AALBORG UNIVERSITY

Testing and XP

Thursday: A4-108; 8:15

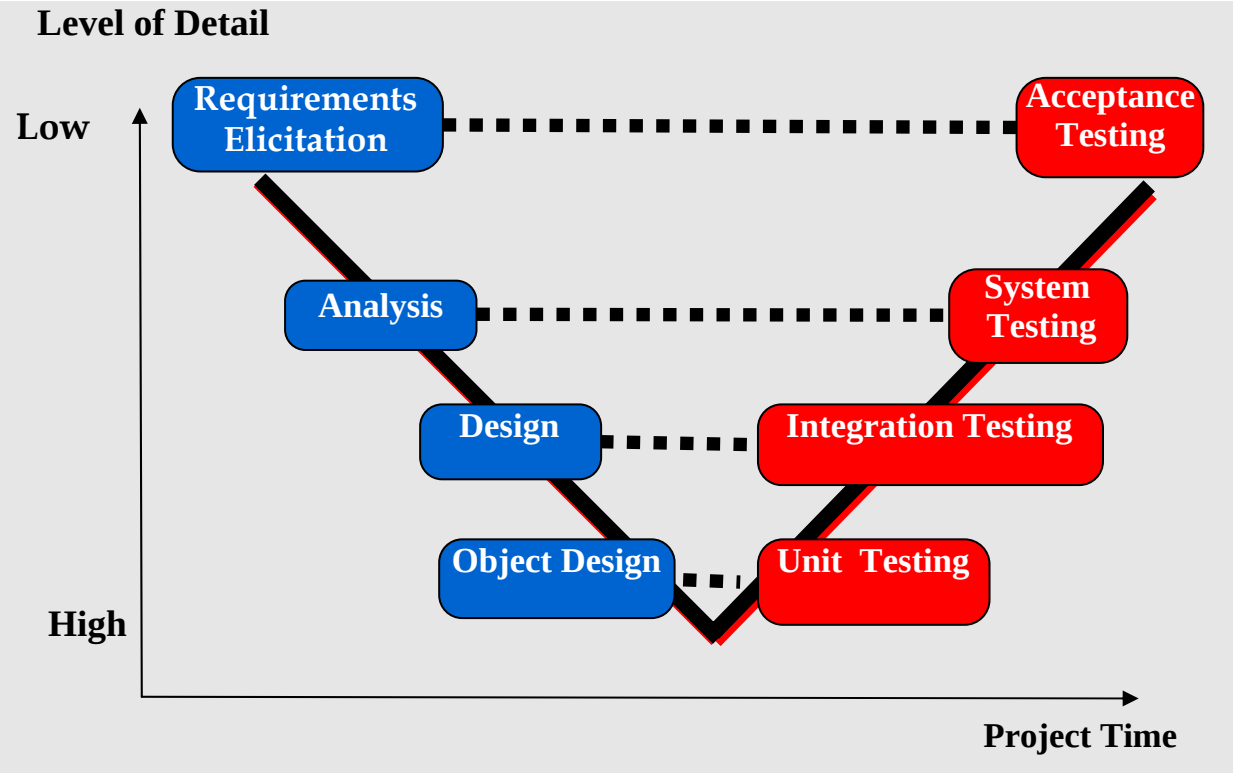
Peter Dolog
dolog [at] cs [dot] aau [dot] dk
E2-201
Information Systems
February 20, 2007

