Using checklists for code review

April 24, 2013 (https://blog.beanbaginc.com/2013/04/24/using-checklists-for-code-review/) David Trowbridge (https://blog.beanbaginc.com/author/david/) Code Review (https://blog.beanbaginc.com/category/code-review/)

We've all seen them, and most of us make them. Process lists, to-do lists, lists of all kinds– there's something very satisfying about checking a little box or crossing an item off a list. However, checklists can serve a greater purpose than as a physical reminder that we've accomplished something. In many fields, checklists exist because human brains are fallable and prone to distraction. Pilots use checklists to make sure that amid dozens of relatively simple tasks, none get forgotten. Checklists used before and during surgery reduced the death rate by nearly 50%.

I've been doing code review for a long time. In the open source world, code review happens on pretty much any third party contribution. In my time at VMware, pretty much every non-trivial change was expected to be reviewed before checking it in to source control.

In general, people are pretty good at the code review process, but it's sometimes surprising what can slip through. A natural consequence of the way our brains look at the world is that it's easy to pay a lot of attention to small details and code style flubs, and completely miss the big picture. It's for this reason that a lot of designers will start with rough sketches when they start gathering feedback–if something looks too finished, reviewers will only give them comments about the details. Unfortunately, unless you're starting out with pseudocode or architectural sketches, source code almost always looks "finished" and can therefore encourage some bad habits.

One of the ways that I avoid the pitfall of giving superficial reviews is with checklists. I have a bunch of different sets of checklists that I've developed and tweaked over the past few years, and for any given change, I'll select the sections which apply. I've applied some of these at VMware, working on a large desktop application code-base (mostly at the user interface layer), as well as with Review Board itself.

Here are some examples of some of the many checklists I use in my day to day work:

User interface

- [ ] Are screenshots provided for all UI changes?
- [ ] Does this change introduce any new concepts or models? Evaluate.
- [ ] How much expertise and context will a prospective user need?
- [ ] Do we do anything similar elsewhere? Can we build on existing knowledge or habits?
- [ ] Do wo do anything similar but just different enough to cause problems with old habits?
- [ ] Are there any new multi-step workflows? How long and complex are they?
- [ ] If users make mistakes or errors, what are the failure conditions? Can mistakes be undone?
- [ ] How is the copy? Are sentences well-formed and clear? Any spelling errors or typos?
- [ ] Is the UI layout localizable?

Visual design (for a native desktop app)

- [ ] Are controls laid out in a way which makes sense given visual scan order?
- [ ] Check for alignment
- [ ] Check for layout, spacing, and padding
- [ ] Check for visual consistency

Correctness (for C/C++ code)

- [ ] Correctness of algorithms
- [ ] Loop iteration and off-by-one
- [ ] Memory access errors
- [ ] Memory leaks
- [ ] Incorrect behavior
- [ ] Switch/break fallthrough
- [ ] Exception safety

Security (for C/C++ code)

- [ ] Array bounds on buffers
- [ ] Are files created and read/written? Check permissions, races
- [ ] All user input sanitized or validated?
- [ ] Proper, bounded use of str*, printf, f* functions?
- [ ] Validate remote host cert validity & identity
- [ ] Crypto use

Obviously, not everything is applicable for every change. If the review request isn't making any changes to UI, then I'll skip the first two checklists entirely. If a change is a bug fix, I typically won't review it for architecture and design principles.

Put the big stuff first

Once I've figured out which of my checklists apply to any given change, I sort them to focus on the biggest stuff first. If I have major architectural objections with a new piece of code, it's probably not worth reading through it looking for bugs or style guide violations. One of the worst things I've encountered (on both sides of the code review process) is working through a ton of small issues before realizing that everything has to be rewritten.

Working through the big stuff first also facilitates collaboration, because it's clear at each stage how negotiable my comments are. Architecture can be argued about for a long time, and in the end a lot of it is a personal decision on behalf of the person writing the code. On the other hand, if there's a lurking crash bug, it's not an option whether or not to fix it.

Do checklists work?

I didn't know what to expect when I first started applying checklists to my code review process. I anticipated that it would slow down my reviews quite a bit, but I didn't know what the payoff was. I was pleasantly surprised to discover that it improved the quality of my reviews immensely, and after a few months of application, the amount of time I had to spend reviewing was almost the same as it was before.

My approach is to do a pass through the code for each and every item in the checklist. By only looking for a very specific type of defect, each pass goes relatively quickly, even for large changes. Focusing on only one type of defect allows an extreme focus that makes it trivial to find issues. One good example is scanning for (most) memory leaks: simply check each allocation for its equivalent deallocation, and verify that all code paths will go through that point. Even for large changes, a pass looking for memory leaks might take 5-10 minutes. Compare that to the last time you tried to chase down a leak after the fact, and the payoff is obvious.

In contrast, I find it incredibly difficult and error prone for me to read through code once looking for memory leaks and UI design issues and API usability and possible cross-platform build issues and everything else all at the same time.

I personally feel like when I pull out my checklists, my code reviews are much higher quality. My coworkers have also commented that they find my reviews to be pretty good, so I'm hoping it's not imaginary.

Do you use checklists for code review? How are they working for you?

DAVID TROWBRIDGE (HTTPS://BLOG.BEANBAGINC.COM/AUTHOR/DAVID/)

Beanbag co-founder