Demos and Prototypes

JonJon Clark (clrjon005@myuct.ac.za)

CS3003S: Advanced Software Development

Slides taken from Melissa Densmore

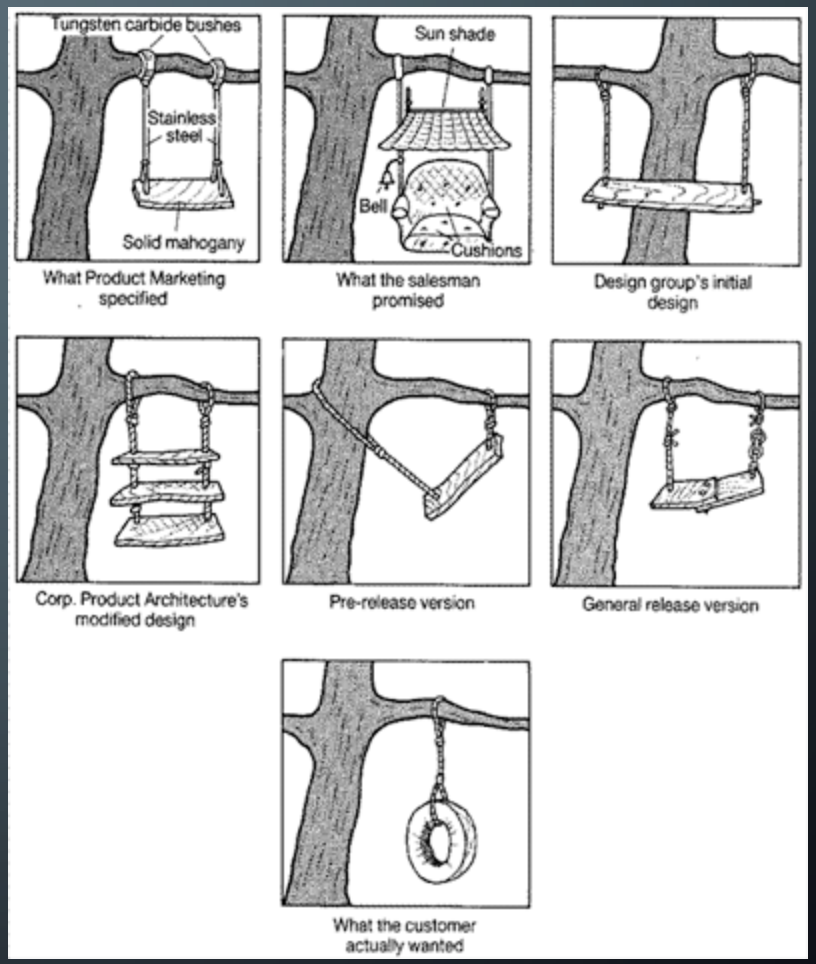
What is a Software Prototype?

- It depends…
- Helps to investigate
  - Technical issues
  - Work flow, task design
  - Screen layouts, information display (e.g. user interface)
  - Difficult, controversial, critical areas
  - Match between engineering and customer specification
- Demonstrate proof of concept
- More concrete than a narrative

"Initial, incomplete version of a software system being developed that is used to learn about the problem, explore designs and solution techniques."

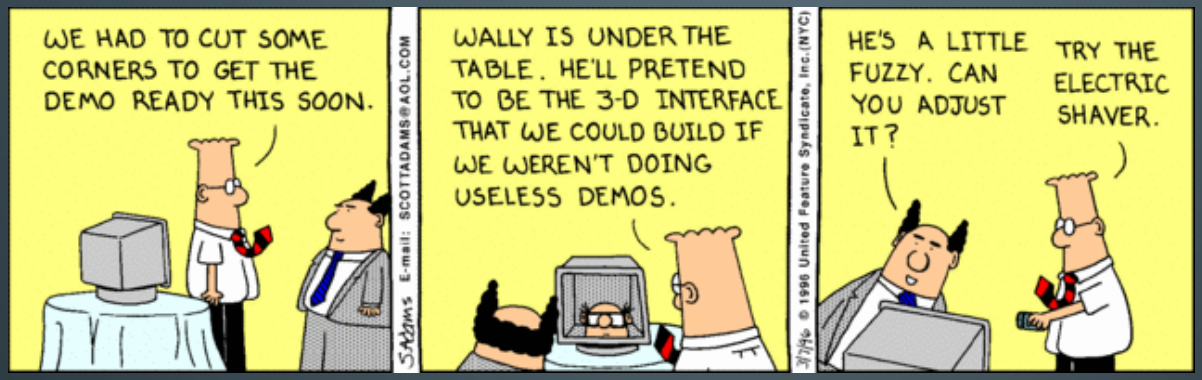
Why Prototype?