Software Testing

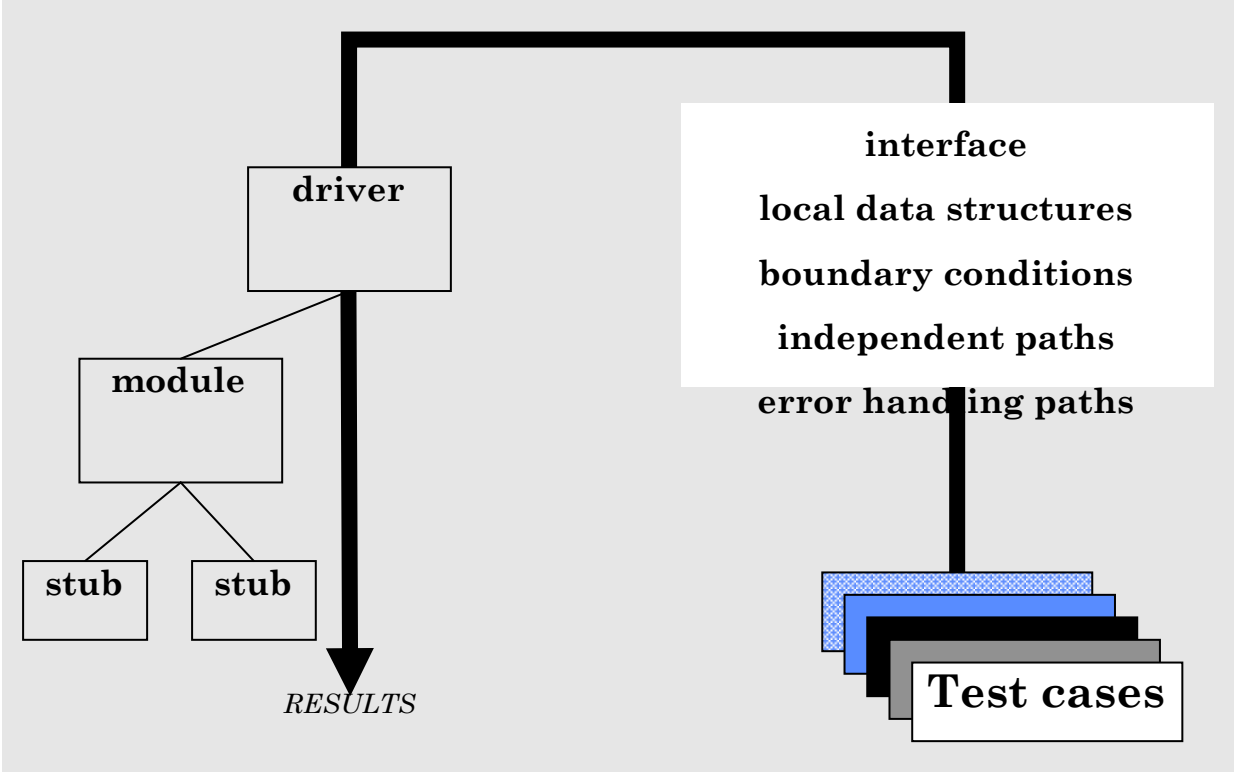
Software Testing

Modelling the software environment
Selecting test scenarios
Running and evaluating the test scenarios
Measuring testing progress

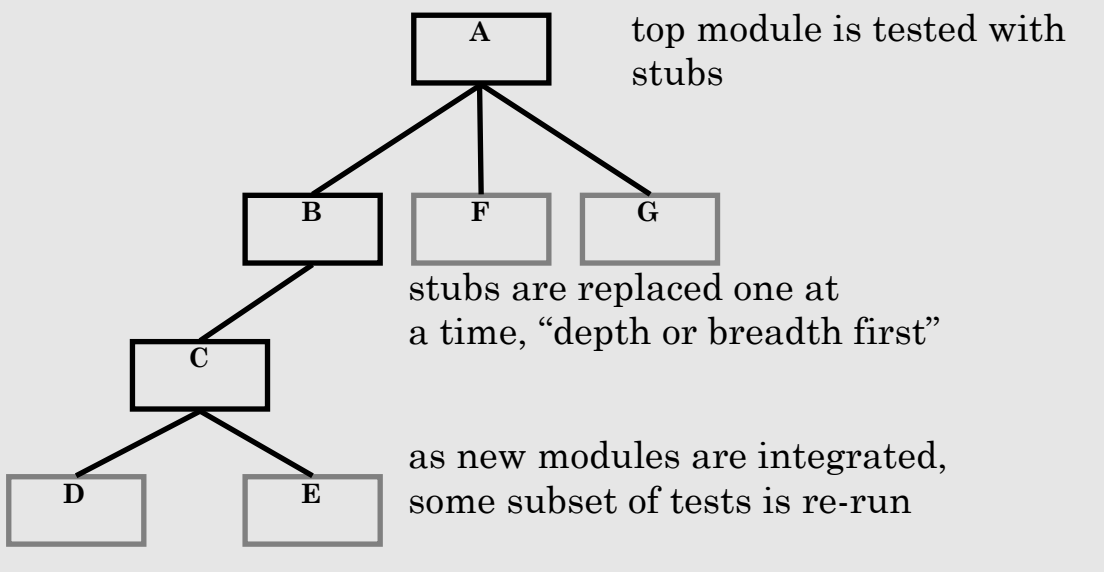
V Model


Level of Detail
Low
High
Requirements Elicitation
Analysis
Design
Object Design
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Project Time

Unit Testing Environment



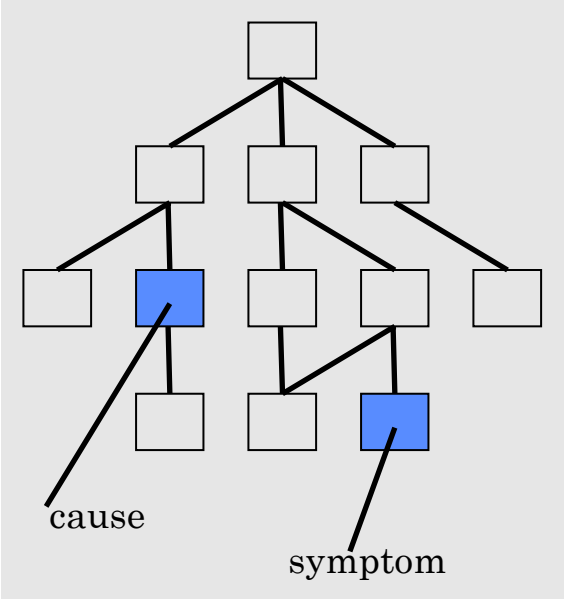
Top Down Integration



High-Order Testing

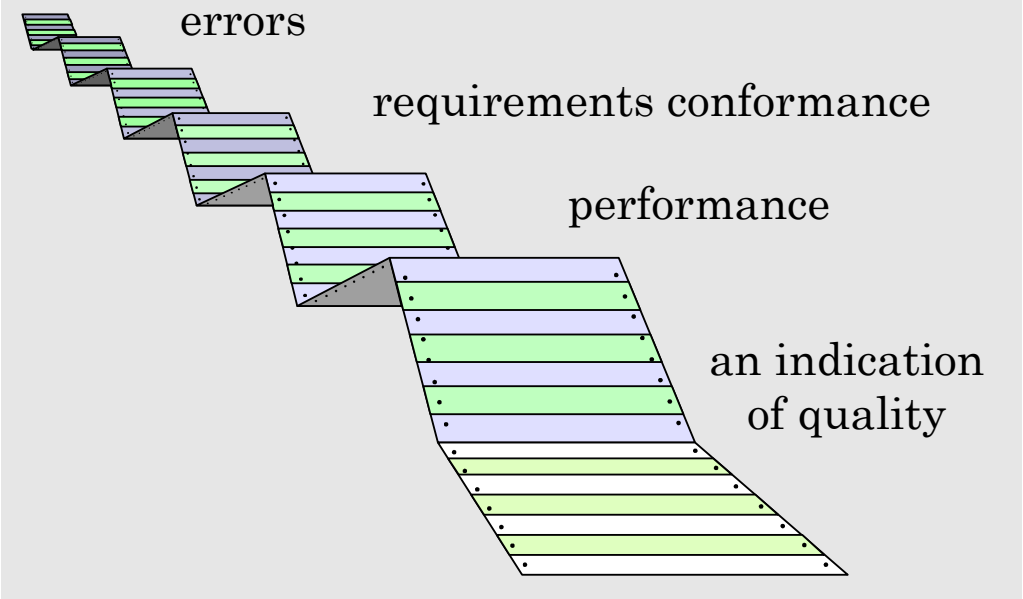
Validation test
System test
Alpha and beta test
Other specialized testing

