

Using static code analysis for Agile software development

Andrew Yang

3/23/2010 2:10 AM EDT

Since the goal of Agile development is to have working software early, source code analysis enables developers to analyze the quality and security of code from day one of coding " one of the earliest points in the software development process Source code analysis (sometimes called "static analysis") is a technology which analyzes source code for the purpose of detecting defects, understanding architecture, collecting statistics on the software and more.

One of the most prominent commercial uses of static analysis is for defect detection. Vendors like Coverity, Klocwork, Fortify Software and others have sophisticated products that analyze the structure of the code and detect anomalies that can lead to real bugs.

For instance, many of these products can analyze paths in the code to find situations where memory may be allocated but not freed, signaling a potential memory leak. There are many other types of checks that can be performed to detect program crashes, security vulnerabilities, concurrency problems and more just by examining the source code.

Source code analysis appears to be a silver bullet for software quality and security. It doesn't operate at runtime so it requires no test cases. Therefore, you can find interesting problems even before the code is remotely operational.

It also finds problems right in the source code, pointing to the specific line in the code to where the problem exists. It doesn't get much easier to identify and fix problems. By contrast, fixing a bug reported in the field often requires a good testcase and a debugging process to locate the specific problem. It can take days to find and fix a bug.

Of course, source code analysis is not without its challenges, namely that it finds problems by making educated guesses. While the algorithms may be sophisticated, it has no knowledge of artifacts outside of the actual source code, such as assumptions in the environment or code that might be linked in afterwards.

The analysis may simply not have strong enough algorithms to handle strange and complex code. This can lead to false positives " bugs reported by the tool which are not in fact, bugs.

Older static analysis tools were notoriously "noisy", producing thousands of false positives that rendered the tools unusable. Modern source code analysis tools are significantly better but can still generate a lot of noise if not configured properly for the specific code they are analyzing.

In addition, process, organizational, tool chain integration, performance, tool configuration and other such considerations greatly affect the effective use and adoption of source code analysis.

Keep in mind also that source code analysis is designed to find defects in coding. It is a highly useful verification tool in the quality and security arsenal but not a good validation tool. It does not determine

that code has met business requirements. Source code analysis helps existing quality practices, such as code review, but does not replace them.

Why source code analysis is ideal for Agile

Teams that have adopted Agile or some variant of Agile have derived a significant benefit from source code analysis. The very nature of Agile is to have working software early. Source code analysis enables developers to analyze the quality and security of code from day one of coding " one of the earliest points in the software development process.

In test driven development (TDD) environments, as most Agile shops are, developers are used to creating test cases early in the process -- oftentimes before any code is written. In source code analysis, the "test cases" are already built in the tool (although initial configuration is often still required). Helping developers build quality into their code during development is a critical requirement for most organizations.

Feedback is fast. These tools can analyze a million lines of code in a few hours producing thousands of potential defects. A typical day's worth of developer changes should take minutes to tens of minutes to analyze. A developer should be able to analyze his or her code and fix problems multiple times before check-in (often called, "clean before check-in")

Source code analysis work particularly well in automated environments. Many organizations using Agile development have automated their build and test environments. Source code analysis plugs neatly into this process.

The breadth of analysis from modern source code analysis tools fits well with the concept of continuous integration. One of the key benefits of continuous integration is the ability to test all components of the software working together early and often.

Modern tools perform interprocedural analysis which analyzes the code as a whole entity rather than analyzing individual functions. Thus, source code analysis is a relevant automated test for integration-related bugs.

Not A Well Worn Path

Software development organizations undergoing Agile are seeing the light when it comes to source code analysis. But many organizations have also experienced the many pitfalls that can occur because change is always hard and the technology is relatively new. We outline a few high level tips so that you don't have to reinvent the wheel.

Goals. The most common problem we see is a lack of defined goals. Organizations clearly want defects to be found and fixed early in the development process (they wouldn't have bought the tool otherwise) however when they deploy and rollout the tool, the technology, process and rollout are all set up as voluntary.

Not surprisingly, with pressing deadlines, the source code analysis results are ignored over time particularly by the developers who arguably need it the most.

