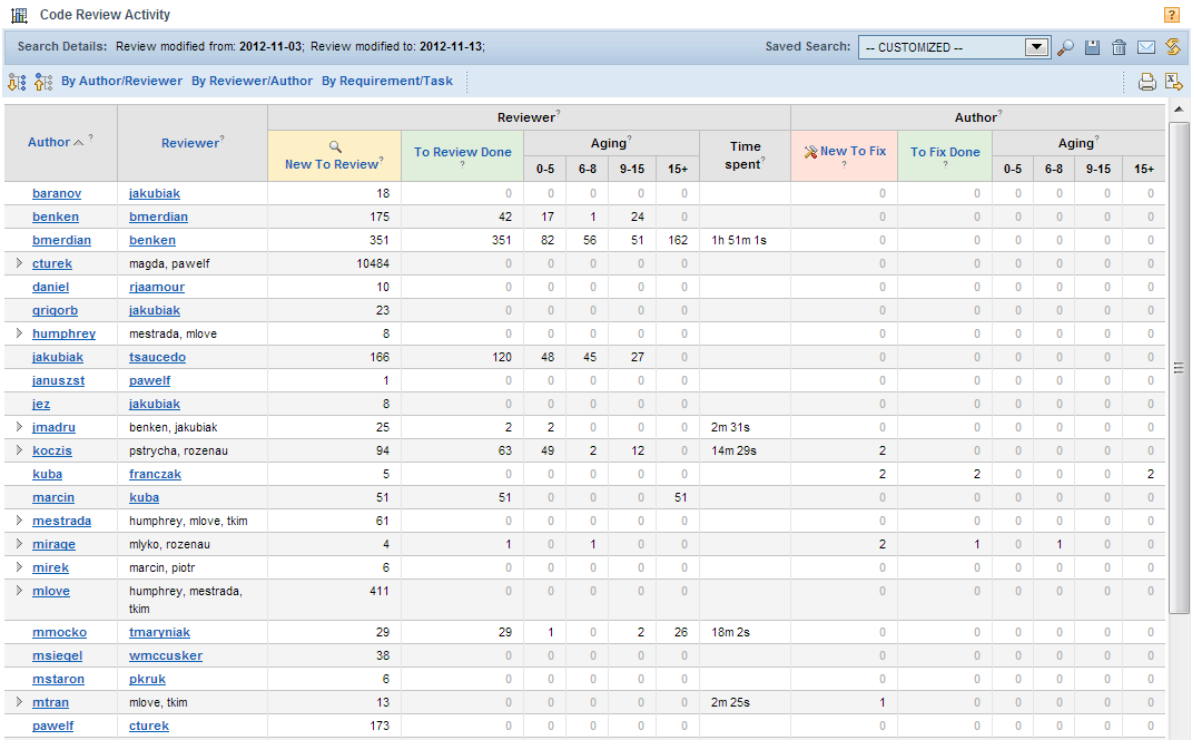
Automated code review is faciliated by the code review functionality executed by the Parasoft Test solution (Parasoft Jtest, Parasoft C++test, and Parasoft dotTEST, and Parasoft SOAtest). Report Center stores code review data from these tools and provides a wide variety of code review reports.The reports provide the visualization and statistical data of Code Review processes performed in your project.

The following types of Code Review reports are provided:

- Code Review Activity: Presents statistics for the specific period
- Code Review Status: Presents up-to-date status
- Code Review summary chart on Architect Dashboard

Code Review Activity

The Code Review Activity report shows code review process statistics for the selected project during the specified period.



Code Review Activity

Search Details: Review modified from: **2012-11-03**; Review modified to: **2012-11-13**; Saved Search: -- CUSTOMIZED --

By Author/Reviewer | By Reviewer/Author | By Requirement/Task

Author	Reviewer	Reviewer: New To Review	Reviewer: To Review Done	Reviewer: Aging 0-5	Reviewer: Aging 6-8	Reviewer: Aging 9-15	Reviewer: Aging 15+	Reviewer: Time spent	Author: New To Fix	Author: To Fix Done	Author: Aging 0-5	Author: Aging 6-8	Author: Aging 9-15	Author: Aging 15+
baranov	jakubiak	18	0	0	0	0	0		0	0	0	0	0	0
benken	bmerdian	175	42	17	1	24	0		0	0	0	0	0	0
bmerdian	benken	351	351	82	56	51	162	1h 51m 1s	0	0	0	0	0	0
cturek	magda, pawelf	10484	0	0	0	0	0		0	0	0	0	0	0
daniel	rjaamour	10	0	0	0	0	0		0	0	0	0	0	0
grigorb	jakubiak	23	0	0	0	0	0		0	0	0	0	0	0
humphrey	mestrada, mlove	8	0	0	0	0	0		0	0	0	0	0	0
jakubiak	tsaucedo	166	120	48	45	27	0		0	0	0	0	0	0
januszst	pawelf	1	0	0	0	0	0		0	0	0	0	0	0
jez	jakubiak	8	0	0	0	0	0		0	0	0	0	0	0
jmadru	benken, jakubiak	25	2	2	0	0	0	2m 31s	0	0	0	0	0	0
koczis	pstrycha, rozenau	94	63	49	2	12	0	14m 29s	2	0	0	0	0	0
kuba	franczak	5	0	0	0	0	0		2	2	0	0	0	2
marcin	kuba	51	51	0	0	0	51		0	0	0	0	0	0
mestrada	humphrey, mlove, tkim	61	0	0	0	0	0		0	0	0	0	0	0
mirage	mlyko, rozenau	4	1	0	1	0	0		2	1	0	1	0	0
mirek	marcin, piotr	6	0	0	0	0	0		0	0	0	0	0	0
mlove	humphrey, mestrada, tkim	411	0	0	0	0	0		0	0	0	0	0	0
mmocko	tmaryniak	29	29	1	0	2	26	18m 2s	0	0	0	0	0	0
msiegel	wmccusker	38	0	0	0	0	0		0	0	0	0	0	0
mstaron	pkruk	6	0	0	0	0	0		0	0	0	0	0	0
mtran	mlove, tkim	13	0	0	0	0	0	2m 25s	1	0	0	0	0	0
pawelf	cturek	173	0	0	0	0	0		0	0	0	0	0	0

Each row in the default view of the report shows code review activity metrics for each Author/Reviewer pair. The following Reviewer metrics are grouped and displayed in each row:

- **New To Review**: Number of new code reviews created for the specific Reviewer(s) during the specified period.
- **To Review Done**: Number of code reviews marked as Accepted or Done during the specified period.

- **Aging**: Number of days the reviewer took to complete the review(s).
- **Time Spent**: Actual time spent on code review actions, i.e. reviewing source code in Eclipse or Visual Studio with Parasoft Test.

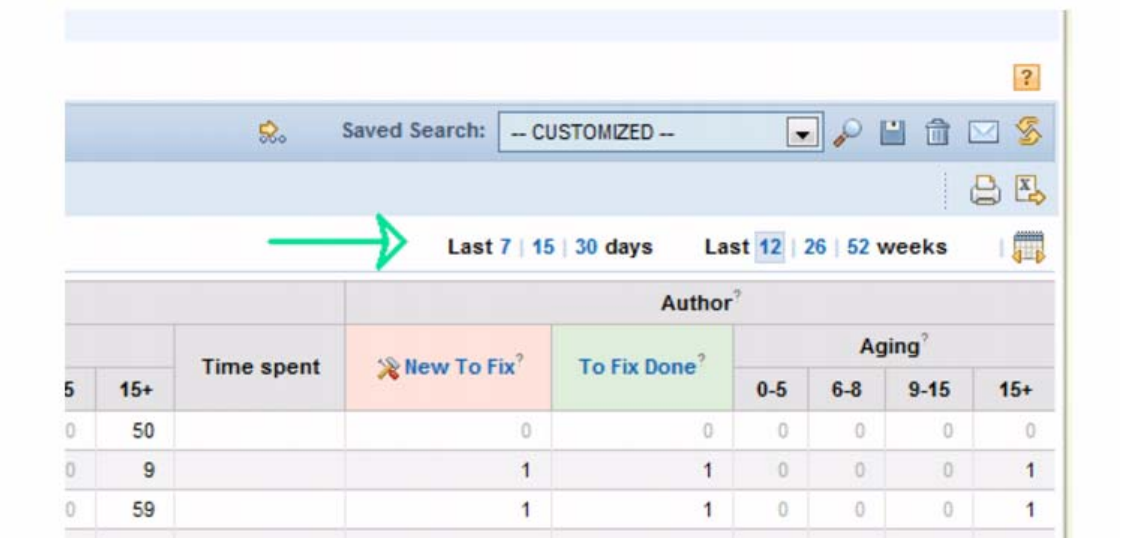
The following Author metrics are grouped and displayed in each row:

- **New To Fix**: Number of new code review issues created for the specific Author(s) in the selected period.
- **To Fix Done**: Number of code review issues marked as Done during the selected period.
- **Aging**: Number of days the author took to fix the code review issue(s).

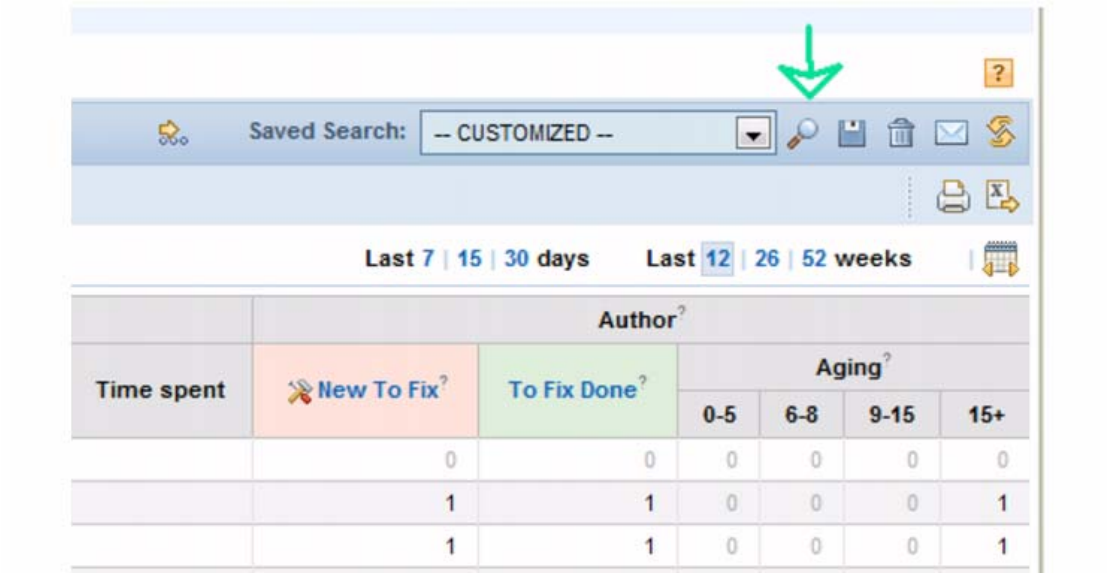
Adjusting Report Filters/Parameters

Users can easily filter the metrics by adjusting the following report parameters:

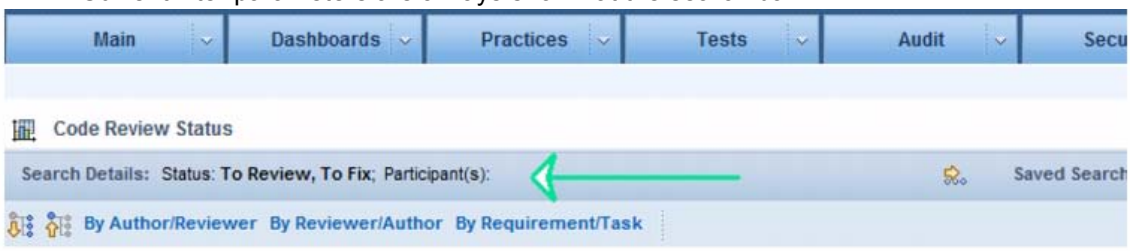
- Click on a time range to view different periods



- Open the search filter to find specific code review data (e.g. only code from specific author or only the highest severity issues) from the selected period.



- Current filter parameters are always shown at the search bar:



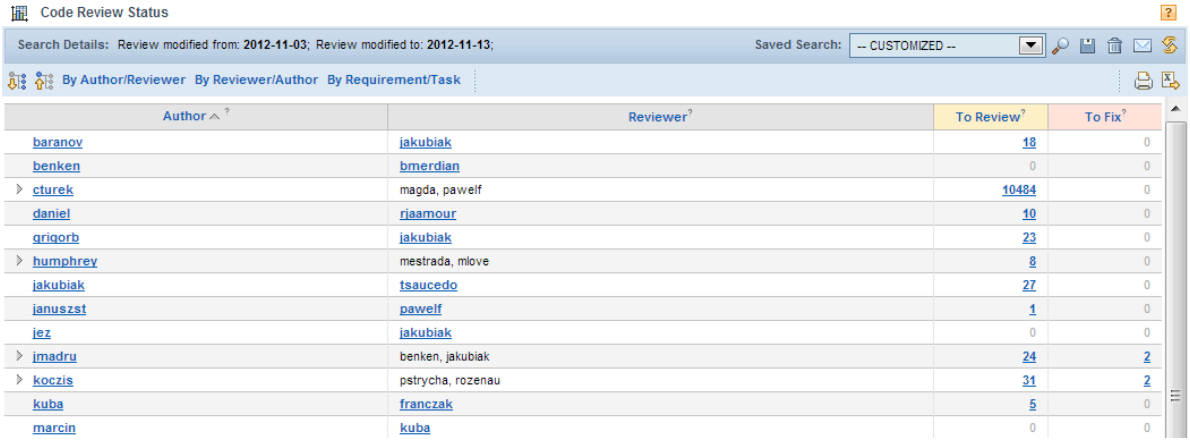
By default, the statistics are displayed for the project team members, including session tags (if specified in the Project Code Review filter on project configuration page in Development Testing Platform Administration).

- For details about configuring team members for projects, see “Adding Users to Projects” on page 303.
- For details about configuring projects code review filter, see “Code Review Filter” on page 169 and see “Integrating with Code Review” on page 267.

Users can rearrange the report by grouping the data by reviewer/author or author/reviewer or requirement/task. Click any developer or reviewer name to see the Code Review activity details for the pair. See “Code Review Details.” chapter for a more information.

Code Review Status

This report shows code review status for the selected project and date--including the number of pending code reviews, as well as the number of code review issues to be fixed..



Author	Reviewer	To Review	To Fix
baranov	jakubiak	18	0
benken	bmerdian	0	0
cturek	magda, pawelf	10484	0
daniel	rjaamour	10	0
grigorb	jakubiak	23	0
humphrey	mestrada, mlove	8	0
jakubiak	tsaucedo	27	0
januszst	pawelf	1	0
jez	jakubiak	0	0
jmadru	benken, jakubiak	24	2
koczis	pstrycha, rozenau	31	2
kuba	franczak	5	0
marcin	kuba	0	0

Each row in the default view of the report shows the following code review status for each Author/Reviewer pair

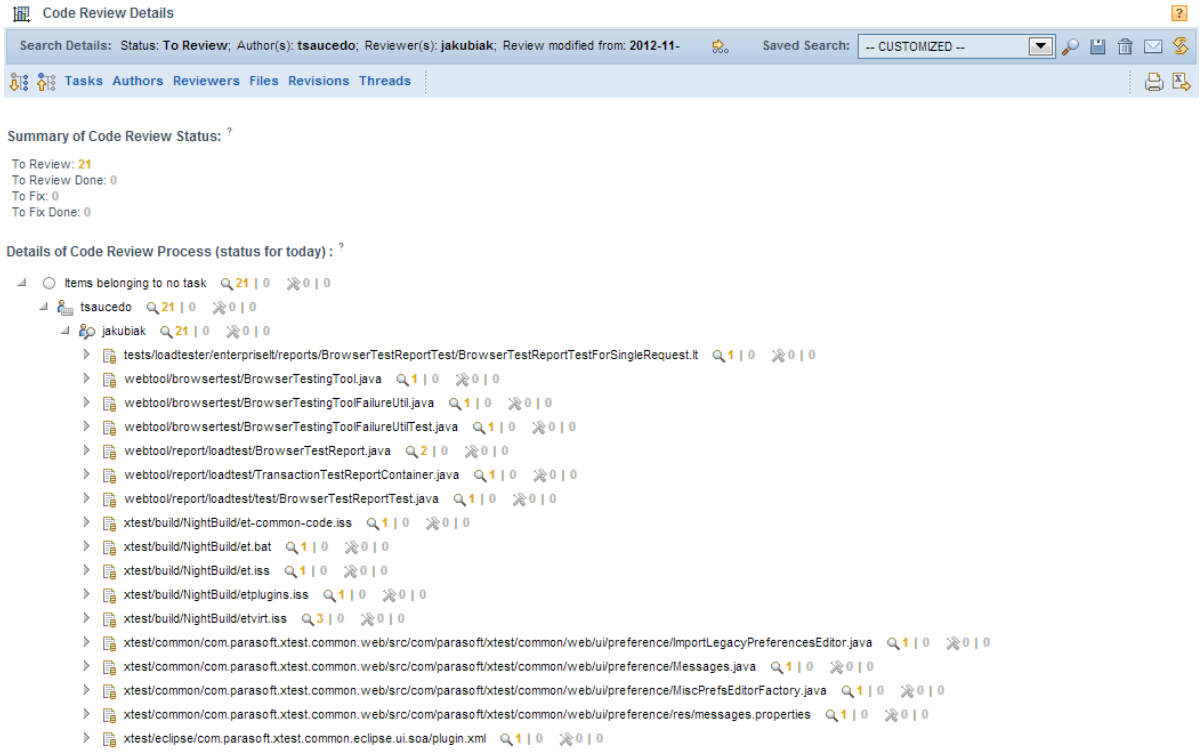
- To Review - Number of code reviews pending to be done by the specific reviewer(s)
- To Fix - Number of code review issues pending to fixed by the specific author(s)

Filtering and regrouping the report data is similar to filtering and regrouping data in the Code Review Activity report (see “Code Review Activity” on page 61).

Code Review Details

The Code Review Details report shows details of code review process for the specified filter (selected project by default), including:

- Code review process participants.
- Source code files under review.
- Issues that have been found.
- Messages exchanged between the process stakeholders.



Click on category in the filter tool bar to expand the data and to show details by:

- Tasks
- Authors
- Reviewers
- Files Revisions
- Threads

Click **Diff.** to view the source code that has been modified by the developer and is subject to code review by the reviewer.

Related Topics

The code review process is also covered in the following areas:

- “Integrating with Code Review”, page 267
- The Parasoft Test User’s Guide

Tools Usage

The Tools Usage page (Figure 13) displays statistics useful for administration purposes, such as the number of users who requested a license, number of users who ran listed products, and license usage by product and platform.

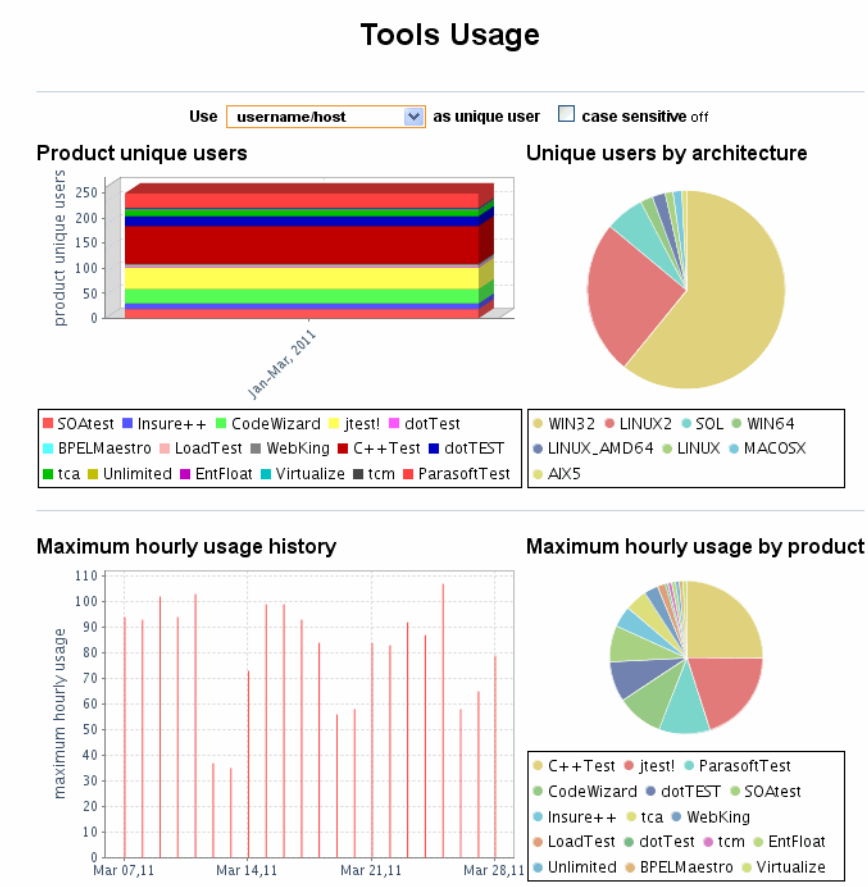
Tools Usage data is displayed in the following reports:

- Product Unique Users Report
- Maximum Hourly Usage by Product
- Maximum Hourly Usage History
- Unique Users by Architecture

You can also customize the data used to generate these reports. To do so, perform the steps described in the following section:

- WIN32: Number of users that ran the correlating product on this architecture.

Figure 13: Tools Usage

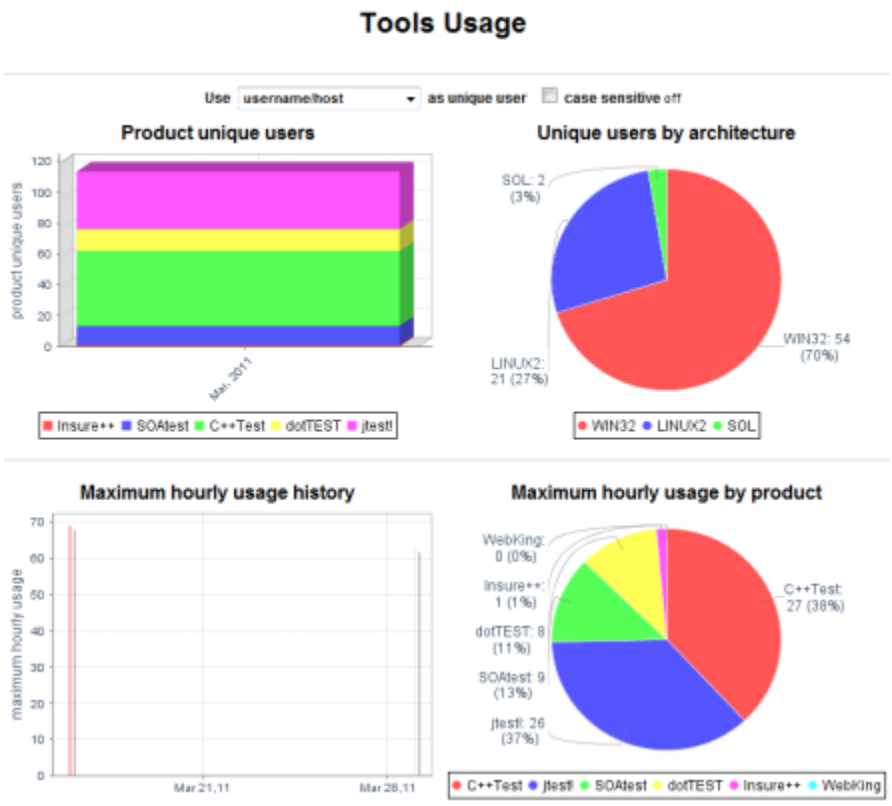


How do I get here? *Practices > Tools Usage*

Product Unique Users Report

For each listed product, the Product Unique Users Report shows the maximum number of unique users who requested license(s) during any one hour period.

Figure 14: Product Unique Users Report



How do I get here? *Practices > Tools Usage > Product Unique Users Report*

Following is the information displayed on the Product Unique Users Report page:

- **Day:** Select to specify that statistics should be verified on a daily basis.
- **Month:** Select to specify that statistics should be verified on a monthly basis.
- **For:** (Product drop-down list) Choose the appropriate product for which to view its data.
- **Day:** Date on which license access was requested.
- **[product name]:** On the corresponding date for each product, the table shows the highest number of licenses that were being used at the same time. Look at Jtest on 2006-06-30 as an example. As you can see, the number 2 is listed. That means that on June 30, 2006, the highest number of licenses for Jtest that were used concurrently was 2.
- **Total:** Lists the maximum concurrent number of licenses for all products stored in License Server for which requests to access have been submitted on the listed date.

For example, a report similar to Figure 14 could have been produced if:

- USER_1 used the Jtest license from 8:00 am to 10:00 am.
- USER_2 used the Jtest license from 9:00 am to 9:30 am.
- USER_1 used the Jtest license again from 1:00 pm to 2:00 pm.

- USER_3 used the Jtest license from 3:00 pm to 4:00 pm.

In this scenario, the maximum hourly usage by product is 2—when USER_1 and USER_2 license usage overlapped.

On rare occasions the following scenario is possible:

- USER_1 used license for 15 minutes (from 9:00 am to 9:15 am) and released it.
- After 10 minutes passed, the license was requested by USER_2 (from 9:25 am to 11:00 am).

Here, since the license was requested twice within a one-hour period (once at 9:00 am and again at 9:25 am), the concurrent usage is 2.

To view more details about license requests, click the number specified for the appropriate product and date. The License Requests Details page opens. See "License Requests Details" for details.

Note: *In addition to user names, host names are also taken into consideration to measure license usage. See* "License Summary", page 137 *regarding the Unique Users Recognition Policy setting for more details.*

License Requests Details

The License Requests Details page (Figure 15) lists the history of users' License Server requests for the listed product on the listed date.

Figure 15: License Requests Details

License Requests Details

Product: **SOAtest**
Date: **2006-06-04**

Time	Status	User	Host	Arch	LS Host
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9
21:05	GRANTED	bchan	opossum	WIN32	10.10.32.9

How do I get here? *Practices > Tools Usage > Maximum Hourly Usage by Product > [number of licenses used]*

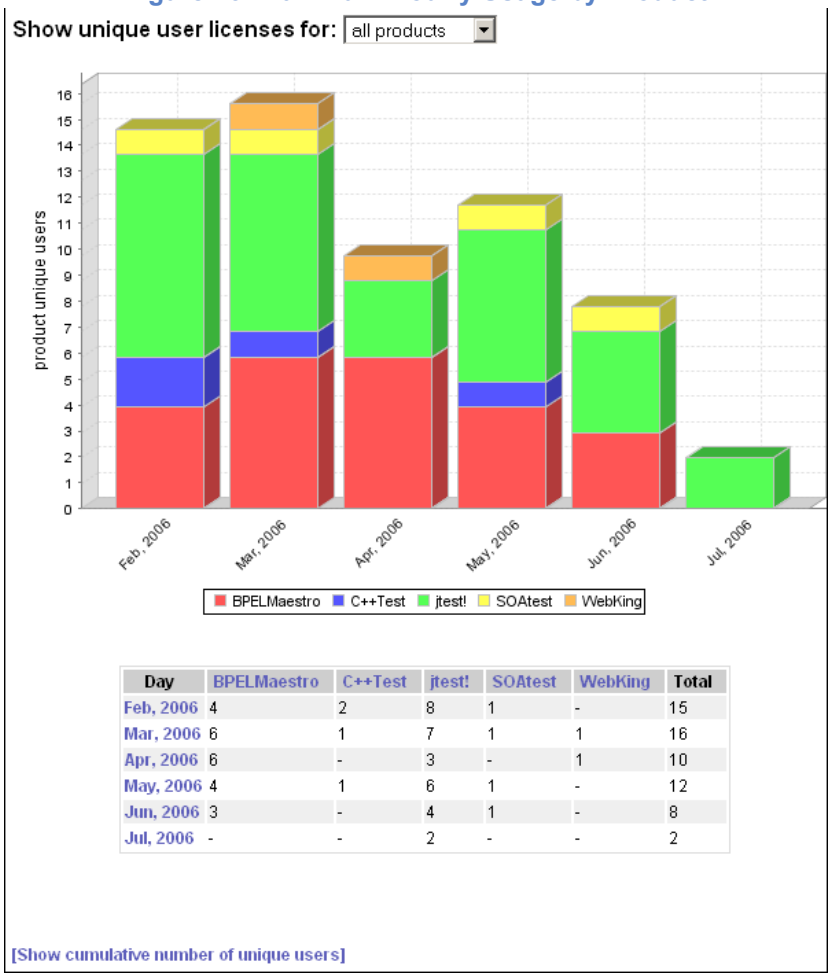
The License Requests Details page lists the following information for the selected product and date:

- **Time:** Time at which access was requested.
- **Status:** Access is either "GRANTED" or "DENIED".
- **User:** Name of the user who requested access to the displayed product.
- **Host:** Name of the host through which the product is accessed.
- **Arch:** (architecture/platform) Name of the platform on which the license is installed.
- **LS Host:** Name of the License Server host.

Maximum Hourly Usage by Product

The Maximum Hourly Usage by Product graph (Figure 16) shows the total number of unique users who were granted a license for each listed product. (Unique users are recognized by either the user's name and machine ID or the user's name only.) The data displayed is grouped by month, so each bar represents one month of data.

Figure 16: Maximum Hourly Usage by Product



Day	BPELMaestro	C++Test	jtest!	SOAtest	WebKing	Total
Feb, 2006	4	2	8	1	-	15
Mar, 2006	6	1	7	1	1	16
Apr, 2006	6	-	3	-	1	10
May, 2006	4	1	6	1	-	12
Jun, 2006	3	-	4	1	-	8
Jul, 2006	-	-	2	-	-	2

How do I get here? *Practices > Tools Usage > Maximum Hourly Usage by Product*

The Maximum Hourly Usage by Product page lists the following information:

- **Show unique user licenses for:** Select the name of the product for which to view license information.
- **Day:** Click to open the Maximum Hourly Usage by Product Details page. The data shown is filtered and shows product usage during the selected month. See “Maximum Hourly Usage by Product Details” for details.
- **<Product name>:** Lists the number of product licenses that were granted during the corresponding month.

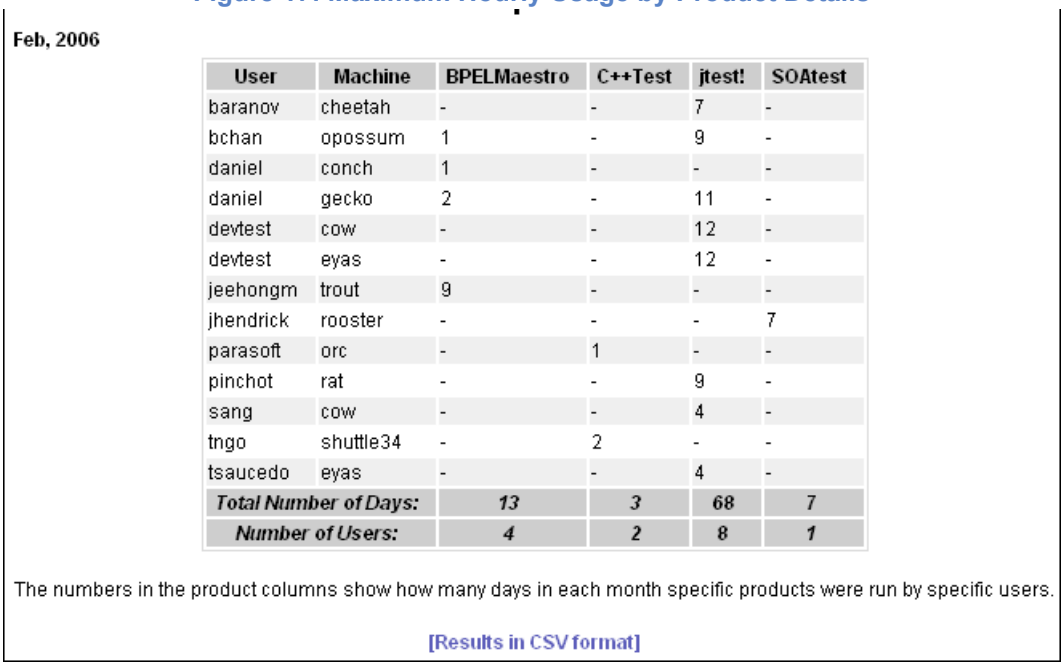
  By default, data for all products is displayed. To filter data, click the appropriate product name to show data only for the selected product.

- **Total:** Lists the total number of licenses granted for the selected products on the listed dates.
- **[Show cumulative number of unique users]:** Click to open the Cumulative Number of Unique Users page, which shows the cumulative statistics of unique users for all products. See “Cumulative Number of Unique Users” for details.

Maximum Hourly Usage by Product Details

The Maximum Hourly Usage by Product Details page (Figure 17) provides license use data, such as which products were run, by which users, and how many days the products were active. This data enables you to verify whether products are being used often enough and how many licenses were granted during the month. It also shows whether the users responsible for using the product are really running it.

Figure 17: Maximum Hourly Usage by Product Details



Feb, 2006

User	Machine	BPELMaestro	C++Test	jtest!	SOAtest
baranov	cheetah	-	-	7	-
bchan	opossum	1	-	9	-
daniel	conch	1	-	-	-
daniel	gecko	2	-	11	-
devtest	cow	-	-	12	-
devtest	eyas	-	-	12	-
jeehongm	trout	9	-	-	-
jhendrick	rooster	-	-	-	7
parasoft	orc	-	1	-	-
pinchot	rat	-	-	9	-
sang	cow	-	-	4	-
tngo	shuttle34	-	2	-	-
tsaucedo	eyas	-	-	4	-
Total Number of Days:		*13*	*3*	*68*	*7*
Number of Users:		*4*	*2*	*8*	*1*

The numbers in the product columns show how many days in each month specific products were run by specific users.

[Results in CSV format]

How do I get here? *Practices > Tools Usage > Maximum Hourly Usage by Product > [date]*

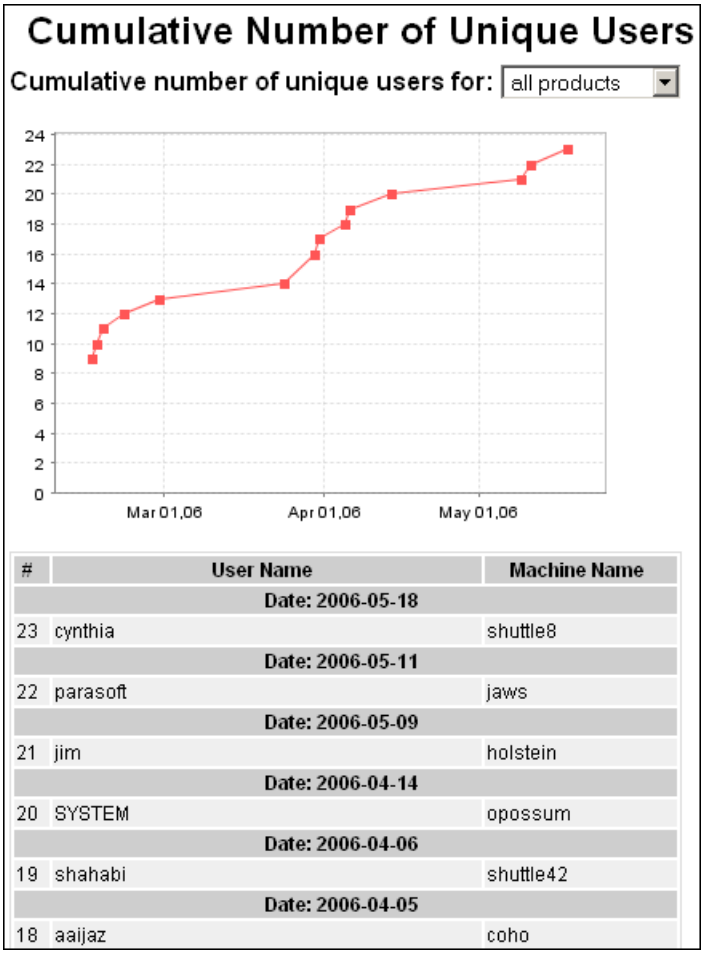
The Maximum Hourly Usage by Product Details page lists the following information:

- **User:** Name of the user who ran the product.
- **Machine:** Name of the machine on which the product was run.
- **<Product name>:** Name the product used.
- **Total Number of Days:** Number of days that the product was active within the displayed month.
- **Number of Users:** Number of users who ran the product.
- **[Results in CSV format]:** Allows you to download the results of this report in Comma Separated Values format in order to upload them to a specific user’s system—in Excel, for example.

Cumulative Number of Unique Users

The Cumulative Number of Unique Users shows the cumulative statistics of unique users for all products (default) or the selected product.

Figure 18: Cumulative Number of Unique Users



#	User Name	Machine Name
	Date: 2006-05-18	
23	cynthia	shuttle8
	Date: 2006-05-11	
22	parasoft	jaws
	Date: 2006-05-09	
21	jim	holstein
	Date: 2006-04-14	
20	SYSTEM	opossum
	Date: 2006-04-06	
19	shahabi	shuttle42
	Date: 2006-04-05	
18	aaijaz	coho

How do I get here? *Practices > Tools Usage > Maximum Hourly Usage by Product > [Show cumulative numbers of unique users]*

The Cumulative Number of Unique Users page lists the following information:

- **#:** Number of times the product was run.
- **User Name:** Name of the user who ran the product.
- **Machine Name:** Name of the machine on which the product was run.
- **Date:** Date on which the product was run.

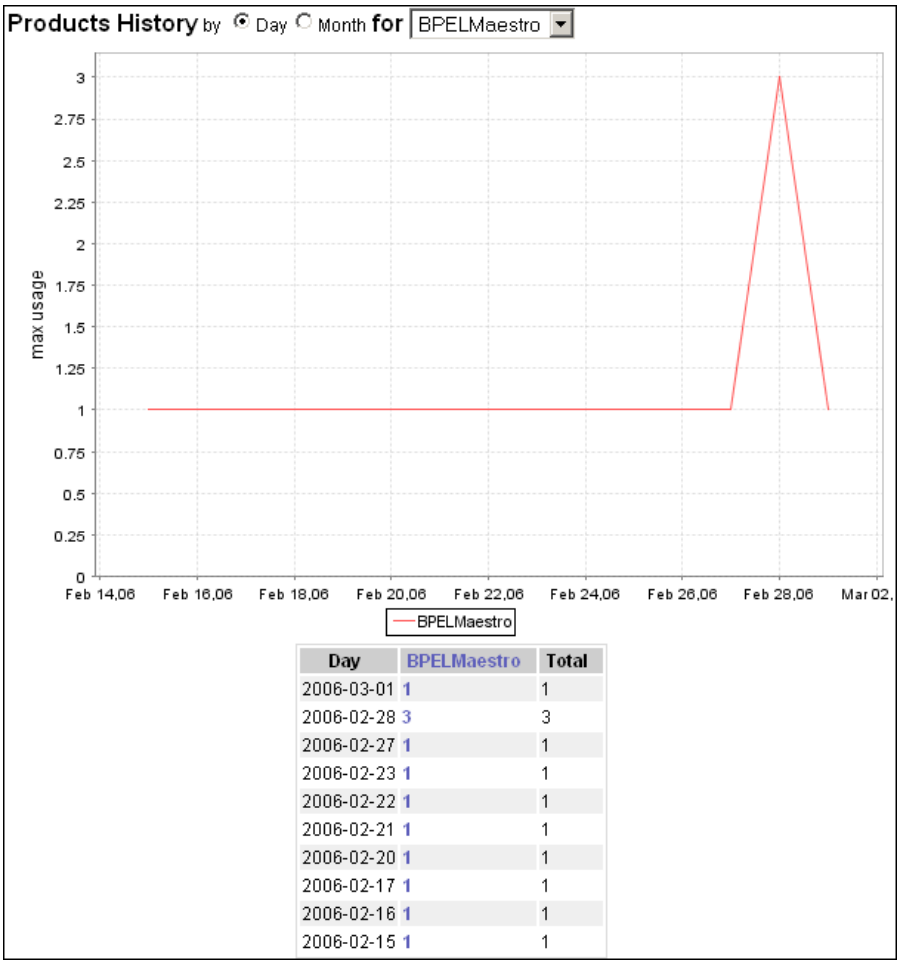
Maximum Hourly Usage History

The Maximum Hourly Usage History uses the same data as the Maximum Hourly Usage by Product, but instead of sorting data by date, sorts it by product.

In the Maximum Hourly Usage by Product pie chart shown on the main Tools Usage page (Figure 13), click the product for which you want to view its history. The Product History report (Figure 19) is

displayed in which you can view statistics of granted licenses for the selected product. See “Maximum Hourly Usage by Product”, page 69 for details about the data shown in this report.

Figure 19: Maximum Hourly Usage History



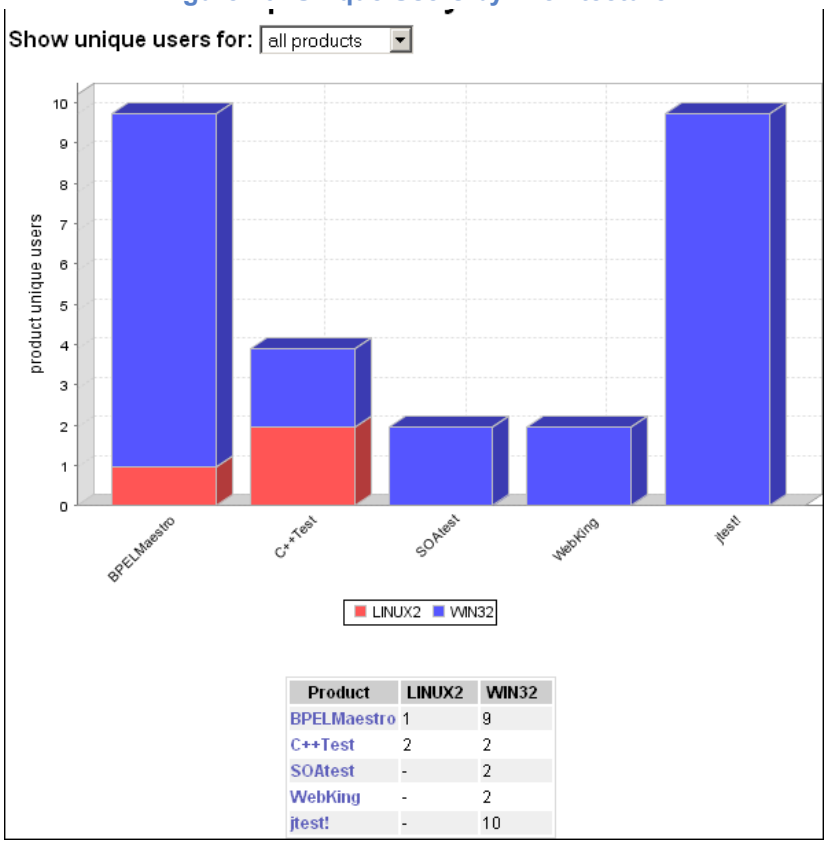
Day	BPELMaestro	Total
2006-03-01	1	1
2006-02-28	3	3
2006-02-27	1	1
2006-02-23	1	1
2006-02-22	1	1
2006-02-21	1	1
2006-02-20	1	1
2006-02-17	1	1
2006-02-16	1	1
2006-02-15	1	1

How do I get here? *Practices > Tools Usage > Maximum hourly usage history*

Unique Users by Architecture

The Unique Users by Architecture page (Figure 20) shows the number of unique users running Parasoft products based on operating system architecture (Win or Linux).

Figure 20: Unique Users by Architecture



Product	LINUX2	WIN32
BPELMaestro	1	9
C++Test	2	2
SOAtest	-	2
WebKing	-	2
jtest!	-	10

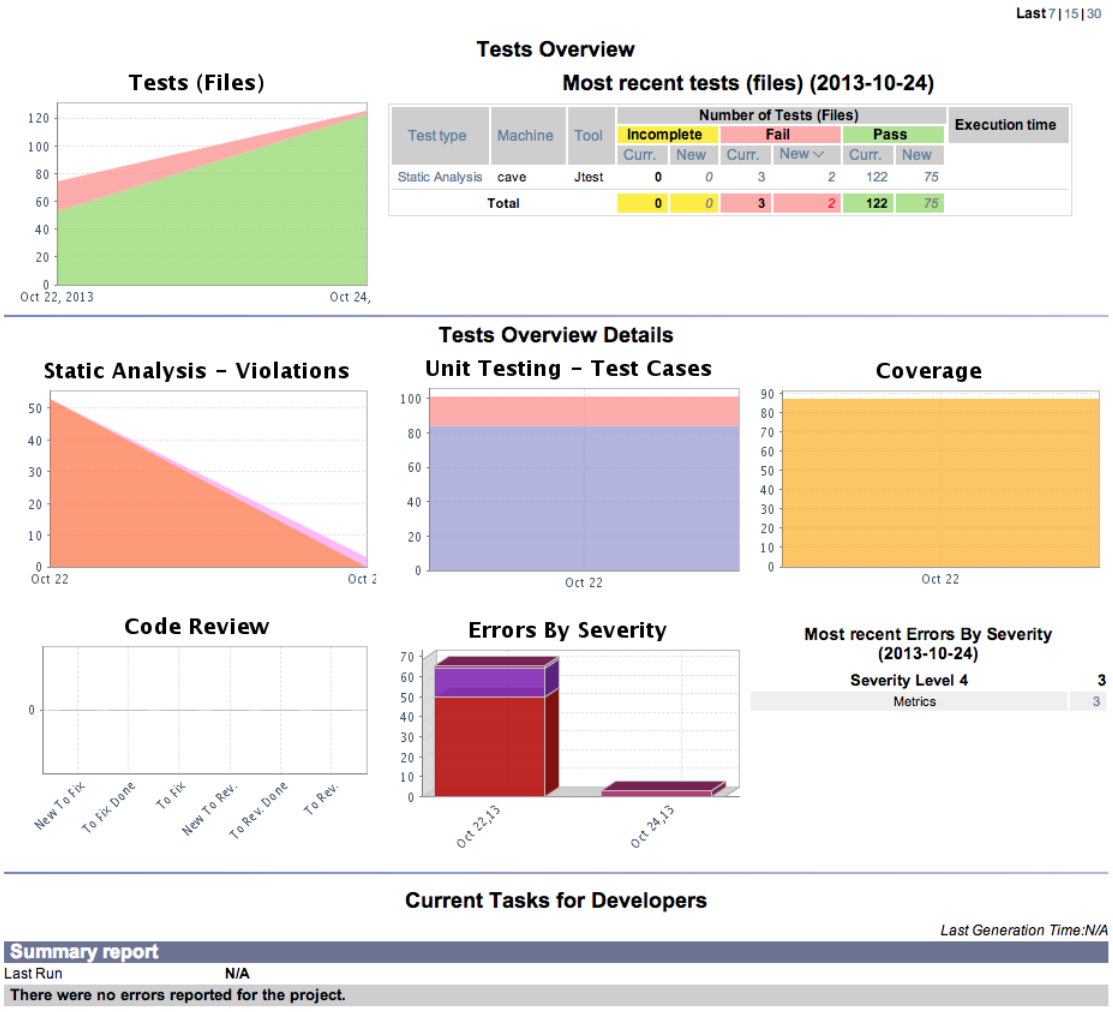
How do I get here? *Practices > Tools Usage > Unique users by architecture*

The Maximum Hourly Usage by Product page lists the following information:

- **Product:** Name of the product that was run. Click a specific product name to filter the report by that product.
- **LINUX2:** Number of users that ran the correlating product on this architecture.
- **WIN32:** Number of users that ran the correlating product on this architecture.

Tests Overview

From the Reports view, choose Tests> Tests Overview to open the Tests Overview page. This page shows the results of all tests run for the selected project during the specified period of time. These reports display the overall trend of test results, as well as details of the last tests that were run. the Tests Overview section shows the overall trend of test results. The tests include all of the automated tests performed for the selected project, including Static Analysis, Unit Testing, Functional Testing, and so on.



The **Tests (Files)** graph included in the Tests Overview section reflects the tendency of the tests over the specified time period.

The graph shows data for unit tests and static analysis differently:

- Unit testing. Number of test (suite) files
- Static analysis. Number of of tested files.

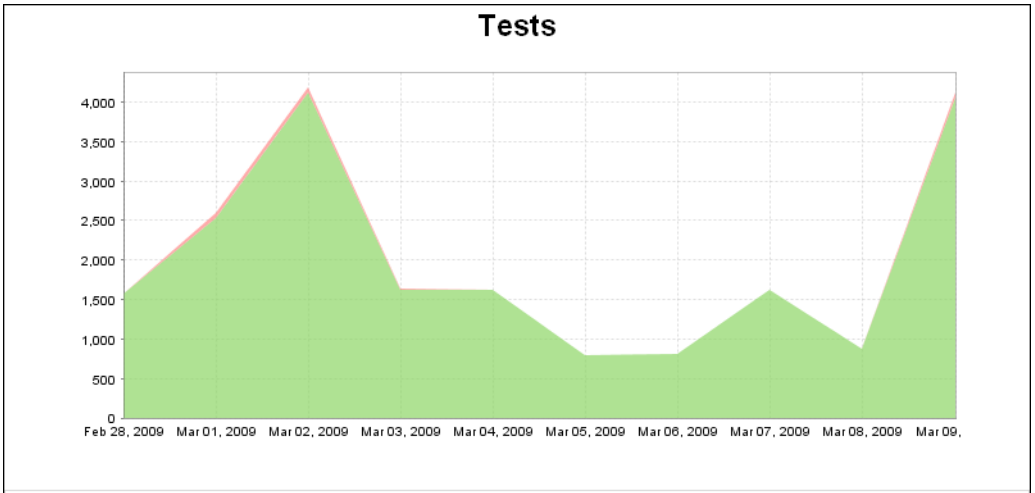
For details about the **Tests (Files)** report, see “Tests (Files)”, page 119.

Tests By Date

The **Tests** report (Figure 21) is a quick, overall view of your project in its current state. It lists tests that ran during the selected time period. Each row shows the number of incomplete, failed, and passed test cases for all tests that were run on a specific date, including coding standards analysis, unit testing, regression testing, and so on.

For details, see “Tests (Files)”, page 119.

Figure 21: Tests by Date



	Date	Incomplete	Fail	Pass	Total	Duration	Execution Time Diff
+	Mar 09, 2009	0	60	4051	4111	2h 56m 31s	1h 46m 41s
+	Mar 08, 2009	0	0	876	876	1h 9m 50s	-1m 6s
+	Mar 07, 2009	0	4	1618	1622	1h 10m 56s	43m 54s
+	Mar 06, 2009	0	0	810	810	27m 1s	24m 43s
+	Mar 05, 2009	0	0	788	788	2m 18s	-1h 7m
+	Mar 04, 2009	0	4	1618	1622	1h 9m 18s	-15m
+	Mar 03, 2009	0	4	1622	1626	1h 24m 19s	-50m 27s
+	Mar 02, 2009	0	56	4117	4173	2h 14m 46s	17m 19s
+	Mar 01, 2009	0	60	2521	2581	1h 57m 27s	1h 29m 24s
+	Feb 28, 2009	0	2	1562	1564	28m 3s	

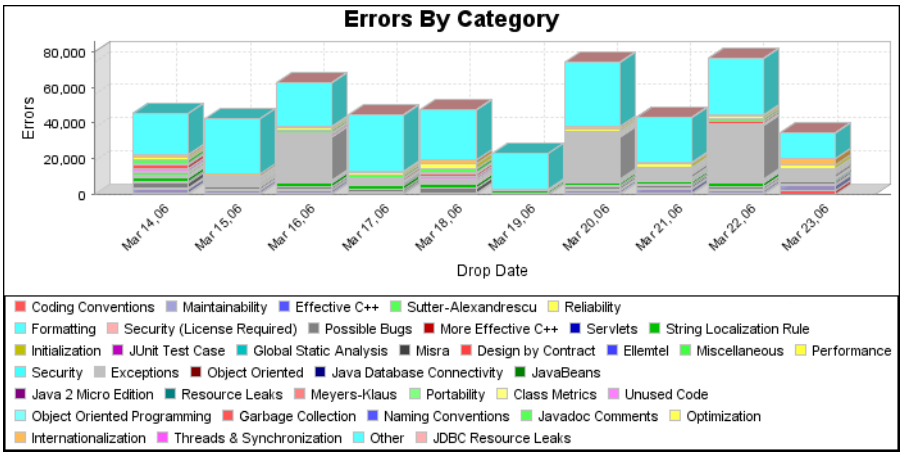
How do I get here? *Tests > Tests by Date*

Errors by Category

The Errors by Category graph enables you to keep an eye on how coding standards and unit testing errors in the code are distributed. Error categories vary based on the tool that sends the report.

To open the Errors by Category graph, choose **Tests >Errors by Category**. This graph uses visualization as opposed to definitive numbers to show ratios between different error categories. Each error category (type) is assigned a different color. For each date within the selected drops for the selected project, colored blocks are stacked atop each other to reflect the number of coding standards and unit testing errors found during testing of that project so that you can see which types of errors are most predominant.

Figure 22: Errors by Category



How do I get here? *Tests > Errors by Category*

Drill-downs from the Errors by Category graph:

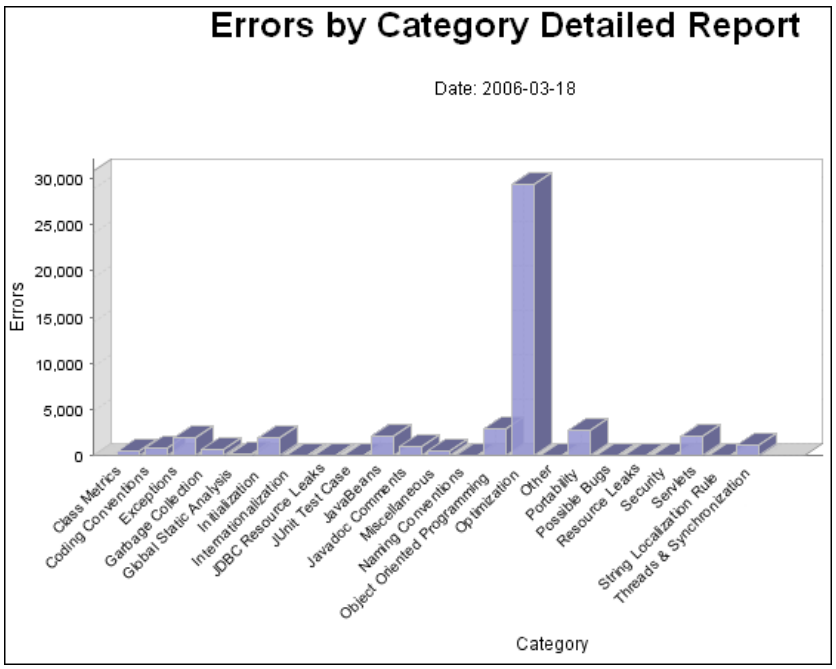
Click a specific date in the Errors by Category graph to open the Errors by Category Detailed Report for that date.

Errors by Category Detailed Report

The Errors by Category Detailed Report graph is displayed at the top of the page and shows the number of coding standards and unit testing errors sorted by category type for the selected drop date.

Each bar represents a specific type. You can change the drop date from the Options bar. The graph looks similar to the one shown in Figure 23.

Figure 23: Errors by Category Detailed Report Graph

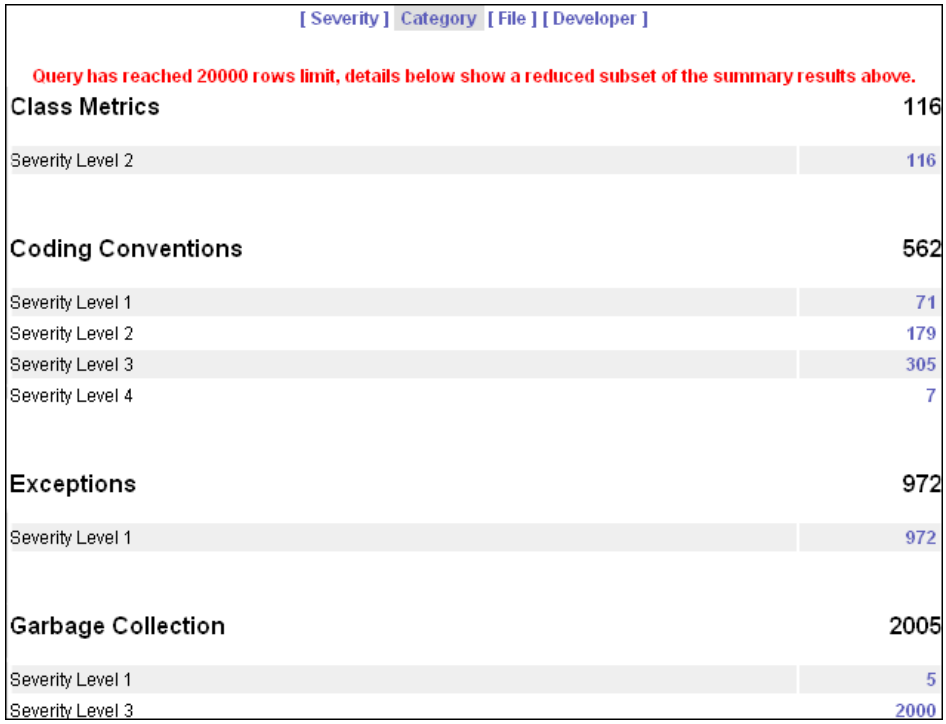


How do I get here? *Tests > Errors by Category > [date] (from graph)*

Below the Errors by Category Detailed Report graph, a table (Figure 24) lists each category in alphabetical order and shows the total number of coding standards and unit testing errors for that

category. Below each category, the number of errors is broken down further and listed by severity level (1 through 5; 1 is the most severe error, while 5 is the least severe).

Figure 24: Errors by Category Detailed Report Table



[Severity] Category [File] [Developer]

Query has reached 20000 rows limit, details below show a reduced subset of the summary results above.

Class Metrics	**116**
Severity Level 2	116
Coding Conventions	**562**
Severity Level 1	71
Severity Level 2	179
Severity Level 3	305
Severity Level 4	7
Exceptions	**972**
Severity Level 1	972
Garbage Collection	**2005**
Severity Level 1	5
Severity Level 3	2000

Drill-downs from the Errors by Category Detailed Report table:

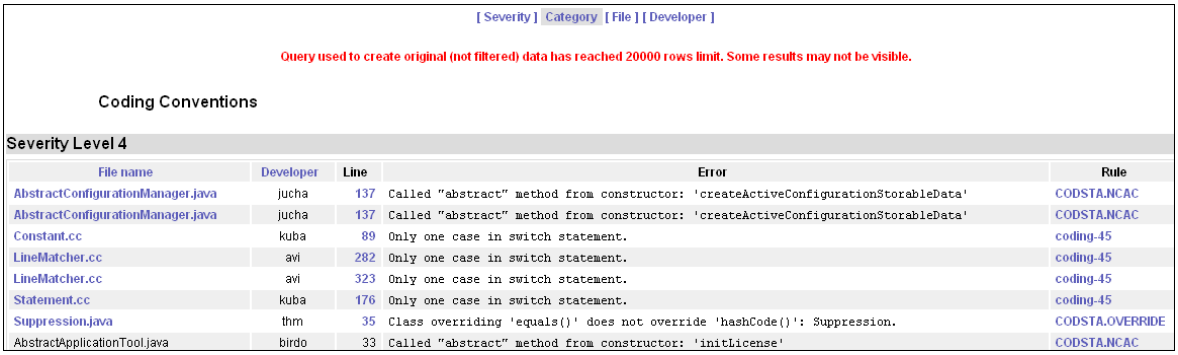
To change the graph and table view:

- Click one of the following links located above the table to view errors from a different perspective:
  - Severity: Opens Errors by Severity Detailed Report.
  - File: Opens Errors by File Detailed Report.
  - Developer: Opens Errors by Developer Detailed Report.

To view more details about the errors listed by category:

- Click the statistic that corresponds with the appropriate category and severity level. A page similar to Figure 25, which contains even more details about the errors appears:

Figure 25: Errors by Category Detailed Report (More Details)



[Severity] Category [File] [Developer]

Query used to create original (not filtered) data has reached 20000 rows limit. Some results may not be visible.

Coding Conventions

Severity Level 4

File name	Developer	Line	Error	Rule
AbstractConfigurationManager.java	jucha	137	Called "abstract" method from constructor: 'createActiveConfigurationStorableData'	CODSTA.NCAC
AbstractConfigurationManager.java	jucha	137	Called "abstract" method from constructor: 'createActiveConfigurationStorableData'	CODSTA.NCAC
Constant.cc	kuba	89	Only one case in switch statement.	coding-45
LineMatcher.cc	avi	282	Only one case in switch statement.	coding-45
LineMatcher.cc	avi	323	Only one case in switch statement.	coding-45
Statement.cc	kuba	176	Only one case in switch statement.	coding-45
Suppression.java	thm	35	Class overriding 'equals()' does not override 'hashCode()': Suppression.	CODSTA.OVERRIDE
AbstractApplicationTool.java	birdo	33	Called "abstract" method from constructor: 'initLicense'	CODSTA.NCAC

How do I get here? *Audit > Errors by Category > [date] (from graph) > [number of errors]*

Following are descriptions of each column heading in the Errors by Category Detailed Report (more details) table, along with drill down options:

- **Filename:** Name of the file that contains the error.

  Click the Filename heading to open Errors by File Detailed Report. Click a specific filename listed beneath the heading to open its source file.

- **Developer:** Username of the developer responsible for the corresponding error.

  Click the Developer heading to open Errors by Developer Detailed Report.

- **Line:** Number of the line on which the error is located in the corresponding error.

  Click a specific line to open the corresponding source file with this line highlighted in yellow.

- **Error:** Description of the error that occurred.

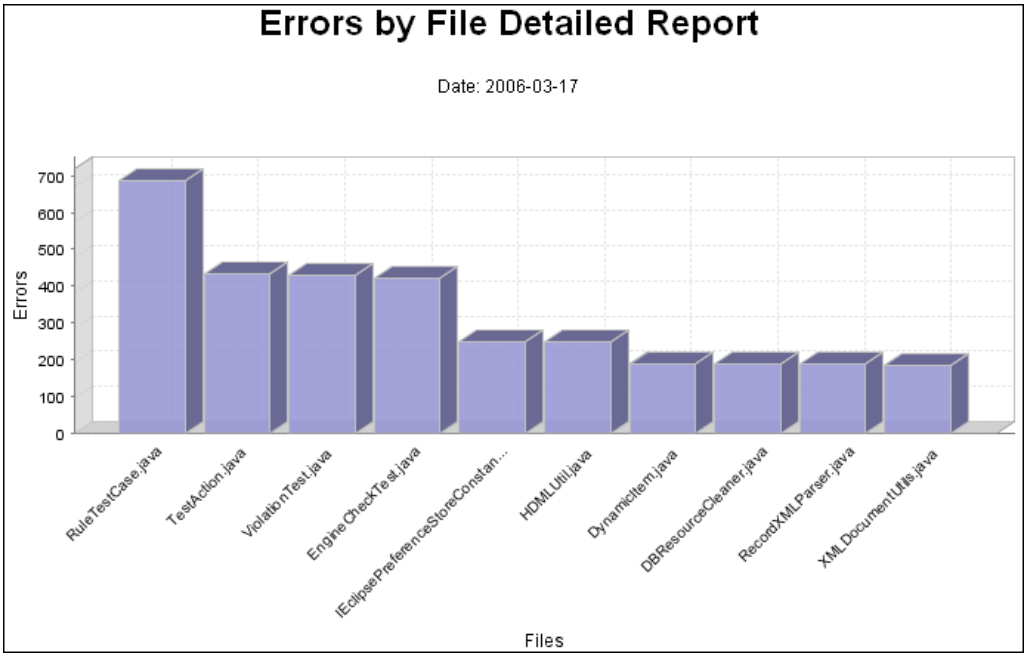
- **Rule:** Name of the rule that was violated in the corresponding error.

  Click a specific rule to open a pop-up box that contains a description of that rule, along with any relevant notes, security relevance, parameters, benefits, drawbacks, examples, repairs, and references.

Errors by File Detailed Report

The Errors by File Detailed Reports shows the ten files that contain the most errors and looks similar to the graph shown in Figure 26.

Figure 26: Errors by File Detailed Report



How do I get here?

Option 1: Audit > Errors by Category > [date] (from graph) > [File]

Option 2: Audit > Errors by Severity > [date] (from graph) > [File]

Below the Errors by File Detailed Report graph is a table (Figure 27) that lists the ten files with the most errors, along with the total number of errors contained in each listed file.

Figure 27: Errors by File Detailed Report Table

[Severity] [Category] File [Developer]	
RuleTestCase.java	717
TestAction.java	452
ViolationTest.java	450
EngineCheckTest.java	441
IEclipsePreferenceStoreConstants.java	261
HDMLUtil.java	259
DynamicItem.java	199
DBResourceCleaner.java	198
RecordXMLParser.java	197
XMLDocumentUtils.java	195

Drill-downs from the Errors by File Detailed Report table:

To change the graph and table view:

- Click one of the following links located above the table to view errors from a different perspective:
  - Severity: Opens Errors by Severity Detailed Report.

- Category: Opens Errors by Category Detailed Report.
- Developer: Opens Errors by Developer Detailed Report.

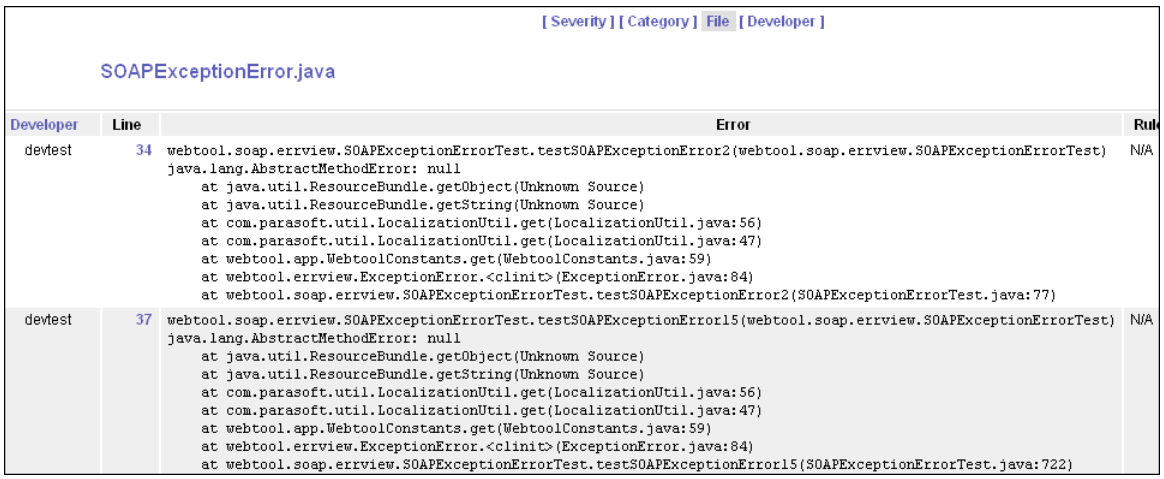
To view source code of a listed file:

- Click the appropriate filename, if link is available.

To view more details about the errors listed by file:

- Click the number of errors that corresponds with the appropriate file name. A page similar to Figure 28, which contains even more details about the file errors appears:

Figure 28: Errors by File Detailed Report (More Details)



[Severity] [Category] File [Developer]

SOAPExceptionError.java

Developer	Line	Error	Rule
devtest	34	webtool.soap.errview.SOAPExceptionErrorTest.testSOAPExceptionError2(webtool.soap.errview.SOAPExceptionErrorTest) java.lang.AbstractMethodError: null at java.util.ResourceBundle.getObject(Unknown Source) at java.util.ResourceBundle.getString(Unknown Source) at com.parasoft.util.LocalizationUtil.get(LocalizationUtil.java:56) at com.parasoft.util.LocalizationUtil.get(LocalizationUtil.java:47) at webtool.app.WebtoolConstants.get(WebtoolConstants.java:59) at webtool.errview.ExceptionError.<clinit>(ExceptionError.java:84) at webtool.soap.errview.SOAPExceptionErrorTest.testSOAPExceptionError2(SOAPExceptionErrorTest.java:77)	N/A
devtest	37	webtool.soap.errview.SOAPExceptionErrorTest.testSOAPExceptionError15(webtool.soap.errview.SOAPExceptionErrorTest) java.lang.AbstractMethodError: null at java.util.ResourceBundle.getObject(Unknown Source) at java.util.ResourceBundle.getString(Unknown Source) at com.parasoft.util.LocalizationUtil.get(LocalizationUtil.java:56) at com.parasoft.util.LocalizationUtil.get(LocalizationUtil.java:47) at webtool.app.WebtoolConstants.get(WebtoolConstants.java:59) at webtool.errview.ExceptionError.<clinit>(ExceptionError.java:84) at webtool.soap.errview.SOAPExceptionErrorTest.testSOAPExceptionError15(SOAPExceptionErrorTest.java:722)	N/A

How do I get here?

