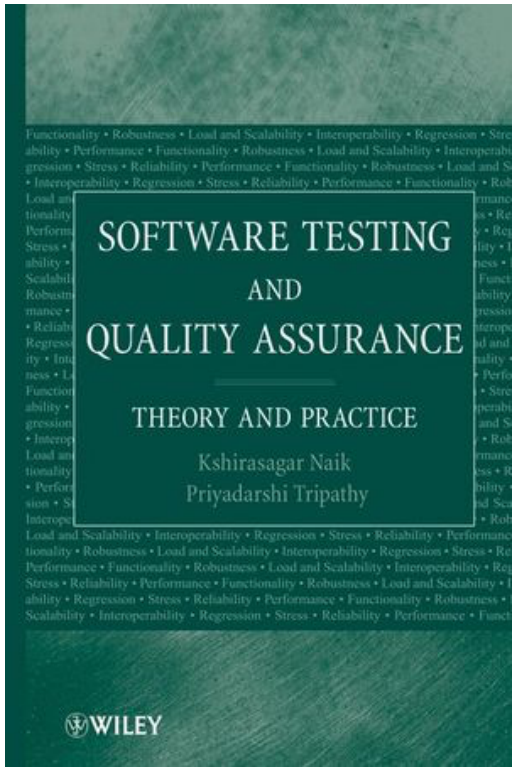
Software Testing

ETS 200

http://cs.lth.se/ets200



Naik 7.1-7.4, 12.1-12.9

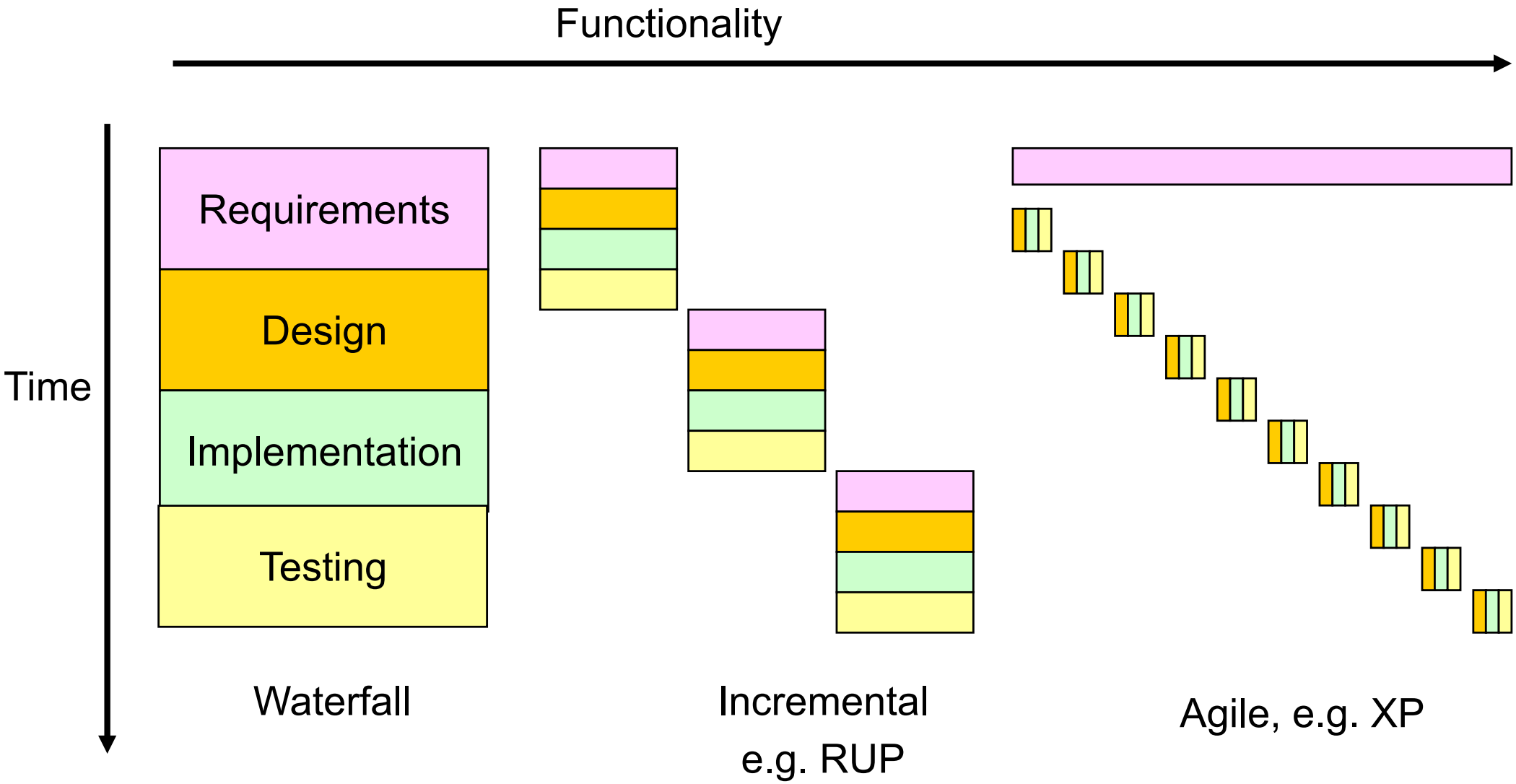
Lifecycle, Documentation

Prof. Per Runeson

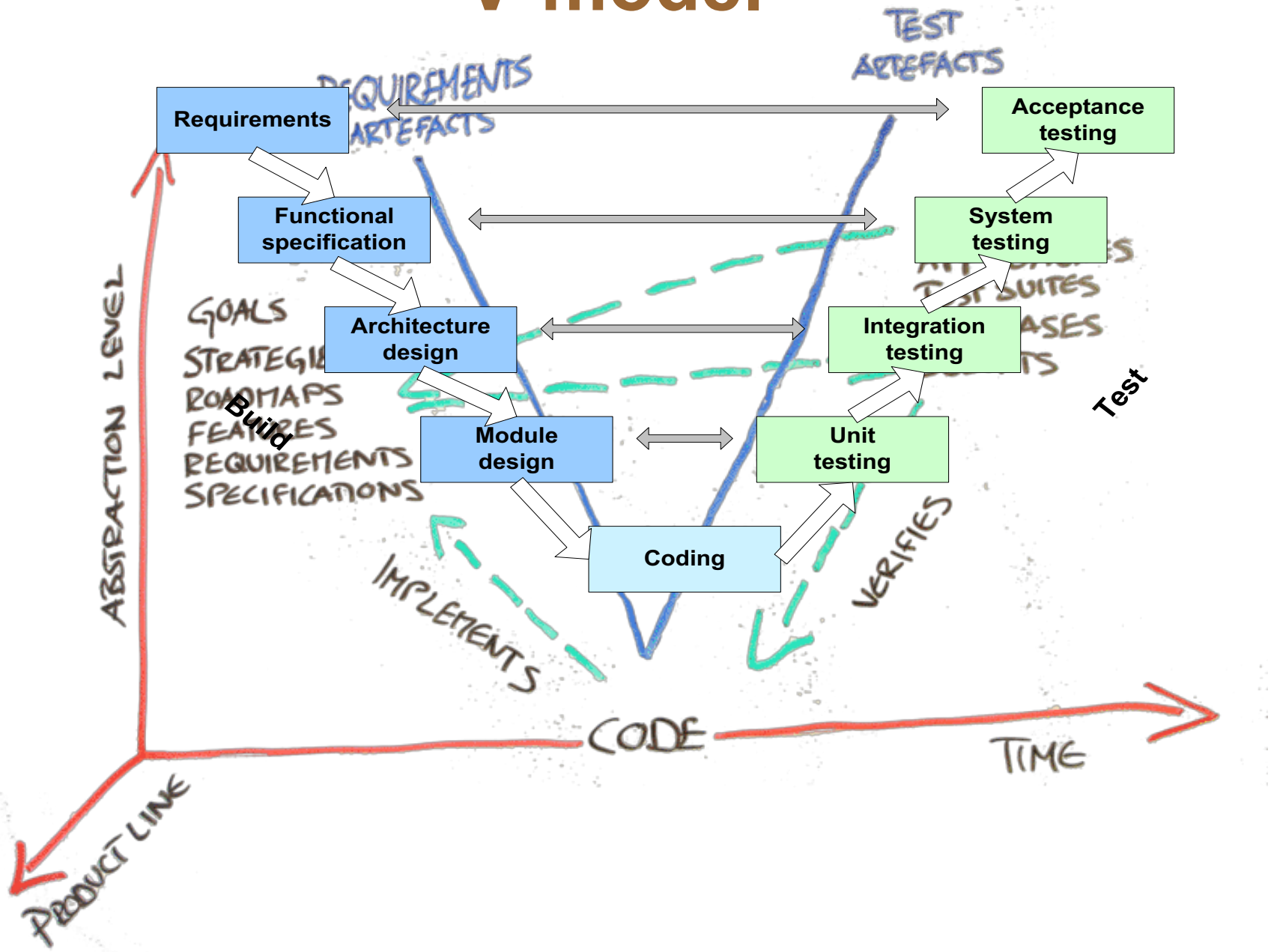
Lecture

- System integration testing
  - Ch 7.1-7.4
- Documentation
  - Ch 12.1-12.9

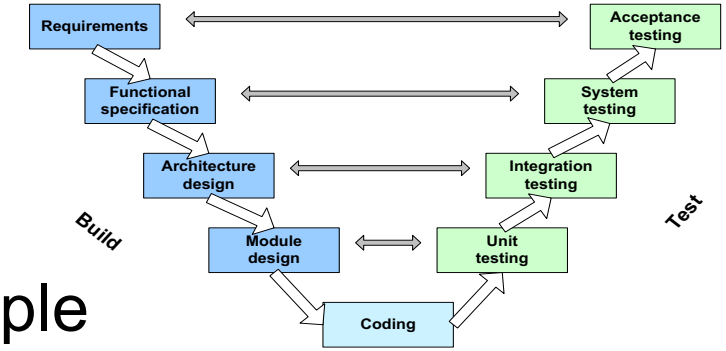
