SMARTBEAR
Collaborator

Whitepaper

Improve Quality and Morale: Tips for Managing the Social Effects of Code Review

SMARTBEAR

How do you get your team on board so they can reap the benefits of code review?

And, deal with the social issues that come with criticizing another person's work? This White Paper provides tips so that managers and developers can run successful peer code reviews.

Introduction: Wailing and Lamentations

Have you seen this before? Your development team has just been told they need to do code review. Maybe the mandate is coming from upper management or from an industry regulation. Or, maybe even from a visionary (now troublemaker) on the team itself. And, although everyone agrees that releasing "better code" is a good thing, the new initiative to review each other's code incites a cacophony of whining, wailing, and lamentation.

Let's say you're the group's manager. Even though you explain to your team that code review has been consistently proven to improve code quality – effectively and efficiently – they're still skeptical and reluctant. This reaction is entirely normal. Developers resist code review because they associate it with paperwork, meetings, overhead, inefficiency, and criticism. They often view it as an impediment to productivity rather than as a vehicle to better understand and improve the code and its design. So you have a challenge: how do you get your team on board so they can reap the benefits of code review? And, how do you deal with the social issues that come with criticizing another person's work?

Getting Started Fast: Focus Review Efforts at the Beginning

If your team is not already doing code reviews, it's a good idea to ease into them by reviewing only a little bit of the code at the beginning. Introducing full code review all at once can be overwhelming to a change-resistant team.

Focus your efforts in the following areas to get maximum results from code review in the least time and to quickly demonstrate the benefits:

- **Review changes to the stable branch only.**
- **Review changes to the core module that all other code depends on.**
- **Review changes to the "top 10 scariest files" as voted by the developers.**
- **Review only unit tests. If they are complete and not over-specified, any bugs or other problems can be fixed safely later.**
- **Find problems as you go by reviewing only changes, not entire files.** You'll still discover problems around the code being changed, even if they weren't relevant to that check-in, and you can fix them then.
- **Review code whenever developers think it's necessary,** such as when they have concerns about a section of code, when a domain expert can lend specific expertise, or when they're working on code they didn't write and may not fully understand.
- **If you have a large code base, pick out files you know are trouble and review them first.**

Handling Objections: Legitimate Fears and Groundless Concerns

Some of your team's objections to code review are probably based on myth and assumptions that are just plain wrong. Other resistance stems from legitimate fears...but these fears can be addressed and potential problems solved before they become issues.

To get your team to participate willingly and enthusiastically in code review, address their fears directly:

1. ***It's too much hassle. We hate meetings and paperwork –*** for many years, code review involved an extensive framework of cumbersome printouts and multiple meetings to discuss points found during review. Often the whole team attended these meetings, which seemed to never end. Today, code review can be conducted with software tools that make it pain-free and even fun. With a tool, developers do code review online at a time that's convenient for them. They review the code, make comments directly on the relevant code snippets, and carry on chat-style conversations with colleagues in the next cube or on another continent. The code review tool automatically gathers metrics that the team can use to produce any necessary reports. No meetings, no paperwork, no overhead, no pain!

2. ***Code review will ruin our team culture. Some people will be jerks about code review and use the opportunity to terrorize others.*** The goal of code review is to make the software as bug-free as possible, and to teach and learn in the process. When the right attitudes are set up front, most teams unite around the concept of becoming a better team, learning from others to become better developers, and producing better products! In our experience, code review almost always brings teams together rather than divides them. It gives developers a framework for communication, which is a very good thing. Once they start communicating online, it's amazing how much better they work together in person.

Of course, occasionally team members use code review as an opportunity to try to establish superiority over others. "I found bugs in your code! That must mean I'm smarter than you! In fact, clearly you're an idiot." This attitude is obviously not productive. By fostering a culture of respect and open communication, you can set the tone for more positive attitudes. Everyone makes mistakes, and these mistakes should be viewed as opportunities to learn and mentor.