Debugging: Symptoms & Causes



- symptom and cause may be geographically separated
- symptom may disappear when another problem is fixed
- cause may be due to a combination of non-errors
- cause may be due to a system or compiler error
- cause may be due to assump–tions that everyone believes
- symptom may be intermittent

What Testing Shows



Who Tests the Software?

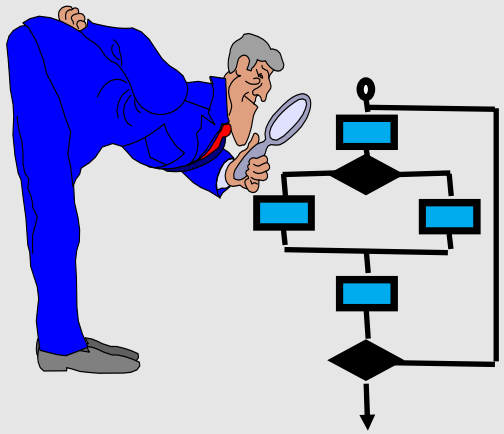
developer

Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

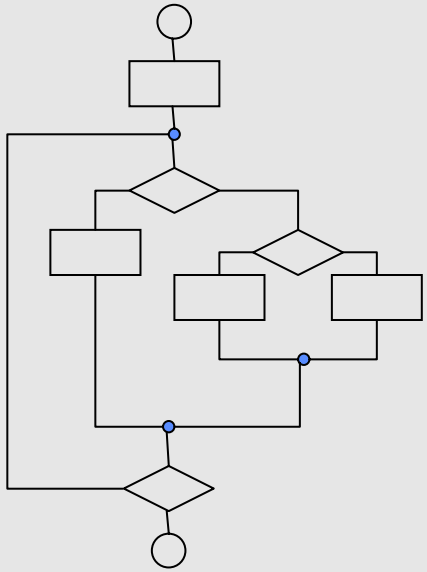
Must learn about the system,
but, will attempt to break it
and, is driven by quality

White-Box Testing



... our goal is to ensure that all statements and conditions have been executed at least once ...

Basis Path Testing



First, we compute the cyclomatic complexity:

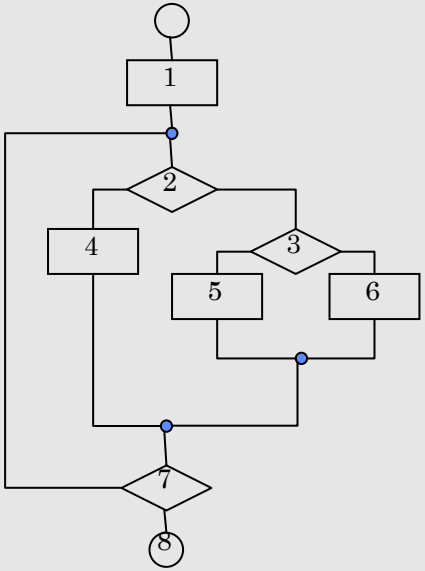
number of simple decisions + 1

or

number of enclosed areas + 1

In this case, V(G) = 4

Basis Path Testing



Next, we derive the independent paths:

Since V(G) = 4, there are up to four paths

Path 1: 1,2,3,6,7,8

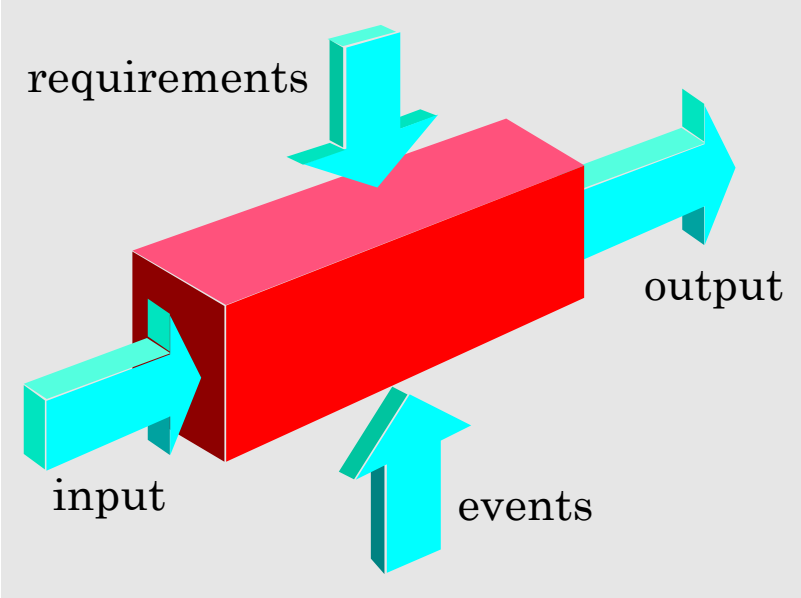
Path 2: 1,2,3,5,7,8

Path 3: 1,2,4,7,8

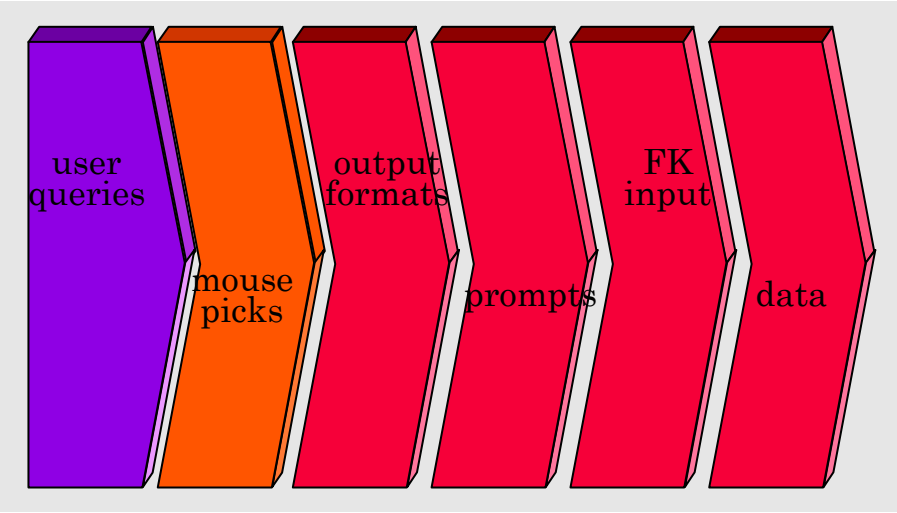
Path 4: 1,2,4,7,2,4, …7,8

Finally, we derive test cases to exercise these paths.

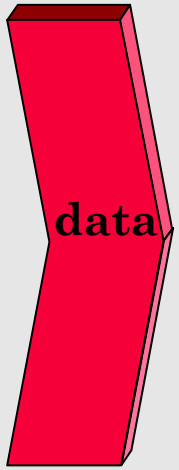
Black-Box Testing



Equivalence Partitioning



Sample Equivalence Classes



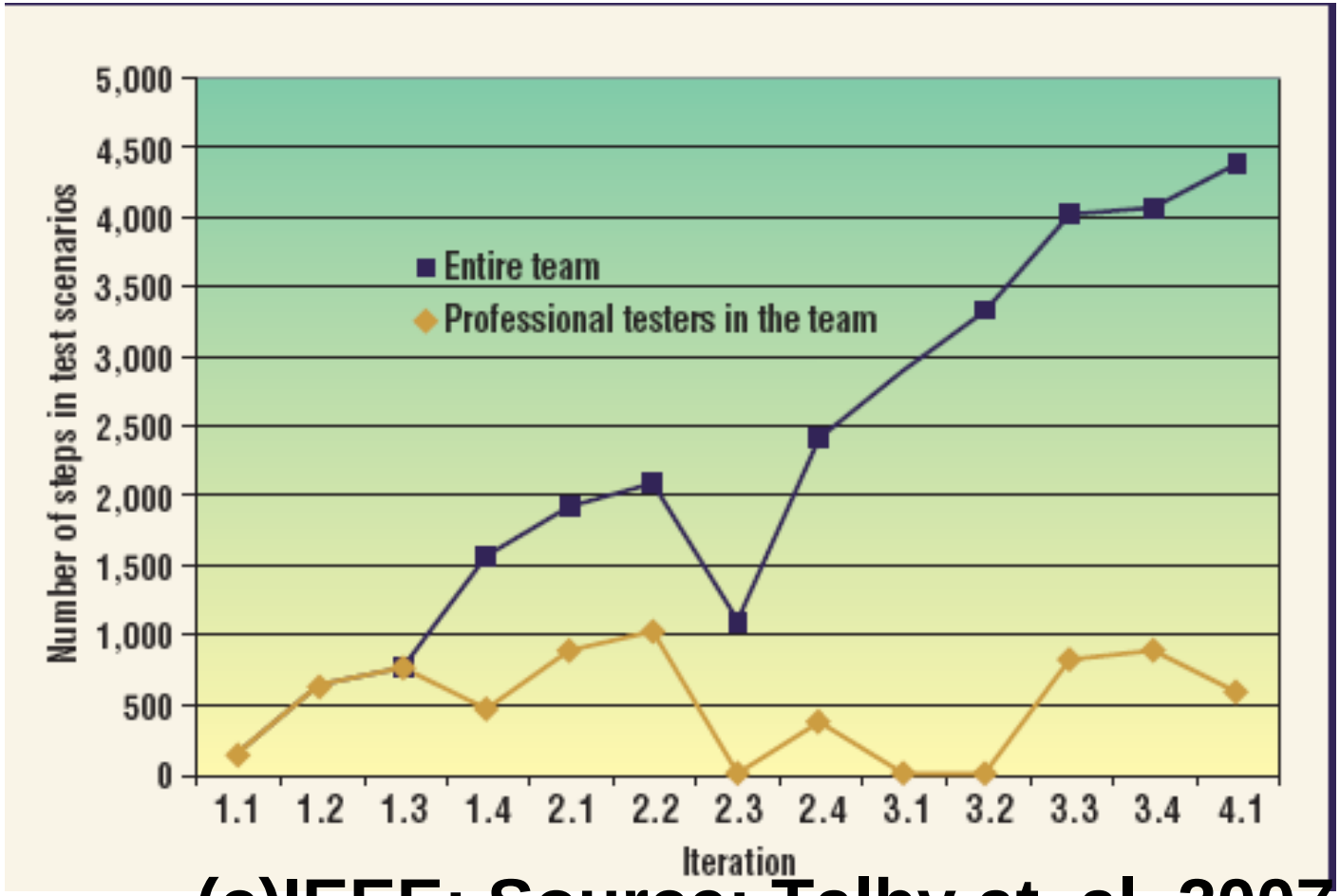
Valid data

- user supplied commands
- responses to system prompts
- file names
- computational data
  - physical parameters
    - bounding values
    - initiation values
- output data formatting
- responses to error messages
- graphical data (e.g., mouse picks)

