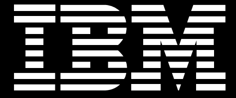
Development solutions
White paper
May 2008

Rational software



Automating static analysis to deliver higher-quality software.

Rational software, IBM Software Group

Contents

2 Helping to improve the quality of increasingly complex software and systems
3 What is static analysis?
5 Why use static analysis?
7 Rational Software Analyzer: broadening the scope and flexibility of static analysis
7 Analyzing code while it is being written
10 Displaying results
11 Exporting and reporting
14 Customizing rules and categories
18 Extending static analysis into existing build systems
19 Making software analysis an integrated part of ALM

Helping to improve the quality of increasingly complex software and systems

As software becomes more complex, the probability of exposing end users to program defects increases exponentially. It was once possible to modify code and then stage a review with fellow developers to identify any issues. However, modern software projects typically involve hundreds of classes and millions of lines of code, making peer review significantly less effective in helping to ensure software quality. Even the most conscientious peer review can miss defects because of inattention to the complexities of the code.

Additionally, the knowledge of overall code quality is often confined to development silos. This specialization can deny other stakeholders the opportunity to make informed decisions about overall product quality. Project and development managers monitor defect trends in software projects, but they often have little understanding of the challenges and details of the development team's work. Similarly, senior managers who need to make ship assessments lack a holistic understanding of code quality. Under these circumstances, the probability of offering customers substandard software products is unacceptably high. There is a clear benefit to pushing code review information into the application lifecycle management (ALM) and governance processes in the enterprise, but manual code reviews offer only a tedious and largely inaccurate route to this kind of information.

To help resolve some of the quality assurance weaknesses in the software development process, a range of static analysis tools has evolved to automatically detect and often correct common problems in the source code. Commercial static analysis tools have traditionally been expensive, and most of the freely available open source tools suffer from limited capabilities and crippling scalability issues. And while most commercial or open source tools can help the developer, few of these products provide the capability to capture and assess static analysis data and deliver it up the enterprise management chain to support effective governance of the software and systems lifecycle.

Highlights

The IBM Rational® Software Analyzer application provides a new means for assisting development organizations to produce better software by including static analysis information in the ALM and governance processes of the enterprise. Rational Software Analyzer is offered in two configurations: IBM Rational Software Analyzer Developer Edition software, which is designed as an easy-to-use developer tool to scan both Java™ and C/C++ code, and IBM Rational Software Analyzer Enterprise Edition software, which is designed to work seamlessly with IBM Rational Build Forge® software and can be integrated into most other build environments.

Organizations may be able to produce better software at a lower cost by integrating static analysis information into the software development process.

This white paper explores the basic concepts of static analysis and provides an overview of Rational Software Analyzer from three perspectives. First, it highlights out-of-the-box benefits from a user perspective. Then it examines the solution advantages from the point of view of developers who want to create their own analysis rules. Finally, it looks at how Rational Software Analyzer can help developers and build managers who want to integrate static analysis information into the software and systems build process.

What is static analysis?

Static analysis is many things to many people. If you look at the product landscape, you discover dozens of companies claiming to offer static analysis tools. Some companies focus only on C/C++ code review, while others offer only software metrics for Java code. Some analyze code for security problems for Web applications, and others scan code for dependency problems. Static analysis is a diverse and confusing concept that needs clarification. So what is it?

Quite literally, static analysis means the study of things that are not changing. However, in software terms, this definition should be refined to include the study of source and/or binary code that is not currently executing. To analyze running code, you need a debugger or profiler, but you can learn a lot from code without ever initiating a program.

Highlights

The IBM Rational Software Analyzer application enables three types of static code analysis to support routine software development: code review, code dependency and code complexity.

For example, if you simply parse all the source files for a program, you help to ensure that the source adheres to a predefined coding standard. You can also detect common performance problems, such as calling a method multiple times even though the result it produces does not change. You can even examine the imports of each class to understand which classes the code depends on or which classes depend on it. None of this requires the program to run—or even to compile.

There are of course many other types of static code analysis available, but table 1 captures the key types that relate to routine software development. These types are classified into three common categories based on the value they provide: code review, code dependency and code complexity. Rational Software Analyzer supports all three of these value categories.

Type of static analysis	Value
Code review	This type of static analysis tool automates code parsing. Each source file is loaded and passed through a parser, which looks for particular code patterns that violate a set of established rules. In some languages like C++, many of these rules are built into the compiler or available in external programs like Lint, a C program checker. In other languages, such as Java, the compiler does little in the way of automated code review. Code review software is a good tool to enforce coding standards, locate basic performance problems and find possible application programming interface (API) abuse. Note that code review software can also include deeper forms of analysis, such as data flow, control flow and type state.
Code dependency	Rather than examining the format of individual source files, code dependency tools examine the relationships between source files (typically classes) to build a map of the overall architecture of a program. Dependency tools are commonly used to discover known design patterns (good) or common antipatterns (bad) in code.
Code complexity	Code complexity tools analyze the program code and compare it to established software metrics to determine whether it is unnecessarily complex. If a particular piece of code exceeds a given threshold, it can be flagged as a candidate for refactoring to help improve maintainability.