Another common problem team member is the "control freak," who uses his or her seniority or greater experience to terrorize less knowledgeable team members. A good technique for getting this personality type to play nice is to explain to the tyrant that he is a teacher, a guru, a guide, a mentor to everyone else. Teachers don't berate their pupils; they take pride when pupils learn at their knee. This approach lets your control freak retain control and status, but in a helpful way. If you set the stage right, code review doesn't ruin your team culture – it improves it!

3. ***I don't like to be criticized.*** Writing code is an art, and developers put their souls into creating beautiful programs. Like anyone else, they can be sensitive when problems are pointed out. A similar situation exists for writers: you wouldn't publish a book without at least one person editing your work, because mistakes happen. Even the best authors have editors. You can't see your own mistakes – not when writing prose, not when writing code.

With an extra pair of eyes, the output is better in the end. In fact, many developers (and writers) actively seek others' feedback when working on something particularly difficult or new to make sure they get it right. The trick is to make sure code review suggestions are given in a positive way.

4. *We don't want to change our process.* Many teams are resistant to change. To these folks we say, try just a little change for just a little while! Try code review for a week, capture metrics on how many bugs are found vs. the amount of time spent, and let them see the positive results. Then ask how they feel about continuing with it. A trial period is the perfect non-threatening way to let your team see immediate benefits without feeling like more unpleasant processes are being thrust upon them.

> *"The benefits of inspections are so profound that even the smallest outfits must take advantage of this technique."*
>
> *– Jack Ganssle, Consultant/Columnist Ganssle Group*

5. *Code review takes time we don't have, and we'll miss our deadline!* This one only takes a moment to think through. If you have time to go back and do it again later! The time (and reputation) cost to fix a bug once the software has gone to customers – or even QA – is substantially larger than when you catch it in review. If you're really pressed for time, at least selectively do code review: only review the most complex sections, those with the most changes, and those the developers feel are highest risk.

6. *Big Brother is now watching and grading me on my defects! –* Tool-assisted code review is great about capturing metrics about defects. These metrics are crucial to measure and improve the process, but they can be used for good or ill. If your team thinks they're being evaluated based on the code review metrics, they'll likely focus on tasks that improve the metrics rather than those that improve the code. And they won't feel comfortable with the review process.

This fear is not difficult to combat. Simply make it clear to the team that they will never be graded on how many defects their code has (and carefully follow this policy). "Number of defects introduced into your code" is not an accurate way to evaluate someone anyway: the more senior developers will likely be working on new code, more complex code, and code with many changes...all of which are likely to have more bugs. In addition, this code is likely to (and should) be more carefully reviewed... and the more a piece of code is reviewed, the more bugs are likely to be found. So make sure your team understands they won't hear about their bug rate during reviews, because the goal is to review the code – not the coder!

Another great way to combat "Big Brother" concerns is to reward defects as a successful team result of both author and reviewer. You can even create a "leader-board" to track who finds the most defects. Thus a potential penalty is converted into a reward.

One of the best things about code review is that it substantially improves team communication. Before you start reviews, put an environment in place that fosters and encourages communication and respect. After you debunk the myths and set the right attitude for code review, you'll likely be surprised at how successful it is for your team.

Why Review Code?

If you're reading this paper, you probably already know that code review finds bugs when it's cheapest to fix them – before the software goes to QA or to customers. But it does many other great things too:

- **Finds and fixes maintain ability issues** such as documentation, organization, architecture, usability, efficiency, robustness, maintainability, and portability. All of these things affect overall code quality.
- **Trains developers to spot errors they might miss, and makes them much more cautious about their check-ins.**
- **Provides real-time feedback on code while it's still fresh in people's minds,** rather than six months later when a customer finds a bug.
- **Teaches both reviewers and author's new tricks.** Reviewers can learn when they see new techniques and good habits in code, and authors learn from the feedback they receive.
- **Makes engineers more familiar with parts of the code base they might never see otherwise,** breaking down the "my code/your code" silos and giving your engineering team much broader exposure to the entire codebase.
- **Provides a vehicle to educate junior team members, mentor others on the team, and make the whole team more competent on all of the code, rather than having just one or two experts on each section.** By educating everyone, you raise your "bus factor" (the number of people who would have to get hit by a bus – or otherwise become unavailable – before your knowledge base about your code is crippled).
- **Saves developers humiliation later when bugs go out to customers** (not to mention preserving company reputation and saving the cost of fixing the problem)!
- **Unites teams over the concept of a better overall product, and gives them reasons to talk to each other and improve working relationships.** Team members enjoy learning and teaching, and everyone gets the feeling that the whole development organization is accelerating.