Option 1: Audit > Errors by Category > [date] (from graph) > [File] [number of errors] (from table)

Option 2: Audit > Errors by Severity > [date] (from graph) > [File] [number of errors] (from table)

Following are descriptions of each column heading in the Errors by File Detailed Report (more details) table, along with drill down options:

- **Developer:** Username of the developer responsible for the corresponding error.
- Click the Developer heading to open Errors by Developer Detailed Report.
- **Line:** Number of the line on which the error is located in the corresponding error.

  Click a specific line to open the corresponding source file with this line highlighted in yellow.
- **Error:** Description of the error that occurred.
- **Rule:** Name of the rule that was violated in the corresponding error.

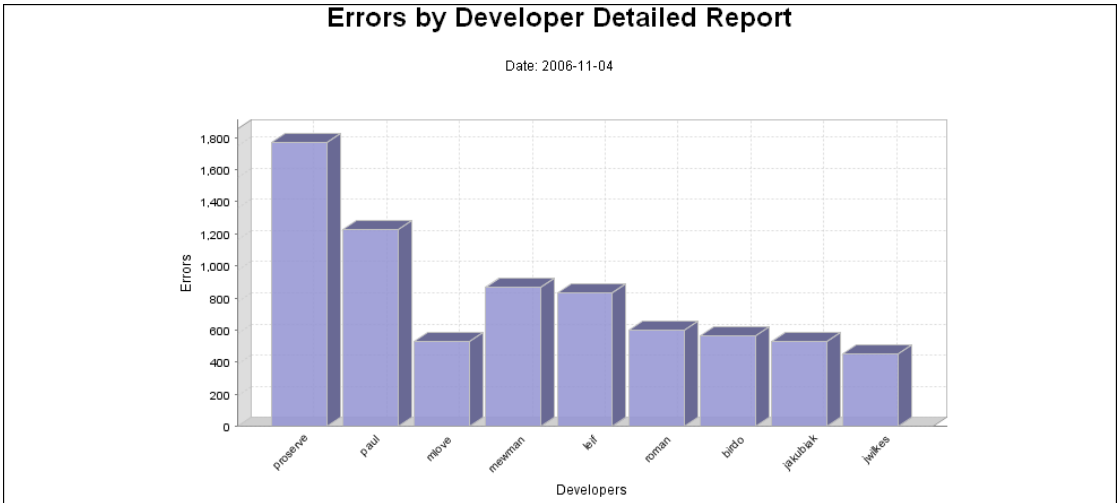
  Click a specific rule to open a pop-up box that contains a description of that rule, along with any relevant notes, security relevance, parameters, benefits, drawbacks, examples, repairs, and references.

Errors by Developer Detailed Report

The Errors by Developer Detailed Report shows the ten developers with the most errors in their files and looks similar to the graph shown in Figure 29.

The table below the Errors by Developer Detailed Report graph lists all of the developers who are responsible for the coding standards and unit testing errors that are detected, along with the total number of errors for which they are responsible.

Figure 29: Errors by Developer Detailed Report Graph



proserve	1823
paul	1264
mlove	3620
rnewman	891
leif	855
roman	664
birdo	590
jakubiak	755
jwilkes	469

How do I get here?

Option 1: Audit > Errors by Category > [date] (from graph) > [Developer]

Option 2: Audit > Errors by Severity > [date] (from graph) > [Developer]

Option 3: Errors by File Detailed Report > [number of errors] (from graph) > [Developer]

Drill-downs from the Errors by Developer Detailed Report table:

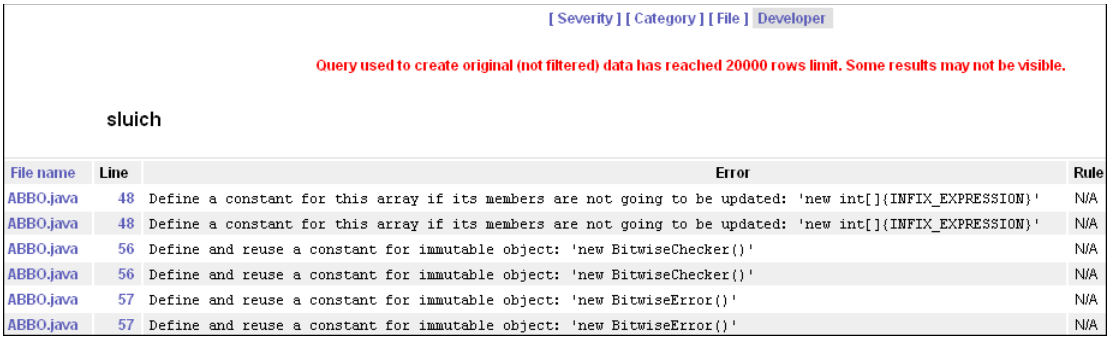
To change the graph and table view:

- Click one of the following links located above the table to view errors from a different perspective:
  - Severity: Opens Errors by Severity Detailed Report.
  - Category: Opens Errors by Category Detailed Report.
  - File: Opens Errors by File Detailed Report.

To view more details about the errors listed by developer:

- Click the number of errors that corresponds with the appropriate developer name. A page similar to Figure 25, which contains even more details about the developer's errors appears:

Figure 30: Errors by Developer Detailed Report (More Details)



[Severity] [Category] [File] Developer

Query used to create original (not filtered) data has reached 20000 rows limit. Some results may not be visible.

sluich

File name	Line	Error	Rule
ABBO.java	48	Define a constant for this array if its members are not going to be updated: 'new int[]{INFIX_EXPRESSION}'	N/A
ABBO.java	48	Define a constant for this array if its members are not going to be updated: 'new int[]{INFIX_EXPRESSION}'	N/A
ABBO.java	56	Define and reuse a constant for immutable object: 'new BitwiseChecker()'	N/A
ABBO.java	56	Define and reuse a constant for immutable object: 'new BitwiseChecker()'	N/A
ABBO.java	57	Define and reuse a constant for immutable object: 'new BitwiseError()'	N/A
ABBO.java	57	Define and reuse a constant for immutable object: 'new BitwiseError()'	N/A

How do I get here?

Option 1: Audit > Errors by Category > [date] (from graph) > [Developer] > [number of errors]

Option 2: Audit > Errors by Severity > [date] (from graph) > [Developer] > [number of errors]

Option 3: Errors by File Detailed Report > [number of errors] (from graph) > [Developer] > [number of errors]

Following are descriptions of each column heading in the Errors by Developer Detailed Report (more details) table, along with drill down options:

- **Filename:** Name of the file that contains the error.

  Click the Filename heading to open Errors by File Detailed Report. Click a specific filename listed beneath the heading to open its source file.

- **Line:** Number of the line on which the error is located in the corresponding error.

  Click a specific line to open the corresponding source file with this line highlighted in yellow.

- **Error:** Description of the error that occurred.
- **Rule:** Name of the rule that was violated in the corresponding error.
- Click a specific rule to open a pop-up box that contains a description of that rule, along with any relevant notes, security relevance, parameters, benefits, drawbacks, examples, repairs, and references.

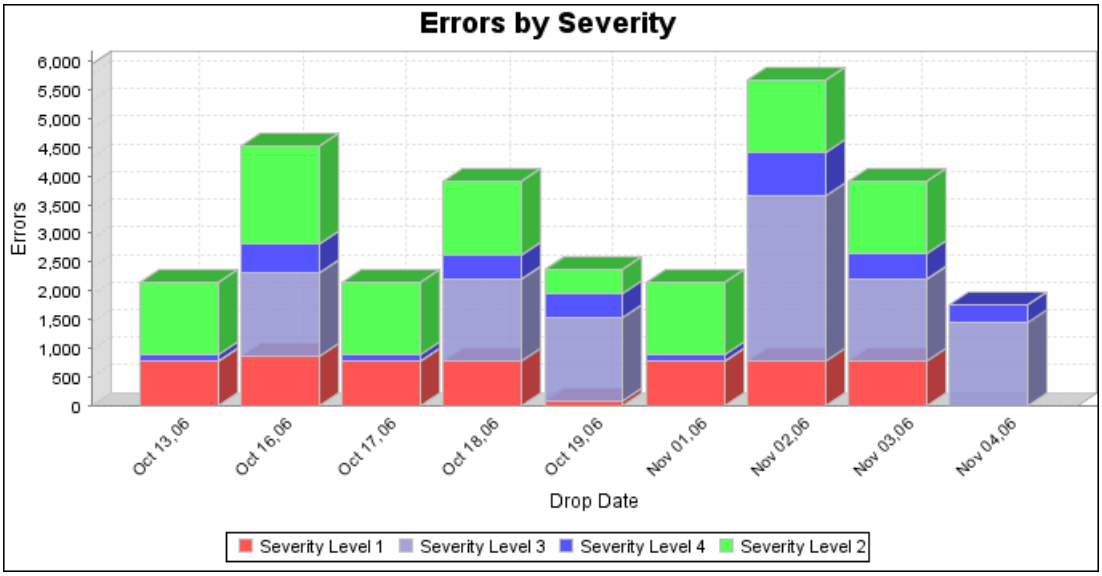
Errors by Severity

To open the Errors by Severity graph, choose **Tests >Errors by Severity**. This graph uses visualization as opposed to definitive numbers to show ratios between different coding standards and unit testing error severities. Each error severity level is assigned a different color. Severity levels are determined by you, but typically, severity levels range from 1 to 5—1 being most severe and of highest concern.

For each date within the selected drops for the selected project, colored blocks are stacked atop each other to reflect the number of errors found during testing of that project so that you can see which severity levels are most predominant.

The Errors by Severity graph is also a useful means to confirm whether coding standards and unit testing errors with highest severity are being resolved. The bar height in the Errors by Severity graph should decrease from day to day to reflect whether errors are being resolved.

Figure 31: Errors by Severity



How do I get here? *Tests > Errors by Severity*

Drill-downs from the Errors by Severity graph:

Click a specific date in the Errors by Severity graph to open the Errors by Severity Detailed Report.

Errors by Severity Detailed Report

The Errors by Severity Detailed Report shows the number of coding standards and unit testing errors sorted by severity level for the selected drop date. If, for example, there are five levels of severity (1 being the most severe, 5 being the least severe), a graph similar to the one shown in Figure 32 is displayed. Each bar represents a different severity level.

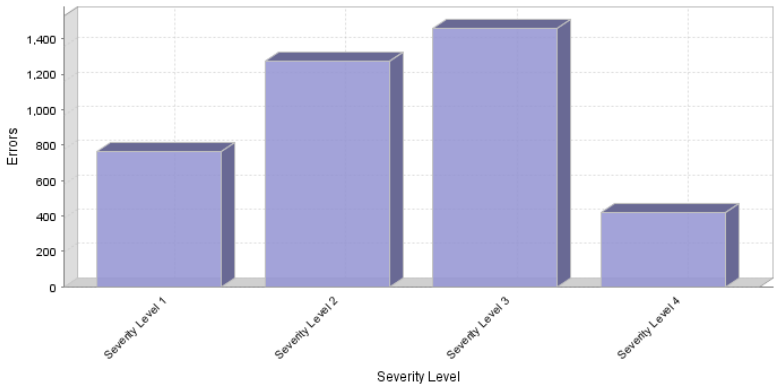
The table located below the Errors by Severity Detailed Report graph lists coding standards and unit testing errors that are detected sorted by severity level. For each severity level, the total number of coding standards and unit testing errors detected is displayed for the selected date on which testing

was performed. Below each severity level, all of the rule categories for which coding standards and unit testing errors were detected are listed, along with the number of errors found for each.

Figure 32: Errors by Severity Detailed Report

Errors by Severity Detailed Report

Date: 2006-10-18



Severity [Category] [File] [Developer]

Severity Level 1	**790**
Coding Conventions	29
Exceptions	614
JDBC Resource Leaks	8
Java Database Connectivity	9
JavaBeans	1
Optimization	38
Possible Bugs	17
Resource Leaks	40
Security	34
Severity Level 2	**1317**
Coding Conventions	3
Initialization	1
JDBC Resource Leaks	866
Possible Bugs	13
Unused Code	434
Severity Level 3	**1507**
Coding Conventions	2
Optimization	4
Other	1489
Possible Bugs	5
Serialization	4
Unused Code	3
Severity Level 4	**436**
Optimization	47
Other	335
Possible Bugs	10
Serialization	44

How do I get here? *Tests > Errors by Severity > [date] (from graph)*

Drill-downs from the Errors by Severity Detailed Report table:

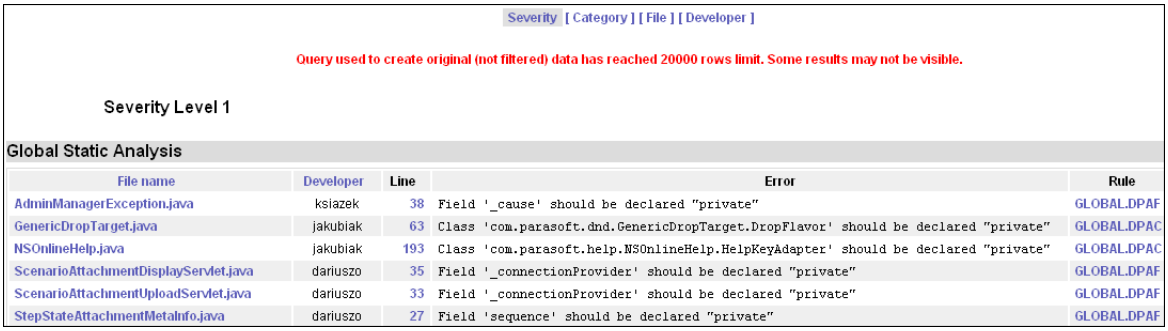
To change the graph and table view:

- Click one of the following links located above the table to view errors from a different perspective:
    - Category: Opens Errors by Category Detailed Report.
    - File: Opens Errors by File Detailed Report.
    - Developer: Opens Errors by Developer Detailed Report.

To view more details about the errors listed by severity:

- Click the number of errors that corresponds with the appropriate rule category. A page similar to Figure 33, which contains even more details about the errors appears:

Figure 33: Errors by Severity Detailed Report (More Details)



Severity [Category] [File] [Developer]

Query used to create original (not filtered) data has reached 20000 rows limit. Some results may not be visible.

Severity Level 1

Global Static Analysis

File name	Developer	Line	Error	Rule
AdminManagerException.java	ksiazek	38	Field '_cause' should be declared "private"	GLOBAL.DPAF
GenericDropTarget.java	jakubiak	63	Class 'com.parasoft.dnd.GenericDropTarget.DropFlavor' should be declared "private"	GLOBAL.DPAC
NSOnlineHelp.java	jakubiak	193	Class 'com.parasoft.help.NSOnlineHelp.HelpKeyAdapter' should be declared "private"	GLOBAL.DPAC
ScenarioAttachmentDisplayServlet.java	dariuszo	35	Field '_connectionProvider' should be declared "private"	GLOBAL.DPAF
ScenarioAttachmentUploadServlet.java	dariuszo	33	Field '_connectionProvider' should be declared "private"	GLOBAL.DPAF
StepStateAttachmentMetaInfo.java	dariuszo	27	Field 'sequence' should be declared "private"	GLOBAL.DPAF

How do I get here? *Tests > Errors by Severity > [date] (from graph) > [number of errors]*

Following are descriptions of each column heading in the Errors by Severity Detailed Report (more details) table, along with drill down options:

- **Filename:** Name of the file that contains the error.

  Click the Filename heading to open Errors by File Detailed Report. Click a specific filename listed beneath the heading to open its source file.

- **Developer:** Username of the developer responsible for the corresponding error.

  Click the Developer heading to open Errors by Developer Detailed Report.

- **Line:** Number of the line on which the error is located in the corresponding error.

  Click a specific line to open the corresponding source file with this line highlighted in yellow.

- **Error:** Description of the error that occurred.

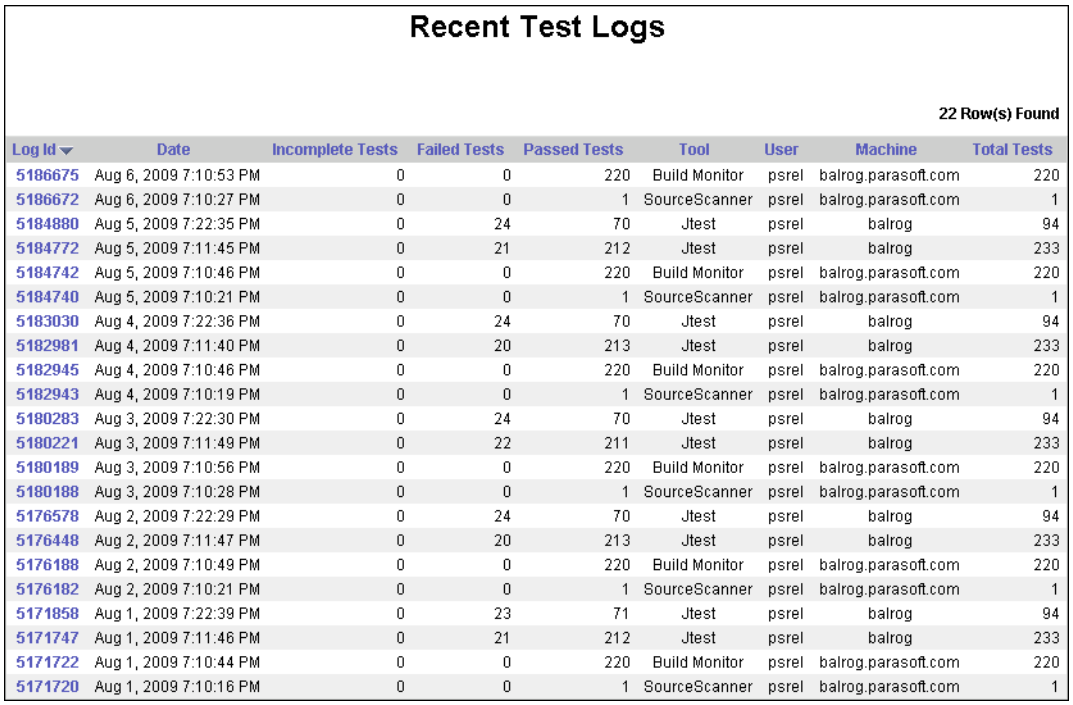
- **Rule:** Name of the rule that was violated in the corresponding error.

  Click a specific rule to open a pop-up box that contains a description of that rule, along with any relevant notes, security relevance, parameters, benefits, drawbacks, examples, repairs, and references.

Recent Test Logs

The Recent Test Logs report (Figure 34) lists all logs that were successfully sent by tools to the Report Center database for the selected project and time period. It shows whether logs were successfully retrieved by the Report Center database and contains statistics for administrators, QA team members, or other designated parties who are responsible for sending logs to Report Center. If logs were not successfully retrieved, you should check the Report Center tool configuration, and then run the tests again.

Figure 34: Recent Test Logs



Recent Test Logs

22 Row(s) Found

Log Id	Date	Incomplete Tests	Failed Tests	Passed Tests	Tool	User	Machine	Total Tests
5186675	Aug 6, 2009 7:10:53 PM	0	0	220	Build Monitor	psrel	balrog.parasoft.com	220
5186672	Aug 6, 2009 7:10:27 PM	0	0	1	SourceScanner	psrel	balrog.parasoft.com	1
5184880	Aug 5, 2009 7:22:35 PM	0	24	70	Jtest	psrel	balrog	94
5184772	Aug 5, 2009 7:11:45 PM	0	21	212	Jtest	psrel	balrog	233
5184742	Aug 5, 2009 7:10:46 PM	0	0	220	Build Monitor	psrel	balrog.parasoft.com	220
5184740	Aug 5, 2009 7:10:21 PM	0	0	1	SourceScanner	psrel	balrog.parasoft.com	1
5183030	Aug 4, 2009 7:22:36 PM	0	24	70	Jtest	psrel	balrog	94
5182981	Aug 4, 2009 7:11:40 PM	0	20	213	Jtest	psrel	balrog	233
5182945	Aug 4, 2009 7:10:46 PM	0	0	220	Build Monitor	psrel	balrog.parasoft.com	220
5182943	Aug 4, 2009 7:10:19 PM	0	0	1	SourceScanner	psrel	balrog.parasoft.com	1
5180283	Aug 3, 2009 7:22:30 PM	0	24	70	Jtest	psrel	balrog	94
5180221	Aug 3, 2009 7:11:49 PM	0	22	211	Jtest	psrel	balrog	233
5180189	Aug 3, 2009 7:10:56 PM	0	0	220	Build Monitor	psrel	balrog.parasoft.com	220
5180188	Aug 3, 2009 7:10:28 PM	0	0	1	SourceScanner	psrel	balrog.parasoft.com	1
5176578	Aug 2, 2009 7:22:29 PM	0	24	70	Jtest	psrel	balrog	94
5176448	Aug 2, 2009 7:11:47 PM	0	20	213	Jtest	psrel	balrog	233
5176188	Aug 2, 2009 7:10:49 PM	0	0	220	Build Monitor	psrel	balrog.parasoft.com	220
5176182	Aug 2, 2009 7:10:21 PM	0	0	1	SourceScanner	psrel	balrog.parasoft.com	1
5171858	Aug 1, 2009 7:22:39 PM	0	23	71	Jtest	psrel	balrog	94
5171747	Aug 1, 2009 7:11:46 PM	0	21	212	Jtest	psrel	balrog	233
5171722	Aug 1, 2009 7:10:44 PM	0	0	220	Build Monitor	psrel	balrog.parasoft.com	220
5171720	Aug 1, 2009 7:10:16 PM	0	0	1	SourceScanner	psrel	balrog.parasoft.com	1

How do I get here? *Tests > Recent Test Logs*

The Recent Test Logs table shows the following information:

- **Log ID:** Identification number assigned to the listed log. Click to open Log Details.
- **Date:** Shows the date of listed log.
- **Incomplete tests:** Number of tests that were incomplete in the corresponding log.
- **Failed Tests:** Number of tests that failed in the corresponding log.
- **Passed Tests:** Number of tests that passed in the corresponding log.
- **Tool:** Name of the tool from which the log was sent to the Report Center database.
- **User:** Name of the user who sent the log.
- **Total Tests:** Total number of tests that were run in the corresponding log.

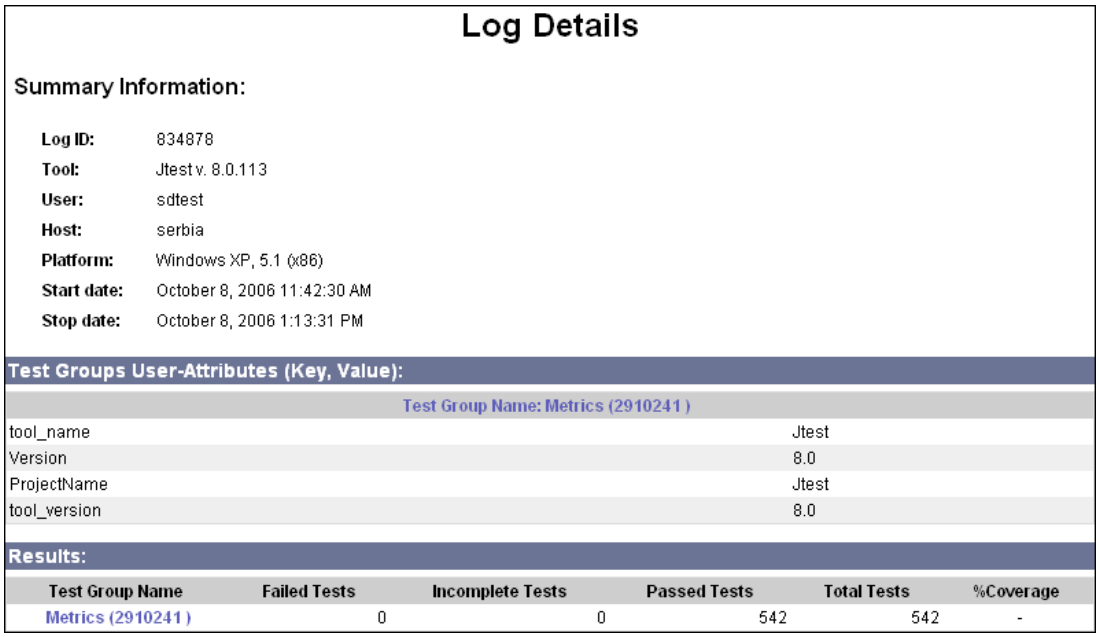
You can sort the data displayed in the Recent Test Logs table by clicking any column header.

Log Details

The Log Details page shows log information such as ID, tool, host, start and stop date, and the test groups included in this log. The following information is shown for each group of tests that ran:

- Test group user attributes and test results.
- Number of failed, incomplete, passed, and total tests performed.
- Percentage of line coverage (it is presented for Unit Testing results).

Figure 35: Log Details



Log Details

Summary Information:

Log ID: 834878
Tool: Jtest v. 8.0.113
User: sdtest
Host: serbia
Platform: Windows XP, 5.1 (x86)
Start date: October 8, 2006 11:42:30 AM
Stop date: October 8, 2006 1:13:31 PM

Test Groups User-Attributes (Key, Value):

Test Group Name: Metrics (2910241)	
tool_name	Jtest
Version	8.0
ProjectName	Jtest
tool_version	8.0

Results:

Test Group Name	Failed Tests	Incomplete Tests	Passed Tests	Total Tests	%Coverage
Metrics (2910241)	0	0	542	542	-

How do I get here? *Tests > Recent Test Logs > [Log ID]*

The following information is displayed on the Log Details page:

- **Log ID:** Identification number assigned to the log.
- **Tool:** Name of the tool from which the log was sent to the Report Center database.
- **User:** Name of the user who sent the log.
- **Host:** The name of the computer from which the log was sent.
- **Platform:** Platform on which the log was created.
- **Start date:** The date on which testing represented by this log began.
- **Stop date:** The date on which testing represented by this log finished.
- **Test Groups User Attributes (Key, Value):** Lists the key and value of all test groups included in the log.
- **Test Group Name:** Name assigned to a group of tests. Click to open the Test Group Details page. See “Test Group Details” for details.
- **Failed Tests:** Number of tests that failed within the group.
- **Incomplete Tests:** Number of tests that were incomplete within the group.

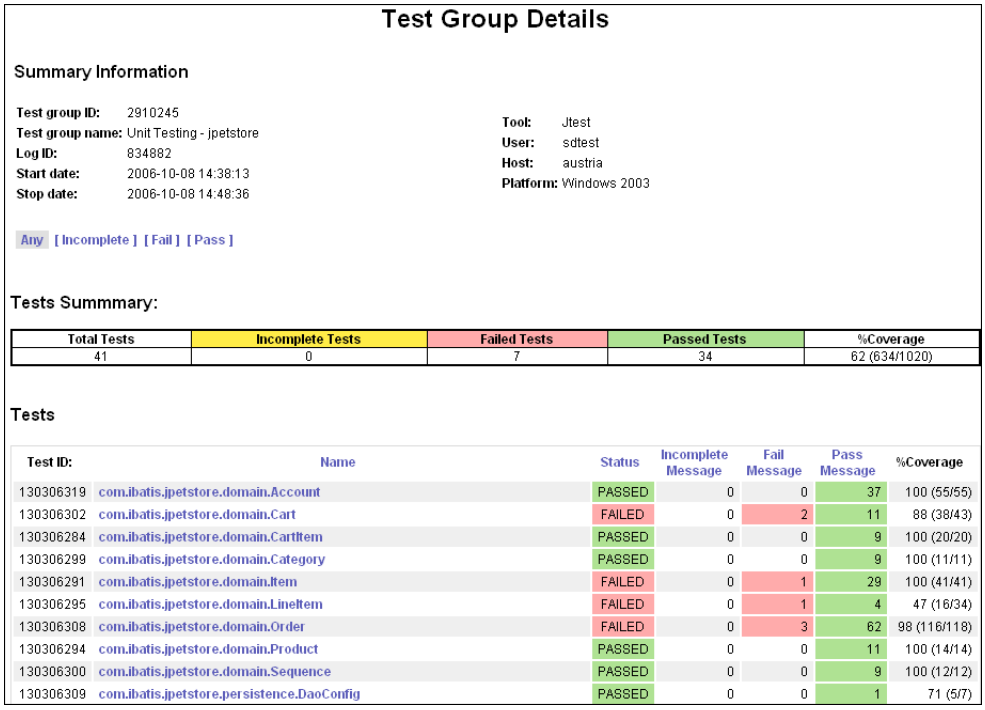
- **Passed Tests:** Number of tests that passed within the group.
- **Total Tests:** Number of total tests within the group.
- **%Coverage:** Percentage of unit testing line coverage (when applicable).

Note*: Coverage applies to unit testing.*

Test Group Details

The Test Group Details report (Figure 36) displays identifying details about the group of tests, along with information about the tool, host, and platform on which that group of tests were run. It also provides a list of each test included in the group.

Figure 36: Test Group Details



Test Group Details

Summary Information

Test group ID: 2910245
Test group name: Unit Testing - jpetstore
Log ID: 834882
Start date: 2006-10-08 14:38:13
Stop date: 2006-10-08 14:48:36

Tool: Jtest
User: sdtest
Host: austria
Platform: Windows 2003

Any [Incomplete] [Fail] [Pass]

Tests Summmary:

Total Tests	Incomplete Tests	Failed Tests	Passed Tests	%Coverage
41	0	7	34	62 (634/1020)

Tests

Test ID:	Name	Status	Incomplete Message	Fail Message	Pass Message	%Coverage
130306319	com.ibatis.jpetstore.domain.Account	PASSED	0	0	37	100 (55/55)
130306302	com.ibatis.jpetstore.domain.Cart	FAILED	0	2	11	88 (38/43)
130306284	com.ibatis.jpetstore.domain.CartItem	PASSED	0	0	9	100 (20/20)
130306299	com.ibatis.jpetstore.domain.Category	PASSED	0	0	9	100 (11/11)
130306291	com.ibatis.jpetstore.domain.Item	FAILED	0	1	29	100 (41/41)
130306295	com.ibatis.jpetstore.domain.LineItem	FAILED	0	1	4	47 (16/34)
130306308	com.ibatis.jpetstore.domain.Order	FAILED	0	3	62	98 (116/118)
130306294	com.ibatis.jpetstore.domain.Product	PASSED	0	0	11	100 (14/14)
130306300	com.ibatis.jpetstore.domain.Sequence	PASSED	0	0	9	100 (12/12)
130306309	com.ibatis.jpetstore.persistence.DaoConfig	PASSED	0	0	1	71 (5/7)

How do I get here? *Tests > Recent Test Logs > [date] > [test group name]*

The following information is displayed on the Test Group Details page:

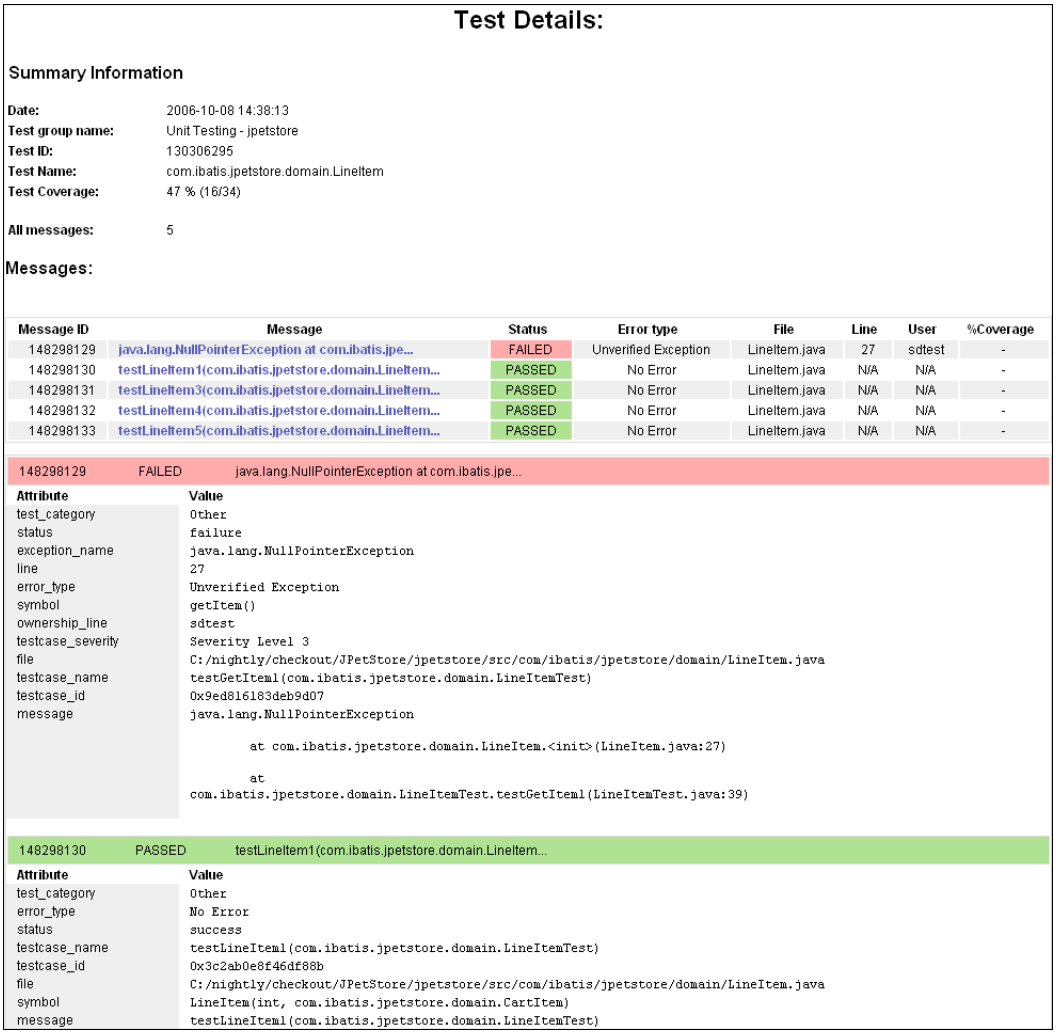
- **Test group ID:** Identification number assigned to the group of tests.
- **Test group name:** Name assigned to a group of tests.
- **Log ID:** Identification number assigned to the log.
- **Start date:** Date on which tests included in this group started running.
- **Stop date:** Date on which tests included in this group stopped running.
- **Tool:** Name of the tool from which the log was sent to the Report Center database.
- **User:** Name of the user who sent the log.
- **Host:** The name of the computer from which the log was sent.
- **Platform:** Platform on which the group of tests were run.

- **Total Tests:** Complete number of tests included within the group.
- **Incomplete Tests:** Number of tests that were incomplete within the group.
- **Failed Tests:** Number of tests that failed within the group.
- **Passed Tests:** Number of tests that passed within the group.
- **%Coverage:** Percentage of unit testing line coverage (when applicable)
- **Test ID:** Identification number assigned to the test included in the displayed group.
- **Name:** Name of the test included in the displayed group. Click to open the Test Details page, which provides more details about the test. See “Test Details” for details.
- **Status:** Status of the test included in the displayed group.
- **Incomplete Message:** Number of incomplete messages generated by the test.
- **Fail Message:** Number of failed messages generated by the test.
- **Pass Message:** Number of passed messages generated by the test.
- **%Coverage:** Percentage of unit testing line coverage (when applicable).

Test Details

The Test Details report (Figure 37) provides information about the selected test, along with a detailed list of all messages sent in the log.

Figure 37: Test Details



Test Details:

Summary Information

Date:	2006-10-08 14:38:13
Test group name:	Unit Testing - jpetstore
Test ID:	130306295
Test Name:	com.ibatis.jpetstore.domain.LineItem
Test Coverage:	47 % (16/34)
All messages:	5

Messages:

Message ID	Message	Status	Error type	File	Line	User	%Coverage
148298129	java.lang.NullPointerException at com.ibatis.jpe...	FAILED	Unverified Exception	LineItem.java	27	sdtest	-
148298130	testLineItem1(com.ibatis.jpetstore.domain.LineItem...	PASSED	No Error	LineItem.java	N/A	N/A	-
148298131	testLineItem3(com.ibatis.jpetstore.domain.LineItem...	PASSED	No Error	LineItem.java	N/A	N/A	-
148298132	testLineItem4(com.ibatis.jpetstore.domain.LineItem...	PASSED	No Error	LineItem.java	N/A	N/A	-
148298133	testLineItem5(com.ibatis.jpetstore.domain.LineItem...	PASSED	No Error	LineItem.java	N/A	N/A	-

148298129 FAILED java.lang.NullPointerException at com.ibatis.jpe...

Attribute	Value
test_category	Other
status	failure
exception_name	java.lang.NullPointerException
line	27
error_type	Unverified Exception
symbol	getItem()
ownership_line	sdtest
testcase_severity	Severity Level 3
file	C:/nightly/checkout/JPetStore/jpetstore/src/com/ibatis/jpetstore/domain/LineItem.java
testcase_name	testGetItem1(com.ibatis.jpetstore.domain.LineItemTest)
testcase_id	0x9ed816183deb9d07
message	java.lang.NullPointerException at com.ibatis.jpetstore.domain.LineItem.<init>(LineItem.java:27) at com.ibatis.jpetstore.domain.LineItemTest.testGetItem1(LineItemTest.java:39)

148298130 PASSED testLineItem1(com.ibatis.jpetstore.domain.LineItem...

Attribute	Value
test_category	Other
error_type	No Error
status	success
testcase_name	testLineItem1(com.ibatis.jpetstore.domain.LineItemTest)
testcase_id	0x3c2ab0e8f46df88b
file	C:/nightly/checkout/JPetStore/jpetstore/src/com/ibatis/jpetstore/domain/LineItem.java
symbol	LineItem(int, com.ibatis.jpetstore.domain.CartItem)
message	testLineItem1(com.ibatis.jpetstore.domain.LineItemTest)

How do I get here? *Tests > Recent Test Logs > [date] > [test group name] > [name] (test name)*

The following information is displayed on the Test Details page:

- **Date:** Date on which the displayed test was run.
- **Test group name:** Name assigned to the group to which the displayed test belongs.
- **Test ID:** Identification number assigned to the displayed test.
- **Test name:** Name assigned to the displayed test.
- **Test Coverage:** Line coverage in this test (file).
- **All messages:** Number of messages generated by running the displayed test.
- **Message ID:** Identification number assigned to the message.
- **Message:** Summary text generated by running the displayed test. Click to jump to details about the message—specifically, a list of attributes, along with values for each.
- **Status:** Status of the message—passed, failed, incomplete.

- **Error type:** Type of error generated by the test. If none, "No Error" is displayed.
- **File:** Name of the file where the message is stored.
- **User:** Name of the user who ran the displayed test.
- **%Coverage:** %Coverage: Percentage of unit testing line coverage (when applicable).

Change-Based Testing

The Change-Based Testing reports indicate what you should re-test in response to recent source code changes.

This change-based testing identifies exactly which tests, requirements, tasks, and defects are impacted by source code changes. Not having to re-test the entire system after each modification yields tremendous productivity improvements.

You can determine which tests require re-testing in the following ways:

- To see which manual test scenarios should be retested due to changes in correlated code, **c**hoose **Tests> Change-Based Testing> Test Scenarios**.
- To see which requirements/defects should be retested due to changes in correlated code, choose **Tests> Change-Based Testing> Requirements/Defects**.

How are Tests Recommended?

Development Testing Platform tracks what source code is related to project functionality If this source code changes, Development Testing Platform knows that the related implementation should be re-tested to check whether the changes introduced any regressions.

While developers implement requirements and fix defects, Development Testing Platform monitors the related source code and associates it with the corresponding requirements/defects. The resulting functionality is then typically tested by QA.

Often, developers later modify portions of the code that are related to this previously-developed and tested work. These modifications may change or break the previously-verified functionality.

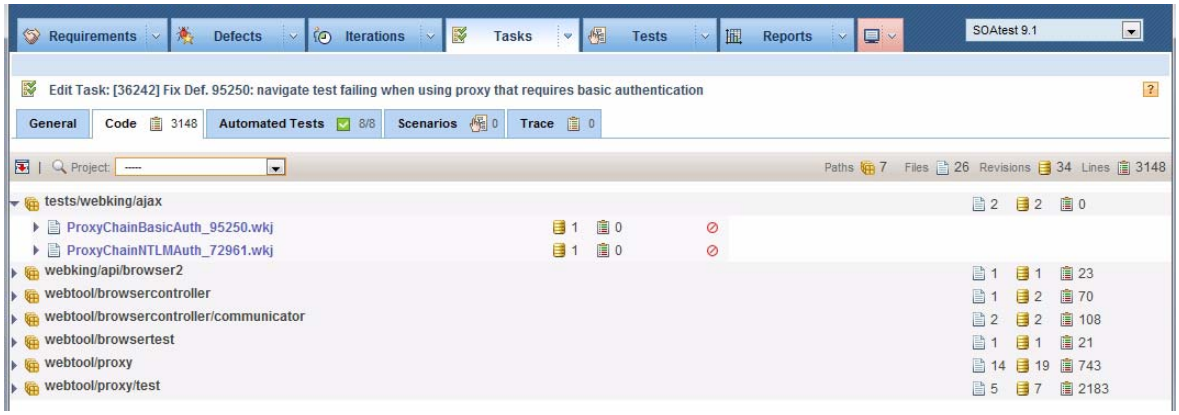
The Recommended Tests reports help the team identify which functionality needs to be re-tested in order to verify that the functionality is still working as expected after the code changes. Project artifacts (requirements, defects, enhancements) are marked as needed re-testing when the source code associated with them has been changed within a specified timeframe. Re-testing tasks are prioritized; the highest priority is assigned to the items with the most source code modifications.

How is Source Code Tracked?

Development Testing Platform tracks source code that has been developed or modified when implementing specific functionality. Code can be tracked in two ways:

- Developers add special @task taskID annotations during source control checkins (see Work with Tasks for more details)
- Developers work with tasks via the Development Testing Platform Task Assistant (available in Parasoft Test).

Development Testing Platform reports code correlations in the **Code** tab of the particular artifact (requirement, defect, enhancement) details page.



When tracked source code changes, Development Testing Platform can determine what specific functionality (requirements, defects, manual tests) should be re-tested, and alerts you via the Recommended Tests report.

The Recommended Tests report shows you what functionality should be re-tested from two perspectives:

- Recommended Tests by Test
- Recommended Tests by Requirement/Defect

Change-Based Testing: Test Scenarios

Assumes usage of Development Testing Platform Manual Testing - described in "Tests in Project Center", page 157.

This report indicates which manual test scenarios should be retested because there have been changes in the source code that was correlated to the previously-tested functionality. In other words, if a manual test scenario A was correlated to requirement B and the source code related to requirement B has changed, manual test scenario A will be marked as a recommended test.

The following Manual Test Scenarios should be retested:

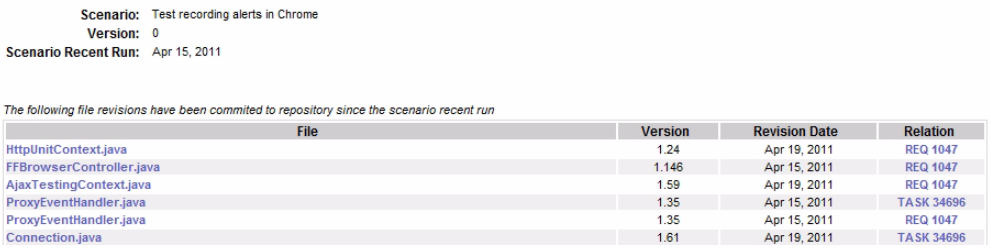
Scenario/Manual Test	Last run status	Executed on	Performed by	Test duration	Relations	Priority	Retest
Test recording alerts in Chrome	Pass	Apr 15, 2011	jakubiak	1m	REQ 1047 TASK 34696	Very high	
Validate Browser Contents Correct after Recording in Chrome	Pass	Apr 15, 2011	jakubiak	1m	REQ 1047 TASK 33779	Medium	
Test recording confirm dialogs in Chrome	Pass	Apr 15, 2011	jakubiak	1m	REQ 1047 TASK 34696	Medium	
Test recording prompt dialogs in Chrome	Pass	Apr 15, 2011	jakubiak	1m	REQ 1047 TASK 34696	Low	
Task 32257 - PR 87984: Test SOAP Client tool configuration wizard	Pass	Apr 15, 2011	truong	0d 0h 0m	FR 87984 TASK 32257	Low	
Task 34339 - Chrome - Test recording with onbeforeunload dialogs	Pass	Apr 15, 2011	jakubiak	0d 0h 0m	REQ 1047 TASK 34339	Very low	
Task 34487 - Handle case where Chrome process started when a leftover process still exists	Not Run				REQ 1047 TASK 34487	Very low	
Task 33781 - Support headless mode in Chrome	Not Run				REQ 1047 TASK 33781	Very low	
Task 36462 - PR 94925 - Javascript parsing error on object literal containing regex	Not Run				PR 94925 TASK 36462	Very low	

The **Priority** column sorts tests by priority. The more changes that are introduced to the specific functionality related to a test, the higher the priority is.

The **Relations** column shows project artifacts with which the scenario is linked.

The report date period specifies the period for which code changes are considered. You can adjust the date range by clicking the **Last 7 | 15 | 30 days Last 12 | 26 | 52 weeks** area—or by clicking the **Switch to range mode** icon then specifying the desired to/from date range.

Clicking the priority cell value will display code details. For example:



Scenario: Test recording alerts in Chrome
Version: 0
Scenario Recent Run: Apr 15, 2011

The following file revisions have been commited to repository since the scenario recent run

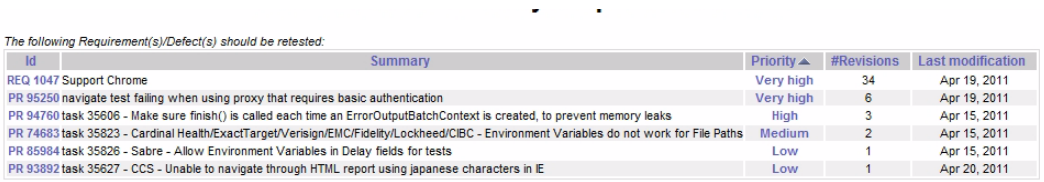
File	Version	Revision Date	Relation
HttpUnitContext.java	1.24	Apr 19, 2011	REQ 1047
FFBrowserController.java	1.146	Apr 15, 2011	REQ 1047
AjaxTestingContext.java	1.59	Apr 19, 2011	REQ 1047
ProxyEventHandler.java	1.35	Apr 15, 2011	TASK 34696
ProxyEventHandler.java	1.35	Apr 15, 2011	REQ 1047
Connection.java	1.61	Apr 19, 2011	TASK 34696

The details report presents all the source code revisions that:

- Are associated with specific functionality and
- Were committed to source control after the recent manual test run.

Change-Based Testing: Requirements/Defects

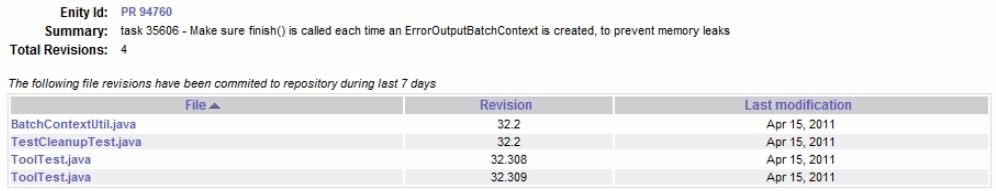
This report indicates which requirements or defects should be retested because there have been changes in the source code that was correlated to them. In other words, if code correlated to requirement A has changed, requirement A will be marked as needing re-testing.



The following Requirement(s)/Defect(s) should be retested:

Id	Summary	Priority ▲	#Revisions	Last modification
REQ 1047	Support Chrome	Very high	34	Apr 19, 2011
PR 95250	navigate test failing when using proxy that requires basic authentication	Very high	6	Apr 19, 2011
PR 94760	task 35606 - Make sure finish() is called each time an ErrorOutputBatchContext is created, to prevent memory leaks	High	3	Apr 15, 2011
PR 74683	task 35823 - Cardinal Health/ExactTarget/Verisign/EMC/Fidelity/Lockheed/CIBC - Environment Variables do not work for File Paths	Medium	2	Apr 15, 2011
PR 85984	task 35826 - Sabre - Allow Environment Variables in Delay fields for tests	Low	1	Apr 15, 2011
PR 93892	task 35627 - CCS - Unable to navigate through HTML report using japanese characters in IE	Low	1	Apr 20, 2011

Clicking the priority cell value will display details about the entity that changed. For example:



Enity Id: PR 94760
Summary: task 35606 - Make sure finish() is called each time an ErrorOutputBatchContext is created, to prevent memory leaks
Total Revisions: 4

The following file revisions have been commited to repository during last 7 days

File ▲	Revision	Last modification
BatchContextUtil.java	32.2	Apr 15, 2011
TestCleanupTest.java	32.2	Apr 15, 2011
ToolTest.java	32.308	Apr 15, 2011
ToolTest.java	32.309	Apr 15, 2011

The details report presents all the source code revisions that:

- Are associated with specific requirement/defect and
- Were committed to source control in the specified timeframe.

Code Metrics Reports

Development Testing Platform includes reports that show source code metrics data (statistics), such as cyclomatic complexity, number of classes, number of methods, number of static attributes, and more. Metrics are calculated in code analysis tools, and the results are sent to Development Testing Platform. The results provide details for each code level (package, class, method), enable you to display documentation for each metric, and view source code to which the results apply. The following metrics reports are available:

- Metric Top Results
- Single Metrics Overview (9.x)
- Metrics Overview Report
- Single Metric Overview Report

Metric Top Results

The Metric Top Results report presents data DTP collects about the packages, classes, and methods of your projects that have the highest metric values. By default, the system lists twenty items displayed for each level, but this setting can be changed, if necessary.

Choose **Audit> Metric Top Results** in the reports view to open the Metric Top Results report.

Figure 38: Metric Top Results - Packages

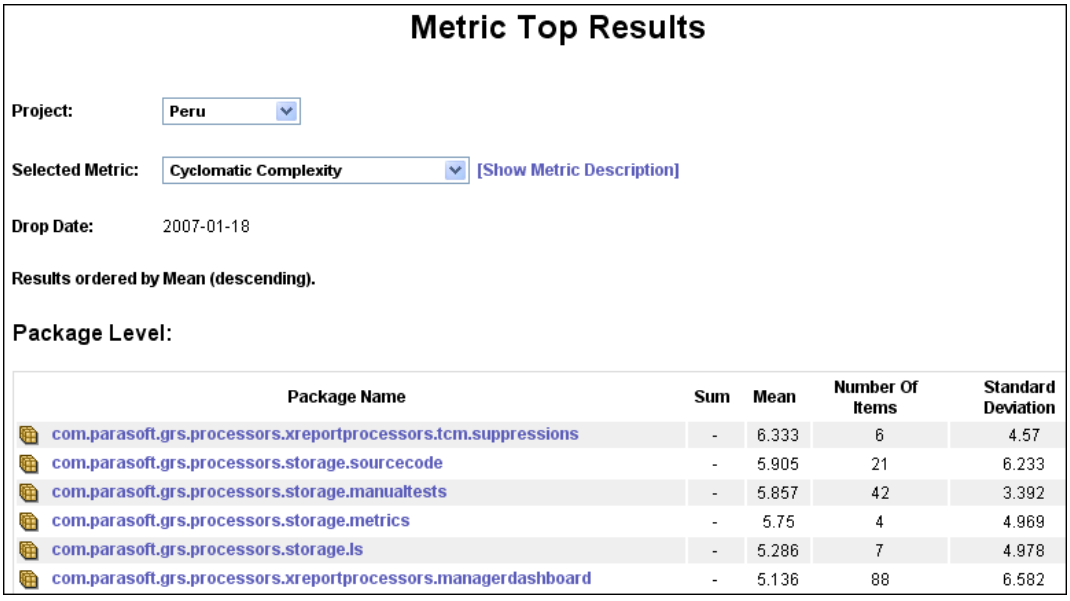
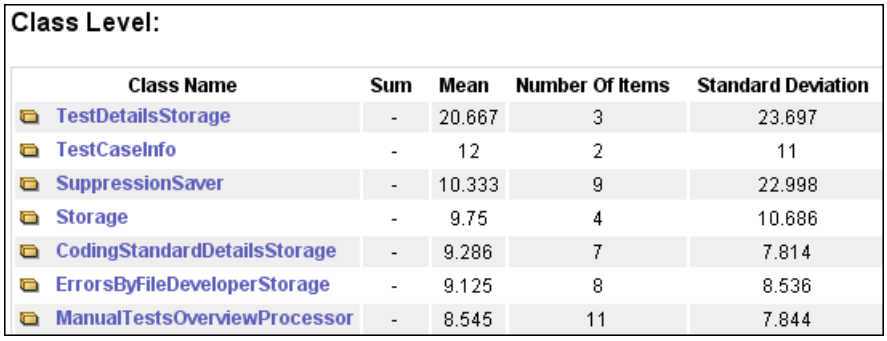


Figure 39: Metric Top Results - Classes



Class Level:

Class Name	Sum	Mean	Number Of Items	Standard Deviation
TestDetailsStorage	-	20.667	3	23.697
TestCaseInfo	-	12	2	11
SuppressionSaver	-	10.333	9	22.998
Storage	-	9.75	4	10.686
CodingStandardDetailsStorage	-	9.286	7	7.814
ErrorsByFileDeveloperStorage	-	9.125	8	8.536
ManualTestsOverviewProcessor	-	8.545	11	7.844

Figure 40: Metric Top Results - Methods

Method Level:

Method Signature	Sum	Mean	Number Of Items	Standard Deviation
saveSuppression(long, java.sql.Timestamp, int, int, int, int, int, int, int, int, int, long, int, java.lang.String)	75	75	1	0
getDetails(com.parasoft.grs.processors.xreportprocessors.TestDetails$PropertiesWrapper, com.parasoft.grs.processors.xreportprocessors.TestDetails$TestSummaryInfo)	54	54	1	0
prepareDataElement(com.parasoft.grs.rserver.xreportengine.XReportDocument, java.util.Map, com.parasoft.grs.rserver.xreportengine.XReportRuntimeProperties)	48	48	1	0
createMaxUsageData(com.parasoft.grs.rserver.xreportengine.XReportDocument,				

The Metric Top Results report presents the following information:

- **Project:** Name of a project defined and sent by a Parasoft tool.
- **Selected Metric:** Name of the metric for which values are displayed.

  [Show Metric Description]: Click to open a pop-up window that shows a description of the currently selected metric.
- **Drop date:** Date of the last metric drop. In other words, the last date on which metric analysis for the selected project was performed.
- **Name/Signature:** Name of tested package/class, or signature of tested method.
- **Sum:** Sum of the metric value of the tested object for the selected metric. In many metrics, this field is empty.
- **Mean:** Arithmetic mean of the tested object for the selected metric.
- **Number of items:** Number of children nodes that belong to a specific tested object.
- **Standard Deviation:** Statistical standard deviation of a specific metric result.

When you click any item to drill down the report, you are moved to the appropriate page of Single Metric report. See “Single Metrics Overview (9.x)”, page 97.

Single Metrics Overview (9.x)

The Single Metrics Overview report presents data collected from Parasoft 9.x tools for an individual project and one of its metrics. This report does not include data from DTP Engines.

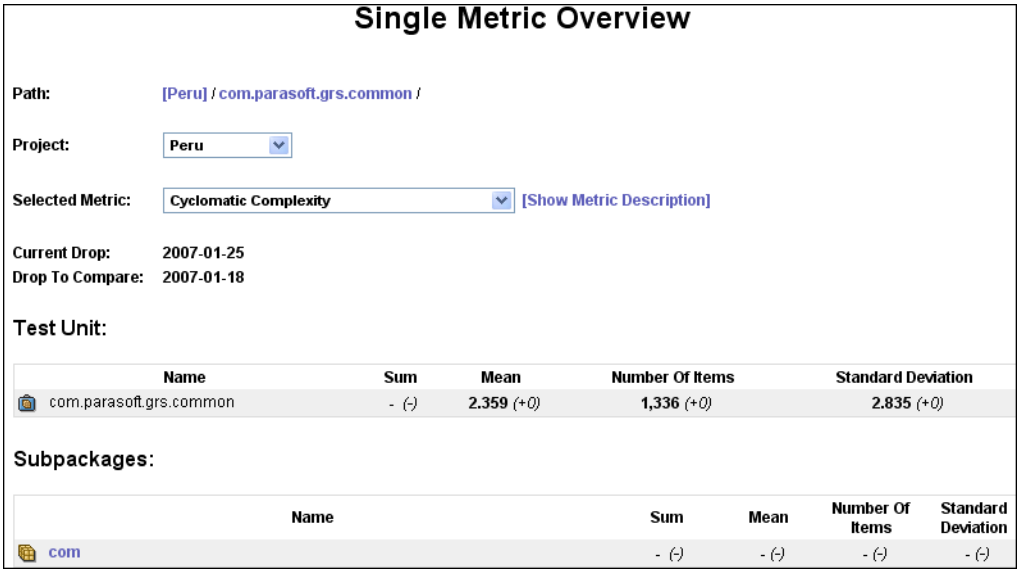
1. Choose **Audit> Single Metrics Overview** in the reports view to open the Single Metrics Overview report.
2. Choose a project and a metric from the drop down menus.

Test units (packages, classes, methods) are organized in a form of a tree so that you can drill down from the project packages to the single method. For each test unit, the following four metric values are presented:

- Sum
- Mean
- Number of items
- Standard deviation

These values are listed in the table next to each tested object respectively.

Figure 41: Single Metric Overview



The Single Metrics Overview report presents the following information:

- **Path:** Indicates where you are in the source hierarchy (on which level, which node within the code source tree)
- **Project:** Name of a project defined and sent by a Parasoft tool.
- **Selected Metric:** Name of the metric for which values are displayed.

  [Show Metric Description]: Click to open a pop-up window that shows a description of the currently selected metric.
- **Current drop:** Date of the last metric drop. In other words, the last date on which metric analysis for the selected project was performed.
- **Drop to compare:** Date of the drop which will be used to compare to current drop values
  - If more then 10 drops are available, then 10th oldest drop is taken to compare.
  - If less then 10 drops are available, then the oldest one is taken to compare.

The Single Metric Overview report includes the following tables:

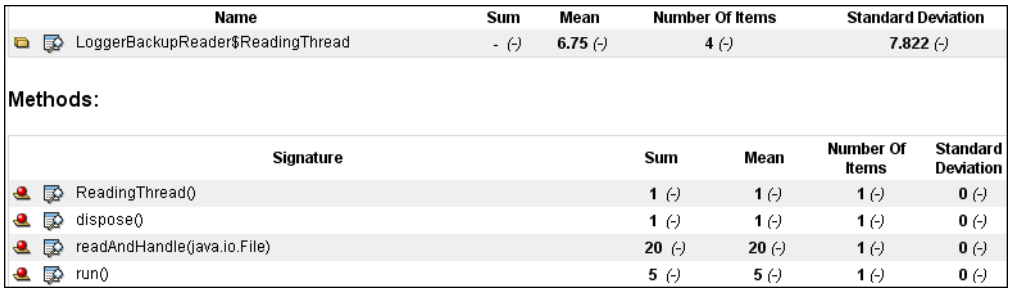
- **Test Unit/Package/Class table:** Displays results for the parent tested object—the parent tested object of the children nodes listed in the table shown in Figure 42.
- **Subpackages (Optional) table:** Displayed when the parent tested object within a package contains subpackages. It lists metric values for subpackages and can be used to drill them down to a single method.

The metric values contain the following info:

- **Name:** Name of the tested object.
- **Sum:** Sum of the metric value of the tested object for the selected metric. In many metrics, this field is empty.
- **Mean:** Arithmetic mean of the tested object for the selected metric.
- **Number of items:** Number of children nodes that belong to a specific tested object.
- **Standard Deviation:** Statistical standard deviation of a specific metric result.

Note: *The number in brackets indicates the difference between the last drop and drop to compare.*

Figure 42: View Source Code



Name	Sum	Mean	Number Of Items	Standard Deviation
LoggerBackupReader$ReadingThread	- (-)	6.75 (-)	4 (-)	7.822 (-)

Methods:

Signature	Sum	Mean	Number Of Items	Standard Deviation
ReadingThread()	1 (-)	1 (-)	1 (-)	0 (-)
dispose()	1 (-)	1 (-)	1 (-)	0 (-)
readAndHandle(java.io.File)	20 (-)	20 (-)	1 (-)	0 (-)
run()	5 (-)	5 (-)	1 (-)	0 (-)

A magnifying glass icon is displayed next to each class and method. Click on it to view the source code of that code item.

Note: *The magnifying glass icon is visible for project sources after SourceScanner has been run for those particular project sources.*

Metrics Overview Report

Click the **more...** link in the Metrics Overview widget to open the Metrics Overview report (see “Metrics Widgets”, page 12). This report shows an overview of metric data for the project selected in the widget.

Metrics Overview

Filter: SDM Platform

General

Metric	Value
Number of Files	5826
Number of Types	7119
Number of Blank Lines in File	119564
Number of Comment Lines in File	311817
Number of Physical Lines in File	949152
Number of Source Lines in File	-
Comments / Number of Logical Lines Ratio in File	2.17

Types/Functions

Metric	Average	Maximum
Cyclomatic Complexity	2.02	66
Essential Cyclomatic Complexity	1.34	57
Modified Cyclomatic Complexity	1.99	66
Strict Cyclomatic Complexity	2.14	86

Types

Metric	Average	Maximum
Coupling Between Objects	4.98	459
Inheritance Depth of Class	1.9	7
Number of Methods in Type	6.5	264
Number of Private Members in Type	3.97	130
Number of Protected Members in Type	0.49	43
Number of Public Members in Type	5.01	264

Only the following metrics are shown in the report:

- General metrics
    - Comments/Number of Logical Lines Ratio in the File
    - Number of Blank Lines in File
    - Number of Comment Lines in File
    - Number of Files
    - Number of Physical Lines in File
    - Number of Source Lines n File
    - Number of Types
- Types/Functions
    - Cyclomatic Complexity
    - Essential Cyclomatic Complexity
    - Modified Cyclomatic Complexity
    - Strict Cyclomatic Complexity
- Types
    - Coupling Between Objects

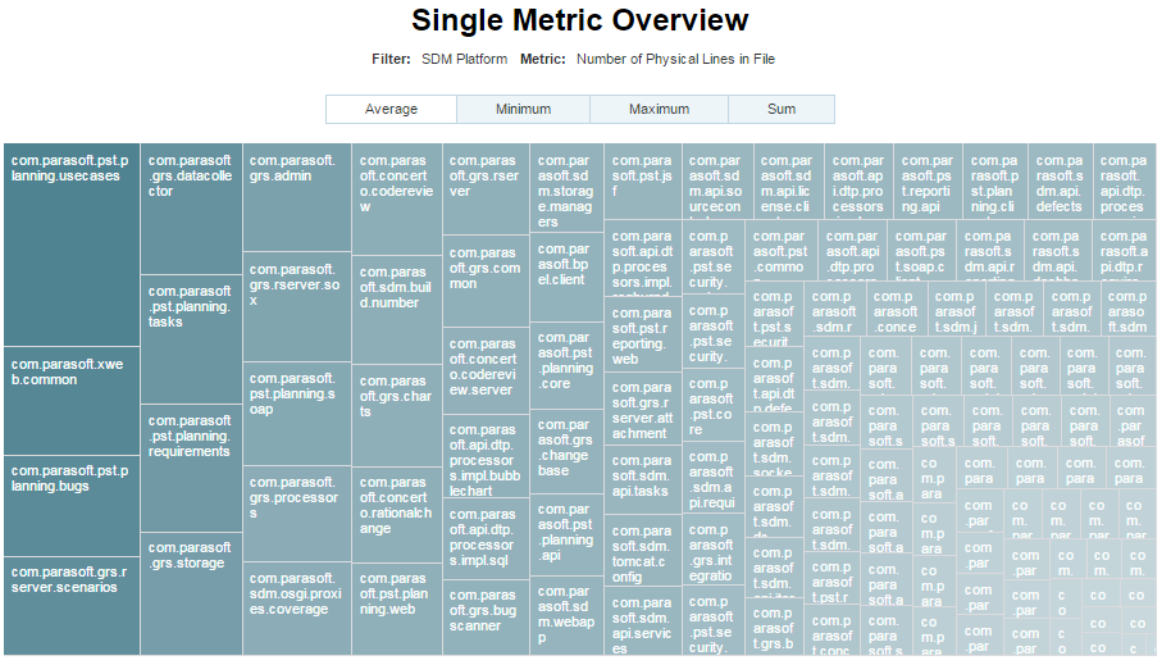
- Inheritance Depth of Class
- Number of Methods in Type
- Number of Private Members in Type
- Number of Protected Members in Type
- Number of Public Members in Type

You can perform the following actions in this report:

- Click on a column header to sort
- Click on a value in the table to open the Single Metric Overview report (see “Single Metric Overview Report”, page 101)

Single Metric Overview Report

This report shows how data for one metric is aggregated across the modules in a selected project. It shows a heat map in which larger, darker tiles contain a higher metric value within the project. Conversely, the smaller lighter tiles have lower metric values. The report also represents the data in a table view. You can access the report by clicking on a value in the Metrics Overview widget or by clicking the **more...** link in the Top 10 Modules - Tree Map widget (see “Metrics Widgets”, page 12).



Module	Avg ▼	Min	Max	Sum
com.parasoft.pst.planning.usecases	662.75	103	3624	7953
com.parasoft.xweb.common	348	46	650	696
com.parasoft.pst.planning.bugs	328.43	13	3269	4598
com.parasoft.grs.rserver.scenarios	325.57	42	1308	6837
com.parasoft.grs.datacollector	315.99	21	2439	23699

You can perform the following actions:

- View different aggregations of the data (method- and type-level metrics only): Click on **Average**, **Minimum**, **Maximum**, or **Sum** to change the heat map view. Changing the aggregation also sorts the table by the selected aggregation. For file-level metrics, only the sum aggregation is available.
- Click on a tile in the heat map or link in the Module column of the table to open the module in the Metrics Explorer view. See “Metrics Explorer”, page 140.
- Click on a column header in the table to sort the data (does not change the heat map).

Working with the Security Menu

Report Center offers you the ability to create security reports from the Security menu.

In this section:

- Security Violations
- Security Tests

Security Violations

To display a Security Violations report, choose **Security Violations** from the **Security** menu. The Security Violations report is displayed for the specific period of time.

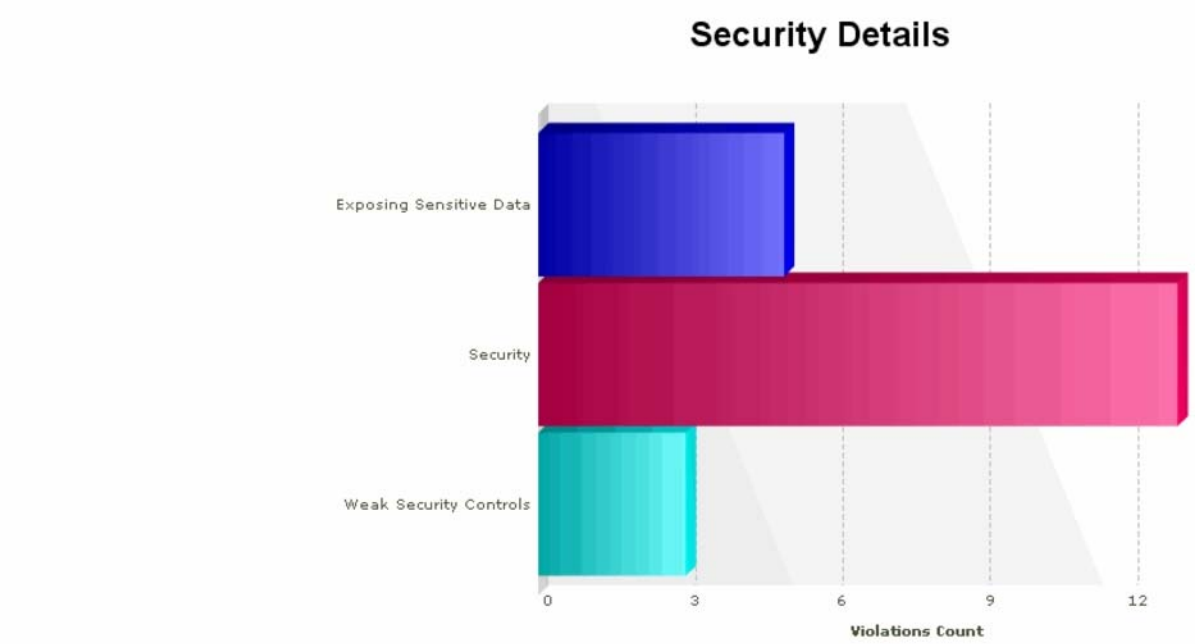
Figure 43: Security Violations Graph

The Security Violations report shows various categories of Security-type errors. Each bar in the Security Violations graph represents one day's test results for the selected project.

