6.S096 Lecture 7 – Intro to Projects

Unit testing, third-party libraries, code review

Andre Kessler

Outline

1. Best Practices
2. Final Project
3. Structure of a Large Project
4. Unit Testing
5. Code Reviews
6. Wrap-up

Best Practices

If you haven't already, look at our standards page.

General Important Points

- Write in a clear and consistent coding style.
- Code should be self-documenting.
- Keep your headers clean.
- Don't expose your class' private parts.
- Use `const` wherever possible.
- Write portable code.
- Don't leak memory!

Final Project

Groups of 2-4 people; 3 recommended

- Writing an interactive *n*-body gravity simulation- details to come tomorrow
- External libraries: using OpenGL, gtest framework
- Lots of room for hacking independently and getting cool additional features!

How are projects different?

- We'll have lots of programmers, a large amount of code, many updates over time.
- Need a revision control system. Examples: git, SVN, CVS, Mercurial are some free ones.
- We said many updates over time: lots of potential for bugs. Code should be tested.
- Need a unit testing framework. Examples: gtest, cppunit.
- Still want to be able to build our code in one go ("make").
- May want to use an integrated development environment, or IDE. Examples: Microsoft Visual Studio, Eclipse, XCode.

Structure of a Large Project

```
|-- build
|   `-- project
|-- include
|   `-- project
|-- install
|   |-- bin
|   |-- include
|   |-- lib
|   `-- test
|-- make
|   |-- all_head.mk
|   |-- all_tail.mk
|   |-- project.mk
|   `-- project-test.mk
...
 |-- Makefile
 |-- src
 |   |-- gcd.cpp
 |   `-- rational.cpp
 |-- test
 |   |-- project-test.cpp
 |   `-- rationalTest.cpp
 `-- third_party
     `-- gtest
```

Separation of build and source

Have a clean build.

- Since this is definitely under revision control, we want to keep our directories free from clutter
- Hence, all object (.o) files will go in the bin/ directory.
- Third-party libraries live in their own directory `third_party/gtest` or whatever.
- Headers for our project named "project" are deployed to the install directory.

Be able to build in one step

- We have an upper-level Makefile so that we can still just `make` our project.
- However, that's been split up into more modular sub-makefiles (`make/*.mk`).

Unit Testing and Test-Driven Development

Testing your source code, one function or "unit" of code at a time.

- Test-driven development: write the tests first and then write code which makes the tests pass
- Decide how you want the interface to work in general, write some tests, and go develop the specifics.

gtest: the Google C++ Testing Framework

Highly cross-platform, available from here.

- Runs your full suite of tests (potentially each time you compile)
- Tests are very portable and reusable across multiple architectures
- Powerful, but very few dependencies.

Example from their primer:

```
ASSERT_EQ(x.size(), y.size()) << "unequal length";

for (int i = 0; i < x.size(); ++i) {
  EXPECT_EQ(x[i], y[i]) << "differ at index " << i;
}
```

Code Reviews

Why do code review?

- Two pairs of eyes are better than one. Catch bugs early.
- Forces someone to read it. If your code wasn't readable, they'll let you know and help you improve it.
- According to Steve McConnell in *Code Complete*:
  In a software-maintenance organization, 55 percent of one-line maintenance changes were in error before code reviews were introduced. After reviews were introduced, only 2 percent of the changes were in error. When all changes were considered, 95 percent were correct the first time after reviews were introduced. Before reviews were introduced, under 20 percent were correct the first time.

Tips for effective code review

- Most important features for the code: correctness, maintainability, reliability, and performance. Consistent style is good, but those other points come first!
- Keep your review short and to the point.
- Check the code for compliance with the class coding standards.
- Take the time for a proper review, but don't spend much more than an hour; additionally, don't review much more than about 200 lines of code at once.

Examples

Let's see some examples...

Pimpl idom...

Wrap-up & Monday

Third assignment due Saturday at midnight.

Final project to be released Saturday.

Class on Mon.

- Concerns of large projects

Questions?

- Office hours immediately after class today
- Office hours Mon, Tues

6.S096 Effective Programming in C and C++

IAP 2014