Tips for Managers

If you're a manager, your most difficult task is probably dealing with your team's emotions and human interactions. To ensure positive interactions, it's your job to set the tone for code review. The following guidelines can help you get code review started on the right foot to ensure team acceptance and ultimate success.

1. ***Make sure everyone (including you) understands that code review is all about the code, not the person.*** The point is to eliminate as many defects as possible, regardless of who introduced them, and to learn to be better programmers in the process. Make it clear this is about the whole team producing a better product and learning and becoming better developers. It's not about who's the smartest, or who finds (or introduces) the most defects. Code review isn't personal.

2. ***Encourage team members to review different people's code so everyone can get to know others and their styles.*** By exposing everyone on the team to everyone else, this technique encourages maximal sharing of knowledge and learning.

3. ***Big Brother is not watching you.*** Never use metrics from code review as part of your team's performance evaluations. Make it clear to the team that review statistics such as number of code defects will never be used in

reviews. You want them to be comfortable with the process, not resentful or suspicious of it. Besides being counterproductive, evaluating people on the number of bugs they introduce or find is inaccurate. "Hard" code inherently has more defects. In addition, studies show that the more time a developer spends reviewing code, the more bugs they will find. [For details on these studies, get our free book, Best Kept Secrets of Peer Code Review.

4. ***Explain to the team that you want them to find defects.*** The more the better. Each defect they find is another one that doesn't clog up the QA pipeline, or worse, get into the customer's hands. Another bug that doesn't cause a maintenance nightmare or a complete code rewrite later, months after layers have been added on top of it.

5. ***If someone is not approaching code review with the right attitude, don't single him or her out.*** Calling the person out in front of the team will likely cause more trouble than it fixes. Instead, address the team as a group and remind them that finding defects is a good thing and that defect density does not correlate to developer abilities.

6. ***Leverage the Ego Effect by always reviewing at least some code.*** When developers know that their peers will be reviewing their code, they instantly become better developers due to a simple phenomenon we call the Ego Effect. They don't want their colleagues catching their silly or repetitive mistakes, so they give their code a quick review themselves before checking it in and they pay a little closer attention as they work – so you get better code before reviews even happen. The Ego Effect works even if your team doesn't have time to review all code. As long as your team is doing spot checks and reviewing some code (maybe 25%), the Ego Effect will make your developers more careful.

Try code review for one week, measure actual results, and evaluate the benefit for your team:

Not sure if code review is worth your team's time or if they'll accept it? Try it for a week, measure the results, and make an informed decision.

- **Download a free evaluation copy of CodeCollaborator, the premiere code review tool from SmartBear Software.**
- **Have your team spend 25 minutes per day reviewing code for one week**
- **Encourage them to have fun with the process, help each other, and embrace the opportunity to learn and teach**
- **Directly measure the value of code review by examining bugs found vs. time spent.** Using a report produced by CodeCollaborator, count the number and type of defects you find, and note the amount of time your team spent doing reviews (CodeCollaborator tracks all these details automatically). Divide the amount of time spent by the number of defects found to find the amount of time spent per bug.

> *"When we introduced [SmartBear Software's code review tool] CodeCollaborator, it was like someone broke the ice in our group... As a result, now we collaborate more often to design and test features as well as review them."*
>
> *– A. Kalvanavarathan, Manager*

Most companies find and fix one bug for every 10-15 minutes spent doing reviews. How much would each bug cost to fix if it went to customers?

Tips for Developers

If you're a developer, you almost certainly take pride in your code and in your knowledge. Someone's criticism – even constructive criticism – can be hard to take. While you know it provides an opportunity for growth, criticism can still hurt your feelings or even embarrass you in front of the team. The following tips set the stage for you as both reviewer and author. They will help you accept other people's feedback as well as deliver suggestions in a positive fashion that encourages learning.

