Engineering Talk 0: Programming in a Project Team

0.0 Motivation

Big projects require more than one person (or a long, long time)

It's very hard to make software project teams work correctly

0.1 Big projects

0.1.1 What's a big project?

- **Time / work estimation is hard**

0.1.2 Can a project be efficiently partitioned?

- Partitionable task decreases in time as you add people
- But, if you require communications:
    - Time reaches a minimum bound
    - With complex interactions, the time goes up!

This is the *mythical person-month* problem!

You estimate how long a project will take. But, then it starts to fall behind its deadlines, so you add more people and it takes more time!

0.1.3 How to partition a project?

- **Functional**
    - Person A implements threads, Person B implements semaphores, Person C implements locks…
    - Problem: Lots of communication across APIs.

      If B changes the API for semaphores, A may have to make changes.
- **Task**
    - Person A designs, Person B writes code, Person C tests
    - May be difficult to find right balance, but can focus on each person's strengths (Theory versus systems hacker).
    - Debugging is hard

Most CS 162 project teams are functional, but people have had success with task based divisions.

0.2 Project teams make life hard

0.2.1 Communication

- More people means more communication
- Miscommunication is common!
- Who makes decisions?

0.2.2 Coordination

- More people means no one can make every meeting.
  They miss decisions, and the discussion/reasoning associated with them.
- People have different work styles.
  Some people work in the morning, some at night. How do you decide when to meet or when to work together?
- What about project slippage?
  Adding people is a road to disaster, especially when a project is already underway.
  - Current project members have already figured out how to work together
  - More people means more communication and coordination (the mythical person-month problem!)

0.3 Solutions

How do companies succeed?

Most don't. But here are some of the rules used by those that do succeed:

0.3.1 People are human, recognize it

People will make mistakes, miss meetings, miss deadlines, etc. But, you have to live with it and adapt – not get angry.

It is better to anticipate problems, then to have to clean up afterwards. Following these guidelines will help, but they're not a perfect solution.

0.3.2 Document, document, document

Documentation is the key.

Why document?

- Expose decisions and communicate to others
- Easier to spot mistakes ahead of time
- Progress is easier to follow

What to document?

- Everything (but, don't overwhelm people or nothing will be read)

Standardize!

- One programming format: variable naming convention, tab indents, etc.
- Header file format
- Comments (Requires, effects, modifies)

Suggested documents:

1. Project objectives: goals, constraints, and priorities
2. Specifications: The manual plus performance specs.
3. Meeting notes. Document all decisions (then you can cut & paste for the design document)
4. Schedule. This document is critical!
5. Organization chart
6. Budget, space allocations, marketing (estimates, forecasts, prices)

0.3.3 Use software tools

Using the right software tools will make your life easier.

- Source revision control software
  Easy to go back and see change history (where and why did a bug get introduced)
  Communicates changes to everyone (use cvs's features)

- Use automated testing tools
  Write scripts for non-interactive software.
  Use "expect" for interactive software.
- Use E-mail and instant messaging to leave a history trail

0.3.4 Test continuously

Integration tests all the time, not at 11pm.

- Write dummy stubs with simple functionality.
  Let's people test continuously, but more work.
- Schedule periodic integration tests.
  Get everyone in the same room, check out code, build, and test.
- Test early, test later, test again.

0.4 Summary

Start early, develop a good organization plan, document everything, use the right tools, and develop a comprehensive testing plan.