Invalid data

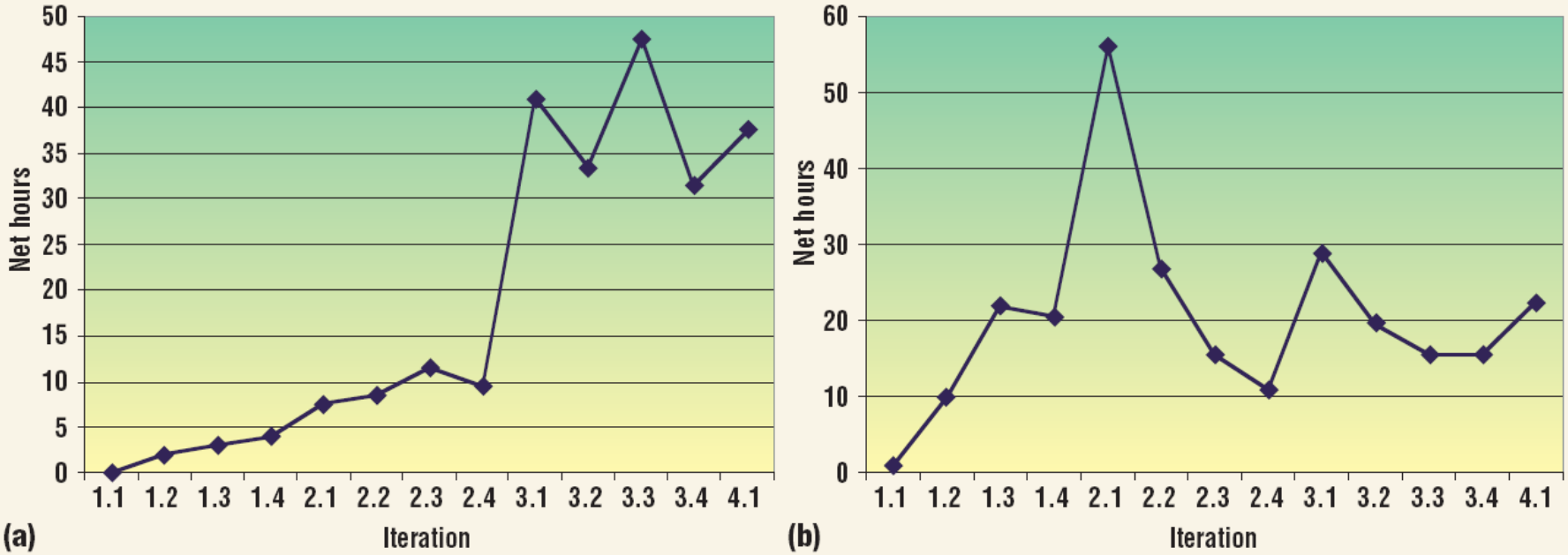
- data outside bounds of the program
- physically impossible data
- proper value supplied in wrong place