1. ***Remember that you're critiquing code, not the coder.*** Make sure your tone is not personal and that it's clear your intent is to improve the code, not belittle, lambaste or take shots at its author. Taking a teaching or mentoring attitude helps immensely.

2. ***Offer generous amounts of praise.*** Even if you find a lot of problems with someone's code, surely you can also find some good things to comment on. Positive comments remind the coder that even though they may have much to learn, they still add value. And inserting some complimentary comments takes the sting out of the corrective ones. One good way to set the tone of a review is to put a summary comment at the top of the code that includes positive feedback.

3. ***Be respectful to – and patient with – everyone, especially team members who aren't as knowledgeable or experienced as you.*** Use the opportunity to teach them. This approach makes the whole team better, they won't forget the kindness, and they'll return your respect in spades.

4. ***Remember that everyone – even you – makes mistakes.*** You're human like the rest of us, and we all mess up sometimes. By all means, try to minimize your errors. But go easy on yourself and others when it happens, and focus on the opportunity to improve rather than berate.

5. ***Take advantage of the opportunity to learn from others.*** No matter how much you know, you can always learn new tricks and techniques from your teammates. Embrace the chance to expand your knowledge.

6. ***Review your own code and create a checklist of the problems you frequently make.*** Before you send your code for review, check your own code for the errors on the list. This way, your colleagues don't find as many errors and they don't see you making the same mistake over and over. After a while, you'll stop making those mistakes (a wonderful thing!), so update the list occasionally.

> *"Our team balked at doing code reviews at first. Now we can't imagine working without Code Collaborator."*
>
> *– Brian Toombs, Lead Developer, Cisco Systems*

7. ***To lessen a criticism's sting, ask a question instead of making a statement.*** Asking for the author to explain their reasoning behind something acknowledges that you respect them. "What was your thinking in using this function call instead of that one?" is much less confrontational than "This is the wrong function call. Use the other one." Asking a question also gives you the opportunity to learn – perhaps their thought process included a technique you didn't know or a reason hadn't considered!

8. ***Avoid asking accusatory "Why" questions.*** Instead of saying, "Why didn't you…" ask something like, "What did you have in mind when you…?" The tone of the ensuing conversation is now entirely different. Not only does it avoid inherent accusations, this approach opens the door for dialog andlearning.

9. ***When disputes arise, win and lose gracefully.*** Don't rub in your victories or sulk and pout if you were wrong… this behavior sets the stage for more confrontational interactions later. And remember it's about the code, not the person.

Tool-assisted code review makes it much easier to deliver review feedback more gently than an "over the shoulder" review or one in a meeting. You can review your comments before you send them to ensure that they reflect a positive tone, show respect and do not imply personal criticism.

Conclusion: Code Review Improves Quality and Morale

There's no question that code review improves overall code quality (including design, structure, comments, maintainability, documentation, and unit tests) and reduces the number of defects that go to QA and to customers.

And when executed with the right attitude, code review also has extremely positive effects on teams and their personal interactions. We've worked with hundreds of teams that have implemented code review, and they report a variety of unexpected benefits:

- **Code review creates an environment where developers work together instead of in parallel.** By creating a venue for easy communication, it encourages conversation so developers don't work in their own little silos… even if they work on different continents.
- **Developers unite over the prospect of a better team and better code.** The team picks up momentum as the product quality continues to improve. And, everyone's pleased that their development skills are improving too.
- **The team has fun and interacts more in person.** Tool-assisted code review leaves plenty of room for humor and play. Now, team members who didn't know each other well can establish common ground and start to get to know each other online. Once the team starts communicating and joking online, they tend to work better together in person too.

About SmartBear

At SmartBear, we focus on your one priority that never changes: quality. We know delivering quality software over and over is complicated. So our tools are built to streamline your process while seamlessly working with all the tools you use – and will use. Our tools are easy to try, easy to buy, and easy to integrate. We're used by over 16 million developers, testers, and operations engineers at over 24,000 organizations. Wherever you're going, we'll help you get there.