Peer Code Review: An Agile Process

A SmartBear White Paper

Contrary to what many believe, the benefits of code review closely align to the tenets outlined in the Agile Manifesto. Statistics prove that peer code review is one of the most effective ways to improve software quality by reducing defects upstream. By aligning a peer code review approach with your specific goals and Agile sprints, code review becomes incredibly Agile and delivers many soft benefits that evolve from renewed focus on interaction and collaboration. Case study synopses provide quantitative evidence and metrics that support the case for Agile code review.

Contents
Moving Beyond the “Code Review” Stigma .................................. 2
How Does Code Review Align with Agile? ............................... 3
The Agile Case for Code Review .............................................. 3
Saving More than $200K: Case Study ..................................... 4
The Economics of Peer Code Review ....................................... 4
Case Study Overview ............................................................. 5
Case Study: Cisco ............................................................... 6
Case Study Overview ............................................................. 7
Types of Lightweight Code Review .......................................... 8
Best Practice Techniques for Code Reviews .............................. 8
Timing Agile Code Reviews ................................................... 9
Useful Metrics to Assess Agile Reviews .................................. 9
Overcoming Resistance .......................................................... 10

Summary .................................................................................... 11
About CodeCollaborator ......................................................... 12
About SmartBear ................................................................. 12
What Makes Peer Code Review Agile?

Some Agile practitioners consider peer code review an unwelcome legacy that persists from the dark ages of waterfall development. As a result, they vehemently reject its inclusion in Agile projects. This paper demonstrates both how code reviews reinforce the key tenets that underlie Agile development philosophies, and also how code reviews can be conducted in a manner that aligns perfectly with Agile processes.

The fundamental principles of Agile development are discussed at length on the Agile Manifesto web site. The essential Agile Manifesto tenets value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Although Agile followers acknowledge that the concepts on the right have value, they value the concepts on the left more. The premise is to adapt to changes in requirements, respond to user needs faster, streamline processes, decrease development iterations, and enable higher productivity. All this enhances business value by making development more transparent and getting software to the market faster.

So how can code review support Agile processes?

Consistently, research shows that code review produces software with fewer defects, which aligns with the emphasis on working software. What could be more interactive than software developers collaborating about the code and making real-time improvements? As you’ll learn in this paper, code review fully supports Agile tenets by promoting the development of working software, collaboration and interaction among teams, continuous attention to technical excellence and the ability to respond to change—all while maintaining a high level of quality.

Consider three, well-known Agile shops: Apple, Google, and Microsoft. Like most successful companies that attest to being Agile, they enjoy high product adoption and loyal customers. Unlike Microsoft, Apple and Google are sticklers for code review, and their product quality reflects this added effort. Apple in particular delivers major upgrades and fixes to the market at an impressive pace. Both companies closely tie code review to their development culture; they “bake” code reviews into Agile sprints and iterations to identify potential defects earlier in their cycles. At these companies, code is never “checked-in” before it is reviewed.

So why do so many software development teams resist adopting code review when industry titans attest to its success?

Moving Beyond the “Code Review” Stigma

Historically, the process for conducting code review has been pretty “anti-Agile.” As originally conceived by Michael Fagan in 1976, code inspections\(^1\) were a heavyweight code review process that led to an entire generation of software developers who believed that meetings were required to review code. Lots of meetings. Boring meetings that emphasized processes and tools over individuals and interactions.

The result is that today, heavyweight processes and meetings are regarded unfavorably by Agile devotees. This stigma tainted the entire concept of code review, causing code reviews to be perceived as an impediment by Agile project teams. Although this “guilt by association” has eroded over time, misconceptions still linger.

The biggest misconception is that you need a meeting to do code review. Although Fagan mandated meetings for his inspections, Lawrence Votta of AT&T Bell Labs was unconvinced that meetings were required. His study\(^2\) showed that when developers read the code to identify defects before the meeting, actually having the meeting only increased the total defects found by 4% (and consumed several hours of valuable time per meeting participant).


Recent studies confirm that code review can be effective without meetings. Jason Cohen, founder of SmartBear Software®, conducted a study at Cisco Systems® which demonstrated that a lightweight peer code review approach was not only as effective as a heavyweight code inspection process, but also more time-efficient by a factor of 7.

