Lecture 35 — April 8, 2013

Patrick Lam *version 1*

Software Maintenance

Your engineering programme has a lot of emphasis on design; who ever heard of a Fourth Year Maintenance Project, for instance? However, in the real world, maintenance accounts for a lot of engineer effort.

Software maintenance modifies existing software to fix defects, improve performance, or make the software work in new environments (porting).

Software maintenance is hard. You have to understand the existing code, which can be difficult (no matter whether you or someone else originally wrote it). It's unglamorous, especially if you're fixing bugs. It's constrained; you better not break compatibility.

It turns out that a lot of software maintenance goes beyond fixing bugs. According to T. M. Pigosky[1], approximately 80% of software maintenance activities are unrelated to defect fixes. Here's a classification of different types of maintenance; these all apply to already-shipped code.

- *Corrective Maintenance*: correct known defects;
- *Adaptive Maintenance*: keep a software product usable in a changing environment;
- *Perfective Maintenance*: improve performance or maintainability; and
- *Preventive Maintenance*: correct latent faults in the product before they manifest themselves.

Patching can be problematic and lead to gnarly code with no design. What's the solution?

It is always tempting to start over from scratch; sometimes it looks like the existing software is beyond hope, and it's more fun to redesign rather than maintain. However, the "second system effect" says that you may well do worse by starting over.

Managing Maintenance. Software projects are huge and therefore have huge numbers of defects, shipped software is by no means exempt from defects. Many projects have tens of thousands of known defects[2].

Triage is key to avoiding analysis paralysis: some defects are more important than others. Security fixes should go in right away, while minor defects can wait. For instance, Microsoft has deployed monthly software patches in the past, but pushes security fixes as needed.

[1] T.M. Pigosky, *Practical Software Maintenance*, John Wiley & Sons, 1997.

[2] We've found that the average bug lifetime in the Linux kernel was 1.38years.

Patch discipline. Any change can cause problems, even more problems than it solves. Before pushing a change, be sure to check that it makes things better rather than worse. Testing is particularly critical; imagine what would happen if a patch to the Microsoft Windows operating system caused 1% of the world's computers to breakdown?

You can use reviews, regression tests, and other verification techniques to help ensure that you do less harm than good.