Table 1: Rational Software Analyzer supports three broad categories of static code analysis capabilities.

Highlights

The benefits of static analysis extend well beyond reducing the time and expense of creating reliable software. Static analysis also helps enable development organizations to mitigate business risk and the high cost of customer-reported code defects.

Why use static analysis?

There are two basic reasons to embrace static analysis. The first is to reduce the time and cost of developing high-quality code. The second is to increase revenues and reduce business risk by providing reliable software to customers.

The first benefit of using static analysis tools is well documented. Many studies—including some done within IBM—claim that even simple automated code review will find 5 percent to 15 percent of all defects in code. These same studies claim that a defect reported by an organization's customers can cost between US$12,000 and US$18,000.* If you consider that a typical large and complex piece of software can have a thousand defects during its life span, you quickly realize why using automated code review tools can save between US$600,000 and US$2.7 million.

Certainly, avoiding customer-reported defects is the most obvious way to reduce the cost of code failure. You can address this goal with a comprehensive testing process; however, using static analysis tools such as code review offers a way to reduce costs even further. Figure 1 shows the benefits of finding defects earlier in the development process. Early detection can significantly reduce the costs involved in resolving issues. Using a simple automated code review provides an opportunity to start finding defects during the coding phase of a project—perhaps even while the developer is typing the code.

Highlights

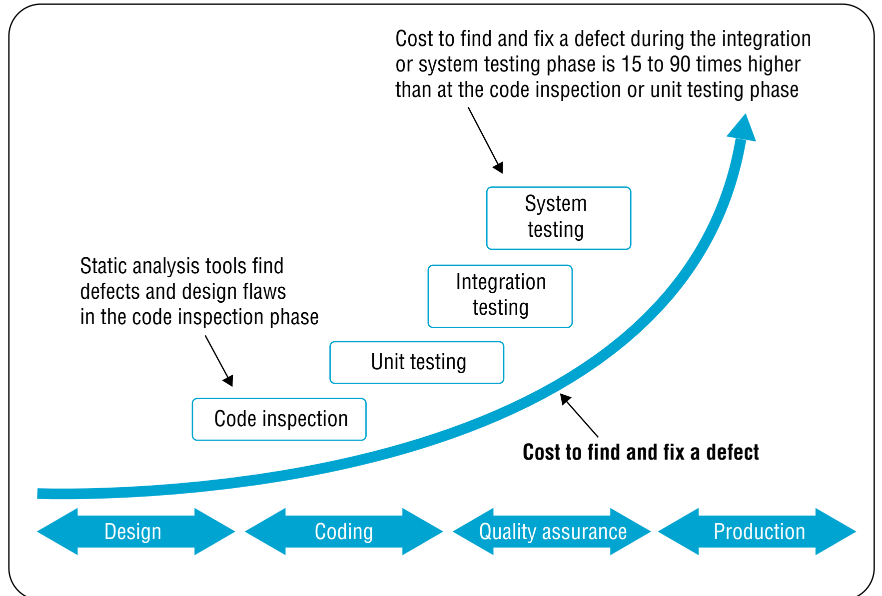


Figure 1: Defect repair costs rise dramatically the longer it takes to find and correct coding issues.

An organization that has a reputation for delivering reliable software is well positioned to increase sales to existing and new customers.

The second benefit of static analysis contributes to the success of the business as a whole by contributing to the success of those who use or purchase the software you've developed. Higher-quality code means they lose no time waiting for you to fix a defect they have reported. Furthermore, their business processes are not affected while they wait. When you have a reputation for providing reliable software, you can increase revenue from sales to existing and new customers. And when your software delivers the expected value, you run less risk of being sued. Customers today are more willing to pursue legal action against vendors when software fails to provide the expected business results.

Highlights

The Rational Software Analyzer application allows developers to check the quality of their code as they write it. Because results are available within seconds, static analysis can be run as often as desired without losing valuable development time.

Rational Software Analyzer: broadening the scope and flexibility of static analysis

Rational Software Analyzer is designed to help you address developer and enterprise needs in assessing code quality. First, it integrates tightly into an Eclipse, IBM Rational Software Architect or IBM Rational Application Developer workbench, allowing developers to analyze their code while it is being written. Second, it is available in both a command-line task form to support integration into existing build systems, as well as in a crawler program (such as Ant) task form to support information searches on the Web. Finally, extracting analysis results and generating reports both in the workbench and in exported forms, such as HTML, allows developers, project managers and executives to assess overall code quality.