Drill-downs from the Security Violations graph:

You can also view the details of the Security Violations report, click any selected bar on the graph. Once you click on the selected bar, the Security Details page for the selected date displays.

Figure 44: Security Details



Sort by: File Line Owner Severity Message

Exposing Sensitive Data

File	Line	Owner	Severity	Message
AbstractRuleDocCreator.java	319	kelvin	5	Make sure 'System.out.println()' is not a leftover debug statement
BugDetectiveHtmlDocCreator.java	105	fedorov	5	Make sure 'System.out.println()' is not a leftover debug statement
Xml2Properties.java	113	kelvin	5	Make sure 'System.out.println()' is not a leftover debug statement
XmlFileDeprecatedRuleCategory.java	95	kelvin	5	Make sure 'System.out.println()' is not a leftover debug statement
XmlFileRuleCategory.java	81	kelvin	5	Make sure 'System.out.println()' is not a leftover debug statement

This page presents all the security error categories, along with errors belonging to every category. For each error the basic properties, like File Name, Owner, and Severity are shown.

Security Tests

The Security Tests report shows security-related tests performed in your project, including test resuts and trend over specified period of time. For static analysis, the number shown represents the number of files being tested with coding standard rules in the "Security" category.

Figure 45: Security Tests

Date	Incomplete	Fail	Pass
Sep 03, 2010	0	0	3604
Sep 01, 2010	0	0	1251
Aug 31, 2010	0	0	3604
Aug 27, 2010	0	0	3604
Aug 26, 2010	0	0	3604
Aug 25, 2010	0	0	2385
Aug 23, 2010	0	0	1674
Aug 20, 2010	0	0	3604
Aug 19, 2010	0	0	2391
Aug 18, 2010	0	0	3601

For each listed drop-date, the Security Tests report (Figure 45) shows the number of incomplete, failed, and passed security tests run.

Drill-down from Security Tests report:

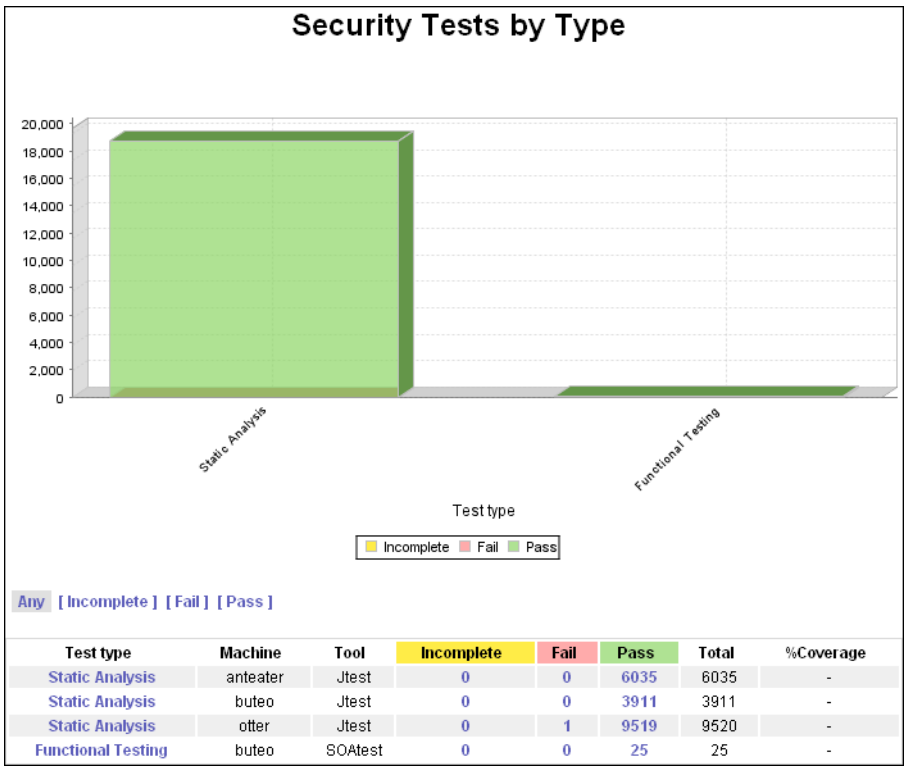
If there is a problem that warrants investigating, further drill-downs provide more details about the tests. Click the appropriate date, or corresponding Incomplete, Fail, or Pass data to open the Security Tests by Type report (Figure 46).

Security Tests by Type

To display a Security Tests by Type report, choose **Security Tests** from the **Security** menu. Then click on one of the date/incomplete/fail/pass table links.

The Security Tests by Type report shows each of the different types of tests that were run on the selected date. It breaks this information down further to show the exact machine and tool on which the tests were run.

Figure 46: Security Tests by Type: Pass



Any [Incomplete] [Fail] [Pass]

Test type	Machine	Tool	Incomplete	Fail	Pass	Total	%Coverage
Static Analysis	anteater	Jtest	0	0	6035	6035	-
Static Analysis	buteo	Jtest	0	0	3911	3911	-
Static Analysis	otter	Jtest	0	1	9519	9520	-
Functional Testing	buteo	SOAtest	0	0	25	25	-

Security Tests by Type Details

Drill-down from Security Tests by Type report:

You can continue to obtain more details about statistics for tests, rules, and test cases by clicking on the links in the Security Test by Type report. You can drill down to the group of tests run against the code, and then down to the exact erratic lines of code that contain errors from failed test cases.

Figure 47: Security Tests by Type Details

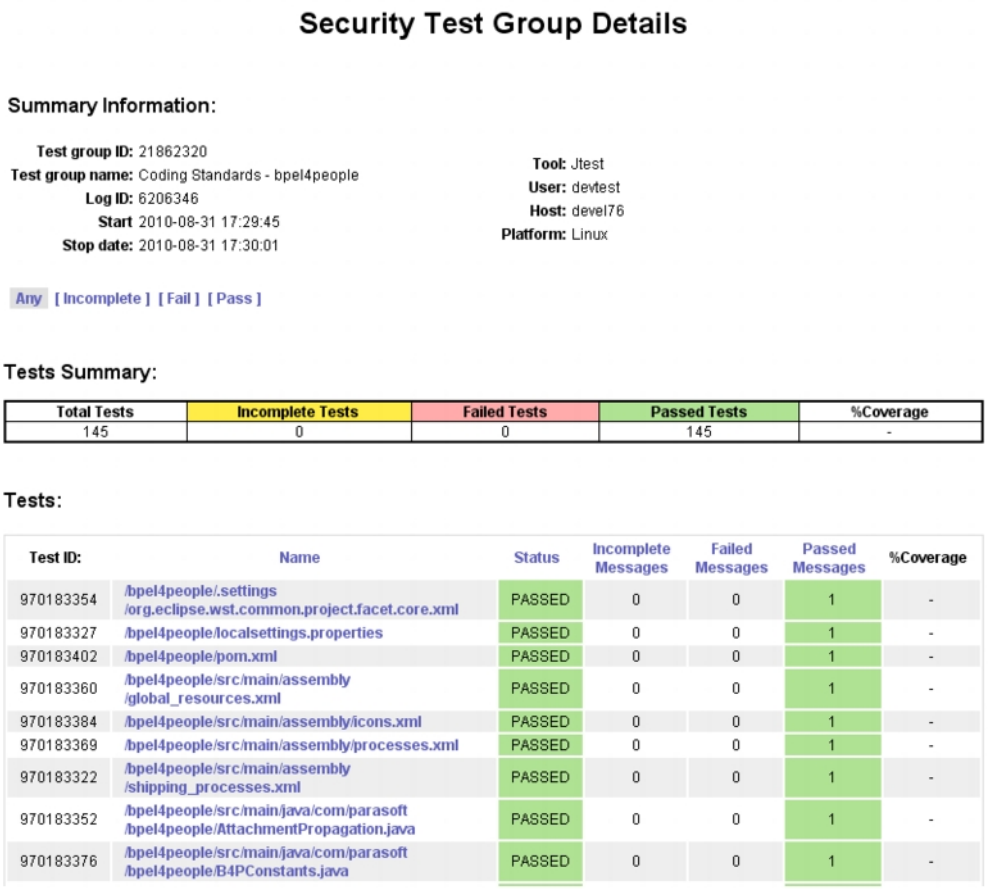
Security Tests by Type Details

Group Name	Machine	Tool	Incomplete	Fail	Pass
Coding Standards - com.parasoft.concerto.qc.client	gobos	Jtest	0	0	8
Coding Standards - com.parasoft.grs.admin	gobos	Jtest	0	0	57
Coding Standards - com.parasoft.grs.bugscanner	gobos	Jtest	0	0	108
Coding Standards - com.parasoft.grs.bugscanner.jiraclient	gobos	Jtest	0	0	2

Security Test Group Details

The Summary section lists each type of test. Click on a specific **Group Name** in the **Summary** section to navigate to **Security Test Group Details** where each individual test run is listed.

Figure 48: Security Test Group Details



Security Test Group Details

Summary Information:

Test group ID: 21862320
Test group name: Coding Standards - bpel4people
Log ID: 6206346
Start 2010-08-31 17:29:45
Stop date: 2010-08-31 17:30:01

Tool: Jtest
User: devtest
Host: devel76
Platform: Linux

Any [Incomplete] [Fail] [Pass]

Tests Summary:

Total Tests	Incomplete Tests	Failed Tests	Passed Tests	%Coverage
145	0	0	145	-

Tests:

Test ID:	Name	Status	Incomplete Messages	Failed Messages	Passed Messages	%Coverage
970183354	/bpel4people/.settings /org.eclipse.wst.common.project.facet.core.xml	PASSED	0	0	1	-
970183327	/bpel4people/localsettings.properties	PASSED	0	0	1	-
970183402	/bpel4people/pom.xml	PASSED	0	0	1	-
970183360	/bpel4people/src/main/assembly /global_resources.xml	PASSED	0	0	1	-
970183384	/bpel4people/src/main/assembly/icons.xml	PASSED	0	0	1	-
970183369	/bpel4people/src/main/assembly/processes.xml	PASSED	0	0	1	-
970183322	/bpel4people/src/main/assembly /shipping_processes.xml	PASSED	0	0	1	-
970183352	/bpel4people/src/main/java/com/parasoft /bpel4people/AttachmentPropagation.java	PASSED	0	0	1	-
970183376	/bpel4people/src/main/java/com/parasoft /bpel4people/B4PConstants.java	PASSED	0	0	1	-

Policy Report

This report shows compliance with development policy over time.

1. Open the Reports view
2. Choose **Reports> Policy Report**

Development Testing Platform automatically monitors compliance with policies and generates a score. The policies that are checked are customizable and include test results, features being implemented, code size, project schedule, budget, code review process, build monitoring, etc. See “Policy Center (Legacy)”, page 139 for information about configuring project policies.

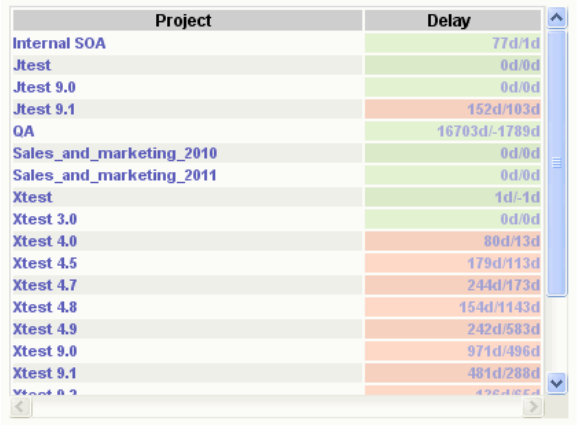
Project Portfolio Report

Project Portfolio provides managers with a quick view of the status of projects that they supervise. Project Portfolio answers the following essential questions for managers:

- Will the project finish on time?
- How effectively and efficiently the team is working,
- How much of the source code is covered with automated tests?

Projects Overview Table

This table lists the projects currently in-progress of which the logged-in manager is in charge. Click on a project to view its detail page. See “Project Details”, page 110. Projects highlighted in red contain problems, such as delays or potential defects, and require further investigation.



Project	Delay
Internal SOA	77d/1d
Jtest	0d/0d
Jtest 9.0	0d/0d
Jtest 9.1	152d/103d
QA	16703d/-1789d
Sales_and_marketing_2010	0d/0d
Sales_and_marketing_2011	0d/0d
Xtest	1d/-1d
Xtest 3.0	0d/0d
Xtest 4.0	80d/13d
Xtest 4.5	179d/113d
Xtest 4.7	244d/173d
Xtest 4.8	154d/1143d
Xtest 4.9	242d/583d
Xtest 9.0	971d/496d
Xtest 9.1	481d/288d

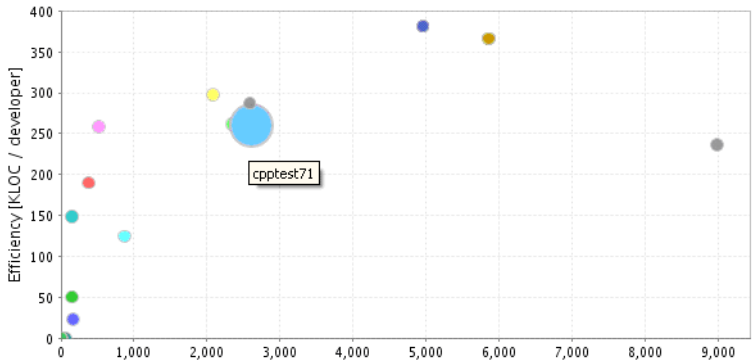
- The delay column shows the Planned Working Time and Deviation from Plan.
- If the Deviation from Plan is a positive number, then the project is already delayed.
- If Deviation from Plan is a negative number, then the project is on time.

Click the Planned Working Time/Deviation from Plan of the appropriate project to open the Iteration Status report; see “Iteration Reports”, page 201.

Efficiency Chart

This chart provides an overview of work efficiency of a project for a specific period. It shows the lines of code in the project against the number of team members working on the project.

[



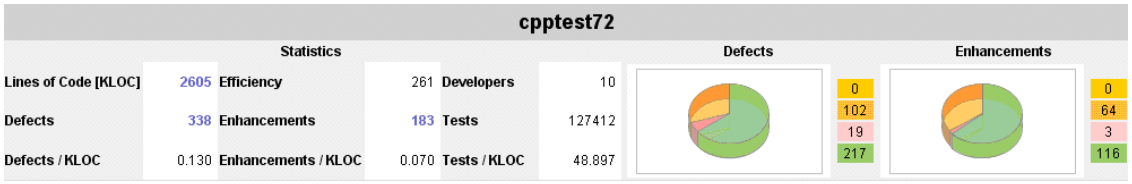
Each dot in the chart represents one project. The higher it is in the chart, the more efficiently team members are working. The size of the dot is related to code coverage: a large dot means the project code coverage is high.

In the above graphic, the C++test 7.1 project is close to ideal because team members are working efficiently (reflected by the location of the dot high on the chart) and they are regularly writing tests to cover the source code (reflected by the large size of the dot).

Project Details

This table shows statistics for each project in the Projects Overview Table. You can perform the following actions:

- Click on the number in the Lines of Code (KLOC) table cell to open the Code Base Size report. See “Code Base Size”, page 111, for additional information.
- Click on the defects chart or the number in the Defects table cell to open the Defects Detailed Report. See “Defects and Enhancements Reports”, page 129, for additional information.
- Click on the enhancements chart or the number in the Enhancements table cell to open the Daily Enhancements report. See “Defects and Enhancements Reports”, page 129, for additional information.

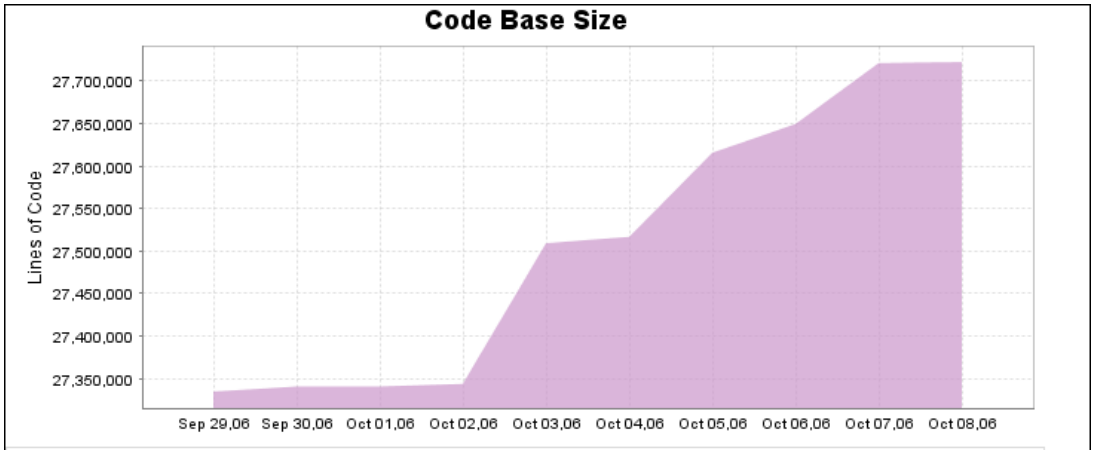


cpptest72									
Statistics						Defects		Enhancements	
Lines of Code [KLOC]	2605	Efficiency	261	Developers	10		0 102 19 217		0 64 3 116
Defects	338	Enhancements	183	Tests	127412				
Defects / KLOC	0.130	Enhancements / KLOC	0.070	Tests / KLOC	48.897				

Code Base Size

You can access the Code Base Size report by clicking on a Code Base Size widget in the Report Center dashboard view. See “Code Widgets”, page 8, for details.

The Code Base Size graph shows the number of lines of code developed in the project. When the line goes up it means that people are working and adding code to the code base. You can look at the Code Base Size graph to see whether code is being checked in on a daily or weekly basis. From this graph, you can also see the amount of code, files, and lines modified each week. You can actually look at each of the modifications and see exactly what was modified in the files.



Date	Lines of Code
Oct 08,06	27722316
Oct 07,06	27721160
Oct 06,06	27649418

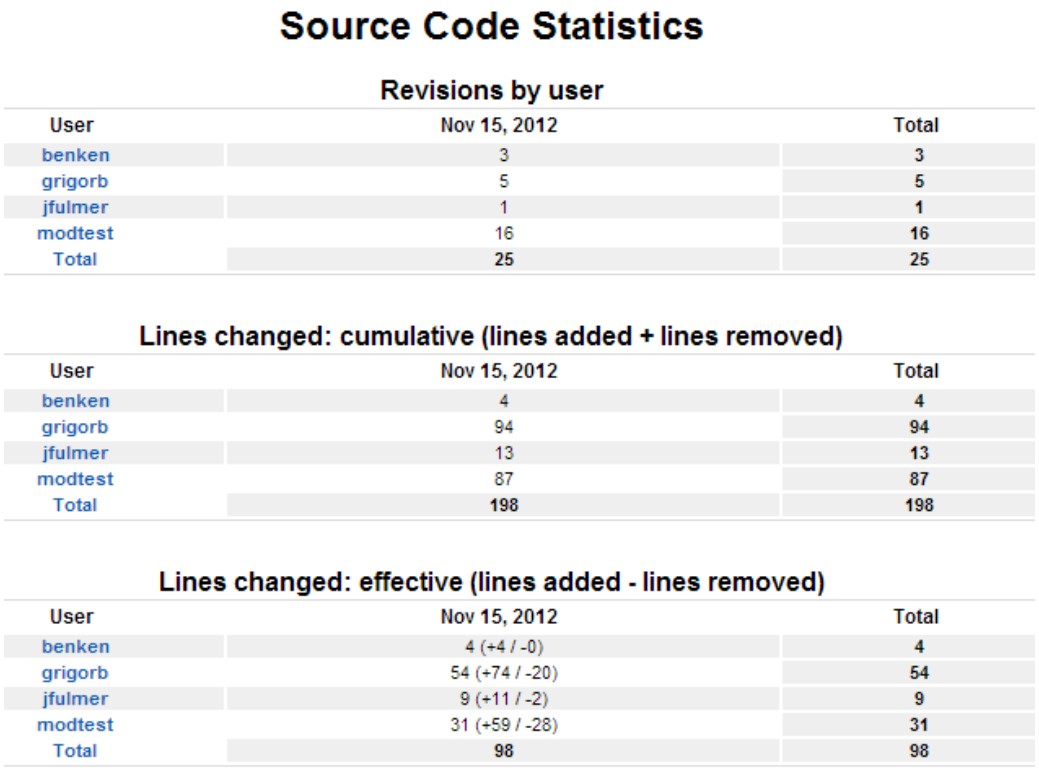
The following information is displayed on the Code Base Size page:

- **Date:** Date on which code was added to source control. Click on a date to open the Source Code Statistics report.
- **Lines of Code:** Total number of lines of code.
- **[View Project Source Code]:** Click to open the Project Source Code report.

Click in the Code Base Size graph to open the “Source Code Check-ins”, page 142 report for details.

Source Code Statistics

Click on a date in the Code Base Size report table to access the Source Code Statistics report.



Source Code Statistics

Revisions by user

User	Nov 15, 2012	Total
benken	3	3
grigorb	5	5
jfulmer	1	1
modtest	16	16
Total	25	25

Lines changed: cumulative (lines added + lines removed)

User	Nov 15, 2012	Total
benken	4	4
grigorb	94	94
jfulmer	13	13
modtest	87	87
Total	198	198

Lines changed: effective (lines added - lines removed)

User	Nov 15, 2012	Total
benken	4 (+4 / -0)	4
grigorb	54 (+74 / -20)	54
jfulmer	9 (+11 / -2)	9
modtest	31 (+59 / -28)	31
Total	98	98

Revisions By User

The Revisions by User table lists all team members who made revisions to any source control files associated with the specified project during the specified date/date range. For each listed team member, it shows how many revisions were made during each listed week as well as the total sum of revisions made to the files for the displayed weeks. f

To view more details about the revisions, click the user name of the appropriate team member. The Source Control Summary page is displayed.

Lines Changed

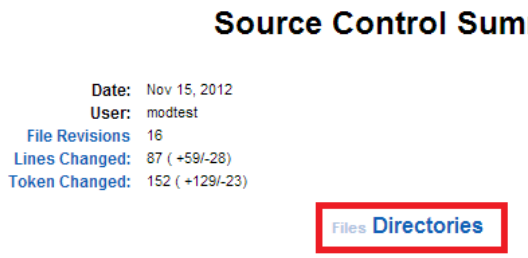
This report lists the team members who made revisions to any source control file associated with the specified project during the specified date/date range. There are two Lines Changed views:

- **Cumulative**: Totals in this view do not adjust for lines removed that have were removed.
- **Effective**: Totals in this view reflect the net lines of code; total lines of code adjusted per lines of code removed.

Click on a team member to open the Source Control Summary page, which provides more details about the line changes.

Source Control Summary

Click on a user in a Source Code Statistics table to access the Source Control Summary page. Information in the Scource Control Summary report can be viewed by file or by directory. Click the **File** or **Directory** to toggle views.



The Source Control Summary contains the following information:

- **Date:** Selected date, which was specified in the Source Code Statistics graph and originally selected from the Code Base Size graph.
- **User:** Name of the user selected in the Source Code Statistics graph.
- **File revisions:** Shows the number of files updated on the selected date by the selected user.
    - **Reviewer Tasks** shows the number of code reviews performed on this file on the selected date (or date range) for the selected user.
    - **Issues** shows the number of code review issues added for this file on the selected date (or date range) for the selected user.
    - **File Revisions Covered by Reviewer Tasks** calculates coverage as follows: (Total # of 'Reviewer Tasks' / Total # of file updates) * 100This report shows three types of source control changes:
- **File Updates:** Shows the files or directories containing the files that have been modified. Click on a file (File view) top open the File Details report. Click on a directory (Directory view) to open the Source Control Summary report.

Files Updates

File	11/15/12	Total
xtest/common/com.parasoft.xtest.execution.api.web/.classpath	1	1
xtest/common/com.parasoft.xtest.execution.api.web/src/com/parasoft/xtest/execution/api/web/runnable/WebFunctionalTestVisualizator.java	1	1
xtest/common/com.parasoft.xtest.results.api.web/.classpath	1	1
xtest/common/com.parasoft.xtest.standards.api.web/.classpath	1	1
xtest/common/com.parasoft.xtest.ws.web/src/com/parasoft/xtest/ws/web/workspace/WorkspaceUtil.java	1	1
xtest/eclipse/com.parasoft.xtest.changeadvisor.eclipse.ui.web/.classpath	1	1
xtest/eclipse/com.parasoft.xtest.common.eclipse.ui.web/.classpath	1	1
xtest/eclipse/com.parasoft.xtest.execution.eclipse.ui.soa/.classpath	1	1
xtest/eclipse/com.parasoft.xtest.results.eclipse.core.soa/.classpath	1	1

- **Lines Changed:** Shows the total number of lines modified on the selected date by the selected user. The positive number in parenthesis represents the number of lines added. The negative number in parenthesis represents the number of lines removed.

Click on a file (File view) top open the File Details report. Click on a directory (Directory view) to open the Source Control Summary report.

Lines changed

Directory	11/15/12	Total
xtest/common/com.parasoft.xtest.execution.api.web/	+1/-0	+1/-0
xtest/common/com.parasoft.xtest.execution.api.web/src/com/parasoft/xtest/execution/api/web/runnable/	+6/-3	+6/-3
xtest/common/com.parasoft.xtest.results.api.web/	+1/-0	+1/-0
xtest/common/com.parasoft.xtest.standards.api.web/	+1/-0	+1/-0
xtest/common/com.parasoft.xtest.ws.web/src/com/parasoft/xtest/ws/web/workspace/	+34/-22	+34/-22
xtest/eclipse/com.parasoft.xtest.changeadvisor.eclipse.ui.web/	+1/-0	+1/-0
xtest/eclipse/com.parasoft.xtest.common.eclipse.ui.web/	+1/-0	+1/-0
xtest/eclipse/com.parasoft.xtest.execution.eclipse.ui.soa/	+1/-0	+1/-0
xtest/eclipse/com.parasoft.xtest.results.eclipse.core.soa/	+1/-0	+1/-0
xtest/eclipse/com.parasoft.xtest.results.eclipse.core.virt/	+1/-0	+1/-0
xtest/eclipse/com.parasoft.xtest.standards.eclipse.ui.web/	+1/-0	+1/-0
xtest/test/com.parasoft.xtest.common.eclipse.ui.web.tests/	+1/-0	+1/-0
xtest/test/com.parasoft.xtest.junit.eclipse.core.web/	+1/-0	+1/-0
xtest/test/com.parasoft.xtest.standards.eclipse.ui.web.tests/	+1/-0	+1/-0
xtest/test/com.parasoft.xtest.testcases.eclipse.ui.web.tests/	+1/-0	+1/-0
xtest/test/com.parasoft.xtest.testcases.eclipse.ui.web.tests/src/com/parasoft/xtest/testcases/web/	+6/-3	+6/-3
Total	+59/-28	+59/-28

- **Tokens Changed:** Shows the total number of tokens modified on the selected date by the selected user. The positive number in parenthesis represents the number of tokens added. The negative number in parenthesis represents the number of tokens removed.

  Click on a file (File view) top open the File Details report. Click on a directory (Directory view) to open the Source Control Summary report..

Tokens Changed

Directory	11/15/12	Total
xtest/common/com.parasoft.xtest.execution.api.web/	+0/-0	+0/-0
xtest/common/com.parasoft.xtest.execution.api.web/src/com/parasoft/xtest/execution/api/web/runnable/	+2/-0	+2/-0
xtest/common/com.parasoft.xtest.results.api.web/	+0/-0	+0/-0
xtest/common/com.parasoft.xtest.standards.api.web/	+0/-0	+0/-0
xtest/common/com.parasoft.xtest.ws.web/src/com/parasoft/xtest/ws/web/workspace/	+124/-22	+124/-22
xtest/eclipse/com.parasoft.xtest.changeadvisor.eclipse.ui.web/	+0/-0	+0/-0
xtest/eclipse/com.parasoft.xtest.common.eclipse.ui.web/	+0/-0	+0/-0
xtest/eclipse/com.parasoft.xtest.execution.eclipse.ui.soa/	+0/-0	+0/-0
xtest/eclipse/com.parasoft.xtest.results.eclipse.core.soa/	+0/-0	+0/-0
xtest/eclipse/com.parasoft.xtest.results.eclipse.core.virt/	+0/-0	+0/-0
xtest/eclipse/com.parasoft.xtest.standards.eclipse.ui.web/	+0/-0	+0/-0
xtest/test/com.parasoft.xtest.common.eclipse.ui.web.tests/	+0/-0	+0/-0
xtest/test/com.parasoft.xtest.junit.eclipse.core.web/	+0/-0	+0/-0
xtest/test/com.parasoft.xtest.standards.eclipse.ui.web.tests/	+0/-0	+0/-0
xtest/test/com.parasoft.xtest.testcases.eclipse.ui.web.tests/	+0/-0	+0/-0
xtest/test/com.parasoft.xtest.testcases.eclipse.ui.web.tests/src/com/parasoft/xtest/testcases/web/	+3/-1	+3/-1
Total	+129/-23	+129/-23

About Programming Tokens

A (programming) token is the basic component of source code. Characters are categorized as one of five classes of tokens that describe their functions (constants, identifiers, operators, reserved words, and separators) in accordance with the rules of the programming language. Consider the following line in the C programming language:

```
sum=3+2;
```

The line can be tokenized as specified in the following table:

Token	Token Type
sum	indent
=	assign_op
3	number
+	add_op
2	number
;	semicolon

Source Control Summary lists only project files that were edited on the specified date. For a list of all project files see “Project Source Code”, page 116.

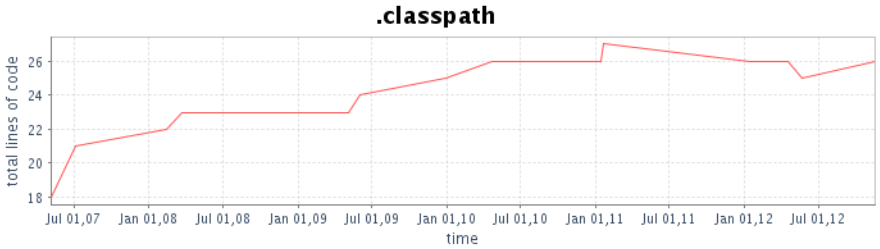
File Details

The File Details table shows the history of the selected file, including the history of all revisions made to that file beginning from the date it was created. The following information displays for each revision made to the selected file:

- **File:** Name of the file. Click to view source code.
- **Date:** Date and time on which the revision was made.
- **Version:** Version number of which the revision is part.
- **User:** User responsible for the code in the file.
- **Lines changed:** Number of lines changed in this revision.
- **Tokens changed:** Number of tokens changed in this revision.
- **Comment:** Comments relevant to the listed revision.

File Details

File: xtest/common/com.parasoft.xtest.execution.api.web/.classpath



Date	Version	User	Lines changed	Tokens Changed	Comment
11/15/12 4:43 AM	1.20	modtest	+1/-0	+0/-0	Task 20167: Module - Results
5/22/12 5:07 PM	1.19	jakubiak	+1/-2	+0/-0	@task 41135 - Enable dev mode on Monrovia projects.
4/17/12 5:25 PM	1.18	benken	+1/-1	+0/-0	@task 42996 - compile with Java 6
1/12/12 2:41 AM	1.17	dario	+0/-1	+0/-0	Removed unused reference to com.parasoft.xtest.sourcecontrol project
1/19/11 5:35 PM	1.16	truong	+1/-0	+0/-0	@task 34009 - Add dependency on com.parasoft.xtest.filedefect
1/12/11 9:52 AM	1.15	jakubiak	+2/-2	+0/-0	Add back correct path to source folders
1/11/11 8:21 AM	1.14	maciekg	+2/-2	+0/-0	Fix illegal link to source folder.
4/21/10 4:51 AM	1.13	marzec	+1/-0	+0/-0	task 12240: Fix Enh. 85697 : Put Source Control revision information in reports.
12/30/09 12:21 PM	1.12	zohrabb	+1/-0	+0/-0	PR 87340 @task 6555 - The "View Rule Documentation" option is broken for the Security "rules"
6/1/09 4:53 PM	1.11	jakubiak	+1/-0	+0/-0	PR 83703 - @task 3745 - Use WebInsureFunctionalTestViolation instead of InsureViolation in SOAtest
5/5/09 6:51 AM	1.10	maco	+1/-1	+0/-0	@task 11142 @PR 83318 : add libs.base project
4/22/09 10:44 AM	1.9	benken	+1/-1	+0/-0	@task 2519 - use J2SE-1.5 execution environment
4/21/09 5:03 PM	1.8	benken	+1/-1	+0/-0	@task 2519 - build Monrovia code with Java 1.5
3/13/09 7:05 PM	1.7	benken	+1/-1	+0/-0	PR 80047 - @task 2542 - use stax in xtest.libs
8/25/08 10:36 AM	1.6	benken	+3/-3	+0/-0	PR 76630 - rename webking folder to root
3/20/08 4:28 PM	1.5	jliu	+2/-1	+0/-0	add in direct dependency on stax-api.jar
2/12/08 11:53 AM	1.4	jeehongm	+3/-2	+0/-0	PR 72156 - Dependence on results.api.web
7/3/07 4:45 PM	1.3	jakubiak	+3/-0	+0/-0	Added functional test violation type.
5/4/07 7:36 AM	1.2	marzec	+1/-1	+0/-0	Refactored.
5/4/07 7:25 AM	1.1	marzec	+18/-0	+0/-0	Added.

Project Source Code

Click the **[View Project Source Code]** link in the Code Base Size report to access the Project Source Code page. The Project Source Code page lists the main directories of the selected project. Click a directory to view subdirectories and files. The Project Source Code page lists all project files. By contrast, the Source Control Summary page lists only project files that were edited on the specified date.

Project Source Code

Directory of:
Directories: 39
Files: 0

Name ▲	Type
	[directory]
alex	[directory]
Attic	[directory]
bpelmaestro	[directory]
classes	[directory]
com	[directory]
config	[directory]
cpp	[directory]
dbc	[directory]
devadmin	[directory]
dtracer	[directory]

The following information is listed on the Project Source Code page:

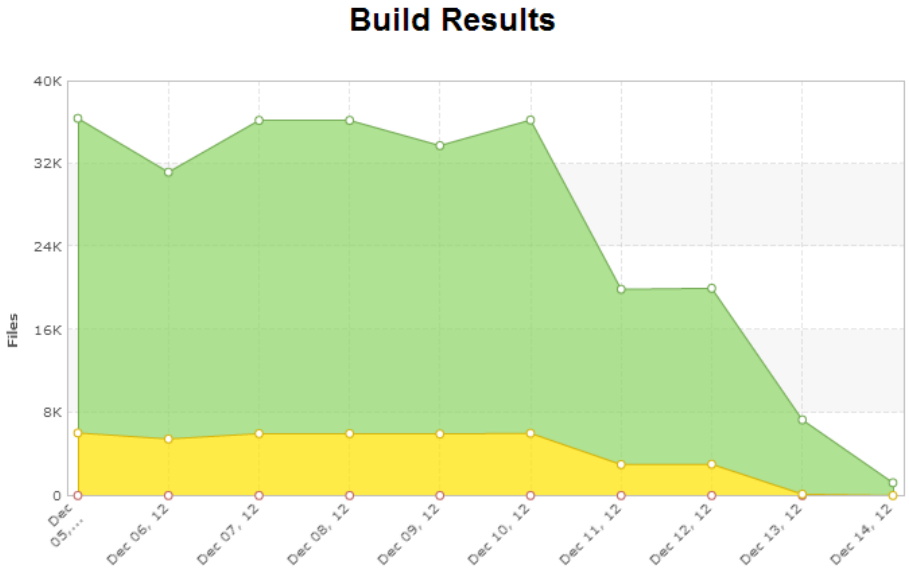
- **Directory of:** Displays where you are within the source repository tree. The example shown in displays a blank field because the root directory contents are listed and a directory has not been selected, yet.
- **Directories:** Shows the number of directories included in the selected directory.
- **Files:** Shows the number of files located in the selected directory.
- **Name:** Lists the names of the directories/files located in the selected directory. Click to drill down to the next level of the directory until you reach the appropriate file. When you click the filename, the File Details report is displayed. See “File Details”, page 115 for details.
- **Type:** Shows whether the listed name is a directory or, if it is a file, it lists the format extension of the file.

Build Results

From the Reports view, choose **Practices> Builds** to access the Build Results report. For details about the Build Results page, see “Builds”, page 56.

The Build Results graph shows the following build information for each listed drop date:

- Number of files that failed
- Number of files that contain warnings and are incomplete
- Number of files and modules that passed



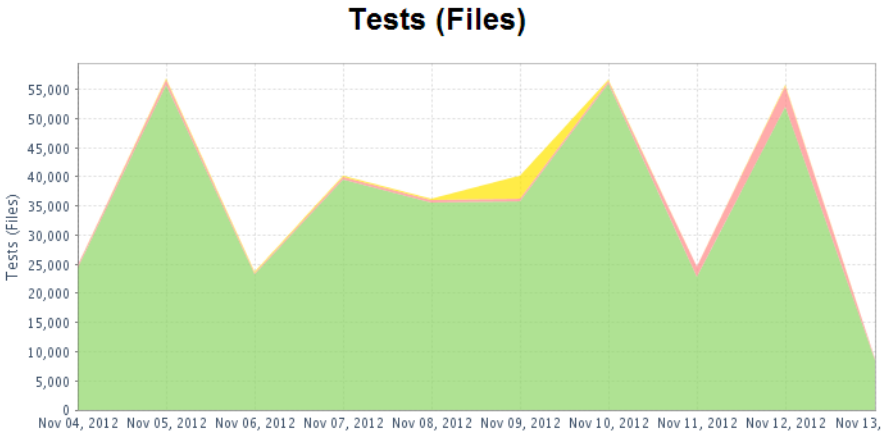
Date	Failed	Warning	Passed
Dec 14, 2012	0	21	1234
Dec 13, 2012	0	138	7174
Dec 12, 2012	0	3037	16938
Dec 11, 2012	3	2980	16909
Dec 10, 2012	1	6002	30198
Dec 09, 2012	2	5938	27784
Dec 08, 2012	2	5980	30176
Dec 07, 2012	2	5980	30184
Dec 06, 2012	4	5448	25709
Dec 05, 2012	3	6027	30320

Tests (Files)

The **Tests (Files)** report provides an overview of your project's current state. It lists tests that ran during the specified time period.

1. From the Reports view, choose **Tests> Tests Overview**
2. Click **Tests (Files)**

Each row shows the number of incomplete, failed, and passed test cases for all tests that were run on a specific date, including coding standards analysis, unit testing, regression testing, and so on—with the exception of manual testing. Results of manual tests are shown in the Manual Testing Efforts graph. See "Manual Test Sessions", page 135, for details about manual tests.)



	Date	Number of Tests (Files)				Duration	Execution Time Diff
		Incomplete	Fail	Pass	Total		
+	Nov 13, 2012	0	252	8410	8662	3h 47m 43s	-3d 10h 11m 2s
+	Nov 12, 2012	237	3501	51968	55706	3d 13h 58m 45s	7h 45m 33s
+	Nov 11, 2012	0	1718	22763	24481	3d 6h 13m 11s	1h 27s
+	Nov 10, 2012	238	509	55827	56574	3d 5h 12m 43s	2d 13h 16m 48s
+	Nov 09, 2012	3936	507	35596	40039	15h 55m 55s	-1d 3h 14m 32s
+	Nov 08, 2012	237	408	35520	36165	1d 19h 10m 28s	3h 19m 17s
+	Nov 07, 2012	237	484	39424	40145	1d 15h 51m 11s	3h 46m 25s
+	Nov 06, 2012	229	122	23190	23541	1d 12h 4m 45s	-1d 8h 12m 57s

Click on a date to view static analysis and unit tests:

- For static analysis, the number of files tested are shown.
- For unit testing, test (suite) files are shown.

Additionally, the Tests (Files) report enables you to quantitatively and qualitatively compare tests. You can also check how many tests have been added to a selected test group, as well as see view the number of tests that changed from Failed status to Passed status and vice versa. Dor example, if the report shows that previously passing tests are now failing, you can investigate the cause and solve any problems. Use the drill-downs to help uncover the source of the problem.

Drilling down enables you to compare two separate test runs for your project from specified dates. You can compare tests at different levels—from general run results to single test message results. Report Center enables enables test analysis by comparing data from the following processes:

- Test runs by trio (Tool, Machine, Analysis type). See "Tests (Files) By Type", page 120.
- Test group runs within single trio. See "Tests By Type Details", page 122.

- Test runs within single test group. See “Test Group Details”, page 123.

The following entity factors are compared:

- Total number of executed test runs, test groups, or tests.
- Number of incomplete/passed/failed within total number of executed test runs, test groups, or tests.
- Cumulative time execution of test run, test group, or test.

For each executed test listed, the Tests report shows the following information:

- **Manage Baselines ( ):** Enables you to mark results from the selected date as baseline results. Base line results indicate tested source code that is in good condition and serve as a reference against which to draw comparisons. (See “Setting a Baseline”, page 126 for details.)
- **Date:** Date on which tests were run.
- **Incomplete:** Number of test cases that were incomplete due to an unexpected error that occurred during the test run.
- **Fail:** Number of test cases that failed the test run.
- **Pass:** Number of test cases that passed the test run.
- **Total:** Total number of test cases (except for manual tests) that were run on the selected date.
- **Execution Time:** Cumulative amount of time that it took for all test cases to run on the selected date.
- **Execution Time Difference:** Time difference between the previous test case run and the current test case run.

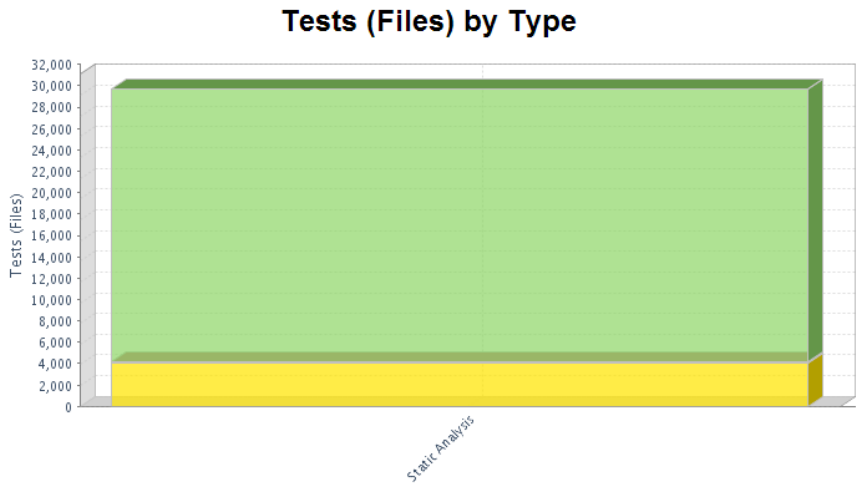
Tests (Files) Drill-down Reports

There are several statistics available to view by drilling down from the Tests (Files) report. Click any of the available links listed beneath the Date, Incomplete, Fail, and Pass columns to open the Tests (Files) By Type report.

Tests (Files) By Type

Open the Test (Files) report and click on a date or a value in the Incomplete, Fail, or Pass column to open the Tests (Files) By Type report.

The Tests (Files) By Type graph shows test result comparisons by way of the unique trio (test type, tool, and machine) on which tests were run. Each row shows the number of tests per trio that were incomplete, failed, or passed for the date on which you clicked in the Tests table. The table presents both quantitative and qualitative changes in test runs.



Test type	Machine	Tool	Number of Tests (Files)												Execution time		
			Incomplete			Fail			Pass			Total					
			Base	Curr.	New	Base	Curr.	New	Base	Curr.	New	Base	Curr.	Diff.	Base	Curr.	Diff.
Static Analysis	devel199	Jtest		229	229		47	47		15346	15346		15622			59m 40s	
Static Analysis	anteater	Jtest	2	8	8	6	25	20	2716	6989	4264	2724	7022	4298	10m 11s	27m 12s	17m
Static Analysis	meadowlark	Jtest	3988	3995	8	9	9	5	3850	3917	74	7847	7921	74	7m 7s	7m 32s	25s
Total			3990	4232	245	15	81	72	6566	26252	19684	10571	30565	19994	17m 18s	1h 34m 25s	1h 17m 6s

Following is additional information available on the Tests (Files) By Type page:

- **Base:** Test results for the date against which you want to compare current test results.
- **Curr.:** Current (most recent) test results.
- **New:** Either test results that were present in the current test run, but were not present in the comparison run (Base), or test results that had a status change.
- **Diff:** Difference between Base and Curr. results.

Qualitative Changes (Example)

To explain qualitative changes, let's work with an example. The Tests (Files) By Type report shows test activity. The numbers in the table are differentiated by different fonts to help you spot anything that might be wrong or should be improved:

- **Bold:** Reflects current data and indicates that the analysis was run on that very date. Data in bold links to drill-down reports.
- *Gray, italic:* Indicates that analysis was not run for the current date—leaving an inability for the system to compare data and only showing analysis data for the date against which a comparison was to be made.
- *Red, italic:* Indicates that the analysis uncovered discrepancies between the current data in comparison to the base data. In other words, there is an unwelcome situation that should be examined thoroughly.

The report enables you to see improvements to the source code, as well as how your changes affected tests and functionality of the application.

Test type	Machine	Tool	Incomplete			Fail			Pass			Total			Execution time		
			Base	Curr.	New	Base	Curr.	New	Base	Curr.	New	Base	Curr.	Diff.	Base	Curr.	Diff.
Functional Testing	thorn/10.9.1.193	JUnit	0	**0**	0	8	**6**	3	39	**41**	5	47	**47**	0	6h 2m 6s	**4h 50m 44s**	-1h 11m 22s
Static Analysis	gobos	Jtest	0	**0**	0	144	**138**	1	1182	**1188**	7	1326	**1326**	0	2h 55m 16s	**2h 39m 50s**	-15m 26s
Unit Testing	gobos	Jtest	0	**0**	0	25	**23**	1	736	**737**	4	761	**760**	-1	2h 36m 54s	**2h 6m 54s**	-29m 59s
Functional Testing	zuza/10.9.1.171	SOAtest	0	**0**	0	0	**0**	0	133	**133**	0	133	**133**	0	56s	**1m 6s**	9s
Total			0	**0**	0	177	**167**	5	2090	**2099**	16	2267	**2266**	-1	11h 35m 14s	**9h 38m 35s**	-1h 56m 38s

Take a look at the first trio listed in the graphic above—JUnit executed Functional Testing on the Thorn machine. As reflected beneath the **Total** column, you can see that the total number of current test runs (**Curr.** column) did not change from the set baseline (**Base** column). Previously, there were 47 test runs and the current day shows the same number. The exact comparison dates are displayed in the upper-right corner of the Tests (Files) By Type page (not shown).

Now, take a look at the Fail and Pass columns. You can see that there are some differences in the number of tests that passed and failed. This means that some tests' statuses changed—either due to application source code changes or test scenario variations.

Let's consider failed tests. There were 8 failed tests previously (**Fail / Base** column); today there are 6 failed tests (**Fail / Curr.** column) but 3 of them are new (**Fail / New** column. New failed tests are tests that were previously passed, incomplete or were not run; but changed to failed.

Walking though the math:

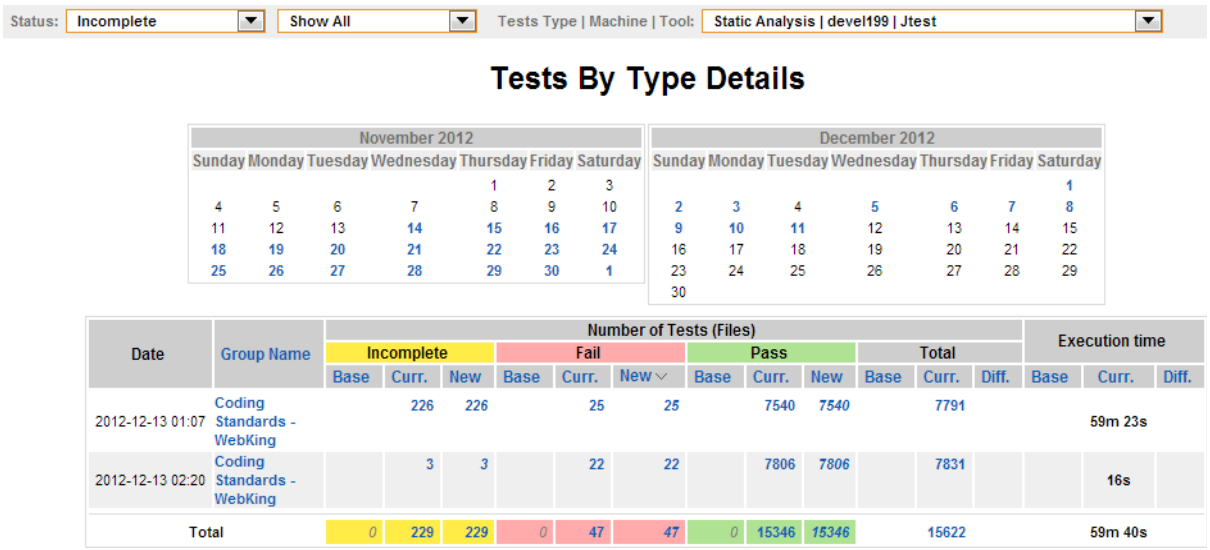
- 6-3= 3: Indicates that only 3 previously failed test cases are still failed.
- 8-3= 5: Indicates that 5 of the previously failed test cases changed to passed and explains why there are 5 new tests in the **Pass** section—most likely, some erroneous code was fixed so that those test cases could pass.
- Previously, 39 test cases passed. From the calculation above, we have 5 new test cases that passed; that is 39+5= 44. However, we can see that only 41 test cases passed.
- 44-41= 3: Indicates that 3 of the previously passed test cases now failed. That is 3 new fails.

You can click any of the available links within the Tests (Files) By Type table to drill down to the Tests By Type Details report.

Tests By Type Details

Open the Tests (Files) By Type report and click on any link to open the Tests By Type Details report.

The data displayed in the Tests (Files) By Type Details table reflects details of the trio (test type, tool and machine) selected in the Tests By Type table. Essentially, the Tests By Type Details table compares test group runs within the selected trio.



Following is additional information available on the Tests By Type Details page:

- **Total (row):** Displays summarized data for all test groups existing within the selected trio. There are some drill-downs available from this row. See below for details.

  Note: *If necessary, you can filter the information displayed in the Tests By Type Details table directly on this page. In other words, there is no need to return to the Test by Type report to select a new trio.*

- **Group Name:** Name assigned to the displayed group of tests.

Following are the data filters available on the Tests By Type Details page:

- **Status:** Shows only test results with the selected status.
- **New Status:** Shows only test results with a new status. (A new status is derived when the analysis uncovers discrepancies between the current data and the base data.)
- **Tests Type/Machine/Tool:** Shows test results for the selected trio on the selected date.
- **Base Date/Curr. Date:** Compares test results between the selected dates.

  Aside from the Calendar icon located to the right of each date field, you can also use the calendar(s) displayed in the middle of the Tests By Type Details page to select the Base Date. Dates that have data available for comparison are underlined in the calendar(s). Drops for up to two months are available. You can click any underlined date to compare it against the current date's data, and then display the results. A grayed out date indicates the baseline date. (See "Setting a Baseline", page 126).

Click one of the following to open the Test Group Details report (Figure):

- Number of failed test cases for a specific test group
- Test group name (listed beneath the **Group Name** column)
- Total number of failed new tests (listed in the **Total** row)

Test Group Details

Open the Tests By Type Details report and click any link the table to open the Test Group Details report.

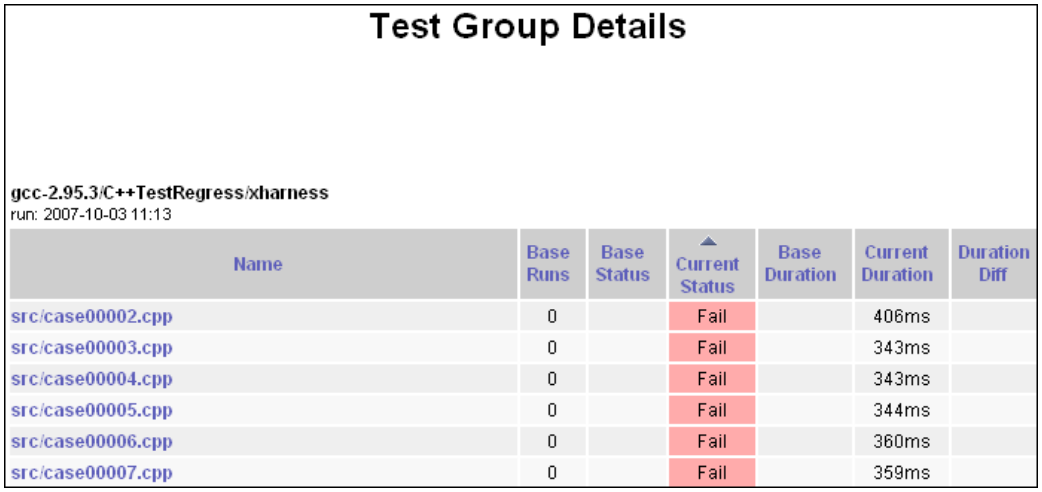
The Test Group Details report displays details for the selected test group, and filters the list of test groups based on the selected result and change status.

Comparing results from different dates helps uncover code regressions. A change in base/current status is a good indication that code introduced between the old and new results caused incorrect behavior. Similarly, a drastic increase in duration could indicate a performance problem (or optimization, if the duration decreases).

The data displayed in the Test Group Details report is sorted based on how you access the report:

If accessed by clicking...	This report lists...
The number of incomplete/failed/passed test cases for a specific test group on the Tests By Type Details report.	All of the test cases run for that particular test group and with the selected status.
A specific test Group Name on the Tests By Type Details report.	All of the test cases run for that test group with any status.
The *total* number of incomplete/failed/passed tests (listed in the Total row).	All test groups that were run for the selected trio, along with each test case run for each test group. It provides a comprehensive view of the condition of the project per test Group Name.

For all results except for SOAtest Results, test group details are presented as follows:



Test Group Details

gcc-2.95.3/C++TestRegress/xharness
run: 2007-10-03 11:13

Name	Base Runs	Base Status	Current Status	Base Duration	Current Duration	Duration Diff
src/case00002.cpp	0		Fail		406ms	
src/case00003.cpp	0		Fail		343ms	
src/case00004.cpp	0		Fail		343ms	
src/case00005.cpp	0		Fail		344ms	
src/case00006.cpp	0		Fail		360ms	
src/case00007.cpp	0		Fail		359ms	

This report shows the following information:

Item	Description
Name	Name assigned to the listed test
Base Runs	Number of times the set baseline test was run
Base Status	Status of the current test—incomplete, failed, or passed
Current Status	Status of the current test—incomplete, failed, or passed

Item	Description
Base Duration	Amount of time it took for the set baseline test to complete
Current Duration	Amount of time it took for the current test to complete
Duration Diff	Difference between the amount of time it took for the set baseline test to complete and the amount of time it took for the current test to complete

Color indicators are used to mark files as follows:

Color	Indicates
Green	The problem is fixed - it used to occur (on the base date) but did not occur in the date for which report is shown
Red	The problem is new - it did not occur (on the base date) but did occur in the date for which report is shown.

For these results, you can:

- Check what problem occurred where by clicking the test group name to open the Test Details report. Here, you can see the source code by clicking on the line number link (if SourceScanner runs on your project).
- Filter results by status:
  - **Any** will display all tests.
  - **Any Error** will display tests with a current status of "Fail" or "Incomplete".
  - **Fail, Incomplete, and Pass** will display the tests with the selected status.
- Filter results to show all tests or only tests whose results changed (e.g., from "Pass" to "Fail" or from "Incomplete" to "Pass."
- Combine the two filtering options to display specific types of results—for instance. "Pass / Show Changed" will show only tests whose status changed from "Fail" to "Pass".

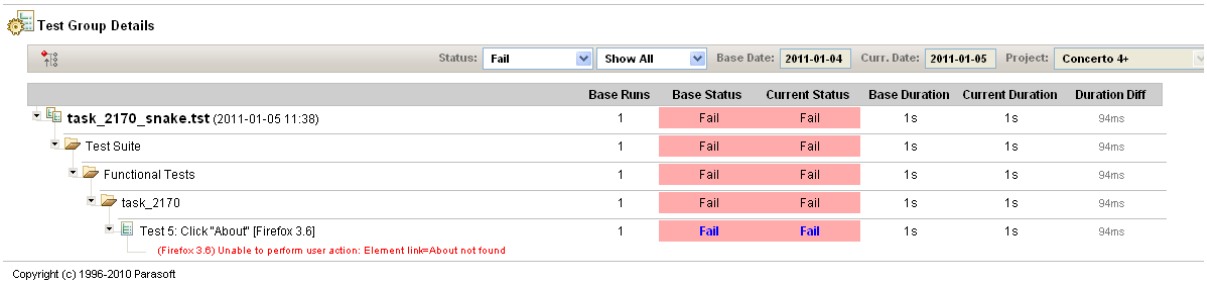
SOAtest Results

Open the Tests By Type Details report and click on any available link to SOAtest results in the table. SOAtest results are displayed somewhat differently than other results:

- Tests are grouped by test suite. Tests are arranged in a tree as shown below. The nodes are expandable and collapsible.
- Test suite nodes contain aggregate information of all the child tests. If test suite information is not available, all tests are children of the root node.
- Test status for a suite is "Fail" if one or more tests fail; otherwise it is "Pass".
- Durations are the sum of all child tests.

For these SOAtest results, you can:

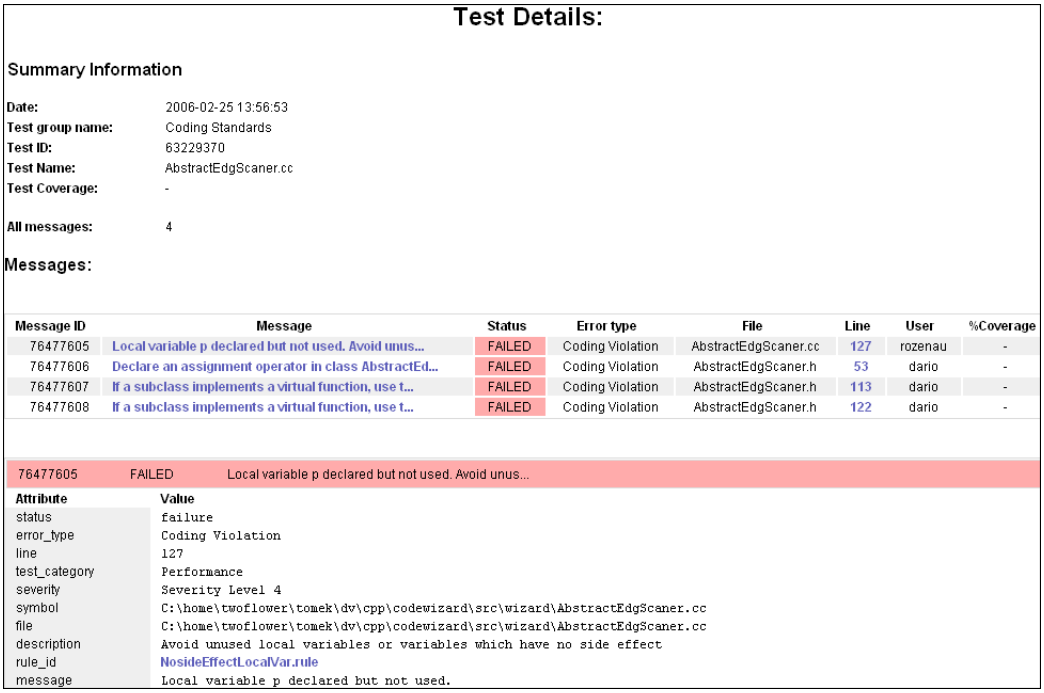
- Click the arrows on the tree to expand or collapse the nodes.
- View either the previous or current test details by clicking the base and current status for individual tests.



Test Details

Open the Tests By Type Details report and click on a name to open the Test Details report.

The Test Details report provides information about the selected test, along with a detailed list of all messages sent in the log.

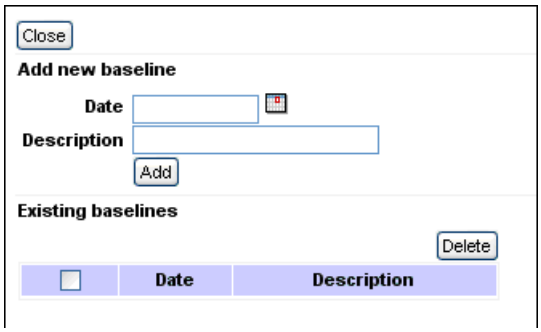


Setting a Baseline

The baseline option lets you mark a specific report (test results on selected date) as a reference for tested source code in good condition. The purpose is to have a point of reference against which you can compare future results and see the project development tendency.

You can access the Baseline pop-up window in the following ways:

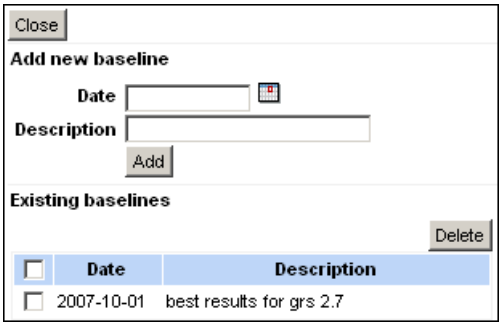
- From the Tests report, either click the baseline icon located at the top of the first column of the table, or one of the plus symbols listed beneath it (select the one that corresponds with the appropriate date).
- From any Tests report drill-down page (Tests By Type, Tests By Type Details; or Test Group Details), click the baseline icon located in the upper-right corner.



To set a baseline:

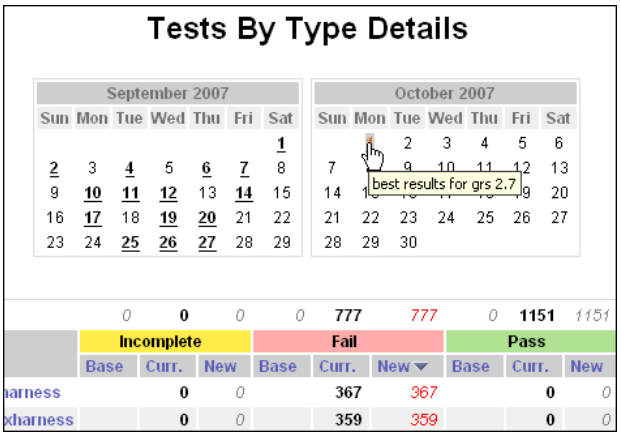
1. In the **Date** field of the Baseline pop-up window, type or select a date to set as your base for comparison against future test results.
2. In the **Description** field, type an appropriate description to identify the baseline, such as "best results for Report Center 2.7", and then click **Add**.

   The new baseline is listed beneath **Existing baselines**:



3. Click the check box next to the new baseline to activate it, and then click **Close**.
4. Open or refresh the Tests By Type Details page.

   The baseline date is displayed in gray:



5. Click the baseline date.

   The report displays data from the most recent test run, along with the baseline data so that you can compare the results.

To deactivate a baseline:

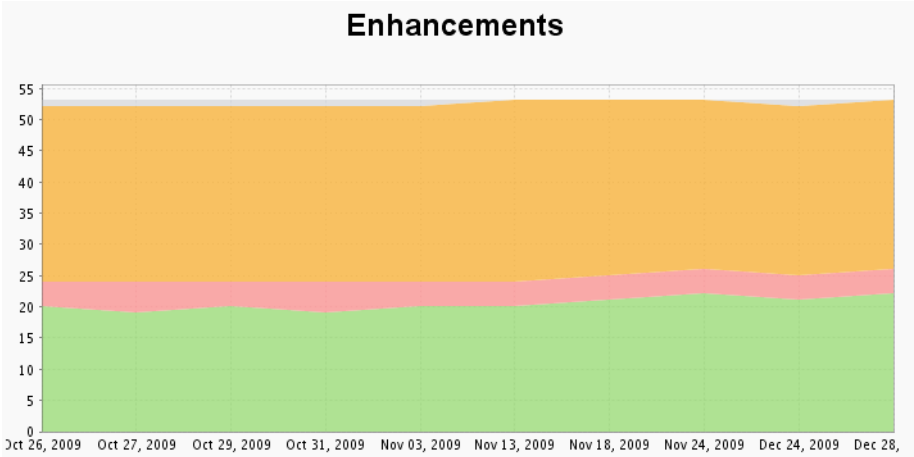
- In the Baseline pop-up window, clear the check box next to the baseline date, and then click **Close**.

To delete a baseline:

- In the Baseline pop-up window, verify that the appropriate baseline date is selected, and then click **Delete**. Next, click **Close**.

Defects and Enhancements Reports

The Defects/Enhancements report shows the trend of your project's defects/enhancements for the given status (Unresolved, Not Tested, Test Failed, Fixed) in a specified period of time. This helps you monitor work on your project's defects/enhancements.



Date	Unresolved	Not Tested	Test Failed	Fixed	Total
Dec 28, 2009	0	27	4	22	53
Dec 24, 2009	1	27	4	21	53
Nov 24, 2009	0	27	4	22	53
Nov 18, 2009	0	28	4	21	53
Nov 13, 2009	0	29	4	20	53
Nov 03, 2009	1	28	4	20	53
Oct 31, 2009	1	28	5	19	53
Oct 29, 2009	1	28	4	20	53
Oct 27, 2009	1	28	5	19	53
Oct 26, 2009	1	28	4	20	53

Understanding the Defects and Enhancements Reports

These reports provide key information about project status. They give you a snapshot of the project quality and show you where and how the project is progressing.

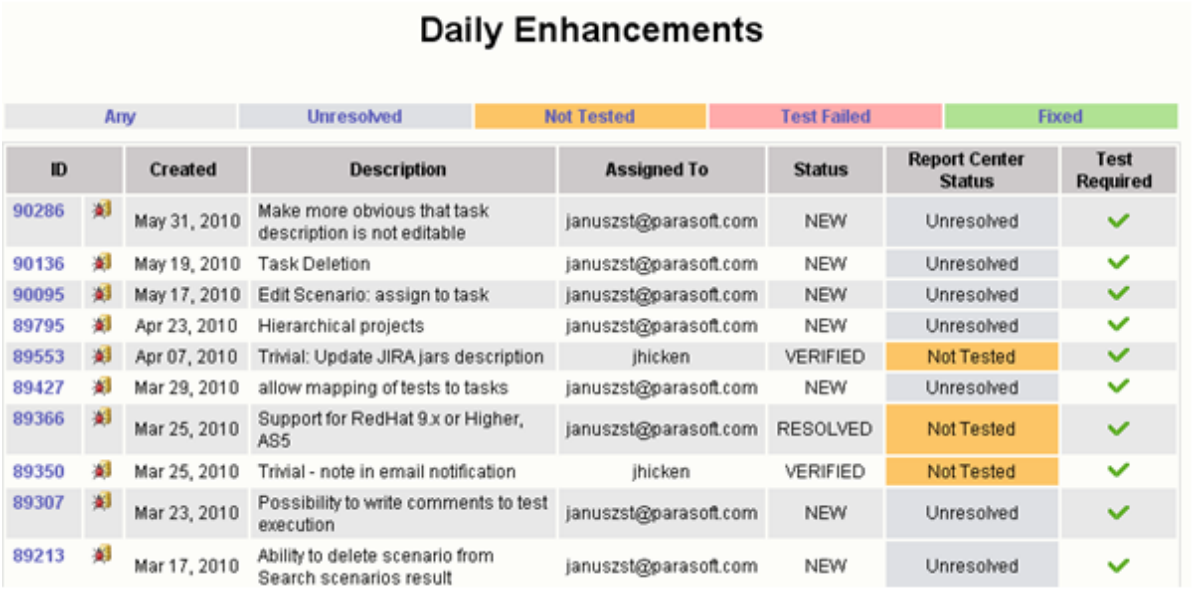
For example, when you view your project defects/enhancements for last 12 weeks period, you might see that:

- **The total number of defects is growing fast, but the trend of items being resolved is not keeping pace.** This can be a trigger to spend more development time on fixing the defects or to take some measures to prevent the bugs.
- **The number of untested items is growing over a time**. This means that the team is resolving PRs/FRs in the Bug Tracking System, but they are not adding automated or manual tests to ensure the quality of their work.

