Testing in the Lifecycle

Conrad Hughes

School of Informatics
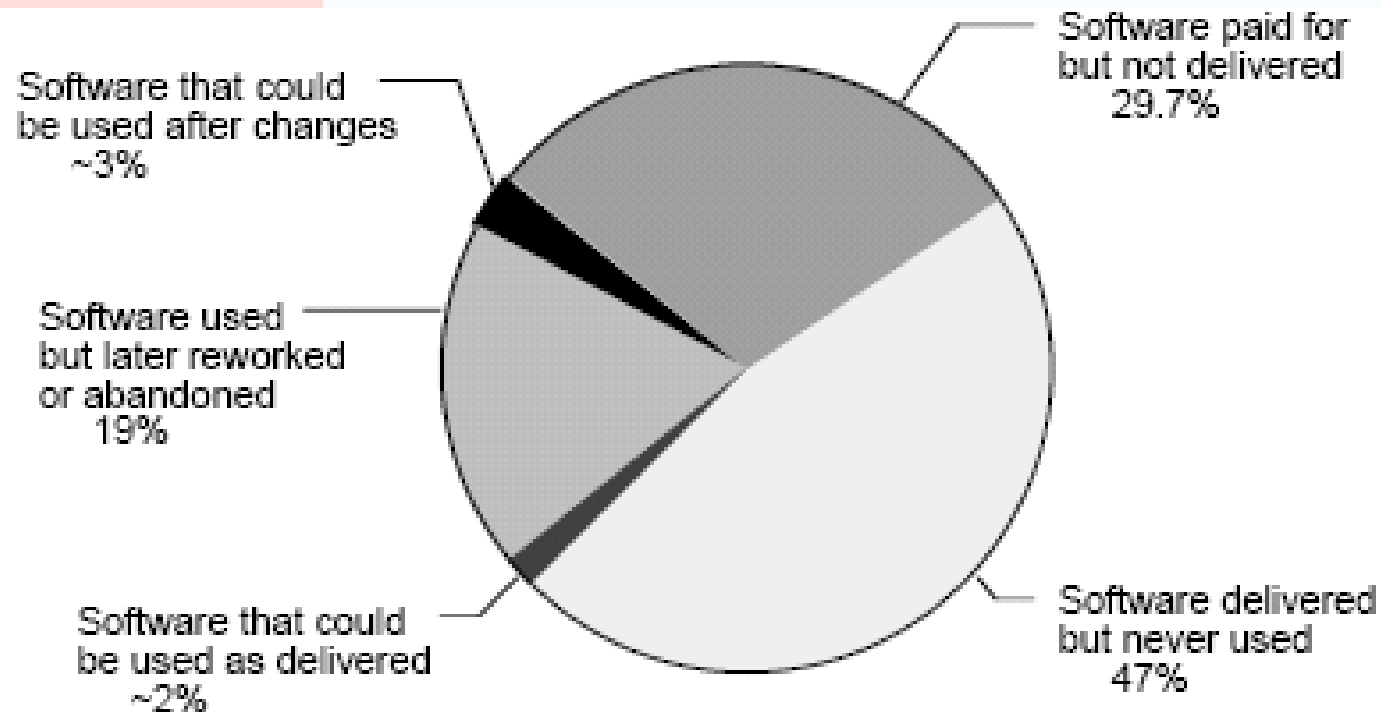
Slides thanks to Stuart Anderson