Yet even Agile devotees who recognize that meetings are not required have misconceptions about code review. They say, “We only use the latest techniques here; code review is from the past and provides no value,” or “All the unit tests pass, so why do we need to do code reviews?” If you take away the meetings and the heavyweight process, but leave the interaction, responsiveness, and dedication to continuous improvement, then code review is very much an Agile practice.

**How Does Code Review Align With Agile?**

For developers to be open to the notion of adopting code review in Agile environments, they must first accept the premise that code review can be agile. Delving into principles that underlie the Agile Manifesto provides specific evidence that code review is Agile.

**Working software is the primary measure of progress.** Software developers make mistakes. Some mistakes are detected automatically by unit tests, static analysis tools, etc. Just as professional writers rely on human editors and spell-check software to identify mistakes, developers benefit from having other developers examine their code.

During iterations, developers devote a lot of time discussing requirements with stakeholders and talk about the emerging design and architecture with other software developers. That’s true even when they tend to work in isolation to actually write the code (giving developers a reputation as introverts, at the very least). We know and accept that flaws are uncovered during discussions of requirements, architecture, and design. The same principle applies to the writing of the code. Code reviews not only uncover flaws, but they also offer another key benefit prized by Agilists – the feedback is kept close to the point of creation and happens sooner – before the code gets into the hands of either QA or customers.

**Agile processes promote sustainable development.** Project sponsors, developers, and users should be able to maintain a constant pace indefinitely. That’s a tall order. To meet this objective, Agile teams frequently practice collective code ownership. The goal here is to invite team members to get familiar with and understand the code base.

Collective code ownership fuels sustainable development in many ways, because it:

- Transforms team member focus from “specialist” to “generalist” in certain domains.
- Helps to eliminate delays that result when team members are out sick or on vacation.
- Helps to establish common coding standards.
- Fosters a common “look and feel” to the code base.
- Ensures that everyone is generating detailed documentation.

Continuous attention to technical excellence and good design enhances agility. All software developers have egos, and most are naturally curious people who enjoy learning new things. In anticipation of code reviews, developers tend to write better code, because they know that their reputation is on the line. No one wants to be thought of as the weak link in the chain.

**An Agile Corollary:** A developer who reviews another’s code can learn new tricks and techniques; a key step in mastering any craft is to benefit from the experience of others.

**The Agile Case for Code Review: Find Bugs Early and Often**

The Agile methodology is grounded in a desire to deliver working software in the most efficient manner possible. As Agilists work to achieve this goal, they generally accept that recognizing and eradicating bugs closer to the point of inception promotes software quality. But have these developers considered the economic impact of doing so?
A SmartBear customer set out to test exactly how much money they would have saved if the company had peer review in a certain three-month, 10,000-line project with 10 developers. The business tracked how many bugs were found by QA and customers in the subsequent six months. Then they went back and had another group of developer’s peer-review the code in question. Using metrics from previous releases of the development project, the company knew the average cost of fixing a defect at each phase of development, so they could measure directly how much money they would have saved if code review was used from the start.

The result: Code review would have saved more than half the cost of fixing bugs. Plus they would have found 162 additional bugs.

Why is the effect of code review so dramatic? A lack of collaboration in the development phase may be the culprit. With requirements and design, you always have meetings and invite input from customers, managers, developers, and QA to synthesize a result. You do this because mistakes in requirements or architecture are expensive and can lead to lost sales. You then debate the relative priorities, difficulty, and long-term merits of your choices and techniques.

Saving more than $200k: A real-world case study

![Diagram showing the comparison of bugs remaining after development, after QA/Tests, and after customer with and without code review.](image)

Actually writing the source code, however, is a solitary activity. Since developers tend to create code in quiet places, collaboration is limited to occasional whiteboard drawings and a few shared interfaces. No one catches the obvious bugs; no one is making sure the documentation matches the code. Peer code review puts the collaborative element back into this phase of the software development process.

Consider: No book is commercially published without review by professional editors. Scan the acknowledgments in any book, and you find reviewers who helped “remove defects.” The correction and review process is a requirement to produce a high-quality work. (And even then, what author hasn’t found five more errors after perusing the first edition?)

Do we think it’s any different in software development? How can we expect developers to write pages of detailed code (and prose) without mistakes? We shouldn’t.

If “review” improves the quality of novels and software design, it also improves the quality of code. Peer code review adds a much-needed collaborative element to the development phase of the software development process.