One of the more successful strategies we've seen is taking a two-pronged approach. Static analysis results can be divided into bugs that already exist in your code (the "backlog") and defects that are found in recently changed code.

Organizations should have a process for dealing with the backlog but should also adopt a policy where no

new defects may be introduced. Some call this the "new no defects" or "you break it, you fix it" models.

Automate. Integration with the build process is vital. When source code analysis is a separate, non-critical path item, it will get ignored. Identify the gates where source code analysis *must* occur and where it *should* occur to support the goals. For instance, the following milestones are places where static analysis is often a requirement:

- * Branch pull, to establish a baseline
- * Integration from development branch into integration or main branch to ensure that no defects are being introduced
- * Release to ensure that no defects are being introduced

To help meet these goals with minimal risk, process architects enable static analysis to be performed in:

- * Developer sandbox so that developers can find and fix problems prior to check-in
- * Continuous integration build to provide feedback often
- * Nightly analysis as part of full nightly build

Finally, bad results from source code analysis tool should be able to stop a release.

Make it Easy For Your Developers

Workflow automation and process integration reduce the additional steps that must be taken to utilize source code analysis. Some set up work to consider includes:

- 1) Using automated alerts to let developers know their status and when they are in violation of any metric
- 2) Integrating it into your process to reduce new steps. For instance, source code analysis defects can be prioritized and seamlessly plugged into your change management / bug tracking process with the right process and integration. Dumping all source code analysis bugs into the bug tracking system is not recommended however, smart integrations can make this process seamless and scalable.
- 3) Leveraging your assets " whether it's LDAP to make it easier to log in, bug tracking integration or automatic assignment of defects by components or by who last touched the code, these steps will not only make it easier for developers and improve adoption but also improve efficiency.

Make Performance Fit Within Your Requirements

On the one hand, source code analysis is working very quickly " by processing at times millions of paths in the code to find dozens of classes of problems. However, when operating a business, developer time is of paramount concern.

Some analyses take hours and even days to perform which may be at odds with the desired cycle times of Agile and continuous integration. There are many techniques for improving the performance of static analysis, sometimes dramatically.

These performance optimizations should be taken to fit the overall wall clock time to one that fits within the process. We've seen smart tuning and optimization result in five to ten times faster analyses.

It's Change Management

Installing the tool and making it available to developers is far from "mission accomplished." Treat rollout like any other change management task. Changing people's behaviors doesn't happen overnight. While no article can do justice to all of the intricacies of change management, here are a few points to consider:

- 1) Internal champions with credibility in the organization must exist to provide vision and shepherd the process throughout the organization
- 2) Management support must be strong. What are the incentives and consequences if the tool is not used? Does it truly have support from the tools team, the development organization, product management and QA?
- 3) Goals must be aligned with management. It is highly recommended for users to have a very clear idea of what success and failure are.
- 4) Users must be trained and have the necessary support ongoing to be successful. Sending them to call the vendor's support line may not be sufficient.

There is no cookie cutter approach to instituting source code analysis in an organization using Agile methodology. In fact, there are as many flavors of Agile as there are companies "doing Agile", each customizing the methodology to their specific situation.

The Agile methodology also stresses "individuals and interactions" over "tools and processes". This is clearly the main area of focus. That being said, source code analysis is a highly valuable tool that improves quality and security at the most opportune time during Agile.

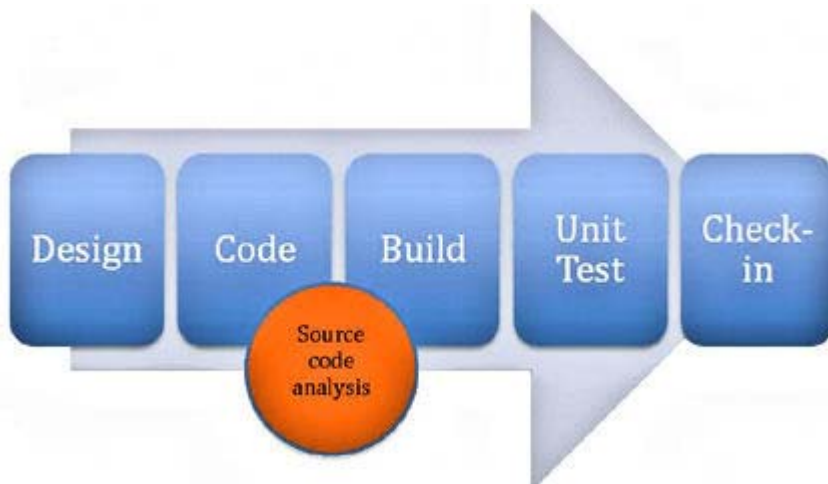


Figure 1. Source code analysis requires no testcases and can find defects in non-operational code.