informatics

Software was difficult to get right in 1982

Software that could
be used after changes
~3%

Software used
but later reworked
or abandoned
19%

Software paid for
but not delivered
29.7%

Software delivered
but never used
47%

Software that could
be used as delivered
~2%

Year 1982: Nine Contracts Totalling $6.8 Million

It was still difficult in 1995
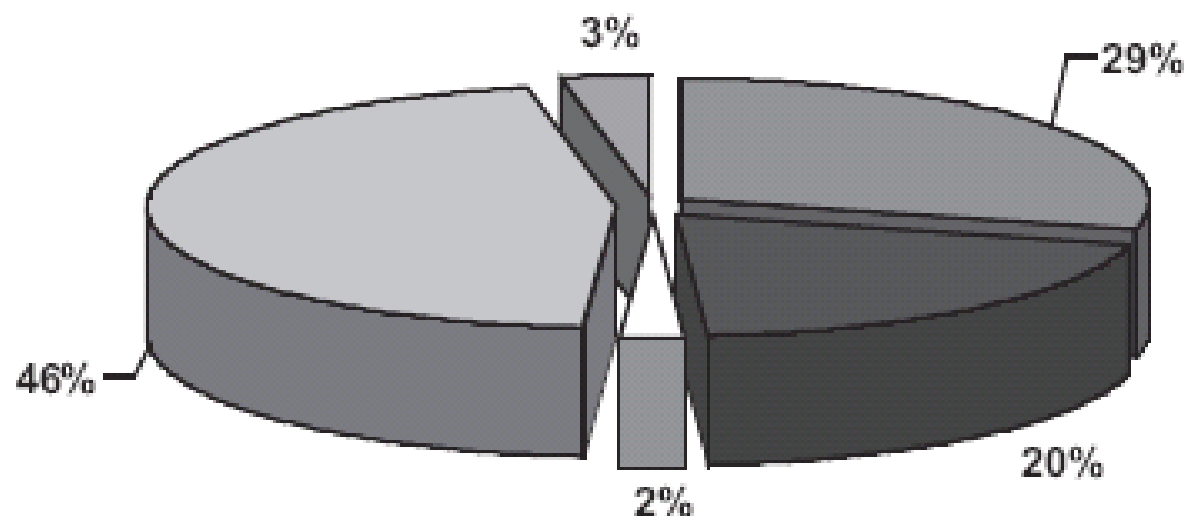

3%

29%

46%

2%

20%

Figure 1: *Findings of a 1995 Department of Defense Software Study*

☐ Software paid for, but not delivered - 29%

■ Software used, but extensively reworked or abandoned - 20%

☐ Software used as delivered - 2%

☐ Software delivered, but not successfully used - 46%

☐ Software used after changes - 3%

Total Software Costs - $35.7 billion

Today

Success rate of government IT projects and programmes

63%

30%

7%

- On budget, on time, on spec
- Anything in between
- Never saw the light of day
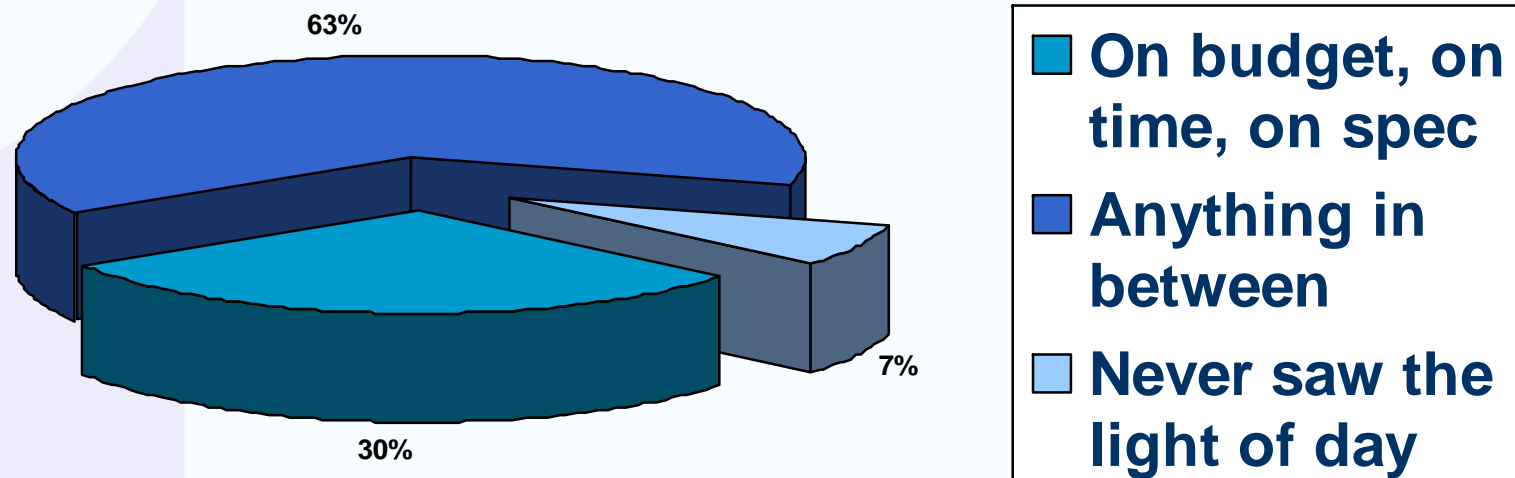
Source: The Guardian, 18 May 2007
Figures from Department for Work and Pensions spokesman (63%)
And Joe Harley, Chief Information Officer, DWP (30%)

And testing costs are significant

	METRIC	Proj. A	Proj. A1	Proj. B	Proj. C
1.	Testing as a part of the overall project effort	= 15.3%	= 11%	= 16.6%	=8.57%
2.	Integration effort per line of code (man months /kloc)	= 0.4			
3.	Cost of Requirement Change [man hours / change]	23.3	16.7	13	8.75
4.	Functional Coverage	Coverage =80%		Not available	Not available
5.	Code Coverage	Coverage =55% According to literature			

Costs of fixing defects found at different stages

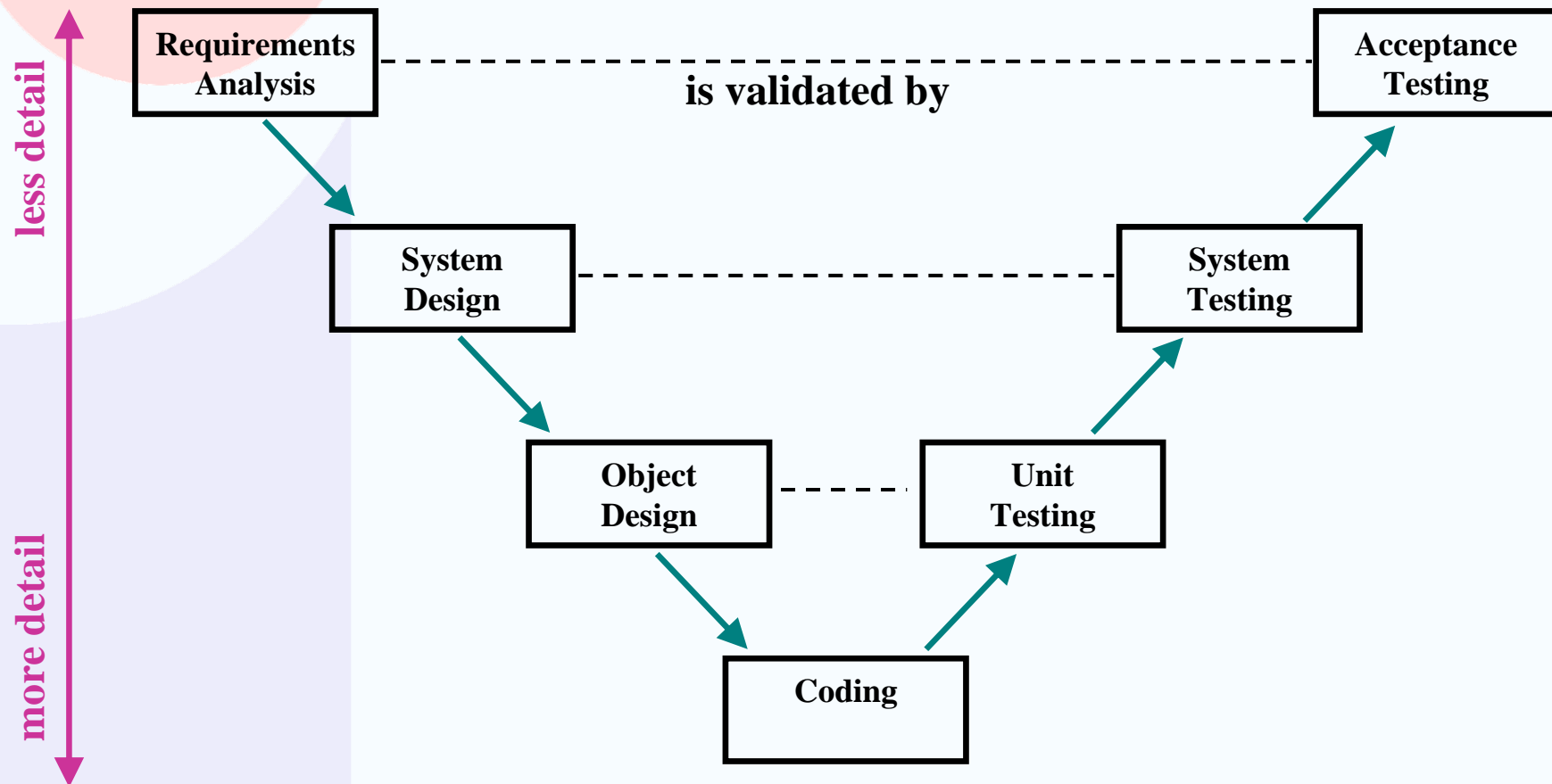Source: Applied Software Measurement, Capers Jones, 1996
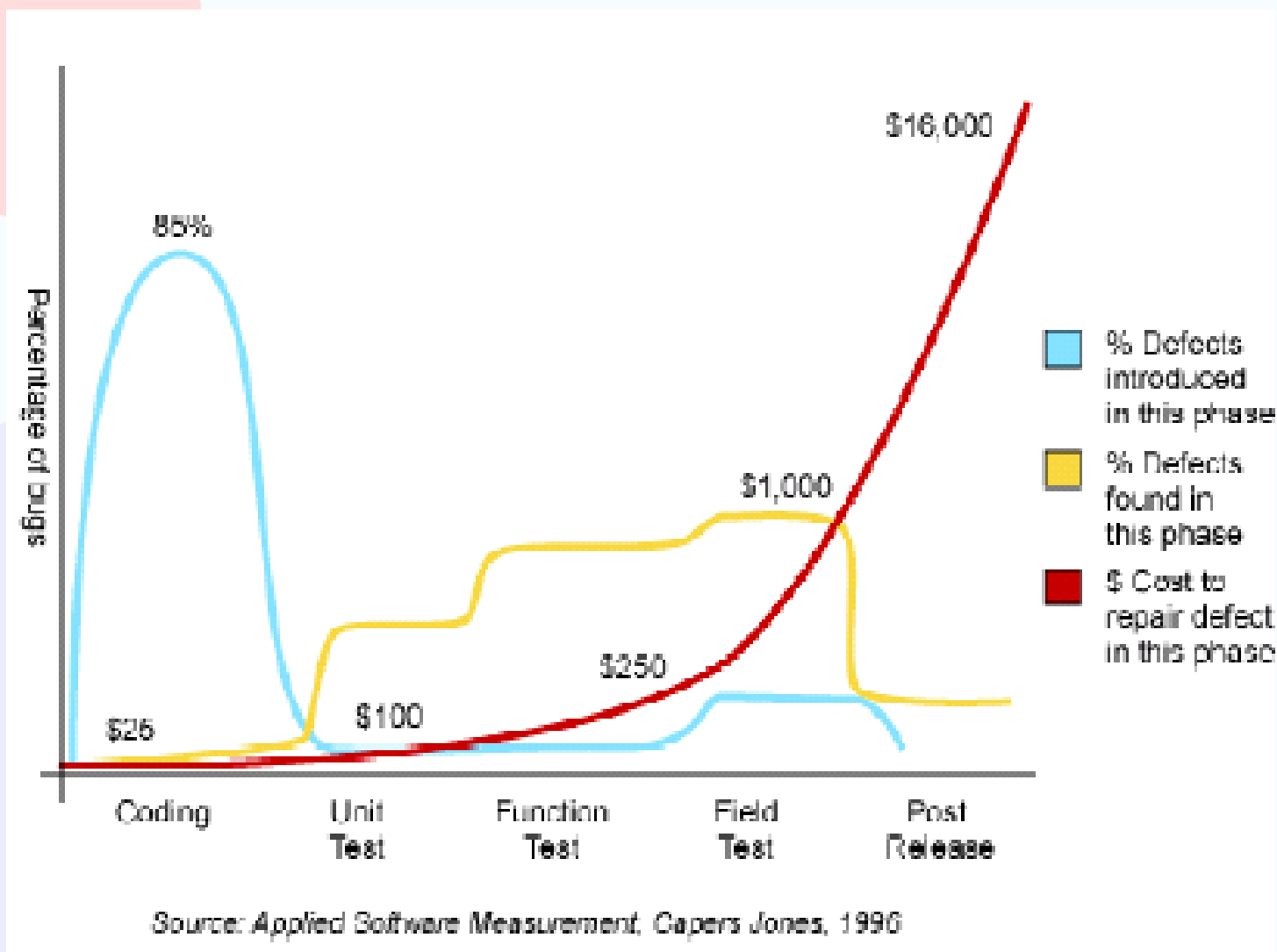
Costs of Defects

