Code Review Best Practices

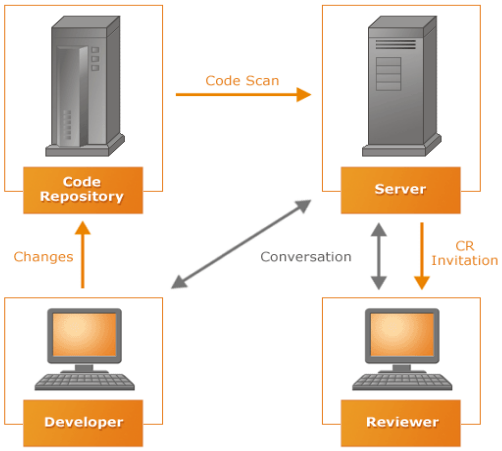
With Adam Kolawa, Ph.D.

This paper is part of a series of interviews in which Adam Kolawa—Parasoft CEO and *Automated Defect Prevention: Best Practices in Software Management* (Wiley-IEEE, 2007) co-author—discusses why, when, and how to apply essential software verification methods. The series also addresses code analysis, unit testing, memory error detection, message/protocol testing, functional testing, and load testing.

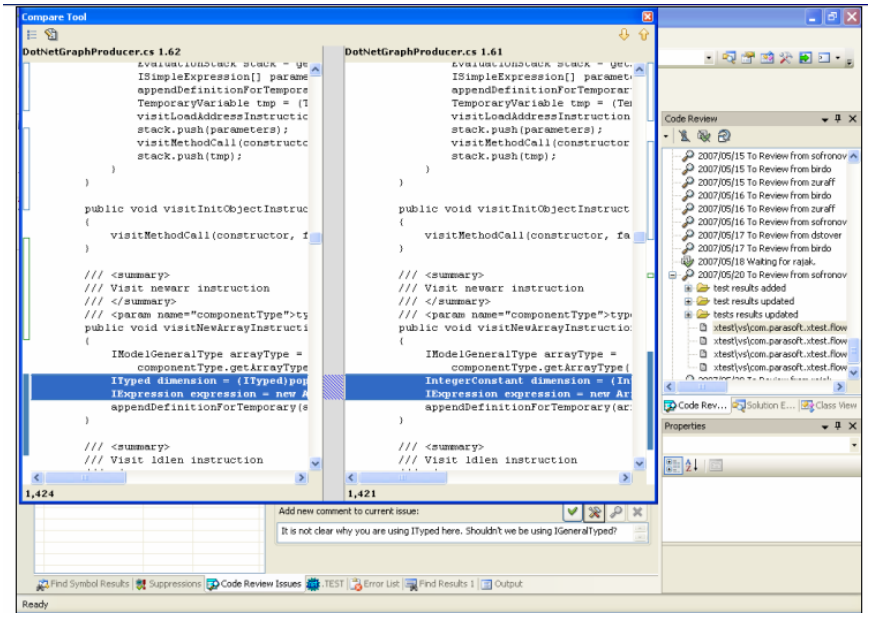
Different people mean different things by code review. What's your definition?

First off, I think that the only practical peer code review process—for centrally-located teams as well as geographically-distributed ones—is one that that's managed automatically. For example, using Parasoft's Code Review module:

1. Developers check in code to the source code repository as normal.
2. A server-driven code review scanner automatically detects what code needs to be reviewed, generates code review packages that show the difference between the new code and the old code, and automatically notifies the designated reviewer(s) that a review is needed.



3. Reviews are performed at each reviewer's convenience from his familiar IDE (Eclipse, Visual Studio, RAD, Wind River Workbench, ARM RVDS, etc.).



4. After examining each change, the reviewer either accepts it or requests additional revisions.
5. If additional revisions are requested, the author is notified of the request, and the cycle continues.

In a nutshell, that's how to make the code reviews practical.

To make them effective, they must be tied to requirements, with reviewers focused on determining if the code is actually doing what it's supposed to be doing. If you don't do this, the code review typically degenerates into developers scanning the code for problems that automated code analysis could find. That's a shame because code analysis tools could do this scanning faster, better, and more accurately—and developers could be doing something much more valuable and interesting.