The Economics of Peer Code Review

While we can make claims about the value of code reviews, we can illuminate these points with specific examples and quantitative evidence. Note that all of the companies profiled on the following pages employed CodeCollaborator as the solution of choice.
Case Study Overview - Soft Corp

Company Profile: Develops technology-intensive systems for sensitive government projects and ruggedized, mission-critical mobile applications.

Pain: Lack of standardized code review process. No obvious way to enforce consistent code reviews across hundreds of programmers and multiple time zones. Existing code review tools delivered various levels of quality and performance (Codestriker, custom-built tools and spreadsheets). Many teams conducted laborious and intensive code reviews that lasted for 1.5 to 2 hours.

Goals:
- Conduct reviews for 100% of the code for all projects.
- Make reviews more efficient in less time.

Benefits:
- Ease of use and focus on efficiency promoted swift product adoption by the development team.
- Because developers actually use it, they are finding twice as many bugs in a quarter of the time.
- Integration with ClearCase (source code management) and ClearQuest (problem reporting and tracking) was easy.
- Setting up reviews was fast and easy.

Results: Programmers can conduct reviews at their own pace, on their own schedule, sitting at their own desks. Reviews now take about 20 minutes – a savings of about 75% of code review time.

Environment:
- Platform: Microsoft Windows, Linux, UNIX
- Source Control Management: Rational ClearCase
- Development: Various
Case Study Overview - Cisco

Company Profile: Code review is essential to their legendary quality control. Cisco participated in the largest case study ever done on lightweight code review process. To view the actual metrics generated during this assessment, please download a free copy of *Best Kept Secrets of Peer Code Review*.

Pain: Every code review required significant preparation.

- Author had to upload files from Perforce® and select 1-4 reviewers
- Files were reviewed using before/after side-by-side view
- Comments and defects threaded by file, line number
- Review rework required that authors fix defects and upload fixes for verification
- Fixes verified before review is complete

Goals:

- All code must be reviewed before checked into version control
- No in-person meetings
- Process is enforced by tools
- Metrics collected automatically
- Review-level and summary-level reporting

Benefits:

- By implementing lightweight code reviews, Cisco saved, on average, 6.5 staff hours per review.
- Their results drove definitive best practices for code reviews. (Read more on page 9.)

Results: Cisco proved that lightweight reviews were just as effective as formal inspections at uncovering defects.

Environment:
Platform: Various  |  Source Control Management: Perforce®  |  Development: Various

### Case Study Parameters

| Team: Cisco® Meeting Place® product group |
| Scope: 2,500 reviews  
50 developers  
Multiple geographies including San Diego, Bangalore, Budapest |
| Lines of Code: 3,200,000 |
| Duration: 10 months |
**Case Study Overview - Developer of Crisis Communication & Response Technology**

**Company Profile:** Leading provider of crisis communication and response technologies. Software quality is critical; code reviews are a company policy. Must identify bugs while software is still in development.

**Pain:** Team regularly reviews 500 lines of code projected on a wall and discuss “diffs” (distributed prior to meeting.) During the meeting, participants walk through code to identify and resolve defects.

**Goals:**
- Increase team productivity.
- Allow online collaboration.
- Provide a single repository for reviews that can track issues in their defect tracking tool.

**Benefits:**
- Ease of use and collaboration features enable online code reviews by virtual teams.
- Reviewers can correlate inline comments and live chats to individual lines of code.
- ROI calculated in months based on time savings.
- **Key win:** “The time we saved by eliminating the manual process is time we now spend improving the quality of the code.”

**Results:** Eliminated manual activities and inefficient formal meetings so developers can focus on the code reviews themselves rather than on review trappings. CodeCollaborator flexibility enables teams to enforce code reviews before or after source control commits based on Agile process requirements.

**Environment:**
- Agile Platform: Windows, Linux
- Source Control Management: Perforce
- Development: .NET, C#, C/C++, Embedded C/C++
Types of Lightweight Code Review

Lightweight code review provides the right mix of code review process with Agile practice, allowing effective and efficient code reviews without the overwhelming time investments and the burdens that result. There are four different approaches to lightweight code review. As with so much in the Agile world, each team must decide for itself which approach works best for them.

Over-the-shoulder. This is the easiest technique of all: when it is time for a code review, find a developer and sit down with him or her in front of the code. Face-to-face communication is an easy, high-bandwidth medium for the author’s explanation of the code.

E-mail pass-around. Once the code is ready, it’s distributed by e-mail. Although emails offer a more passive approach to code review, content can become nested in multiple replies and difficult to manage and search. Further, it can be hard to determine when the review is actually finished.