Product Size per Iteration



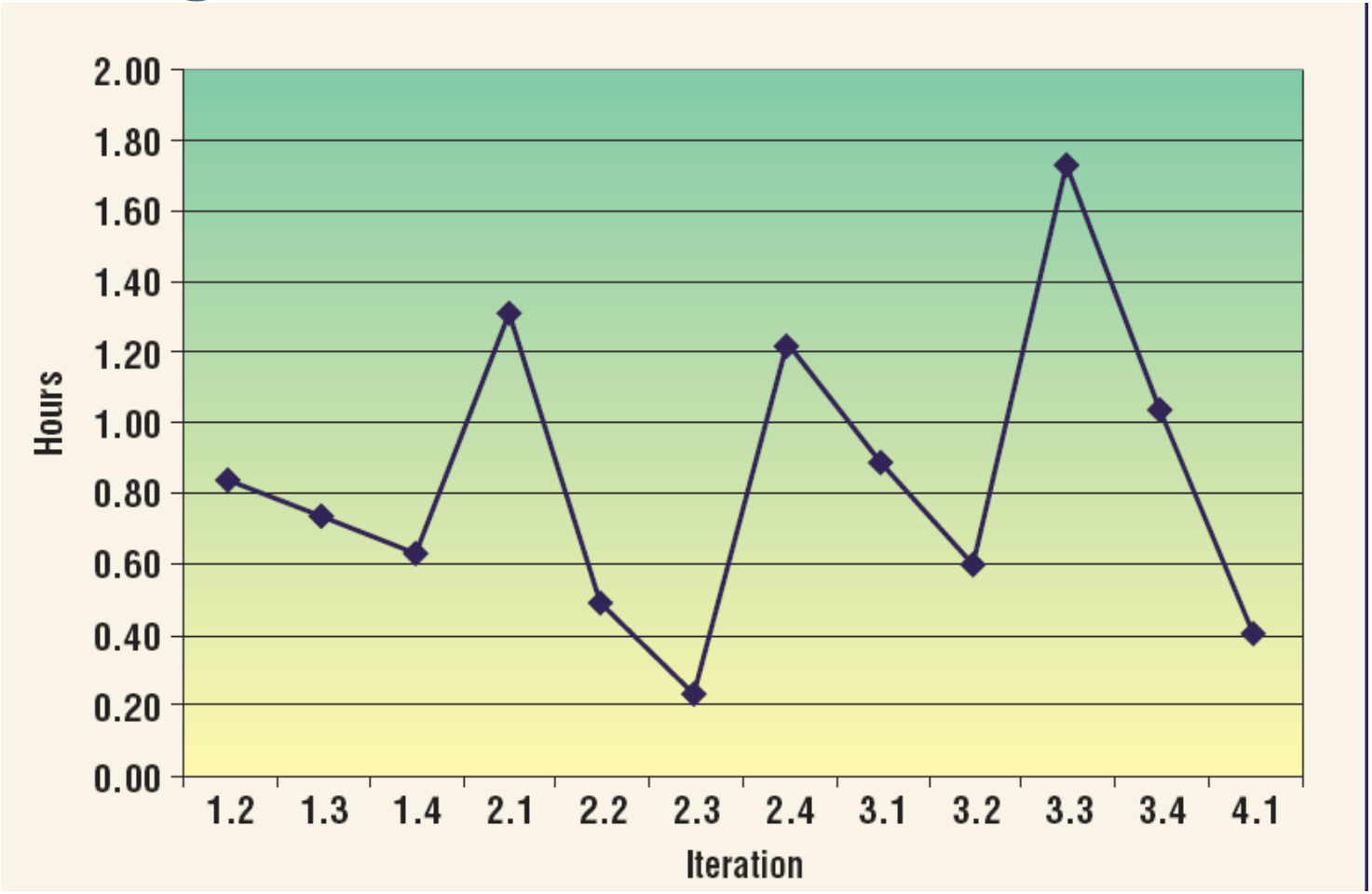
(c)IEEE; Source: Talby at. al. 2007

Testing and Defect Repair



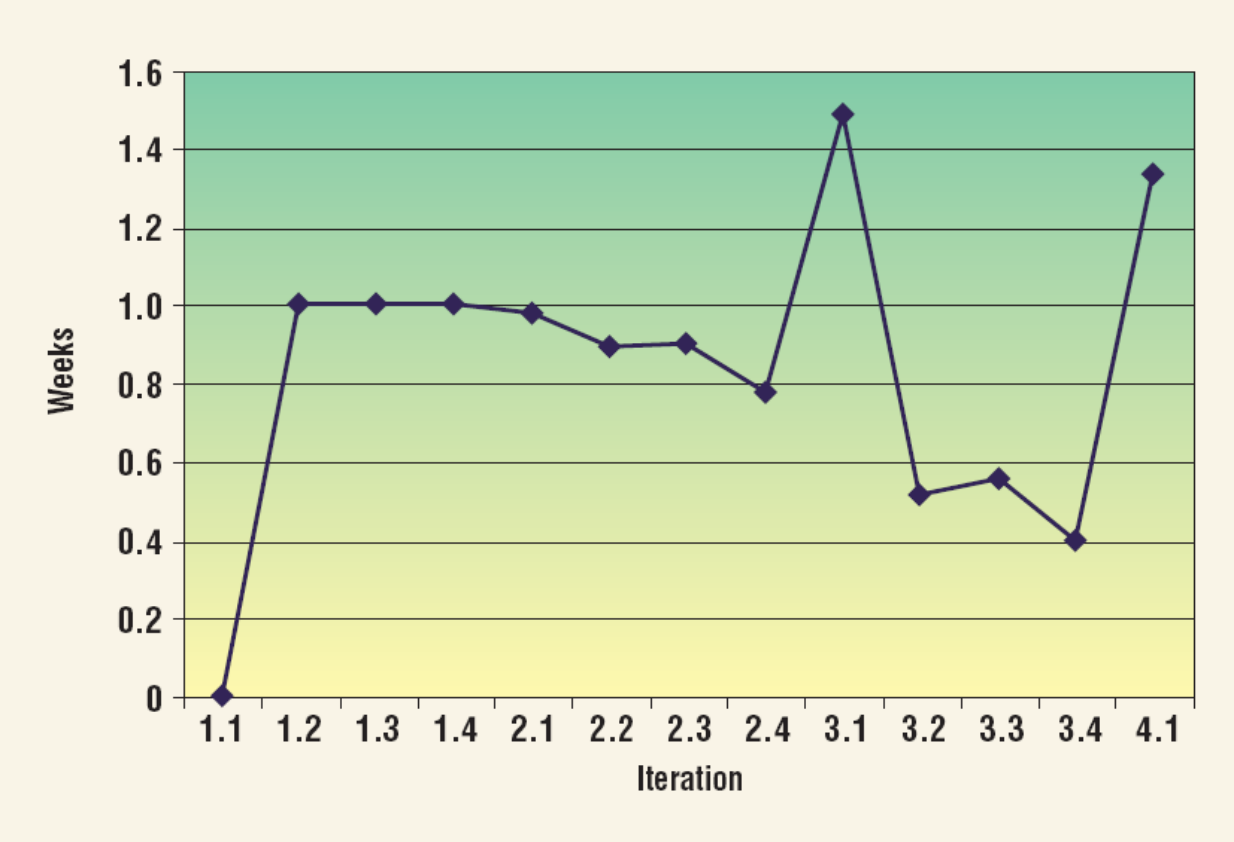
(c)IEEE; Source: Talby at. al. 2007

Average Net Time to Fix a Defect



(c)IEEE; Source: Talby at. al. 2007

A Defect Average Longevity



(c)IEEE; Source: Talby at. al. 2007

XP II

Larman Ch. 8