Drilling Down into Details

You can drill down into each status column to see the list of the project defects/enhancements that belong to the specified category (Unresolved/Not Tested/Test Failed/Fixed).The following pages show

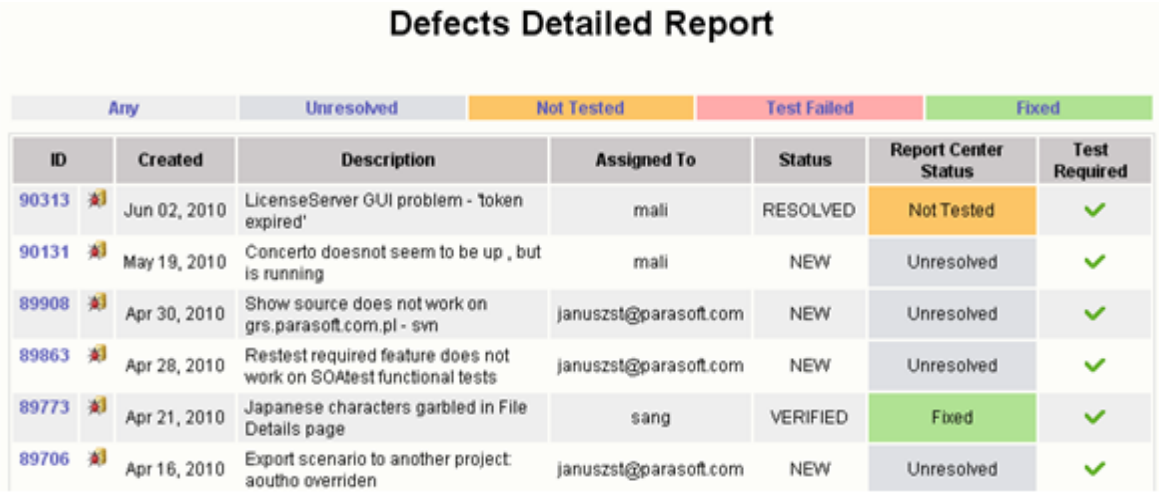
all defects/enhancements reported for selected date and status, along with basic properties like name, creation date, and owner:



Daily Enhancements

Any | Unresolved | Not Tested | Test Failed | Fixed

ID	Created	Description	Assigned To	Status	Report Center Status	Test Required
90286	May 31, 2010	Make more obvious that task description is not editable	januszst@parasoft.com	NEW	Unresolved	✓
90136	May 19, 2010	Task Deletion	januszst@parasoft.com	NEW	Unresolved	✓
90095	May 17, 2010	Edit Scenario: assign to task	januszst@parasoft.com	NEW	Unresolved	✓
89795	Apr 23, 2010	Hierarchical projects	januszst@parasoft.com	NEW	Unresolved	✓
89553	Apr 07, 2010	Trivial: Update JIRA jars description	jhicken	VERIFIED	Not Tested	✓
89427	Mar 29, 2010	allow mapping of tests to tasks	januszst@parasoft.com	NEW	Unresolved	✓
89366	Mar 25, 2010	Support for RedHat 9.x or Higher, AS5	januszst@parasoft.com	RESOLVED	Not Tested	✓
89350	Mar 25, 2010	Trivial - note in email notification	jhicken	VERIFIED	Not Tested	✓
89307	Mar 23, 2010	Possibility to write comments to test execution	januszst@parasoft.com	NEW	Unresolved	✓
89213	Mar 17, 2010	Ability to delete scenario from Search scenarios result	januszst@parasoft.com	NEW	Unresolved	✓



Defects Detailed Report

Any | Unresolved | Not Tested | Test Failed | Fixed

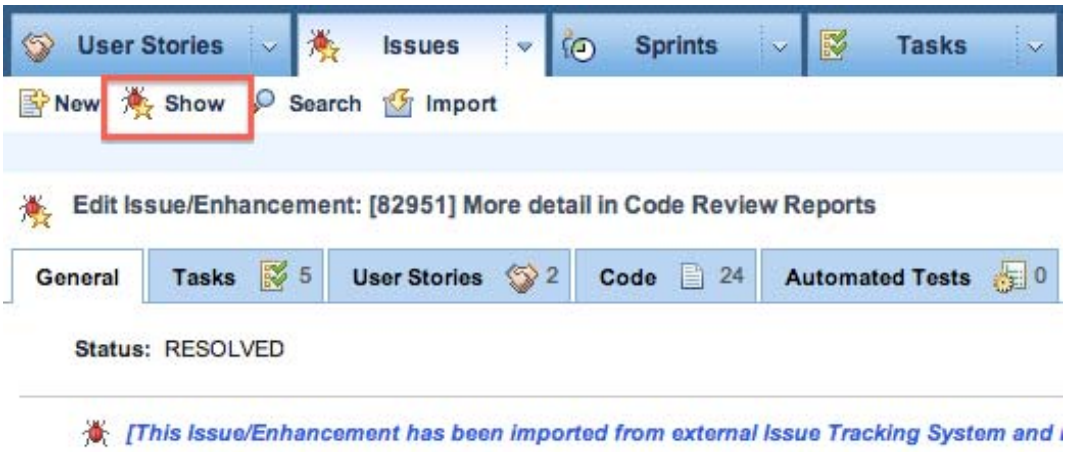
ID	Created	Description	Assigned To	Status	Report Center Status	Test Required
90313	Jun 02, 2010	LicenseServer GUI problem - 'token expired'	mali	RESOLVED	Not Tested	✓
90131	May 19, 2010	Concerto doesnot seem to be up , but is running	mali	NEW	Unresolved	✓
89908	Apr 30, 2010	Show source does not work on grs.parasoft.com.pl - svn	januszst@parasoft.com	NEW	Unresolved	✓
89863	Apr 28, 2010	Restest required feature does not work on SOAtest functional tests	januszst@parasoft.com	NEW	Unresolved	✓
89773	Apr 21, 2010	Japanese characters garbled in File Details page	sang	VERIFIED	Fixed	✓
89706	Apr 16, 2010	Export scenario to another project: aoutho overriden	januszst@parasoft.com	NEW	Unresolved	✓

Exploring Details

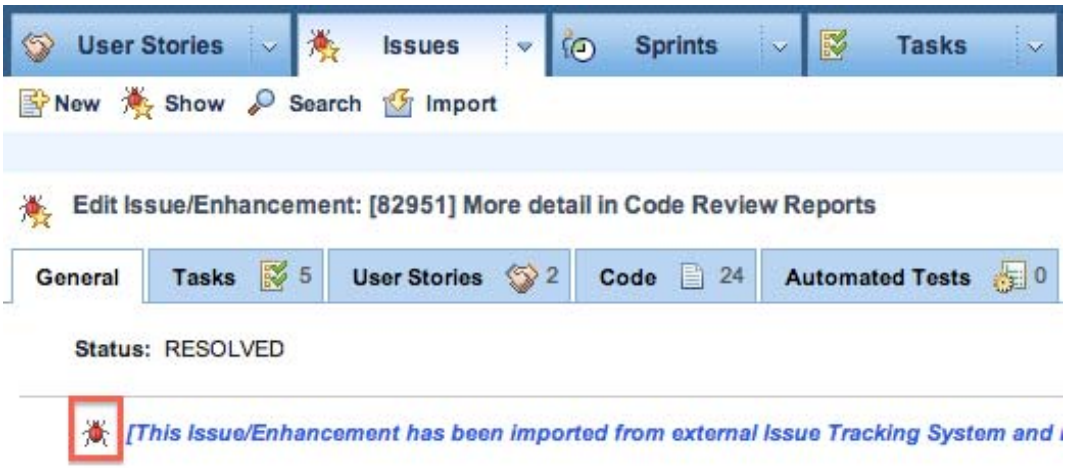
From these pages you can:

- **Open a specific defect/enhancement details page**: Click a defect/enhancement ID number.

- **Open the original Bug Tracking System page:** Click on the Show button. A Bug Tracking System that has a web interface, such as JIRA and Bugzilla, is required.



- **Check the defect/enhancement status from the Bug Tracking System:** Click the bug icon in the General tab**.**

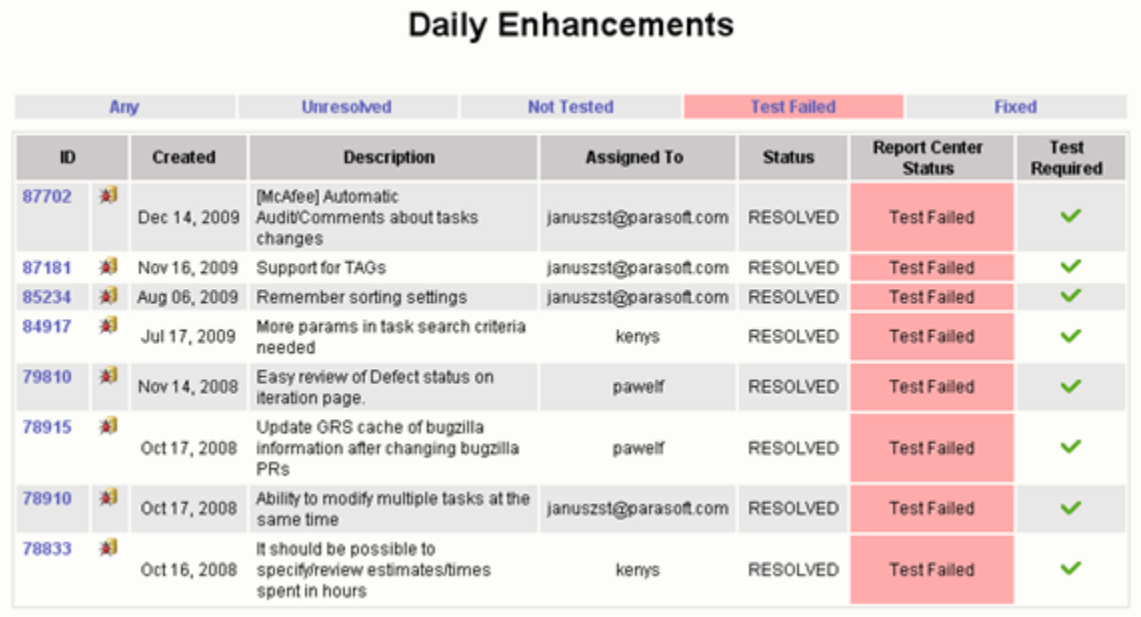


- **See the Report Center status,** which indicates the defect/enhancement's test status as follows:

Status	Description
Unresolved	The defects/enhancements are not yet resolved in the Bug Tracking System.
Not tested	The defects are resolved, but not tested.
Test failed	The defects are resolved and tested. Tests failed.
Fixed	The defects are resolved and tested. Tests passed successfully.

Filtering by Status

Using the Status bar located above the table, you can focus on defects/enhancements with a given status (e.g., defects/enhancements that are resolved, but not yet tested). For example, if you wanted to check all defects/enhancements whose tests failed and see the details of those tests, you would filter this page by clicking the **Test Failed** status link:



Daily Enhancements

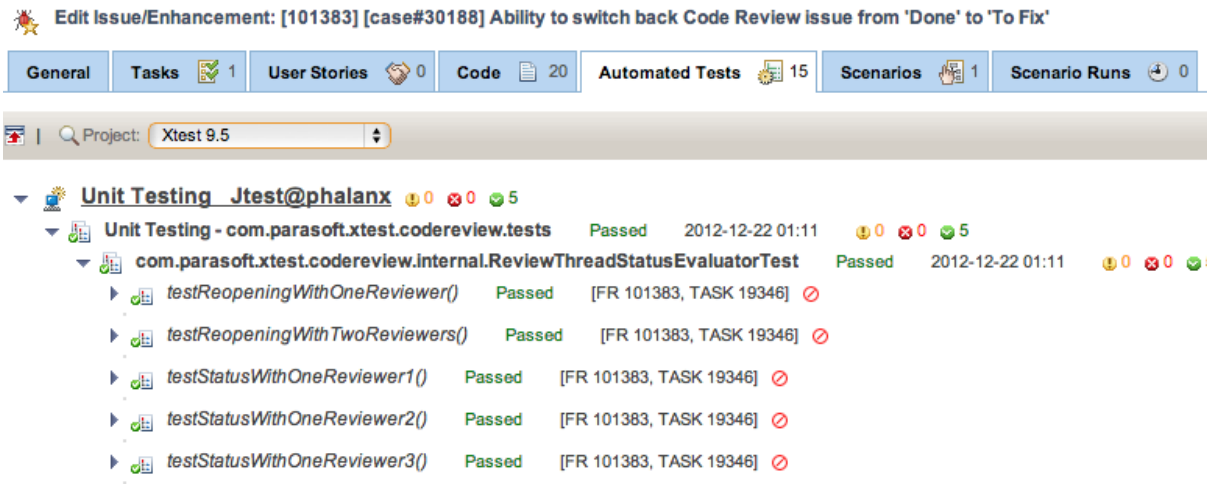
Any | Unresolved | Not Tested | Test Failed | Fixed

ID	Created	Description	Assigned To	Status	Report Center Status	Test Required
87702	Dec 14, 2009	[McAfee] Automatic Audit/Comments about tasks changes	januszst@parasoft.com	RESOLVED	Test Failed	✓
87181	Nov 16, 2009	Support for TAGs	januszst@parasoft.com	RESOLVED	Test Failed	✓
85234	Aug 06, 2009	Remember sorting settings	januszst@parasoft.com	RESOLVED	Test Failed	✓
84917	Jul 17, 2009	More params in task search criteria needed	kenys	RESOLVED	Test Failed	✓
79810	Nov 14, 2008	Easy review of Defect status on iteration page.	pawelf	RESOLVED	Test Failed	✓
78915	Oct 17, 2008	Update GRS cache of bugzilla information after changing bugzilla PRs	pawelf	RESOLVED	Test Failed	✓
78910	Oct 17, 2008	Ability to modify multiple tasks at the same time	januszst@parasoft.com	RESOLVED	Test Failed	✓
78833	Oct 16, 2008	It should be possible to specify/review estimates/times spent in hours	kenys	RESOLVED	Test Failed	✓

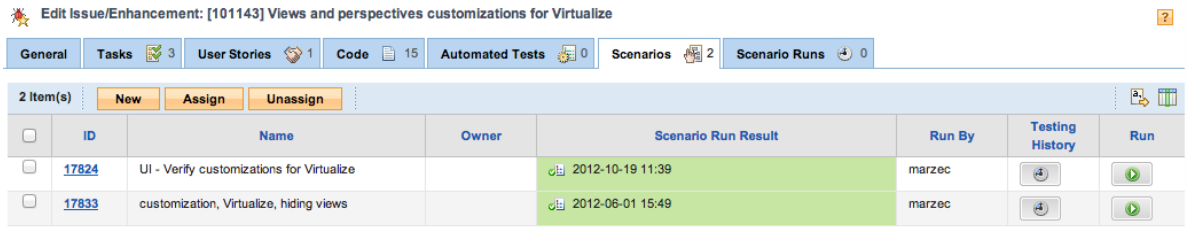
Viewing Additional Test Run Details

You can click the ID of a specific defect/enhancement to see additional test run details:

- **Automated test runs** are shown in the Automated Tests tab.

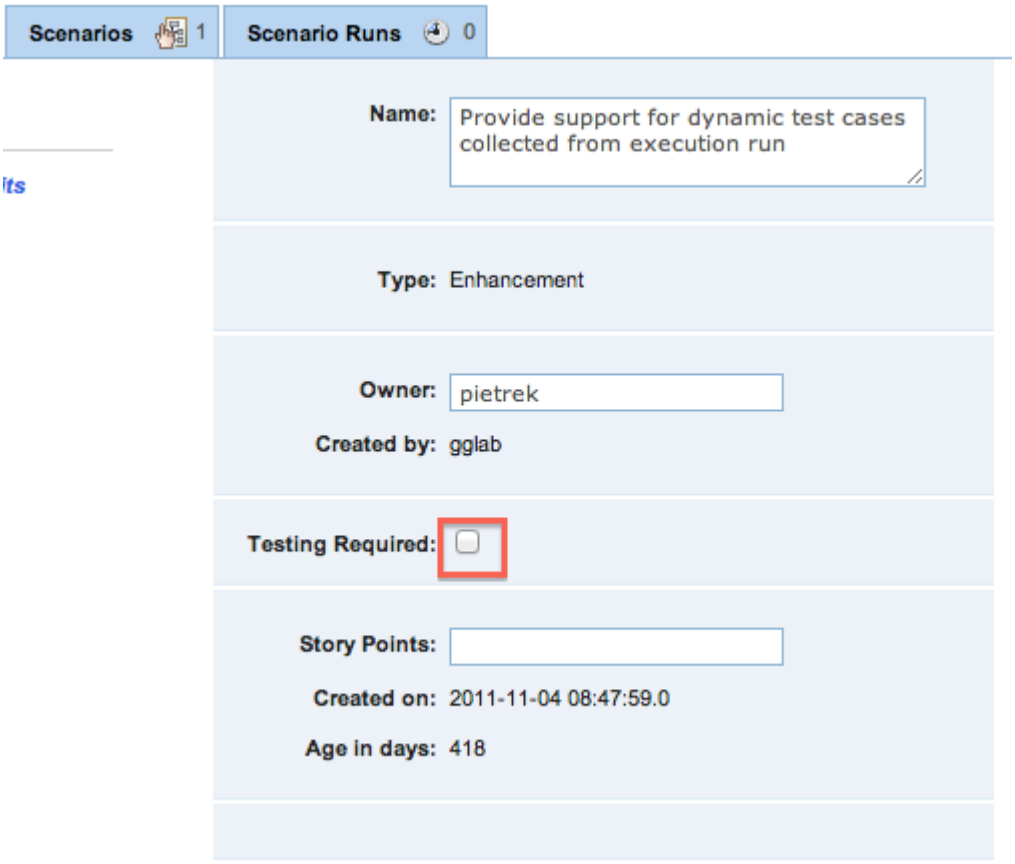


- **Manual test runs** are shown in the Scenarios tab.



What if Testing Is Not Required for a Specific Defect/Enhancement?

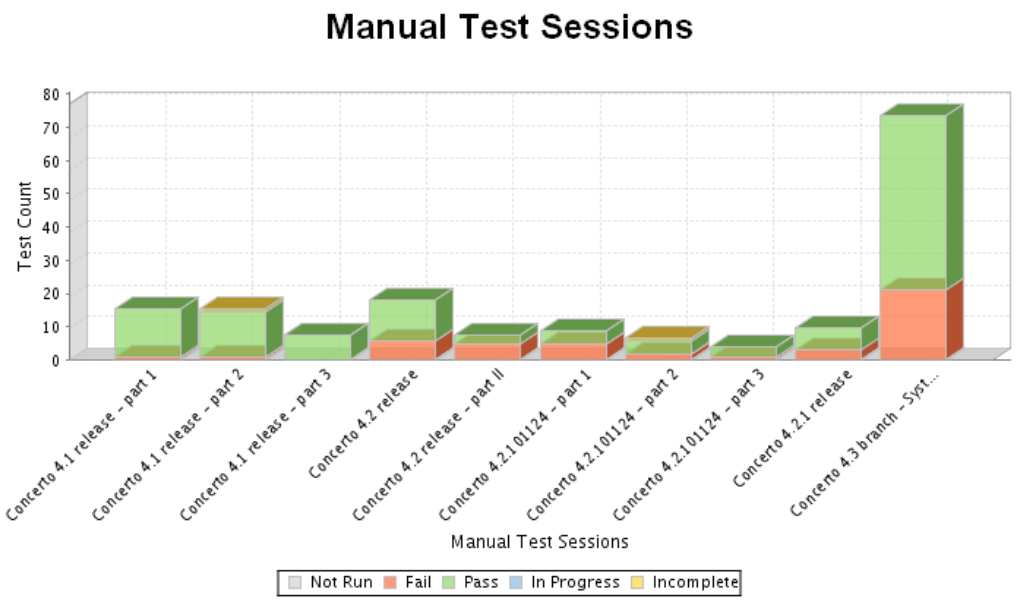
If you do not want to require testing for a specific defect/enhancement, you can mark that defect/enhancement as not requiring testing by clearing its **Test Required** check box.



Manual Test Sessions

From the Reports view, choose **Tests> Manual Test Sessions** to open the Manual Test Sessions report.

This report lists all the manual test sessions that have been run or scheduled for a selected project. The Test Session Status Chart shows the number of tests and their status.



The details for each test session are displayed in a table, which lists data regarding the tests run in each test session, as shown in.

Session Name	Start Date	Closure Date	Duration	Iteration ID	Planned Start Date	Deadline	#Not Run	#In Progress	#Fail	#Pass	#Incomplete	Rank
Concerto 3.5.1 - Basic Acceptance Tests before minor release	Dec 02, 2009	...	1d 2h 3m	417	Dec 02, 2009	Dec 03, 2009	0	0	0	4	0	
Concerto 3.5 release testing [2]	Nov 18, 2009	Nov 20, 2009	1h 33m				0	0	1	0	0	1
Concerto 3.5 release testing	Nov 02, 2009	Nov 20, 2009	12d 5h 34m		Nov 02, 2009	Nov 20, 2009	0	0	18	50	0	3
Scenario runs not belonging to any Manual Test Sessions							-	0	0	19	0	
Total:							0	0	19	73	0	

The Test Session Details Table provides the following information:

- **Session Name**: Name of each session. Click to open the Edit Manual Test Session page, where you can edit the test session details. See the Project Center User's Manual for details about scheduling and editing Manual Test Sessions.
- **Start Date**: Actual date on which test session is started.
- **Closure Date**: Date when the session was marked as closed.
- **Duration**: Based on whether the listed session is in progress or complete, duration means one of the following:
    - **In progress**: Length of time session has run, so far.

    - **Complete**: Length of time session took to complete.
- **Iteration ID**: Identification number of the iteration to which a test session belongs.
- **Planned Start Date**: Planned start date of test session.
- **Deadline**: Scheduled end date of test session.
- **Not Run**: Number of manual tests not yet run.
- **In Progress**: Number of manual tests in progress.
- **Fail**: Number of manual tests failed.
- **Pass**: Number of manual tests passed.
- **Incomplete**: Number of manual tests incomplete.
- **Rank**: Final grade given to the listed session based on manual test results.

Coverage Report

1. From the Reports view, Choose **Tests> Tests Overview**
2. Click the Coverage graph to open the Coverage report.

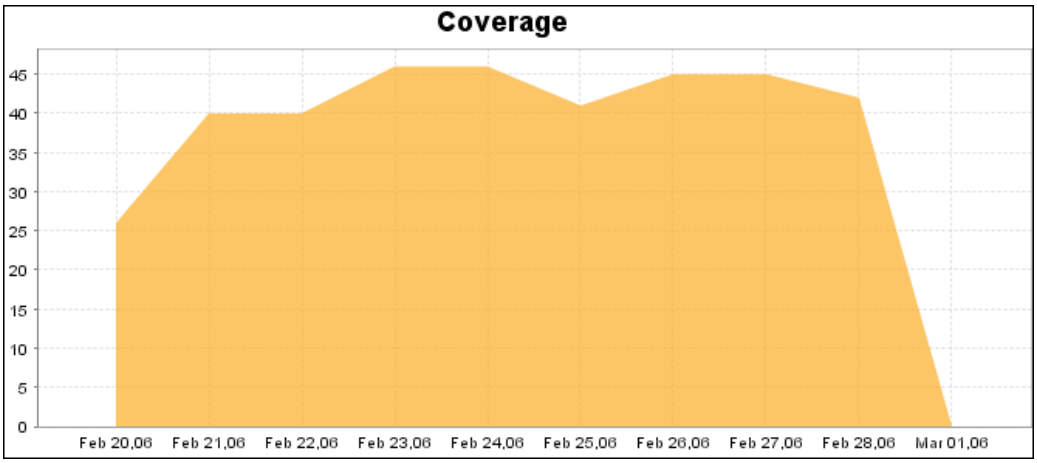
The Coverage graph shows the following information:

- Ratio of tested code to total lines of code.
- Unit test coverage of the selected project.
- A count of tested units and units that still need to be tested.

Notes

- Only line coverage is represented in the graph. MC/DC, branch coverage, and other forms are not shown.
- Coverage is the percentage of succesfully tested lines of code to the total lines of source code selected for testing.

The coverage report helps monitor the progress of unit testing covereage over time. Ideally, the percentage should increase; no change is an indicator that test are not sufficiently being created.



The Coverage table is displayed beneath the Coverage graph and enables architects to see whether coverage is improving.

Date	Unit Tests Coverage
Feb 25, 2008	**35%** (32800 / 94055)
Feb 24, 2008	**38%** (46833 / 123234)
Feb 23, 2008	**38%** (46629 / 123419)
Feb 22, 2008	**39%** (48939 / 124743)
Feb 21, 2008	**38%** (44739 / 118361)
Feb 20, 2008	**51%** (14674 / 28886)
Feb 18, 2008	**39%** (19366 / 50291)
Feb 17, 2008	**42%** (32919 / 78771)
Feb 16, 2008	**43%** (35645 / 82239)
Feb 15, 2008	**41%** (31061 / 74998)

For each drop date listed, the Coverage table shows the following information:

- **%Coverage:** Percentage of coverage.
- **Tested Units:** The number of tested lines of source code.
- **Units To Test:** Total number of lines of source code files that were tested.

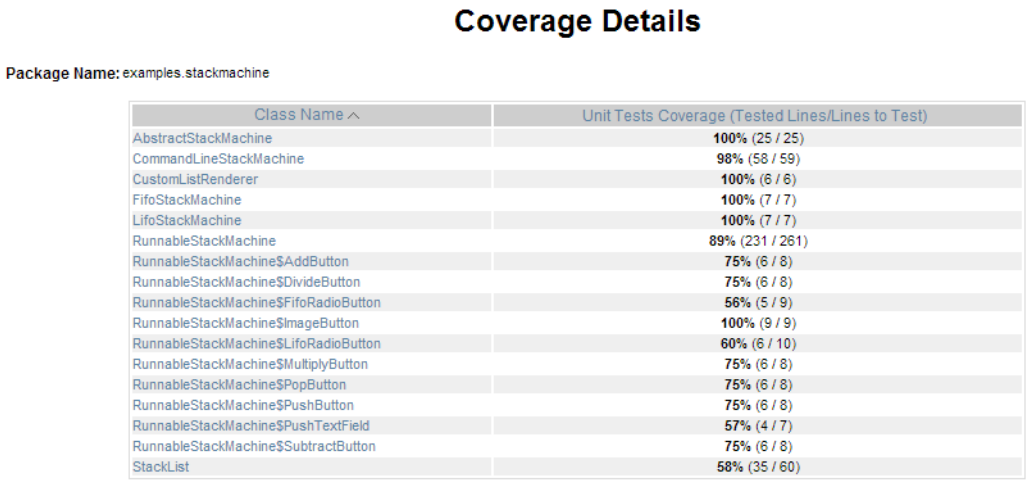
Coverage Details Report

Click on a date in the Coverage table to open the Coverage Details report (for Jtest, dotTEST and Emma results only).

Coverage Details

Package Name	Unit Tests Coverage (Tested Lines/Lines to Test)
examples.stackmachine	**87%** (369 / 425)

Click on a package name to view coverage for the classes in the package.

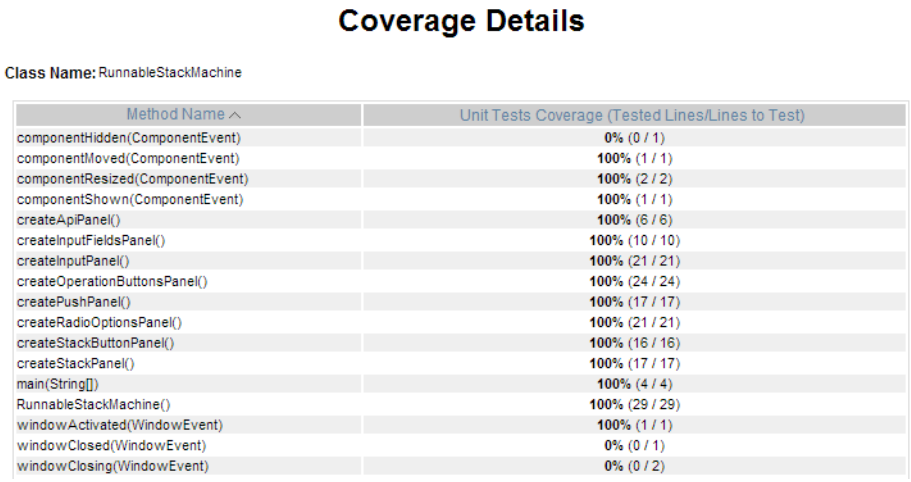


Coverage Details

Package Name: examples.stackmachine

Class Name	Unit Tests Coverage (Tested Lines/Lines to Test)
AbstractStackMachine	**100%** (25 / 25)
CommandLineStackMachine	**98%** (58 / 59)
CustomListRenderer	**100%** (6 / 6)
FifoStackMachine	**100%** (7 / 7)
LifoStackMachine	**100%** (7 / 7)
RunnableStackMachine	**89%** (231 / 261)
RunnableStackMachine$AddButton	**75%** (6 / 8)
RunnableStackMachine$DivideButton	**75%** (6 / 8)
RunnableStackMachine$FifoRadioButton	**56%** (5 / 9)
RunnableStackMachine$ImageButton	**100%** (9 / 9)
RunnableStackMachine$LifoRadioButton	**60%** (6 / 10)
RunnableStackMachine$MultiplyButton	**75%** (6 / 8)
RunnableStackMachine$PopButton	**75%** (6 / 8)
RunnableStackMachine$PushButton	**75%** (6 / 8)
RunnableStackMachine$PushTextField	**57%** (4 / 7)
RunnableStackMachine$SubtractButton	**75%** (6 / 8)
StackList	**58%** (35 / 60)

Click on a class to view the unit test coverage by method.



Coverage Details

Class Name: RunnableStackMachine

Method Name	Unit Tests Coverage (Tested Lines/Lines to Test)
componentHidden(ComponentEvent)	**0%** (0 / 1)
componentMoved(ComponentEvent)	**100%** (1 / 1)
componentResized(ComponentEvent)	**100%** (2 / 2)
componentShown(ComponentEvent)	**100%** (1 / 1)
createApiPanel()	**100%** (6 / 6)
createInputFieldsPanel()	**100%** (10 / 10)
createInputPanel()	**100%** (21 / 21)
createOperationButtonsPanel()	**100%** (24 / 24)
createPushPanel()	**100%** (17 / 17)
createRadioOptionsPanel()	**100%** (21 / 21)
createStackButtonPanel()	**100%** (16 / 16)
createStackPanel()	**100%** (17 / 17)
main(String[])	**100%** (4 / 4)
RunnableStackMachine()	**100%** (29 / 29)
windowActivated(WindowEvent)	**100%** (1 / 1)
windowClosed(WindowEvent)	**0%** (0 / 1)
windowClosing(WindowEvent)	**0%** (0 / 2)

Policy Center (Legacy)

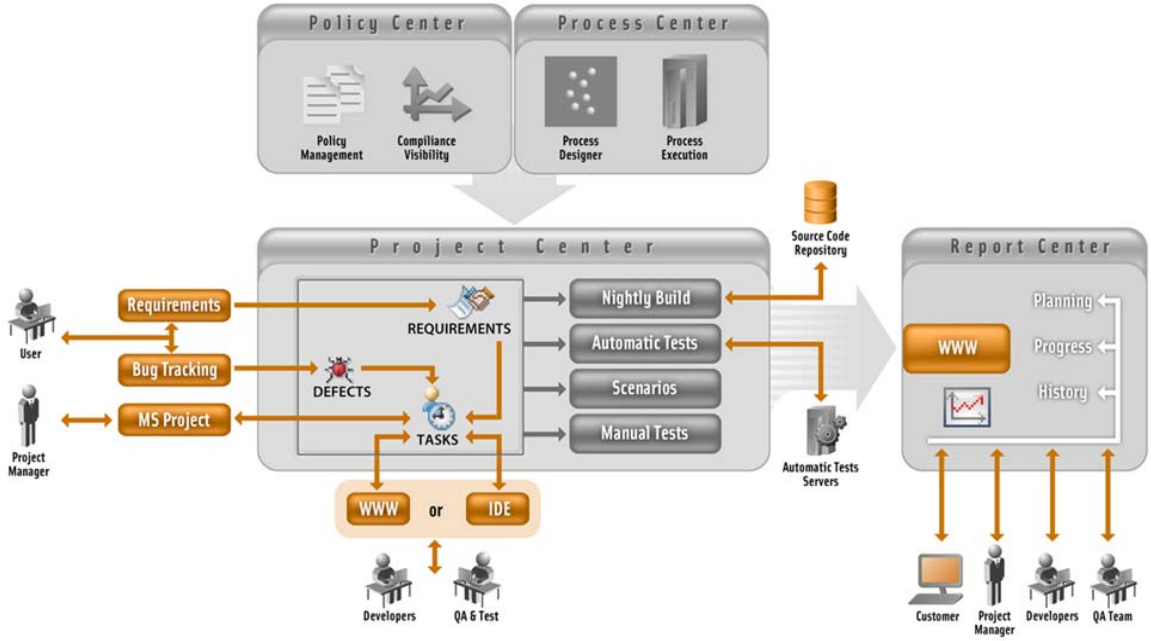
This chapter describes the Policy Center technology that ships with Parasoft Development Testing Platform. You can also download and install an advanced edition of Policy Center that leverages additional features that are not available in the standard edition. Contact your Parasoft representative for additional information.

- Policy Center Overview
- Connecting to Policy Center (Standard Edition)
- Configuring Policy Checking and Reporting
- Reviewing Policy Check Results
- Policy Settings

Policy Center Overview

Policy Center is a system for automatically monitoring the overall status of a software development project. The core of the technology defines several criteria, each of which measures an aspect of the software development process. Each criterion assimilates data from one or more sources and determines whether, and to what extent, a resulting calculation warrants corrective action. Finally, the system determines an overall score and emails all results to specified personnel in a very simple and direct format.

The Policy Center process works by collecting information about the software life cycle from disparate tracking systems (source control, bug tracking, reporting, etc) and applying project specific settings and metrics to determine the overall status of a project each morning.



Connecting to Policy Center (Standard Edition)

If you have a license for Policy Center, you can download and install the advanced edition or activate the standard edition. Contact your Parasoft representative for information on obtaining a license for Policy, as well as information about accessing the advanced Policy Center.

To activate the standard edition of Policy Center:

1. Open the `<DTP_HOME>/conf/ExternalServicesConfig.xml` configuration file.
2. Add the following line in the `<policyCenter>` element:

   ```
   <url>/grs/jsf/policy/project_policy.jsf</url>
   ```

Configuring Policy Checking and Reporting

Configure Policy Check settings to check your policies at the intervals you designate, email reports to the recipients you designate, as well as report policy check results in Report Center's Management Dashboard.

Setup for automated checking and reporting requires:

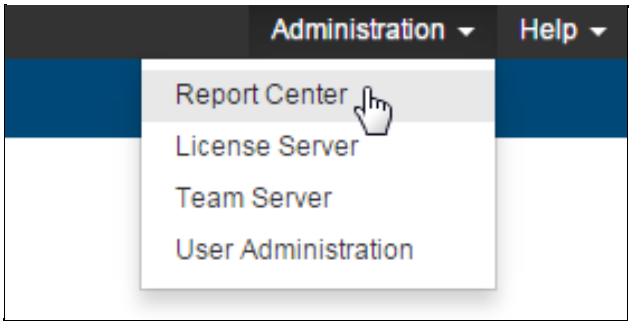
- Setting Up Email Report Recipients
- Setting Up Project Policy Checking and Reporting

In addition to (or in place of) automated checking, you can also run policy checks on demand as described in Running a Policy Check on the Fly.

Setting Up Email Report Recipients

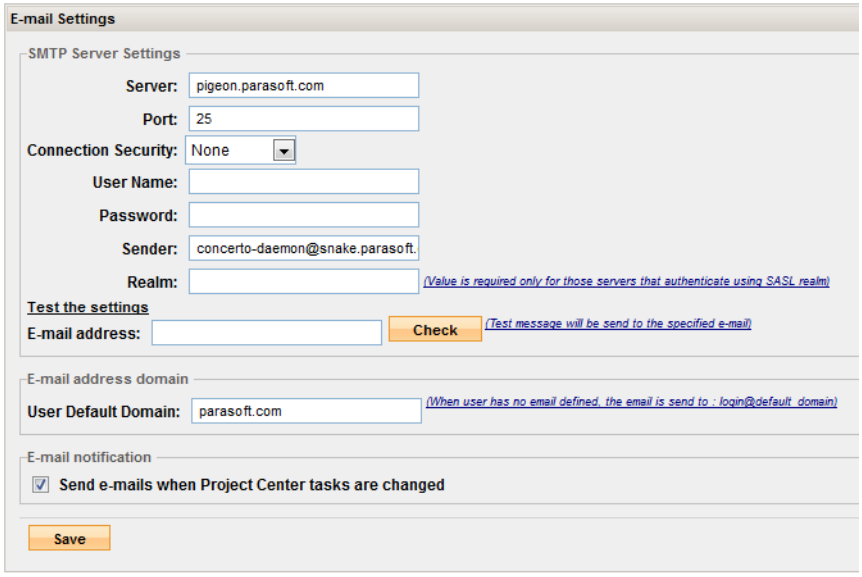
Before using Policy Center, you must perform the following steps to specify who receives email report with policy compliance status:

1. Choose **Administration> Report Center** from the administration toolbar.



2. Choose **Settings> E-mail** and configure your email settings

E-mail Settings



3. Click **Save**.

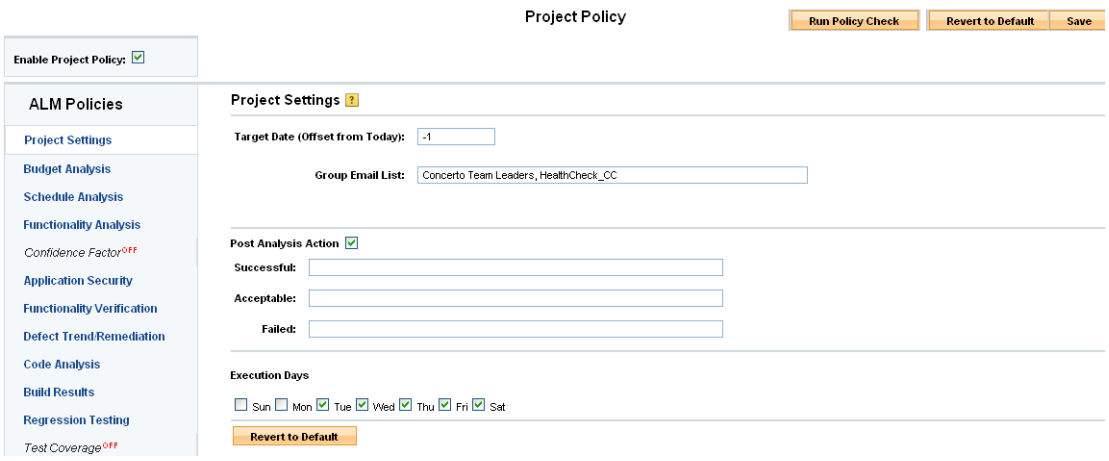
Setting Up Project Policy Checking and Reporting

To enable Policy Center checking and reporting, perform the following steps:

1. In Policy Center, click the **Policy** tab and select **Project Policy** from the menu.

   The Project Policy page is displayed, as shown in Figure 49

Figure 49: Project Policy Page



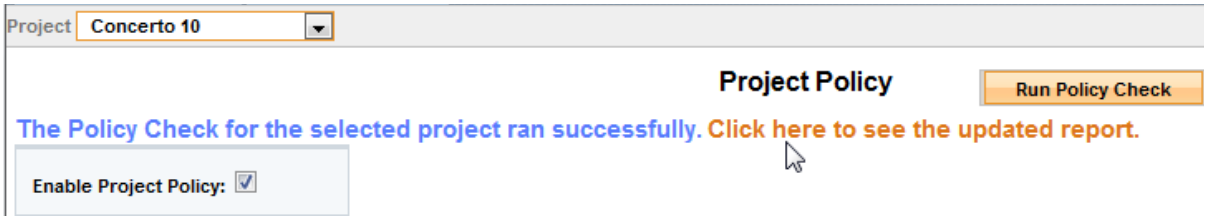
2. Click the **Enable Project Policy** check box.
3. Configure the Policy Settings and other settings as described in “Policy Settings”, page 154. *Be sure to specify which days you want emails generated and mailed.*

4. Click the **Save** button.

Running a Policy Check on the Fly

Policy check is set to run based on your specifications. Clicking the **Run Policy Check** button enables you to run a policy check on the selected project "on the fly" at any time. To run a policy check, perform the following steps:

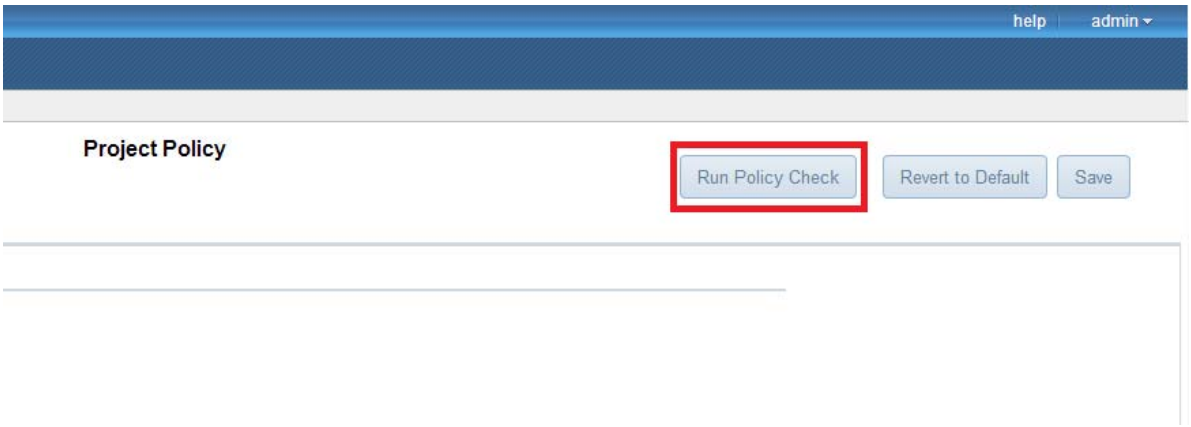
1. Click the **Run Policy Check** button.
2. When the policy check finishes successfully and you see a status message that includes a link to the latest manager report, click that link to view the manager report.



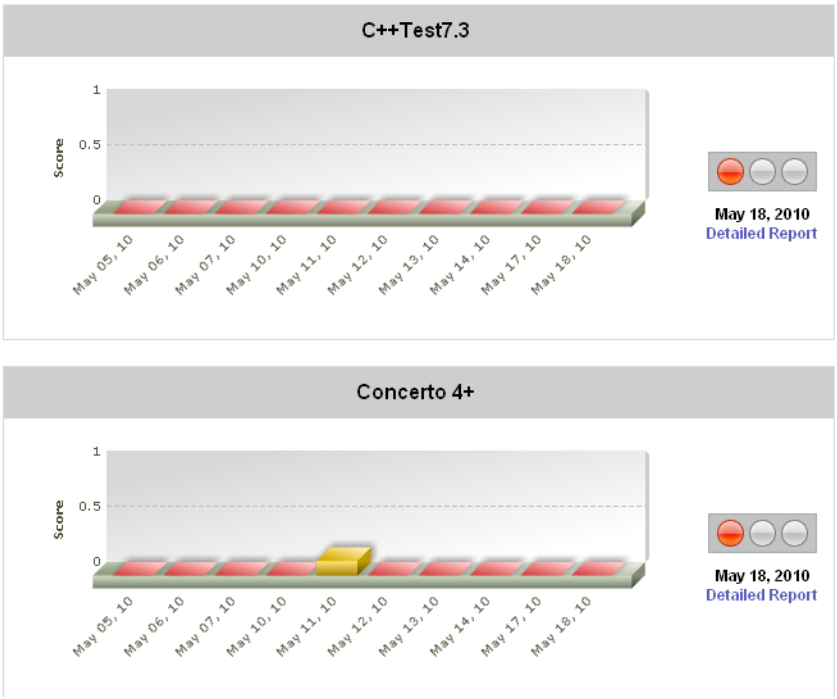
Reviewing Policy Check Results

1. Choose a project from the Project drop-down menu

2. Click **Run Policy Check** button. This may take a few moments

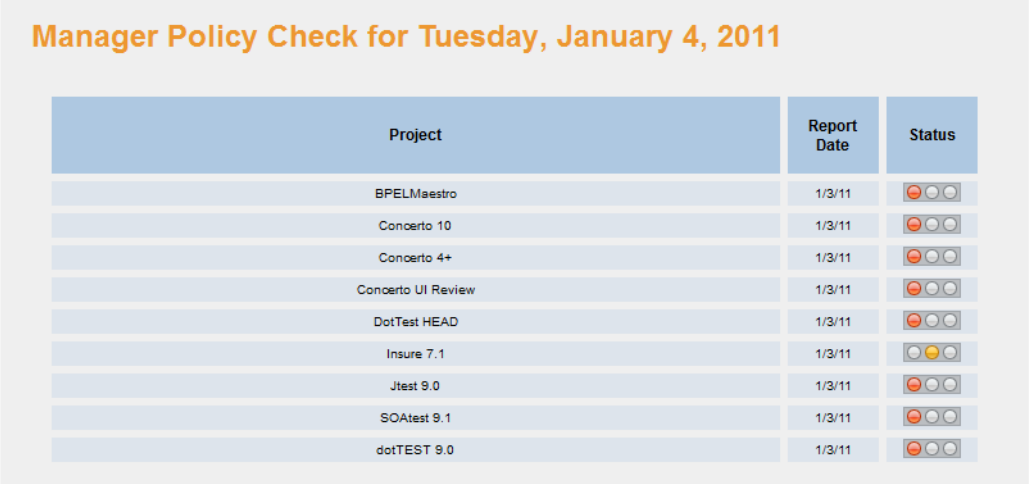


3. Click the **Click here to see the updated report** link to view the Policy Check results.



Reviewing the Email Report

With the appropriate configuration, managers and designated team members receive an email that indicates the status of all projects, and provides a breakdown of each project's status in terms of its budget, schedule, and functionality.



Manager Policy Check for Tuesday, January 4, 2011

Project	Report Date	Status
BPELMaestro	1/3/11	
Concerto 10	1/3/11	
Concerto 4+	1/3/11	
Concerto UI Review	1/3/11	
DotTest HEAD	1/3/11	
Insure 7.1	1/3/11	
Jtest 9.0	1/3/11	
SOAtest 9.1	1/3/11	
dotTEST 9.0	1/3/11	

Responding to Results

The recommended process for working with these reports is:

- Step 1: Review the Report of Overall Project Health
- Step 2: For "Unhealthy" Projects, Review the Project's Overall Deadline, Budget, and Functionality Risk
- Step 3: For Each Reported Risk, Explore Iteration Level Details
- Step 4: For Projects with Quality Risks, Review the Project-Level Health Details
- Step 5: Determine the Appropriate Response

Step 1: Review the Report of Overall Project Health

Yellow or green indicators mean that projects are on schedule, on budget, and on track with the expected functionality—so no action is required.

A red indicator means that the manager should click the **Detailed Report** link to access the project budget analysis, deadline analysis, and functionality completion analysis.

The manager can "manage by exception" and only dedicate resources to addressing items that don't adhere to expectations or policies.

Step 2: For "Unhealthy" Projects, Review the Project's Overall Deadline, Budget, and Functionality Risk

The manager clicking on a light in the top-level report opens a dashboard for the project that shows what part or parts of the project are at risk.

At a glance, the manager can obtain objective, real-time answers to the questions needs to effectively manage the project

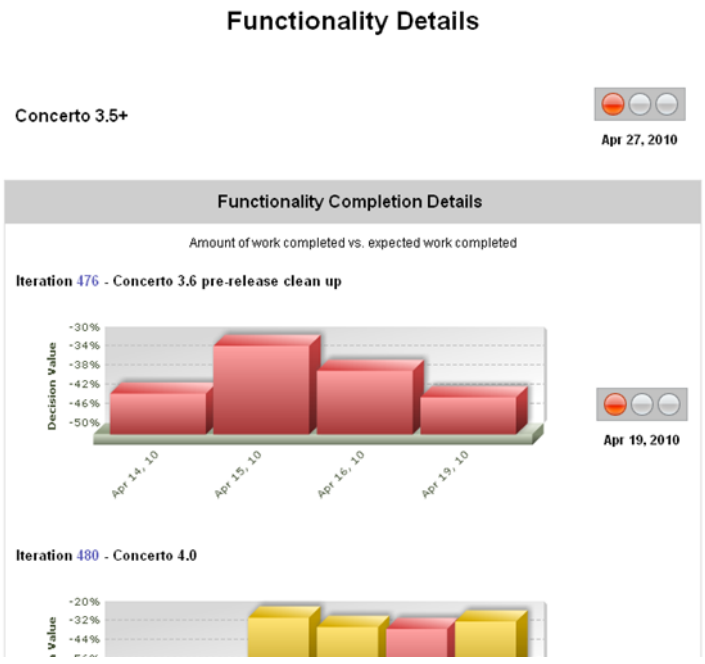
- Is the project's expected functionality implemented properly—and will it work? (Answered in Functionality Completion Analysis)
- Will the project hit the deadline? (Answered in Deadline Analysis)
- Is the project on budget? (Answered in Budget Analysis)

The manager can then drill-down into a per-iteration, detailed report on the at-risk variable, as described in steps 3 and 4. This exploration allows him to get the details needed to make informed decisions about how to address the problem, as covered in step 5.

It is important to note that the functionality completion light carries more information than the budget and deadline lights. It not only considers whether tasks to implement the expected functionality were completed, but also looks at whether the functionality is implemented according to the multiple policies, selected by the management, to guard against quality failures. For example, the manager can use the web interface to specify that the team's test cases must cover at least a certain percentage of the project's source code.

Step 3: For Each Reported Risk, Explore Iteration Level Details

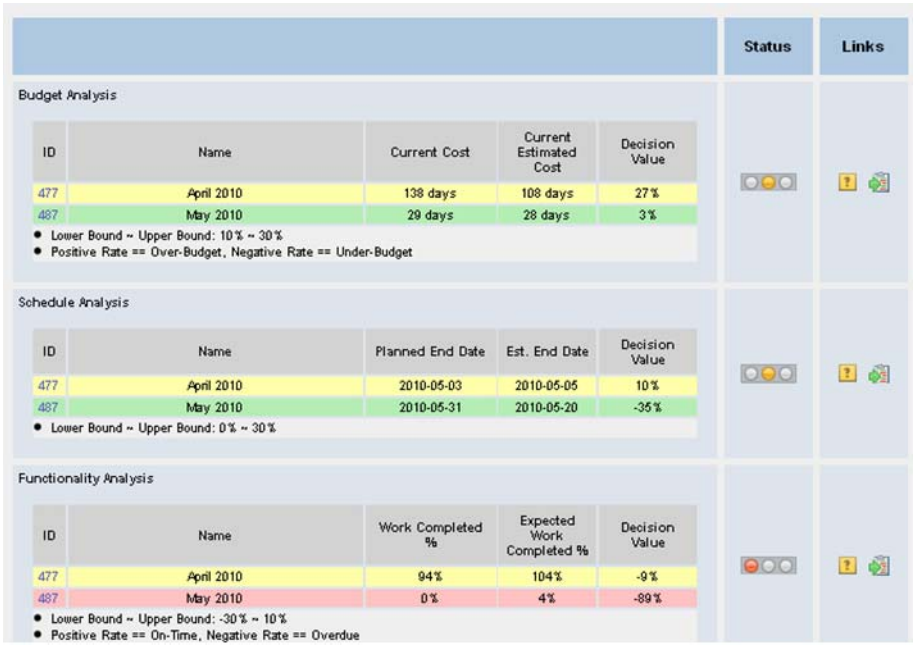
After the manager clicks on the part of the dashboard that corresponds to a project-level red light for budget, deadline, or functionality, he is shown a detailed page that provides a per-iteration breakdown of what iteration is at risk, and why it is at risk The precise page displayed at this point depends on which score was clicked (budget, deadline, or functionality). The various pages are described in "Exploring Policy Check Reports", page 151.

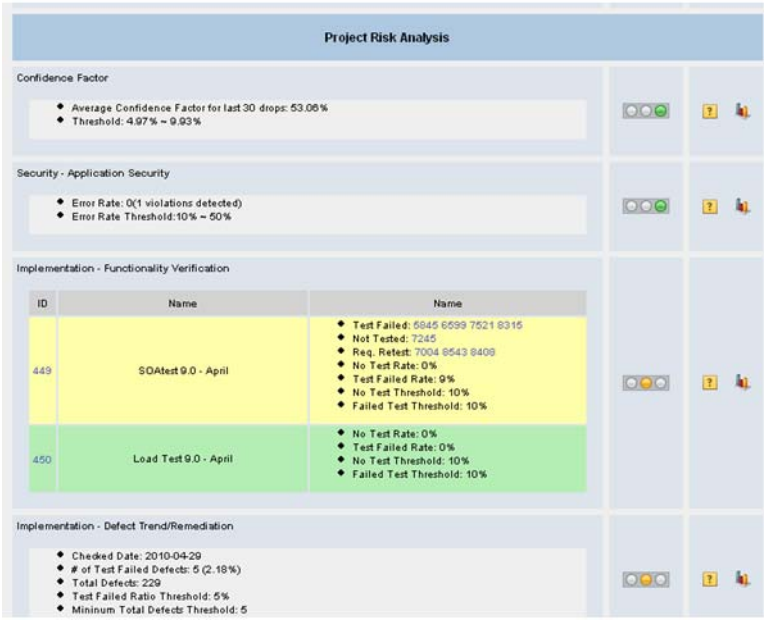


Step 4: For Projects with Quality Risks, Review the Project-Level Health Details

If a Work Quality problem was reported in the Functionality Details report at the previous level, the manager will want to click on any color-coded (red, yellow, or green) item in that report. This opens a

Policy Check Details report that provides easy access to additional details that the manager might be interested in exploring (e.g., to determine why the unacceptable Work Quality score was reported, and to review it in the context of all the project-level health indicators).





Step 5: Determine the Appropriate Response

At this point, the manager can use the information provided to make an informed decision about how to address the reported problem. If he feels that the policies or variable thresholds are not reasonable, he can adjust them in the GUI. For instance, if a budget problem is reported, but the manager wants to be more lenient on the Budget Analysis, he can adjust the related criteria.

Or, if the manager is being alerted about functionality problems because he imposed too strict of a policy for fixing static analysis violations, he could adjust those settings.

Otherwise—if he believes that the project is truly in danger—he needs to determine how to get it back on track. Software development management inevitably involves compromises. For instance, assume that the manager saw a red light reported for Deadline Analysis. He has two main options:

- Reduce the functionality (reducing the amount of work that needs to be complete)
- Increase the budget (complete the remaining work faster by adding more people to the project)

The best solution depends on the manager's priorities. If the manager can find some functionality that is not critical, the first option might be the best course of action. If all of the expected functionality must be completed by the desired deadline, increasing the budget is probably the most viable option.

However, if the manager signed a fixed cost contract and budget is a constraint, then he needs to determine how to manage the functionality and deadline in a way that allows him to stay in the budget. If the budget is allocated incrementally (several disbursements over the project duration) he needs to lay out the project accordingly.

Exploring Policy Check Reports

Both the emailed reports and the manager dashboard report present policy compliance results in a similar manner.

Overall Status

One of three lights (red, yellow, or green) is used to indicate the overall status of each project, which is based on analyses of the project's budget, deadline, and functionality.

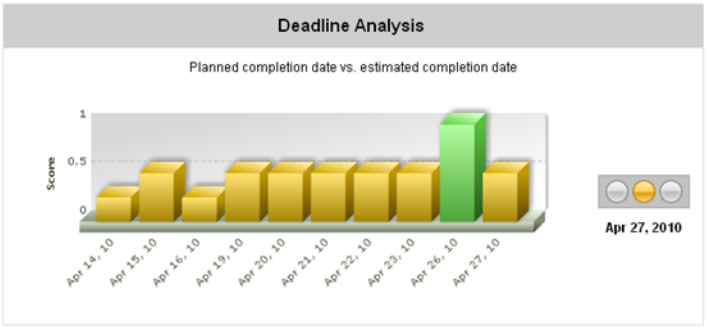
These lights, like every light in the Policy Center reports, correspond to a variable score from 0 to 1. A green light always indicates a score of 1. A red light always indicates a score of 0. Yellow lights indicate that the score is somewhere in between. Default values for acceptable or unacceptable scores for each measured variable are provided by default, and can be customized as described in "Policy Settings", page 154.

Deadline Analysis

This indicates if the project is currently late or is in jeopardy of being delayed as of the day of the report.

A green light is presented when the project is on-time or within a reasonable threshold of being on-time as determined by a "not to exceed" boundary percentage. The bounds can be specified as described in "Policy Settings", page 154.

Real-time feedback on how the project is progressing is used to improve the accuracy of the deadline that was initially estimated for the project.



Functionality Completion Analysis

The Functionality Analysis light communicates whether planned features are completed and comply with the designated quality policies.

The Functionality Completion Details reports the actual amount of work completed vs. the expected amount of work completed.

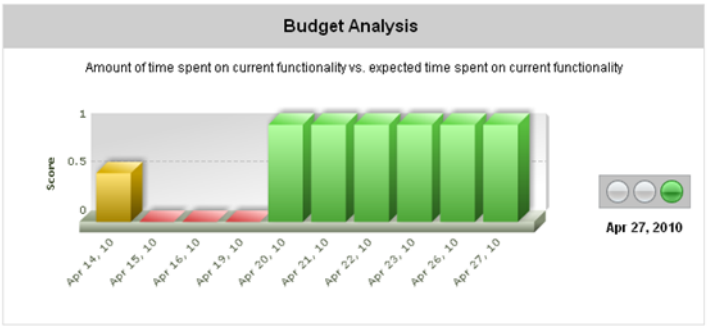
If planned functionality is currently late or is in jeopardy of being delayed as of the day of the report, you will receive a warning. A green light is presented when the planned functionality is delivered on-time or within a reasonable threshold of being on-time, as determined by a “not to exceed” boundary percentage. The bounds can be specified as described in “Policy Settings”, page 154.

If the team does not adhere to any of the designated policies (e.g., perform static analysis, perform unit testing, etc.), this raises doubt about whether the functionality is actually implemented properly.

Budget Analysis

This report indicates if the project is currently exceeding budget or is in jeopardy of exceeding budget on the day of the report.

A green light is presented when the project is on budget or within a reasonable threshold, which is parameterizable. The bounds can be specified as described in “Policy Settings”, page 154.


Budget Analysis
Amount of time spent on current functionality vs. expected time spent on current functionality
Score
1
0.5
0
Apr 14, 10
Apr 15, 10
Apr 16, 10
Apr 19, 10
Apr 20, 10
Apr 21, 10
Apr 22, 10
Apr 23, 10
Apr 26, 10
Apr 27, 10
Apr 27, 2010

Policy Settings

A policy is the basis for how well a project is performing. Development Testing Platform allows you to set the threshold for each policy.

From Project Center's Policy drop-down menu, you can configure:

- Manager E-mail Settings: Allows you to define the list of people who will receive manager report e-mails after the policy check is run.
- Global Policy: Allows you to set default values for each checker.
- Project Policy: Allows you to enable Policy Center reports for Development Testing Platform and configure the policy checkers for each project. Policy Center can also send e-mail to managers about the policy status of projects.

Configuring Policy Center Settings

The data presented in manager reports is calculated based on the values set in the configuration of each policy on the Project Policy page. There are default values provided but we recommend that you configure each checker that best fits your environment. You could also enable/disable each checker.

The following project policies can be enabled and configured for Policy Center.

- Project Settings
- Budget Analysis
- Schedule Analysis
- Functionality Analysis
- Application Security
- Application Security
- Functionality Verification
- Defect Trend/Remediation
- Code Analysis
- Build Results
- Regression Testing
- Test Coverage
- Defect Trending
- Source Code Trending
- Peer Review
- Unit Tests Executed

UI Indicators

- An asterisk (*) is used to indicate changed values
- Italics are used to indicate that a checker is disabled.

Project Settings

Project Settings specifies general settings for project policies. They indicate when the policies should be checked and what actions should be performed after the check is completed.

.

Setting	Description
Target Date	The specified project is checked daily by a background job. This "Health Check Back End" job is run at 7 am by default. The date offset specifies which day's data should be considered for the policy check. For instance, if it is set to -1, Policy Center looks for the previous day's data (e.g., if it is run at 7 a.m. Tuesday, it will analyze the project data (test results, schedule/budget) from Monday). If was set to 0, it would look for the current day's data.
Group Email List	After the policy check is run, it sends an email report to all members of the User Administration groups specified here. For details about User Administration groups, please see "User Administration", page 301 To specify multiple groups, use CSV format (e.g., MyProjectA, MyProjectB, MyProjectC).
Post Analysis Action	Specifies one or more commands (for example .bat/.sh scripts) that the server will execute after the policy check is completed. If you want to specify post analysis actions, check the **Post Analysis Action** box, then specify the command in the appropriate field: • **Successful:** If you want the command (script) to be executed when the specific project policy check has a "Successful" result (Green light). • **Acceptable:** If you want the command (script) to be executed when the specific project policy check has an "Acceptable" result (Yellow light). This occurs when the result falls between the defined Lower/Upper bounds. • **Failed:** If you want the command (script) to be executed when the specific project policy check has a "Failed" result (Red light). This occurs when one (or more) of the policy factors has "Failed" (Red light) status Since this command (script) will be executed by the server, it's important to ensure it is available to run on this machine.
Execution Days	The days of the week for which Policy Check executes to generate the report.

Budget Analysis

Budget Analysis reports if the project is currently exceeding budget or is in jeopardy of exceeding budget on the day of the report. A green light is presented when the project is on budget or within a reasonable threshold (parameterizable).

Setting	Description
Lower Bound	Lower bound for budget threshold in percent. Lower than this threshold will be reported as good.
Upper Bound	Upper bound for budget threshold in percent. Higher than this threshold will be reported as bad.

In Development Testing Platform, a project's budget and costs are measured in days (not funds).

The initial budget analysis is based on the sum of the estimations of work for individual tasks in an iteration. Individual tasks are estimated by team members when they add tasks. Development Testing Platform determines an estimation for the entire project by combining this information. This estimation of the total number of work hours is spread over the expected duration of the project as a linear function.

As team members work on tasks, Development Testing Platform measures how much time they are actually spending on each task. Then, Development Testing Platform compares this data with the estimates to determine if the iteration is on budget. If measurements indicate that the team is not within the expected range of work hours projected for the current point in the project, then a red light is used to alert the manager to this problem.

You can also select which Task Types will be included when Policy Center calculates the budget.To select task types, check the appropriate box corresponding to the task type you would like to include. For an explanation of different task types, see /* x-ref to creating tasks in pc man */.

Schedule Analysis

Schedule Analysis reports if the project is currently late or is in jeopardy of being delayed as of the day of the report. A green light is presented when the project is on-time or within a reasonable threshold of being on-time as determined by a not to exceed boundary percentage.

Setting	Description
Lower Bound	Lower bound threshold for deadline due in percent. Lower than this threshold will be reported as good.
Upper Bound	Upper bound threshold for deadline due in percent. Higher than this threshold will be reported as a problem.

Any percentage value in between lower bound and upper bound is considered to be yellow. The schedule analysis checker only analyzes iterations whose “estimated end date" is in the future, relative to the day of analysis.

Functionality Analysis

Functionality Analysis reports on the status of planned features. If planned functionality is currently late or is in jeopardy of being delayed as of the day of the report you will receive a warning. A green light is presented when planned functionality is delivered on-time or within a reasonable threshold of being on-time, as determined by a not to exceed boundary percentage.

Setting	Description
Lower Bound	Lower bound for task completion in percent. Lower than this threshold will be reported as a problem (Overdue).
Upper Bound	Upper bound for task completion in percent. Higher than this threshold will be reported as good (On-time).
Grace Period Threshold	Percentage of expected task completion required before reporting current task completion as a problem. A warning is reported only if you pass the grace period threshold.

The functionality analysis indicates—for each project iteration—the actual amount of work completed vs. the expected amount of work completed. A red light for any iteration will trigger a red light at the project level.

Work Completed is calculated as follows:

> Sum of the estimated cost of work completed / all Estimated work completed * 100

Expected Work Completed is calculated as follows:

> all Estimated work completed / # of days for the iteration * # of days past since their iteration begins

Application Security

The Application Security report monitors the number of failed security tests per thousand lines of code (KLOC). A warning is issued based on where the number of failed security tests per KLOC falls on the specified bounds. A green light is presented when this number is below the lower bound.

Setting	Description
Lower Bound	Lower bound for # of security tests per 1KLOC.
Upper Bound	Upper bound for # of security tests per 1KLOC.

Functionality Verification

Functionality Verification monitors the percentage of "development tasks" that have either missing or failing test cases as of the day of the report. A warning is issued based on the percentage of tasks with

a missing or failing test suite. A green light is presented when the percent of missing or failing test suites are in an acceptable range.

Setting	Description
% of Tasks with Missing Tests	% of Tasks missing test cases for upper bound.
% of Tasks with Failed Tests	% of Tasks with failed test cases for upper bound.

Defect Trend/Remediation

Defect Trend/Remediation report monitors the percentage of "defects" with failed test cases or missing tests cases. A warning is issued based on the percentage of defects with a missing or failing test suite or a minimum count of defects with missing or failing test suites is exceeded. A green light is presented when the percent of missing or failing test suites are in an acceptable range.

Setting	Description
% of Defects with Failed Tests	% of Defects with failed tests.
Acceptable Minimum Total Defects	Acceptable or allowable minimum # of total defects
% of Defects with Missing Tests	% of defects with missing tests.

DefectTestsChecker checks to ensure that every defect has at least one test case associated with it—and that the test passes during the nightly test. If the test case associated with a defect fails, this indicates that a serious regression was introduced and the defect should be reviewed as soon as possible.

If no tests are associated with a defect, it is not possible to automatically check whether code modifications re-introduced the defect. This should be reviewed as a second priority (yellow) issue.

If the ratio of unresolved defects is higher than the threshold, it should be reviewed (yellow).

Code Analysis

The Code Analysis report is triggered by a range of detected code analysis violations per KLOC. If code analysis violations exceed a parameterized number of violations per KLOC as of the day of the report you will receive a warning. Additionally, a warning is issued when an upper bound of new

violations are discovered on the day of the report. A green light is presented when code analysis violations are within an acceptable range of defects per KLOC.

Setting	Description
Acceptable New Violations Per Day	Upper bound for # of new violations for the day
Lower Bound	Lower bound for # of violations per 1KLOC
Upper Bound	Upper bound for # of violations per 1KLOC

Build Results

Build Results monitors the warning messages from the build process and reports if the project exceeds a percentage of files with warning messages. A green light is presented when the percentage of files without a warning message is within a reasonable threshold.

Setting	Description
Acceptable % of Warning Messages	Acceptable % of files with warning messages for the project.

Regression Testing

The Regression Testing report monitors the percentage of "test failures" versus an upper and a lower boundary. A warning is issued based on exceeding a specified percentage of test failures. A green light is presented when the percentage of test failures is within an acceptable range.

Setting	Description
Lower Bound	Lower bound for % of test failures.
Upper Bound	Upper bound for % of test failures.

Test Coverage

The Test Coverage report monitors the percentage of unit testing line coverage for the project versus an upper and a lower boundary. A warning is issued based on too low of test coverage. A green light appears when the unit testing line coverage percentage is within an acceptable range.

Setting	Description
Lower Bound	Lower bound for % of test coverage.
Upper Bound	Upper bound for % of test coverage.

Defect Trending

The Defect Trending report monitors an upper and lower bound of defects injected in the project. A warning is issued based on a relatively high number of new defects discovered versus total defects.

Setting	Description
Lower Bound	Lower bound for # of new defects / # of new defects.
Upper Bound	Upper bound for # of new defects / # of new defects.