Pair programming. A key contribution from the Extreme Programming world is pair programming, which in some ways is a continuous code review. A great way to share expertise and promote the mentoring of more junior developers, pair programming review comes with both high overhead and the risk that the reviewer is too “close” to the code, making the review less effective.

Tool-assisted review. Code review tools help overcome the shortcomings associated with each approach listed above. Tools automate the mundane tasks of code review and can package source files, send reviewer notifications, facilitate overall communication, and ensure that defects are tracked and resolved.

Best Practice Techniques for Code Reviews

Regardless of which type of lightweight code review your team chooses, developers can apply proven techniques that prevent wasted time and improve results.

1. Limit the time dedicated for code review
   Spend no more than 60 to 90 minutes performing code reviews.

2. Go slowly
   To ensure an effective review, do not review more than 200 to 500 lines of code per hour.

3. Limit the amount of code reviewed
   Try to limit the total amount of code for a review to 200 to 400 lines.

4. Annotate the materials before the review starts
   By looking at code using tools other than their standard editors, developers can often spot problems in their own code before the review begins.

5. Keep Review checklists short
   If your team relies on checklists, make sure that they don’t include items that are obvious or can be detected via automation. Instead make sure that they focus on things that are easy to forget (e.g. “Are all errors handled correctly everywhere?”).

Best practices are valuable in that they help users deliver repeatable process that they can tie to a ballpark prediction of results. The next point of consideration is the timing of code reviews in the Agile environment.
Timing Agile Code Reviews

The beauty of code reviews is that they not only can be timed to add the most value to your existing process, but they also scale easily in Agile environments to accommodate changing requirements and software quality benchmarks.

Effective code reviews can be invaluable to the Scrum methodology. In some Scrum teams, developers invest a target percentage of their overall capacity performing code reviews. In such environments, team members are encouraged to burn out their allotted hours for code reviews before the end of each sprint. They also apply best practices around defining tasks that are granular enough so that the code requiring review is in manageable chunks and does not overwhelm the review process.

Many team leaders attest to improvements to the overall Agile development experience as well as the productivity of Scrum teams when they designate code review as a requirement before a story can be marked complete. Such development organizations have succeeded in incorporating code review into their Agile processes and thus, reap the resulting benefit: higher quality software.

In reality, although higher quality software is certainly a tenable goal, actual metrics are needed to prove the impact of code reviews on software quality. At first blush, allocating time to gather metrics may seem anti-Agile. The question is, how can this endeavor be "agilized"?

Useful Metrics to Assess Agile Reviews

Say the word "metrics" and many Agile practitioners balk. In no way are we recommending that your Agile team implement a cumbersome metrics gathering process. What we do suggest is you choose metrics that are pretty easy to collect to gain useful insight into the value of code reviews.

Keep in mind two things:

- These metrics values are for illustrative purposes only. They vary based on the type of code you are developing. By obtaining and saving baseline data, you can see how code review adds value to your development efforts over time.
- Code review tools can save you substantial time and automate the collection of these metrics, analyze ratios, and generate related defect logs.

Figure 1: Timing code reviews in Agile environments. Development teams can implement code reviews before or after code check-in depending on their process constraints.
To make decisions and comparative analyses, analytical metrics can provide useful data across many reviews in a group by author, by group, by reviewer, or by set of files.

<table>
<thead>
<tr>
<th>Raw Metric</th>
<th>Definition</th>
<th>Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>Lines of Code</td>
<td>Indicative of review “size.” Can be measured in lines of code, pages in document, or lines on pages.</td>
</tr>
<tr>
<td>Inspection Time</td>
<td>Amount of time reviewers spend looking for defects</td>
<td>Indicative of code review time investment</td>
</tr>
<tr>
<td>Defect Count</td>
<td>Anything that a reviewer wants changed</td>
<td>Indicative of the review’s impact.</td>
</tr>
</tbody>
</table>

Analytical Metric | Definition | Common Values                                                                 |
|------------------|------------|-------------------------------------------------------------------------------|
| Inspection Rate  | The speed at which a certain amount of code is reviewed | Meticulous 100-200 LOC
Typical 200-500 LOC
Perfunctory 800-1000 LOC |
| Defect Rate      | The speed at which reviewers uncover defects. | Typical 5-20 defects* per hour depending on review technique |
| Defect Density** | The number of defects found in a certain amount of code | The higher the defect density, the more defects the team is uncovering, usually indicative of an effective code review |

**Defects** refers to anything that the developer wants changed in the code, e.g. a bug in the algorithm, re-formatting of a comment, correcting a typo, etc.