The 12 XP Practices

Planning

1. Planning Game
2. Small Releases
3. On-site Customer
4. Sustainable Pace

Coding

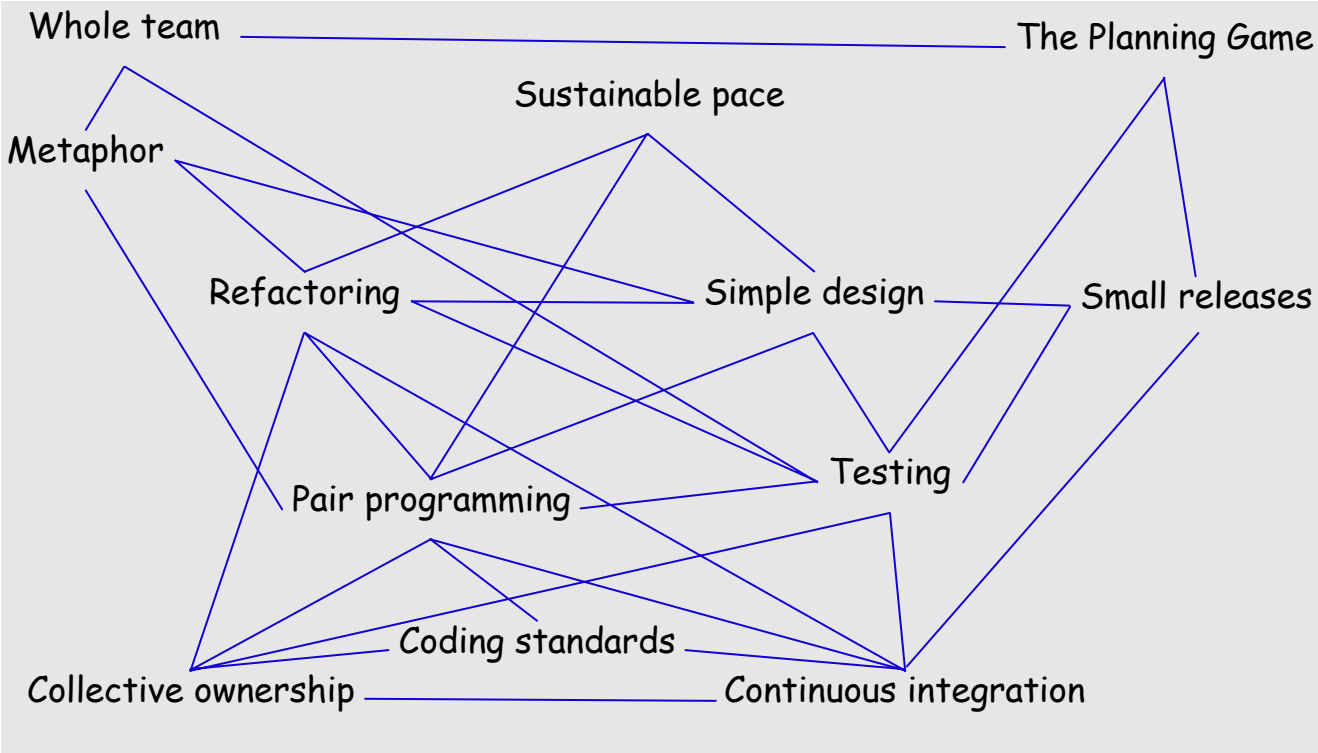
5. Pair Programming
6. Continuous Integration
7. Collective Code Ownership
8. Coding Standards

Designing

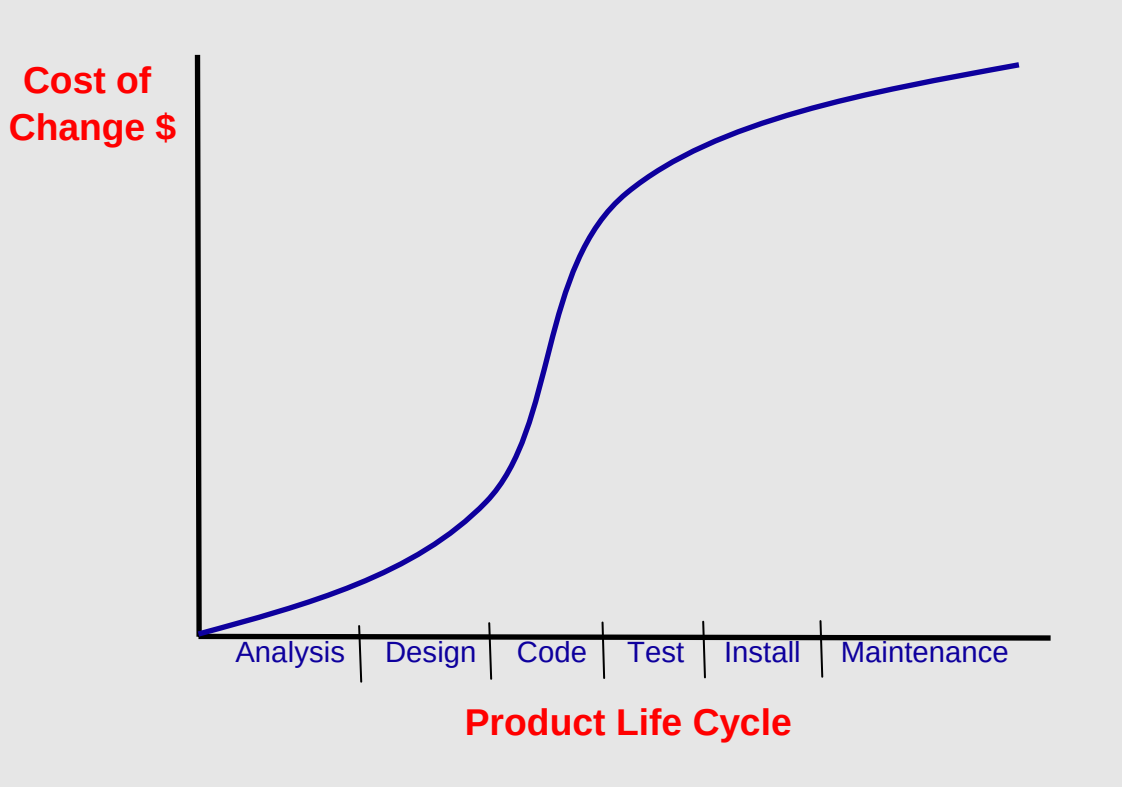
9. Simple Design
10. Test Driven Design
11. Refactoring
12. System Metaphor