Analyzing code while it is being written

Using Rational Software Analyzer, you can perform static analysis against your lines of code to improve quality as you write code—without losing valuable development time. Unlike the typical debug version of code, which can take minutes to execute, the IBM Rational static analysis tool enables you to check the quality of your code within seconds. It's almost like hitting your *Save* key, so you can run it as often as you like.

Of course, at the start of the coding activity, you must tell the static analysis tool what you want analyzed. As shown in figure 2, analysis configurations can be added or removed using the buttons in the top left part of the Rational Software Analyzer dialog box. As the name implies, a configuration is used to determine which forms of analysis and which rules should be executed, as well as the scope of analysis (for example, a project, a working set or the whole workspace).

Highlights

Assume you are in the early stages of developing a new feature for an existing Web-based application. To get started, you select the *Analysis* element in the *Configurations* list on the left side of the configuration dialog and then click the *New* button. You will notice that the right side of the dialog changes to show the basic configuration interface.

At the beginning of coding activity, developers must tell the Rational Software Analyzer application what it is they want analyzed.

The first step in creating an analysis configuration is to determine the default range of resources on which the analysis will be performed. This is accomplished by selecting the desired range on the *Scope* tab. The available options currently perform analysis of the entire workspace, a working set or a set of projects. In this instance, the *Analyze entire workspace* option is selected.

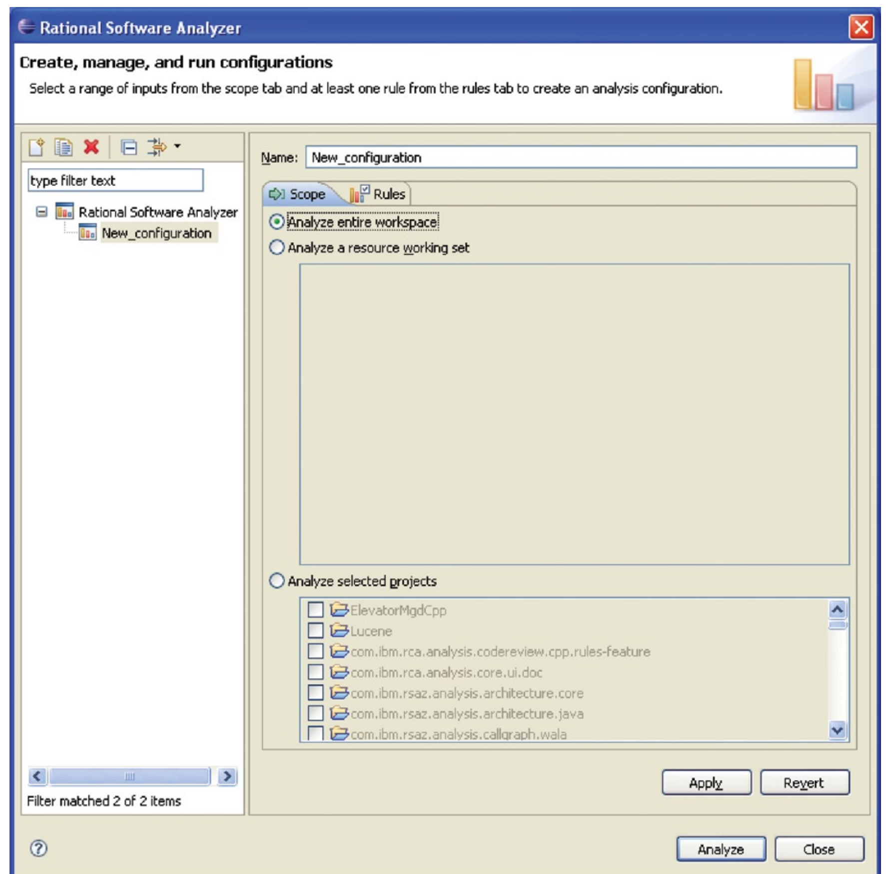


Figure 2: The first step in creating an analysis configuration is to determine the default range of resources on which the analysis will be performed—in this instance, the entire workspace.

Highlights

After the scope of the analysis is selected, the developer can quickly choose the analysis elements and import or export rules as required.

The *Rules* tab determines the forms of analysis that will be performed. As shown in figure 3, this tab displays a tree that allows you to select and deselect analysis elements. It also provides some additional buttons for importing and exporting rules. The top-most nodes of the domain tree are analysis providers, which represent the types of analysis tools that are known to the analysis framework. These analysis providers contain categories—loose organizations of rules and/or other categories. Rules perform all the heavy lifting by defining the conditions that generate results during the analysis process.