The Defect Trending checker checks for the total number of new defects introduced the day before, divided by the total number of defects fixed in same time period. If the ratio is greater than upper bound, it means that more defects were filed than fixed. This indicates that a project may go into defects creep and should be reviewed.

Source Code Trending

Source Code Trending is a macro monitor for the development team's progress. A warning is sent if a high percentage of lines of code are removed compared to the total lines of code. A warning is also issued when code is not committed to the source code repository within a determined day range. A green light is presented when the code is being committed to the source repository and in a reasonable time interval and total lines of source code is growing.

Setting	Description
Line Change Lower Bound	Lower bound % for the line difference between target date and previous date
Line Change Upper Bound	Upper bound % for the line difference between target date and previous date.
Activity Date Lower Bound	Lower bound for last commit date range in days.
Activity Date Upper Bound	Upper bound for last commit date range in days.

Peer Review

The Peer review report monitors if peer code reviews are being executed as expected and issues discovered via the peer review process are being fixed. A warning is issued if an individual has too many peer review tasks queued in their task list or if the number of tasks with issues exceeds an

acceptable range or the total number of outstanding tasks exceeds an acceptable count. A green light is presented when the peer review is being executed as expected.

Setting	Description
Maximum Tasks Lower Bound	Lowerbound for # of Maximum tasks allowed.
Maximum Tasks Upper Bound	Upper bound for # of Maximum tasks allowed.
Issues Lower Bound	Lower bound for # of tasks with issues (Reviewer found issues in code).
Issues Upper Bound	Upper bound for # of tasks with issues (Reviewer found issues in code).
Outstanding Lower Bound	Lower bound for # of outstanding tasks (Reviewer didn't review issues yet).
Outstanding Upper Bound	Upper bound for # of outstanding tasks (Reviewer didn't review issues yet).

A green light is presented if there are fewer outstanding tasks than the Outstanding lowerbound, as well as fewer tasks with issues than the Issues Lowerbound.

A red light is presented if there are more outstanding tasks or tasks with issues than their upper bounds allow, respectively.

The rate of issues is calculated as:

> If Outstanding rate is lower than Lower bound for outstanding rate threshold AND issue rate is lower than Lower bound for issues, check if total # of today's review is lower than acceptable outstanding issue.

The rate of outstanding code review tasks is calculated as:

> (Today's Total review (including "To Review" and “Issues for developer" - Yesterday's Total Review)/Yesterday's Total Review * 100

Unit Tests Executed

The Unit Test Executed report monitors both the execution of all unit tests associated with the project as well as the percentage of unit test failures. If the total percentage of unit tests executed falls below an acceptable percentage (over the last 10 drops) or the percentage of test case failures exceeds an acceptable percentage then a warning will be issued. A green light is presented when unit tests are being executed as expected with an acceptable range of failures.

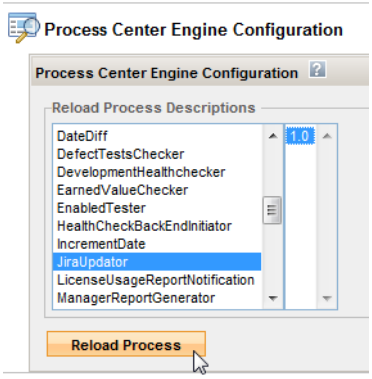
Setting	Description
Total Test Cases Difference Lower Bound	Lower bound for % of total test case difference during last 10 drops.
Total Test Cases Difference Upper Bound	Upper bound for % of total test case difference during last 10 drops.

Setting	Description
Test Failed Ratio Lower Bound	Lower bound for % of failed test case.
Test Failed Ratio Upper Bound	Upper bound for % of failed test case.

Redeploying a BPEL Process Over the File System

If you are deploying a BPEL process over the file system, you perform the following steps to reload it:

1. Copy the process into the appropriate directory.
2. In Report Center, select **Administration**,
3. Select **Settings > Policy Center > Process Center Engine**. The Process Center Engine page is displayed, as shown below.



4. Select the reload process description, click the version number on the right column, and then click **Reload Process**.
5. Check to see if the "process description has been reloaded" message is displayed at the top of the page.

Administration

- Installing Development Testing Platform
- Configuring Development Testing Platform
- Integrations
- Development Testing Platform APIs
- Build Administration
- Appendix for Development Testing Platform Administration

Project Creation and Configuration

Most activities are performed in context of specific project. Generally, a project name and the team members involved need to be specified. The Project Creation and Configuration chapter in the DTP user manual covers most project-related actions. Click the **help** link in the navigation bar in the DTP UI to access the documentation.

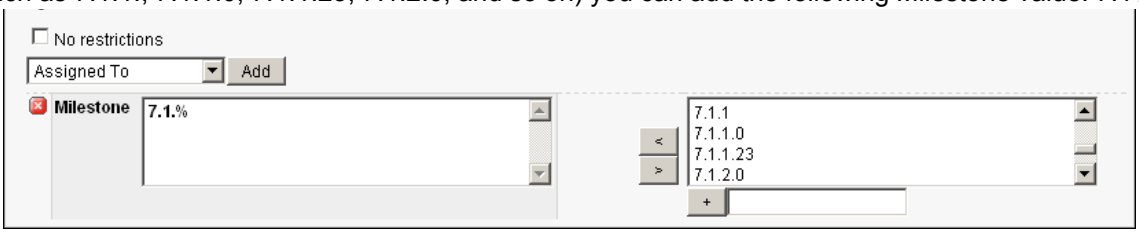
Project Definitions

You can define project parameters, such as filters that allow you to control the data associated with your project. By default, team members are restricted from all data for all projects, but a project manager can configure the project definitions to grant team members access. All data is available when the **No restrictions** option is enabled for a definition, but project managers can disable the option to restrict data.

Wildcard Usage in Project Definition Filters

You can use the percentage symbol (%) as a wildcard to represent any character or string in your search to expand your search. The following example demonstrates how to use the wildcard:

If you want your search to include all milestones for C++test 7.1, instead of listing each one separately (such as 7.1.1., 7.1.1.0, 7.1.1.23, 7.1.2.0, and so on) you can add the following Milestone value: 7.1%.



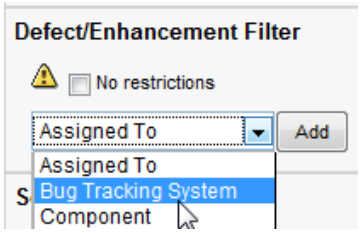
Search results will include all milestones for C++test 7.1.

Defect/Enhancement Filter

Project Center integrates with many bug tracking systems, as described in “Integrating with Bug Tracking Systems and Requirement Management Systems”, page 215.

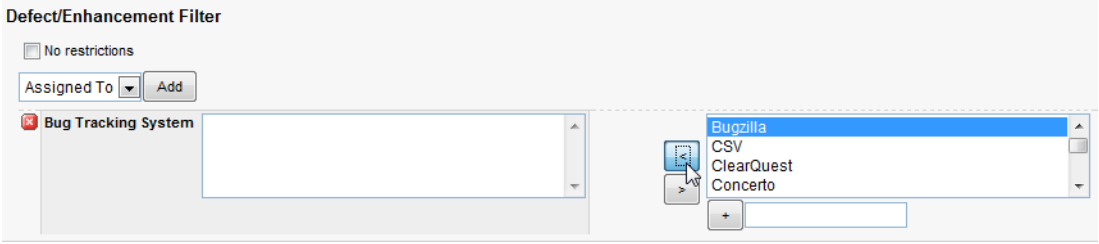
To associate a project with a bug tracking system:

1. Choose **Bug Tracking Systems** from the Defect/Enhancement Filter drop-down menu and click **Add**



A list of bug tracking systems that have been integrated with DTP appears. The list uses labels assigned by your DTP administrator.

2. From the list of systems on the right, select the bug tracking system (BTS) you wish to integrate, then click the < button to associate this BTS with your project.

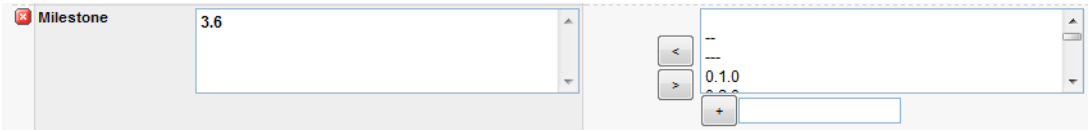


This integrates the project with *all* items from the selected bug tracking system.

To narrow the filter criteria to a specific project:

1. From Bug Filter drop-down menu, choose **Project**.
2. Click **Add**. A list of projects is displayed. These are the projects that were imported from the integrated BTS.
3. From the list of project on the right, choose the component you wish to integrate, then click the < button to associate this component with your project.

You can continue fine-tuning the filter in this manner (selecting a filter category from the Bug Filter drop-down, then adding the desired filter values in to the left side list).



Log Filters

You can use the Log Filter to configure restrictions based on logs that are sent to Report Center, for example, configuring Report Center to show only data sent by Jtest. Every tool that sends results to Report Center creates a log, which is a record of a tool run. Every log has the following pre-defined properties:

- **Machine:** Name of the machine on the system that ran the tool.
- **OS Architecture:** Architecture of the system that ran the tool.
- **OS Name:** Name of the operating system on the system that ran the tool.
- **OS Version:** Version of the operating system on the system that ran the tool.
- **Tool:** Name or symbol of the tool that conducted the test, for example, if the test was run by Jtest 4.5.1, the tool is Jtest.
- **Tool Version:** Version number of the tool that conducted the test, for example, if the test was run by Jtest 4.5.1, the version is 4.5.1.
- **User:** Account of the user who was logged in to the system when the system ran the tool.

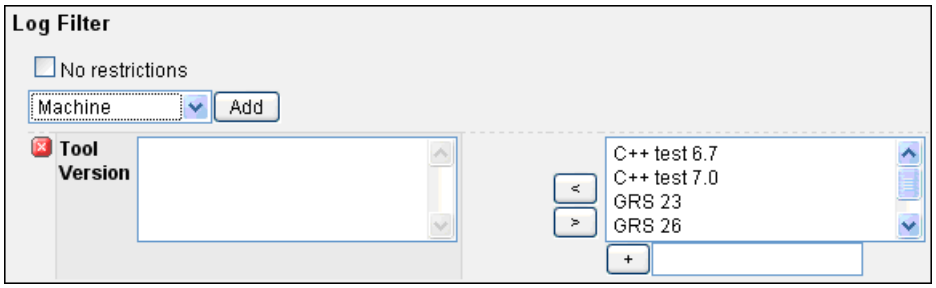
To create a filter that restricts data based on one of the log filters listed above:

1. In the Log Filter box, ensure that the **No restrictions** option is cleared.

Figure 50: Project Filters Definition

2. Choose an item from the Log Filter drop down list, and then click the **Add** button next to it. A page similar to the following is displayed:

Figure 51: Log Filter



3. Specify your restrictions.

   The list on the right shows all of the values for the given property that have already been sent to Report Center. Consequently, if C++test x.x has already sent results to Report Center, that list will contain the value “C++test x.x”. Some values are added to the Report Center database during installation, so this list will contain values for tools even if the tools have not run yet.

   - **To configure the log restriction:** Select the proper item(s) from the list on the right, and then click the less than symbol (<). The selected items move to the list on the left to define restrictions for data.
   - **To remove any restrictions:** From the list on the left, click the more than symbol (>). The selected items move back to the list on the right.
   - **To set restrictions for a value that is not yet represented in the Report Center database (and, thus, is not available in the drop-down list):** Type the name of the value in the text field below the multi-selection list, and then click the plus symbol (+). The value you typed is added to the multi selection list and can be selected from that list.

4. Save the configuration.

Log Properties Filter

Each log can have attributes as well as fixed properties, and you can use the Log Properties Filter to filter data based on these attributes.

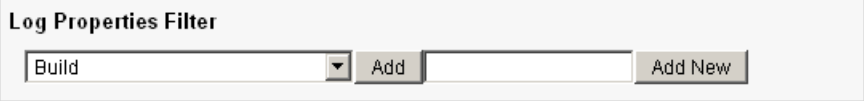
To configure a Log Properties Filter:

1. In the Log Properties Filter Box, deselect the **No restrictions** option (if it isn’t already).

2. Select the appropriate log property from the drop-down list, and then click **Add**.

   If the log property that you want to add is not listed, go to the next step.

Figure 52: Log Properties Filter



3. Perform any of the following tasks:

   - **To add a new log property to the list:** Type the name of the value in the Add New text field, and then click **Add New**. The value you typed is added to the list and selected in that list.
   - **To configure multiple log properties:** You can configure as many log properties as required. Each log filter can be reconfigured as often as needed, but only one configuration per log can exists for given project.
   - **To remove log properties:** Next to each previously configured log property that you want removed, click the red button with the plus symbol (+) on it.

4. Click **Save** at the bottom of the Project Filters Definition page.

Test Group Properties Filter

You can filter data by test group attributes. The Test Group Properties Filter works the same way as the Log Properties Filter does—with one difference: the Test Group Properties Filter attributes are assigned to a test group rather than to a log.

Report Center handles many tests and to keep the results of those tests organized, they are grouped into "test groups". A test group is made up of a set of tests and other test groups. Each test group can have associated attributes (just as logs have associated attributes). The attributes describe a given test group, not the entire test run. For example, a test group of static analysis tests will have attributes associated with static analysis. A test group of white box tests will have attributes related to white box testing.

You can use the Test Group Properties Filter to provide greater flexibility in defining projects for Report Center reporting. Apply the Test Group Properties Filter in the same way that you apply the Log Properties Filter.

Figure 53: Test Group Properties Filter

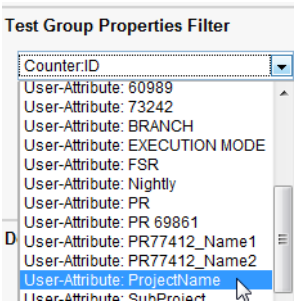
Example

You can create filters for defining which automated test results received by Development Testing Platform should be associated with a specific project.

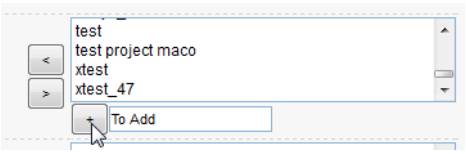
Before defining a filter, you must ensure that your Parasoft Test tools are sending results; for details, see “Connecting Report Center and Project Center to Parasoft Test”, page 209. Once this configuration is performed, each automated test run by Parasoft tools (Jtest, C++test, dotTEST, etc.) will send results marked with "User-Attribute: ProjectName".

To configure Project Center to associate the related automated tests with a project:

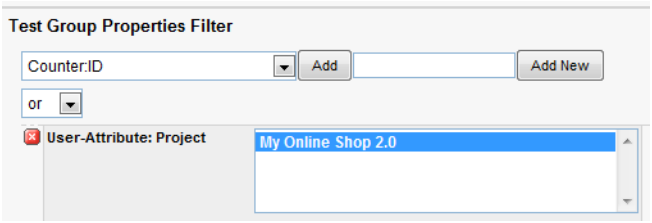
1. From Test Group Properties Filter drop-down menu, choose **User-Attribute: ProjectName**.



2. Click **Add**. You will see a list of project names that a) match your search criteria and b) have test results in Development Testing Platform.
    - Note that if your project name is not listed, it means that test results marked with such attributes have not yet been received by Development Testing Platform. You can add such attribute manually by typing the name add clicking the [+] sign.



3. Select the name of the appropriate project, then click the < button to add it the left panel.



Now when automated test results are sent from Parasoft Tool to Development Testing Platform and they are marked with the attribute you specified (in this example, "User-Attribute: ProjectName=My Online Shop 2.0"), the results will be associated with the project.

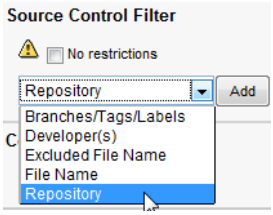
Source Control Filter

Project Center integrates with multiple source controls, as described in “Integrating Report Center with Source Control Management Systems”, page 275.

When you integrate a source control system with Parasoft Development Testing Platform, it is periodically scanned by Parasoft Source Scanner, and your project code revision information is fed into the Project Center database.

To integrate Project Center with a source control system, perform the following steps:

1. From the Source Control Filter drop-down menu, choose **Repository**.



2. Click **Add**. A list of repositories (the names of Source Scanner projects that were specified by your Development Testing Platform Administrator) will be displayed.
3. From the list on the right, choose the name of the repository you wish to integrate, then click the < button to add this as a part of source filter for your project.

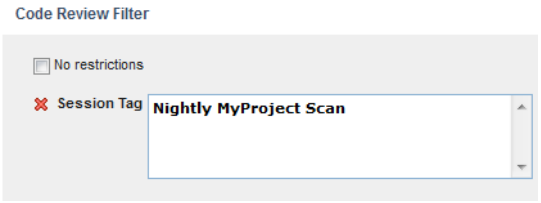
To further narrow the source filter, continue adding conditions in this manner. For example, to specify only the specific source path which should fall into the project:

1. From the **Source Control Filter** drop-down menu, choose **File Name.**
2. In the values field, specify the path to the associated source code.
   - All paths to source code that begin with the chosen File Name will be associated with your project.
   - You can use the percentage sign as a wildcard.
3. Click **Add**. A list of File Names is displayed.
4. From the list of File Names, choose the one you wish to associate with your project.
5. Click the < button to add this as a File Name associated with your project.
6. Click **Save** to complete the project configuration and save it.

If you want to apply this filter to your existing data, click **Recalculate**.

Code Review Filter

This filter determines what code review results are displayed in Report Center's Code Review reports for any specific project. The procedure for defining code review filters is similar to that of defining log filters (see "Log Filters", page 165).



A code review session tag is a custom string that is attached to code review results and used to distinguish different code review data packs. The session tag should be set to match the Session Tag from your Parasoft Test "Code Review" Test Configuration as described in "Configuring Code Review Reporting for a Project", page 267.

In addition to specifying session tags, you can also control restrictions as follows:

- With **No restrictions** selected: The Code Review report shows ALL code review results of the selected project team members (regardless of the project or source file for which the selected team members performed the code review).
- With **No restrictions** cleared (and session tags defined and set): The Code Review report shows all code review results marked with the selected session tag *as long as at least one project team member defined in the Development Testing Platform Project settings participates in the code review*. Code reviews of users who are not on the project's team list will not be shown, even if their code review data is present in results with the specified session tag. See "Configuring Code Review Reporting for a Project", page 267 for details.

Parasoft Test Settings

This page lets you specify the "localsettings" to be used when using a Parasoft Test product (C++test, dotTEST, SOAtest, Jtest) to run tests for the current project.

Instead of creating files for each of your Parasoft Test tools, you can specify your preferred settings once in this centralized location, and they will automatically be propagated to the Parasoft Test products when they connect to a Development Testing Platform project.

For example:

```
#Report Center Settings
grs.data.port=32323
#License Settings
license.network.host=dtp.company.com
license.network.port=2222
license.use_network=true
#Mail Settings
report.mail.domain=company.com
report.mail.from=john.doe
report.mail.password=123456789
report.mail.server=mail.company.com
report.mail.username=john
#Team Server Settings
tcm.server.accountLogin=true
tcm.server.enabled=true
tcm.server.name=dtp.company.com
tcm.server.password=123456789
tcm.server.port=18888
tcm.server.username=team_user
```

If the values listed below are not defined, Development Testing Platform automatically completes them. The values are completed as follows, based on the general configuration (such as in Data Collector port, Mail Server port, etc.):

```
#Report Center Settings
grs.data.port
#License Settings
license.network.host
license.network.port
#Mail Settings
report.mail.domain
report.mail.from
report.mail.password
```

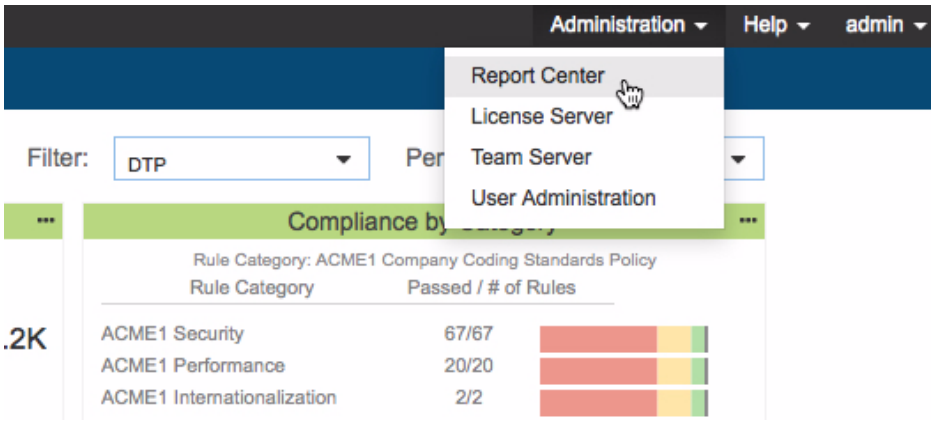
```
report.mail.server
report.mail.username
#Team Server Settings
tcm.server.name
tcm.server.port
```

The automatically-completed values will be overwritten when you specify them manually.

For more details, see the Parasoft Test User's Guide.

Report Center Administration Pages

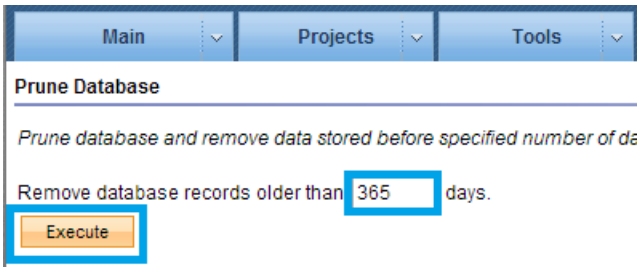
Users with administrator privileges can use Report Center Administration. This page provides access to administrative settings for projects, tools and reports. Choose Report Center from the **Administration** drop-down menu to enter Report Center Administration:



Pruning the GRS Database

Use the following command to remove obsolete or over-limit data as necessary:

1. Choose **Tools> Update Database> Prune Database (GRS DB)**.
2. Adjust the time period if necessary and click **Execute.**



Files older than the time period specified will be pruned from the database. You can also enable automatic database pruning; see “Automatically Pruning the Database”, page 180.

Uploading Reports to Data Collector

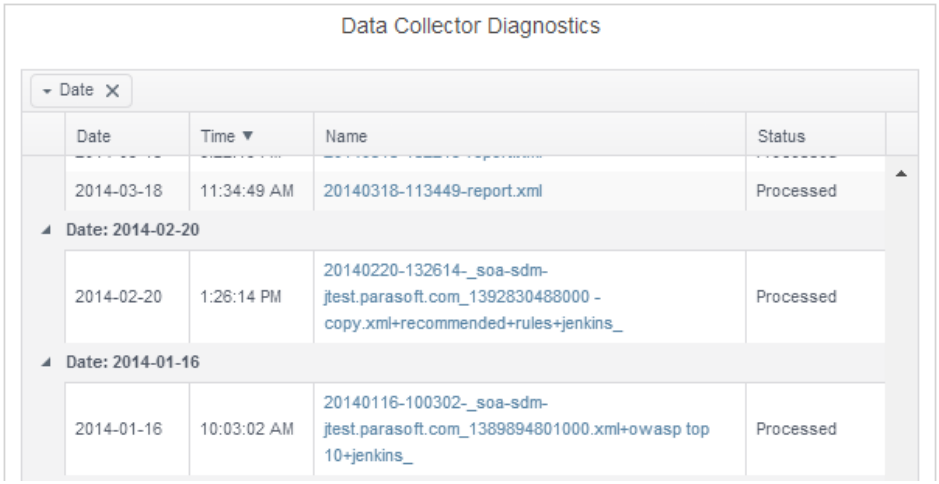
You can upload XML report files generated by Parasoft code analysis tools directly to DTP.

1. Choose **Tools> Data Collector Uploader Form**
2. Click **Choose File** and browse for the report you want to upload
3. Click **Upload**
4. Use the browser navigation buttons to return to Report Center Administration.

Checking Status of Reports Directly Uploaded to DTP

1. Open the Report Center dashboard view (see “Dashboards”, page 26)
2. Add the Data Collector Diagnostics widget to your dashboard (see “Adding Widgets”, page 15)

The widget displays information about data collected by DTP (see “Diagnostics Widgets”, page 65).

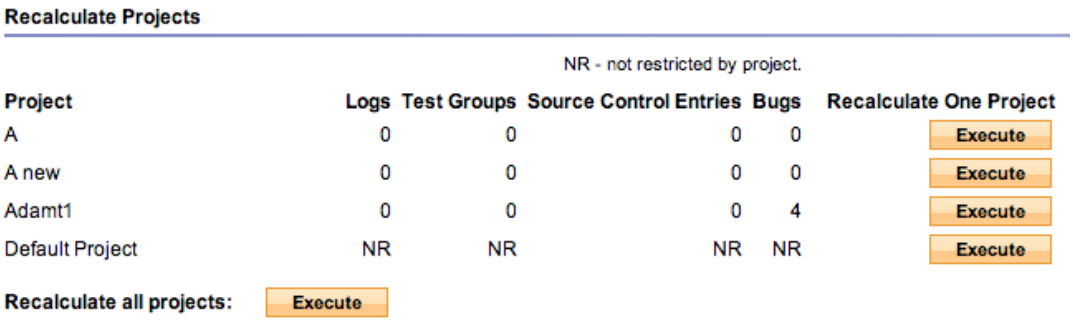


Check your report configuration settings if the widget returns a status error. See the Parasoft Test or Static Analysis Engine documentation for additional information.

Recalculating Project Data in GRS Database

The following command recalculates project logs, test groups, source control entries, and bugs:

1. Choose **Tools> Calculate (GRS DB)> Recalculate Projects**
2. Click **Execute** for individual projects or recalculate all project data at once



Recalculating Artifacts in GRS Database

You can recalculate test statuses of defects and enhancements. After executing this calculation, the updated status will display in individual defects and enhancements in Project Center. The test status is also used in the Project Center > Defect/Enhancement Status report.

1. Choose **Tools> Calculate (GRS DB)> Recalculate Defects/Requirements**
2. Click **Execute** in one of the following sections:
    - The **Defects/Enhancements test status quick calculation** option calculates status based on changes made since last calculation.
    - The **Defects/Enhancements test status full calculation** option calculates the full test status history.

Recalculating Run Jobs

1. Choose **Tools> Calculate (GRS DB)> Run Recalculation Jobs**
2. Click **Execute**

Automatically Run Scans and Calculations

By default, BTS Scanner runs in incremental mode every 15 minutes. Immediately following a scan, a quick calculation of test statuses is run for defects and enhancements. See "Running BTS Scanner", page 223 for more details.

You can modify BTS Scanner scan and calculation settings to meet your needs by editing `$DTP_HOME/grs/config/CronConfig.xml`. The following table describes the settings you can add to the `CronConfig.xml` file:

Defect/Enhancement Calculation Option	CronConfig.xml Setting
Full rescan of Defects/ Enhancements from Bug Tracking Systems.	`com.parasoft.grs.rserver.cronjobs.bts.BtsFullScannerJob`
Incremental rescan of Defects/Enhancements from Bug Tracking Systems.	`com.parasoft.grs.rserver.cronjobs.bts.BtsScannerJob`
Defects/Enhancements test status quick calculation.	`com.parasoft.grs.rserver.cronjobs.RequirementCalculationJob`
Defects/Enhancements test status full calculation.	`com.parasoft.grs.rserver.cronjobs.RequirementFullCalculationJob`

You can also run the BTS Scanner on-demand. See "Running BTS Scanner On Demand", page 224, for details.

Clearing Report Center Data Cache

Use the following command to remove all data stored in the data cache when users browse through Report Center reports:

1. Choose **Tools> Invalidate (GRS DB)> Invalidate Data Cache**
2. Click **Execute**

Invalidating File Restrictions

File restrictions map the information about files reported by Parasoft testing tools, such as failed coding standards tests, to specific file information imported by SourceScanner. Use the following command to invalidate current file restrictions stored in the Report Center database:

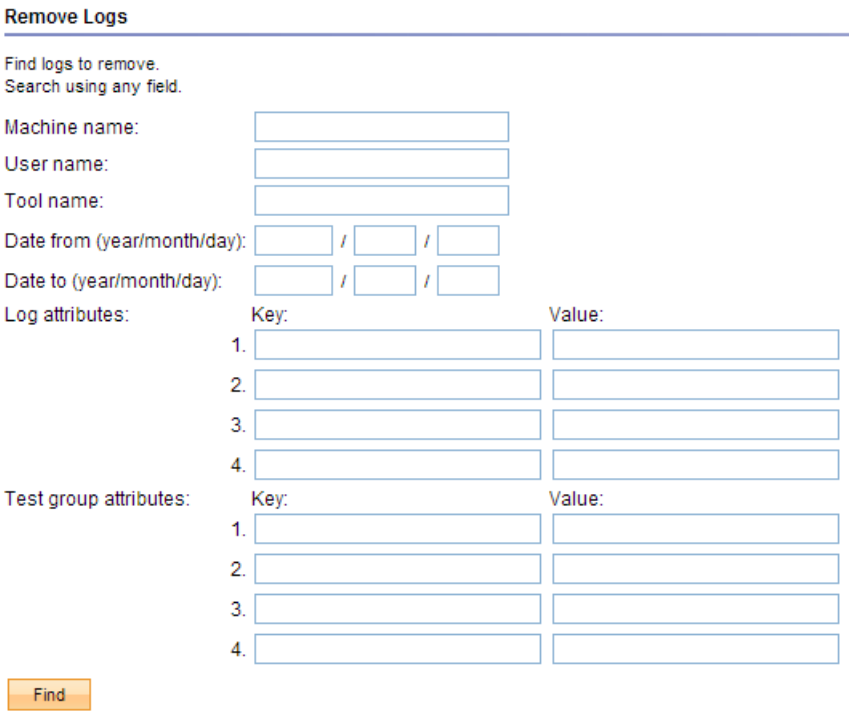
1. Choose **Tools> Invalidate (GRS DB)> Invalidate File Restrictions**
2. Click **Execute**.

This action can take several minutes.

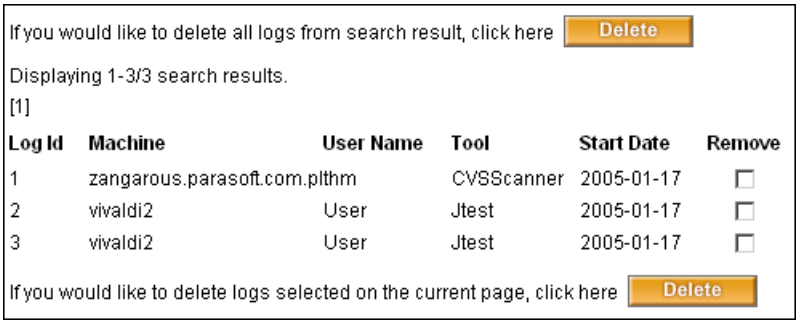
Deleting Logs

You can search for logs you want to remove from the database:

- Choose **Tools> Update Logs (GRS DB)> Remove Logs**.
- Enter search criteria (wildcards are not supported) and click **Find**.



3. Select a row in the search results and click **Delete** to remove a single log or click **Delete** as indicated in the user interface to remove all logs.

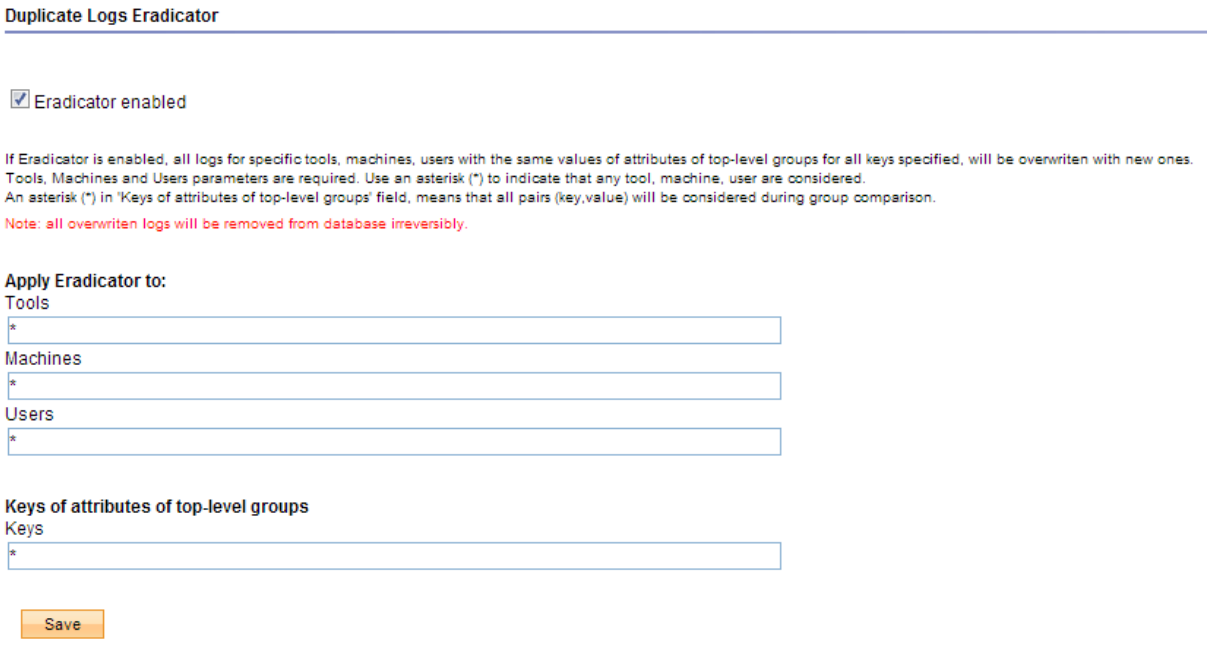


Logs cannot be retrieved once they've been removed.

Automatically Overwriting Logs

The Duplicate Logs Eradicator is an integral part of the Development Testing Platform Data Collector. It automatically overwrites previous logs to ensure that your database contains the most recent data. You must configure the main parameters (tool, user, and machine) and attribute keys.

1. Choose **Tools> Update Logs (GRS DB)> Duplicate Logs Eradicator**
2. Select the **Eradicator enabled** option
3. Enter parameters (see "About Eradicator Parameters", page 177) and attribute keys (see "About Eradicator Keys", page 177).
4. Click **Save** .



Logs cannot be retrieved once they've been removed.

About Eradicator Parameters

You can set the following parameters for the eradicator:

- **Tools:** Specify the tool(s) the eradicator must verify to prevent log duplication.
- **Machines:** Specify the machine(s) the eradicator must verify to prevent log duplication.
- **Users:** Specify the user(s) the eradicator must verify to prevent log duplication.

To specify multiple tools, machines, or users, separate items with a comma (e.g. Jtest, SOAtest, C++test). To specify all tools, machines, or users, insert an asterisk (*) in the appropriate fields.

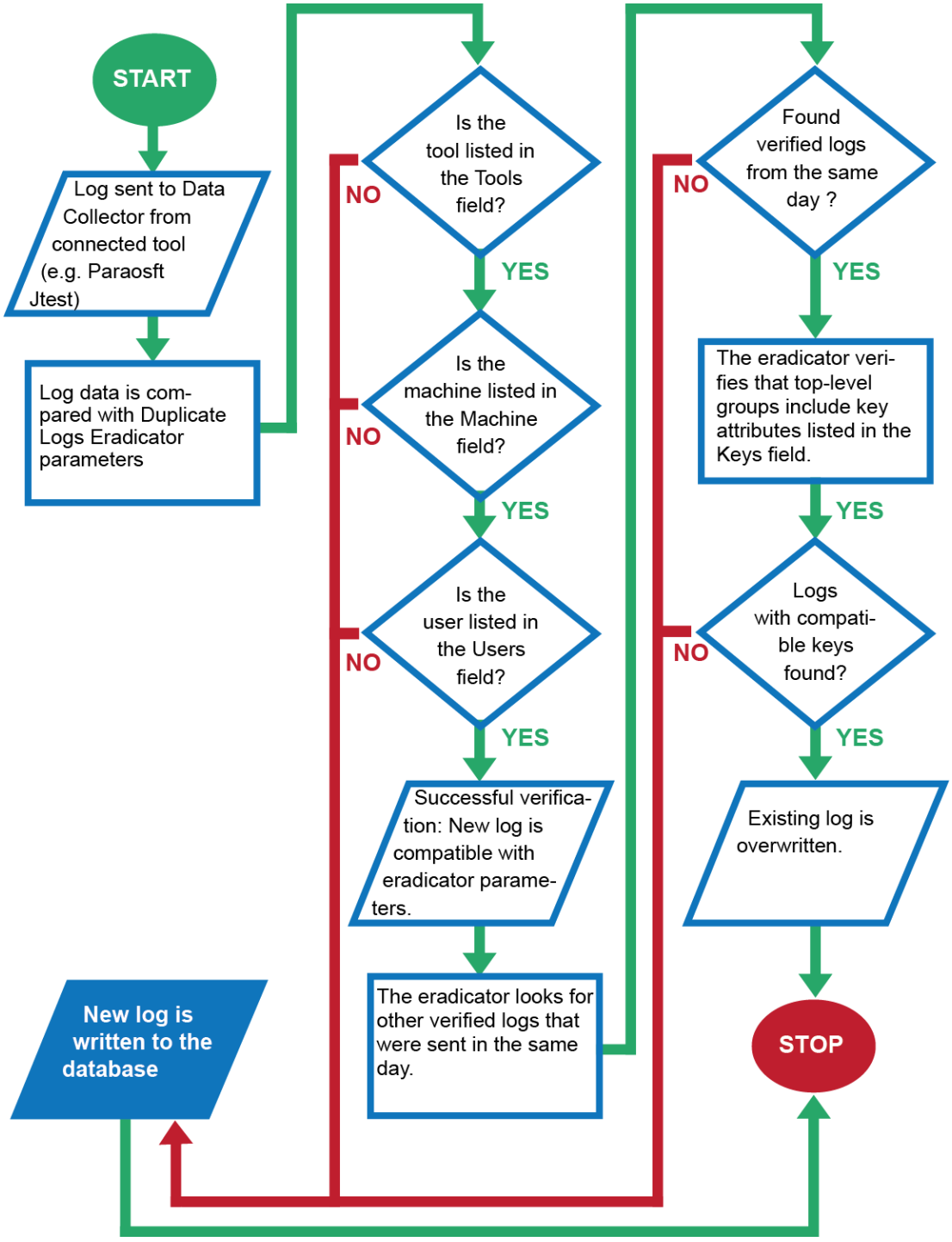
About Eradicator Keys

Enter a value into the keys field to specify the top-level group attributes the eradicator should use to compare logs and determine if they can be overwritten.

- If the field is blank, the test group keys are not taken into consideration when the new log report is received.
- If an asterisk (*) is used, all test groups key values are taken into consideration.
- "User-Attribute: Project" is the most common key. This key references the General Project field in the Parasoft Test GUI. The Duplicate Logs Eradicator will look for this key and overwrite logs for results containing the same value in the General Project field.
- You can specify multiple keys in a comma-separated list, e.g. "User-Attribute: Project, User-Attribute: TestMode".
- When specifying multiple keys, all keys must be present in the test results for the eradicator to remove existing logs.

About the Duplicate Logs Eradicator

When a new log is sent to Data Collector, the eradicator checks it against defined parameters, compares time-stamps, and performs other checks to determine if the log is should be overwritten to prevent duplicates. The following chart describes the Duplicate Logs Eradicator verification processes:



Scanning Code Review Results

You can force Report Center to read the latest code review scan results in Team Server:

1. Choose **Tools> Scan Code Review (GRS DB)**
2. Click **Execute**.

The command runs the `CRHistoryHarvestingJob` job, which reads code review scan results and feeds the data into the Report Center database. The command connects to Team Server using configurations defined in the `DTP_HOME/grs/config/CRHistoryScanners.xml` file.

Configuring Emails Notifications

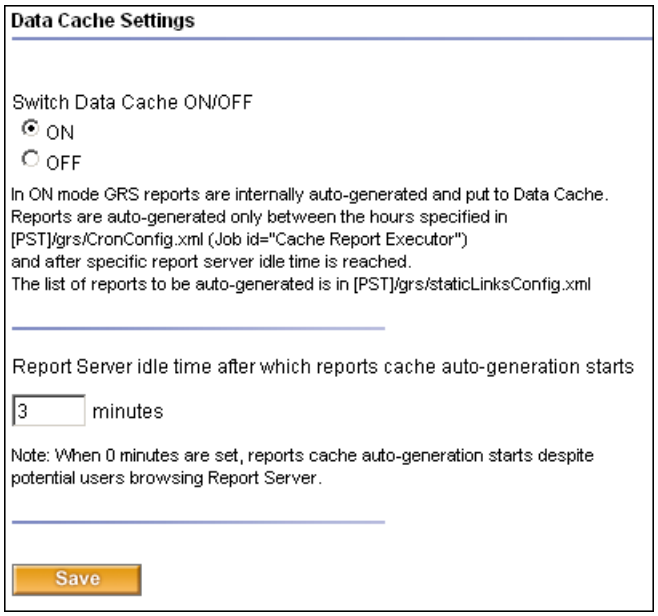
See "Task, Defect, and Requirements Email Notifications", page 226, for additional information about configuring setting for emailing notifications to team members.

1. Choose **Settings> E-mail**
2. Enter email settings
3. Click **Save**

Enabling/Disabling Report Center Data Cache

Data Cache stores data read from the database while you view reports.

1. Choose **Settings> Report Center (GRS DB)> Data Cache**
2. Choose the **on** or **off** option to enable or disable the cache
3. Enter a value in the idle time section to set how long the report server should wait before automatically generating reports
4. Click **Save**.



Data Cache Details

When you generate a Report Center report, the DTP GRS Server records and saves the following information to Reports Statistics:

- The number of times each report (with specified parameters) has been displayed.
- The amount of time it took to generate each specific report.

Report Statistics is used by Report Center to auto-generate selected reports. Report Center refers to the statistics to generate and store data in the cache that is either frequently used or requires considerable time to generate. The cache can significantly speed up users' work, but it can also slow down the server while reports are auto-generated. The administrator's task is to configure the data cache settings to achieve a balance that facilitates efficient activity. See “Clearing Report Center Data Cache”, page 174, for more information about emptying the cache.

Setting Data Collector Time Restrictions

You can specify a block of time during which data is temporarily stored on disk instead of being saved directly to the Report Center database. Setting Data Collector time restrictions ensures that retrieving and saving data does not significantly slow down Report Center.

1. Choose **Settings> Report Center (GRS DB)> Server**
2. Select the **Enable Data Collector Time Restrictions** option
3. Enter a start and end time
4. Click **Save**

Automatically Pruning the Database

You can configure Report Center to automatically remove data in the database once a week.

1. Choose **Settings> Report Center (GRS DB)> Server**
2. In the **Enable database pruning** section, select **Yes** or **No** to enable or disable automated pruning
3. Enter the maximum number of days data should be kept in the field provided
4. Click **Save**

Using Deprecated Team Server-based Code Review

You can switch to Team Server-based code review, which draws data from Team Server instead of the Report Center database. For more details, see “Integrating with Code Review”, page 267**.**

1. Choose **Settings> Report Center (GRS DB)> Server**
2. Enable the **Use deprecated Team Server based Code Review** option
3. Click **Save**

Configuring Manager Policy Center Email Settings

You can configure which team members receive comprehensive reports on all projects, set the range of data for the report, and specify which days the report should run and be emailed to specified users.

1. Choose **Settings> Policy Center> Manager Email**
2. Perform one of the following tasks:
    - Click in the Manager(s) field and choose users from the drop-down menu
    - Click **Show**, select users in the overlay, and click **Assign** to add them to the Manager(s) field
    - Click **Clear** to remove users from the Manager(s) field
3. Set the report range
4. Select the days of the week that the Manager Email report should be sent
5. Click **Save**

Integrating with Development Components

Report Center integrates with bug tracking systems, requirement management systems, and other third-party software development tools that provide visibility into the development process. Integration with bug tracking systems, requirement management systems, and Emma code coverage tool for Java are discussed in the following sections:

- Integrating Report Center with Emma
- Integrating with Bug Tracking Systems and Requirement Management Systems
- Integrating with Custom Processors

See “Integrations”, page 208, for information about all Development Testing Platform integration capabilities.

Connecting Parasoft Test

Parasoft Test is the infrastructure that facilitates the configuration, usage and interoperation of Parasoft’s family of development testing technologies. You must configure Development Testing Platform to connect to Parasoft Test to import and use project settings defined Parasoft products that use Parasoft Test (e.g. C++test, Jtest, dotTEST, SOAtest, etc.).

1. Choose **Settings> Parasoft Test Global Settings**
2. In the space provided, enter the configuration settings from Parasoft Test (see documentation for your Parasoft Test-based tools)
3. Click **Save**

About File Encoding

By default, DTP assumes that source code is encoded in UTF-8. If the source code is encoded with a different character set, you can specify it in the Parasoft Test Settings form by adding the `file.encoding.name` property. See “Parasoft Test Settings”, page 170, for additional information.

Viewing Report Center Administration Reports

The administration includes several reports for understanding Report Center activity.

Data Collector Activity Report

You can see when Data Collector retrieves data to be stored in the database and when developers retrieve data from the same database. Understanding the difference between data is stored and when it’s needed enables you optimize the data collection and storage process so that you can prevent bottlenecks and improve efficiency associated with the movement of data.

1. Choose **Reports> Data Collector Activity (GRS DB)**
2. Choose a preset time period or click the calendar icon to select a custom range
3. Click **Refresh** to update the report without reloading the page

Last 30 days (SUM)

h\min	05	10	15	20	25	30	35	40	45	50	55	60	Total
0	43	44	62	53	54	49	48	129	110	93	24	27	736
1	27	34	34	48	96	101	20	22	25	51	98	100	656
2	268	354	205	598	574	359	311	356	198	233	315	282	4053
3	200	335	230	209	257	171	255	297	163	84	67	83	2351
4	124	82	95	29	25	33	9	9	11	8	11	6	442
5	29	14	5	6	7	13	10	4	45	82	72	67	354
6	127	35	37	71	16	84	25	6	5	5	1	10	422
7	8	3		4	4	5	3	2	2	5	3		39
8	2	1	1		1	3	2	2	1	1		3	17
9	1	3	5	6	4	10	5	6	17	21	19	12	109
10	15	6	3	16	11	10	12	4	8	5	13	8	111
11	14	21	26	17	19	17	4	23	9	18	12	13	193
12	7	12	8	10	15	16	16	9	13	6	8	4	124
13	11	14	6	9	6	10	17	16	8	12	13	13	135
14	19	11	14	9	8	8	18	21	13	16	5	8	150
15	10	48	30	16	11	10	16	16	25	17	7	1	207
16	14	5	5	10	5	5	5	6	13	49	36	22	175
17	26	21	20	19	19	12	5	7	8	5	2	3	147
18	3	1	2	3	4	1		1	1		2	2	20
19	1	1	2	1		13	58	20	16	53	22	84	271
20	170	77	124	120	85	58	85	57	58	76	58	41	1009
21	445	424	223	184	212	176	80	82	97	97	94	74	2188
22	82	71	70	78	73	72	80	69	66	62	76	75	874
23	94	68	58	43	48	54	48	51	47	33	75	24	643
													15426

Database Stats Report

Choose **Reports> Database Stats (GRS DB)** to view all tables that in the Report Center database and how many rows each table contains. If Report Center is running slowly, you can look at this report to check the sizes of all the tables.

Database Statistics

Table Name	# Rows
ACTIVITY	550752
BUG	49796
BUG_ACTIVITY	172923
BUG_ATTR	98950
BUG_ATTR_KEY	2
BUG_ATTR_TRIO	98950
BUG_ATTR_VALUE	349143
COD_FILE	98039
COD_LOG	11349
COD_MESSAGE	24629537

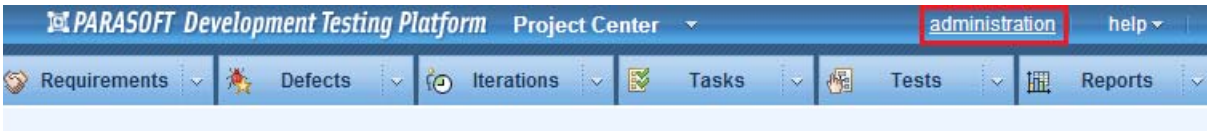
Project Center Administration Pages

A Project Center Administration web page is accessible to users with administrator privileges. This page provides access to configurations for items such as task notifications, custom statuses and fields, e-signature requirements, and more. This chapter describes the following functions available in Project Center administration:

- BTS and RMS Scanner Configuration
- Entering SOAtest Server Settings

Accessing Project Center Administration Page

1. Click **administration** from the Project Center main navigation bar to access the administration page.



BTS and RMS Scanner Configuration

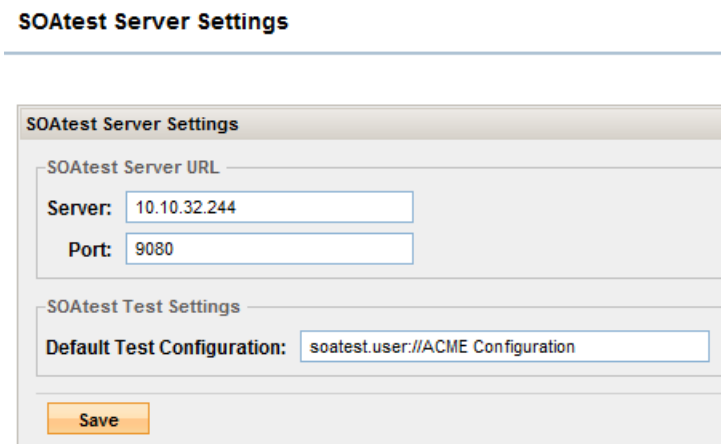
Development Testing Platform can integrate with bug tracking and requirement management systems either through the graphical user interface or by configuring the integration XML files stored in the $DTP_HOME/grs/config/bts directory. See “Integrating with Bug Tracking Systems and Requirement Management Systems”, page 215 for more information.

Entering SOAtest Server Settings

If you want to enable the team to run SOAtest .tst files from the Project Center interface (via the SOAtest Server web service), enter SOAtest server settings in this page as follows. Please note that **Parasoft Development Testing Platform does not currently support accessing SOAtest server when access controls are required.** If Development Testing Platform needs to access SOAtest server, it will need to use a SOAtest server that has access controls disabled.:

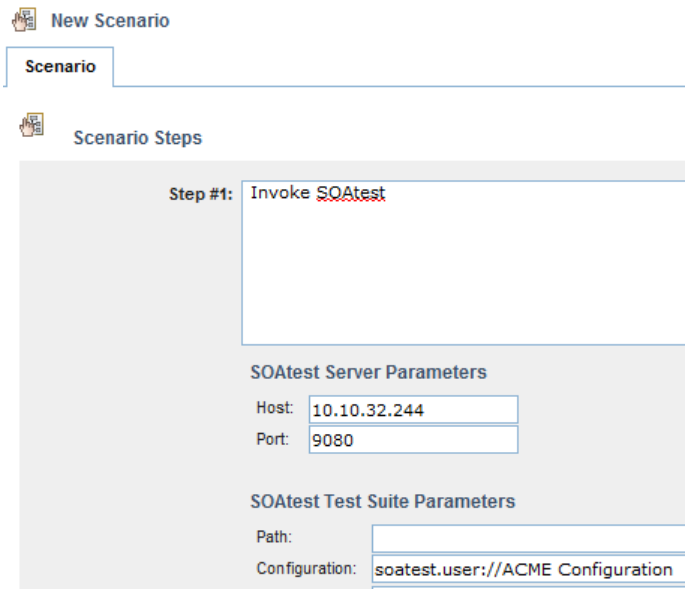
1. Under **SOAtest Server URL**, enter the server and port for the SOAtest server web service interface.

2. Under **SOAtest Test Settings**, specify the Test Configuration that you want set as the default for SOAtest tests run from the Project Center interface.



3. Click **Save**.

Note that these settings will be set as the default for SOAtest test types. If desired, they can be altered at the test level (for instance, if a specific test requires a different Test Configuration).



Optional Report Center Configurations

Configuring Report Center to Work with Source Code Branches/Tags

Several Report Center reports show statistics associated with development project source files. You can define the source repository files that should be scanned for data retrieval, along with their folders, naming patterns, and so on.

For instances when a project is developed in a source repository branch, or when you want to see statistics of source files from a specific tag, Report Center can be configured to work with source repository branches and tags.

Following are the tasks to perform to configure Report Center to work with branches/tags:

- Configuring SourceScanner to Scan Specific Branches/Tags
- Configuring Report Center Projects: Assigning Specific Branch/Tag to a Project
- Viewing Source Statistics on Report Center Reports

Configuring SourceScanner to Scan Specific Branches/Tags

When SourceScanner runs, it scans specified CVS files (branch/tag), and then sends that data to Report Center. In order to be able to view the source code changes as necessary in Report Center, SourceScanner needs to be configured so that it scans the appropriate branch or tag, for example, my_project_6_0_branch.

For details about how to accomplish this task, see *Parasoft SDLC Integration Extensions User's Guide*.

Important! *It is strongly recommended that any one specific source repository location be scanned by only ONE SourceScanner project. For instance, if you intend to scan the sources for two branches and trunk of the same files, then you should define it in only one SourceScanner project. Do not define and run three separate projects.*

Note About Parasoft SDLC Extensions

SourceScanner does not ship with Development Testing Platform. It is part of the Parasoft SDLC Extensions module. Contact your Parasoft representative if you would liike to use Parasoft SDLC Extensions.

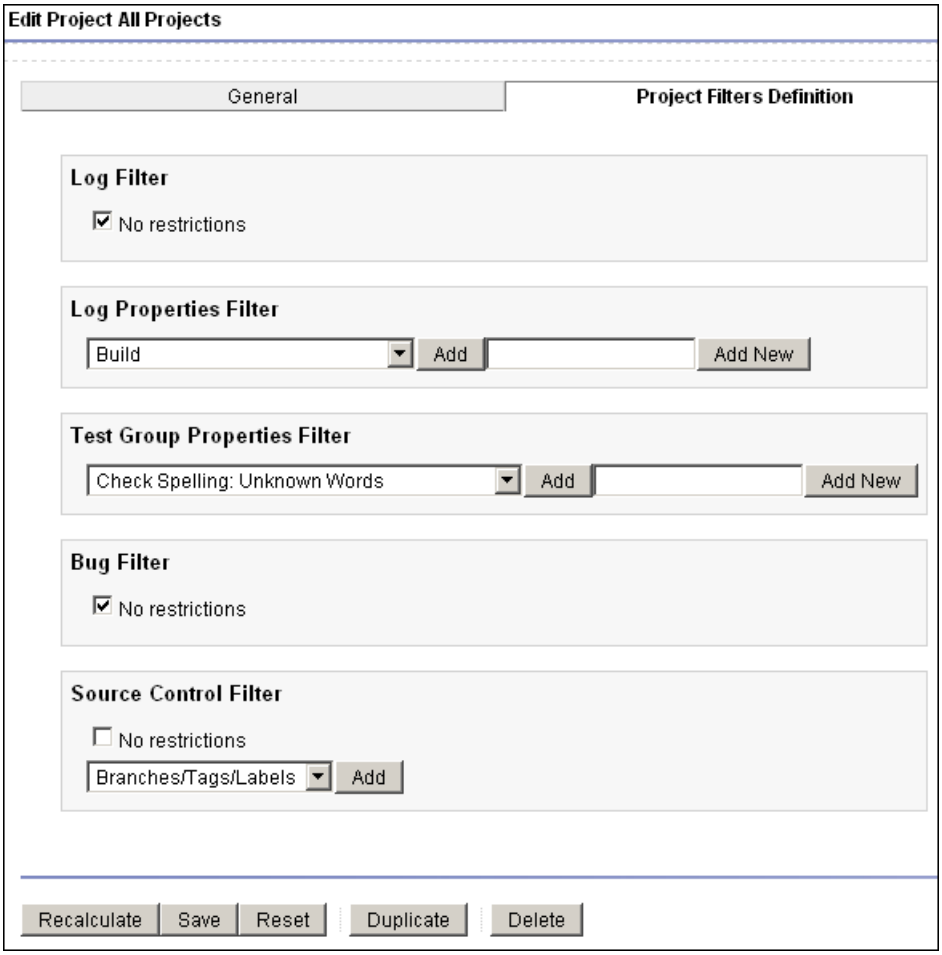
Configuring Report Center Projects: Assigning Specific Branch/Tag to a Project

Now that your Report Center project is configured, you must specify which source files scanned by SourceScanner should be shown in your specified Report Center project. To do so, follow these steps:

1. From the Report Center Admin menu, choose **Project> Search**.

   Find the appropriate project or create a new one.

2. Select the **Project Filters Definition** tab and go to the **Source Control Filter** section:

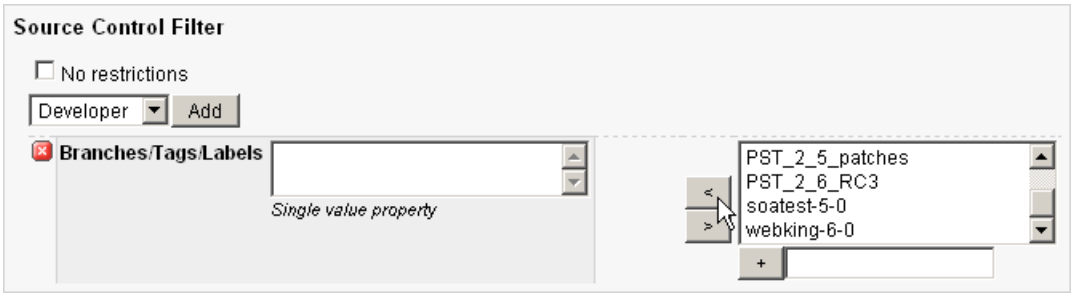


3. In the Source Control Filter section, specify any restrictions.

    a. Select the **Branches/Tags/Labels** from the drop-down box, and then click **Add**:

    The window on the right lists the names of all branches/tags that have been scanned by SourceScanner so far.

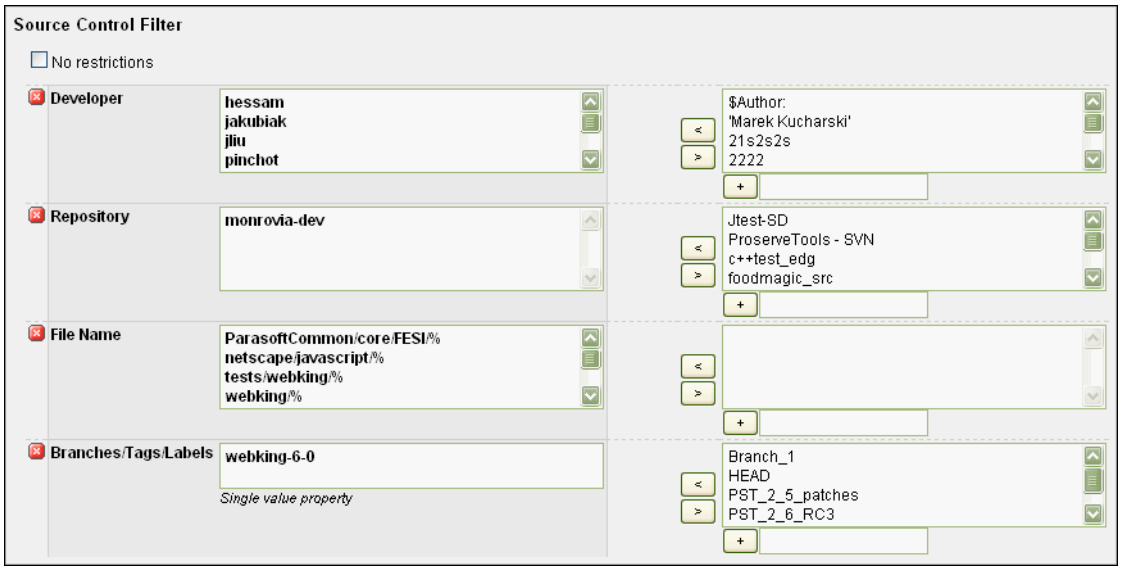
b. Select the appropriate branch/tag/label—continuing with the example, let's select webking-6-0—and then, click the Less Than (<) symbol:



The branch is marked as a part of the Source Control Filter of the Report Center project that was just modified.

c. *(Optional)* If you have any restrictions that need to be set for **Developer**, **File Name**, or **Repository**, perform steps similar to Step a. and Step b.

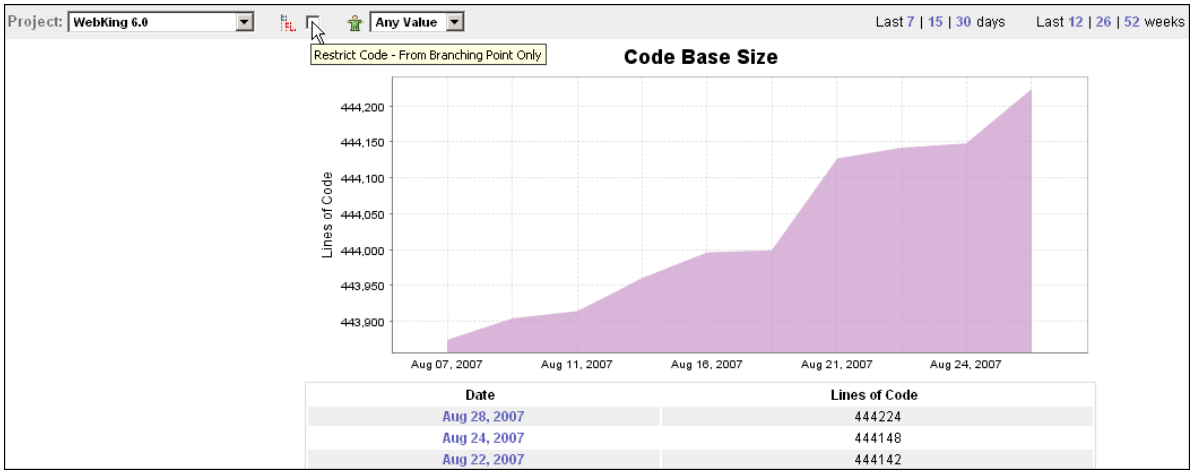
Note: *Restrictions for developer, file name and repository are usually set when defining the Source Control Filter for your Report Center project.*



4. Save the configuration, and then recalculate the project.

Viewing Source Statistics on Report Center Reports

To view the statistics of the branch sources for which you configured SourceScanner and Report Center, go to any Report Center report that displays source files data. Code Base Size (shown below) and Check-ins are two examples of such Report Center reports.



Notice the **Restrict Code – From Branching Point Only** option near the top of the page:

- If **Restrict Code - From Branching Point Only** is selected, then only the file revisions that were committed after the date on which the branching point occurred are taken into account when calculating the specific report.

  The branching point occurs when a branch is created in a revision. In other words, the branch is separated from the main trunk/head or its parent branch.

- If **Restrict Code - From Branching Point Only** is not selected, then all file revisions are taken into account to calculate the specific report.

Configuring Cache Report Executor

The Cache Report Executor job browses reports defined in the staticLinksConfig.xml configuration file, thus making them cached. By default, this job is scheduled to start at 5 a.m. (allowFromHour=5). If the default settings do no accommodate your needs, you can modify the Cache Report Executor job to change scheduled cache times and the list of reports to cache.

If you encounter any difficulties with the Cache Report Executor, see “Cache Report Executor” on page 305.

Configuring Scheduled Cache Times

You can configure the scheduled cache time in the CronConfig.xml file in the `Job id="Cache Report Executor"` xml element.

Configuring List of Reports to Cache

To configure the list of reports to cache when the Cache Report Executor job runs, edit the file in staticLinksConfig.xml. Following is an example of the configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<reports>
    <report descr="Architect Dashboard" xreport-id="architect_dashboard/
composite_desc" composite="true" params=""/>
    <report descr="Practices > Coding Standards" xreport-id="practices/
cs_composite_desc" composite="true" params=""/>
    <report descr="Practices > White Box" xreport-id="practices/wb-bb/
wb_composite_desc" composite="true" params=""/>
    <report descr="Practices > White Box > Coverage" xreport-id="practices/
wb-bb/WBCoverageDetails" composite="false" params="period=10drops&amp;date-
Mode=period"/>
    <report descr="Practices > Black Box" xreport-id="practices/wb-bb/
bb_composite_desc" composite="true" params=""/>
    <report descr="Practices > Black Box > Coverage" xreport-id="practices/
wb-bb/BBCoverageDetails" composite="false" params="period=10drops&amp;date-
Mode=period"/>
    <report descr="Audit > Drop Grade" xreport-id="audit/DropGrade" compos-
ite="false" params="period=10drops&amp;dateMode=period"/>
    <report descr="Audit > Errors By Category" xreport-id="errors/ErrorsBy-
Category" composite="false" params="period=10drops&amp;dateMode=period"/>
    <report descr="Audit > Errors By Severity" xreport-id="errors/ErrorsBy-
Severity" composite="false" params="period=10drops&amp;dateMode=period"/>
</reports>
```

The following table lists some of the tags contained in the file above, along with descriptions of each:

Tag	Description
reports	Root element.

Tag	Description
report	Tag that contains information for single report to cache.
descr	Simple description of report to cache.
xreport-id	Contains xreport descriptor, which identifies report to cache.
composite	If report is a composite report, then set this attribute to `true`. If not, set it to `false`.
params	Attribute that contains report parameters used to cache report. Multiple parameters should be concatenate by the an ampersand symbol: `&`.

Adding a New Report to Cache

An xreport-id attribute can be set by copying it from the appropriate report URL. To explain this more clearly, the following steps contain the architect dashboard as an example:

1. Go to Report Center web page: http://localhost/grs.
2. Go to the report that you want to be cached, such as:

   ```
   http://localhost/grs/
   xarchitect_dashboard.jsp?xreportcomposite=architect_dashboard/
   composite_desc.
   ```

3. Copy the link. In this example it is `architect_dashboard/composite_desc`, and then paste it as `xreport-id` attribute value.
4. Verify that the composite attribute is set to `true`.

Working with the Reports Caching Mechanism in Report Center

Reports in Report Center are cached in two different ways:

- When users browse and then generate a report, it is stored in cache.
- Each night the Report Center background job is run, which generates cache for reports specified in the dtp\grs\config\staticLinksConfig.xml configuration file.

Reports cache can be invalidated in three different ways:

- Nightly (after midnight) run of the Report Center background job. The invocation time can be configured in dtp\grs\config\CronConfig.xml, in CleanCacheReportExecutor part. See “Customizing Reports Caching” for more details about configuring nightly jobs.
- Manually from the Report Center Administration menu: Settings> Report Center> Data Cache.
- Restarting the Development Testing Platform server.

Notes:

- *When a specific report is cached and a new log from the Parasoft tool is received by Data Collector, that report cache is not invalidate. The report is shown from cache, so in order to see*

the new log on the report, the cache should be invalidated from the menu. Otherwise, it will be done automatically after midnight.

- *Reports cache data is stored on disk in the dtp/reportcenter/datacache directory. For each report with a specific parameter, there are two different files stored there: one with .dat extension and the other with .par extension.*

Customizing Reports Caching

There are two locations where the caching mechanism can be customized:

- dtp/grs/config/CronConfig.xml configuration file contains Report Center background jobs configuration.

  The following two jobs deal with caching:

  - CacheReportExecutor: Background job that browses the reports defined in the dtp\reportcenter\staticLinksConfig.xml configuration file, thus making them cached. By default, this job run at 5 a.m. each morning.
  - CleanCacheReportExecutor: Job that invalidates reports cache. By default, it runs just after midnight each night at 12:01 a.m.
- From the Report Center administration page: Settings> Report Center> Data Cache
  1. Switch caching on/off.
  2. Report Server idle time (in minutes) after which reports cache auto-generation CacheReportExecutor continues to work after being paused.

CacheReportExecutor starts at the hour defined in CronConfig.xml (allowFromHour=5 a.m., by default) and works until any of the following circumstances occur:

- All reports from staticLinksConfig.xml are cached.
- Any user begins to browse Report Center reports during CacheReportExecutor work, CacheReportExecutor pauses to work, and continues when "Report Server idle time" elapses after all users stop browsing Report Center reports.
- CacheReportExecutor reaches `allowToHour`, which is defined in CronConfig.xml. (CacheReportExecutor stops here if it did not stop earlier when all reports from staticLinksConfig.xml were cached).

Report Center Tools

Report Center tools include sizer (which helps you monitor the size of the Report Center database), Report Center testing tools such as chk and send, and additional utilities that will help you install the AEP infrastructure and troubleshoot problems.

Extracting and Customizing the Scripts

To use the Report Center tools, untar the distribution (SDLCExtensions--{version}.tgz or SDLCExtensions--{version}.exe) to some directory on your system. For example:

Unix: `$HOME/proserve`

Windows: `C:\Program Files\Parasoft\ProserveTools`

The following shell scripts are included:

UNIX scripts:

- chk.sh checks the data in Report Center
- send.sh sends test message to Report Center
- size.sh checks size of Report Center and stores it in Report Center

Windows scripts:

- chk.cmd checks the data in Report Center
- send.cmd sends test message to Report Center
- size.cmd checks size of Report Center and stores it in Report Center

Note: *These scripts typically must be modified before they run properly. You usually need to modify them so that their path information specific to the system where they are installed. The scripts are designed only to be run from the local directory, but can easily be modified to run from your path.*

UNIX Script Customizations

Edit the shell script of your choice. It is likely that you will need to alter JAVA_HOME to point to the location where your java is installed.

For bash users: `export JAVA_HOME=/usr/local/java`

For tcsh users: `setenv JAVA_HOME/usr/local/java`

If you want to run from your path, you will need to comment out the line that says "set GTDIR" and make a new line that sets GTDIR to the full path of your installation directory:

```
#export GTDIR=`pwd`
export GTDIR=$HOME/proserve/ReportCentertools
```

Windows Script Customizations

Note: *If you install in the* C:\Program Files\Parasoft\ProserveTools *and have Report Center installed on the same system, you won't need to make any changes.*

Edit the cmd file of your choice. It is likely that you will need to alter JAVA_HOME to point to the location where your java is installed:

```
set JAVA_HOME=C:\j2sdk5.0
```

If you want to run from your path, you will need to comment out the line that says "For /F "tokens=*" and make a new line that sets GTDIR to the full path of your installation directory.

