Transforming software development and delivery
White paper
April 2008

Rational. software

Best practices for software analysis.

An introduction to the IBM Rational Software Analyzer application

Contents

Introduction

As software becomes more complex, the probability of exposing end users to application defects increases exponentially. While it was once enough to modify code and then stage a code review with fellow developers, this practice frequently missed intricate coding problems. Today, even subtle quality problems can cause unexpected failures, potentially leading to lost business and a damaged reputation for product quality.

To go beyond peer review and improve time to market for complex applications, a range of static analysis tools has evolved that can automatically detect—and often correct—common source code problems. Although there is still room for improvement, static analysis tools are doing a good job of solving developer problems.

The IBM Rational® Software Analyzer application is designed to address the needs of your organization by automatically detecting and, in many cases, correcting coding problems. It adds value to the application lifecycle management (ALM) and governance process, helping to improve and raise awareness of overall product quality. It can also integrate into your existing developer environment and/or build system, adding value to the development process almost immediately.

Because Rational Software Analyzer can automatically review code, many users assume it is a black box in which you can insert bad code at one end and extract perfect code at the other. No static analysis tool can convert bad code into good code without some intervention. Instead, think of Rational Software Analyzer as a tool for highlighting areas of concern.

Highlights

As with any technology introduced into the development process, there are traps to avoid and tips that can help you derive immediate value. This paper explains the benefits of successfully introducing static analysis into your organization using Rational Software Analyzer. Additionally, it identifies some common pitfalls that can hinder the effective use of static analysis tooling. Here are 10 simple strategies designed to help you quickly realize the value of static analysis using Rational Software Analyzer.

Using static analysis tools to find a code defect is much less costly than resolving the defect after a customer finds it.

Heed the studies

A quick Internet search brings up dozens of studies showing that software quality improves with the use of automated static analysis tools. Depending on the study and type of analysis used, static analysis tools can find 5 percent to 30 percent of all defects in code. Many reliable studies have proved that if you wait until a customer finds one of your defects, resolution will cost US$15,000 to US$20,000. Finding these defects during development costs much less.

Even if you are quite conservative and believe that static analysis tools can discover only 1 percent of your defects, if you have 1,000 total defects, you can still potentially save US$150,000 on a typical project. But cost savings is just one part of the equation. You can also soften the blow to your business's reputation if you find yourself in a situation where you've shipped poor-quality code. IBM, for example, has lowered its defect rates 33 percent by introducing static analysis tools into some of its key applications. In fact, on one product we achieved a cost savings of more than US$250,000.

Highlights

No code is completely perfect, so it's important to establish parameters that can help ensure a reasonable level of quality.

Establish goals and expectations

There is no silver bullet in software analysis. You cannot put your bad code into a magic box, send it through the build process and get nicely formatted, high-quality code on the other side. The reality is that your code is probably never going to be perfect no matter what technologies are implemented in your development process.

To establish realistic expectations, it's important to outline some common goals and parameters that will help ensure a reasonable level of quality. Here are some examples of policies you might put in place:

- *For previously unanalyzed code, strive to eliminate 60 percent to 70 percent of the code review problems detected by Rational Software Analyzer.*
- *Mandate that your team scan any modified code using at least code review and data flow analysis before delivering it to the code management system. Ask developers to justify why they have not addressed all analysis results.*
- *Although Rational Software Analyzer allows users to ignore code review results, your team should not neglect individual source files with more than 10 ignored results. If it does, it should justify the exceptions.*
- *Comment/code ratio metrics should be within range before delivering the code. Good in-line documentation is the simplest form of code maintainability, and code should provide enough comments to allow a new developer to quickly understand and take over the code.*
- *Data flow analysis should produce no results (without justification). This type of analysis can identify possible resource and memory leaks, and the team should leverage it early to prevent subtle defects later on.*

Highlights

Many static analysis tools force developers into a particular workflow that makes them play by their rules. In contrast, Rational Software Analyzer has the flexibility necessary to fit into your organization and does not require you to modify your development process in order to benefit from static analysis. You can deploy it on developer desktops or as an integral part of the build process without affecting your quality goals or forcing you to change your existing procedures.

Use focused analysis

Rational Software Analyzer provides approximately 1,000 rules for various forms of analysis. Although there is a tendency to select all of them and perform static analysis in one step, you should avoid this. It's better to focus on a much smaller rule set so the results you produce are more manageable.

More focused rule sets yield smaller, more digestible results that you can deal with more practically; they also consume less scanning time.

For example, if you are executing basic code review, select the subset of rules that is focused on performance instead of the entire collection of rules. This will produce a much smaller and more digestible result set and more immediate results by helping to eliminate code that may be introducing performance problems. In addition to producing a more focused set of results, using a smaller subset of rules consumes less time for code scans.