Note that each node in the tree is preceded by a checkbox that controls the enabling state of the element. When an element is checked or unchecked, all of its child nodes are set to the same state, which allows for quick selection of entire categories or even the entire tree. In this example, the *Java Code Review* branch is selected.

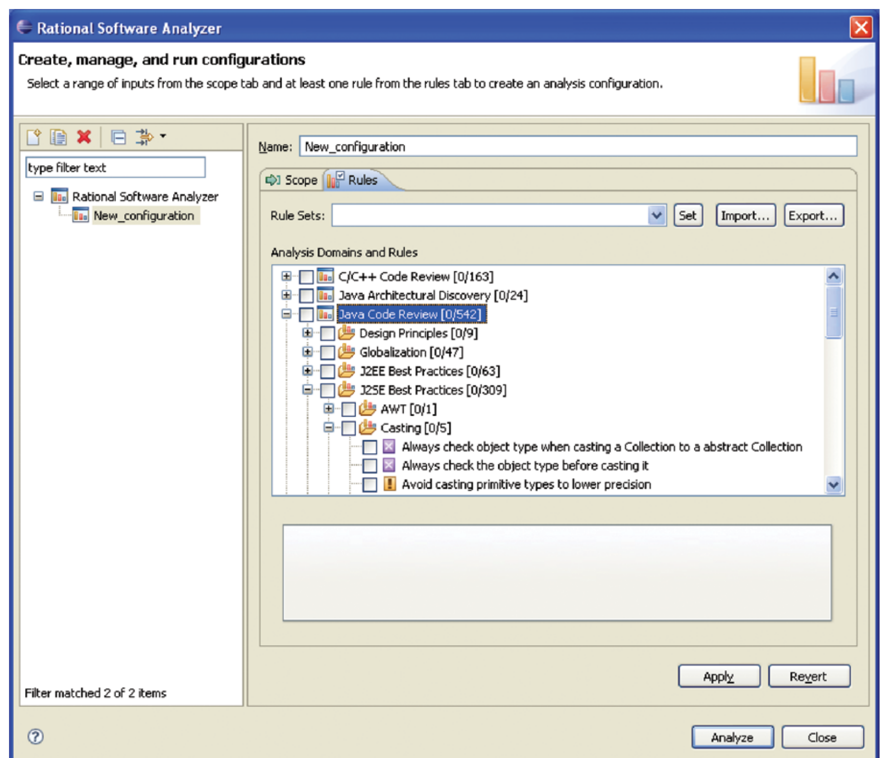


Figure 3: The Rules *tab on the Rational Software Analyzer dialog box displays a tree that enables rapid selection and deselection of analysis elements.*

Highlights

Depending on the kind of analysis you are performing, result views can be seen in multiple formats, including a table or a tree.

Displaying results

To start the analysis process, you click the *Analyze* button and the *Analysis Results* view appears in the Eclipse workbench. The results views may differ depending on the kind of analysis you are performing. In fact, some results views, like the one provided by Java code review, allow you to see results of the analysis in multiple formats (for example, a table or a tree).

If your analysis configuration contains selections for more than one type of analysis (in this case, code review and architectural discovery), the results view will include a tab for each analysis provider's results (see figure 4).

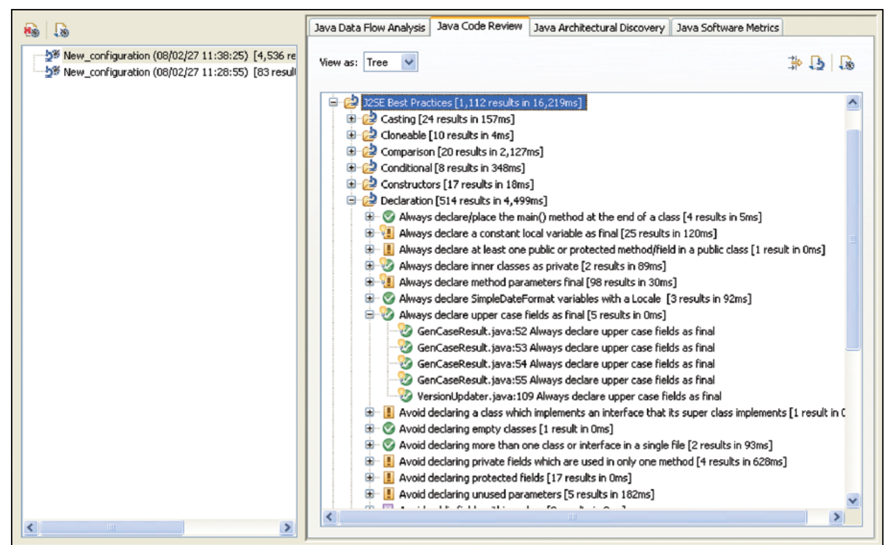


Figure 4: The Analysis Results *view includes a tab for each type of analysis selected in the analysis configuration. In this case, two types were specified: code review and architectural discovery.*

Right-clicking on a result provides a menu of available tasks you can perform, including *View Result* and *Ignore Result* to view or ignore the source code where the problem occurred. If the rule author has provided an automated defect correction routine for a rule, the *Quick Fix* menu option is enabled (see figure 5). Selecting this option walks you through the process of correcting the problem.