Quality Assurance

Supporting each other

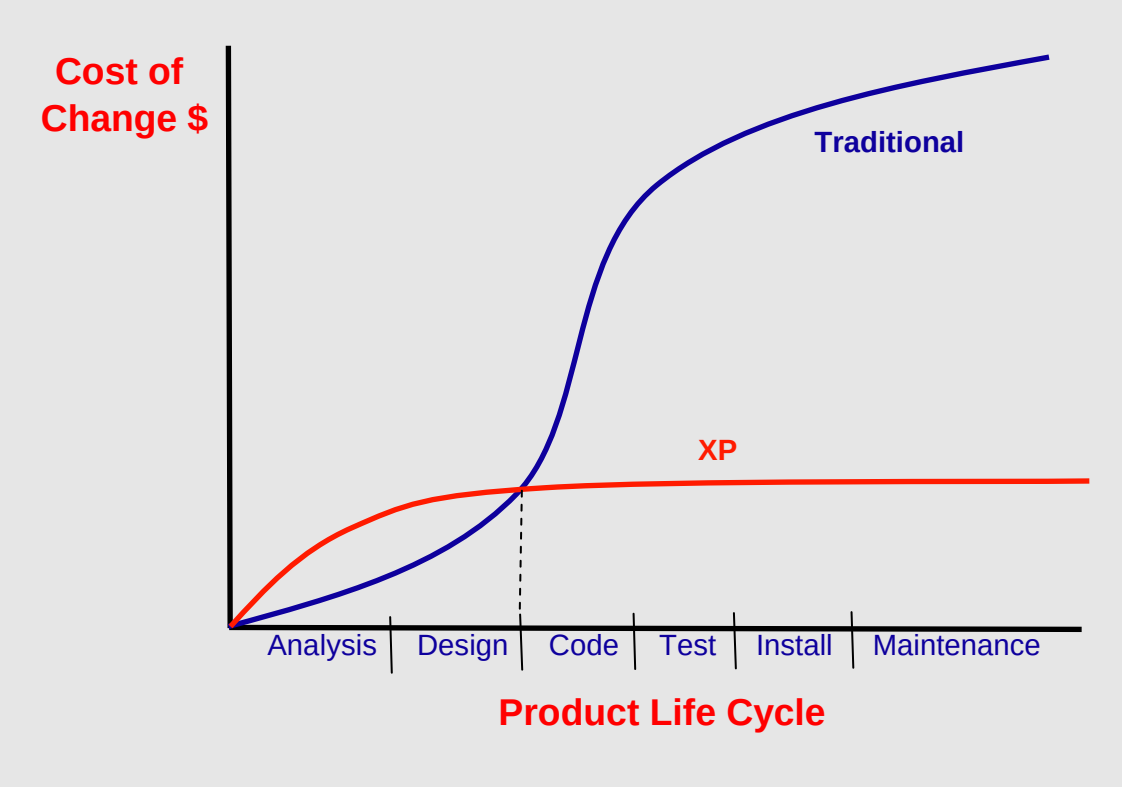


Traditional Cost of Change



Source: Brad Jensen, Sabre Airline Solutions

Lowering the Cost of Change



Source: Brad Jensen, Sabre Airline Solutions

On-site Customer

The "XP Customer" is a business expert who is empowered to decide product features.

At Sabre, this is a subject matter expert in Product Management.

XP Customer duties:

Determine product features
Prioritize the features
Write (or approve) a story for each feature
Write (or approve) acceptance criteria
Write (or approve) functional acceptance tests
"Sign off" on completion of features
Be available at all times to answer programmer's questions

The XP Customer controls scope, and scope is very flexible.

Source: Brad Jensen, Sabre Airline Solutions

Pair Programming


Source: Sabre Airline Solutions

"Test-First" and "Simple Design"

Traditional Waterfall Method:

Requirements ⟹ Design ⟹ Code ⟹ Test

XP Method:

Requirements ⟹ Test ⟹ Design ⟹ Code

You must write automated tests before you write code!

You must run the tests and prove they fail before you write code!

If it is hard to write the test, then the design is poor!

Test First enforces **Simple Design** principles and enables **Refactoring**. It results in high quality code that is easy to change.

Simple Design

Changes in requirements are likely to supersede general solutions anyway

Refactor constantly (Refactoring yields general solutions as they are required).

Design in XP is not a one-time thing, or an up-front thing, it is an all-the-time thing - Ron Jefferies

Create the best design that can deliver the functionality today.

If you believe that the future is uncertain, and you believe that you can cheaply change your mind, then putting in functionality on speculation is crazy. Put in what you need when you need it. – Kent Beck

Other Coding Practices

Continuous Integration

- Code changes checked in at least daily
- Automated Build
- Continuous Build

Collective Code Ownership

- Any developer can change any line of code at any time
- No one “owns” any subsets

Other Coding Practices

Coding Standards

- Enables collective ownership and makes code easier to read and refactor

Sustainable Pace

- Does not mean strict 40-hour weeks – work at a pace that prevents fatigue and burn-out

Refactoring and Metaphor

Refactoring

- Refactoring is changing the internal structure of the code without changing functionality
- Examples:
  Remove duplicate code
  Leverage existing code
  Remove unused code
- Refactoring mercilessly requires good unit tests and functional tests that can easily be executed

System Metaphor

- All project team members use a common language to describe the functionality of the system

XP Disciplines and Tools

XP Practice	Supporting Tools
Pair Programming	IDE Version Control
Collective Code Ownership	Version Control
Test Driven Design	Unit Test Framework Functional Test Framework IDE
Simple Design	Metrics tools
Refactoring	IDE Version Control
Continuous Integration	Automated and Continuous Build Version Control IDE
Coding and Design Standards	IDE Metrics Tools