Process models from waterfall to agile



V-model

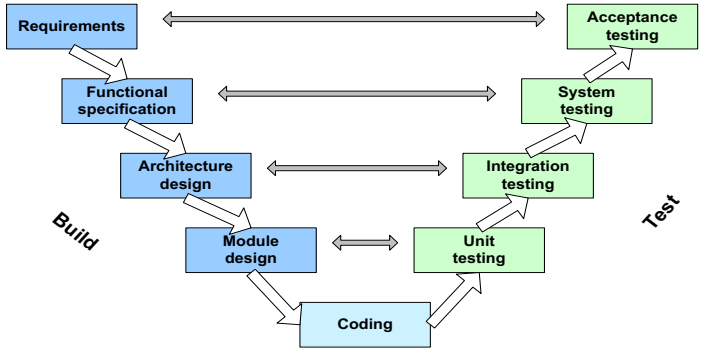

Requirements
Functional specification
Architecture design
Module design
Coding
Unit testing
Integration testing
System testing
Acceptance testing
TEST ARTEFACTS
GOALS
STRATEGIES
ROADMAPS
FEATURES
REQUIREMENTS
SPECIFICATIONS
Build
Test
IMPLEMENTS
VERIFIES
ABSTRACTION LEVEL
CODE
TIME
PRODUCT LINE

Benefits of the V-model



- Intuitive and easy to explain
  - Makes a good model for training people
  - Shows how testing is related to other phases of the waterfall process
- Quite adaptable to various situations
  - If the team is experienced and understands the inherent limitations of the model
- Beats the code-and-fix approach on any larger project

