Overview of Software Development Practices

Relevant Links

- Code Complete, 2nd Edition[1] A bible/handbook on software development. Must read for any developer.

Notes

Overview of Software Development Practices

Writing new code to perform some task is but a small, albeit important, part of being a software developer. But a considerable part of your work involves other tasks. The key common element in all of these is the need to manage a large code base:

> Software development practices need to consider a reality of tens or hundreds of thousands of lines of code to be managed over many years by multiple programmers, along with an ever-changing set of requirements.

Here is a brief listing of some common needs:

- Decide the program requirements.
- Keep track of progress towards meeting the requirements.
- Review other people's code.
- Have others review your code.
- Add new features without breaking existing features.
- Manage listing and priority of fixing for existing problems.
- Rework existing code without breaking functionality.
- Incorporate your code changes into a larger codebase, without conflicts with other people's changes.
- Review and incorporate someone else's code contributions into your code base.

A number of common practices have been developed in response to these needs:

Modularity Designing an application as a set of loosely couple modules, each with its own clear requirements and interactions. If each part of your code depends on as little as possible, it is easier to make changes without breaking stuff.

[1] http://learning.acm.org/books/book_detail.cfm?id=1096143&type=safari

Testing *Unit tests* that test single functions in the code, as well as *functional/integration tests* that test overall behavior and module interaction. These offer a certain degree of reliability when changing code: If you break something, your tests will tell you about it (usually).

Version Control Version control allows you to see the evolution of code over time: Who committed what and when, and how that differs from before. You can revert changes, merge changes, etc.

Issue/Bug Tracking There should be a simple mechanism for identifying issues that need to be addressed, documenting a discussion of those issues, setting up priorities and assignments.

Coding Standards It is important that everyone follows a common and clear coding standard. When reviewing someone's code, it is hard enough to try to understand what they are trying to do without getting distracted by the lack of a semicolon or some weird spacing between terms, or a bizarre control flow structure.

Documentation When a project involves many constituents and a long passage of time, proper documentation is crucial. This occurs at various levels: user manuals, function and module documentations that turn into developer docs, and inline comments on complex code parts are all important components of a large project.

Pair Programming Working with another programmer at your side as you consider each line of code prevents errors early on, and helps programmers learn from each other and establish common patterns.

We will try to touch on all those in this class, both during the labs and as you work on your term project. In particular, this rubric[2] (pdf version[3]) will be used to evaluate your project (and your senior projects when you get to them).

[2] https://hanover-cs.github.io/Comps_Documents/development_rubric.html
[3] https://hanover-cs.github.io/Comps_Documents/development_rubric.pdf