Highlights

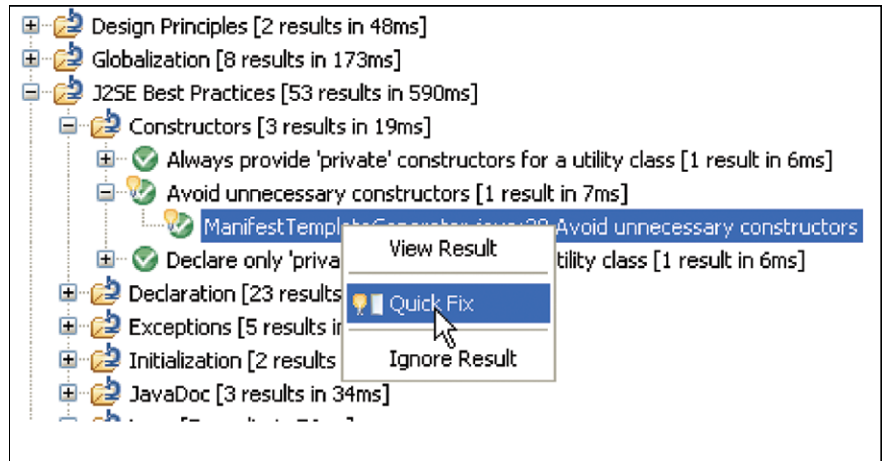


Figure 5: Rational Software Analyzer allows rule authors to provide automated quick-fix routines for their rules to help developers correct code errors quickly and accurately.

There is no common way to view a result. The Rational Software Analyzer application viewer used to render a result is determined by the rule author based on the type of data contained in the result.

It is important to note that the viewer used to render a result is a function of the type of data it contains. When viewing results, you might see a source file opened in the editor with highlighted text, or a Unified Modeling Language (UML) diagram or a table of statistical data. There is really no common way to view a result—the view is determined by the rule author. For the Java code review analysis provider, all results are viewed as editable Java source files.

Exporting and reporting

Depending on the type of analysis being performed, two other common functions may be available in the results view—data exporting and data reporting.

As the name implies, the data exporting function allows you to export the raw analysis results to a file and typically supports an XML file format. The type of data exported has been determined by the analysis provider, which will supply a list of known data exports so you can select the format you prefer (see figure 6).

Highlights

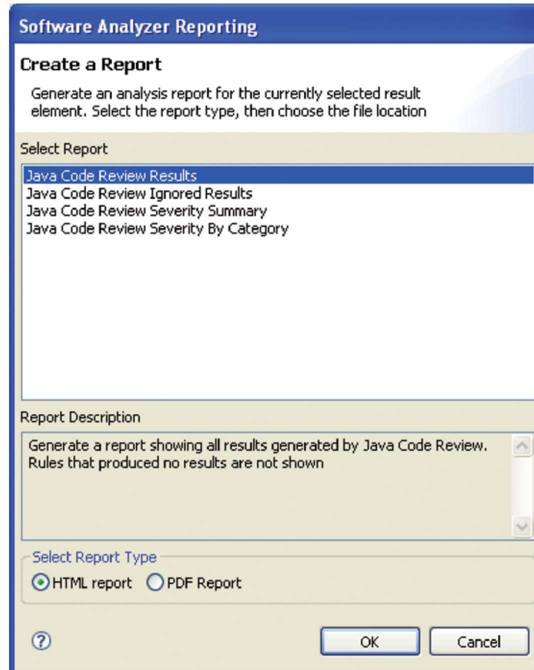


Figure 6: The Rational Software Analyzer supports multiple analysis reporting formats.

The Rational Software Analyzer data reporting function generates nicely formatted pages that can be printed, stored or written directly to a remote Web site.

The data reporting function is in many ways similar to the data exporting function. Both functions share exporters, but reporting generates nicely formatted pages that can be stored locally or written directly to a remote Web site. Developers or project managers can take an existing report file and modify it to suit their needs (for example, to support a company logo). Figures 7 and 8 show two of the many different kinds of reports that can be generated with Rational Software Analyzer.

Highlights

Using the report format that lists lines of problematic code and provides warnings and recommendations, a developer is able to address issues based on degree of severity.

Rational. software

Page: 1

Java Code Review

May 14, 2007 2:04 PM

Design Principles

Avoid methods with more than 3 parameters

/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/iAnalysisLaunchConfigurationDelegate.java:62

/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/AnalysisLaunchConfigurationDelegate.java:206

/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/IAnalysisResultView.java:26
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/ResultTab.java:50
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/ResultViewDefault.java:108

Avoid using the negation operator "!" more than 3 times
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/ResultsFrameView.java:314

Globalization

Avoid using java.lang.String + operator
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/UIMessages.java:20
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/actions/AbstractResultAction.java:100
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/actions/AbstractResultAction.java:125
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/dialogs/ExportDialog.java:204

Avoid using java.lang.String.equals() for multilingual strings
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/actions/AbstractResultAction.java:87
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/actions/AbstractResultAction.java:112
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/ResultViewDefault.java:220
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/ResultViewDefault.java:346
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/ResultViewDefault.java:355
/org.eclipse.tptp.platform.analysis.core.ui/src/org/eclipse/tptp/platform/analysis/core/ui/views/ResultViewDefault.java:361

Figure 7: This report format enables developers to navigate quickly through a list of problematic areas in the code they've created and right-click on each line to access the Quick Fix *routine, make a recommended design change or rewrite code to comply with best practices.*

Highlights

Whatever the format, Rational Software Analyzer reports enable developers, project leads and managers to review the kinds and scope of code quality issues—from those that are severe and could result in code failure to those that will work but could be improved by following recommended design principles.

The code analysis results provide warnings and recommendations. For example, the report shown in figure 7 lists lines of code that can be improved by following design principles like `Avoid methods with more than 3 parameters` and `Avoid using the negation operator "!" more than 3 times`. Warnings (preceded by an exclamation point and/or light bulb icon) indicate severe problems that can result in code failure as well as code with errors that could potentially affect application performance.

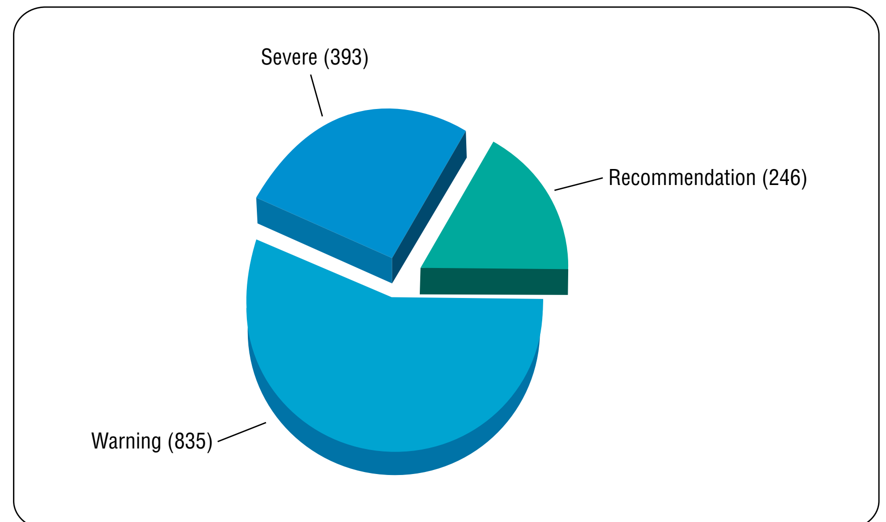


Figure 8: This metrics report format provides project leads and managers with a point-in-time assessment of the kinds and the scope of code quality issues.

Customizing rules and categories

In addition to the rules supplied by Rational Software Analyzer and the rules contributed by third-party developers, custom categories and custom rules for static analysis can be created from templates—without your having to write any code. To do so, you go to the *Preference* pages by selecting the *Window->Preferences* option. Then, in the Preference tree, you select the *Analysis->Custom Rules and Categories* page (see figure 9).

Highlights

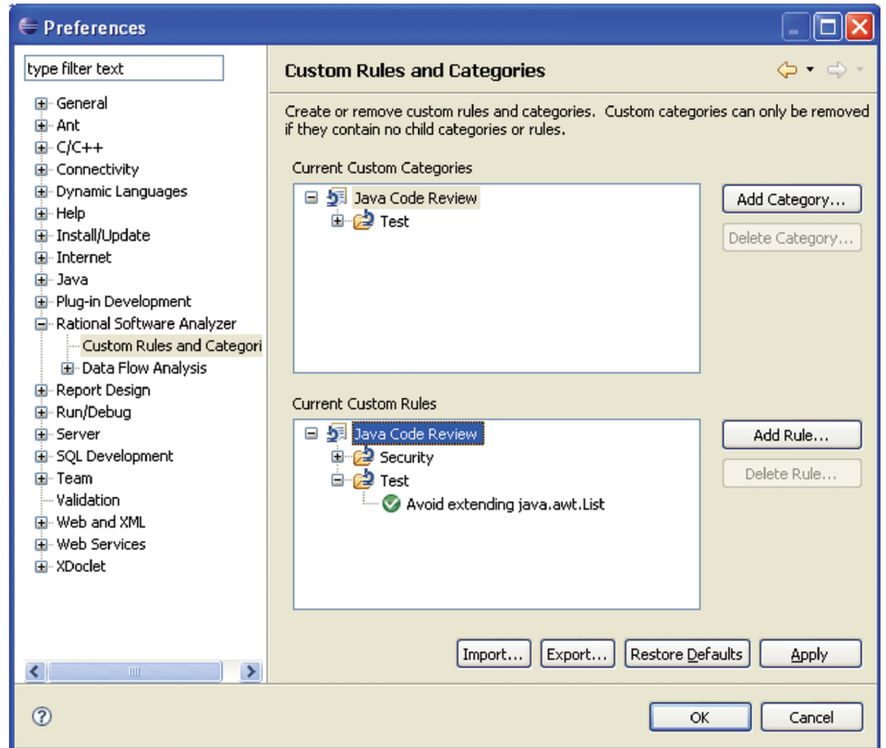


Figure 9: Rational Software Analyzer enables you to create custom categories and custom rules for static analysis using templates and wizards.

In addition to the categories and rules supplied by the Rational Software Analyzer application and those contributed by third-party developers, custom categories and custom rules for static analysis can be created from templates—without having to write any code.

Clicking on the *Add Category...* button activates a simple wizard that takes you through the process of choosing the parent category and naming your new category. The tree control for custom categories shows the complete path for any new categories you create. Note that only previously defined custom categories can be deleted.

Similarly, to start the rule creation wizard, you click the *Add Rule...* button. The first wizard screen allows you to select where in the analysis category tree you want the rule to be located.

Highlights

When a developer selects a static analysis category, such as* Java Code Review*, on the* Custom Rules and Categories *dialog box (shown on the previous page), she can add a rule by simply clicking on the* Add Rule… *button. This action takes her to a list of all known rule templates where she can select the rule template she wishes to use as a basis for the new rule.

The second wizard page provides a list of all known rule templates (see figure 10). You can select the rule template you wish to use as the basis for your new rule. Note that not all analysis providers support custom rules. However, Java Code Review supplies several that are at your disposal.

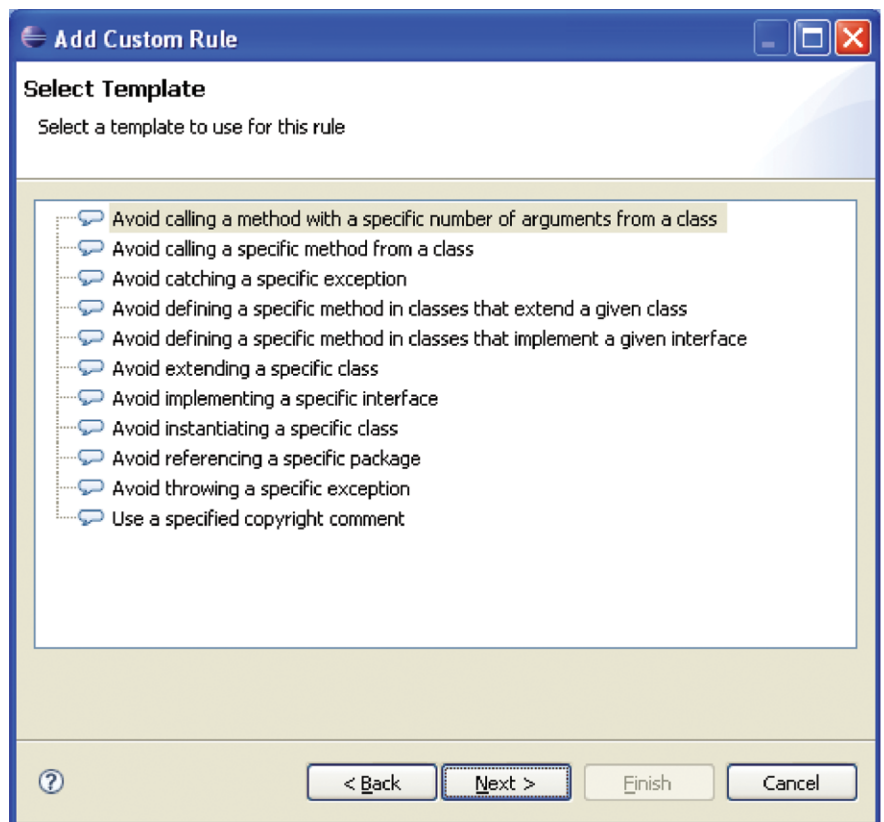


Figure 10: The Rational Software Analyzer custom code wizard provides a list of all known rule templates, in this case the rule templates for Java Code Review.