- Defects in the specification are even more costly to remove if we don't eliminate them early.
- Different software lifecycles distribute testing (verification – "building the thing right" and validation – "building the right thing") differently.
- The different distributions of test activity can have an impact on where bugs are discovered.
- We consider three representative lifecycles and consider where testing is located in each:
  - The V-model
  - Boehm's spiral model
  - eXtreme Programming ("XP")

Recap: "waterfall" model of software development

1. Requirements
2. Design
3. Implementation
4. Testing
5. Release and maintenance

■ Sequential, no feedback

■ Ironically its "author", Royce, presented it as an example of a broken model

V-model

less detail

more detail

| Requirements Analysis | .... is validated by .... | Acceptance Testing |

System Design .... System Testing

Object Design ---- Unit Testing

Coding

build system ← → validate system
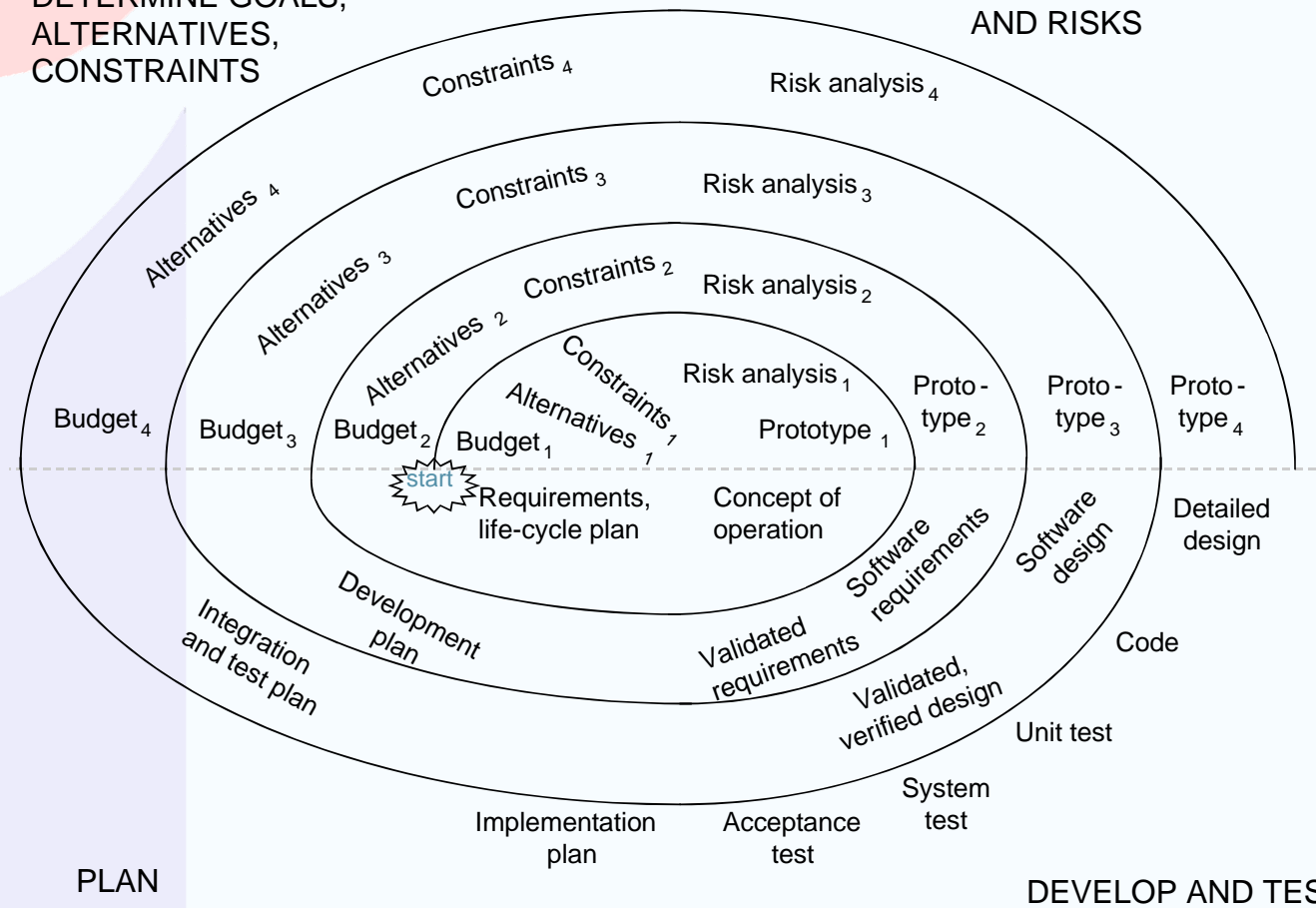
V-model Rationale

- This is a modified version of the waterfall model.
- Tests are created at the point the activity they validate is being carried out.
- So, for example, the acceptance test is created when the systems analysis is carried out.
- Failure to meet the test requires a further iteration beginning with the activity that has failed the validation.
- V model is focused on creating tests in a structured manner.
- It is popular with developers of systems that are highly regulated because it is well suited to creating evidence that can be used to justify a system to a regulator.

Boehm's Spiral Model

DETERMINE GOALS, ALTERNATIVES, CONSTRAINTS

EVALUATE ALTERNATIVES AND RISKS

Constraints 4

Risk analysis 4

Alternatives 4

Constraints 3

Risk analysis 3

Alternatives 3

Constraints 2

Risk analysis 2

Alternatives 2

Constraints 1

Risk analysis 1

Proto-type 2

Proto-type 3

Proto-type 4

Budget 4

Budget 3

Budget 2

Budget 1

Alternatives 1

Prototype 1

start

Requirements, life-cycle plan

Concept of operation

Software requirements

Software design

Detailed design

Development plan

Validated requirements

Validated, verified design

Code

Integration and test plan

Unit test

System test

Implementation plan

Acceptance test

PLAN

DEVELOP AND TEST

Spiral Model Rationale

- The spiral model is focused on controlling project risk and attempting formally to address project risk throughout the lifecycle.

- V & V activity is spread through the lifecycle with more explicit validation of the preliminary specification and the early stages of design. The goal here is to subject the early stages of design to V&V activity.

