

Comment lines: Joey Bernal: What's in your development toolbox?

Skill Level: Introductory

[Anthony \(Joey\) Bernal](#) (abernal@us.ibm.com)
Executive IT Specialist
IBM

12 Nov 2008

Whether you believe function or style is the more important element in software development, discipline, efficiency, and consistency are almost always required for a development project to be a success. Static analysis, the task of reviewing application source code, is a method you and your team can use to fulfill these goals. This article explains the benefits of static analysis and important characteristics to look for in a static analysis tool.

But is it art?

We call it *software engineering*, but in the same breath we often insist that a software solution consists as much of art as it does of science. If this truly is the case, then we practitioners might try to use the title *software artist* rather than software engineer or software architect, though it's unclear whether someone with this title would be taken as seriously. Our goal as architects and engineers is to produce robust, working code that provides the desired results in a scalable and secure fashion. I will concede that there are occasional opportunities to use artistic license -- for example, when designing a user interface -- but it is still a matter of function rather than style that eventually impresses or satisfies the user. In other words, while a user might like your interface because it's pretty, they won't use it if it doesn't work.

While working on a project several years ago, my friend and colleague Yixing Gong commented that every component being built by our then project team was like an individual work of art by a particular developer -- when what we really needed at the time was to turn the effort into more of an assembly line process: crank out portlets

in less time with better quality and improved consistency. Even if all we could build at first was a Model T, continuous improvement would some day let us advance to building something more sleek and modern.

That initial discussion and subsequent experiences with many other projects eventually led me to write an early set of developerWorks articles on modeling portlets, and became the precursor to many of the development best practices I discuss with users today. Still, I am always looking for better ways to run projects, and to deliver robust and scalable applications on time and under budget.

Removing the “art” from the development effort could be looked at primarily as a matter of discipline, something that I think is lacking in many of today’s development teams. But the bulk of this discipline must come from within; that is, from the development team or the organization it is part of, since the business is focused on getting projects done on time and within budget, rather than the details of how things are accomplished. Short of having the business (which supports and funds IT efforts) require independent audits and reviews of ongoing work to enforce project discipline, what might help is a set of tools that enables us developers to perform measurements of our systems similar to how other types of engineers measure their work.

Such a “Software Engineer’s Toolkit” would consist of a useful set of key components that would measure elements of a project or solution for consistent and correct results. The same way a carpenter’s set of tools consists of a tape measure and a level to ensure a cabinet is the right size and is installed the right way, software can be measured for similar consistencies. Of course, there is no one toolkit for all developers and situations, but there are fundamental tools and capabilities that we should all have on hand. **Static analysis** is one of these, and it is one of the more powerful. It also seems to be one of the least used development tools, so perhaps by outlining the basics of static analysis and when it might be helpful and appropriate to use within your organization or development project, I can advance the spread of project discipline and single-handedly improve the quality and value of projects industry-wide. (I can dream.)

Static analysis

The term *static analysis* is often equated with the idea of automated bug searching in software. While there are many types of potential problems that static analysis can help find, it is not a catch-all bug-finding device. In fact, it is often difficult to find bugs by just performing a simple static analysis. This is especially true if you have run time bugs in your production system. What static analysis can do is become the catalyst to help you understand the overall quality of the code, help you identify pointers where problems might occur, and identify areas where you can perform additional analysis.

Static analysis is the task of reviewing application source code files without actually running the programs. In truth, static analysis can be performed on byte code as well as on source code files, which means that an application can be compiled frequently, depending on what you are looking for and which approach you choose. For the purpose of this article, we will focus mainly on source code reviews.

In my research I often try new tools. I look for those that have the best array of functionality, and these are often the most popular. There are many open source tools and commercial products for static analysis available, and you can find a list of them on [Wikipedia](#). I'm not going to describe them all here, but I do want to mention four as typical examples to give you an idea of full-featured commercial options and readily-available open source tools. It might take several open source tools to match the functionality of one commercial product. The tools listed here are not presented as recommendations, but they are well known in the industry and can provide you with an understanding of important features with which you can perform your own product comparison and feature analysis.

There has been some analysis of source code versus byte code analysis and which provides the better results, but that discussion is beyond the scope of this article. I should say that the extent of analysis that is used within your project should not be affected by either approach; in other words, performing either type of static analysis is better than performing none.