Highlights

After a new template-based rule is created and placed into the rule tree of the Rational Software Analyzer, any developer can select it as part of virtually any analysis configuration.

On the final screen of the wizard for rule creation, you see entries for each parameter defined in the rule template. For the example provided in figure 11, the selected rule template defines only one parameter, so you can enter only a qualified class name in the field provided. Note that a button is available to browse to an existing class, or you can manually enter a valid class name in the text box.

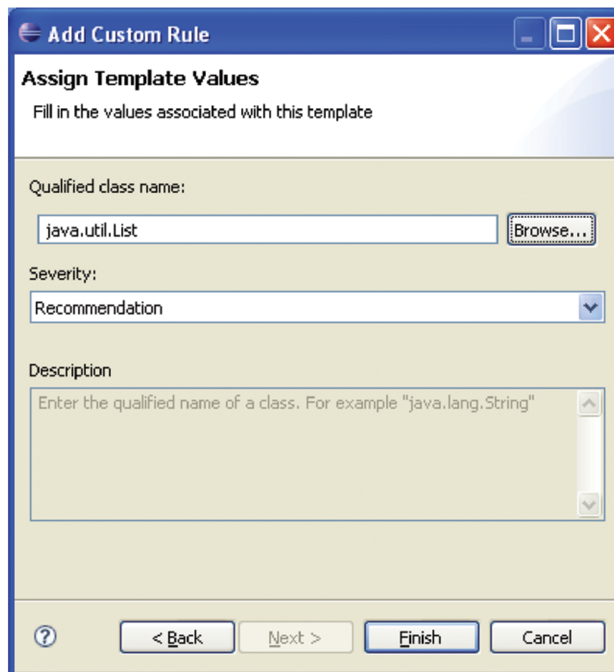


Figure 11: The selected rule template defines only one parameter.

Highlights

When you select the *Finish* button, the template-based rule is created and placed into the rule tree. Any developer is now free to select this rule as part of any analysis configuration.

Extending static analysis into existing build systems

As more organizations adopt the Agile approach to development, the scope of developer activities is expanding to include what has traditionally been considered production work. Developers are running their own builds and taking on a greater role in testing.

Because many developers are now running their own builds, IBM enables organizations to integrate static analysis into the software and systems build process.

The Rational Software Analyzer Enterprise Edition application is an extension of the Rational Software Analyzer Developer Edition application that provides the capability to plug into the enterprise build environment. It provides a full Eclipse-based user interface for defining analysis configurations. To integrate static analysis into the software and systems build process, the developer simply exports the configuration in the developer edition so it can be consumed by the command-line tool in the enterprise edition.

If Rational Build Forge is managing the build environment, this exported configuration can be specified in the Rational Build Forge adapter definition. If another build management tool is used, the configuration file can be specified using the Rational Software Analyzer Enterprise Edition command-line tool.

Highlights

Through integration and centralization of static analysis, development and build teams can identify and correct code defects earlier in the software delivery process— accelerating project timelines, reducing defect-related costs and enhancing the customer experience.

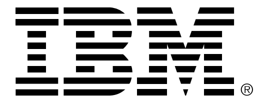
Making software analysis an integrated part of ALM

The IBM Rational Software Analyzer application helps organizations make software analysis an integrated part of the software and systems lifecycle. The tool is easy to install and use on the desktop. Virtually all contributing code can be reviewed, regardless of the source. With automated code analysis, developers can create higher-quality code in a shorter period of time. And through centralization of static analysis, development and build teams can gain deeper insight into whether overall IT governance and compliance standards are being met. They're able to identify and correct code defects earlier in the process—accelerating project timelines and reducing defect-related costs.

For more information

To learn more about IBM Rational Software Analyzer software, please contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/awdtools/swanalyzer



© Copyright IBM Corporation 2008

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-08
All Rights Reserved

Build Forge, IBM, the IBM logo and Rational are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product and service names may be the trademarks or service marks of others.

The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided “as is” without warranty of any kind, express or implied. In addition, this information is based on IBM’s current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

---

1 McKenney, Paul E. and Rumbaugh, Jim, “Static Analysis in Software Processes, Projects, and Products,” *IBM Academy of Technology Study,* July 22, 2007.

RAW14024-USEN-00