- At the early stages there may be no code available so we are working with models of the system and environment and verifying that the model exhibits the required behaviours.
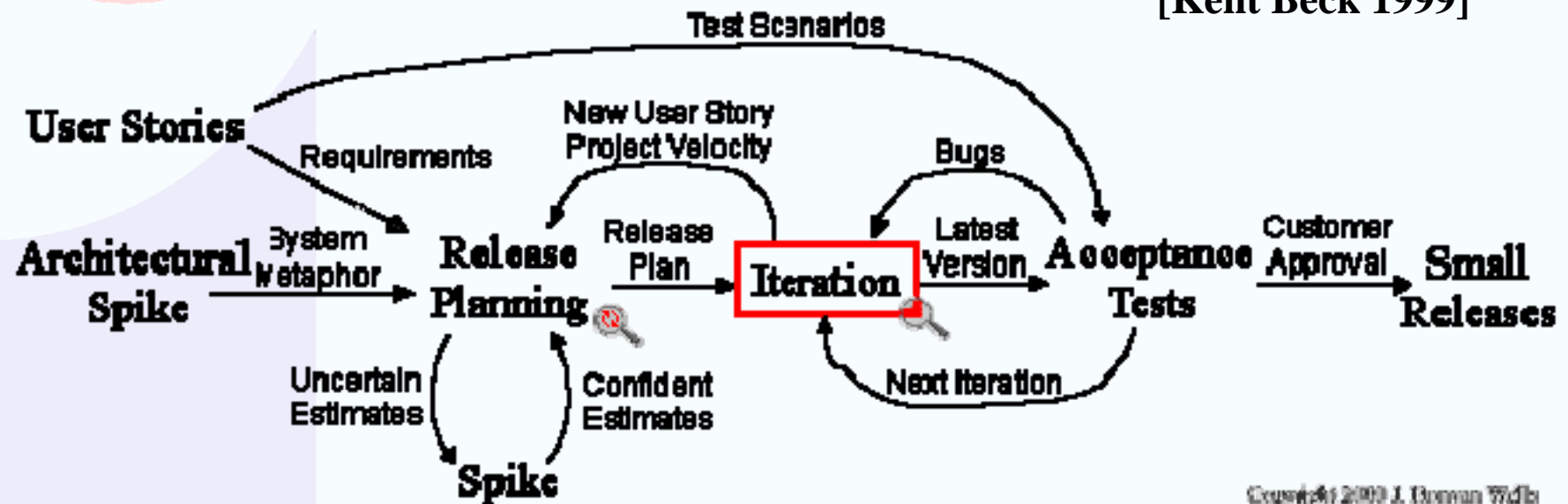
XP principles

- eXtreme Programming advocates working directly with code almost all the time.
- The 12 principles of XP summarise the approach.
- Development is test-driven .
- Tests play a central role in refactoring activity.
- "Agile" development mantra: Embrace Change.

1. Test-driven development
2. The planning game
3. On-site customer
4. Pair programming
5. Continuous integration
6. Refactoring
7. Small releases
8. Simple design
9. System metaphor
10. Collective code ownership
11. Coding standards
12. 40-hour work week

Extreme programming (XP)

[Kent Beck 1999]

Test Scenarios

User Stories

New User Story
Project Velocity

Bugs

Requirements

Architectural
Spike

System
Metaphor

Release
Planning

Release
Plan

Iteration

Latest
Version

Acceptance
Tests

Customer
Approval

Small
Releases

Uncertain
Estimates

Confident
Estimates

Next Iteration

Spike

Copyright 2000 J. Donovan Wells

http://www.extremeprogramming.org/map/project.html

Summary

- We have considered three different approaches to the lifecycle and have seen how testing fits in the lifecycles.
- Each approach will have a different testing cost and cost-profile through the lifecycle.
- Lifecycles are often dependent on the type of product and how well we understand project risk.