These two things are actually quite closely related: with a computer handling all of the tasks that are not creative, the brain can focus on thinking about the code in terms of the requirements.

Why is this focus on requirements so important?

It's the best way to identify improperly-implemented requirements—one of the three main categories of defects (along with missing requirements and poorly-designed interfaces that allow users to wander in unintended directions).

With at least one other person inspecting the code and thinking about it in context of the requirements, you gain a very good assessment of whether the code is really doing what it's supposed to. Automated analysis simply cannot uncover these algorithmic functional issues—this high-level analysis requires a human brain.

And what's the value of the workflow automation? What do you have against a good old-fashioned sit-down code review?

Well, with sit-down code reviews, the cost usually outweighs the benefit.

The cost is significantly higher with sit-down code reviews. First, everyone has to figure out a time and place that's convenient to meet. This is difficult for centrally-located teams, and nearly impossible for many geographically-distributed teams. Then the developers have to spend lots of time on preparation—trying to remember what code was changed and why, marking the changes, correlating the new version and the old version, and so on. Finally, you have all the time required for the review itself.

What's worse, the benefit is typically lower with sit-down code reviews. They typically uncover fewer problems than automated code reviews do. Why? When everyone is sitting together in a room, they don't give themselves enough time to really think through the code, identify all the possible problems, and come up with viable solutions. The brain does not work instantly; it needs time to think. That's why it's significantly more valuable to have a code review process that allows you to review code at your desktop, at your convenience, with enough time to vet potential problems and determine how to make the code more effective.

How does all this time thinking about the code affect the team's productivity?

Surprisingly, it actually improves productivity.

First, it enables early error detection—and the more defects you identify in code review, the fewer you need to test out later in the process, when it's exponentially more difficult, costly, and time-consuming to do so.

Second, when one developer reviews another's code, he learns about code complexities and connections he would not know about otherwise. This ends up expanding the amount of code that each developer can work on, which enables the team to accomplish more with its existing resources.

Third, when a developer receives feedback on his code, he becomes a better developer. This is not only invaluable for getting new developers up to speed rapidly, but it's also key to promoting the entire team's continued growth. With all developers engaged in this "continuing education" process, developers end up accomplishing more because they are working smarter.

Of course, a good review process will take up some time, but with automation, the additional work and level of disruption are kept to a minimum. When you consider the scope of the benefits it delivers, you definitely come out ahead in the long run.

Since we're on the topic of time, let's move on to the "when" part of "why, when how"—when in the SDLC should code review be performed?

Like any other quality task, it should be a continuous process that runs throughout the SDLC. Any time code is modified, it should be reviewed immediately—whether it's your first attempt at implementing some new functionality, or a bug fix made right before the release. In a sense, it's an intelligent form of continuous regression testing: checking whether modifications introduce high-level issues such as design weaknesses, algorithmic issues, and complex defects that automated tools can't catch.

How does it fit in with respect to other quality tasks?

Code review should follow static analysis for two reasons.

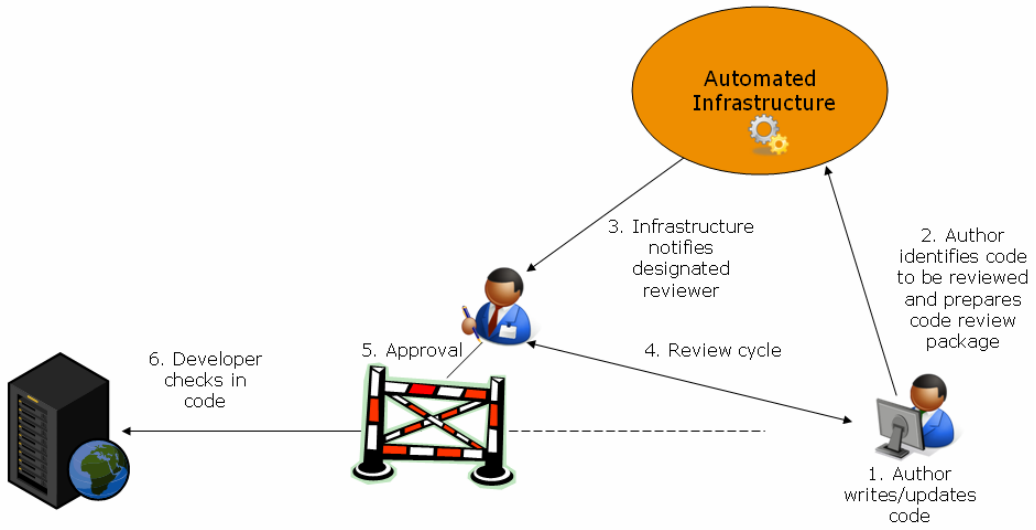
First, when teams are truly committed to running static analysis with a rule set customized to suit their policies and priorities, it eliminates the need for line-by-line inspections during peer code reviews. Reviews can then focus on examining algorithms, reviewing design, and searching for subtle errors that automatic tools cannot detect. This really takes a lot of the drudgery out of the peer code review process—making it not only more productive, but also more engaging for developers.