The rule sets in Rational Software Analyzer have been divided into logical collections to simplify the task of selecting a focus. There are rule categories for performance, globalization, best practices and more, so try to focus on these categories to make your analysis more efficient. Although many static analysis tools assume that you want to run all analysis rules, Rational Software Analyzer gives you the freedom to select groups of rules or even individual rules.

Highlights

Start with small blocks of code

You might have millions of lines of code and want to get a complete set of results. However, you should avoid scanning the entire code base, especially if you are just starting to use a static analysis tool. Although Rational Software Analyzer can certainly handle a million-line code scan, you probably would not want to deal with the results. If the code you are scanning is legacy and has never been exposed to static analysis tools before, the scan is bound to generate massive amounts of data, which would inundate any development team.

It's better to analyze a much smaller collection of code, for example, the project, package or even the file that is being modified. Performing code review on more than a few thousand lines of code at one time is generally ineffective, so it's better to analyze only the code that is actively being modified, which will typically be in the range of 50 to 1,000 lines of code.

Because scanning an entire code base is usually costly and ineffective, Rational Software Analyzer Developer Edition lets you scan a more finely grained selection of code, such as a single project or an individual class.

Many static analysis tools assume the entire code base will be scanned, but this is usually ineffective and costly. In contrast, the IBM Rational Software Analyzer Developer Edition application offers a finely grained selection of the code to be scanned. For example, it can analyze the entire workspace, a single project or an individual class. This enables developers to quickly scan only the code they are editing without waiting for unmodified code to be examined unnecessarily, causing delays in the development process.

Beware of false positives

When describing static analysis results, it is often confusing to use the term "false positive." Developers sometimes claim they have false positive results, but they are really saying they do not like the answer—and there is a big difference between the two. A genuine false positive is one in which the static analysis tool reports something that is simply untrue; for example, failure to close a stream that is actually closed. This scenario is usually unique to deeper forms of analysis such as data flow and control flow analysis. However, most forms of static analysis involve detecting patterns in code or execution flow and rarely report a valid false positive.

Highlights

Rational Software Analyzer helps deal with false positives by enabling you to ignore results or turn off rules that don't apply to your situation.

With most static analysis tools, there is no convenient way to identify a genuine false positive, and as a result, the same invalid results appear every time the code is scanned. Rational Software Analyzer offers several ways around this problem. First, developers can assess analysis results and use a one-click action to ignore results, thereby avoiding any future distraction the false positive would cause. Additionally, because Rational Software Analyzer offers a finely grained selection of analysis rules, you can simply turn off rules that do not apply in your situation (e.g., there is no need to pay attention to globalization results if the code is targeted toward a single locale).

Don't be a slave to the tool

It would be unwise to assume that any static analysis is going to find and fix all of your coding problems. Static analyses are simply tools you can deploy to improve overall code quality by identifying the parts of your code that require attention. Developers should scan code as they are writing it and use full code analysis on daily builds to help ensure ongoing code quality.

With Rational Software Analyzer, you control what code to scan, what rules to enforce and what results to emphasize.

You also cannot assume that any static analysis tool has completely understood your code, processes or practices, or that every result the tool produces is something that should concern your entire development team. Instead, you should use static analysis tools as learning aids to help you avoid problem code in the future.

It's important to manage your coding effort and not let analysis tools control you. With Rational Software Analyzer, you can control what code is scanned, what rules are enforced and what results are important to you.

If you think a result is inappropriate for your situation, simply ignore it or disable the rules. Even though the results might be correct, not every rule will apply to your code. For example, not all users will find enforcement of the Java™ Javadoc tool useful. If your code is not part of an application programming interface (API), it is possible that Javadoc rules can be relaxed. Likewise, just because the structural analysis rules detected a code tangle does not mean you have a fundamental architectural problem. Depending on your design, it may be completely logical to have circular references in your code.

Highlights

The basic rule of thumb: If the results from a given rule do not apply or make sense for your design, then simply disable the rule so Rational Software Analyzer does not waste your time highlighting issues you already understand. Rational Software Analyzer is not designed to replace human code reviews. Instead it is designed to assist in improving manual code reviews by providing consistent analysis of common code problems.

To get the best results from static analysis, you should start with clean code that you have manually reviewed and cleansed.

Be prepared for a lot of results

Automated static analysis involves a complex set of processes that is designed to find common coding problems. To achieve the best experience from static analysis, you should make sure you start with reasonably clean code so the tool has an opportunity to find the real problems. We recommend that you manually review your code and clean it up before using any static analysis tool. The cleaner your code before the scan, the fewer initial results you will need to work through.

For example, if the Javadoc tool is important to your application, you should do a manual check of the code to make sure it is in place and that the code compiles and performs the intended tasks. Early in the software design, consider using a modeling tool such as the IBM Rational Software Architect application to help ensure that the code you are developing starts from a sound design. If your design or coding is poor, Rational Software Analyzer is not going to make perfect code. A relatively clean design and code base will give the analysis tool more opportunity to find real problems rather than simply highlighting problems associated with a poor design.