Source code analysis (**Figure 1, above**) plays a pivotal role in increasing efficiency, improving output of software engineers and helping organizations deliver working software faster and more predictably. However, the time and investment required to make it truly work rarely is sufficiently allocated.

Clearly organizations shouldn't devote valuable resources to becoming experts outside of their core competence, but they do need to devote enough expertise and resource to make their sometimes-deep investments pay off handsomely.

Source code analysis helps developers find and fix problems in the earliest part of the development process. Source code analysis requires no testcases and can analyze and find defects in non-operational code. It can be performed in a number of critical points during a typical Agile environment (**Figure 2, below**).

Main Trunk

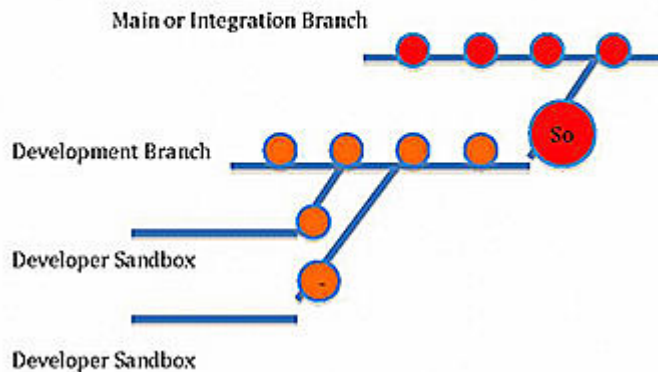


Figure 2. Where static analysis should occur during the agile development process.

In this particular example, the red circles represent where static analysis should occur. The orange circles represent where it should occur to help developers meet their criteria early and often.

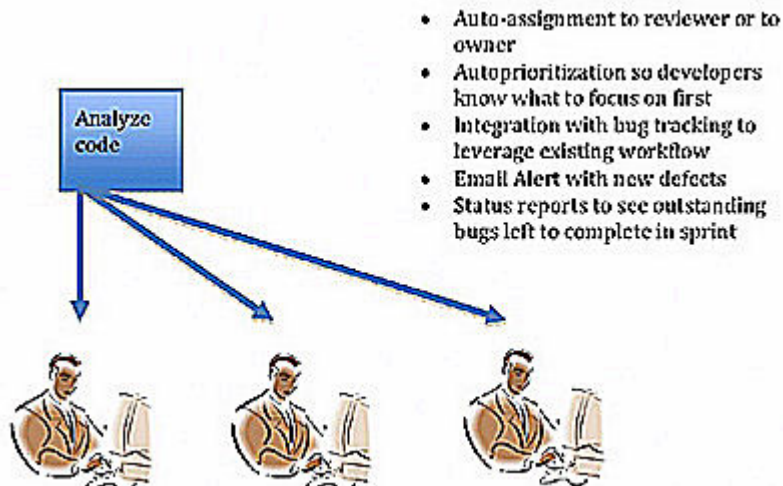


Figure 3. Key steps in the Agile static analysis process include auto-assignment, autoprioritization, integration with bug tracking, email alerts and status reports.

When a branch collapses into an integration branch or into main, it needs to fit the acceptance criteria that no new defects are introduced. Daily analyses on the main branch (**Figure 3, above**) ensure quality and security issues are being addressed often and serve as a baseline for branch pulls. Automation improves efficiency and reduces error, and fewer steps in the development process improves adoption and usage.

Andrew Yang is Managing Director and Co-Founder of [Code Integrity Solutions](#), a professional services firm specializing in source code analysis and software build consulting. Andrew can be reached at

andy@codeintegritysolutions.com.