```
REM For /F "tokens=*" %%v in ('CD') Do @Set GTDIR=%%v
set GTDIR=C:\Program Files\Parasoft\ProserveTools
```

Running the Tools

Running chk.sh | chk.cmd (v2.1)

There are several ways to run the chk tool that checks the data in the database. The default is only to look at the current day and one day previous. The basic mode to test is type the command and name of the Report Center server:

Unix:

```
./chk.sh reportcenter_server_name
```

Windows:

```
chk reportcenter_server_name
```

To check the source scanner data instead, just add the -scanner parameter:

Unix:

```
./chk.sh reportcenter_server_name -scanner
```

Windows:

```
chk reportcenter_server_name -scanner
```

To dump the basic Report Center data to a csv file in the local dir:

Unix:

```
./chk.sh reportcenter_server_name -dump -all
```

Windows:

```
chk reportcenter_server_name -dump -all
```

Note for Oracle users:

If your Report Center server is using an Oracle db instead of the default MySQL server, add "-oracle" to any of the above commands. For example:

```
chk reportcenter_server_name -oracle
```

Running send.sh | send.cmd (v1.2)

To run the send tool that sends data to Data Collector, type the command and the name of the Report Center server to which you want it to send data:

Unix:

```
./send.sh reportcenter_server_name
```

Windows:

```
send reportcenter_server_name
```

Running size.sh | size.cmd (v2.1.1)

To run the size tool that checks the size of the Report Center database, type the command and the name of the Report Center server:

Unix:

```
./size.sh reportcenter_server_name
```

or for oracle

```
./size.sh -oracle reportcenter_server_name
        to do local size check use "-f"
./size.sh -f reportcenter_server_name
```

Windows:

```
size reportcenter_server_name
```

or for oracle

```
size -oracle reportcenter_server_name
        to do local size check use "-f"
size -f reportcenter_server_name
```

Creating and Locking Down Sandboxes

As utilization of Team Server grows to include sharing data of data and uploading reports, it is now imperative to provide a clearer delineation between data sets.

Using sandboxes on Team Server (organized as directories to the file structure) allows for logical separation of data for different development groups within the organization. It also provides a scalable way to control permissions as well as isolating backup and restore procedures.

Built-in corporate default configurations are stored in the default area as templates to allow deriving custom team sandboxes by the team's leads. Default tool installations have no username/password set in the Team Server properties of the tool. Therefore, by default, it will have read access to the corporate default configurations, but will not have the ability to alter them or upload results to the default area.

Administering Team Server

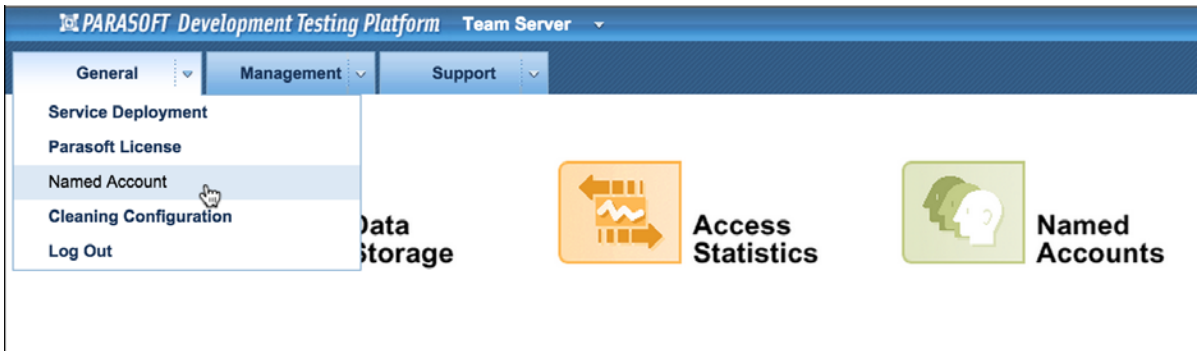
To administer Team Server, complete the following tasks:

- Lock Down Team Server and Create Admin Account
- Creating New Sandboxes
- Loading the Default Configurations into a Team's Sandbox

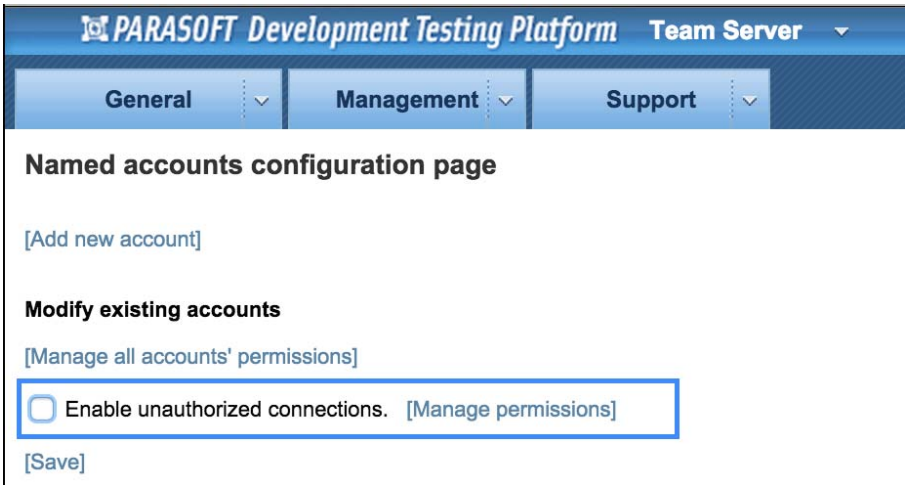
Lock Down Team Server and Create Admin Account

Lock Down Team Server and create the admin account:

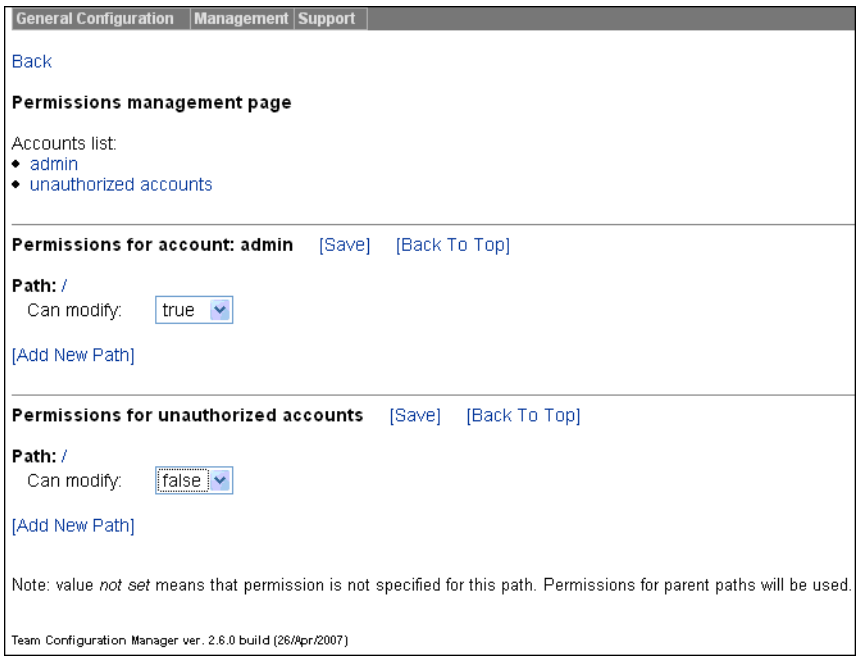
1. Go to the Team Server Home Page and choose **General> Named Account** .



2. Deselect **Enable unauthorized connections**, and then click **[Save]**.



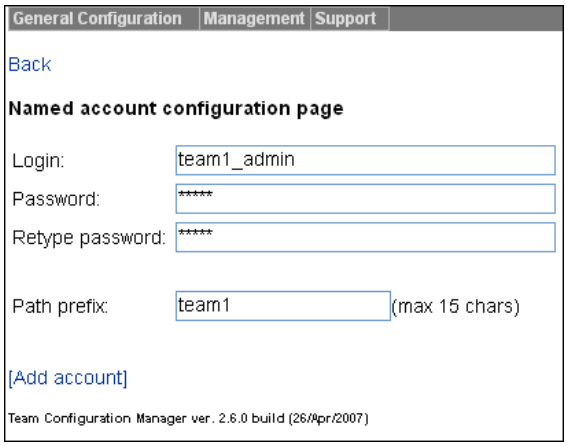
3. Click **[Add new account]** to add a global admin account.
4. Type the appropriate information in the following fields as specified, and then click **[Add account]**:
    - Username: Type "admin".
    - Password: Type "password".
    - Path prefix: Leave blank.
5. Validate access to the root file system.
    a. Choose **Management > Data Storage** from the menu.
    b. Click **Manage path permissions** to confirm the following settings for the admin and unauthorized users:



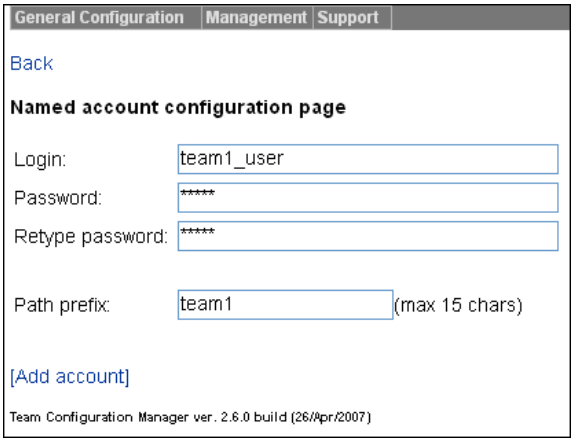
Creating New Sandboxes

When a new project needs to be created, the team lead is required to contact the Team Server Administrator and they should follow the steps outlined below (e.g. for 'team1'):

1. Go to the Team Server Home Page, and select **Named Accounts**.
2. Add the following users by clicking **[Add new account]**:
    - Team account: admin account
        - Username: team1_admin
        - Path Prefix: team1
        - Select **[Add account]**



    - Team account: user account
        - Username: team1_user
        - Path Prefix: team1
        - Select **[Add account]**

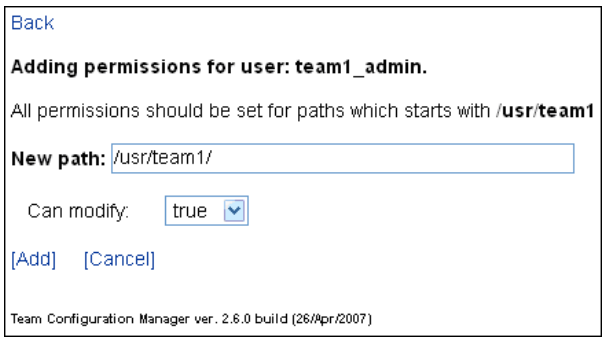


3. Setup team account permissions:
    - For the team admin account (team1_admin in this example):

a. Click **Manage**.

a. In **Path:/**, change **can modify** to **false**.

2. Select **[Add New Path]**, and then enter the default path (for example, /usr/team1).
3. Set **can modify** to **true**, and then **click [Add]**.
4. Click **[Save]** to save all of the user settings.



- For the team user account, (team1_user in this example)

a. Click **Manage**.

a. In **Path:/**, change **can modify** to **false**.

b. Click **[Save]** to save all the user settings.

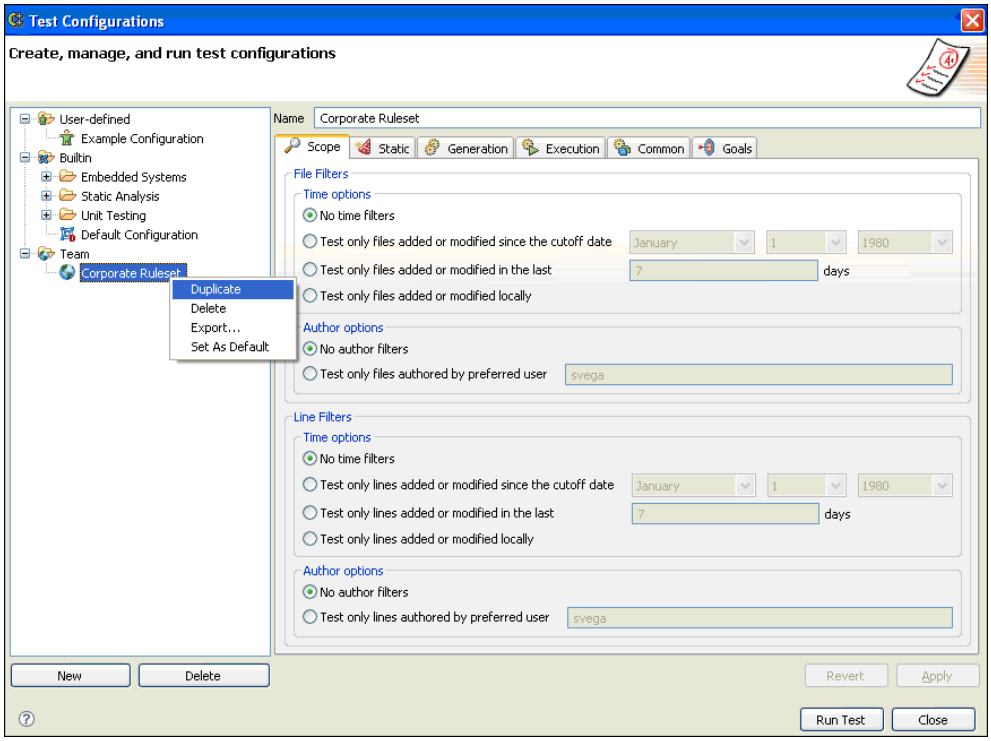
Loading the Default Configurations into a Team's Sandbox

Since the sandbox is created with no configurations loaded, the team lead or Team Server administrator is required to load the default configurations by doing the following:

1. Start up the appropriate tool, such as Jtest.
2. Load Test Configuration by choosing **Jtest > Test Configurations** from the menu.
3. Select the appropriate configuration from the **Team** folder.
4. Right click on the configuration and select **Duplicate**.

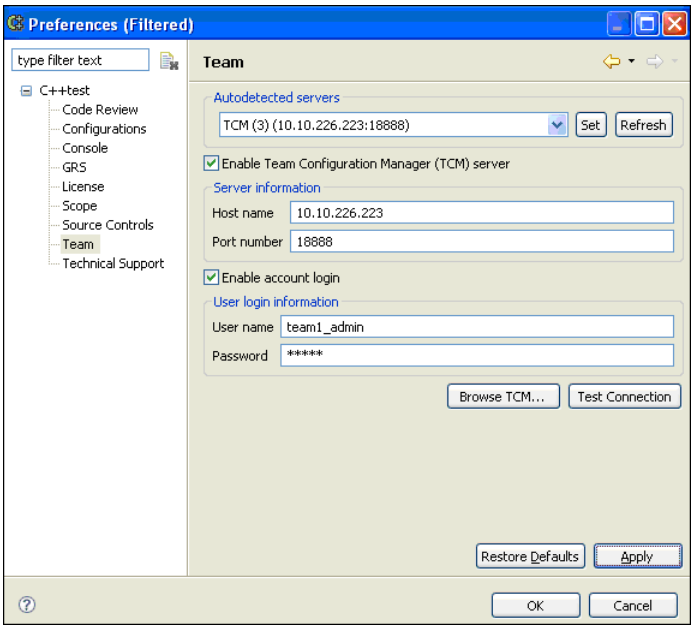
   A copy of the configuration is placed in your **user-defined** folder.

5. Rename the copied **user-defined** configuration as appropriate.



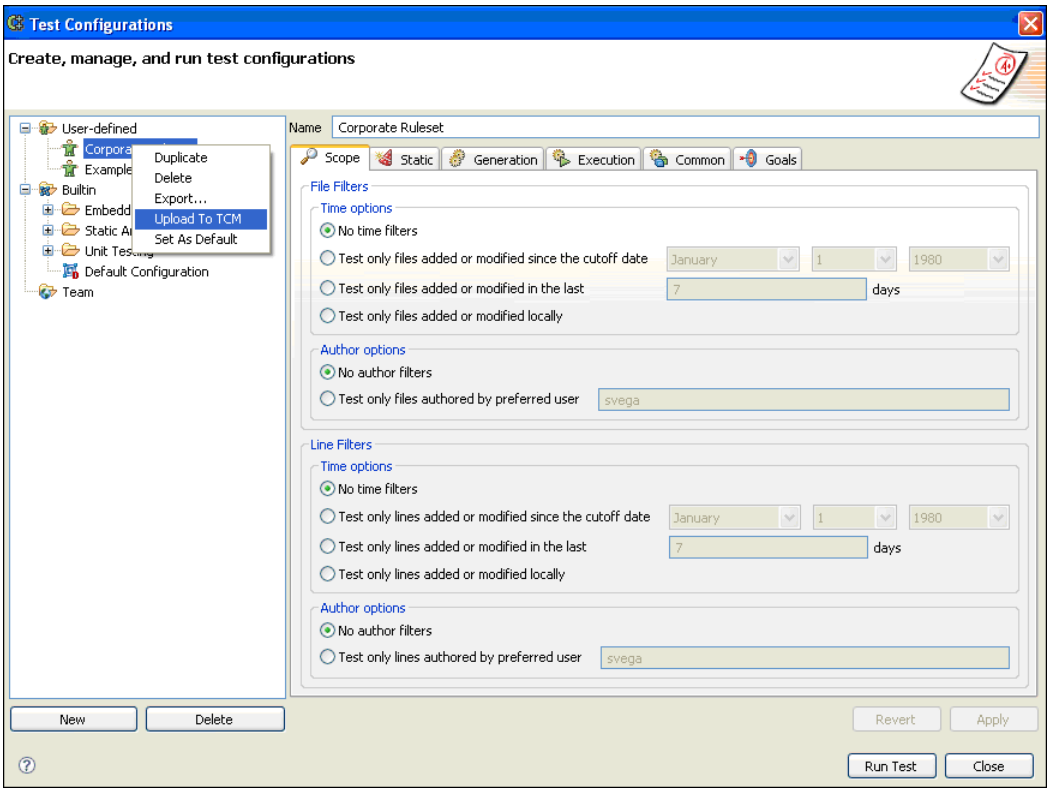
6. Close the Configuration dialog and load the **Preferences** dialog by using the tool's pull-down menu (for example, **Jtest > Preferences**).
7. Select the **Team** item.
8. Check the **Enable Team Server** and **Enable account login** check boxes.
9. Enter the team admin account supplied by Team Server administrator.

10. Click **Test connection** to assure the Parasoft Application has successfully connected to the Team Server manager.



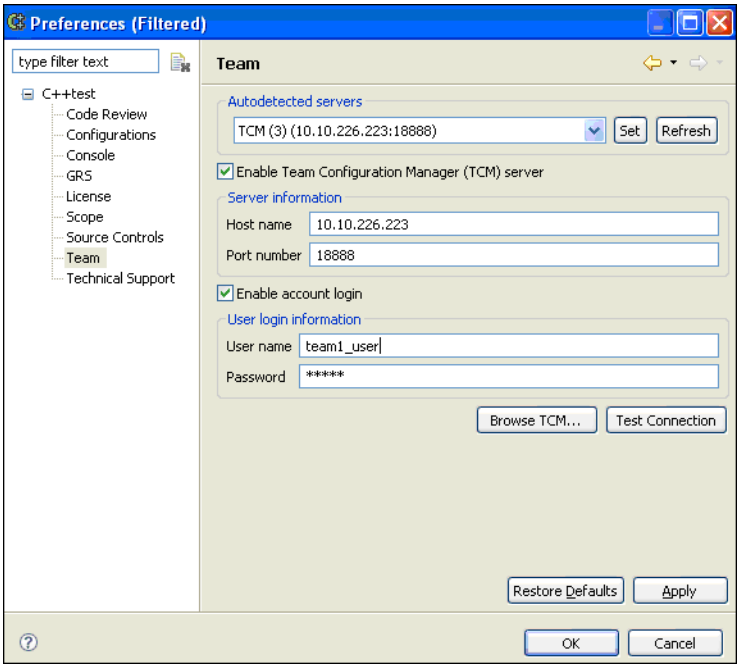
11. Reload the **Test Configuration** dialog.
12. Right click on defined **user configuration** and select **Upload to Team Server**.

    A copy of the configuration entry created will appear under the **Team** folder.



Important!

- *Developers on the team should be using the* **team user** *account to connect to the Team Server supplied by either the team lead or Team Server administrator.*



- *The command line/server execution of the appropriate tool (such as, Jtest) should be using the team admin logon details in the local settings to ensure that the data from the run is uploaded to the correct sandbox.*
- *The Team Server administrator should setup a backup process for the data stored on the Team Server.*
- **Cleaning Configuration** *on the Team Server (***Team Server Home Page > General Configuration > Cleaning Configuration***) should be set to every 30 days. The time of day for cleaning can be customized as appropriate. Be sure to click* **Save** *after every modification.*

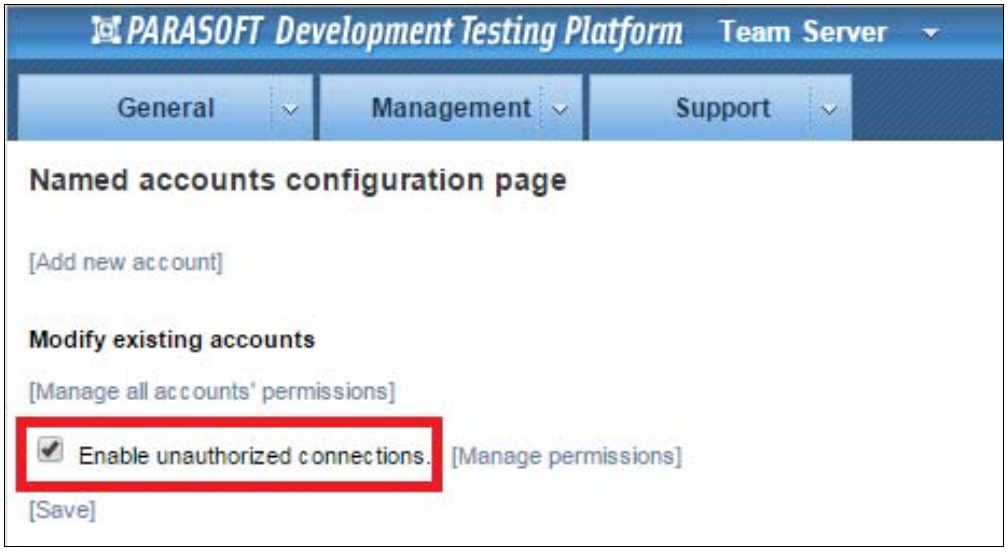
Forwarding DTP Engines 10.x Reports From Data Collector to Team Server

You can configure DTP so that static analysis reports from DTP Engines 10.x are forwarded to Team Server. This enables users to download static analysis violations detected by DTP Engines to a desktop IDE version of Parasoft Test 9.x. This may be useful if your organization has a combination of DTP Engines 10.x on the build server (such as DTP Engine for C/C++) and Parasoft Test 9.x desktop machines (such as Parasoft C/C++test 9.6).

1. Open the `DTP_HOME\conf\PSTRootConfig.xml` configuration file
2. Add the following entry under the `<root-config>` tag:

   ```
   <enable-forward-to-team-server>true</enable-forward-to-team-server>
   ```

If the **Enable unauthorized connections** option is enabled for Team Server, all static analysis reports from DTP Engines received by Data Collector are forwarded to Team Server by default.



If the **Enable unauthorized connections** option is disabled and named accounts are in use (see “Configuring Named Accounts”, page 91), the Parasoft Test Global settings or Parasoft Test settings for the project must specify the username and password. Basic usage:

```
tcm.server.username=team_user
tcm.server.password=123456789
```

For more information, please refer to Parasoft Test User Guide.

Changing Multicast DNS Usage

Development Testing Platform uses multicast DNS to broadcast its services in the local network—to Parasoft Test and to other Development Testing Platform servers. This allows Parasoft Test to automatically detect available Development Testing Platform servers, which enables easy configuration of Parasoft Test settings.

We strongly recommend that you leave this service enabled. However, you can disable it if needed.

To disable multicast DNS:.

1. Open the <Development Testing Platform installation directory>/conf/pcc.conf file and add the `<jmdns-autoconf-disabled>true</jmdns-autoconf-disabled>` node as a child of the `<Root>` node.

   The file should now look like this:

```
<Root>
    <license>
        ...
    </license>
    <jmdns-autoconf-disabled>true</jmdns-autoconf-disabled>
    ...
...
```

2. Open the <Development Testing Platform installation directory>/tomcat/webapps/bpel/WEB-INF/engine.xml file and set the `advertiseEndpoint` option to `false`.

Customizing Exports to Microsoft Word

Some reports provide the option to export to a Microsoft Word document (docx). Clicking the **[Export to Word]** button opens the current report in a docx file. This file is based on a Microsoft Word template that is predefined with the Parasoft Development Testing Platform header and footer.

If you prefer to use a custom template (e.g., with your organization's branding):

1. Copy your `.docx` template to the `$DTP_HOME/grs/xreports/planning/common/docx` directory.
2. Change its name to `template.docx`.

Disabling and Enabling Applications from the Toolbar

You can disable applications, such as Report Center and Policy Center, from the DTP toolbar.

1. Open the `[INSTALLATION_DIR]\conf\PSTRootConfig.xml` configuration file in an editor.
2. Uncomment the `<visible-apps>` section and set each application to `false` to disable or `true` to enable in the toolbar.

   If your installation does not have the <visible-apps> section, you must manually add it under the `<root-config>` node:

```
<!--   Configures which web application can be visible in DTP head menu -->
   <visible-apps>
      <report-center>true</report-center>
      <project-center>true</project-center>
      <policy-center>true</policy-center>
      <team-server>true</team-server>
      <license-server>true</license-server>
      <user-administration>true</user-administration>
   </visible-apps>
```

3. After re-configuring, the Parasoft DTP Server needs to be restarted.

A license for each application is also required. Individual users must also have credentials to see enabled applications (see “User Administration”, page 301).

Integrations

In this section:

- Connecting Report Center and Project Center to Parasoft Test
- Importing BTS Data from CSV
- BTS and RMS Scanner and Updater
- Integrating Report Center and Project Center with Third Party Tools
- Subscribing To Development Testing Platform Events

Connecting Report Center and Project Center to Parasoft Test

ITo import results from Parasoft Test products, such as Jtest, C++test, dotTEST, and SOAtest, connection parameters must be properly configured:

1. In your Parasoft Test product, choose **Parasoft> Preferences** to open the Preferences dialog.
4. Select the **Parasoft> Concerto**.
5. Specify the appropriate connection properties. For details, see the "Connecting to Parasoft Report Center and Project Center" topic in the Parasoft Test User's Guide.
6. Click **Apply**.
7. Click **OK** to set and save your settings.

Importing BTS Data from CSV

Development Testing Platform can be integrated with any bug tracking system by means of using CSV (Comma Separated Values) files. Any defect/enhancement data that your bug tracking system can export to a CSV file can be imported into Development Testing Platform.

To integrate Development Testing Platform with a BTS/RMS using CSV files, perform the steps in the following sections:

- Preparing CSV Files
- Formatting your CSV Files
- Providing a Configuration File for BTS Scanner

Note: If your CSV file contains national characters (for example, in defects names), please ensure that it is saved using UTF-8 encoding before it is imported into Development Testing Platform.

You can also import internal defects to projects in Development Testing Platform from a .csv, .xls, or .xlsx file. When defects are imported from file, Development Testing Platform recognizes them as internal defects, which enables more properties to be defined. See “Importing Defects from File”, page 45 for more information.

Preparing CSV Files

Development Testing Platform requires two files for BTS/RMS integration via CSV:

- **`CSV_bugs.csv`:** This file should contain a list of defects from your BTS.
- **`CSV_activities.csv`:** This file should contain the history of the status changes of your defects.

  Examples of these files can be found in: `DTP\grs\extras\csv-bts`.

These files will be scanned periodically by the BTS Scanner job (every 20 minutes by default, or on-demand from the Web interface) and the data will be fed into Development Testing Platform.

It is recommended that you update these data files (export from your BTS) periodically, e.g. every day (or ideally on each change in BTS), to ensure that Development Testing Platform has up-to-date defect data.

The following sections describe how to create the CSV files used for BTS integration.

Creating a CSV Defects File

Development Testing Platform can use a CSV defects file to read the basic properties of each defect/enhancement. Let’s assume this file is named `CSV_bugs.csv`.

A	B	C	D	E	F	G	H	I
ID	Type	Summary	Status	Created On	Owner	Priority	Severity	Resolution
40174	defect	Problem with add-ins.	VERIFIED	12/21/09 06:26 AM	user2	medium	Low	FIXED
40185	enhancement	Reorganize categories in HTML Report	NEW	12/21/09 12:26 PM	user4	high		
40175	defect	Unit testing for Z project fails.	RESOLVED	12/21/09 07:06 AM		medium	High	FIXED
40176	defect	[internal] Update plug-ins	RESOLVED	12/21/09 09:49 AM	user5	medium	Medium	FIXED

Note that Development Testing Platform scans this file periodically.

There are 2 possible approaches to what data should be exported to the `CSV_bugs.csv` file:

- Delete `CSV_bugs.csv` periodically and save into it only the defects which have changed their properties or which have been added recently and which you want to be imported into Development Testing Platform.

  For example, if one defect (already scanned into Development Testing Platform) has changed its status, and another defect has been added to your BTS (and has not been scanned into Development Testing Platform), the `CSV_bugs.csv` should contain 2 rows with the current properties of these defects.

- Include all the defects from your BTS in `CSV_bugs.csv` - When scanning periodically, Development Testing Platform will update only the properties of bugs which have changed or have been added since the last scan.

  Note: when a specific defect is imported into Development Testing Platform, it is never deleted from it. You can only update the defect properties or add new defects by changing the `CSV_bugs.csv` file.

CSV_bugs.csv columns:

The following defect properties are *required* in the CSV file:

- **ID:** The identifier of your defect/enhancement in your bug tracking system.
- **Type:** This field is used to identify the item in your bug tracking system as either a defect or an enhancement. Values of this column will be interpreted as features if and only if they match one of the items specified under the <feature-request> tag (defined in #2 step ix) `<bts>/<feature-request>`). Otherwise, values will be interpreted as defects.
- **Summary:** The summary of the defect.
- **Status:** Designates whether the defect is in open or closed state. See also *#2 vi) <bts>/<resolved-status>* below
- **Created On:** The date when the defect has been created in BTS

The following defect properties are *strongly recommended* to be present in CSV file (as they are displayed on Development Testing Platform reports):

- **Owner:** The current assignee of this defect in BTS.

  Tip: For the owner property, it is recommended that you use a login name which matches the user's login in Development Testing Platform, or the email address as defined for the user in Development Testing Platform, in the Users Administration module. This facilitates scheduling tasks from defects/enhancements in the Project Center module, and assigning them to Development Testing Platform owners.

- **Priority/Severity:** priority of the defect
- **Resolution:** designates the way in which the defect has been resolved. Leave empty if not yet resolved. See also *vii) <bts>/<inactive-resolution>*

The following additional properties (columns) can be present in `CSV_bugs.csv`: Version, Milestone, Hardware, Os, Project, and Component. These properties will be imported into Development Testing Platform for future use. An example file that includes these properties can be found in `DTP\grs\extras\csv-bts\CSV_bugs_ext.csv`.

Creating a CSV Defect History File

Development Testing Platform can use a CSV defect history file to read the history of status changes made to defects in your BTS. Let's assume this file is named `CSV_activities.csv`.

> **Note:** You can also provide the history of changes of the other (than status) fields of your defects, however they will not be used by Development Testing Platform until a future release.

As in `CSV_bugs.csv`, there are 2 possible approaches to what data should be exported to the `CSV_activities.csv` file:

- Delete `CSV_activities.csv` periodically and save into it only the changes made to the statuses of the defects recently and you would like to update them in Development Testing Platform.
- Have whole the history of the status of the defects changes from your BTS in `CSV_activities.csv` - Development Testing Platform when scanning periodically will import only the new entries which have appeared since the last scan.

The history of status changes is used in Report Center reports. For example, Tester Dashboard presents a graph that charts the trends of statuses of your project defects over time.

Changes made to _gresolution_h property can also be saved in this file. This can be used to specify if a particular defect is made inactive.

ID	User	Activity Date	Changed Field	Old Value	New Value
40174	user5	01/17/10 06:05 AM	Status	NEW	RESOLVED
40174	user5	01/25/10 01:18 AM	Status	RESOLVED	VERIFIED
40175	user5	01/23/10 07:03 AM	Status	NEW	ASSIGNED
40175	user1	01/23/10 07:44 AM	Status	ASSIGNED	RESOLVED
40176	user5	01/26/10 09:50 AM	Status	NEW	RESOLVED

CSV_activities.csv columns:

The activities file should contain the following info:

- **ID:** The defect ID.
- **User:** user who performed the change.
- **Activity Date:** Date on which change occurred.
- **Changed Field:** The defect attribute which has changed. In practice, the status attribute is used on Development Testing Platform reports, however changes to additional attributes, such as owner, resolution etc., can be imported to Development Testing Platform.
- **Old Value:** The value of the field before the change
- **New Value:** The value of the field after the change

Formatting your CSV Files

Apply the following formatting constraints to your CSV files:

- Cells should be separated by commas. If cell contains comma char, it should be enclosed in quotes.
- Quotes in cells should be doubles
- The last field of a line is not followed by a comma.
- Null/empty fields are represented by two commas in a row.
- Files should typically end with a single `END-OF-LINE`.

Providing a Configuration File for BTS Scanner

Development Testing Platform requires that you provide a configuration file for the BTS Scanner job.

There is an example of such a configuration file in `DTP\grs\config\bts\examples\ExampleCSVScannerConfig.xml`.Please make a copy of it and adjust all the parameters described below.

After you are done, please move this file to the `DTP\grs\config\bts` directory. Development Testing Platform periodically scans this folder and preforms BTS scanning for each configuration file found in this directory.

To configure BTS Scanner for CSV integration, please provide values for the following attributes:

i) *<bts>/<name>*

- The label you want to use to label this instance of your BTS; for example, "My BTS Server."

This name appears in Project Center > Search Defects/Enhancements page in "Defect Tracking System" drop-down menu.

ii) *<bts>/<bugs-file>*

the location of **CSV_bugs.csv**

iii) *<bts>/<activities-file>*

the location of **CSV_activities.csv**

iv) *<bts>/<fields-mapping>*

This section specifies the names of the columns in CSV_bugs.csv file. Columns from *ID* through *Created On* are required to be present. Columns from *Owner* through *Severity* are strongly recommended to be present. The rest are optional fields which can be imported to Development Testing Platform for future use.

```
<fields-mapping>
        <id>ID</id>
        <bug-type>Type</bug-type>
        <summary>Summary</summary>
        <status>Status</status>
        <creation-date>Created On</creation-date>
        <assigned-to>Owner</assigned-to>
        <priority>Priority</priority>
        <severity>Severity</severity>
        <resolution>Resolution</resolution>

        <version>Version</version>
        <milestone>Milestone</milestone>
        <hardware>Hardware</hardware>
        <os>Os</os>
        <project>Project</project>
        <component>Component</component>
</fields-mapping>
```

v) *<bts>/<activity-fields-mapping>*

This section specifies the names of the columns in **CSV_activities.csv** file. All the columns are required to be present in **CSV_activities.csv** file.

```
<activity-fields-mapping>
        <id>ID</id>
        <who>User</who>
        <date>Activity Date</date>
        <changed-field>Changed Field</changed-field>
        <old-value>Old Value</old-value>
```

```
                    <new-value>New Value</new-value>
                </activity-fields-mapping>
```

vi) *<bts>/<resolved-status>*

Specify which values of your BTS defects designate that the defect is resolved, in other words define what indicates that the defect is fixed or enhancement implemented.

This info is used by Development Testing Platform on various reports, for example Tester Dashboard graph shows the over-time trends of defects open vs not tested/tested. Defect should be closed to be treated tested or not tested.

For example:

```
<resolved-status>
        <status>RESOLVED</status>
        <status>CLOSED</status>
</resolved-status>
```

vii) *<bts>/<inactive-resolution>*

Specify which values of your defects designate that the defect is inactive. Such defects will not be imported into Development Testing Platform.

For example:

```
<inactive-resolution>
        <resolution>DUPLICATE</resolution>
        <resolution>INVALID</resolution>
</inactive-resolution>
```

viii) *<bts>/<date-formats>*

A date format which is used in **CSV_bugs.csv** and **CSV_activities.csv** files should be specified here. The most common formats have been specified:

```
<date-formats>
        <date-format>yyyy-MM-dd hh:mm</date-format>
        <date-format>yyyy-MM-dd hh:mm:ss.0</date-format>
        <date-format>yy MMM-dd hh:mm</date-format>
        <date-format>MM/dd/yy hh:mm aa</date-format>
</date-formats>
```

The format of entries should meet the one defined for SimpleDateFormat class in java. (See http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html)

ix) *<bts>/<feature-request>*

The values of *<item>* nodes specify which values of _gType_h column in **CSV_bugs.csv** identifies the specific BTS item as enhancement (not a defect).

For example:

```
        <feature-request>
                <item>enhancement</item>
        </feature-request>
```

3. How to verify the imported CSV defects/enhancements.

After you adjust parameters of your CSVScannerConfig.xml file, please copy it to DTP\grs\config\bts directory. Development Testing Platform periodically scans this folder and preforms BTS scanning for each configuration file found in this directory. For details on how Development Testing Platform scans BTS configuration files, see “Running BTS Scanner”, page 223.

Integrating with Bug Tracking Systems and Requirement Management Systems

Development Testing Platform integrates with several bug tracking systems (BTS) and requirement management systems (RMS) using either BTS Scanner or RMS Scanner, which are internal components that must be configured to work with each system integration. BTS Scanner or RMS Scanner transfer bug or requirement data from supported systems into the database.

Each scanner retrieves essential information from a BTS or RMS database and sends it to the Report Center database. The transferred data is then used by Report Center/Project Center to create reports related to bugs or requirements. After configuration, BTS or RMS Scanner will automatically run at a default interval of every 15 minutes. It can also be run on-demand.

Configuring BTS or RMS Scanner with the Built-in UI

You can configure BTS or RMS Scanner from Report Center or Project Center administration. This option is only available for the following systems:

Bug Tracking Systems

- Bugzilla
- JIRA 4.x
- JIRA 5.x/6.x (requires SSL to be running on DTP server; see “Running Development Testing Platform Under SSL”, page 153, for more information)

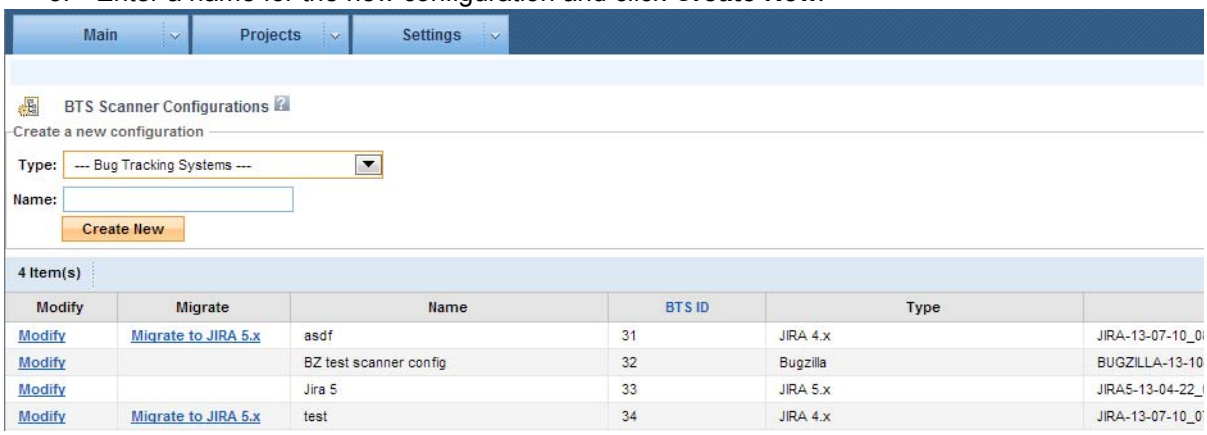
Requirements Tracking Systems

- Blueprint
- DOORS

Adding BTS Configurations

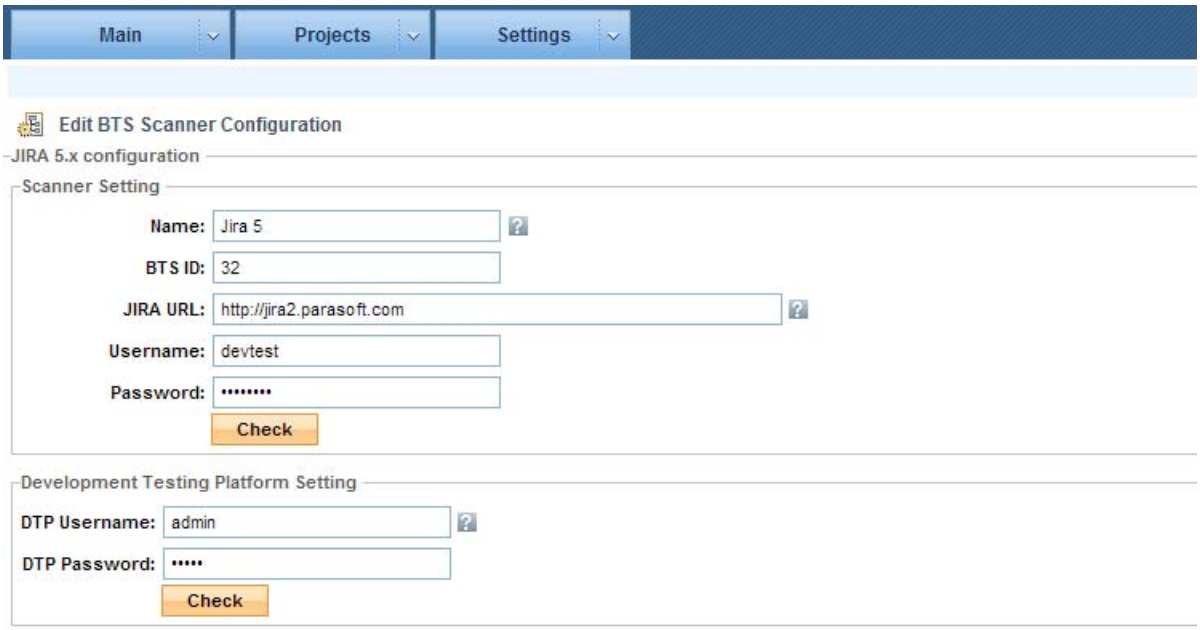
1. Open the administration page and choose **Settings> BTS Scanners**.
2. Choose a system from the **Type** drop-down menu

3. Enter a name for the new configuration and click **Create New**.



Configuring BTS Scanners

1. Open the BTS Scanners administration page and click **Modify** in the Modify column.
2. Enter the server settings and login information

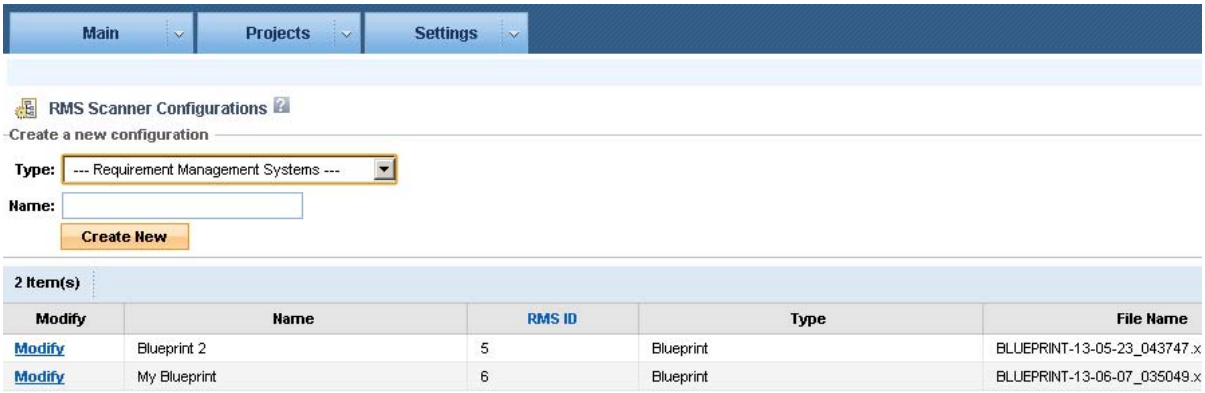


3. Configure how artifact statuses should be mapped to Development Testing Platform.

4. Click **Save** when finished.

Adding RMS Configurations

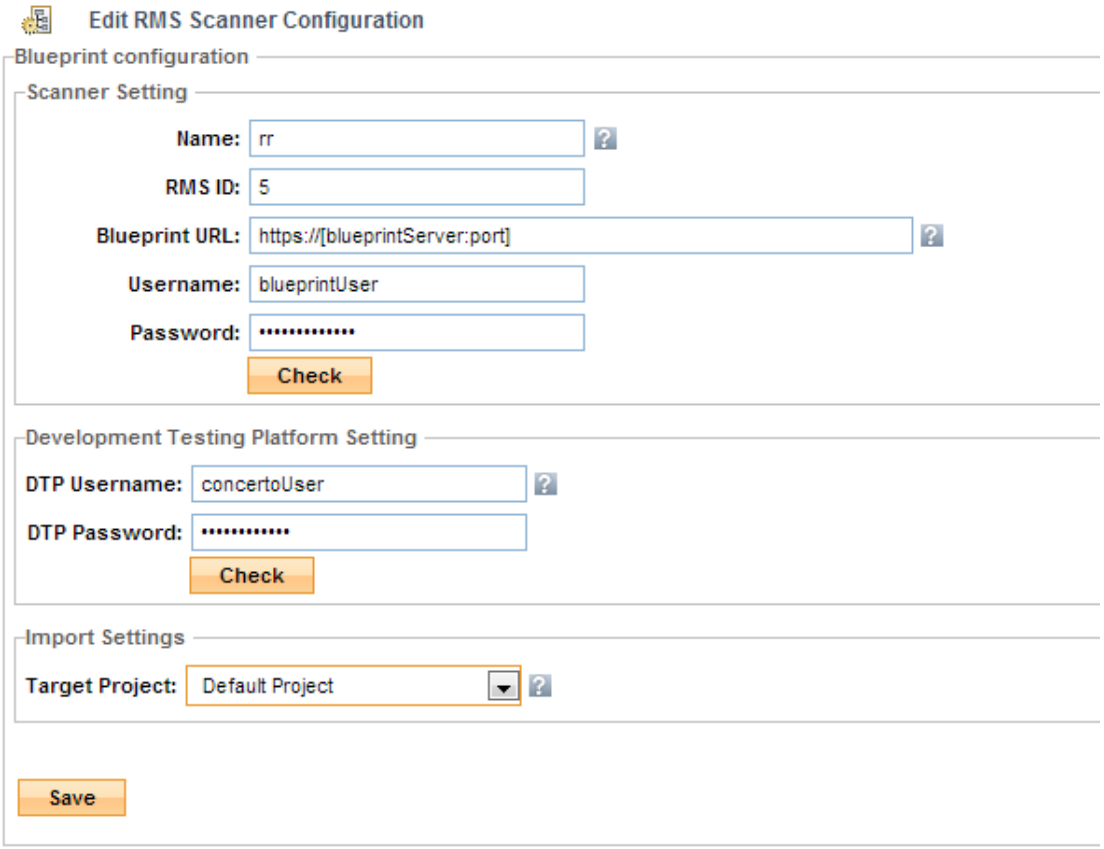
1. Open the administration page and choose **Settings> RMS Scanners**.
2. Choose a system from the **Type** drop-down menu.
3. Enter a name for the new configuration and click **Create New.**



Configuring RMS Scanners

1. Open the RMS Scanners administration page and click **Modify** in the Modify column.

2. Enter the server settings and login information



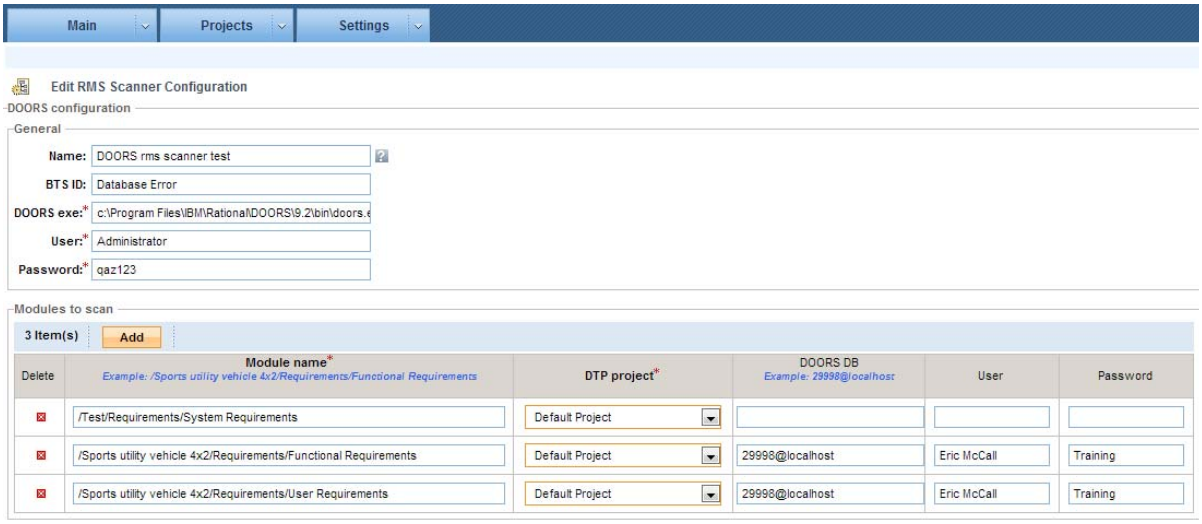
Configuring IBM Rational DOORS Through the UI

The DOORS configuration page includes parameters that unique to that system.

1. Follow the instructions for adding a new configuration above.

1. Open the **RMS Scanners** administration page and click **Modify** in the Action column.
2. Enter the server settings and login information

3. Click **Add** to define a DOORS module from which requirements should be imported to DTP Server. The path you specify must point to a DOORS Formal Module or Linked Module and not to, for example, a DOORS Folder.



4. Specify additional DOORS integration information:
    - ***DTP project:*** *Development Testing Platform project to which the specified DOORS module requirements will be imported.*
    - ***DOORS DB:*** *(Optional) Alternative DOORS database.*
    - ***Note:*** *This field should only be used if the corresponding Module name is defined at a DOORS db other than the one linked to the* ***DOORS Exe****, which is specified in your General configuration.*
    - ***User:*** *Valid DOORS username with access to DOORS database and read privileges for all objects from* ***Module Name****.*
    - ***Password:*** *Valid password for* ***User****.*
5. Click **Save**.

    A new DOORS BTS Scanner configuration file is created in your `$DTP_HOME/grs/config/bts` directory.

Manually Configuring BTS or RMS Scanner

You can edit the corresponding XML file stored in the DTP_HOME/grs/config/bts directory to manually configure BTS and RMS Scanner. See “BTS and RMS Scanner and Updater”, page 221., for instructions on configuring and using BTS and RMS Scanner.

Development Testing Platforms supports simple integration with several systems. The following sections provide instructions on manually configuring BTS/RMS Scanner for use with each system:

- Integrating with HP Quality Center
- Integrating with Bugzilla
- Integrating with IBM Rational ClearQuest
- Integrating with Atlassian JIRA

- Integrating with IBM Rational Change and Synergy

BTS and RMS Scanner and Updater

You can manually configure BTS Scanner to scan your BTS/RMS by providing the appropriate settings in one of BTS Scanner's .xml configuration files, which are stored in `DTP_HOME/grs/config/bts`, on the host where your Development Testing Platform server is installed.

The following topics are discussed in this chapter:

- Preparing an Example Configuration File
- Adding a Prepared Configuration File
- Custom BTS Scanner/Updater
- Running BTS Scanner
- Verifying that BTS Scanner is Working
- Configuring BTS Updater
- Using Original IDs in Reports

Preparing an Example Configuration File

An example configuration file is provided for each supported BTS/RMS in `DTP_HOME/grs/config/bts/examples`.

To prepare an example configuration file, follow these steps:

1. Navigate to the configuration examples directory: `DTP_HOME/grs/config/bts/examples`.
2. Choose the example file provided for your particular BTS.

   For example, to configure BTS Scanner with JIRA, you would choose `ExampleJiraScannerConfig.xml`.
3. Copy your chosen example file to the same directory: `DTP_HOME/grs/config/bts/examples`.
4. Rename the file as appropriate.

   For example, `ExampleJiraScannerConfig.xml` could be renamed as `MyJiraScannerConfig.xml`.
5. Edit the configuration file by providing all the settings for your BTS/RMS, depending on the vendor:

   BTS Scanner can currently be configured to support several BTS/RMS, each of which requires a particular configuration.

   For details on configuring BTS Scanner to work with a specific BTS/RMS, see:

   - Integrating with HP Quality Center
   - Integrating with Bugzilla
   - Integrating with IBM Rational ClearQuest
   - Integrating with Atlassian JIRA*--older versions*

BTS Scanner can also import defect records from CSV files, so any BTS/RMS that can export its data to a CSV file can also be integrated with Development Testing Platform.

For details on using BTS Scanner with CSV files, see "Importing BTS Data from CSV", page 210.

Adding a Prepared Configuration File

After you have adjusted a configuration file for your specific BTS/RMS, it is ready to be read by BTS Scanner. BTS Scanner runs periodically and processes each .xml configuration file from `DTP_HOME/grs/config/bts`.

In a new Development Testing Platform installation, this directory is empty. Several configuration files can be added to this directory so that BTS Scanner can connect to several different BTS/RMS (or several instances of the same BTS/RMS with different parameters). Each configuration file you add to this directory is responsible for integration with one instance of the corresponding BTS/RMS.

To add a prepared configuration file to be read by BTS Scanner:

- Copy or move the chosen file to `DTP_HOME/grs/config/bts`.

Custom BTS Scanner/Updater

You can implement the following Java APIs to create a custom BTS Scanner:

- `com.parasoft.api.dtp.defects.scan.DefectScanner`
- `com.parasoft.api.dtp.defects.scan.DefectScannerFactory`
- `com.parasoft.api.dtp.defects.update.DefectUpdator`
- `com.parasoft.api.dtp.defects.update.DefectUpdatorFactory`

See "Java API", page 289, for information on accessing the Java API documentation. See "Viewing Log Files", page 289, for log information.

Installing Your Custom Implementation

1. Create a jar file for your custom scanner/updater.
2. Copy the jar into the `$DTP_HOME/tomcat/webapps/grs/WEB-INF/lib` directory.
3. Restart Development Testing Platform.

Configuring a BTS Scanner to Use Your Custom Implementation

You must create an XML configuration file in `$DTP_HOME/grs/config/bts`. You can use `$DTP_HOME/grs/config/bts/examples/ExampleCustomScannerConfig.xml` as a template.

The two most important parts of the custom XML configuration:

- The `scannerFactoryClass` element value must be set to the name of the class that implements the `com.parasoft.api.dtp.defects.scan.DefectScannerFactory` interface.

- The `updatorFactoryClass` element value must be set to the name of the class that implements the `com.parasoft.api.dtp.defects.update.DefectUpdatorFactory` interface.

Existing documentation for BTS Scanner integration is in this chapter.

Custom RMS Scanner

You can implement the following Java APIs to create a custom RMS scanner:

- `com.parasoft.api.dtp.requirements.scan.RequirementScanner`
- `com.parasoft.api.dtp.requirements.scan.RequirementScannerFactory`

See "Java API", page 289 for information on accessing the Java API documentation. See "Viewing Log Files", page 289, for log information.

Installing Your Custom Implementation

1. Create a jar file for your custom scanner/updater.
2. Copy the jar into the `$DTP_HOME/tomcat/webapps/grs/WEB-INF/lib` directory.
3. Restart Development Testing Platform.

Configuring RMS Scanner to User Your Custom Implementation

You must create an XML configuration file in `$DTP_HOME/grs/config/bts`. You can use `$DTP_HOME/grs/config/bts/examples/ExampleCustomRmsScannerConfig.xml` as a template.

The `scannerFactoryClass` element value must be set to the name of the class that implements the `com.parasoft.api.dtp.requirements.scan.RequirementScannerFactory` interface; this is the most important part of the custom XML configuration.

Running BTS Scanner

After you have finished configuring the integration with your BTS/RMS (as described in "Configuring BTS Scanner") and placed your .xml file in the `$DTP_HOME/grs/config/bts` directory, BTS Scanner processes it, connects to the defined BTS/RMS, and imports basic info about your defects/requirements to the Development Testing Platform database.

BTS Scanner runs automatically, but it can also be run on-demand, to ensure that data in Development Testing Platform reflects the latest changes made in your BTS/RMS. Either way, you can verify that BTS Scanner is working as you expect it to.

Running BTS Scanner Automatically

BTS Scanner is a Development Testing Platform background job. (for general background jobs description see chapter [reference]). BTS Scanner runs automatically every 15 minutes, for each file in

the `$DTP_HOME/grs/config/bts` directory, connects to the specific BTS/RMS, and imports data to Development Testing Platform.

You can configure the frequency with which BTSScanner is run. To do so, perform the following steps:

1. Adjust the frequency parameter in `$DTP_HOME/grs/config/CronConfig.xml`:

```
<Job frequency="15" id="BTS Scanner" runDayOfWeek="*"
runHour="*" run-Minute="*">
      <class
      name="com.parasoft.grs.rserver.cronjobs.bts.BtsScannerJob
      "/>
</Job>
```

2. Restart your Development Testing Platform server.

Running BTS Scanner On Demand

You can run the BTS Scanner on-demand from the Development Testing Platform Web site, from either of the following locations:

- Project Center: Search Defects/Enhancements

  See “Running BTS Scanner On Demand from Search Defects/Enhancements Page”, page 224 for details.

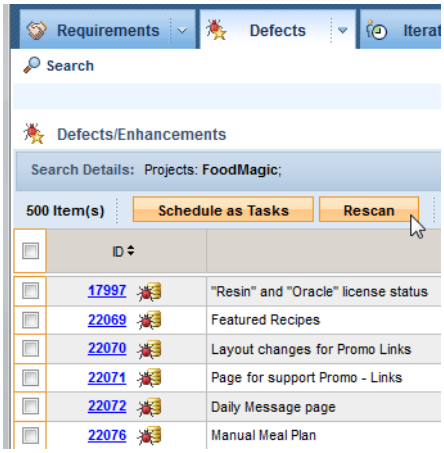
- Report Center: Administration > Tools > Calculate> Recalculate Defect/Enhancement.

  See “Automatically Run Scans and Calculations”, page 174 for details.

Running BTS Scanner On Demand from Search Defects/Enhancements Page

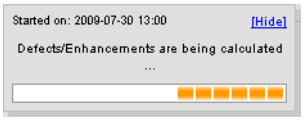
1. From Project Center, select Search Defects/Enhancements.
2. On the Defects/Enhancements page, click **Rescan**, as shown in Figure 2:

Figure 2: Defects/Enhancements - Rescan Defects/Enhancements



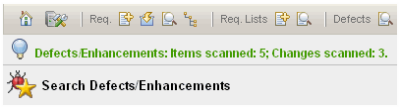
A pop-up is displayed to show scan is in progress (Figure 3):

Figure 3: Scan In-progress Message



Upon scan completion, the Status Bar displays the number of bugs and changes that were scanned (Figure 4):

Figure 4: Scan Complete Message



Verifying that BTS Scanner is Working

Because BTS Scanner can run both automatically and on-demand, you may either want to verify that BTS Scanner is running, or that it has imported data to Development Testing Platform as expected.

To verify BTS Scanner background work, view the `bts_scanner.log` file, which can be found in `DTP_HOME/logs/` (see “Viewing Log Files”, page 289).

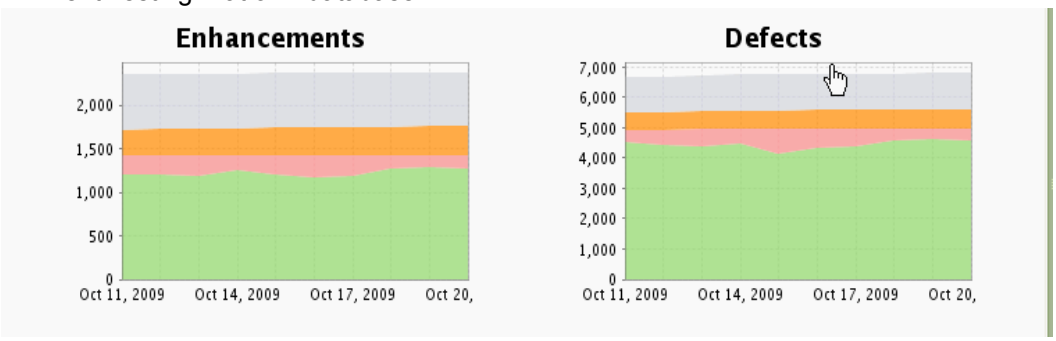
To verify that BTS Scanner has imported your BTS/RMS data to the Development Testing Platform database, perform one of the following:

1. Seach Defects/Enhancements
   - *Expand the "Bug Tracking System" drop-down menu, as shown in Figure 2.*

     This menu displays the names of BTS/RMS that are integrated with your installation of Development Testing Platform. These names are defined in the <bts/name> .xml node of each of your .xml configuration files.
   - *Click the **Search** button.*

     All the defects imported to the Development Testing Platform database are searched for and displayed.

     For more details on searching defects/enhancements, see *Project Center User’s Manual* section "Searching Defects/Enhancements."
2. In the Report Center Reports view, choose **Tests> Change-based Testing> Requirements/ Defects**
3. Switch to Default Project. This view displays all defect/enhancement data from the Development Testing Platform database.



4. For further details on using Report Center and Project Center with Defects/Enhancements, please see:
    - “Source Control Filter”, page 168
    - “Defects and Enhancements Reports”, page 129
    - “Working with Defects/Enhancements”, page 108
    - “Creating Tasks for Requirements and Defects/Enhancements”, page 120

Configuring BTS Updater

Development Testing Platform modifies status and comments for issues in supported Bug Tracking and Requirement Management Systems when the tasks associated with these issues are modified. This is done through the BTS Updater utility, which propagates Development Testing Platform updates to integrated/supported bug tracking systems.

Note: BTS Updater currently supports comment updates for all versions of Bugzilla; however, status updates only work for Bugzilla 3.4 or higher:

- For supported versions of Bugzilla, see “Integrating with Bugzilla”, page 243.
- For more information about how Development Testing Platform interacts with bug tracking systems, see “About Bug Tracking Systems Synchronization”, page 229

You can configure BTS Updater to propagate status and comment changes to your BTS/RMS by providing the appropriate settings in one of BTS Scanner’s .xml configuration files, which are stored in `DTP_HOME/grs/config/bts`, on the host where your Development Testing Platform server is installed.

To enable BTS Updater, add a new user and password to the `<connection-settings>` node of your existing Bugzilla .xml configuration file.

For example:

```
<bts>
        <connection-settings>
                <user>root</user>
                <pass encrypted="false">root</pass>
        </connection-settings>
</bts>
```

Important! This user must have an account for the Web interface to the integrated BTS, and that account should have read/write privileges for all projects in the BTS.

Using Original IDs in Reports

If you want to see a task or requirement’s Original ID instead of Development Testing Platform IDs in the Requirements Code Review and Requirements Code Review Details reports, then:

1. Go to the `$DTP_HOME/grs/xreports/planning/req/RequirementsCodeReview.xml` file.
2. And add the entry
   `<parameter name="useOriginalId">true</parameter>`
   under the <private-parameters> tag.

3. Add the same entry in the `DTP_HOME/grs/xreports/codereview/CodeReviewDetails.xml` file.

You can configure BTS Scanner to work with HP Quality Center (formerly known as Test Director). Test Director and Quality Center are synonymous, so both will be referred to as "QC" for the remainder of this section. Integration has been tested with the following Quality Center versions: 9, 10 and 11.

BTS Scanner for QC allows you to import requirements and defects from the QC database to Report Center.

Development Testing Platform provides two ways of integration with HP QC:

- Direct HP QC database access. In this mode, Development Testing Platform reads QC data directly and imports:
  - *QC requirements as defects (of type Enhancement) to Development Testing Platform*
  - *QC defects as defects in Development Testing Platform*
- Using Open Test Architecture (OTA). In this mode, Development Testing Platform reads QC data directly and imports:
  - *QC requirements as requirements to Development Testing Platform*
  - *QC defects as defects in Development Testing Platform*

Using Direct HP QC Database Access

Note: *Most users can use the default settings for all steps below except for step 2. QC database-connection settings need to be customized to suit your specific connection.*

To configure integration through direct HP QC database access.

1. Provide the following general settings:

Setting	Description
<bts>/<type>	Should be set to "TestDirector".
<bts>/<name>	The name you want to use to label this instance of your HP QC server (for example, "Test Director"). Each HP QC server instance must have a unique name. This name will be shown in Project Center's Search Defects/Enhancements page (in the Defect Tracking System drop-down menu).

   For example:

```
<bts type="TestDirector">
<name>Test Director</name>
```

2. Provide QC database-connection settings.

   Since data for each QC project is stored in a separate database, you are required to provide at least one database name to be scanned.

   The name of databases for each project can be accessed from Quality Center Site Administration, or from any MS SQL client.

Note: *It is recommended that you write down the names of the defined QC databases. You will need this information for database connection settings.*

The following QC database connection settings are required for QC integration:

Setting	Description
<bts>/<db-connection>/<db-type>	MSSQL only.
<bts>/<db-connection>/<user>	Database user.
<bts>/<db-connection>/<pass>	Database password.
<bts>/<db-connection>/<url>	QC database connection URL. ***Note:*** *Do NOT include database name in* `connection url`.
<bts>/<db-connection>/<database>	QC database name. At least one database name is required.

For example:

```
  <db-connection>
          <db-type>MSSQL</db-type>
          <!-- currently only MS SQL server supported. -->
          <user>sa</user>
          <!-- valid MS SQL server user -->
          <password encrypted="true">abc123</password>
          <!-- valid MS SQL server user passwd -->
          <url>jdbc:sqlserver://localhost;</url>
<!-- sample MSSQL connection url: <url>jdbc:sqlserver://HOST;</url> -->
```

Note: *Do NOT include database name in connection URL.*

```
          <!-- At least one db name is required -->
          <database>default_test_db</database>
          <database>QualityCenter_Demo_db</database>
  </db-connection>
```

3. Provide a fields-mapping configuration.

   Fields-mapping for both defects and requirements is based on the database column names: BUG and REQ table. For a detailed explanation of each field, see the Quality Center manual: `quality_center_db.chm`

   Section field-mapping for both defects and requirements is commented out. It describes default mapping configuration. You can remove any comments from this section and change settings. However, this typically is not necessary.

   Note: *N/A indicates that a field is not available.*

   For example:

```
  <defects>
          <fields-mapping>
                  <summary>BG_SUMMARY</summary>
                  <status>BG_STATUS</status>
                  <resolution>BG_STATUS</resolution>
```

```
                <priority>BG_PRIORITY</priority>
                <severity>BG_SEVERITY</severity>
                <project>BG_PROJECT</project>
                <reporter>BG_DETECTED_BY</reporter>
                <assigned-to>BG_RESPONSIBLE</assigned-to>
                <creation-date>BG_DETECTION_DATE</creation-date>
                <version>BG_DETECTION_VERSION</version>
                <milestone>BG_PLANNED_CLOSING_VER</milestone>
                <hardware>N/A</hardware>
                <os>N/A</os>
                <component>BG_PROJECT</component>
                <modification-date>BG_VTS</modification-date>
        </fields-mapping>
</defects>

<requirements>
        <fields-mapping>
                <summary>RQ_REQ_NAME</summary>
                <status>RQ_REQ_STATUS</status>
                <resolution>RQ_REQ_STATUS</resolution>
                <priority>RQ_REQ_PRIORITY</priority>
                <severity>N/A</severity>
                <project>RQ_REQ_PRODUCT</project>
                <reporter>RQ_REQ_AUTHOR</reporter>
                <assigned-to>RQ_REQ_AUTHOR</assigned-to>
                <creation-date>RQ_REQ_DATE</creation-date>
                <version>N/A</version>
                <milestone>N/A</milestone>
                <hardware>N/A</hardware>
                <os>N/A</os>
                <component>RQ_REQ_TYPE</component>
                <modification-date>RQ_VTS</modification-date>
        </fields-mapping>
</requirements>
```

4. Provide values for resolved statuses and inactive resolutions.

   Configuration of resolved status and inactive status mapping might depend on the fields-mapping.status notation. However, reconfiguration of this section is typically not necessary.

   For example:

```
<resolved-status>
        <!-- Requirement -->
        <status>Passed</status>
        <status>Failed</status>
        <status>Reviewed</status>
        <status>Not Completed</status>

        <!-- Defects -->
        <status>Closed</status>
        <status>Fixed</status>
</resolved-status>

<inactive-resolution>
        <!-- Requirement -->
```

```
            <resolution>N/A</resolution>

            <!-- Defects -->
            <resolution>Rejected</resolution>
    </inactive-resolution>
```

Notes:

- *After importing to Report Center, identifiers of scanned QC items have the following formats:*
  - *For PR: BUG_(real QC ID)*
  - *For FR: REQ_(real QC ID)*
- *Currently, functionality to link from Report Center reports to open PR/FR details in QC is not available.*
- *The Task Activity report requires the addition of QC users to the Report Center database in order to display any data.*
- *This is an incremental scan, which means that only requirements/defects which were modified in QC since the last scan are updated to Development Testing Platform. To ensure that Development Testing Platform can see what items changed in QC, you need to have the proper settings in QC's Project Customization/Project Entities forms. All fields which are listed in the "Provide a fields-mapping configuration" section should have an active History property in QC.*

Connecting to Quality Center's Database using Windows Authentication

The Development Testing Platform services must be installed and running on a Windows environment in order to connect to the MSSQL Server via Windows Authentication (because the official MSSQL Server JDBC driver only uses the credentials from the user account that's currently running the services for Windows Authentication).

To prepare for the Development Testing Platform setup:

1. Download the official MSSQL Server JDBC driver package, then follow the provided instructions to unload the package.
2. Open the **Start** menu, then right-click **My Computer** to open the menu. Click **Manage** to open the Computer Management window. (Alternatively, open the Control Panel from the **Start** menu, then go to **Administrative Tools> Computer Management**.)
3. In the right pane, expand **Services and Applications**, then double-click **Services** to open the list of Windows Services.
4. Right-click the **Parasoft Development Testing Platform**, then choose **Properties** to open the Parasoft Development Testing Platform Properties window.
5. Open the **Log On** tab. Select the **This account** option, then provide a Windows user account in the format `LOCATION\username`. This user account must be allowed access to the MSSQL Server database.
6. Provide the password and the user account in the next two fields, then click **OK**.
7. With the Parasoft Development Testing Platform service still selected in the Services list, click the **Stop the service** link to stop the Development Testing Platform service.
8. Open the `${DTP_HOME}/tomcat/lib/thirdparty` directory in a new window.
9. Create a copy of the sqljdbc4.jar file for backup purposes.

10. Copy the sqljdbc4.jar file from <MSSQL_JDBC_DRIVER>\sqljdbc_<VERSION>\<LANG> sqljdbc4.jar and into the current directory.
11. Access the `${DTP_HOME}/jre/bin` directory and add in the `sqljdbc_auth.dll` file from the `<MSSQL_JDBC_DRIVER>\sqljdbc_<VERSION>\<LANG>\auth\<ARCHITECTURE>` directory .

To set up Development Testing Platform:

1. Access the `${DTP_HOME}/grs/config/bts` directory and open the QC BTS Scanner configuration file.
2. Open the BTS Scanner configuration file, then find the `<db-connection>` element. If the configuration file is not already configured, follow the instructions from "Using Direct HP QC Database Access", page 235 to properly modify the configuration file.
3. For the URL value in the `<url>` element within in the `<db-connection>` element, append the property `integratedSecurity=true`.
4. Make sure the settings resemble the example provided below, then save the changes.