- **FindBugs**

[Findbugs](#) is a popular open source tool that performs static analysis on Java™ byte code to find common programming errors. Findbugs can be included into your build as either an [Ant](#) task or a [Maven Findbugs plug-in](#). Most build systems use one of these approaches, so it would be easy to incorporate this minimal set of information into your system.

Like other static analysis tools, FindBugs enables you to build new rules into your analysis. Unlike some of the other tools, however, Findbugs uses byte code and the [Byte Code Engineering Library](#), so it can be a challenge to learn how to implement new rules within the system.

- **IBM Rational Application Developer**

IBM®Rational® Application Developer V7 provides built-in error detection and static analysis tools with several hundred rules that have good descriptions around them. Also included are many samples and examples that outline many errors and how you might fix them.

IBM Rational Software Architect adds additional rules, such as Architectural Discovery for Java and UML Model Analysis. If you are already using Rational Application Developer on your team, this could be a natural first step to implementing a feedback system within your project

or organization. Additional capabilities are available with plug-ins.

- **CodePro Analytix**

CodePro Analytix by [Instantiations](#) is a commercial product that is available as either a standalone or Eclipse plug-in. With more than 900 rules in about 35 categories, CodePro can look at your code in several ways, generating unit tests for different aspects of the code.

As a commercial product, CodePro comes with additional capabilities, including a dynamic code audit facility that catches bugs as you write code, which could help novice programmers learn about mistakes they might be making and understand options for fixing problems.

Another aspect of building discipline within the development cycle is to understand and measure many of the metrics hidden within your code base. Standard metrics, such as lines of code and number of classes, are important when measuring the amount of effort that is being generated for a particular project. While individual development effort cannot always be assessed with such general statistics, these numbers can be used as a guideline to establish benchmarks and for project estimating purposes.

One interesting CodePro feature is the ability to graph dependency analysis on your projects. I find this aspect of static analysis important when trying to understand how an application works, and to ensure that a layered approach is being followed within the system architecture.

- **Parasoft Jtest**

The static analysis capabilities within [Parasoft Jtest](#), another commercial product, consist of over 700 parameterizable rules in over 35 categories. Its static analysis analyzes source code and byte code, is pattern-based as well as flow- and path-based, and is capable of finding rule violations that cross methods, classes, and packages. In addition to all of the rules, the Quick Fix option enables developers to quickly fix many issues that Parasoft Jtest finds on the fly. The RuleWizard module lets you quickly build your own set of rules within the system, which is as easy to do as cutting and pasting the code you want to flag and letting Rule Wizard auto-create the rule. Parasoft Jtest's flow-based analysis, called BugDetective, statically simulates application execution paths, and then uncovers run time errors that could occur in these paths.

Parasoft Jtest can examine your code in several different ways and automatically generate JUnit, Cactus, and HttpUnit test cases for testing things like access and boundary type exceptions. Manually written tests can also be incorporated into the test run, and functional JUnit test cases can be added by running your use cases on the working application.

Additionally, Parasoft Jtest can generate a regression test suite that alerts you when code modifications impact existing functionality. Like other commercial products, Parasoft Jtest can also calculate metrics about your code and development process so you can measure project growth and progress, as well as automate your code review by allowing teams to integrate code review into their IDE.

Beyond the tools

Occasionally, there might be things you look at beyond what the analysis tools provide. For example, when performing a static code review I might do a file-wide search for all the occurrences of a specific item; for example, all the times that `HTTPSession` or `PortletSession` is used to store something, which can be a trouble spot in many applications. But simply knowing how many times the session is accessed, or what data types are being stored in the session are not very useful on their own. For this information to have value, I would review each instance found with the development team directly to identify proper usage and ask questions, such as:

- How much data is usually stored in this structure?
- How often is this data accessed by the system?
- Is the data common across many customers or is it unique to each customer?
- Can we move common data to a separate cache or tier?

By having this type of discussion, you can better see if the code is engineered in a consistent fashion. Unfortunately, these types of scenarios are not often defined well enough to offer you a complete list of potential trouble spots. It is mostly a matter of personal experience on the part of the reviewer, but as the industry matures, best practices such as these will become better defined.