- Enable evaluation and feedback (central to design methodology)
- Improves communication within a team
- Testing ideas out – encourages reflection
- Answer questions!
- Explore alternatives



Prototypes require compromises!

slow response, sketchy icons, fake data, limited functionality, limited parameters, etc



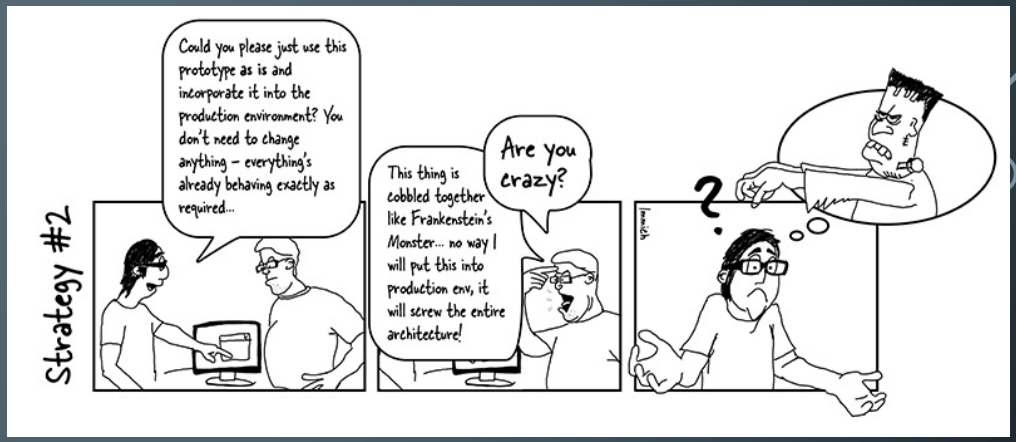
Horizontal Prototype

wide range of functions, very little detail on each e.g. testing user interface before backend is developed

Vertical Prototype

Provide a lot of detail for a few functions

Throw-away Prototypes



- Address high-risk issues
  - Uncertainty in requirements
  - User interface design
  - Alternative implementation strategies
  - Technology platform
- Only enough effort to help address specific issues
  - Focus only on the issue, ignore all others
  - No unit tests
  - Too much effort will make you hesitant to throw it away
- Great for trying alternative ideas

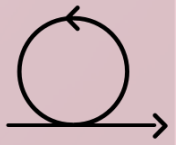
Comic from https://www.smashingmagazine.com/wp-content/uploads/2010/04/ResurrectingUIPrototypes-ComicStrip-Strategy2.jpg

Evolutionary Prototypes

- Intended to be early, not necessarily release-able version of the actual software…
  … will *evolve* into the final product
- Quality is important (unit tests are back!)
- Can be put to limited use (e.g. demo, constrained env)
- Implementing and validating well-understood requirements → providing opportunity for change and reorienting if necessary

Potential Weakness: customers/testers may be hesitant to criticize underlying problems in something that seems heavily invested/developed!

Prototypes vs the Final System?

Throw-away (revolutionary)	Evolutionary (tracer-bullet)	Iterative (agile)
• Clarifies requirements for system 	• Part of the system 	• Product of an iteration/increment, which will lead to final system • Informs later increments 

The Demo

CommitStrip.com

What can go wrong?

How to beat Murphy!

1. Make sure your system works!

How to beat Murphy!

1. Make sure your system works!
2. Make sure your system works!

How to beat Murphy!