```
<db-connection>

<db-type>MSSQL</db-type>
<user>sa</user>
<password encrypted="true">abc123</password>
<url>jdbc:sqlserver://localhost;integratedSecurity=true;</url>

</db-connection>
```

5. Start the Parasoft Development Testing Platform service from the Services list by clicking the **Start the service** link.
6. Follow the instructions from "Verifying that BTS Scanner is Working", page 225 to validate that QC Scanner has been configured properly.

Using Open Test Architecture

BTS Scanner for QC using OTA API allows you to import requirements and defects from the QC database to Development Testing Platform. QC requirements are shown as Project Center Requirements and QC defects are shown as Project Center Bugs. Imported requirements and defects are in read-only mode: you cannot modify most attributes.

If the QC client is installed on the same machine where Development Testing Platform is installed, you can also update QC Requirements with data retrieved from Parasoft Development Testing Platform.

To configure integration using Open Test Architecture (OTA):

1. Provide the following general settings:

Setting	Description
<bts>/<type>	Should be set to "TestDirectorOTA".

Setting	Description
<bts>/<name>	The name you want to use to label this instance of your HP QC server (for example, "Test Director"). Each HP QC server instance must have a unique name. This name will be shown in Project Center's Search Defects/Enhancements page (in the Defect Tracking System drop-down menu).

For example:

```
<bts type="TestDirectorOTA">
<name>QC OTA</name>
```

2. Provide Quality Center database-connection settings.

   Since data for each QC project is stored in a separate database, you are required to provide at least one database name to be scanned.

   The name of databases for each project can be accessed from Quality Center Site Administration, or from any MS SQL client.

   Note: *It is recommended that you write down the names of the defined QC databases. You will need this information for database connection settings.*
   The following QC database connection settings are required for QC integration:

Setting	Description
<bts>/<db-connection>/<db-type>	Database vendor: MS SQL only.
<bts>/<db-connection>/<user>	Database user.
<bts>/<db-connection>/<pass>	Database password.
<bts>/<db-connection>/<url>	Quality Center database connection URL. **Note**: Do NOT include database name in the connection URL.
<bts>/<db-connection>/<database>	The QC database name. At least one database name is required. Additionally, it is required to specify the following attributes for the <database> tag: • qc-domain - specify QC Domain for qc-project • qc-project - specify QC project name • dtp-project - specify Development Testing Platform project name under which the scanned requirements will be stored. **Note**: If the specified project is not found in the Development Testing Platform database during the scanning process, it will be created automatically. qc-domain, qc-project, dtp-project attributes are used to create the Requirement properly.

For example:

```
<db-connection>
<db-type>MSSQL</db-type>
<user>sa</user>
<password encrypted="true">abc123</password>
<url>jdbc:sqlserver://localhost;</url>
<database qc-domain="DEFAULT" qc-project="QualityCenter_Demo"
dtp-project="MyProject">QualityCenter_Demo_db
</database>
</db-connection>
```

3. Configure the ota-connection.

   To enable QC defects to be automatically updated with Parasoft Development Testing Platform data, you need to specify credentials that make communication possible using OTA API. To do this, provide the following general settings:

Setting	Description
<bts>/<ota-connection>/<user>	Name of a user that can communicate using OTA API.
<bts>/<ota-connection>/<password>	Valid password for the specified user.
<bts>/<ota-connection>/<url>	Valid HTTP URL to QC.

   For example:

```
<ota-connection>
<user>sa</user>
<password>sa</password>
<url>http://localhost:8080/qcbin</url>
</ota-connection>
```

Integrating with HP Quality Center

You can configure BTS Scanner to work with HP Quality Center (formerly known as Test Director). Test Director and Quality Center are synonymous, so both will be referred to as "QC" for the remainder of this section. Integration has been tested with the following Quality Center versions: 9, 10 and 11.

BTS Scanner for QC allows you to import requirements and defects from the QC database to Report Center.

Development Testing Platform provides two ways of integration with HP QC:

- Direct HP QC database access. In this mode, Development Testing Platform reads QC data directly and imports:
  - *QC requirements as defects (of type Enhancement) to Development Testing Platform*
  - *QC defects as defects in Development Testing Platform*
- Using Open Test Architecture (OTA). In this mode, Development Testing Platform reads QC data directly and imports:
  - *QC requirements as requirements to Development Testing Platform*
  - *QC defects as defects in Development Testing Platform*

Using Direct HP QC Database Access

Note: *Most users can use the default settings for all steps below except for step 2. QC database-connection settings need to be customized to suit your specific connection.*

To configure integration through direct HP QC database access.

1. Provide the following general settings:

Setting	Description
<bts>/<type>	Should be set to "TestDirector".
<bts>/<name>	The name you want to use to label this instance of your HP QC server (for example, "Test Director"). Each HP QC server instance must have a unique name. This name will be shown in Project Center's Search Defects/Enhancements page (in the Defect Tracking System drop-down menu).

   For example:

```
<bts type="TestDirector">
<name>Test Director</name>
```

2. Provide QC database-connection settings.

   Since data for each QC project is stored in a separate database, you are required to provide at least one database name to be scanned.

   The name of databases for each project can be accessed from Quality Center Site Administration, or from any MS SQL client.

Note: *It is recommended that you write down the names of the defined QC databases. You will need this information for database connection settings.*

The following QC database connection settings are required for QC integration:

Setting	Description
<bts>/<db-connection>/<db-type>	MSSQL only.
<bts>/<db-connection>/<user>	Database user.
<bts>/<db-connection>/<pass>	Database password.
<bts>/<db-connection>/<url>	QC database connection URL. ***Note:*** *Do NOT include database name in* `connection url`.
<bts>/<db-connection>/<database>	QC database name. At least one database name is required.

For example:

```
  <db-connection>
          <db-type>MSSQL</db-type>
          <!-- currently only MS SQL server supported. -->
          <user>sa</user>
          <!-- valid MS SQL server user -->
          <password encrypted="true">abc123</password>
          <!-- valid MS SQL server user passwd -->
          <url>jdbc:sqlserver://localhost;</url>
<!-- sample MSSQL connection url: <url>jdbc:sqlserver://HOST;</url> -->
```

Note: *Do NOT include database name in connection URL.*

```
          <!-- At least one db name is required -->
          <database>default_test_db</database>
          <database>QualityCenter_Demo_db</database>
  </db-connection>
```

3. Provide a fields-mapping configuration.

   Fields-mapping for both defects and requirements is based on the database column names: BUG and REQ table. For a detailed explanation of each field, see the Quality Center manual: `quality_center_db.chm`

   Section field-mapping for both defects and requirements is commented out. It describes default mapping configuration. You can remove any comments from this section and change settings. However, this typically is not necessary.

   Note: *N/A indicates that a field is not available.*

   For example:

```
  <defects>
          <fields-mapping>
                  <summary>BG_SUMMARY</summary>
                  <status>BG_STATUS</status>
                  <resolution>BG_STATUS</resolution>
```

```
                <priority>BG_PRIORITY</priority>
                <severity>BG_SEVERITY</severity>
                <project>BG_PROJECT</project>
                <reporter>BG_DETECTED_BY</reporter>
                <assigned-to>BG_RESPONSIBLE</assigned-to>
                <creation-date>BG_DETECTION_DATE</creation-date>
                <version>BG_DETECTION_VERSION</version>
                <milestone>BG_PLANNED_CLOSING_VER</milestone>
                <hardware>N/A</hardware>
                <os>N/A</os>
                <component>BG_PROJECT</component>
                <modification-date>BG_VTS</modification-date>
        </fields-mapping>
</defects>

<requirements>
        <fields-mapping>
                <summary>RQ_REQ_NAME</summary>
                <status>RQ_REQ_STATUS</status>
                <resolution>RQ_REQ_STATUS</resolution>
                <priority>RQ_REQ_PRIORITY</priority>
                <severity>N/A</severity>
                <project>RQ_REQ_PRODUCT</project>
                <reporter>RQ_REQ_AUTHOR</reporter>
                <assigned-to>RQ_REQ_AUTHOR</assigned-to>
                <creation-date>RQ_REQ_DATE</creation-date>
                <version>N/A</version>
                <milestone>N/A</milestone>
                <hardware>N/A</hardware>
                <os>N/A</os>
                <component>RQ_REQ_TYPE</component>
                <modification-date>RQ_VTS</modification-date>
        </fields-mapping>
</requirements>
```

4. Provide values for resolved statuses and inactive resolutions.

   Configuration of resolved status and inactive status mapping might depend on the fields-mapping.status notation. However, reconfiguration of this section is typically not necessary.

   For example:

```
<resolved-status>
        <!-- Requirement -->
        <status>Passed</status>
        <status>Failed</status>
        <status>Reviewed</status>
        <status>Not Completed</status>

        <!-- Defects -->
        <status>Closed</status>
        <status>Fixed</status>
</resolved-status>

<inactive-resolution>
        <!-- Requirement -->
```

```
            <resolution>N/A</resolution>

            <!-- Defects -->
            <resolution>Rejected</resolution>
    </inactive-resolution>
```

Notes:

- *After importing to Report Center, identifiers of scanned QC items have the following formats:*
  - *For PR: BUG_(real QC ID)*
  - *For FR: REQ_(real QC ID)*
- *Currently, functionality to link from Report Center reports to open PR/FR details in QC is not available.*
- *The Task Activity report requires the addition of QC users to the Report Center database in order to display any data.*
- *This is an incremental scan, which means that only requirements/defects which were modified in QC since the last scan are updated to Development Testing Platform. To ensure that Development Testing Platform can see what items changed in QC, you need to have the proper settings in QC's Project Customization/Project Entities forms. All fields which are listed in the "Provide a fields-mapping configuration" section should have an active History property in QC.*

Connecting to Quality Center's Database using Windows Authentication

The Development Testing Platform services must be installed and running on a Windows environment in order to connect to the MSSQL Server via Windows Authentication (because the official MSSQL Server JDBC driver only uses the credentials from the user account that's currently running the services for Windows Authentication).

To prepare for the Development Testing Platform setup:

1. Download the official MSSQL Server JDBC driver package, then follow the provided instructions to unload the package.
2. Open the **Start** menu, then right-click **My Computer** to open the menu. Click **Manage** to open the Computer Management window. (Alternatively, open the Control Panel from the **Start** menu, then go to **Administrative Tools> Computer Management**.)
3. In the right pane, expand **Services and Applications**, then double-click **Services** to open the list of Windows Services.
4. Right-click the **Parasoft Development Testing Platform**, then choose **Properties** to open the Parasoft Development Testing Platform Properties window.
5. Open the **Log On** tab. Select the **This account** option, then provide a Windows user account in the format `LOCATION\username`. This user account must be allowed access to the MSSQL Server database.
6. Provide the password and the user account in the next two fields, then click **OK**.
7. With the Parasoft Development Testing Platform service still selected in the Services list, click the **Stop the service** link to stop the Development Testing Platform service.
8. Open the `${DTP_HOME}/tomcat/lib` directory in a new window.
9. Create a copy of the sqljdbc4.jar file for backup purposes.

10. Copy the sqljdbc4.jar file from <MSSQL_JDBC_DRIVER>\sqljdbc_<VERSION>\<LANG> sqljdbc4.jar and into the current directory.
11. Access the `${DTP_HOME}/jre/bin` directory and add in the `sqljdbc_auth.dll` file from the `<MSSQL_JDBC_DRIVER>\sqljdbc_<VERSION>\<LANG>\auth\<ARCHITECTURE>` directory .

To set up Development Testing Platform:

1. Access the `${DTP_HOME}/grs/config/bts` directory and open the QC BTS Scanner configuration file.
2. Open the BTS Scanner configuration file, then find the `<db-connection>` element. If the configuration file is not already configured, follow the instructions from "Using Direct HP QC Database Access", page 235 to properly modify the configuration file.
3. For the URL value in the `<url>` element within in the `<db-connection>` element, append the property `integratedSecurity=true`.
4. Make sure the settings resemble the example provided below, then save the changes.

```
<db-connection>

<db-type>MSSQL</db-type>
<user>sa</user>
<password encrypted="true">abc123</password>
<url>jdbc:sqlserver://localhost;integratedSecurity=true;</url>

</db-connection>
```

5. Start the Parasoft Development Testing Platform service from the Services list by clicking the **Start the service** link.
6. Follow the instructions from "Verifying that BTS Scanner is Working", page 225 to validate that QC Scanner has been configured properly.

Using Open Test Architecture

BTS Scanner for QC using OTA API allows you to import requirements and defects from the QC database to Development Testing Platform. QC requirements are shown as Project Center Requirements and QC defects are shown as Project Center Bugs. Imported requirements and defects are in read-only mode: you cannot modify most attributes.

If the QC client is installed on the same machine where Development Testing Platform is installed, you can also update QC Requirements with data retrieved from Parasoft Development Testing Platform.

To configure integration using Open Test Architecture (OTA):

1. Provide the following general settings:

Setting	Description
<bts>/<type>	Should be set to "TestDirectorOTA".

Setting	Description
<bts>/<name>	The name you want to use to label this instance of your HP QC server (for example, "Test Director"). Each HP QC server instance must have a unique name. This name will be shown in Project Center's Search Defects/Enhancements page (in the Defect Tracking System drop-down menu).

For example:

```
<bts type="TestDirectorOTA">
<name>QC OTA</name>
```

2. Provide Quality Center database-connection settings.

   Since data for each QC project is stored in a separate database, you are required to provide at least one database name to be scanned.

   The name of databases for each project can be accessed from Quality Center Site Administration, or from any MS SQL client.

   Note: *It is recommended that you write down the names of the defined QC databases. You will need this information for database connection settings.*
   The following QC database connection settings are required for QC integration:

Setting	Description
<bts>/<db-connection>/<db-type>	Database vendor: MS SQL only.
<bts>/<db-connection>/<user>	Database user.
<bts>/<db-connection>/<pass>	Database password.
<bts>/<db-connection>/<url>	Quality Center database connection URL. **Note**: Do NOT include database name in the connection URL.
<bts>/<db-connection>/<database>	The QC database name. At least one database name is required. Additionally, it is required to specify the following attributes for the <database> tag: • qc-domain - specify QC Domain for qc-project • qc-project - specify QC project name • dtpDevelopment Testing Platform-project - specify Development Testing Platform project name under which the scanned requirements will be stored. **Note**: If the specified project is not found in the Development Testing Platform database during the scanning process, it will be created automatically. qc-domain, qc-project, dtp-project attributes are used to create the Requirement properly.

For example:

```
<db-connection>
<db-type>MSSQL</db-type>
<user>sa</user>
<password encrypted="true">abc123</password>
<url>jdbc:sqlserver://localhost;</url>
<database qc-domain="DEFAULT" qc-project="QualityCenter_Demo"
dtp-project="MyProject">QualityCenter_Demo_db
</database>
</db-connection>
```

3. Configure the ota-connection.

   To enable QC defects to be automatically updated with Parasoft Development Testing Platform data, you need to specify credentials that make communication possible using OTA API. To do this, provide the following general settings:

Setting	Description
<bts>/<ota-connection>/<user>	Name of a user that can communicate using OTA API.
<bts>/<ota-connection>/<password>	Valid password for the specified user.
<bts>/<ota-connection>/<url>	Valid HTTP URL to QC.

   For example:

```
<ota-connection>
<user>sa</user>
<password>sa</password>
<url>http://localhost:8080/qcbin</url>
</ota-connection>
```

Integrating with Bugzilla

Integration has been tested with the following Bugzilla versions:

- 2.16
- 2.18
- 2.20
- 2.22
- 3.0
- 3.4
- 3.6
- 4.0
- 4.2

The BTS Updater works with Bugzilla 3.4 and later.

See “Configuring BTS Updater”, page 226 for instructions on using the built-in UI to integrate Bugzilla.

The following settings should be customized in the DTP\grs\config\bts\bugzila.xml file in order to configure Bugzilla scanning:

1. Provide the following general settings:

Setting	Description
<bts>/<name>	The name you want to use to label this instance of your Bugzilla server (for example, "My Bugzilla Server."). Each Bugzilla server instance must have a unique name. This name will be shown in Project Center’s Search Defects/ Enhancements page (in the Defect Tracking System drop-down menu).
<bts>/<url-prefix>	Prefix needed to create bug details links from your Report Center server to your Bugzilla server.
<bts>/<version>	Specifies the version of Bugzilla with which Development Testing Platform is integrating.

2. Provide Bugzilla database-connection settings:

Setting	Description
<bts>/<db-connection>/<db-type>	Database vendor: either MySQL or Oracle.
<bts>/<db-connection>/<user>	Database user.

Setting	Description
<bts>/<db-connection>/<pass>	Database password.
<bts>/<db-connection>/<url>	Bugzilla database connection URL.

For example:

```
        <bts type="Bugzilla">
                <name>My Bugzilla Server</name>
                <url-prefix>http://mybugzillaserver/bugzilla/
show_bug.cgi?id=</url-prefix>
        <version>3.4</version>
        <db-connection>
                <db-type>MySQL</db-type>
                <user>bugs</user>
                <password encrypted="true">abc123</password>
                <url>jdbc:mysql://bugzilla.host.com:3306/bugs</url>
<!-- sample Oracle connection url: <url>jdbc:oracle:thin:@HOST:PORT:SID</
url> -->
 <!-- sample MySQL connection url: <url>jdbc:mysql://HOST:PORT/DATABASE</
url> -->
        </db-connection>
        </bts>
```

3. Provide values for resolved statuses and inactive resolutions.

   In other words, mark which status value indicates that the bug is resolved and which resolutions describe an inactive bug state. Without this key information, Report Center is unable to create bug history reports.

   For Example:

```
        <resolved-status>
                <status>VERIFIED</status>
                <status>RESOLVED</status>
                <status>CLOSED</status>
        </resolved-status>

        <inactive-resolution>
                <resolution>INVALID</resolution>
                <resolution>WONTFIX</resolution>
                <resolution>DUPLICATE</resolution>
                <resolution>MOVED</resolution>
                <resolution>REMIND</resolution>
                <resolution>LATER</resolution>
                <resolution>WORKSFORME</resolution>
        </inactive-resolution>
```

4. Specify which value for Bugzilla's severity field indicates that a bug is an enhancement, using the `<feature-request>` setting. The severity value that indicates an enhancement to Development Testing Platform varies according to your version of Bugzilla:

   For Bugzilla versions 3.0 and earlier, the severity value that indicates an enhancement is "Feature_Req", so the configuration should be:

```
<feature-request>
        <item>Feature_Req</item>
</feature-request>
```

For Bugzilla version 3.4, the severity value that indicates an enhancement is "enhancement", so the configuration should be:

```
<feature-request>
        <item>enhancement</item>
</feature-request>
```

Integrating with IBM Rational ClearQuest

Integration has been tested with the following ClearQuest versions:

- 2003.06.15.734.000
- 2007
- 7.0.1.1

Before integrating ClearQuest with Development Testing Platform, ensure that the ClearQuest client program (the `cqperl` command) is installed on the same machine as DTP. BTS Scanner uses the `cqperl` command to read items from ClearQuest.

Make a copy of the `ExampleClearQuestScannerConfig.xml` file located in the following directory:

```
${DTP_HOME}\grs\config\bts\examples\
```

To configure ClearQuest scanning, make the following adjustments to the copied file:

1. Provide the following general settings:

Setting	Description
<bts>/<name>	The name you want to use to label this instance of your ClearQuest server (for example, "My ClearQuest Defects Server"). Each ClearQuest server instance must have a unique name. This name will be shown in Project Center's Search Defects/Enhancements page (in the Defect Tracking System drop-down menu).
<bts>/<connection-settings>/<dbset>	ClearQuest database set.
<bts>/<connection-settings>/<dbname>	ClearQuest database name.
<bts>/<connection-settings>/<user>	User name.
<bts>/<connection-settings>/<pass>	User password.

2. Define the ClearQuest entity name that you would like BTS Scanner to read, using the <run-options> setting:

Setting	Description
<run-options>/<entity-type>	The name of the entity in your ClearQuest database which stores info about items (defects, change requests, etc.) you plan to import. BTS Scanner will scan items of this entity.

Note: You must provide a separate BTS Scanner configuration file for each <entity-type> you would like to import from ClearQuest.

3. Specify if you would like Development Testing Platform to treat some or all ClearQuest entities as enhancements. (Optional)

BTS Scanner can be configured to import some or all entities from ClearQeust as either defects or enhancements. However, entities read from ClearQuest are treated as defects by default.

BTS Scanner uses an imported entitiy's "severity" field value to decide whether to treat it as an enhancement.

To treat some or all ClearQuest entities as enhancements, perform the following steps:

- *Prepare a BTS Scanner configuration file, as explained in* "Preparing an Example Configuration File", page 221 *and* "Adding a Prepared Configuration File", page 222*.*
- *Define the <entity-type> you would like to import as enhancements, using whatever name the entity has in Clear Quest*

  For example, assuming ClearQuest stores change requests as "cr" entities, you would define <entitiy-type> in the following way:

```
<run-options>
        <entity-type>cr</entity-type>
</run-options>
```

- *Perform one of the following:*

  To treat some ClearQuest entities as enhancements:

  - Use the <feature-request> tag to specify what items read from ClearQuest should be treated as enhancements.

    For example:

```
<feature-request>/<item> 6-Enhancement </feature-
request>
```

    Using the above configuration, all items which have a severity field-value of "6-Enhancement" would be imported to Development Testing Platform as enhancements, while the rest would be imported as defects.

  To treat all ClearQuest entities as enhancements:

  - Use the <feature-request> tag to specify all possibilities of Severity field values.

    For example:

```
<feature-request>
        <item>1-Critical</item>
        <item>2-High Attention</item>
        <item>3 -Major</item>
        <item>4-Average</item>
        <item>5-Minor</item>
        <item>6-Enhancement</item>
</feature-request>
```

4. Provide ClearQuest database fields-mapping.

BTS Scanner can only import information from ClearQuest bug fields that have been mapped to the proper Report Center fields. Unmapped fields are treated as unknown and will be skipped during the translation process.

> ***Note:*** *If you are not aware what table fields are present in your ClearQuest database, you can run the following script to print the db table fields of one sample item from the ClearQuest db:* `read_single_bug.pl` *This script is located in* `DTP\grs\extras\ClearQuest`.

Before running the script, you should open it and edit the login credentials and sample defect ID.

For example, the "Headline" field in ClearQuest corresponds to the "Summary" field in Report Center, while the "Submitter" field in ClearQuest corresponds to "Reporter" in Development Testing Platform.

Some fields are required for ClearQuest integration, while others are optional.

The following fields-mapping settings are required:

Field	Description
<bts>/<fields-mapping>/<summary>	Bug summary description.
<bts>/<fields-mapping>/<status>	Current status, such as open or closed.
<bts>/<fields-mapping>/<resolution>	Bug resolution, such as fixed or won't fix.
<bts>/<fields-mapping>/<priority>	Bug priority, such as low or medium.
<bts>/<fields-mapping>/<severity>	Bug severity, such as minor or critical.
<bts>/<fields-mapping>/<reporter>	User who reported the bug.
<bts>/<fields-mapping>/<creation-date>	Bug creation date.

The following fields-mapping settings are optional:

Field	Description
<bts>/<fields-mapping>/<project>	A project to which the bug belongs.
<bts>/<fields-mapping>/<version>	Project version.
<bts>/<fields-mapping>/<milestone>	Project milestone name.
<bts>/<fields-mapping>/<hardware>	Hardware affected by this bug.
<bts>/<fields-mapping>/<os>	OS affected by this bug.
<bts>/<fields-mapping>/<component>	Project component.
<bts>/<fields-mapping>/<modification-date>	Last bug modification date.
<bts>/<fields-mapping>/<assigned-to>	User to whom the bug is currently assigned.

For example:

```
<fields-mapping>
        <summary>Headline</summary> *
        <status>State</status> *
        <resolution>Resolution</resolution> *
```

```
        <priority>Priority</priority> *
        <severity>Severity</severity> *
        <reporter>Submitter</reporter> *
        <creation-date>Submit_Date</creation-date> *
        <project>Project</project>
        <version>Version</version>
        <milestone>Milestone</milestone>
        <hardware>Hardware</hardware>
        <os>OS</os>
        <component>Component</component>
        <modification-date>Mofification_Date</modification-date>
        <assigned-to>Owner</assigned-to>
</fields-mapping>
```

Note: Required columns are marked with an asterisk. If you provide fields that are not required, you gain the ability to add filters to Report Center projects based on values of these fields.

5. Provide values for resolved statuses and inactive resolutions.

   In other words, mark which status value indicates that the bug is resolved, and which resolutions describe an inactive bug state. Without this key information, Report Center is unable to create bug history reports.

   For example:

```
<resolved-status>
        <status>Resolved</status>
        <status>Closed</status>
</resolved-status>

<inactive-resolution>
        <resolution>Duplicate</resolution>
        <resolution>Functions as Designed</resolution>
</inactive-resolution>
```

6. The following settings should also be configured:

   - `<date-formats>`

     Ensure that the date formats which are used in your ClearQuest are on the below list. If not please add the appropriate format. This information is required for Development Testing Platform to properly parse date data when reading items from ClearQuest.

     The default configuration appears as follows:

```
    <date-formats>
            <date-format>MMM-dd-yyyy HH:mm</date-format>
            <date-format>MM/dd/yyyy h:mm a</date-format>
            <date-format>yyyy-MM-dd HH:mm:ss</date-format>
    </date-formats>
```

     Note: *The format of entries should meet the one defined for SimpleDateFormat class in java (see http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html).*

   - `<bugs-to-read>`

     This setting is used for testing-purposes only. When testing your settings, you can specify how many items BTS Scanner should import before stopping. For example, specifying 10 would cause BTS Scanner to stop after importing 10 items.

Note: *When 0 is set, all items are read from ClearQuest.*

- `<retries-on-error>`

  There might be rare occasions when one call of cqperl (which is used by BTS Scanner) fails to connect to ClearQuest. This setting lets BTS Scanner repeat cqperl invocation in case of a connection problem.

- `<scan-changed-from>, <scan-changed-to>`

  This setting tells BTS Scanner to only scan items modified between the specified dates.

Integrating with Atlassian JIRA

Development Testing Platform integration with JIRA has been tested with the following versions:

- 3.6.2
- 3.7.3
- 3.10.1
- 3.10.2
- 3.12
- 3.13
- 4.1.1
- 4.1.2
- 4.3
- 4.4
- 5.x
- 6.x.

Minor adjustments to JIRA and Development Testing Platform are required to configure BTS Scanner to work with JIRA. These adjustments are detailed in the following sections:

- Configuring JIRA Server Side
- Configuring JIRA in Development Testing Platform

BTS Scanner can also be configured to work with JIRA over HTTPS/SSL. To do so, see the following section:

- Configuring BTS Scanner for JIRA over HTTPS/SSL

Integrating 5.x and 6.x

Use the built-in UI (see “Configuring BTS or RMS Scanner with the Built-in UI”, page 215 for instructions) for integration with JIRA 5.x and 6.x. SSL must be enabled on the DTP server to integrate with JIRA 5.x/6.x. See “Running Development Testing Platform Under SSL”, page 153, for more information.

Integrating with JIRA 4.4 and Older

The following instructions describe how to configure JIRA 4.4 or earlier. You can use the built-in UI to configure Development Testing Platform for JIRA 4.x, but you will still need to configure JIRA server side. See instructions below.

If you already have a JIRA 4.x scanner configured, e.g. a JIRA 4.x scanner configured in a version of Development Testing Platform (formerly called Concerto) prior to 4.9.3, Development Testing Platform will not allow you to configure an additional JIRA 5.x scanner through the UI. This is not a limitation, but a restriction designed to prevent duplicate JIRA issues in Development Testing Platform.

If you are interested in migrating your existing JIRA 4.x scanner to JIRA 5.x scanner, please contact Parasoft Support. There is a manual process that involves making a small update to the database that will enable the migration.

Configuring JIRA Server Side

Before configuring JIRA, you must first verify that the SOA interface is enabled in your JIRA server. For information about enabling JIRA remote interface, see the topic on enabling the RPC plug-in in JIRA's documentation: https://confluence.atlassian.com/display/ALLDOC/Atlassian+Documentation.

1. Make a a backup the existing plug-in by renaming `atlassian-jira-rpc-plugin.jar` to `atlassian-jira-rpc-plugin.jar.orig.bak` in the `JIRA_INSTALL_DIR/atlassian-jira/WEB-INF/lib` directory.
2. Copy the new extended jar (shipped with Development Testing Platform) onto the original. Jar files for supported versions are in the `[DTP_HOME]\grs\extras\jira` directory:

Version	Jar
3.6.*	`[DTP_HOME]\grs\extras\jira\rpc-plugin.jar`
3.7.3	`[DTP_HOME]\grs\extras\jira\atlassian-jira-rpc-plugin-3.12.1-1.jar`
3.10	`[DTP_HOME]\grs\extras\jira\atlassian-jira-rpc-plugin-3.10.2-1.jar`
3.12	`[DTP_HOME]\grs\extras\jira\atlassian-jira-rpc-plugin-3.12.1-1.jar`
3.13	`[DTP_HOME]\grs\extras\jira\atlassian-jira-rpc-plugin-3.13.1-1.jar`
4.1.2	`[DTP_HOME]\grs\extras\jira\atlassian-jira-rpc-plugin-4.1.2.jar`
4.3	`[DTP_HOME]\grs\extras\jira\atlassian-jira-rpc-plugin-4.3.jar`
4.4	`[DTP_HOME]\grs\extras\jira\atlassian-jira-rpc-plugin-4.4.jar`

   See "BTS and RMS Scanner Configuration", page 184 for information about integration with JIRA 5.x.
3. Restart JIRA server.

Development Testing Platform Ships with an Extended JIRA SOA

The JIRA SOA interface shipped with Development Testing Platform is extended to include an additional method in the `[DTP_HOME]\grs\extras\jira` directory, which retrieves defects history.

The implementation details are documented in JIRA's Agile Board: `http://jira.atlassian.com/browse/JRA-10333`)

`JIRA_INSTALL_DIR/atlassian-jira/WEB-INF/lib/atlassian-jira-rpc-plugin.jar` contains an implementation of SOA access to JIRA. We recommend notifying your JIRA administrator about the additional SOA method.

Development Testing Platform uses other JIRA SOA methods and none of the original JIRA SOA methods are changed. As a result, replacing the jar only extend the SOA interface and will not cause a regression in JIRA.

Though not included in a normal JIRA distribution, this extension has received initial approval from Atlassian.

Configuring JIRA in Development Testing Platform

1. Enter the following settings in the `[DTP_HOME]\grs\config\bts\Jira.xml` file in order to configure Jira scanning:

Setting	Description
`<bts>/<name>`	The name you want to use to label this instance of your JIRA server (for example, "My JIRA Server."). Each JIRA server instance must have a unique name. This name will be shown in Project Center's Search Defects/Enhancements page.
`<bts>/<url-prefix>`	Prefix needed to create bug details links from Report Center server to your JIRA server.
`<bts>/<connection-settings>/ <soap-service>`	JIRA SOAP service address, usually in the following form: http://<YOUR_JIRA_SERVER_ADDRESS>:<SERVER_PORT>/rpc/soap/jirasoapservice-v2</soap-service
`<bts>/<connection-settings>/ <pass>`	JIRA user password.
`<bts>/<connection-settings>/ <timeout>`	Specifies the BTS Scanner/JIRA connection timeout, in milliseconds. If not specified, default axis timeout will be used.

   For example:

```
<connection-settings>
      <soap-service>http://localhost:8080/rpc/soap/jirasoapservice-v2
      </soap-service>
            <user>root</user>
            <pass>root</pass>
            <timeout>720000</timeout>
</connection-settings>
```

2. Define values for resolved statuses and inactive resolutions. Without this key information, Report Center is unable to create bug history reports.

   For example:

```
<resolved-status>
     <status>Resolved</status>
     <status>Closed</status>
</resolved-status>

<inactive-resolution>
     <resolution>Duplicate</resolution>
     <resolution>Won't Fix</resolution>
</inactive-resolution>
```

3. (Optional) Specify the scope for scanning JIRA issues.You can read all issues from JIRA, issues restricted by projects (specified by JIRA project keys), or issues defined by user-defined JIRA filters (specified by JIRA filter IDs). Pagination can be used when reading issues for a filter (e.g., you could set up the scanner to read 1000 issues for each request to JIRA [SOAP

call]). Pagination can be used to avoid out of memory issues when there are very many items in JIRA to be scanned (e.g., 10000+) in a single JIRA request.
For example:

```
<scanner-scope>

          <!--
              Scan Jira items which belong to the following Jira filters
          -->
          <jira-filters enable-paging="true" items-per-page="1000">
              <filter-id>10035</filter-id>
              <filter-id>10036</filter-id>
          </jira-filters>

          <!--
              Scan the following Jira projects (project keys)
             If <jira-filters> is specified, this <jira-projects> section is skipped
          -->
          <jira-projects>
              <project-key>PROJECTONE</project-key>
              <project-key>PROJECTTWO</project-key>
          </jira-projects>

</scanner-scope>
```

- `enable-paging` indicates whether reading pagination is enabled (`true`) or disabled (`false`).
- `items-per-page` specifies how many issues should be read at once (`1000` specifies to read 1000 issues for each JIRA request).
- `<filter-id>` specifies the id of the filter created in JIRA. The filter definition created on JIRA should be visible to the user which who will be connecting the BTS Scanner configuration to JIRA. After this filter is created in JIRA, you can obtain this id from JIRA. First, go to your JIRA web page and choose **Issues> Manage Filters**. On the Manage Filters page, click the **edit** link for the filter. Next, look this page URL and find the `filterId` parameter in this URL. For example, if you see `http://xen3.parasoft.com.pl:8080/secure/EditFilter!default.jspa?atl_token=aO8P-8w_fk&filterId=10032&returnUrl=ManageFilters.jspa`, then the `filterId` is `10032`.

  If it is not please share filter to other users in JIRA

4. (Optional) Specify JIRA custom types mapping.

   By default, when importing JIRA items, Development Testing Platform treats JIRA "New Feature" and "Improvement" types as feature requests in Development Testing Platform, and JIRA "bug" type as a bug in Development Testing Platform. This is predefined in the following portion of XML:

   ```
   <imported-issue-type>
       <issue-type translate-to="BUG">Bug</issue-type>
       <issue-type translate-to="FEATURE_REQUEST">New Feature</issue-type>
   </imported-issue-type>
   ```

   - *When you have custom types defined in JIRA, you can use this node to import them and map them into Development Testing Platform bugs or feature requests. For example, if you have your JIRA custom type as "My Defect" it could be mapped to Development Testing Platform bugs with the following configuration:*

     ```
     <imported-issue-type>
     ```

```
<issue-type translate-to="BUG">My Defect</issue-type>
<issue-type translate-to="BUG">Bug</issue-type>
<issue-type translate-to="FEATURE_REQUEST">New Feature</issue-type>
<issue-type translate-to="FEATURE_REQUEST">Improvement</issue-type>
     </imported-issue-type>
```

- *If there is no <imported-issue-type> node in config xml, the default configuration will be used for backwards compatibility.*
- *If the <imported-issue-type> node is empty, no JIRA items will be imported.*

Configuring BTS Scanner for JIRA over HTTPS/SSL

To set up BTS Scanner to work with JIRA over HTTPS/SSL, configure BTS Scanner for JIRA as you normally would, then perform the steps listed in the following sections:

- Adjusting BTS Scanner Configuration
- Running DTP Service with the Keystore Parameter

Adjusting BTS Scanner Configuration

In SOAP service address, make the following adjustments:

1. Replace http with https
2. Set the proper SSL port (usually 443 or 8443)

   For example:

```
<connection-settings>
          <soap-service>https://jira.somecompany.com:8443/rpc/
soap/jirasoapservice-v2</soap-service>
          <user>root</user>
          <pass>root</pass>
</connection-settings>
```

Running DTP Service with the Keystore Parameter

1. Obtain the SSL key file which is used by JIRA tomcat.

   If needed, you can learn more about this process at:

   `http://confluence.atlassian.com/display/JIRA/Running+JIRA+over+SSL+or+HTTPS` (JIRA documentation)

   and:

   `http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#CreateKeystore`
2. Store the key file on the Development Testing Platform host in any directory.
3. Provide a Java -Djavax.net.ssl.trustStore=<TRUSTED KEYSTORE PATH> parameter for Development Testing Platform server pointing to the SSL key file.

For the following examples, let's assume that the key filename is `.keystore`:

For Linux, assuming you have placed your key file in `/home/ser/.keystore`:

1. Edit `DTP_HOME/bin/reportserver.sh`.
2. Replace:

   ```
   export CATALINA_OPTS="-D$PCC_RECOGNITION \
   ```

   with:

   ```
   export CATALINA_OPTS="-D$PCC_RECOGNITION -Djavax.net.ssl.trustStore=/
   home/ser/.keystore \
   ```

For Windows, assuming you have the key file in `D:\home\user\dtp\.keystore:`

1. Open Windows Registry Editor by running regedit command
2. Find Parasoft Development Testing Platform service invocation parameter.

   This can typically be found at `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Parasoft Concerto\Parameters`
3. Add new JVM Option Number parameter.

   For example, if your installation has 15 such parameters (numbered from 0 to 14) add the 16th parameter as:

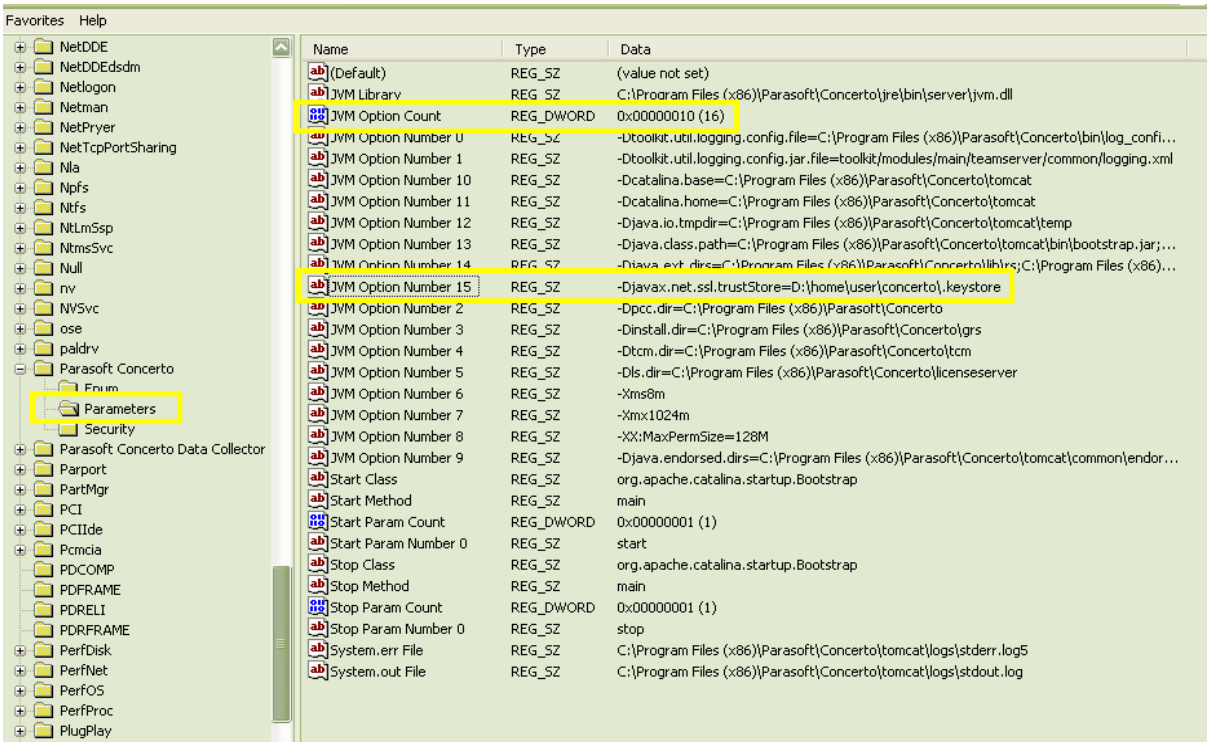
   ```
   JVM Option Number 15
   ```

   value:

   ```
   -Djavax.net.ssl.trustStore=D:\home\user\concerto\.keystore
   ```

4. Increase the parameters counter "JVM Option Count" by one. For example, if the value of "JVM Option Count" is 15, set it to 16:

Figure 5: Editing JVM Option Count with Regedit



Configuring BTS Scanner for JIRA with Basic HTTP Authentication

To set up BTS Scanner to work with JIRA with basic HTTP authentication, configure BTS Scanner for JIRA as you normally would, then perform the adjustment listed below:

Add basic-http-authentication to connection-settings in the ${DTP_HOME}\grs\config\bts\Jira.xml file:

```
<basic-http-authentication>
        <user>admin</user>
        <pass encrypted="false">admin</pass>
</basic-http-authentication>
```

For example:

```
<connection-settings>
        <soap-service>https://jira.somecompany.com:8443/rpc/soap/
jirasoapservice-v2</soap-service>
        <user>root</user>
        <pass>root</pass>
        <basic-http-authentication>
                <user>admin</user>
                <pass encrypted="false">admin</pass>
        </basic-http-authentication>
```

```
</connection-settings>
```

Note: User and pass inside <basic-http-authentication> tags are sent to your JIRA server as an HTTP header parameter.

Integrating with IBM Rational Change and Synergy

This section explains how to import Rational Change artifacts into Development Testing Platform and how to configure Task Assistant to work with Rational Change artifacts imported into Development Testing Platform (in “Configuring Parasoft Test Task Assistant to work with Development Testing Platform and Rational Change Synergy”, page 262).

Rational Change Integration

BTS Scanner for Rational Change and Synergy allows you to import change requests (and the associated tasks) from the Rational Change to Development Testing Platform. The Rational Change change requests are presented as requirements in Development Testing Platform. As is standard with BTS integration, Rational Change and Synergy synchronization is performed by the BTS/RMS Scanner, which is run automatically or can be run on demand.:

1. Make a copy of the DTP_HOME/grs/config/bts/examples/ExampleSynergyScannerConfig.xml file (which presents sample settings for Synergy) and adjust the copy’s settings to suit your Synergy server and Development Testing Platform server. Each setting element is described in the XML file.
2. Copy this file to DTP_HOME/grs/config/bts. After this is completed, your specified Rational Change server will be connected and the “ccm” queries specified in the .xml file will be executed every 15 minutes—or when you click the **RESCAN** button in Project Center Requirements page.

   Change requests found by the queries (as well as the change requests’ associated children and tasks) are synchronized with the requirements and tasks in the Development Testing Platform target project.

   Any changes made in Rational Change are updated in Development Testing Platform as follows:

   - *If a specific requirement (or task) does not exist in Development Testing Platform, it is added.*
   - *If a specific change request (or task) has been modified in Ration Change, it is updated in Development Testing Platform. The Rational Change change request ID and task ID are stored in the Development Testing Platform requirement Original ID and task Original ID fields (respectively).*

Note

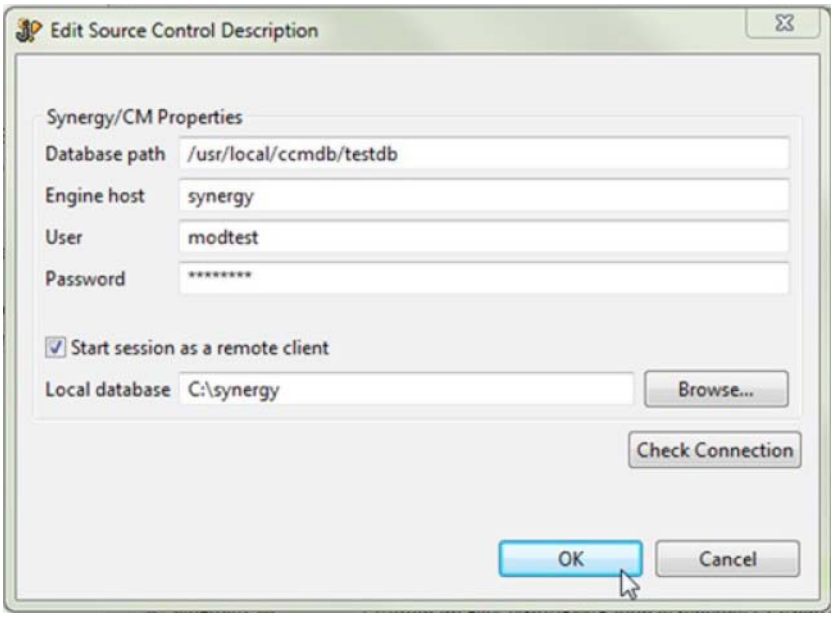
- Synergy "ccm" queries configured in .xml settings file are tightly integrated with the Development Testing Platform target project. For example, when you modify or delete a query (so that a specific configuration returns fewer change requests), the outstanding requirements in the Development Testing Platform target project are deleted. All tasks associated with the deleted requirements change their type to 'Standalone.' Tasks are never deleted.
- The Development Testing Platform requirement's Original ID and target Development Testing Platform project are taken into consideration to find counterparts in Development Testing Platform/Rational Change. For example, if an imported requirement with a specific Original ID does not exist in a specific Development Testing Platform target project, it will be created in Development Testing Platform. If such a requirement exists in the Development Testing Platform target project, it will be updated.
- In some cases, you might have "ccm" queries defined to be synchronized with a Development Testing Platform project, then change the configuration Development Testing Platform target project name...but leave the queries so that they import the same change request into another Development Testing Platform project. When a new (not yet imported into Development Testing Platform) or modified (since the previous Development Testing Platform import) Rational Change change request is going to be imported into Development Testing Platform (i.e. it is retrieved by the "ccm" query)—if a requirement with the same Original ID already exists in Development Testing Platform, but is in another project, then:
  - *It will be deleted from that original project.*
  - *All tasks associated with the deleted requirement will change their type to 'Standalone.' No tasks will be deleted.*
  - *It will be imported to the new project.*
  - *If this imported requirement has associated tasks, they will be either created as new or will be moved to this requirement based on the Original ID.*

3. For each change request being retrieved by query, the associated children requirements and tasks are also synchronized. For instance, if a new task is added to a change request in Rational Change, it will also be added to Development Testing Platform upon synchronization.

At this point, you should be able to see requirements and their tasks in the Development Testing Platform target project. Note that the imported requirements and tasks in Development Testing Platform's Original ID fields contain IDs from Rational Change.

Configuring Parasoft Test Task Assistant to work with Development Testing Platform and Rational Change Synergy

1. Configure the Development Testing Platform project into which you have imported data from Synergy as follows:
   a. In Parasoft Test, get the text properties pointing to your Synergy source repository. To do this, define your Synergy source control in Parasoft Test



then export these settings to a file. Here is an example of the exported properties to be pasted into the Development Testing Platform project Parasoft Test Settings tab:

```
scontrol.rep1.synergy.dbpath=/usr/local/ccmdb/testdb
scontrol.rep1.synergy.host=synergy
scontrol.rep1.synergy.local_dbpath=C\:\\tmp\\local_synergy_db
scontrol.rep1.synergy.login=modtest
scontrol.rep1.synergy.password=6e6b703e6952746f
scontrol.rep1.synergy.remote_client=true
scontrol.rep1.type=synergy
scontrol.synergy.exec=ccm
```

Such settings are used by Parasoft Test when it connects to the named Development Testing Platform project; they are project specific.

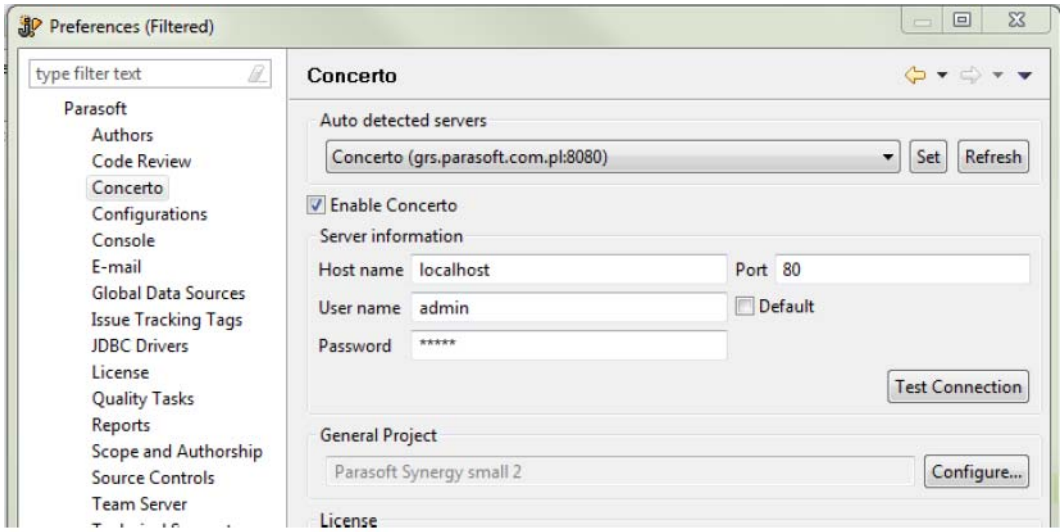
b. From the Development Testing Platform administration pages, open and edit the project you are working with, open the **Parasoft Test Settings** tab, then copy the above properties. If you want to specify these settings globally (for all Development Testing Platform projects), you can paste this information in the **Administration> Settings > Parasoft Test** page rather than the project's Parasoft Test Settings page.

c. Go to the **Administration> Settings> Parasoft Test** page, and paste the following line into the **Global Parasoft Test Settings** field:

```
tasksImportedMode=true
```

This setting switches Task Assistant (which will connect to this Development Testing Platform server) to work in Synergy mode (i.e. operate on Task Original IDs [values of task ID from Synergy are presented in Development Testing Platform as task Original IDs] as if they were task IDs).

2. On the Development Testing Platform server host, install the Synergy client. Since Development Testing Platform uses the ccm command to read files that are associated with tasks in Synergy (Development Testing Platform reads those files when it is queried by Task Assistant).
3. Ensure that:
    - *the “ccm” command is on the system path visible to the Development Testing Platform server service.*
    - *the path defined is not wrapped in quotes (“”), even if it contains spaces.*
4. In Parasoft Test, open the Development Testing Platform preferences, specify your Development Testing Platform server, and set **General Project** to the project that you configured and that has Synergy requirements and tasks imported.



Task Assistant is now ready to work with Synergy. This means that you can define a query to retrieve tasks from Development Testing Platform (which in turn retrieves them from Synergy). If specific task has some files associated with it in Synergy, activating that task in Task Assistant will prompt Development Testing Platform to read the associated file list from Synergy and present it in Task Assistant.

Troubleshooting

Cannot run program ""ccm""

In Parasoft Test’s Task Assistant I get the "Fail reason: Cannot run program ""ccm"": CreateProcess error=2," error when I activate a Synergy task.

Possible cause: This could be caused by any of the following:

- The Synergy client (including ccm command) is not installed on the Development Testing Platform server machine.
- The ccm command is not on the system path visible to the Development Testing Platform service.
- The path to the ccm.exe command on Windows is defined, but it is defined in quotes “”.

Solution: Ensure that the Synergy client (including the ccm command) is available on the Development Testing Platform server machine, the ccm command is on the system path visible to the Development Testing Platform service, and the path to the ccm command on Windows is not in quotes (even if the path contains spaces).

Cannot start new Synergy/CM session

I get the "Cannot start new Synergy/CM session. Warning: IBM Rational Synergy startup failed " error when I activate a Synergy task and go to Development Testing Platform Task Assistant view to see the associated files.

Possible cause: “ccm” command is visible to the Development Testing Platform server, but the system user who is running the Development Testing Platform service does not have permission to start a Synergy session.

Solution: Start the Development Testing Platform service as Windows system user who has permission to start Synergy sessions. For general instructions on how to change user who runs the Development Testing Platform service on Windows, see Solution 2 in “Show Source Functionality”, page 307.

Enabling Debug Mode

If a) you see the files associated with specific task in Synergy, but you cannot see them in Task Assistant, b) you get an error in Task Assistant when you queries for these files, and c) the above hints did not help, then switch Development Testing Platform server logging into debug mode. See “Switching to Debug Logging Mode”, page 309, for instructions.

Integrating with Code Review

Parasoft Development Testing Platform integrates with the code review process you set up through Parasoft Test (SOAtest, Jtest, etc.). In such a process, Parasoft Test serves as the code review engine and team members interact with code review tasks via Parasoft Test.

Development Testing Platform is used to store code review results and provide insight into code review results.

For details about how to set up and perform code reviews with Parasoft Test, see the Parasoft Test User's Guide.

Configuring Code Review Reporting for a Project

Code Review reports (described in "Code Review", page 61) show statistics for the selected project.

When generating Code Review reports, Development Testing Platform considers:

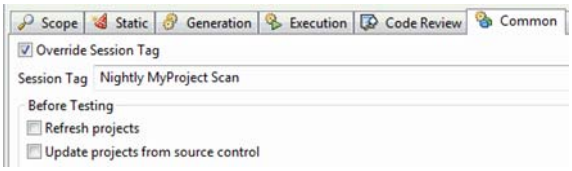
- Which results are associated with team members who are part of the selected project.
- Which results are marked with the identifier/session tag from the code review session.

To configure reporting:

1. Ensure that your project's Team Membership list contains the users who are involved in your code review process.
    - For more details on configuring team members, see "Project Creation and Configuration", page 164.
2. Ensure that project's Code Review Filter has the session tag set to match the Session Tag from your Parasoft Test Code Review Test Configuration.
    - For more details on configuring this filter, see "Code Review Filter", page 169

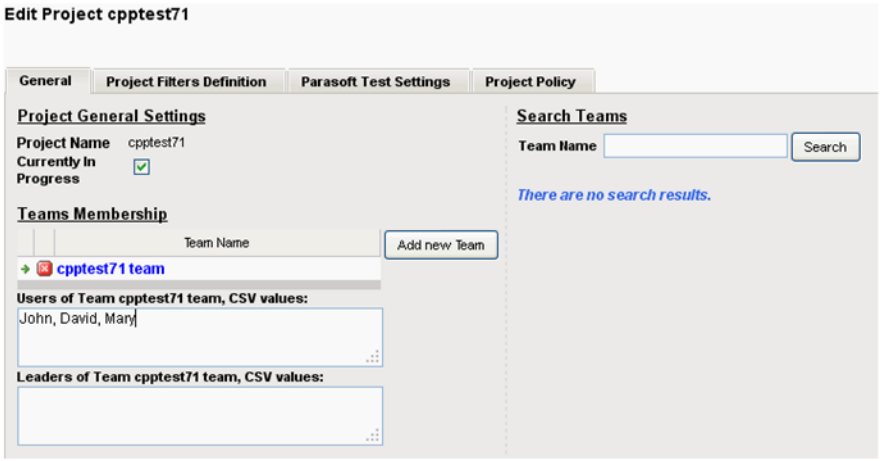
For example, assume that:

- You want to see code review results for John, David, and Mary
- Your team's Code Review Test Configuration uses the **Nightly MyProject Scan** Session Tag.

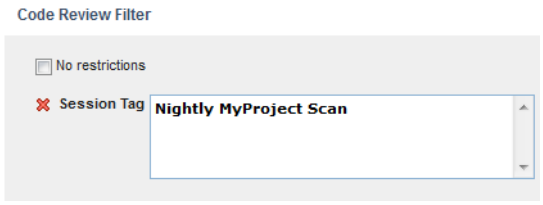


You would configure the following two things in Development Testing Platform:

- You would set the Team Membership list to include John, David, and Mary as follows:



- You would set the Code Review Filter to use the **Nightly MyProject Scan** session tag.



Configuring Code Review Data Storage

There are two ways to store code review data in Development Testing Platform:

- Using the Report Center database (recommended; provides more efficient storage)
- Using Team Server (supported for backwards compatibility)

The Development Testing Platform integration procedure varies depending on which code review storage option you are using.

For Report Center Storage

If you choose this (recommended) option, the code review configuration performed on Parasoft Test is sufficient. No additional configuration is needed on Development Testing Platform side.

Report Center reports are configured to draw data from the Report Center database by default.

For Team Server Storage

Report Center reports are configured to draw data from the Report Center database by default. To switch to Team Server-based code review storage:

1. Go to **Report Center> Administration> Settings> Report Center> Server** and enable **Use deprecated Team Server based Code Review**.
2. Configure a nightly job to read data from Team Server and put them into Report Center database by editing DTP_HOME/grs/config/CRHistoryScanners.xml to provide information on every Team Server server that should be queried for code review data. The following is an example:

```
<configuration>
    <cr-history-scanner>
        <teamserver>
            <host>teamserver1.company.com</host>
            <port>1111</port>
            <login>login</login>
            <password>pass</password>
            <timeout>7200000</timeout>
        </teamserver>
    </cr-history-scanner-->
    <cr-history-scanner>
        <teamserver>
            <host>teamserver2.company.com</host>
            <port>1111</port>
            <login>login2</login>
            <password>pass2</password>
            <timeout>7200000</timeout>
        </teamserver>
        </teamserver>
    </cr-history-scanner>
    ...
</configuration>
```

3. Restart the Development Testing Platform server so that changes take effect.

When a Report Center Code Review History Scanner job is launched, Team Servers from all `<cr-history-scanner/>` sections are sequentially scanned. Their data is then stored into the Report Center database.

This job is pre-configured and should be run as an independent group because the scanning operation can be time-consuming. The time-out parameter tells the scanner how many milliseconds are left until the scanning process is finished. A default value of 7,200,000 milliseconds should be set.

Warning: The first scan of the Team Server repository can take an extreme amount of time because the Report Center Code Review plugin for Team Server needs to create its indexes from scratch. Sometimes it takes a few scanning sessions to fully retrieve full CR operations history. Sequential scans (on the following days) should be noticeably shorter and work on a day-to-day basis.

Integrating Report Center with Emma

Report Center can integrate with the Emma Java code coverage tool. Emma's functional test coverage results are shown in Report Center reports. See "Viewing Emma Results", page 272.

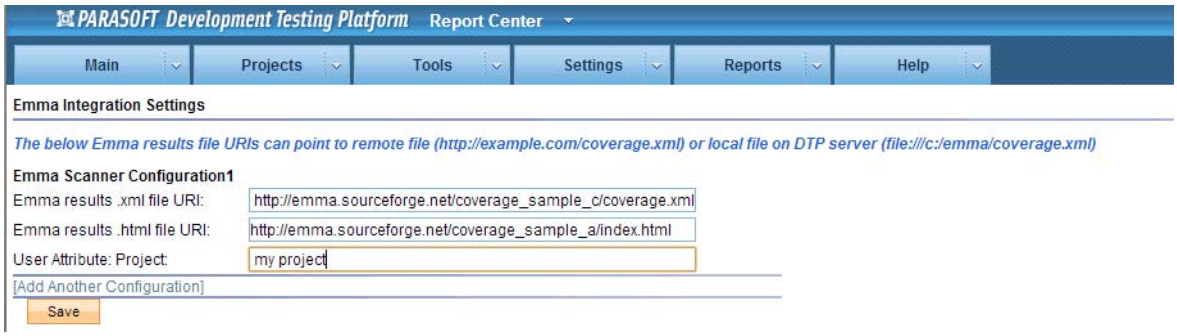
1. In Report Center, click the **administration** link in the top navigation bar.
2. Choose **Settings > Integration> Emma**.
3. Enter the URL to your Emma .xml file containing the coverage results for all of your tested classes.

   This URL location is scanned periodically by Report Center. When new results appear, they are loaded into the Report Center database.

   Note: *The .xml file for which you provide the URL should be served by an http server. You can see an example of .xml output at the official Emma Web site:* `http://emma.sourceforge.net/coverage_sample_c/coverage.xml`
4. (Optional) Enter the URL to your Emma .html coverage results file for all your tested classes. The HTML file should contain the results from your last Emma run.

   This URL location is not periodically scanned. Development Testing Platform links to Emma coverage reports from the Emma Coverage column of the Report Center Coverage report. See "Viewing Emma Results", page 272

   The .html file for which you provide the URL should be served by an http server. You can see an example Emma html report at the official Emma Web site: `http://emma.sourceforge.net/coverage_sample_a/index.html`
5. Specify a Test Group Property attribute filter in the User Attribute: Project field. The filter specifies which Development Testing Platform project the Emma results are associated with. The value of the filter is set in the Edit Project page. See "Test Group Properties Filter", page 167, for more information.



6. You can add additional configurations or click **Save** to continue.
7. Edit DTP_HOME\grs\config\CronConfig.xml file and go to the following xml node:

```
<Job    allowFromHour="0"    allowToHour="23"    frequency="1440"
id="Emma Scanner"runDayOfWeek="*" runHour="2" runMinute="0">
      <class allowInSlave="false"
name="com.parasoft.grs.rserver.cronjobs.EmmaScannerJob"
priority="1"/>
   </Job>
```

This xml node contains the configuration for a periodic Report Center background job, which scans the location defined in Step 1. It is predefined to be run each night at 2:oo a.m.

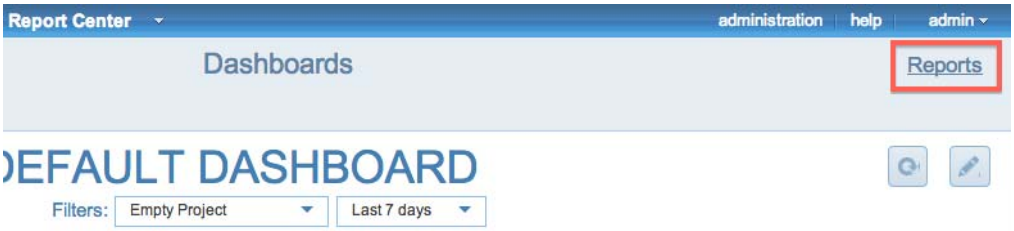
If necessary, you can modify the run hour in this xml node. Upon making the change, restart the Development Testing Platform server for the change to take effect.

8. Switch Report Center coverage reports to Emma mode.
    d. Edit the following files:
        - DTP_HOME/grs/xreports/architect_dashboard/CoverageDetails.xml
        - DTP_HOME/grs/xreports/architect_dashboard/CoverageOverview.xml
        - DTP_HOME/grs/xreports/architect_dashboard/CoverageOverviewDetails.xml
        - DTP_HOME/grs/xreports/architect_dashboard/composite_desc.xml
        - DTP_HOME/grs/xreports/lite_dashboard/composite_desc.xml
    e. Go to the following xml element:
        - `<parameter name="show_emma_results">false</parameter>`
    f. Set its value to true:
        - `<parameter name="show_emma_results">true</parameter>`

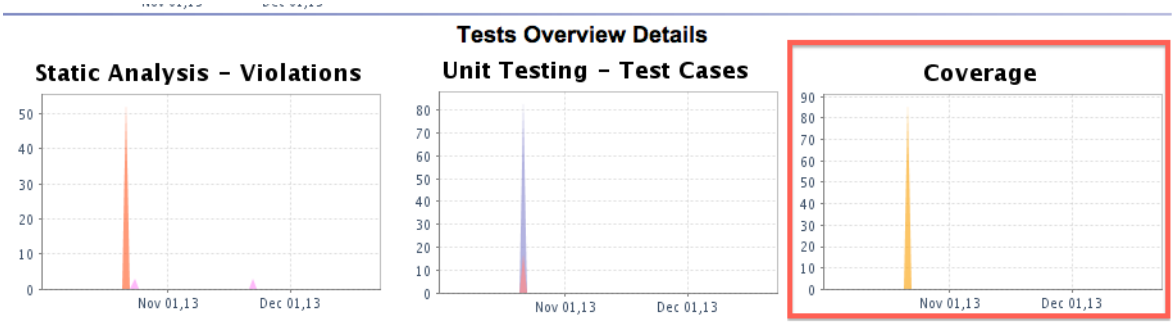
    The Report Center-Emma integration is complete.

Viewing Emma Results

1. Open Development Testing Platform Reports view

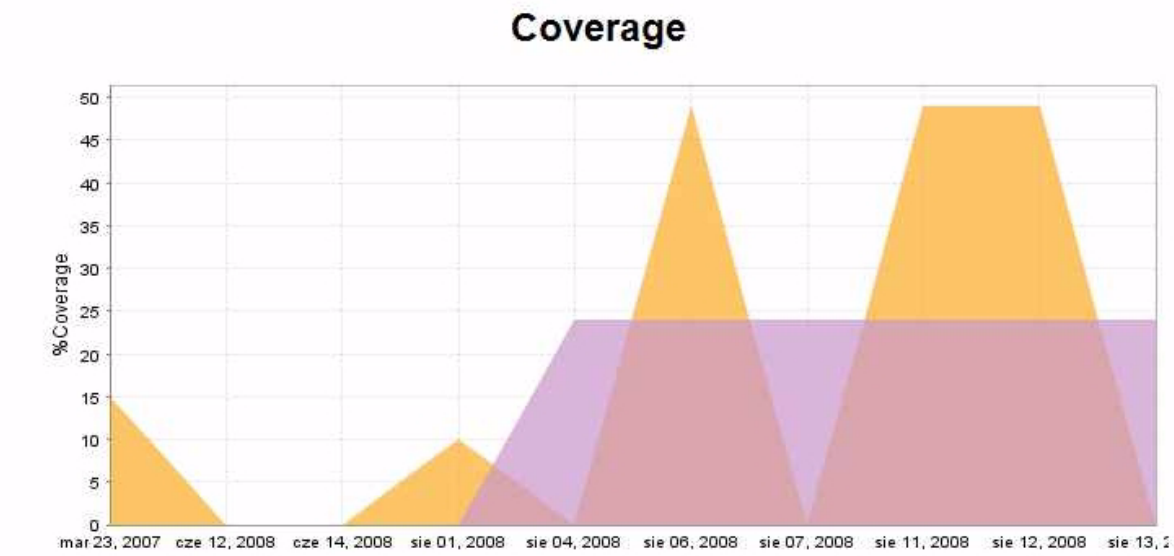


2. Choose **Tests> Tests Overview**
3. Click the **Coverage** report



In Emma mode, the Coverage report and its drill-down reports should display and additional Emma Coverage column that shows the coverage reported by Emma.

Click on the column header to access the Emma HTML file.



Date	Unit Tests Coverage	Emma Coverage
sie 13, 2008	0% (0 / 0)	**24%** (5466 / 23157)
sie 12, 2008	49% (1582 / 3256)	**24%** (5466 / 23157)
sie 11, 2008	49% (791 / 1628)	**24%** (5466 / 23157)
sie 07, 2008	0% (0 / 0)	**24%** (5466 / 23157)
sie 06, 2008	49% (791 / 1628)	**24%** (9110 / 38595)
sie 04, 2008	0% (0 / 0)	**24%** (1822 / 7719)
sie 01, 2008	10% (791 / 8035)	**0%** (0 / 0)
cze 14, 2008	0% (0 / 0)	**0%** (713 / 156813)
cze 12, 2008	0% (0 / 0)	**0%** (1426 / 313626)
mar 23, 2007	15% (26250 / 175526)	**0%** (0 / 0)

Integrating Report Center with Source Control Management Systems

Integrating with your your organization's source control management system (SCM) enables Report Center to show comprehensive statistics based on data pulled from committed source files.

When DTP is integrated with your SCM, SourceScanner (shipped with SDLC Extensions) scans files stored in the source repository and stores revision information in the Report Center database. The data that Report Center collects is used to generate accurate reports. Report Center can show the source code from the appropriate repository in its source-related reports.

If DTP is not integrated with your organization's SCM, you can still view source file content sent to DTP Server from DTP Engines. See "Displaying Source Code published by DTP Engines", page 68, for additional information

About Parasoft SDLC Extensions

SourceScanner does not ship with Development Testing Platform. It is part of the Parasoft SDLC Extensions module. Contact your Parasoft representative if you would liike to use Parasoft SDLC Extensions.

Configuring SourceScanner

SourceScanner should be configured in order to feed data into Report Center.

SourceScanner is available with other SDLC integration extensions. For information about how to configure SourceScanner, see the *Parasoft SDLC Integration Extensions User's Guide*.

Defining Source Control Filter for a Report Center Project

After SourceScanner is configured to feed data into Report Center, you should define the repository files that you want to include in reports for the selected project. For information about how to do this, go to "Source Control Filter", page 168.

Important! *It is strongly recommended that any one specific source repository location be scanned by only ONE SourceScanner project. For instance, if you intend to scan the sources for two branches and trunk of the same files, then you should define it in only one SourceScanner project. Do not define and run three separate projects.*

Viewing Statistics for Source Control and Files' Source Code in Report Center Reports

After configuring SourceScanner and setting source control filters for various Report Center projects, Report Center can include SCM statistics into reports and source code views based on source code data. For additional information, see the following sections:

- "Violations Explorer", page 110

- “Coverage Explorer”, page 125
- “Test Explorer”, page 129
- “Reports”, page 149

Viewing Source Code (Show Source Functionality)

You can configure DTP to include SCM data in Report Center. Your repository client must be installed on the same machine as Report Center to allow DTP to show sources from repositories. The following table lists SCMs that can be integrated with DTP to show source code in Report Center.

SCM	Notes
Accurev	Verify that Accurev client is installed and that the `accurev` command is on the system path. For troubleshooting issues, see “Show Source Functionality”, page 307. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110).
ClearCase	Verify that ClearCase client is installed and that the `cleartool` command is on the system path. For troubleshooting issues, see “Show Source Functionality for ClearCase”, page 308. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110).
CVS	No CVS client is required to be installed. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110).
Jazz (Rational Team Concert)	Jazz version 2.0.0.2 is supported. Jazz Plain Java Native Libraries must be located in the Development Testing Platform machine. Development Testing Platform will be able to locate these native libraries through SourceScanner. Jazz Eclipse Client must be located in the Development Testing Platform machine as it contains the Jazz command line client. The path to the Jazz command line client, which is typically located under [Jazz Directory]/scmtools/eclipse/, must be added to the PATH environment variable. SourceScanner version 5.4.2 is required; SourceScanner and Development Testing Platform must be located in the same machine. Supported for reports view (see “Reports”, page 149) only.
GIT	Verify that GIT client is installed. The git command should be on the system path. Local repository (set as ROOT in Source Scanner) and Development Testing Platform must be located in the same machine. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110).

SCM	Notes
Microsoft Team Foundation Server	Supported for Violations Explorer only. See “Violations Explorer”, page 110.
Microsoft Visual SourceSafe	Supported for Violations Explorer only. See “Violations Explorer”, page 110.
Perforce SCM	Verify that Perforce client is installed. The p4 command should be on the system path. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110).
Serena Dimensions	Verify that the DM_ROOT environment variable is set and it is correct (e.g. `export DM_ROOT=/opt/serena/dimensions/12.2/cm`) Verify that the LD_LIBRARY_PATH environment variable contains the path to Serena Dimensions libs (e.g. `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/serena/dimensions/12.2/cm/lib`) Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110).
StarTeam	Verify that you have the Borland StarTeam SDK installed in the `[DTP_HOME]/lib/thirdparty` directory. You must restart Development Testing Platform services after installing the SDK. You can download the Borland StarTeam SDK for free from the Borland Web site. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110).
Subversion	Verify that Subversion client is installed. The `svn` command should be on the system path. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110). For troubleshooting issues, see “Show Source Functionality”, page 307.
Synergy CM	Verify that Synergy client is installed and that the `ccm` command is on the system path. Supported for reports view (see “Reports”, page 149) and Violations Explorer (see “Violations Explorer”, page 110). For troubleshooting issues, see “Show Source Functionality”, page 307.

See “Show Source Functionality”, page 307, if you experience problems showing sources.

Integrating with PTC Integrity Source Control Extension

Parasoft DTP integrates with PTC Integrity (MKS) 10.x, which enables the following functionality:

- Native support for scoping and identifying authors of Parasoft Test tasks
- Allows users to perform Parasoft Code Review
- View source code from Parasoft test tasks
- Facilitate the Code Review automation process

Following integration support has been tested successfully using PTC Integrity 10.4. However, it has been implemented to support MKS API 4.10 (which support PTC Integrity 10.0 ~ 10.4).

Packages for Integration

Parasoft provides the following packages or MKS integration:

- `xtestMKS.jar`: Parasoft Source Control implementation to support for PTC Integrity 10.x
- `mksapi.jar`: PTC Integrity Java API

Requirements

- The PTC Integrity 10.x client must be installed. Parasoft will use both the Source Integrity (SI) command line from PTC Integrity client and Java API to retrieve information.
- A sandbox for the projects that tools are going to use first must be created. All files must be located under the sandbox.
- Parasoft Test 9.5.10 (Jtest, C++test, SOAtest) make sure to have or newer must be installed.
- Parasoft DTP installation on Linux. This integration is not currently supported for DTP installed on a Windows server.

Installation and Configuration

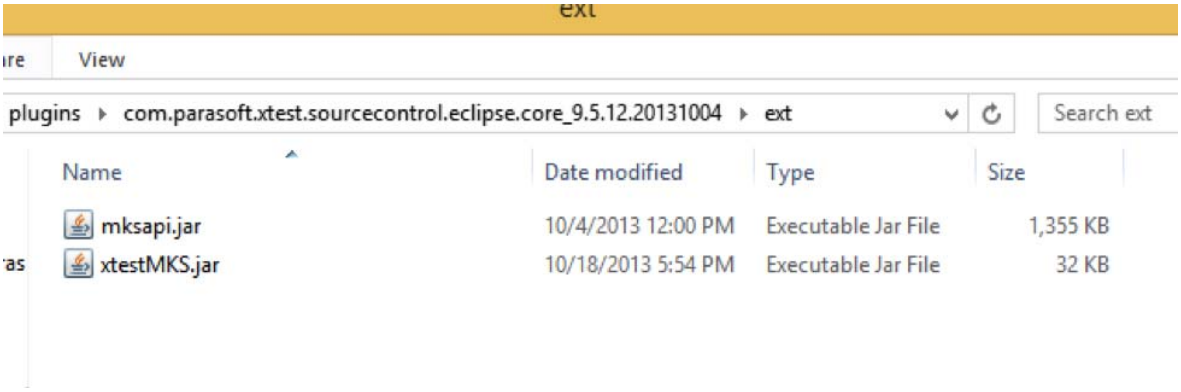
The packages are installed and configured in Parasoft Test and DTP.

Installation on Parasoft Test

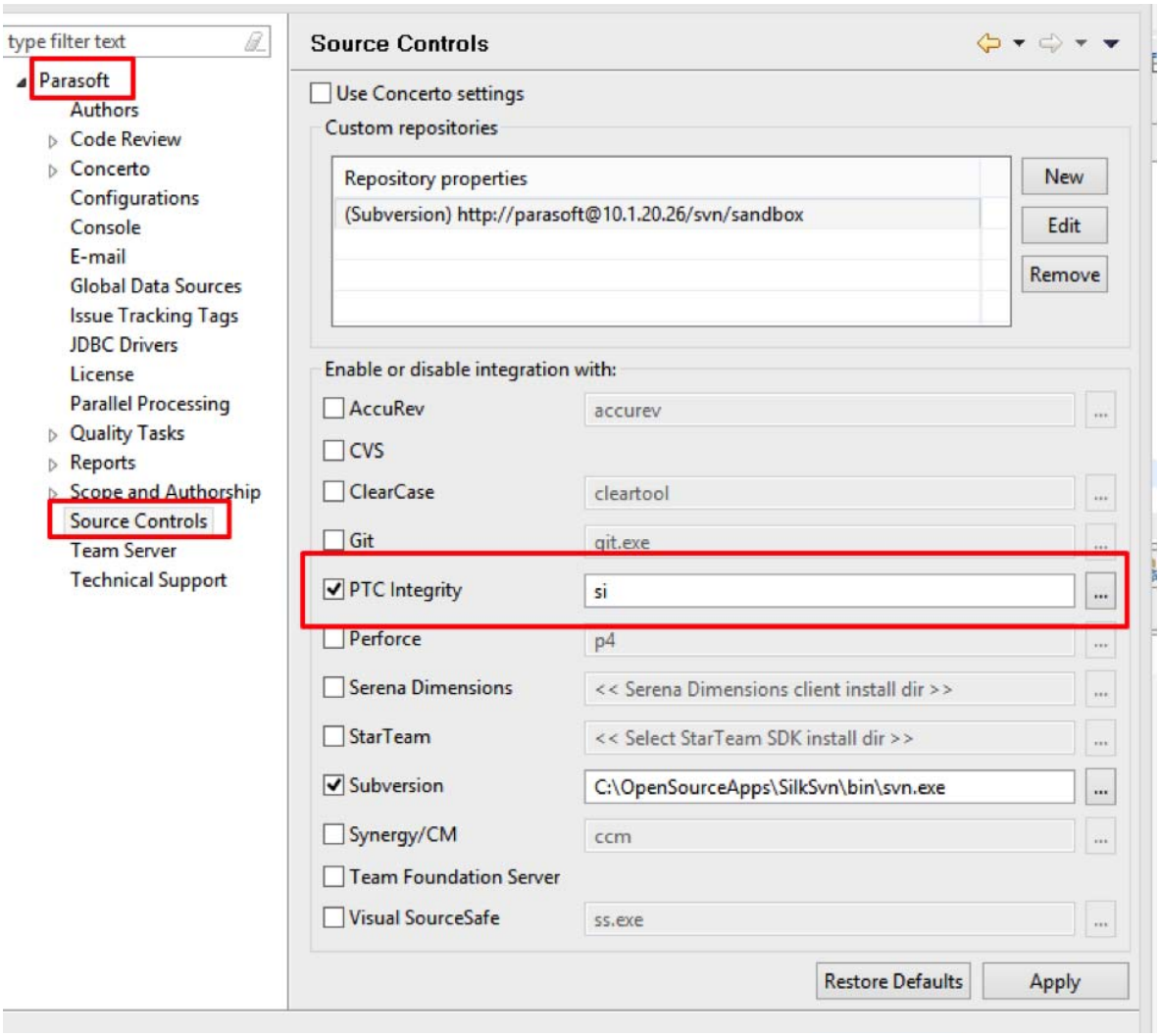
1. Open the installation directory
    - Windows 32-bit standalone: C:\Program Files\Parasoft\Test\9.5
    - Windows 64-bit standalone: C:\Program Files (x86)\Parasoft\Test\9.5
    - Windows 32-bit plug-in: C:\Program Files\Parasoft\Test for Eclipse\9.5
    - Windows 64-bit plug-in: C:\Program Files (x86)\Parasoft\Test for Eclipse\9.5
    - Linux standalone: $HOME/parasoft/test/9.5

- Linux plug-in: $HOME/parasoft/test for eclipse/9.5

2. Create a directory called `ext` in the `[INSTALL_DIR]/plugins/com.parasoft.xtest.sourcecontrol.eclipse.core_<version>` directory.
3. Copy the integration packages to the `ext` directory



4. Start the tool and choose **Parasoft> Preferences> Source Control** to validate that the PTC Integrity option is available.



Installation on Parasoft DTP

1. Extract the contents of the MKSjars.zip file into a temporary location the system where DTP is installed.
2. Create a directory called `scontrol` in the `$DTP_HOME/plugins` directory.
3. Open the `$DTP_HOME/bin/reportserver.sh` script with your favorite editor.
4. Locate the line that reads `CATALINA_OPTS` and add the following line:

   ```
   -Dscontrol.ext.dir=$PST_HOME/plugins/scontrol \
   ```
5. Save the file and exit the editor
6. Restart DTP using the `$PST_HOME/bin/concertoconsole.sh` script.

```
# Needs to be exported to be seen by catalina.sh
export JAVA_HOME
export JAVA_OPTS="-D$PCC_RECOGNITION"

export CATALINA_OPTS="-Djava.ext.dirs=$PST_HOME/lib/rs:$PST_HOME/lib/ext \
                      $JAVA_MEM  \
                      -Dpcc.dir=$PST_HOME -Dinstall.dir=$GRS_HOME -Dtcm.dir=$TCM
_HOME -Dls.dir=$LS_HOME \
                      -Dcom.parasoft.xtest.logging.config.file=$PST_HOME/bin/log
_config.xml \
                      -Dscontrol.ext.dir=$PST_HOME/plugins/scontrol \
                      -Dsource.scanner.log=source_scanner \
                          -Djava.awt.headless=true"

export LICENSE_SERVER_HOME=$LS_HOME

if [ "$1" = "start" ] ; then

    checkPortAvailable

    echo "*** Starting catalina at" `date` >> $RS_OUT_LOG
    exec $CATALINA_HOME/bin/catalina.sh run >> $RS_OUT_LOG 2>> $RS_ERROR_LOG
"reportserver.sh" 148L, 3707C                                  128,23        92%
```

Configration and Setup on Parasoft Test

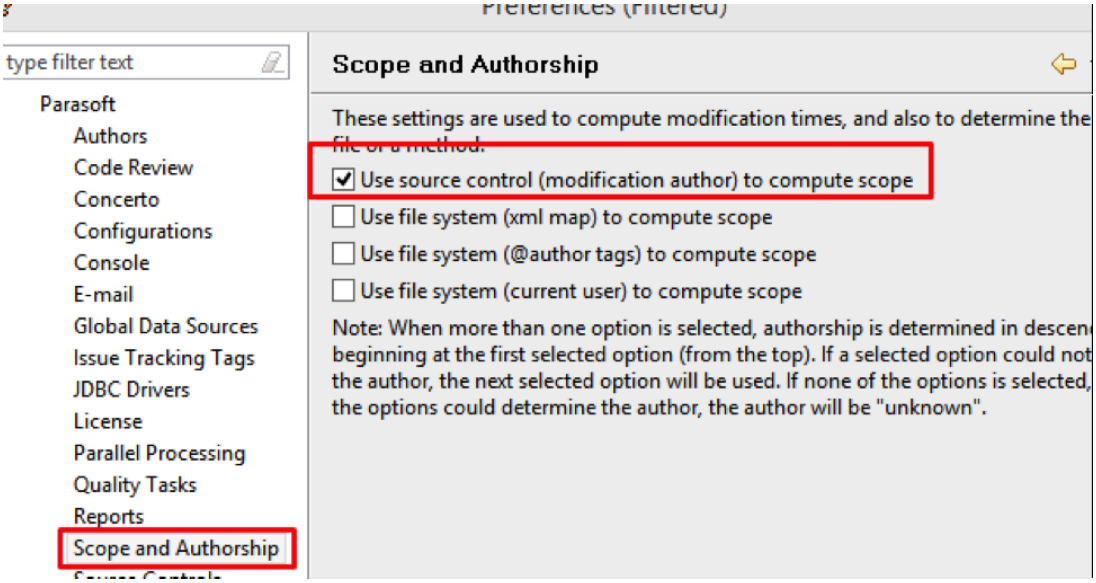
For Linux/Solaris installations, perform the following steps before proceeding the rest of the configuration:

- Make sure to add `$INSTALLDIR/bin` directory to the top of the PATH.
- Make sure to set `LD_LIBRARY_PATH` to point `$INSTALLDIR/lib/linux` (`$INSTALLDIR/lib/solaris` for Solaris).

The `$INSTALLDIR/lib/<linux/solaris>` directory provides both 32-bit and 64bit libraries; MKS API will load proper libraries based on JVM.

1. Start your Parasoft Test application
2. Choose **Parasoft > Preferences > Scope and Authorship**
3. Enable the **Use source control (modification author) to compute scope** option

4. Disable all other options and click **Apply**.



5. Choose **Parasoft> Preferences> Parasoft> Source Control** and verify that the PTC Integrity option is enabled and that si command is set to the path. If si is not under PATH, make sure to point to it with an absolute path.
6. Click **New** in the **Custom repositories** section and enter the following options in the **Properties** window:

   serverport: PTC Integrity port (by default 7001)

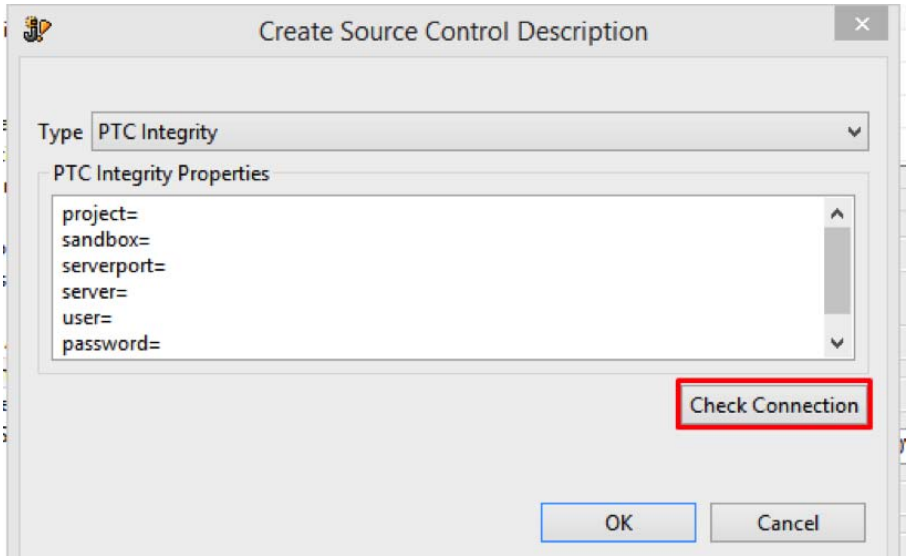
   server: PTC Integrity Server Name

   user: username to authenticate for PTC Integrity

   password: password to authenticate for PTC Integrity

   sandbox: Sandbox location where local resources are synchronized (ends with project.pj in the path)

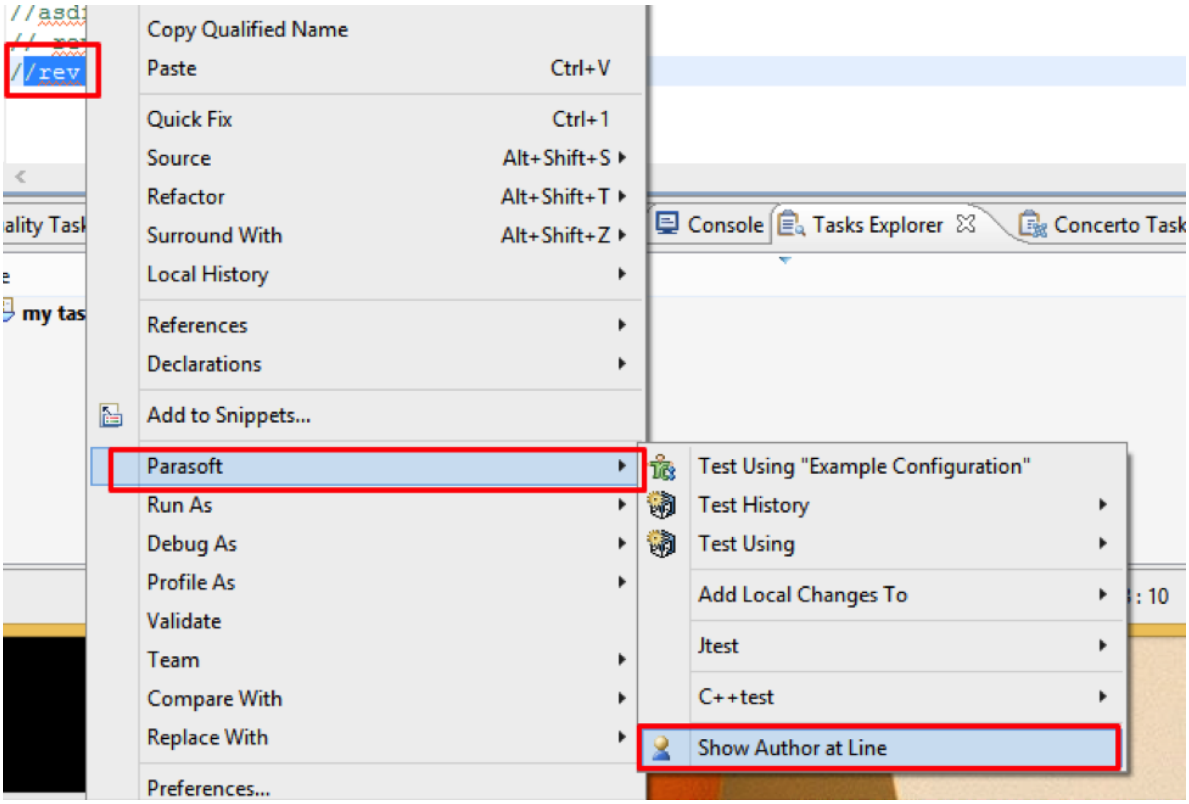
   project: PTC Integrity Project location on the server (example: /Project XZY/project.pj)



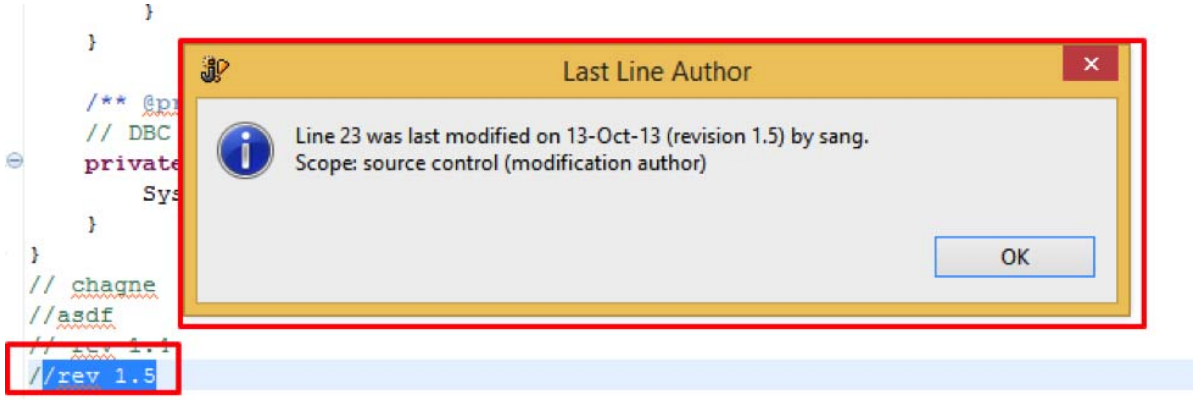
7. Click **Check Connection** to validate your option settings. If there is an error, a PTC Integrity error message will be returned in the dialog box.
8. Click **OK** to save and exit.

Validating the Settings in Parasoft Test

1. Import a project under the sandbox to Parasoft Test solution.
2. Open a file and right click on a line.
3. Choose **Parasoft> Show Author at Line.**



4. A dialog box should open and show the author information and revision number.



Configuring PTC Integrity under Development Testing Platform (DTP) with SourceScanner

Before proceeding the configuration, make sure that the PTC Integrity (MKS) Client is installed and working correctly on the machine where you will run SourceScanner and DTP.

For Linux/Solaris installations, perform the following steps before proceeding the rest of the configuration:

- Make sure to add `$INSTALLDIR/bin` directory to the top of the PATH.
- Make sure to set `LD_LIBRARY_PATH` to point `$INSTALLDIR/lib/linux` (`$INSTALLDIR/lib/solaris` for Solaris).

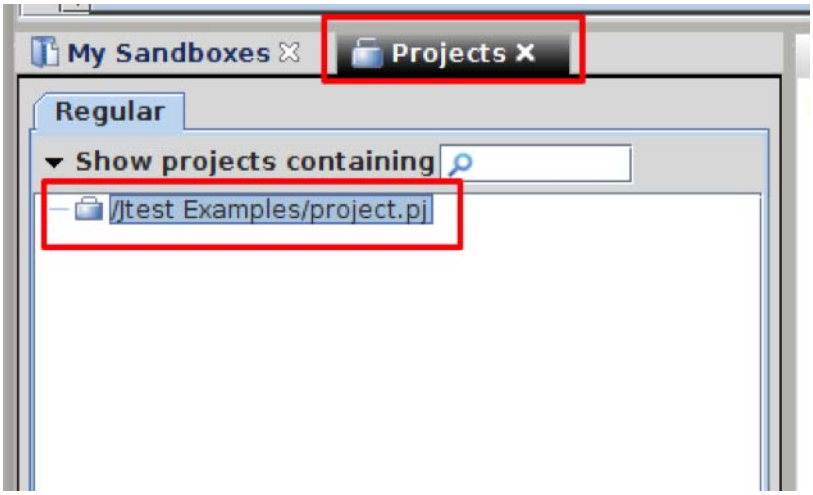
The `$INSTALLDIR/lib/<linux/solaris>` directory provides both 32-bit and 64bit libraries; MKS API will load proper libraries based on JVM.

For Windows, make sure to add %INSTALLDIR%/bin to PATH in the SYSTEM environment.

For Bash:

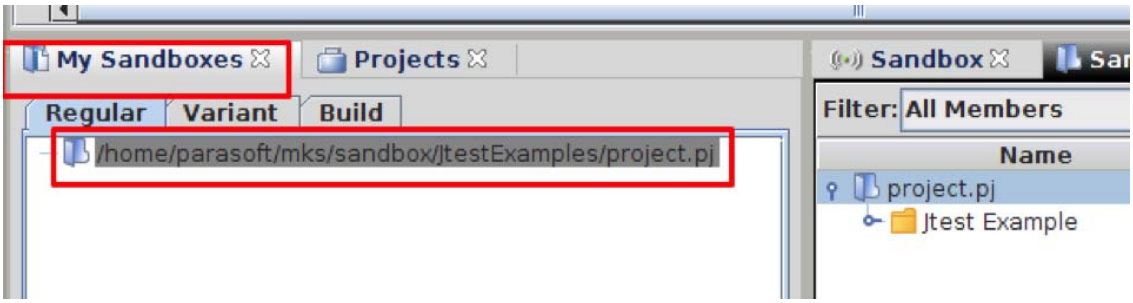
```
PATH==/home/Integrity/IntegrityClient10/bin:$PATH
LD_LIBRARY_PATH=/home/Integrity/IntegrityClient10/lib/
linux:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH PATH
```

1. Start the si gui command to open Integrity Client GUI.
2. Identify project path from Projects tab as follow:



   This value will be used as Project Root under SourceScanner.

3. Create a sandbox for this system and record the value for the sanbox:



This will be used as a Sandbox Location value under SourceScanner.

4. Follow the DTP installation guide to install Development Testing Platform. Make sure PATH and `LD_LIBRARY_PATH` is loaded to the SHELL before starting DTP under Linux.
5. Extract the `SDLCEXtensions-5.7.3.zip` zip file to install SourceScanner.

   For Windows:

   1. Extract the zip file into `C:\Parasoft`, which will create the `C:\Parasoft\SDLCExtensions` directory.
   2. Set `PROSERVE_HOME` environment variable to `C:\Parasoft\SDLCExtensions`

   For Linux/Solaris:

   1. Extract the zip file into $HOME/parasoft (it will create $HOME/parasoft/SDLCExtensions directory)
   2. Change the to bin directory and run `chmod a+x *.sh` to all scripts executables.
   3. Edit `$HOME/.proserve.rc` and add the following lines:

      ```
      export JAVA_HOME=<JAVA_RUNTIME_INSTALL_DIR>
      export PROSERVE_HOME=<where SDLCExtensions are installed>
      ($HOME/parasoft/SDLCExtensions)
      ```

   4. In `$HOME/.bashrc`, make sure PATH and `LD_LIBRARY_PATH` are saved and loaded before starting DTP or SourceScanner.
6. Start `SSGUI.sh` (for Linux/Solaris) or `SSGUI.cmd` (for Windows) to start SourceScanner GUI.
7. Follow SDLCExtensions documentation to setup SourceScanner. In this document, it will only discuss PTC Integrity related information.

Creating a PTC (MKS) Integrity Project

1. Under Project, choose **MKS** as the Source Control type
2. Enter the project name; the value will be used in DTP as the Repository ID
3. Root should represent the server Project Path as captured in Step 2.
4. Sandbox location should have the value from Step 3.
5. Refer to "SDLCExtensions User's Guide" for additional settings and information. (Note: MKS reference under SDLCExtensions documentation is outdated and will be updated in 6.0 release)
6. Run SourceScanner to populate data to DTP.

Sending Test Results from Third-party Tools to Development Testing Platform

Development Testing Platform provides an open API that can be used to send custom and third-party automated test results to Development Testing Platform. Once results are sent to Development Testing Platform, they can be used in your development process, e.g. visualize the results in Report Center dashboards.

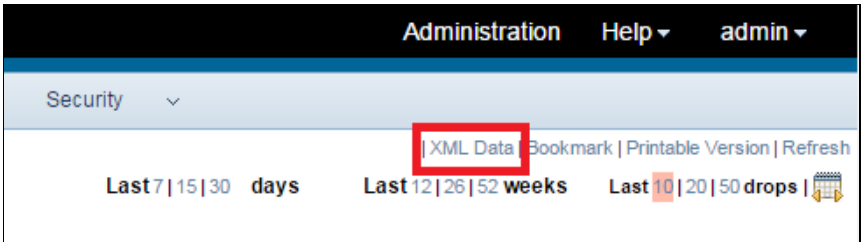
The API is shiped as open source bundle. For details please visit:

http://sourceforge.net/projects/psfconcertoapi/?source=directory

Importing Reports to Microsoft Excel

The XML Data option enables you to export data displayed in Report Center reports, and import it into a spreadsheet application, such as Microsoft Excel. This enables you to create custom tables, reports, and graphs.:

1. Open a report and click the XML Data link

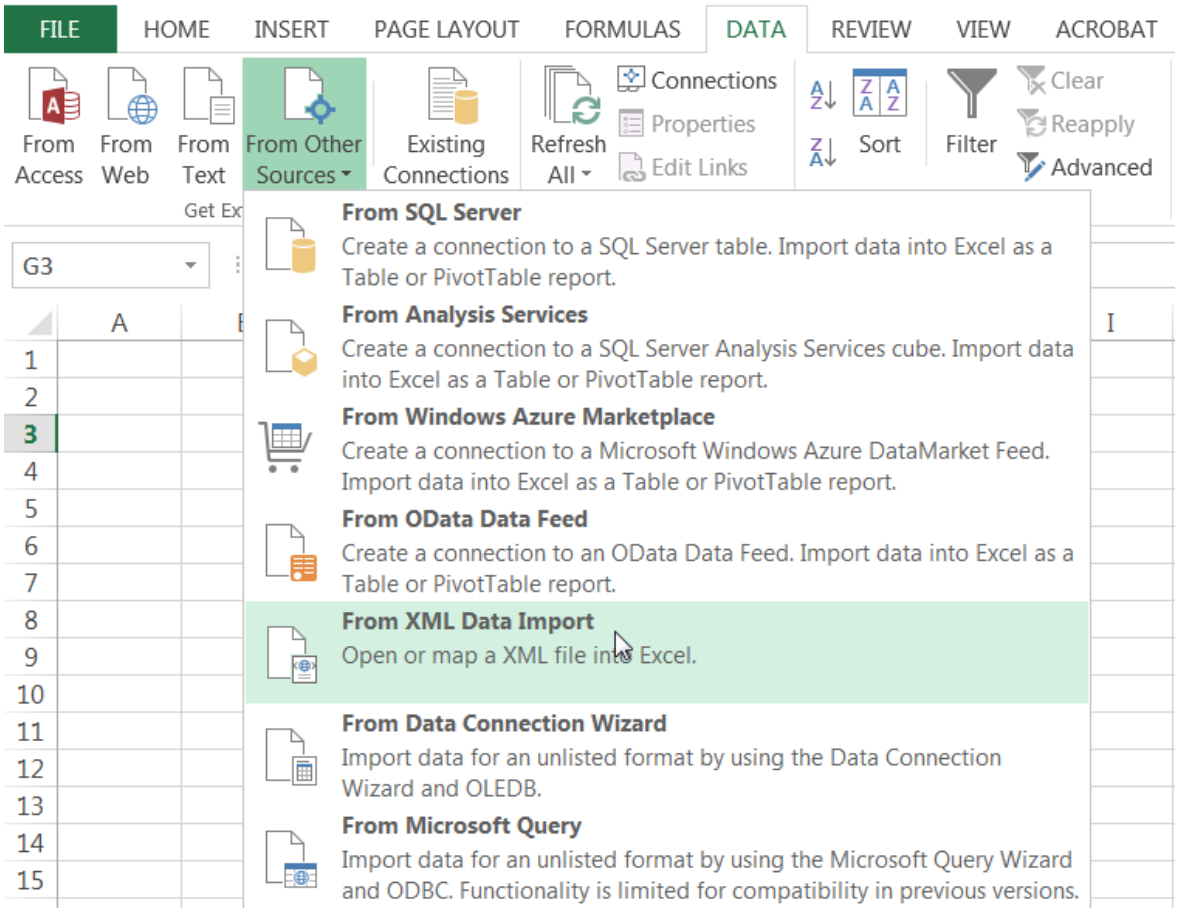


   Some top-level reports do not have an **XML Data** link. In these cases, you must drill down to a specific graph to export the data.

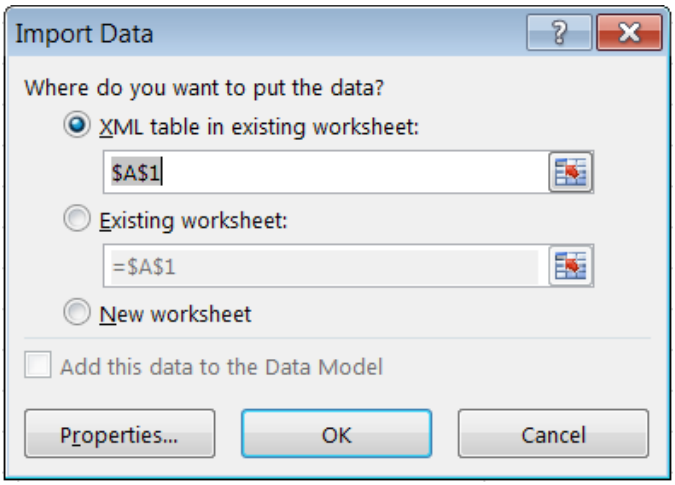
2. Depending on your browser and file handling extensions, you may be prompted to save the report file .

3. Start your spreadsheet application and import the downloaded XML file. In this example, we'll use MS Excel .
4. Choose **File> New > Blank workbook**

5. Click the Data tab and in the Get External Data ribbon, choose **From Other Sources> From XML Data Import**.



6. Specify the region in the spreadsheet where you want to place the data and click **OK** to finish importing your report



Java API

Development Testing Platform's Java API provides a way for users to extend the functionality of Development Testing Platform. The platform provides out-of-the-box integration with Bugzilla and JIRA bug tracking systems, but organizations that use a different bug tracking system can write their own custom bug scanner by extending the bug scanner Java API. Organizations can also write their own custom requirements scanner by extending the requirement scanner Java API.

To access the Java API documentation, enter `https://host:port/grs/java-api` into the browser address bar. Typically, the URL is `https://host/grs/java-api` if Development Testing Platform is deployed on port 80. The platform is typically deployed on port 8443 if port 80 isn't used. For more information about using the Java API, see "Manually Configuring BTS or RMS Scanner", page 219, and "Custom RMS Scanner", page 223.

The Java API for DTP custom processors is available at http://build.parasoft.com/maven. The available build artifacts include both the compiled jar and the packaged javadoc jar file. These artifacts share the following properties:

- groupId: com.parasoft.dtp
- artifactId: com.parasoft.api.dtp.processors
- version: 5.1.0

Managing Report Center Installation

In this section:

- Managing Report Center Installation in Windows
- Managing Report Center in Linux

Managing Report Center Installation in Windows

When you have an outdated Development Testing Platform or Report Center installation, you might want to uninstall MySQL or Report Center, clear your old Report Center database, or upgrade to a new version of Report Center.

Uninstalling Development Testing Platform or MySQL

Report Center cannot be uninstalled without uninstalling Development Testing Platform as well. You must remove Development Testing Platform/Report Center before you remove MySQL.

To uninstall Development Testing Platform/Report Center:

1. Choose **Start > Settings > Control Panel**. The Control Panel window is displayed.
2. Double-click **Add/Remove Programs**. The Add/Remove Programs window is displayed.
3. Select Report Center and click **Remove**. Follow the onscreen instructions.

To uninstall MySQL:

1. Choose **Start > Settings > Control Panel**. The Control Panel window is displayed.
2. Double-click **Add/Remove Programs**. The Add/Remove Programs window is displayed.
3. Select MySQL and click **Remove**. Follow the onscreen instructions.

Clearing the Report Center/Project Center Database

If you do not want to uninstall MySQL but you want to have a fresh database, you can clear the Report Center database, then create a fresh database structure.

Note: The MySQL service must be running before you clear the Report Center database.

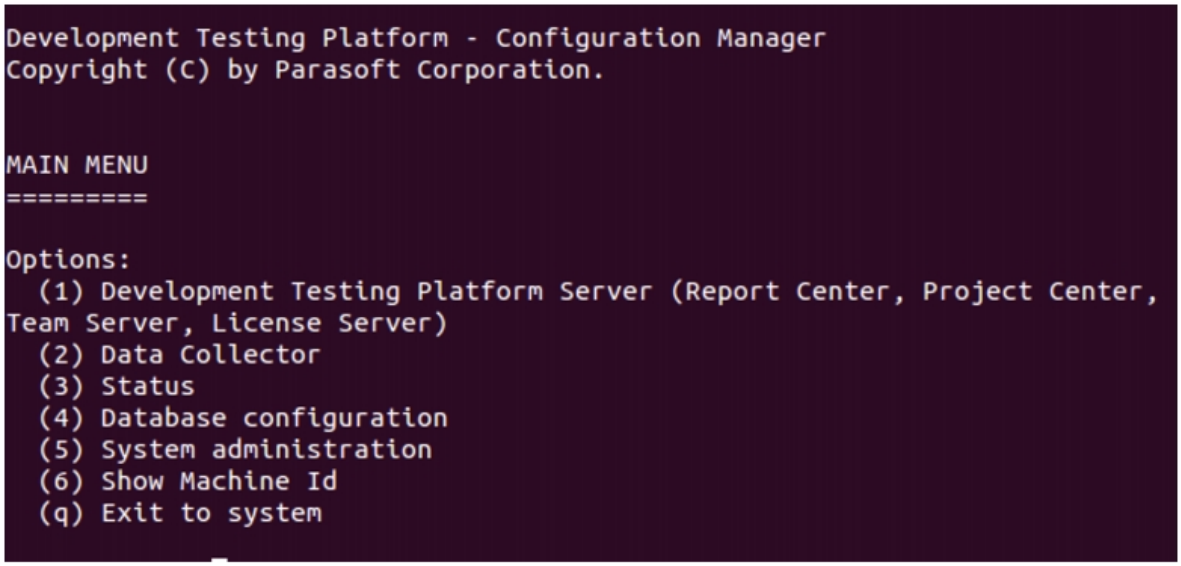
To clear the database:

1. Stop Data Collector.
2. Stop the Development Testing Platform Server.
3. Run the `mysql` utility.
4. Execute the following command at the prompt:
   ```
   drop database GRS
   ```
5. Restart the system.

6. Create a new database by following the procedure described in the Development Testing Plat-from Quick-install Guide..

Managing Report Center in Linux

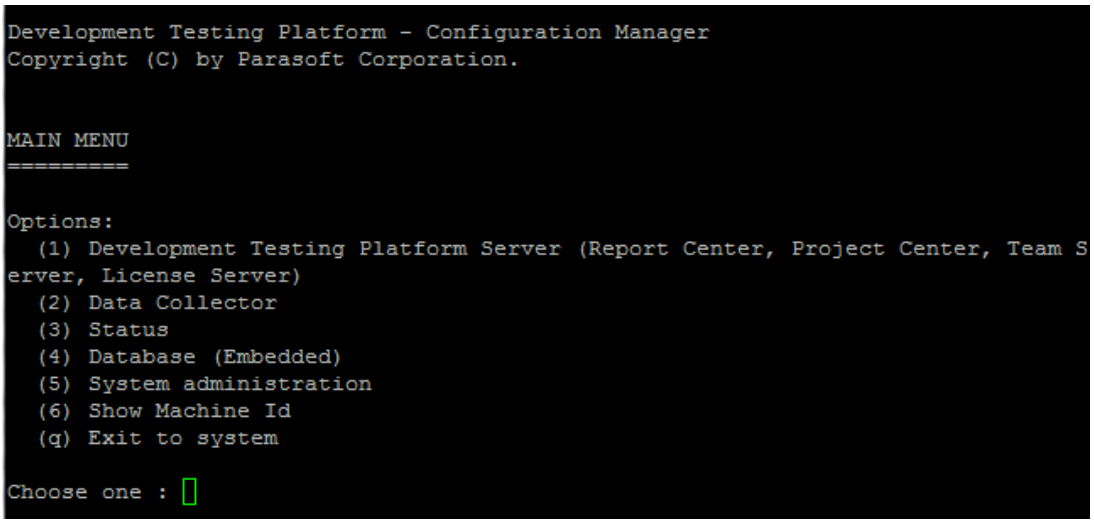
After properly installing Report Center, you can run the `dtpconsole.sh` file to open the main menu. This file is in the `DTP_HOME/bin` directory. If you installed the standard DTP distribution, the following options are available:



- Choose `(1)` to open the Development Testing Platform Server menu; see "Development Testing Platform Server Menu", page 292.
- Choose `(2)` to open the Data Collector menu; see "Data Collector Menu", page 292.
- Choose `(3)` to check for errors, see Parasoft services are running, and check disk space usage.
- Choose `(4)` to open the Database configuration menu
- Choose `(5)` to open the System administration menu
- Choose `(6)` to show your machine ID.
- Choose `(q)` to exit the main menu.

DTP with Embedded Database Server Console

In the embedded database server edition of DTP (Linux only), the `(4) Database configuration` option is replaced with the `(4) Database (Embedded)` option. Choose this option to access the embedded database menu, which you can use to start and stop the database server. All other options are the same as the standard DTP distribution.



Development Testing Platform Server Menu

You can configure the various parameters of the Development Testing Platform Server.

WARNING: *It is recommended to leave the values for each option at the default setting.*

Option	Result
`(1) [Run /Stop] Service`	Starts or stops Data Collector service.
`(q) Exit to previous menu`	Entering `q` at the prompt exits the Server Menu and opens the Main Menu.

Data Collector Menu

Option	Result
`(1) [Run /Stop] Service`	Starts or stops Development Testing Platform services.
`(q) Exit to previous menu`	Entering `q` at the prompt exits the Server Menu and opens the Main Menu.

System Administration Menu

Option	Result
`(1) Enable/Disable administration mailing`	Enables or disables the feature that alerts the Report Center administrator when disk space is running low. A local SMTP server must be running and properly configured in order to use this function. Default is disabled.
`(2) Change administrator address`	Prompts you to specify the email address for the person who should be notified about low disk space.
`(3) Change risky disk space usage [90%]`	Prompts you to set an upper threshhold for disk usage. If disk usage reaches this level, the Report Center administrator will receive an email that alerts him or her to this problem.
`(q) Exit to previous menu`	Exits the Server Menu and opens the Main Menu.

Changing MySQL Root Password

By default a root user for MySQL database has an empty password. In other words, it is enough to press Enter when prompted for a password to be granted access to MySQL database as a root user. The Database Menu in dtpconsole.sh uses this fact when trying to connect to MySQL database to create an initial database, for example. However a good practice is to set some password for MySQL root user, especially when connecting through the network. If so, dtpconsole.sh must know what password to use when trying to connect. Otherwise, an empty password will be used and the connection will fail. You can enter an appropriate password using the Database Menu:

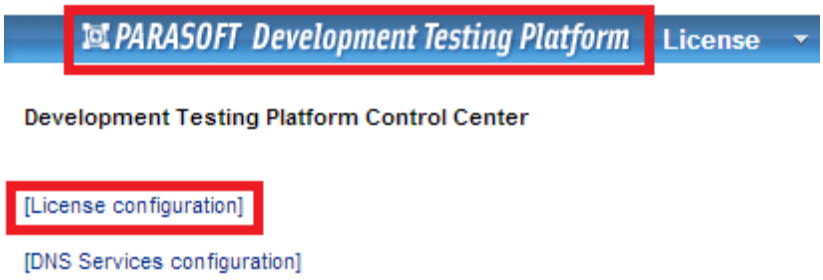
1. Enter 3 at the Main Menu prompt to open the Database Menu. The available options are defined in the following table:

Option	Result
`(1) Create initial database`	Entering `1` at the prompt creates an initial database.
`(2) Set MySQL root password`	Entering `2` at the prompt allows you to enter MySQL root password, which then will be used by dtpconsole.sh when connecting to MySQL database (default is no password).
`(q) Exit to previous menu`	Entering `q` at the prompt exits the Database Menu and opens the Main Menu.

2. Select the **Set mysql root password** option from the Database Menu.

Verifying Development Testing Platform Product Availability

1. Click the Parasoft Development Testing Platform logo link
2. Click **License configuration** t



The Development Testing Platform products enabled and licensed for your installation are displayed:

Migrating Data

You should back up the data from older versions of Development Testing Platform (version 5.x) or Concerto (version 4.x) and migrate it to the new version. You can accomplish this by performing the following tasks:

- Migrating Data from License Server. Do one of the following:
  - Migrating Data from License Server 1.0 or 2.0, or
  - Migrating Data from License Server 2.x to a New Version
- Migrating Team Server Configuration
- Migrating Team Server Data

Migrating Data from License Server 1.0 or 2.0

License data defined for License Server is stored in the `/LicenseServer/.psrc` directory. To import data from License Server 1.0 or 2.0 to Concerto 1.x, follow these steps:

1. Back up your .psrc file.
2. Uninstall the previous License Server and Concerto versions.
3. Install Concerto 1.x.
4. Verify that `/LicenseServer/conf/.psrc.xml` exists, and then rename it to ensure that License Server will not use this configuration upon startup. An example is to rename it to `.prsc.xml.bak`.
5. Copy the backup of your .psrc file (from Step 1.), and then paste it in the `/LicenseServer/conf/` directory to force License Server to use the previous license configuration.
6. Run Concerto 1.x.

Upon startup, License Server will import license configurations from the .psrc file and create a new .psrc.xml file based on data from the previous .psrc.

Migrating Data from License Server 2.x to a New Version

To import data from License Server 2.x to a new version of Development Testing Platform, follow these steps:

1. Back up the `DTP_HOME/LicenseServer/conf/` file.
2. Uninstall the previous version.
3. Install Development Testing Platform.

   See the "Development Testing Platfrom Quick-install Guide" for more information.
4. Copy and paste the backup to the new `DTP_HOME/LicenseServer/conf` location.

If you upgrade through the Repair option (recommended), there is no need to use backups because your previous data/configuration is preserved for use by the new version automatically. Keeping backups is recommended in the even that your new installation fails and you need to reinstall the previous version and restore your data.

Migrating Team Server Configuration

To restore the previous Team Server configuration follow the steps below:

1. Back up the `DTP_HOME/tcm/conf/` file.
2. Uninstall the previous Team Server/Concerto version.
3. Install Concerto 1.x.

   See the "Development Testing Platfrom Quick-install Guide" for more information.
4. Copy and paste the backup to the new `DTP_HOME/tcm/conf` location.

If you upgrade through the Repair option (recommended), there is no need to use backups because your previous data/configuration is preserved for use by the new version automatically. Keeping backups is recommended in the even that your new installation fails and you need to reinstall the previous version and restore your data.

Migrating Team Server Data

To restore the previous Team Server data follow the steps below:

1. Back up the `DTP_HOME/tcm/storage/` file.
2. Uninstall the previous Team Server/Concerto version.
3. Install Concerto 1.x.

   See the "Development Testing Platfrom Quick-install Guide" for more information.
4. Copy and paste the backup to the new `DTP_HOME/tcm/storage` location.

If you upgrade through the Repair option (recommended), there is no need to use backups because your previous data/configuration is preserved for use by the new version automatically. Keeping backups is recommended in the even that your new installation fails and you need to reinstall the previous version and restore your data.

Upgrading MySQL Server

You can upgrade MySQL server when you have an existing Report Center database and want to preserve it. Make sure to back up the Report Center database file before upgrading. See the "Development Testing Platfrom Quick-install Guide" for more information on backing up the Report Center database. For instructions on upgrading MySQL, visit the MySQL documentation library:

http://dev.mysql.com/doc/

Configuration Manager Internal Details (Linux/Solaris)

This topic provides a brief description of the internal Development Testing Platform scripts that are executed when you work on the Development Testing Platform Configuration Manager command line program (DTP_HOME/bin/dtpconsole.sh).

Subtopics include:

- Running Web Server
- Running Data Collector

Running Web Server

When you select (1) to run Development Testing Platform Web Server, the dtpconsole.sh script executes another script:

- DTP_HOME/bin/reportserver.sh.

The configuration file, "$DTP_HOME/.server_params", is also updated with the following line:

- service=ENABLED

All of the logs from Development Testing Platform that contain information—including error messages—are saved in various files in `DTP_HOME\logs\`. The main log file is rs.log. A process ID of the Development Testing Platform server is stored in `DTP_HOME/grs/log/rs.pid`.

For details about other log files, see (“Verification Step 3: Verify that there are no unknown exceptions or messages in log files”, page 66).

If any problems occur during the Development Testing Platform server process, the cron job that was installed during the Development Testing Platform installation will monitor the process and will restart the Development Testing Platform Server whenever the entry "service=ENABLED" is present in the configuration file.

Running Data Collector

When user selects (1) to run Data Collector the dtpconsole.sh script executes another script:

- DTP_HOME/bin/datacollector.sh.

The configuration file DTP_HOME/.collector_params is also updated with the line:

- service=ENABLED

The script "datacollector.sh" runs Data Collector. All of the logs from Data Collector that contain information—including error messages—are saved in `DTP_HOME\logs\dc*` files. `dc.log` contains most of the messages.

A process ID of Data Collector is stored in `DTP_HOME/grs/log/dc.pid`. If any problems occur during the data collection process, the cron job that was installed during the Development Testing Platform installation monitors the process and restarts Data Collector whenever the entry "service=ENABLED" is present in the configuration file.

Data Collector is now running and data collected from Parasoft and third party tools can be stored in the database.

Creating an Initial Database and Upgrading the Database for Linux (Command-line Menu-driven Method)

1. Enter the following command:

   ```
   dtpconsole.sh
   ```

   This script should be on your path. It is located in the `$DTP_HOME/bin` directory.

   After entering this command, the Development Testing Platform Configuration Manager Main Menu displays as follows:

   ```
   MAIN MENU
   =========
   Options:
         (1) Development Testing Platform Server (Report Center, Project
         Center, Team Server, License Server)
         (2) Data Collector
         (3) Database (MySQL only)
         (4) Status
         (5 System Administration
         (q) Exit to system
   Choose one:
   ```

2. Select the **Database** option from the Main Menu by entering `3` at the prompt. For more information on the additional options in the Main Menu, see “Managing Report Center in Linux”, page 291.

   After entering `3`, the Database Menu displays as follows:

   ```
   DATABASE MENU
   ============
   Options:
         (1) Create initial database
         (2) Set mysql root password
         (q) Exit to previous menu
   Choose one:
   ```

3. Select the **Create initial database** option from the Database Menu by entering `1` at the prompt. For more information on the additional options in the Database Menu, see “Changing MySQL Root Password”, page 293.

Note: *By default, a MySQL database installs itself with an empty MySQL root password. If using the empty root password does not allow connection to the database server, the administration script will ask you for a correct MySQL root password.*

4. For new installations, you will be prompted to create a database on the local machine. Choose **Yes**.

For upgrades: You will be prompted whether to perform an upgrade. Choose **Yes**. The more data your previous installation contained, the longer it will take to load the data.

   A prompt is displayed requesting a password for root.

5. Enter the password used for MySQL configuration.

   The script will connect to the MySQL server using the root account and will run a script that creates the initial database.

   Note: *If you encounter any problems, one of two possibilities are likely: 1) MySQL configuration was overlooked; 2) a permission problem on the database exists.*

6. *(optional)* As a precaution, you can request verification that the reports are all up to date. This can be done inside the Report Center Web interface. It is most likely that you will not see anything that needs to be changed.

For upgrades: As a precaution, you might want to perform a second verification of database integrity: `mysqlcheck -u grs -pgrs GRS`

7. (optional) If you plan to move the data directory in MySQL to increase space in your Report Center directory, or for other reasons, then this is a good time to do it.

   Note: *In case of any problems accessing MySQL, you can create an initial database using an SQL script located in the* `DTP_HOME/grs/db` *directory. The script file name is* `"create_db_mysql.sql"`*.*

At this point, Report Center should successfully be installed. (It installed during Development Testing Platform installation.)

Registering Automated Startup Manually (Linux)

During a typical Development Testing Platform installation on Linux, crontab is modified so that it at each system reboot, it calls the cronguard.sh script that automatically starts Development Testing Platform server.

Automated Development Testing Platform startup on server startup is implemented with crontab because root access is not required with crontab. If this was implemented with Unix /etc/init.d services, root access would be required.

If you skipped the crontab configuration step during Development Testing Platform installation, configure it manually as follows:

1. Log in as the user who installed Development Testing Platform.
2. Enter the crontab -e command.
3. Type the following in the editor (after replacing $DTP_HOME with full path where Development Testing Platform is installed)

```
@reboot $DTP_HOME/bin/cronguard.sh > $DTP_HOME/logs/cronguard.log 2>&1
```

Troubleshooting

In this section:

- Viewing Log Files
- Report Center
- Cache Report Executor
- Team Server
- Show Source Functionality

Cache Report Executor

By default, the reports specified in staticLinksConfig.xml should start being cached at 5 a.m. If some of the reports are not cached by 8 a.m., for example, the following should be checked:

- Is report caching ON? See “Changing the Database Connections”, page 276 and “Enabling/Disabling Report Center Data Cache”, page 179.
- What is the allowFromHour set to in the CacheReportExecutor job? See your CronConfig.xml file.
- Has CacheReportExecutor started this morning? See your DTP_HOME\logs\recalculators.log. This is the file where logs from all Report Center background jobs are written.

  Go to the end of this file, and search backwards for the following phrase: "CacheReportExecutor". When CacheReportExecutor runs a specific report from staticLinksConfig.xml, it logs a text line beginning with "Generating cache for", which is another phrase for which you can search.
- CacheReportExecutor might have started, but was paused by users browsing Report Center reports, and might continue after idle time has elapsed following the last user action.

If the report that you need to cache nightly is not in staticLinksConfig.xml, add it to this file. For details, see “Adding a New Report to Cache”, page 191.

Team Server

Issue: Team Server starts, but cannot handle any requests.

Solution: The deployment was not performed correctly. Repeat the deployment procedure described in “General Team Server Configuration”, page 90.

Issue: Team Server starts, but generates an error message for every request (other than connection verification requests); the password is incorrect or missing.

Solution: Repeat the licensing procedure.

Issue: How do I ensure that the latest Jtest plugin is being used?

Solution: Go to the install dir\tomcat\webapps\ directory to verify the date of the jtest dir. If the date is older than the date of jtest.war, remove it and then restart the services. The latest Jtest plugin will be automatically applied.

Show Source Functionality

Check the detailed error message in `DTP_HOME/logs/rs.log` if Report Center is unable to show the file source.

Issue: The message states that the client command (Cleartool, CCM, and so on) is not accessible. The cause could be that the client is not installed.

Solution: Install the client and verify that the command is accessible by issuing it from the command line.

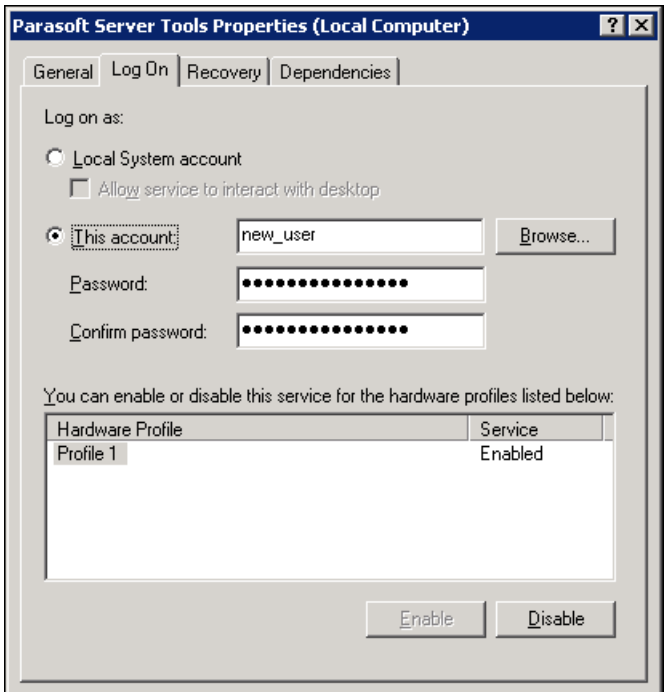
Issue: The client command line works, but Report Center still does not have access to the command. The cause could be that the user who runs the Development Testing Platform server does not have access to the client command.

Solution 1: Provide the user who runs the Development Testing Platform server with access to the client command.

Solution 2: Change the user who runs the Development Testing Platform server to one who already has access to the client command. You can do this in Windows by following these steps:

1. Open Computer Management to access the list of services on your machine:
    a. Right-click the My Computer desktop icon.
    b. Choose **Manage**.
    c. From left pane, choose **Services and Applications> Services**.
2. Select the Parasoft Development Testing Platform service, right-click, and choose **Properties**.

    The Parasoft Development Testing Platform Properties window is displayed.
3. Select the **Log On** tab.
4. Select **This account**, and then type or browse for the appropriate user



5. Click **Apply**, and then **OK**.
6. Restart the service.

Show Source Functionality for ClearCase

Issue 1:

Source Scanner runs on a different server. As a result, it has a different ClearCase view path than Report Center. When Source Scanner scans the source repository, it sends the following information to Report Center:

- Source file information (paths, revisions, and so on)
- Connection parameters to the source repository, which are used by Report Center in Show Source Functionality

Solution: For ClearCase one of the source repository parameters is the path to the view. Source Scanner sends it as an internal CLEAR_CASE_PATH parameter.

Important! *When Source Scanner runs on a different machine than Development Testing Platform and has a different path to the ClearCase view, it is required to specify the ClearCase view path for the Development Testing Platform server.*

To specify the ClearCase view location, edit the following line in the `RSConfig.xml` file:

```
<clearcase-view-path>PUT_CLEARCASE_VIEW_LOCATION_HERE</clearcase-view-path>
```

Issue 2:

Support for ClearCase dynamic and static (snapshot) views.

Solution: Report Center SourceCode Viewer supports both ClearCase dynamic and static (snapshot) views. Additionally, SourceScanner and Report Center can work on different types of views, for instance, it is possible that SourceScanner scans a snapshot view, and then Report Center has a dynamic view defined on its machine. Example:

Dynamic view: M:\my_dynamic_view\myvob\src\com\parasoft\MyFile.java

Snapshot view: C:\ClearCase\views\my_snapshot_view\myvob\src\com\parasoft\MyFile.java

The only requirement for Report Center is to have the path to the view properly defined, as described in “Issue 1:”.

Switching to Debug Logging Mode

You can switch the Development Testing Platform server logging into debug mode as a means of troubleshooting integration issues and other unexpected behavior.

1. Set the Development Testing Platform log level to DEBUG by editing DTP_HOME/bin/log_config.xml and changing

   ```
   <root>
   <priority value="WARN"/>
   <appender-ref ref="FILE"/>
   </root>
   ```

   to

   ```
   <root>
   <priority value="DEBUG"/>
   <appender-ref ref="FILE"/>
   </root>
   ```

   For details see “Changing .log File Properties”, page 158.

2. This step enables an additional debug model Development Testing Platform source-control-association module. Make the following changes only if you encounter issues in source control-related functionality.

   Add the following parameters to Development Testing Platform’s Java VM startup parameters:

   ```
   -Dscontrol.log=true
   ```

   ```
   -Dcom.parasoft.xtest.logging.config.jar.file=/com/parasoft/xtest/logging/log4j/config/eclipse.on.xml
   ```

3. Send the data into Development Testing Platform again and try to reproduce the behavior to obtain the appropriate logging information.

4. Check `rs.log`, which may exceed the 50 MB of data reached during normal startup.

Index

A

adding
    new reports to cache 191
assigning branch/tag to project 186
Atlassian JIRA 253

B

baselines
    deactivating 128
    deleting 128
    managing 120
    setting 127
branches
    assigning to project 186
    scan configuration 186
BTS scanner
    running 223
bug tracking systems
    integrating 215, 243
bugs
    calculating 174
    verifying decrease 33
builds
    monitoring 56
    viewing results 56

C

cache
    adding new reports 191
    configuring
        report executor 190
        report lists 190
    customizing reports 192
    reports 191
    scheduling 190
calculating
    bugs 174
    features/requirements 174
calculating defects/enhancements 174
Change 261
chk.sh and chk.cmd, running 194
ClearCase
    integrating with Report Center 276
clearing database 290
ClearQuest, configuring 247
CM Synergy. See Synergy CM. 277
code
    verifying tests 32
code metrics 96
code review history scanner
    integrating with 267
Concerto
    Linux menu options 293
    uninstalling 162, 290
configuring
    cache
        report executor 190
        report lists 190
    ClearQuest 247
    Report Center projects 186
    Report Center source code files 186
    SourceScanner 186, 275
creating
    initial database for Linux 300, 302
    sandboxes 196
customizing scripts 193
CVS
    integrating with Report Center 276

D

data collector
    activity 182
data, exporting/importing 287
database
    clearing 290
    statistics 182
deactivate baselines 128
defects

calculating enhancements
calculating 174
defining
repository files 275
deleting baselines 128
developer reports, generating 179
duplicate logs eradicator 181

E

Emma, integrating with Report Center 271
eradicator 181
errors
by category 76
by severity 84
exporting data 287
extracting scripts 193

F

features/requirements
calculating 174
files, static analysis 36
finding weak requirements 34

G

generating developer reports 179

H

HP Quality Center 235, 247

I

IBM Change 261
IBM Synergy 261
identifying weak requirements 34
importing data 287
integrating
bug tracking systems 215, 243
code review history scanner 267
Report Center with
ClearCase 276
CVS 276
Emma 271
StarTeam 277
SubVersion 277
Synergy CM 277
source control management 275

J

JIRA 253

L

License Server
activity 181
licenses
requests 68
server activity 66
verifying usage 70
viewing number granted 69
Linux
creating initial database 300, 302
managing Report Center 162, 291
options
Concerto 293
system administration 293
upgrading database 300, 302
locking down, sandboxes 196
logs 87
duplicate eradicator 181

M

managing
baselines 120
menu options (Linux)
Concerto 293
system administration 293
monitoring builds 56
multicast DNS 205
MySQL
changing root password 293
uninstalling 162, 290

P

password root (MySQL), changing 293
projects
assigning branch/tag 186
verifying schedules 29
with weak requirements 34

Q

Quality Center 235, 247

R

Rational Change 261
Rational Synergy 261
recalculating

views 174
refreshing reports 179
Report Center
configuring
projects 186
source code files 186
managing in Linux 162, 291
reports
caching 191
customizing cache 192
data collector 182
metric top results 96
refreshing 179
single metrics overview 97
reports (developer), generating 179
repository files, defining 275
requirements
weak 34
running BTS scanner 223

S

sandboxes, creating and locking down 196
scanning branches/tags 186
scheduling
cache 190
scripts
extracting and customizing 193
UNIX 193
Windows 193
send.sh and send.cmd, running 194
servers
settings 180
setting baselines 127
settings
server 180
size.sh and size.cmd, running 195
SOAtest Server settings 184
sorting tables 27
source code
statistics 96
verifying modifications 30
source code files, configuring for Report Center 186
source control management, integrating 275
source statistics, viewing 188
SourceScanner
configuring (general) 275
configuring to scan branches/tags 186
StarTeam
integrating with Report Center 277
static analysis
violations and files 36
statistics
database 182
statistics, source code 96
SubVersion
integrating with Report Center 277
Synergy 261
Synergy CM
integrating with Report Center 277
system administration, Linux menu options 293

T

tables, sorting 27
tags
assigning to project 186
scan configuration 186
test trio 120
token 114
tools, running 194
trio 120
troubleshooting
team server 306

U

uninstalling
Concerto 162, 290
MySQL 162, 290
unit testing
coverage 51
upgrading
database, for Linux 300, 302

V

verifying
- bugs decreasing 33
- code tested 32
- project schedules 29
- source code modifications 30

violations
- by developers 40
- by file 39
- by types 38
- static analysis 36

W

weak requirements 34