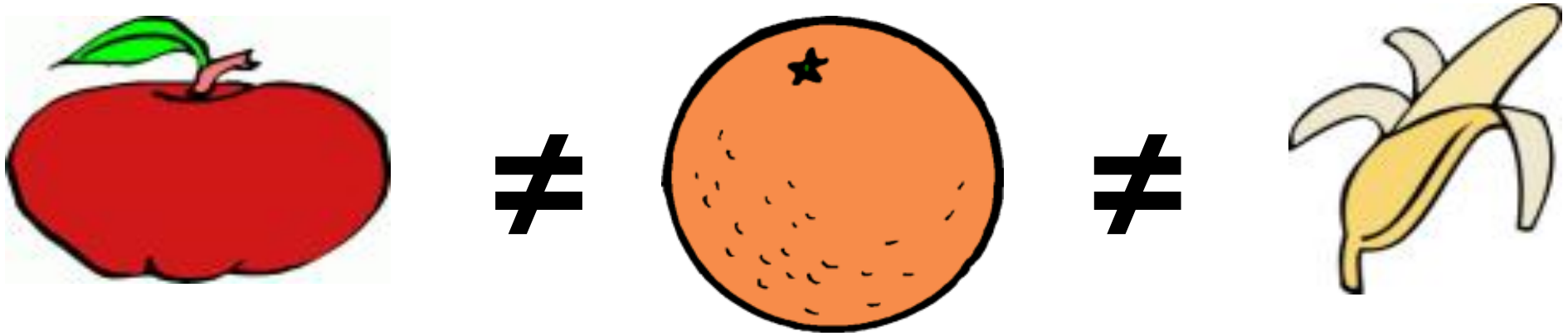
Weaknesses in V-model

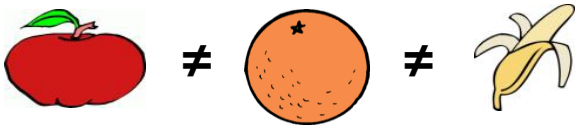


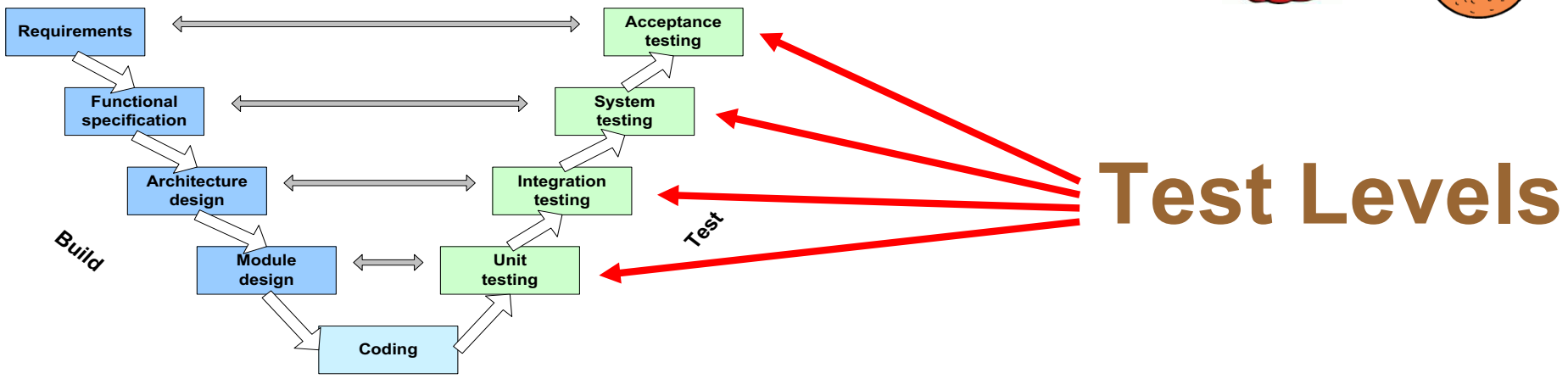
- Hard to follow in practice
- Document driven
  - Relies on the existence, accuracy, and timeliness of documentation
  - Asserts testing on each level is designed based on the deliverables of a single design phase
- Communicates change poorly
  - Does not show how changes, fixes, and test rounds are handled
- Testing windows get squeezed
- Based on simplistic waterfall model
- Does not fit into iterative development

(Brian Marick 2000;
James Christie 2008)