Second, static analysis results can be valuable inputs to code review. For instance, if metrics analysis helps you zero in on overly-complex code, you can ensure that it's examined during peer code review. If a developer does not agree that a certain static analysis rule violation should be fixed (e.g., because they don't think that the rule applies in the given context), then the team can use the code review to discuss whether this violation is a prime candidate for a "suppression." Also, code review can be a good opportunity to analyze the defects reported by flow-based static analysis; in particular, to determine if they might be indicators of missing requirements.

How does each review fit into the daily development workflow?

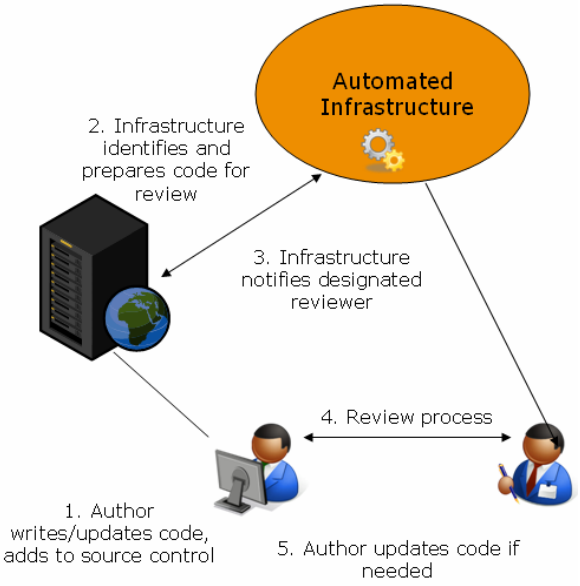
There are two ways to do it: before checking code in to source control (pre-commit), or after code is checked in (post-commit). I strongly recommend the latter because it's much less disruptive.

With a pre-commit process, the developer must do a lot of extra work before adding code to source control. Even with an automated infrastructure managing key tasks, the developer has to remember to initiate the process that identifies code ready for review and enter a description of the intended changes.



Developers might attempt to follow this process initially, but project progress will likely slow to a crawl as a result of the additional work and the new roadblock to checking in code. Eventually, peer code reviews will be nothing more than a bad memory.

With post-commit code review, the burden on the developer is dramatically reduced. He just checks in the code as normal. The only added work is the actual peer review—a creative process that cannot, and should not, be automated.



In both cases, the review packages are automatically distributed to the designated reviewer's IDE. This way, he sees them first thing in the morning, and can review and respond to them from his regular working environment.

By going with the workflow rather than against it, you really increase your chances of sustainability and success. If code reviews are too much of a burden, you'll have a hard time convincing the team that it's worth their time.

In addition to this recommended workflow, do you have any other tips on making sure the process doesn't decay?

Well, if you want to keep the process on track, you also need a way to measure and monitor it. The team might claim to be doing reviews religiously, but how do you really know if reviews are being done regularly, if everyone is participating, and if it's really worth the effort?

When the process is automated and connected to a reporting system like Parasoft Concerto's Report Center, you can actually get objective answers to these questions. You just look at graphs that provide real-time visibility into whether reviews are being performed, whether identified issues are actually being resolved, and how this is impacting team productivity as well as application quality.

