COSC345 Week 21 Notes

There are handouts about the spiral model: a copy of Boehm's paper, and the Wikipedia page about it.

"Prototyping is about risk reduction: if you don't have a question, you don't need a prototype" is the heart of it.

The edition of Pressman I intended is the old one listed in the Bibliography.

The section about "Developers and Users are different" is really quite important. It might even deserve to be in a lecture of its own, though not necessarily in this paper. I once had a student (who became a highly paid consultant in the UK) who was into Neuro-Linguistic Programming; it seems they not only classify people's thinking styles as (Visual/Auditory/Tactile)×(Digital/Analogue) but have techniques for determining which someone is. It's quite interesting to get someone to describe something abstract and note what kinds of metaphors they use; the point is that people use different ones. There are the classic Four Temperaments (Choleric, Melancholic, Phlegmatic, and Sanguine) and the modern refinement of it, the Enneagram. People always seem to be fascinated to be told about themselves, so getting the members of the class to do some kind of quick personality test and then collecting the results anonymously and tabulating them might be a fun way to make the point that people really do think differently. Here's one: http://similarminds.com/test.html

Of course, it's not just developers and users who differ. Users differ from each other. The point is to remind students that they cannot expect other people to think exactly like them and to enjoy exactly what they enjoy. (Tastes in music, for example.) You can't be sure whether other people will like your program until they have tried it.

As for prototyping languages, there are the obvious ones like Python and (ick) Perl. I point to Haskell. For example, I've gone through the Personal Software Process exercises using Haskell and got remarkably short answers in remarkably short times. (Files available on request.) I also have some Programming Contest problems in C and Haskell, although they were done mainly to demonstrate Haskell input/output and don't show off Haskell's strengths.

I now just mention Groovy and Scala; they should be more prominent but I don't know them well enough yet. I would mention Ruby, which has a really neat test development kit called Cucumber, except that Ruby is a slow compromise between Python and a gutted Smalltalk. Actually, for prototyping, it might be quite a good choice: check out what libraries are available for it!

If you have anyone who knows Smalltalk, Squeak and Pharo are free, and VisualWorks NonCommercial is free for academic use and much faster than Squeak. GNU Smalltalk is free. Dolphin Smalltalk is easily the *prettiest* system I've ever seen for building Windows programs. It's really nice showing how you can build a prototype of something quickly in Smalltalk and then measure it in various ways.

Something should probably be said about prototyping environments. Our aim in prototyping is rapid development, so a refactoring browser (first developed in Smalltalk) or a unit testing framework (jUnit is a clone of SUnit, which is of course for Smalltalk) can save quite a bit of time, as can a body of examples (like you get in Smalltalk) that you can copy and modify. The lecture does make the point that a library that provides most of what you need for your prototype already is one of the most important things.

Something I mention in lectures but isn't on the slides is that one of the things that's often sacrificed in prototyping is security. Prototypes are, after all, used in-house. Anything we learn from a prototype has to be applied in a framework that takes security seriously.

I don't say anything much about Domain Specific Languages, which is a shame.

There is a paper, "Middleware Trends and Market Leaders 2011" by A. Dworak, P. Charrue, F. Ehm, W. Sliwinski, M. Sobczak, of CERN, which appeared in the Proceedings of ICALEPCS2011. You can find it on-line at https://accelconf.web.cern.ch/accelconf/icalepcs2011/papers/frbhmult05.pdf. It's a nice example of what prototyping is about. CERN has oodles of measurement devices on a network. These things used to be integrated using CORBA. However, CORBA had some problems, so they were planning to take advance of the 1-year accelerator shutdown from the end of 2012 to rebuild some of this software. They put together a list of 10 candidate middleware systems (including CORBA as one of them). Applying criteria like size, availability, and documentation they reduced the list to 6 systems, which they built benchmarks for. That let them reduce the list to just 3 systems. At the end of the paper they were planning to build protoype programs in all of these 3 in order to evaluate suitability for their needs, and then switch over. The risk here is "choosing the wrong middleware system" and the prototyping goes through three stages:

1. "checklist prototype"
2. small scale prototypes evaluating library size and performance
3. medium scale prototypes trying to use the libraries in realistic code

Scripting languages.

You **must** read the Wikipedia article about scripting languages, http://en.wikipedia.org/wiki/Scripting_language.

The handouts include a simple web server module written in Erlang. It's just 38 SLOC, with comments and blank lines stretching it to 66 lines. It can be so short because the Open Telecom Platform that comes with Erlang already supports so much.