Levels, Types, and Phases – not the same thing

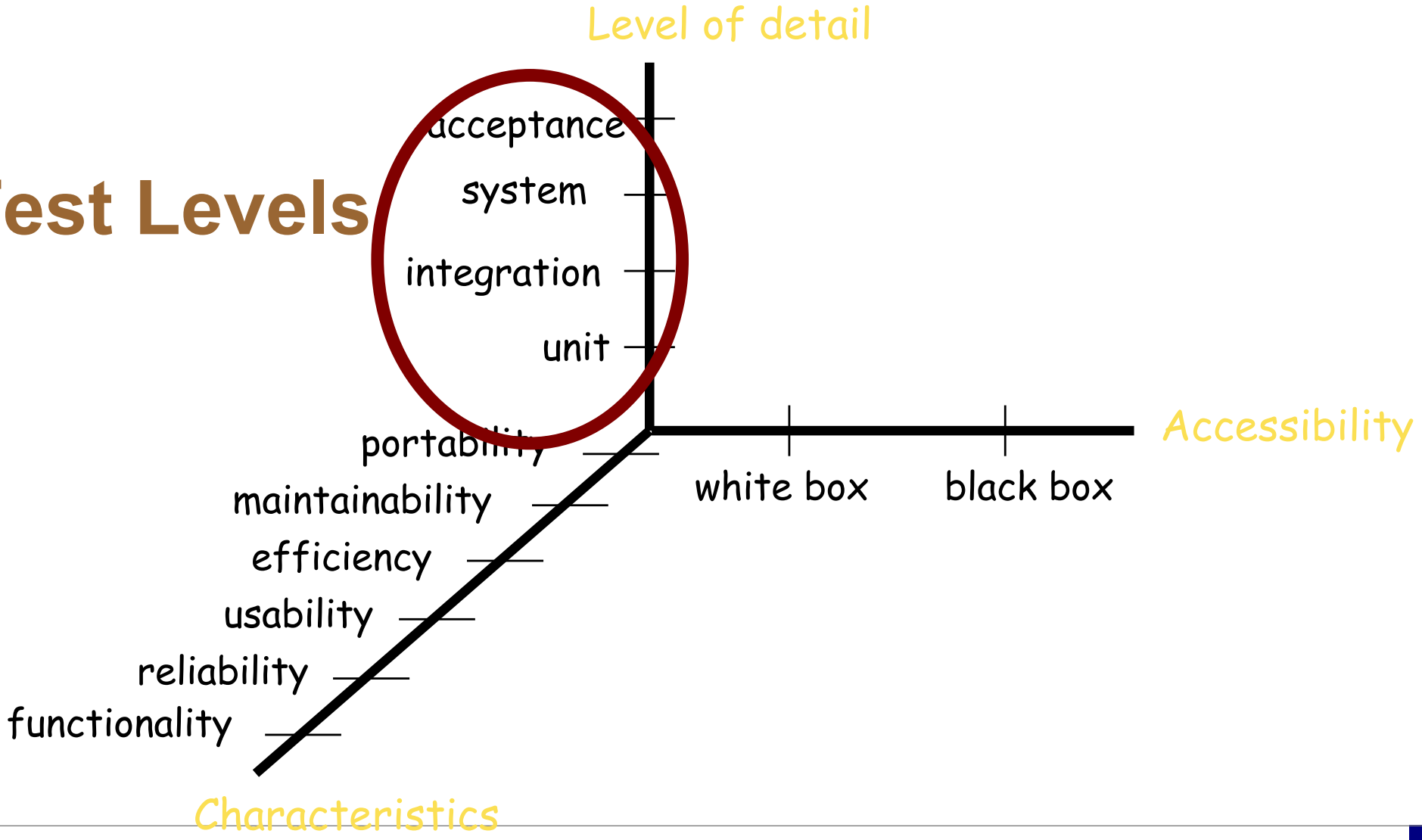




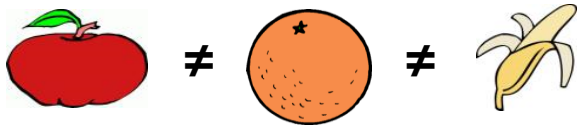


- **Test level** – a test level is a group of test activities that focus into certain level of the test target
  - Test levels can be seen as levels of detail and abstraction
  - How big part of the system are we testing?