If an organization wants to get started with static analysis, what do you recommend as the first step?

Start by defining your code review policy. Do you want the team to review:

- Every line of code that they touch from now on?
- Any code that your metrics calculations identify as being overly complex (e.g., Cyclomatic Complexity over 10)?
- Only code in certain parts of the project?
- Code that meets some custom criteria for requiring manual inspection (as defined in custom static analysis rules)?

You also need to define author to reviewer mappings that makes sense based on your group's dynamics. For instance:

- If you have a new developer, you might want his work reviewed by two or three other developers.
- If you're doing Extreme Programming, you'll want to set up one-to-one mappings to automate pair programming.
- If different team members are working on common interfaces, having them review each other's work will keep them in synch.
- If you have a handful of isolated contractors working on a project, you might want the project's architect or lead developer to review all of the contractors' work to ensure that they are in synch with his expectations and with one another.

You can adopt whatever policy suits your needs. The key is to determine what you can accomplish realistically (given your goals and the available resources), then configure your system to automatically flag code that meets your review criteria, prepare review packages, and route them to the appropriate reviewers.

Parasoft Services can help you define a code review policy based on your organization's unique needs, then establish a system that automates policy application and monitors policy compliance.

About Adam Kolawa

Adam Kolawa, Parasoft co-founder and CEO, is considered an authority on the topic of software development and the leading innovator in promoting proven methods for enabling a continuous process for software quality. In 2007, eWeek recognized him as one of the 100 Most Influential People in IT.

Kolawa has co-authored two books—*Automated Defect Prevention: Best Practices in Software Management* (Wiley-IEEE, 2007) and *Bulletproofing Web Applications* (Wiley, 2001)—and contributed a chapter to O'Reilly's *Beautiful Code* book. He has also written or contributed to hundreds of commentary pieces and technical articles for publications such as The Wall Street Journal, CIO, Computerworld, and Dr. Dobb's Journal, as well as authored numerous scientific papers on physics and parallel processing.

Kolawa holds a Ph.D. in theoretical physics from the California Institute of Technology. He has been granted 15 patents for software technologies he has invented.

About Parasoft Code Review Solutions

Parasoft solutions provide centralized code review policy application and management across heterogeneous environments (Java, C, C++, and .NET languages). They take a unique two-pronged approach, where automated code analysis works in concert with automated peer code review workflow automation and management.

Parasoft's automated code analysis relieves developers from having to perform line-by-line inspections during peer code reviews, freeing them to focus on higher-level analyses that require human intelligence. With automated checking for compliance to the team's and organization's development policies, the team can begin peer code reviews by discussing interesting findings from the automated code analysis results, then move on to examining high-level design, algorithmic, and implementation issues.

Parasoft also automates and manages the peer code review workflow to address the known shortcomings of this very powerful inspection method. Parasoft's code review module automatically identifies updated code, prepares review packages, matches the code with designated reviewers, and tracks the progress of each review item until closure. By automating critical workflow components, teams gain more time to focus on productive tasks. Moreover, managers gain real-time visibility into review status, trends, and impacts—enabling them to ensure that this process remains on track.

Parasoft Services can help you define code review and code analysis policies based on your organization's unique needs, then establish a system that automates policy application and monitors policy compliance across the organization.

To learn more, contact Parasoft as described below, or visit http://www.parasoft.com/solutions.

About Parasoft

For 20 years, Parasoft has investigated how and why software errors are introduced into applications. Our solutions leverage this research to deliver quality as a continuous process throughout the SDLC. This promotes strong code foundations, solid functional components, and robust business processes. Whether you are delivering Service-Oriented Architectures (SOA), evolving legacy systems, or improving quality processes—draw on our expertise and award-winning products to increase productivity and the quality of your business applications. For more information visit: http://www.parasoft.com.

Contacting Parasoft

USA

101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Europe

France: Tel: +33 (1) 64 89 26 00
UK: Tel: +44 (0)1923 858005
Germany: Tel: +49 89 4613323-0
Email: info-europe@parasoft.com

Asia

Tel: +886 2 6636-8090
Email: info-psa@parasoft.com

Other Locations

See http://www.parasoft.com/jsp/pr/contacts.jsp?itemId=268