When you boil it all down, what we have described is something that might commonly be called a code review, although it does not need to be as rigid as the name implies. I rarely see code reviews in practice, possibly because many project leaders either don't feel well versed enough in the code to make constructive suggestions, or they are concerned about exposing their lack of knowledge to the development team. If you are worried about the latter, here's a hint: the development team probably already knows! Get over it and start making your project better by getting involved and asking the right questions. This is the core of project leadership. The value of this second tier analysis should be apparent, but beyond that you get a clearer picture of the application structure and the specifics of the code. Is the development team following the architecture model you defined? Is the model clear

enough, or are there multiple interpretations of how things should be structured?

Discipline

This takes a lot of discipline. Of course, you can't expect a development team to have discipline if we as technical leaders do not have any ourselves. I touched on this earlier, but I really want to emphasize this point. You cannot expect a development team to follow standards and conventions if you have not defined them appropriately within your organization. Discipline starts with the team lead by making sure rules and standards are defined at the beginning of a project. With standards and code in place that all engineers have to follow, inspections can ensure that projects continue only when standards have been followed. This consistency and discipline promotes communication and leads to success for both the business and its users.

The key to making anything work is to act on your analysis in some way to ensure that all developers understand what the team together is trying to accomplish. The obvious follow-up to a static analysis review is to conduct a manual code review, during which everyone actually looks at code as a team to comment, make recommendations, or maybe even learn something about how good code should be put together.

A new book by Dorota Huizinga and Adam Kolawa (CEO and cofounder of Parasoft) called [Automated Defect Prevention](#), takes exactly the right approach in helping organizations instill some discipline into the software development process. Again, static analysis is not about fixing immediate problems (although it can sometimes do that), it is about building a process and continuous improvements into your organization or development team.

Experience

One of the things that teams using static analysis tools often do is turn on all of the rules and then run the analysis. This can be troublesome because the tool could potentially generate hundreds or thousands of potential errors that you won't want to wade through. It takes an experienced developer within the current framework to help the team understand which rules are important. That is not to say that some rules, like formatting rules, are not important, just that they should be weighed in comparison to other issues, such as null pointers or mishandled exceptions.

Formatting and documentation consistency is extremely important in terms of understanding and maintaining your code base, but it is also important to understand that if you let out too much at one time, it will be more painful for your team, so easing the incorporation of new rules and development of new standards within the team is the right approach. For follow-on code reviews, you do not need a major guru to lead the effort, although you might have one on your team. Manual reviews are a team effort where discussion is the tool, not criticism by one person, and no

managers are allowed. This is a chance for everyone to learn and grow as a team. And remember as always, have fun!

Summary

Getting started with static analysis is more than just buying a tool and adding it into your environment. Or is it? If you can make a commitment to performing that simple step, it can go a long way toward instilling the right amount of discipline into your process. The key is what you do with the results of any such analysis, and how consistent the analysis is run within the process. One single run right before you go live is probably not enough to help, but weekly runs followed by a team discussion can help bring your project to the next level, and provide value that far outweighs the time that your reviews will absorb. *We don't have time to review code*, is a common refrain, but reality and experience shows us that taking the time to instill some discipline into your process will save more time later in the project, as well as improve the caliber of all the developers, and the quality of the project.

I recommend taking the time now to look at the right product mix and build some static analysis into your project. Taking baby steps is the way to get this working the right way in your environment and for your development process. The payoff might not be immediately tangible, but in the long term, you will see a stronger set of deliverables and more discipline in your organization than you might have thought possible.

Resources

- [Wikipedia: List of statis analysis tools](#)
- [Findbugs](#)
- [Automated Defect Prevention](#)
- [Byte Code Engineering Library](#)
- [Modeling portlets](#)
- [IBM developerWorks WebSphere](#)

About the author

Anthony (Joey) Bernal

Anthony (Joey) Bernal is an Executive IT Specialist with IBM Software Services for Lotus, and a member of the WebSphere Portal practice. He has an extensive background in the design and development of portal and Web applications. He is the author and co-author of several books, including *Application Architecture for WebSphere*; *Programming Portlets 2nd Edition*; *Programming Portlets*, the *IBM Portal Solutions Guide for Practitioners*; and from a previous life, *Professional Site Server 3.0*. He also contributes to his popular blog, Portal in Action.