inSite

Ask It!

Agile techniques for code review

Much of the literature on the net about performing code reivews in agile gravitate to pair programming. But lets assume for the sake of argument that pair programming is not acceptable (management, SQA, etc are not sold on it). A typical formal inspection is often an anti-lean approach to code review. Does anyone have techniques they have used successfully for code review that don't impeade the flow of the development proess other than pair programming?

Tags: scrum code review peer review
Shared with Group: Agile Software Development
Submitted 2 days ago by Cass Dalton

Suggested Answers

2 days ago at 6:41AM **Bryan Wells** wrote:

Cass:

We recently updated the peer reviews content on the BASP. I would start with this page and see if it helps to answer your questions.

https://basp.web.boeing.com/basp/index.php?title=Peer_Review

Be sure to follow the links embedded in the text for details on the peer-review topics.

Bryan

1 day ago at 8:14AM **Cass Dalton** wrote:

The differences between "code review" and "formal code review" in the flow charts on the BASP peer review page are minimal. Does the formality come from the the use of PIMS/Sherlock (I am unfamiliar with them)?
I think my real question revolves around the block in the flow chart that says "Checks out old code and new code from CM system - Conducts review". I'm interested in techniques for this block that generate acceptable artifacts for CMMI, but don't impede work flow. Depending on the size of the code base, "checks out old code and new code" could be a waste-laden task.

2 days ago at 7:19AM **Chad Beaudin** wrote:

Crucible is a fantastic product for performing code reviews.

1 day ago at 11:49AM **Robert Gajda** wrote:

Crucible is indeed excellent. Prior to using it I found code reviews rather tedious (getting a group together and having a presenter walk through the code) or not really traceable (review comments via email). Crucible solves that and easily fits into any formal review process.

Bonus, since it's an Atlassian product it flawlessly integrates with JIRA and actually comes bundled with Fisheye, a repository inspector/monitor/whatever. However you do need a separate Fisheye license to activate that part of the product.

2 days ago at 8:32AM **Drew Justice** wrote:

We have implemented role-based peer reviews. We have a primary reviewer that is focused on the acutal code change, a secondary reviewer that is focused on the testing that was done to validate the changed code, and an admin reviewer that focuses on the header entry and coding standards. While originally not inteded to be "agile" it actually fits since you are directing the reviewer's focus to one area of the review and we have found it gives us better coverage overall.

1 day ago at 12:01PM **Douglas Bollenbach** wrote:

This entry is strictly opinion. Our shop uses the agile development model. Our environment is C# .NET in Visual Studio 2010 using Resharper along with other various code enhancing plugins. I have reviewed all three processes in BASP for peer reviewing. In my honest opinion, peer reviewing code should be targeted at older development environments, because that is where the real value is. As new tools and technologies become available, "how" we develop software is drastically changing. Quality is becoming more built-in to our processes (lean) rather than requiring a separate effort. Here are some reasons behind my opinion:

1. Resharper is a very good tool that enhances Visual Studio environment code error checking and naming persistence throughout projects along with numerous other things like naming schemes and refactoring automation and suggestions(quality built-in).
2. In Visual Studio using compiled languages, the compiler will not start, build, or complete if there are syntax errors (quality built-in).
3. Additional automated unit and behavior testing should be comprehensive enough to cover any code logic errors as much as possible.
4. Regression testing and demonstrations further cover error handling and overall "does it work"?

In the true context of code "errors", what additional errors could be identified with a peer review embedded in the above process with the accompanying tools? Is there value added in peer reviewing code in this type of environment? Seems kind of wasteful. The only benefit that I could come up with would be any in depth refactoring opportunities that resharper might miss, better design ideas for scalability, or tweaking performance, etc. Notice though, this type of value does not target code "errors" or "defects", and code "improvements" in these areas typically will not affect software workability most of the time, unless performance was the goal for the change.

I realize this probably does not answer your question at all, but it was meant to address and spur people thinking about the probable lack of value-added in performing peer reviews in the above environment scenario.

1 day ago at 8:02AM **Cass Dalton** wrote:

I would agree with you and go even further that even without the environments you mention, code inspection is typically a poor tool for defect reduction. That's what tests are for.
However, I would disagree that those arguments mean that there is no value in code reviews because I don't believe that the main value of code reviews is defect reduction. The primary value from code review comes from

- Learning, interaction, and communication that comes ffrom the discussion, esp between junior and senior engineers
- Reducing maintenance issues like poor interface design or code duplication
- Confirmation that the tests are actually verifying the correctness of the code with respect to the intent of the user stories

You won't get any of that from a development environment. And this is really what's feeding my question. Since we are reducing the reliance on review for defect reduction, then I believe that an overly formal review process contains waste. Therefore, I'm interested in code review techniques that have a minimal impact on the creative flow of the developer and the team.

