Automating Software Development with Static Code Analysis

A paper by Technology Strategy Research
A Practical Guide to Getting the Most Out of Agile Development Processes

No application development professional doubts the value of code reviews during development. The value of standing down for a day or two to enable the entire team to examine source code lies in enabling the team to better enforce coding standards, identify potential problems early in the development process, and ensure that all developers have a clear and unified view of the programming strategy and tactics.

The increasing shift to Agile development methodologies is taking a toll on the ability of the team to conduct formal code reviews. Agile methodologies are typically iterative, delivering new code and features after sprints that last anywhere between a week to a month. The result is that many Agile development methodologies are primarily focused on producing code and rarely have time to stop code production for a day or more to perform manual code reviews.

This lack of time doesn’t mean that Agile development teams have to do without this valuable tool. A process called static code analysis can automate code reviews, which makes it possible to perform code reviews on a regular basis, even in batch, and analyze the results on an ongoing basis. In effect, teams can automate almost every aspect of a code review, from ensuring coding standards are being maintained to identifying errors, often before the code is even executed.

The Value of Code Review

Code reviews play an essential role in any software development process. They enable a development team to step away from the routine and take a different look at the state of the application. The typical process is for the development team (or functional groups if the team is a large one) to meet in a conference room for one to two days of intense scrutiny of its source code. In the past, these reviews were usually conducted with hard-copy printouts all around, but today most developers use laptop computers and wireless access to the source code control platform to find and examine code files, and can often make changes on the fly.

These teams work to review virtually every line of code in every method, with the goal of making the code better. Teams look at several aspects of the code:

- **Compliance to coding standards.** Most organizations have standards for writing code that are intended to improve code readability and understanding by others. These include naming conventions for variables and methods, method lengths, and parameter passing conventions.

- **Application design properties.** Teams determine most design characteristics during application design, not implementation. However, code reviews can identify default user interface configurations and data settings that can make a difference to user acceptance.

- **Correctness of implementation.** This encompasses both conformance with requirements and specifications as well as correct coding. While many errors can’t be identified without actually executing the code, errors such as poor logic can be found through examining the source code.

- **Use of programming language(s).** Many programming languages and frameworks today are large and complex. Developers may not be expert on all of the language features and may use an incorrect or sub-optimal construct. A code review can leverage group expertise in ensuring that language and framework features are appropriately used.
Best practices. Best practices can include coding for application performance, internationalization, and date formatting. Best practices also include code that minimizes security holes.

The code review also provides the opportunity to review the design goals of the project and to discuss the tactics used in achieving those goals. While designs are unlikely to change, it provides the opportunity to emphasize important aspects of the project.

In some Agile projects, manual code reviews may be significantly less formal—having a colleague look at your code, sharing a file via an email or wiki, or even paired programming. These approaches offer some of the benefits of a full code review with less time and effort but can fail to capture important aspects of good coding practices and quality.

Automating the Code Review with Static Source Code Analysis

The ability to stop writing code for a period of time for a formal review is difficult in an Agile process because there is rarely time allotted to stop the development process in this manner. Fortunately, virtually all of the activities done by a team code review can be automated through static source code analysis. Static source code analysis typically operates as a rule-based expert system. It scans and parses the code, then looks for matches with the rules in its database.

The key, then, is the rules database, which codifies the intelligence that goes into the analysis of the source code. Rules define the practices that teams seek to achieve through the manual code review process. Using static source code analysis, teams can apply code reviews consistently and with a minimum of time. Once rules are defined, they are applied without debate to the code, saving discussion time in a formal meeting. While the Agile team can discuss the results, the analysis itself typically takes only a few minutes.

The static source code analysis typically produces reports on deviations from the best practices as defined by the rules. Ideally, the tool not only notes the deviation, but offers an explanation of how the code deviates from the standard and may even provide reference to further information on the topic. This provides a learning opportunity on topics where individual developers may be weak.

One thing that can be difficult for static source code analysis to capture is experience and institutional memory. This makes it imperative for a static code analysis tool to enable development teams to customize existing rules and add rules of their own. This is especially true regarding coding...
standards, which can vary significantly between organizations. Individual teams may also have their own standards concerning performance, design characteristics, and best practices based on their unique applications and deployment environments. Further, custom rules allow teams to focus on the coding practices that are unique and important in their efforts, rather than attempting to catch any and all questionable practices. From a process standpoint, automated code reviews can be run in conjunction with the build process.

This means that a static analysis tool should be able to be integrated into the build process, and be executed in batch. It also has to be able to produce reports that can be disseminated across the development team to enable team members to immediately address the items identified.

In addition, individual developers can employ their own code review, by running static source code analysis locally on their own code. Teams can use this as a gate to pass through to check in new or modified files into source code control.

**Steps to Improving Code in an Agile Process**

Automating code reviews using a static source code analysis tool can bring significant benefits to an Agile development team during iterations and before delivering a complete application. Because it doesn’t require standing down from development for a day or two, static analysis can fit right into an Agile process.

But applying static code analysis requires a strategy for utilizing it during iterations, feeding the results back into the Agile process and improving code quality dynamically while it is being developed. While the exact strategy will depend on the team’s Agile process and build approach, many if not all of these steps can be undertaken by a team seeking to replicate the results of code reviews in an accelerated process.

1. Set up static source code analysis on developers’ local computers and require items flagged to be corrected prior to check in.
2. Incorporate static source code analysis as a part of the nightly or continuous build process, executing prior to the actual build. This sequencing may flag items that can prevent a successful build or smoke test run and save time the next morning if the build process was incomplete.
3. Formally review the results of code reviews during Agile stand-up meetings, highlighting common errors and recommending corrective action if necessary.
4. Document items that have occurred on multiple occasions and apply remedies such as developer training or lessons learned to improve the skills and efficiency of the team.
5. Run full analyses at the conclusion of each iteration. Determine which items should be addressed prior to deployment of that iteration and which can be deferred to the next iteration.

Code reviews remain one of the most effective ways for teams to ensure code quality and identify and correct errors early in the development process. Agile processes can make it difficult to take the time to do traditional code reviews, necessitating an automated approach to examining source code and identifying errors, variations from standards, and other issues.

Fortunately, static source code analysis provides a way to automate code reviews, giving Agile teams much of the same advantage as a code review but without the investment in time. Static source code analysis reviews code and flags deviations without the need for extensive meetings involving the entire team.

The primary goal for any Agile team is high quality code, developed rapidly. Static source code analysis can be an essential tool for achieving just that. Agile development teams can have the best of both worlds—fast and flexible development with the quality found with code reviews.