**This is where contention can develop. A higher defect density means the code reviewers did a great job! It does not mean that the underlying source code is bad or that the developer coded improperly.

**Overcoming Resistance**

By now, we hope you understand how code review can be adapted to Agile environments and the sustainable business value it brings. Promoting adoption within the development organization is the next step.

Overcoming developer resistance to code review adoption can present real issues. Consider a key Agile Principle: The best architectures, requirements, and design emerge from self-organizing teams. It is the coming together of team members that breeds success; those who don’t believe in the process or the tool do not participate in the journey.

It should be no surprise, therefore, that when peer code review is mandated by someone outside the team, its chance of success is compromised. Further, when team members do not want code review to succeed, it will fail. The biggest reason for this is the contentious environments that ensue where code review results are used to measure the caliber of the developer rather than the quality of the code. Attempting to make developers appear at fault is an inappropriate application of code review results and reflects poorly on management that chooses to wield it in this manner.

Another stimulus of contentious review is when developers feel interrogated by those trying to understand why certain assumptions or decisions were made. For example, when a reviewer might ask, “Why did you use that API?” A less threatening way to elicit the same information might be to ask, “What was the idea behind choosing this API?” You will find tone to be a profound influence as developers adopt and accept the process of code reviews.
Even when team members support code review, they may still resist adoption. In these situations the key to success is to start slowly. This means not pursuing endeavors such as: “Starting today, 100% of all code written must be peer reviewed.” Instead, to get the best return from code reviews based on time invested, try restricting code reviews to only a limited portion of the source code.

Some examples:

- Have developers select the “Top 10 Scariest Source Files.” Only review changes to those files.
- Review changes made only to the stable branch of the source code.
- Target and review unit tests. Specify that only when unit tests are complete and passing, that the results validate that the underlying implementation is correct.

Broaden the amount of code that gets reviewed after a team has experience with code review and recognizes its value over time.

The beauty of lightweight code review is that everyone can win. Organizations that follow these guidelines will:

- Improve software quality by managing bug remediation closer to the point of inception
- Sustain business value by better aligning budgets with development effort and cycles
- Find that developers who experience the benefits of code review and are not threatened by its use will embrace it.

**Summary**

Peer code review is in fact an Agile Process. The legacy of heavyweight code inspection processes has biased some Agile developers away from code review, but several types of lightweight code review work well in an Agile environment. By using the right approach and techniques, and by phasing in the use of code review over time, Agile teams can more easily deliver faster sprints and working, high-quality software.

Peer code review also offers soft benefits, many of which evolve from the focus on Agile collaboration and interaction, such as:

- Applying two sets of eyes to search for bugs
- Maintaining a high benchmark for code quality and readability
- Establishing trust among colleagues Mentoring junior developers
- Sharing and disseminating knowledge and elegant coding techniques
About CodeCollaborator

CodeCollaborator, purpose-built for agile and lightweight source code reviews, simplifies peer review by supporting workflows; integrating with incumbent development tools; and automating audit trails, metrics, and reporting. When you use CodeCollaborator, say goodbye to the tedious and time-wasting aspects of code reviews: meetings, printouts, manual metrics collection, finding specific line numbers, correlating comments to code, and losing track of bugs found vs. fixed.


◆ Read about results from the largest-ever case study of peer review at Cisco Systems. Download our free ebook, Best Kept Secrets of Peer Code Review, written by the founders of SmartBear.

◆ 11 Best Practices for Peer Code Review

◆ Be Smart and join our growing community of over 100,000 development, QA and IT professionals in 90 countries at (www.smartbear.com/community/resources/)

Try CodeCollaborator Free for 30 Days

Scan and download your free trial to see why users choose SmartBear for peer code review.

About SmartBear Software

SmartBear Software provides enterprise-class yet affordable tools for development teams that care about software quality and performance. Our collaboration, performance profiling, and testing tools help more than 100,000 developers and testers build some of the best software applications and websites in the world. Our users can be found in small businesses, Fortune 100 companies, and government agencies. For more information about SmartBear Software, our award-winning tools, or to join our active user community, visit www.smartbear.com.