15-413: Introduction to Software Engineering

Jonathan Aldrich

Assignment 10: Code Review

Part 1 Due: Thursday, November 17, 11:59pm
Part 2 Due: Tuesday, November 22, 11:59pm
40 points

This assignment is a group assignment. Each project group should turn in one response to each part, with all the names of the group members.

Part 1: Prepare Materials (15 points)

Document your source code in preparation for a code review by another team. Choose a part of your project that is logically cohesive and about 250-1000 lines of code long.

Email to the TA a zip file with the following information:

a) Your group number.
b) The names of everyone in your group.
c) The name of your system.
d) The name of the subsystem that should be reviewed.
e) A short description of the context of the subsystem, sufficient for reviewers to understand where it fits into the larger system, and what that larger system does. Feel free to include parts of your requirements assignment or architecture assignment as an appendix.
f) An informal specification of the code to be reviewed, sufficient for reviewers to understand what the code is supposed to do. This may be a list of stories implemented by the code, but you may have to add additional details (e.g. algorithms & data structures being used) so your reviewers can understand the specification in enough detail to review the code.
g) A list of the changes you believe are most likely for this source code.
h) The source code.
i) The test suite for the code (all relevant unit tests and functional tests, either in executable format or in a description of how to drive the test manually, if the test is not executable).
j) Anything else you believe will be helpful to reviewers.

The primary criterion for the information above is what your reviewers will need to review the code—put yourself in their place (you will be in their place, for another group) and ask yourself what you would need to review your code, if you had no prior knowledge of the system.

Part 2: Formal Technical Review (25 points)

Every reviewer (i.e. every member of the team) should spend about an hour reviewing the code and test suite on their own before the review meeting is held. If the code is too long to review thoroughly in an hour (hopefully not the case because of the 1000 line limit), agree on what part to focus on.

Each reviewer should produce a list of possible issues to discuss with the larger group. Each issue should include a number, the initials of the reviewer, the location in the code (line number or function), the category, the severity, and a short (1-3 sentence) description of the issue. Try to mention a few good things (category G) about the code as well. Use the checklist on the next page to identify issues in each category. The categories are:

- D – Design issues
- E – Possible errors
- C – Coding/Style issues (other than errors)
- T – Testing issues
- G – Good point: something that is done exceptionally well in the code

The severities are:

- 3 – critical issue: if not resolved would compromise the project
- 2 – significant issue: would benefit project or avoid future problems if fixed
- 1 – minor or style issue
- Q – Unsure if this is really a problem

In the review meeting, discuss each issue that was brought up by reviewers. Combine duplicate issues, agree on the category and severity for each issue, and change the issue description if necessary to reflect the work in the group. Document any new issues that come up in the discussion of the code. Vote on an overall review result. Keep the meeting time to 1 hour; if you are unable to consider all issues in that time frame, separate the issues you were not able to discuss from the ones you were in the final report.

After the meeting, coalesce the issues raised into a single list (point 8 below).

Turn in by emailing to the TA:

1. Your group number and group names
2. The group number whose project you were reviewing
3. The name of the project being reviewed
4. The list of issues that each reviewer brought to the meeting, documented as above with number, initials, location, category, severity, and description. Please present this in tabular format (one column for each part) so it is easy to read. Sort the issues in some semantically reasonable way (e.g. by category or section of the code).
5. The result of the review: Accept, Accept with minor revision, Revision needed, Reject
6. Number of issues in each category
7. Number of issues at each severity level
8. The composite list of issues, as modified/combined/extended in the meeting, documented as above. Please present this in sorted tabular format so it is easy to read.
9. General feedback to the producers on the material they provided
    a. Was the system context description adequate for you to understand the code?
    b. Was the specification adequate for you to understand what the code is supposed to do?

Formal Technical Review Checklist

Please note: this checklist is not comprehensive. Use your judgment in finding other errors that might be relevant to the particular application/language/platform you are reviewing.

Design

Does the design effectively hide information that is likely to change in the anticipated change scenarios given?

Is the design the simplest thing that could possibly work (XP), given the need to accommodate the change scenarios above?

Does the design follow best practices such as design patterns (for OO projects)?

Errors

Are program variables initialized before they are used?

Are there possible array bounds violations?

Are there possible null dereferences?

For concurrent programs, is shared data properly protected by locks or other synchronization?

Have all possible error conditions or exceptions been taken into account? For exceptions, are they handled properly, or just ignored?

Does the code fulfill its specification?

Does the code use libraries and other interfaces correctly?

[C/C++ language] Is there any possibility of buffer overflow?
[C/C++ language] Are there possible memory leaks or dangling pointers?

Code/Style

Is there duplicate code (XP: once and only once)?

Was the code adequately documented for you to understand it?

Have all constants been named?

Testing

Is the test suite adequate, both from a code coverage perspective and a coverage of functionality perspective? If not, what tests are missing?