ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 24 — March 12, 2013

Patrick Lam *version 1*

Design and Planning

We'll start by comparing design and planning for traditional engineering projects versus software projects. Traditionally, you would:

- solicit requirements;
- make a design;
- analyze the design (with calculus);
- stamp and sign the plan.

Then, someone else implements/builds the plan.

People tried this for software as well:

- solicit requirements;
- make UML diagrams;

and hire code monkeys to implement the design.

However, this works poorly.

The Role of Prototyping. Requirements are always difficult to formalize; this is a problem for all types of engineering, and definitely for software. Prototyping is one way to mitigate the risk of building the wrong thing.

> The management question, therefore, is not *whether* to build a pilot system and throw it away. You *will* do that. [...] Hence *plan to throw one away; you will, anyhow.*

Frederick Brooks, *The Mythical Man-Month*, 1975.

The waterfall model is a straw-man for many reasons, but the biggest one is because it pretends that prototypes aren't necessary. You can never build a system without a prototype, because you never really know what you need until you see it. What's more, needs change over time.

Another solution to the problem of shifting and unknown requirements, along with prototypes, is the concept of *iteration*. Different models have different iteration strategies.

Agile Lifecycle Models. The key point is to recognize that change is inevitable, and to deal with it on-demand. Agile models, such as extreme programming, advocate dealing with change on-demand. In principle, they can handle changing requirements much more smoothly.

However, agile models require suitable developers, who must be good at communicating with each other—these models are always highly collaborative. Due to collaboration, agile methods are supposedly resistant to 1) changes to the team over time and 2) differences between experience levels of the developers.

Choosing a Lifecycle Model

Recall that the main difference between models is the pace of iteration.

- With large teams, diverse stakeholder groups: slower iterations;
- With small teams, uncertain requirements, complex technologies: faster iterations.

Questions to Think About

The answers to these questions will affect the pace of iteration and hence the lifecycle model which you'll want to select.

- Do you understand the customer requirements?
- Will you need to make major architectural changes?
- How reliable does the system need to be?
- How much future expansion and growth do you foresee?
- How risky is the project?
- Is the schedule heavily-constrained?[1]
- What is going to change during development?
- How much do customers need visible progress?
- How much does management need visible progress?
- What experience does the design team bring?

Software Process Improvement

We saw engineering design processes last week. One of the things you can design is the design process itself. Here's what you can do to try to improve your software lifecycle model:

- first, figure out what you mean to be doing: document your organization's software process;
- next, figure out what you're actually doing: ensure that you're following the existing process;
- finally, identify areas of potential improvement and implement them.

In continuous process improvement, you constantly review the software development process and its effect upon development. Continuous improvement also implies identifying and fixing issues that are beyond the scope of individual projects: you can find and apply best practices to all projects. Changes can allegedly yield dramatic increases in development capability.

[1]Note: wishing for faster progress won't make it so.