1. Make sure your system works!
2. Make sure your system works!
3. Also **practice** and rehearse the demo

How to make sure the system works..

- No last minute code changes
- Demo stable version with fewer features rather than an untested version with more
- Rehearse the entire demo!
- No, really resist the temptation to tweak the demo!
- Test any peripherals (e.g. speaker, projector, scanner)

NO SOFTWARE CHANGES TO MAKE HERE

Preparing for the Demo

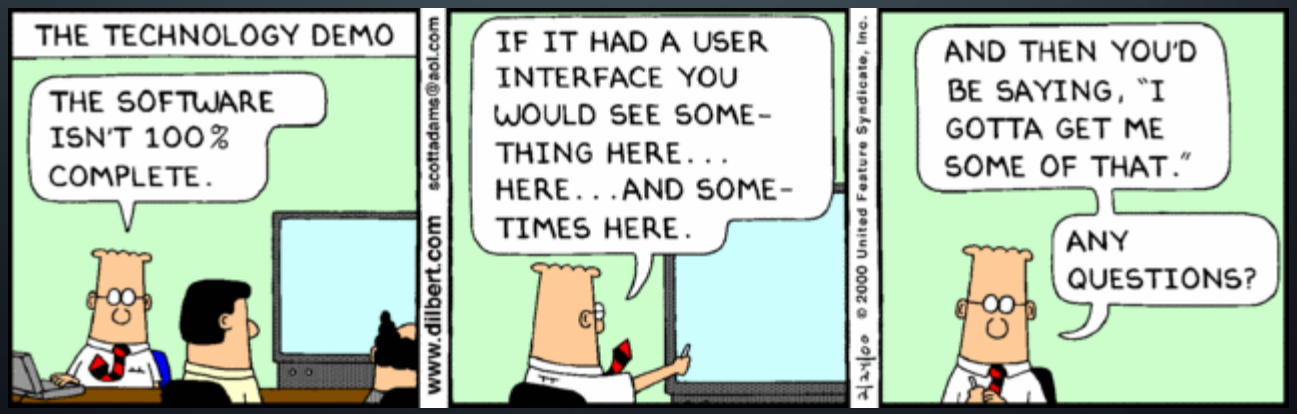
- List (and test) the tasks you will demonstrate
- Plan what to say:
  - (Very) short overview of the system
  - Explain what the demo will show
- Plan what to say during each test, work out the steps involved
- Ensure that each team member participates and can demonstrate contribution to knowledge
- Rehearse and time the demo
- Think of possible questions – prepare answers
  - What has been left out of the demo and why?
  - What are the roles of the team members?

Before the Demo

- Arrive early
- **Make sure the system works!**

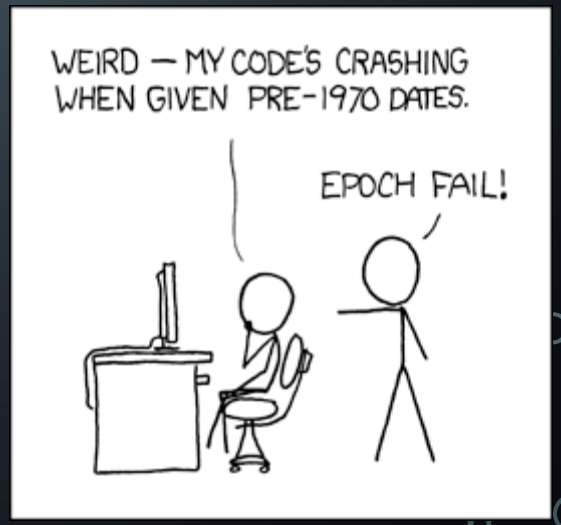
During the Demo

- Be prepared to change direction in response to questions
- Allow and encourage the client to try things (if you are prepared for this and it's safe)
- Be prepared to show code or data or documentation
- Be prepared to answer questions



Uh oh, something bad happened

- Stay calm
- Don't make effusive apologies if anything goes wrong
  - Fix the problems quietly
  - Be able to restart the system without recompilation
  - Or have a backup system ready
  - Let one person go on explaining the system
- Don't argue with or contradict team members



Questions?


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE