ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 30 — March 25, 2013

Patrick Lam *version 1*

Reviews

A *review* is any activity where reviewers examine a work product to provide feedback. Reviews can reveal defects early, when the defects are less costly to fix. You can review anything, including: software requirements specifications, schedules, design documents, code, test plans, test cases, and defect reports.

Reviews can be either informal or formal.

- An *informal review* is a written or verbal review requested by a developer of a work product to improve that product.
- A *formal review* is a written review conducted by a team leader or a moderator for the purpose of identifying, documenting, and fixing defects in a work product.

Types of Reviews. Here are some types of reviews.

- A *desk check* is an informal review where the author distributes a work product to peers for reviews and comments.
- A *walkthrough* is an informal review meeting, moderated by the author of a work product.
- An *inspection* is a formal review meeting, guided by a moderator. The meeting produces a log of identified defects in a work product.
- A *code review* is a software inspection where defects in source code are identified, logged, and perhaps corrected.

It is useful to combine informal and formal reviews. Informal reviews help identify simple defects. Formal reviews can identify complex defects that might be missed by simple desk checks.

Desk Checks. A *desk check* is the first line of defence against defects. You can speed up formal inspections by taking care of simple defects in desk checks first. For many work products, desk checks suffice, and you might not need to go to a formal inspection.

However, desk checks are only effective if taken seriously. It's easy to just say "LGTM" (Looks Good To Me) without actually checking the product. It's important to spend enough time on desk checks, and managers must allocate time for them.

Walkthroughs. A *walkthrough* guides reviewers through the review of a work product. The author of the work product presents the design and ensures that the attendees understand its design.

Walkthroughs allow people with less expertise to review a work product, and users of the work product are often invited to walkthroughs. New points-of-view often help identify defects.

- Prior to the walkthrough, the author should distribute presentation materials to attendees.
- During the walkthrough, the author should solicit feedback from the audience.
- After the walkthrough, the author should follow up with attendees who have helped out by giving comments.

Formal Inspections. An *inspection* is a formal review meeting where participants identify and document defects or possible improvements in a work product. At an inspection, participants aim to identify and propose ideas for solving defects. Having a good mix of participants can help find previously-overlooked defects (perhaps subtle or complex ones).

Here are some steps for a formal inspection:

- *Preparation*: Before the meeting, distribute a printout of the work project to each member of the inspection team, along with a checklist indicating what to review.
- *Overview*: At the inspection meeting, the moderator provides an overview of the work product.
- *Page-by-page Review*: The moderator walks the inspection team through the work product page-by-page and logs defects.
- *Rework*: After the meeting, the author goes through the list of defects and fixes them.
- *Follow-up*: The inspection team members verify that the author has fixed the defects.
- *Approval*: The inspection team approves the work.

We do something similar for master's and PhD theses.

Code Reviews. A *code review* examines source code (or, more commonly, a patch) to identify defects or possible improvements.

Different projects have different review coverage; some projects (Mozilla) review everything, whereas companies might review only a representative sample. One can argue for sampling because developers tend to repeat the same mistakes, so that finding one issue will prompt developers to look for the same issue elsewhere.

When sampling, here are some places to look:

- source code that only one person has the expertise to maintain;
- tricky algorithms that are susceptible to defects;
- source code that calls difficult-to-use libraries;

- code written by inexperienced developers; and
- functions that could fail catastrophically if a defect is present.

In industry, you might have code review by a team; I don't know. However, in the open-source world (more below), you typically have 1 experienced developer doing a review. Generally, code reviews look for: clarity, maintainability, accuracy, reliability and robustness, security, scalability, reusability, and efficiency.

Pair Programming. We've talked about pair programming in the context of agile programming. Ideally, it can serve as instantaneous code review.

Reviews for Open-Source Software Projects

We've mentioned that open-source projects have an official repository. These projects also have a set of committers, who may commit changes to the repository. Outside contributors sometimes send patches to a project, adding new features or fixing bugs; in that case, a committer will typically review the patch before committing it. Committers may also seek review for their patches if they want someone else to take a look at it.

Case Study: Mozilla Review Policy. The Mozilla Project develops the Firefox web browser (and other projects). Due to the size of the codebase, they have an elaborate reviewing policy[1]: they require at least one review ("owner/peer review") for all patches, and a second review ("super-review") for many of the patches as well.

The owner/peer review is by a domain expert who understands the code being modified and the implications of the change. In particular, "A review is focused on a patch's design, implementation, usefulness in fixing a stated problem, and fit within its module." From the Code Review FAQ, reviews check for: whether the patch fixes a problem; the API/design; maintainability; security; integration; testing; and license compliance.

Super-reviews are by "strong hackers", who understand the way Mozilla code is supposed to look, but need not have domain expertise. In particular (per the Code Review FAQ), they are supposed to look out for: proper use of APIs; adherence to Mozilla's portability guidelines; cross-module effects; and respect of Mozilla coding practices.

Other organizations. Google and the Linux kernel both review code extensively. You can also read about Gerrit[2], a tool out of Google for code reviews.

[1] `http://www.mozilla.org/hacking/reviewers.html`, `https://developer.mozilla.org/en/Code_Review_FAQ`
[2] `http://lwn.net/Articles/359489/`