Test Levels

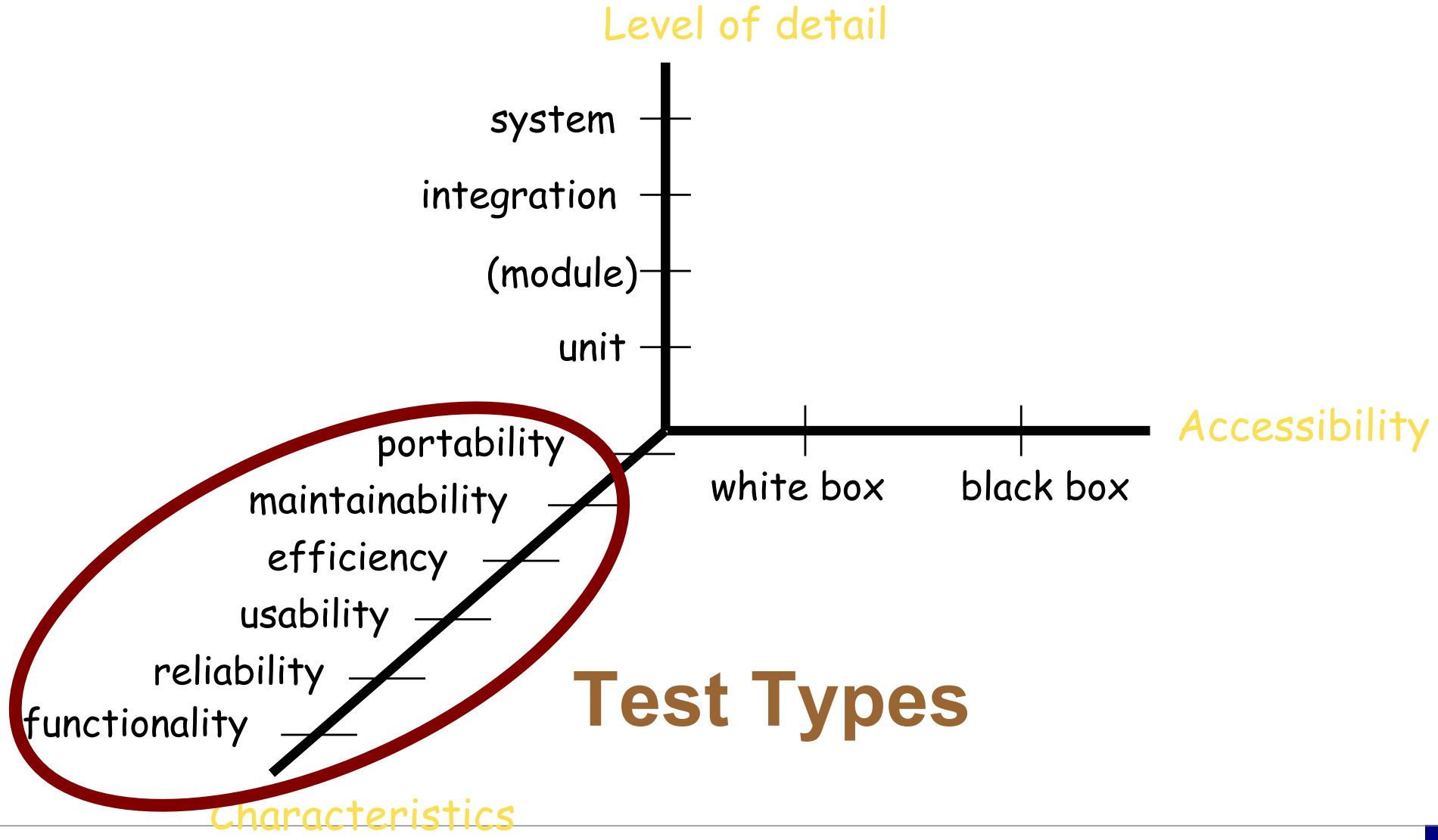

Level of detail
acceptance
system
integration
unit
Accessibility
white box
black box
portability
maintainability
efficiency
usability
reliability
functionality
Characteristics

Test Types

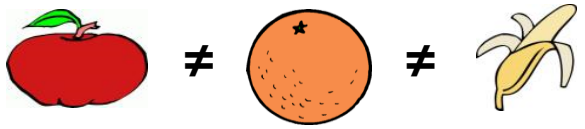


- **Test type** – evaluate a system for a number of associated quality goals
  - Testing selected quality characteristics
  - Testing to reveal certain types of problems
  - Testing to verify/validate selected types of functionality or parts of the system
- Common test types
  - Functional: Installation, Smoke, Concurrency testing
  - Non-Functional: Security, Usability, Performance, …
- A certain test type can be applied on several test levels and in different phases