Of course, Rational Software Analyzer does not assume you write perfect code. Indeed if you did, there would be no reason to run the tool. However, analyzing code is time consuming, and any overly complex code is just going to add more time to the analysis process. Your first code scans are likely to generate many results, and to help you manage and understand these problems, Rational Software Analyzer Developer Edition offers a simple user interface. You can display results in a tree view organized by rule, or you can display them in table format, which you can sort by the result severity, rule type, file name, and so forth.

Highlights

To make sure one problem isn't masking another, you should address problems as they are discovered.

Rational Software Analyzer Enterprise Edition generates and publishes neatly formatted reports with the same result information, and you can view these reports in virtually any browser or print them. The goal is to always provide a simple path to view and manage a potentially large volume of information.

Make sure you address results

Rational Software Analyzer can find many problems consistently and rigorously. However, it is important to address the problems as they are discovered because one problem could be masking another. By ensuring that all reported results that require correction are logged as defects in your defect tracking system, you can reduce the number of results reported and markedly improve overall code quality, resulting in fewer bugs to be reported in the future.

One of the easiest ways to ensure that problems are addressed as you go is to raise their visibility. And Rational Software Analyzer Enterprise Edition can help you achieve the necessary visibility. It expands the reach of code analysis by offering centralized reporting and code scanning. You can also use the IBM Rational Build Forge® adapter to connect to build and release processes.

It's a good idea to make static analysis part of the daily build and to produce ongoing reports that highlight analysis results. Remember, finding and fixing problems early is the key to successful development. However, Rational Software Analyzer is only half of the solution; keeping everyone on the development team focused on fixing the reported results is the other.

Highlights

Static analysis tools should be used to improve overall software quality and to help improve developer skills—rather than to monitor individual performance.

Use static analysis as a tool, not as a weapon

Whenever a development team mandates using a tool such as Rational Software Analyzer, there is a tendency to focus on its potential negative implications. Developers often view these types of tools as a punch clock that's used to monitor their progress and penalize them when things go wrong. Managers who are introducing static analysis tools into either the build process or the developer workflow should present these tools in a way that encourages their use. It's important to emphasize the goal of improving overall software quality rather than monitoring individual developer capabilities.

Developers can actually exploit static analysis tools to help improve their skills. The tools provide a great deal of knowledge codified into a process for reporting common software issues. Developers can use this to improve their knowledge of more ideal coding techniques, which can directly benefit the entire team because not only will developers start writing better code, but there also will be more consistency among developers.

Rational Software Analyzer benefits both developers and managers. First, it offers many rules that can teach best practices to developers and help ensure that they maintain good coding practices. Second, it can be integrated into central enterprise builds for long-term monitoring of quality trends in code and to generate reports for project and development managers.

Because even simple coding issues can become serious after deployment, it's important to pay attention to style rules and coding conventions.

Pay attention to simple things

It is easy for developers to ignore simple issues in code that result in serious issues after deployment. Rational Software Analyzer includes many rules for style and coding convention enforcement that on the surface may seem trivial. However, even these simple code review rules can find subtle errors. For example, a style rule that enforces the placement of curly braces around all single line `if` statements will generate a result for the following code:

```
if( someValue )
    doSomething();
```

Highlights

This example seems simple, even though many developers would argue the case for avoiding curly braces in this code. What if the code looked like this?

```
if( someValue )
    doSomething();
    doSomethingElse();
```

In this example, the developer's intention is unclear because two lines are indented. The code could be correct with the wrong indentation, or it could in fact be incorrect if both indented lines were intended to be called within the `if` block.

Some other commercial analysis tools offer only data flow analysis or focus only on software metrics. Rational Software Analyzer offers these capabilities in addition to more basic analysis, thereby avoiding the need to run several tools. Although very simple rules typically focus on code compliance, even those rules can find big problems in your code.

Start improving your code quality today

The benefits of automated static analysis are well established. We all know that it can reduce effort, improve quality and save money. And if you use the simple strategies presented in this article, you can effectively introduce static analysis into your development environment and quickly benefit from it.

As applications become more complex and increasingly customer facing, developers need to leverage the many forms of analysis at their disposal.

The fact is, software development is challenging, and as systems become larger and more complex, the process of building quality code in a timely manner is becoming more and more difficult. The old techniques of peer development and manual code reviews are no longer sufficient, nor is it enough to focus on just one aspect of code quality. To build class-leading software today, you need to leverage many forms of analysis including software metrics, code review, architectural discovery and data flow analysis. Virtually the only tool available to handle these requirements today with an ability to smoothly adopt new capabilities in the future is Rational Software Analyzer.

For more information

To learn more about how IBM Rational Software Analyzer software can help you identify and rectify code bugs early in the development lifecycle, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/awdtools/swanalyzer

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

Produced in the United States of America
04-08

RAW14019-USEN-00