1 day ago at 9:23AM **Christopher Ellison** wrote:

I disagree - in my experience, if you actually *do* code review in an Agile environment (and here

I include pair programming as a form of code review), I've found and removed more defects than with testing. There are some industry numbers (Casper Jones' defect study) that seem to back that up more generally (though I don't know how many of the Casper Jones projects were using Agile or how unbiased the Casper Jones study really is). The downside, though, is that code review can't be re-run cheaply.

Don't get me wrong, I'm a very big fan of unit testing, TDD, and BDD. I'm also very much *not* a big fan of huge, formal code reviews. I completely agree that those types of "everyone watch this PowerPoint of my code snippets" code reviews are terribly wasteful. A low-overhead review process, though, can be very efficient in removing defects and increasing code maintainability. For example, a small-team process I've used in the past just used a two-stage (Git) repository and a process agreement that you only check in code to the level 1 repository. Code that passed the tests would then have to be reviewed by another team member, who would then commit it to the level 2 repository. We didn't review all of the code, but we followed the process well enough to find enough defects to justify the overhead.

Casper Jones writeup: http://www.sqgne.org/presentations/2010-11/Jones-Nov-2010.pdf
(Slide 23 for defect removal efficiencies)

1 day ago at 10:23AM **Cass Dalton** wrote:

Regarding the Jones presentation, I don't know that the report counters the assertion that good and comprehensive testing can do a better job at finding defects that code inspection. I think i can infer that they measured the results of automated testing and quality testing independently of formal reviews, and shops that are doing formal reviews are probably not giving automated unit and integration testing as much attention as a highly agile shop. Sure, they will find defects, but to really find code bugs via inspection, you effectively have to turn your brain into an emulator, and that will never be better than running it on the target machine.
I also guess that they are grouping bugs (crashes, memory leaks) with maintainability type issues into one "defect" bucket, and I would never argue that reviews are bad at finding maintainability/interface defects. I just don't define those as defects when I claim that inspection is a poor tool for defect reduction.

18 hours ago **Robert Gajda** wrote:

"In my honest opinion, peer reviewing code should be targeted at older development environments, because that is where the real value is."

The problem with this statement is due to the question you ask later:

"In the true context of code "errors", what additional errors could be identified with a peer review embedded in the above process with the accompanying tools?"

I have never done a code review to check for or verify those four things that you correctly indicate our development tools do for us just fine.

Code reviews, as Bonnie below states, are for checking readability, maintainability, consistency, and other actual Quality factors. You can write code that accurately performs its function but if it's unreadable and/or difficult to modify/extend then it is not Good Code. And often the only way you realize this is from someone else pointing it out to you and explaining why it should be done differently.

For everyone that can't visualize how you can do a code review that is not tedious and is intuitive I'm including links to Crucible. Please check it out. On our project it's been a huge benefit for our code quality.

http://www.atlassian.com/software/crucible/overview

I don't know how people do code reviews with this kind of tool. (Well I guess I DO know but I just cringe to think about it)

1 day ago at 12:32PM **Bonnie Triezenberg** wrote:

What has always made a peer review different than an engineering review is that reviewers are assigned to look for specific defects (example given in Drew's comments above). If you augment the human review with automated tools (example given in Doug's comments above), then the types of defects the human needs to find are reduced. Combining the two approaches so that humans have specific assignments to look for the defect types that the tools don't find will keep the number of personnel and the number of hours down.

On our program, we also changed "when" we do the code reviews - we call it a code quality review and we do it only AFTER the code has been shown to produce the correct test results and AFTER it has passed through a static analyzer. Test first is a basic tenant of agile. That way, we aren't "polishing" code that isn't correct. In the review, since we know the code passed tests, we don't look for correctness, instead we are looking for maintainability issues: Does the code structure comply with the architecture standards; are the variable names clear enough that someone will still understand them years from now; does the static analysis results raise any red flags; etc. ?

Personally, I think that not polishing until after we know the code works is one of the single biggest work takeouts we've realized in moving to agile. I can't tell you the thousands of hours I have spent over my career working in the waterfall model reviewing code that was later scrapped when it didn't work in integration test. I look back at those years and programs and just see a huge mountain of wasted effort. Yeah, we got some learning from it, but it was mostly waste.

1 day ago at 3:29PM **Michael Munsey** wrote:

Gerrit is a good web based code review tool, but only works with Git.

1 day ago at 10:37AM **Chris Mercer** wrote:

Have you used gerrit/set it up? Just wondering what the real world user expierence was like.

18 hours ago **Robert Gajda** wrote:

From the Gerrit documentation it looks like a super clunky version of Crucible.

Crucible happily works with git, plus any other standard repo (mercurial, subversion, CVS, Perforce, ClearCase).