Test Types

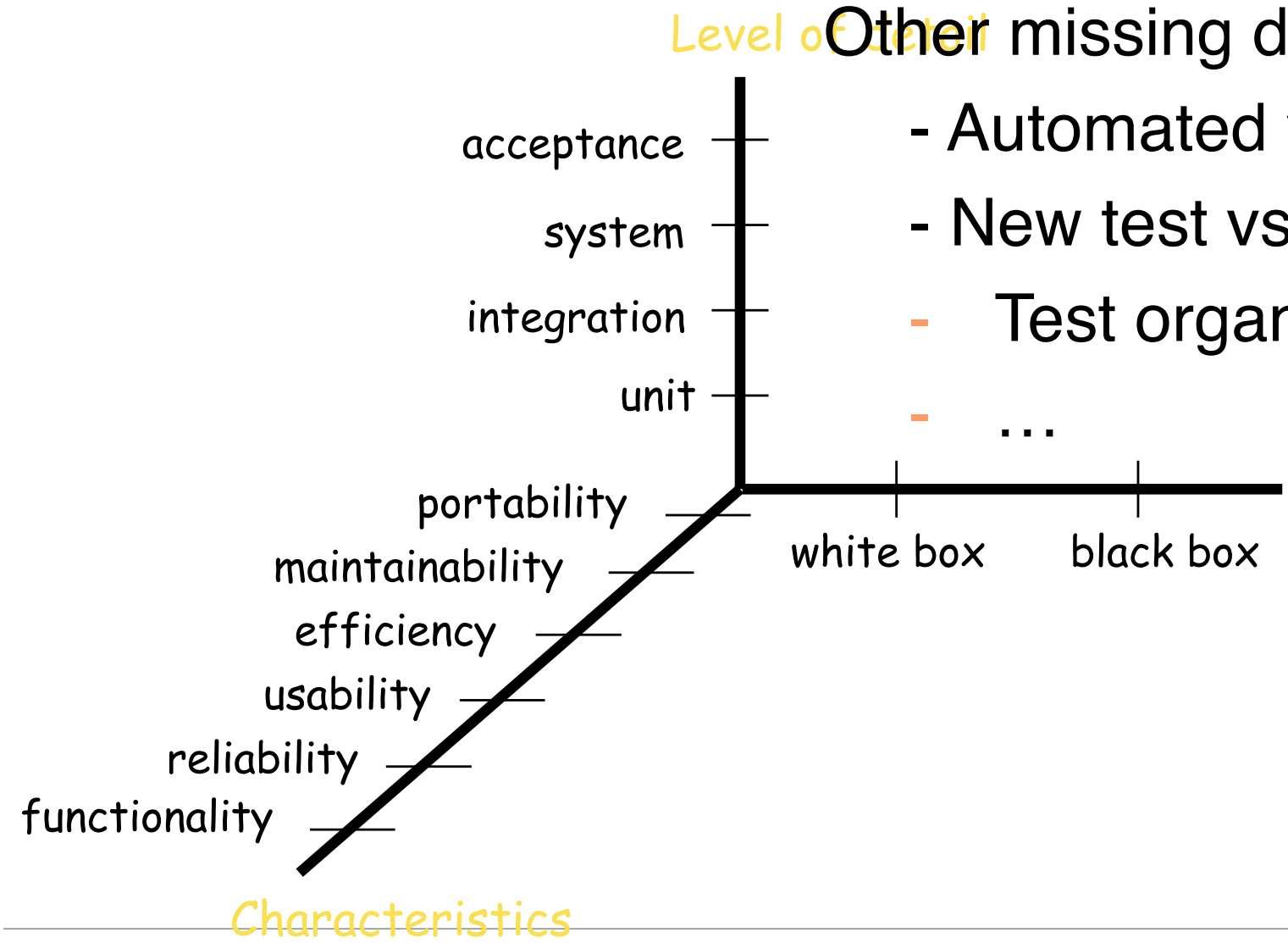

Level of detail
system
integration
(module)
unit
Accessibility
white box
black box
portability
maintainability
efficiency
usability
reliability
functionality
Characteristics

Test Phases



- **Test phase** – Test phases are used to describe temporal parts in testing process
  - Test phases are phases in a development or testing project
- Do not always match to the test levels or types
  - In iterative development the phases may not match the test levels
    - Often, there is unit testing but no unit test phase
    - You can have several integration phases
      - or integration testing without an integration phase
  - Phases might match the test levels
    - as depicted in V-model

Test phase dimension is missing

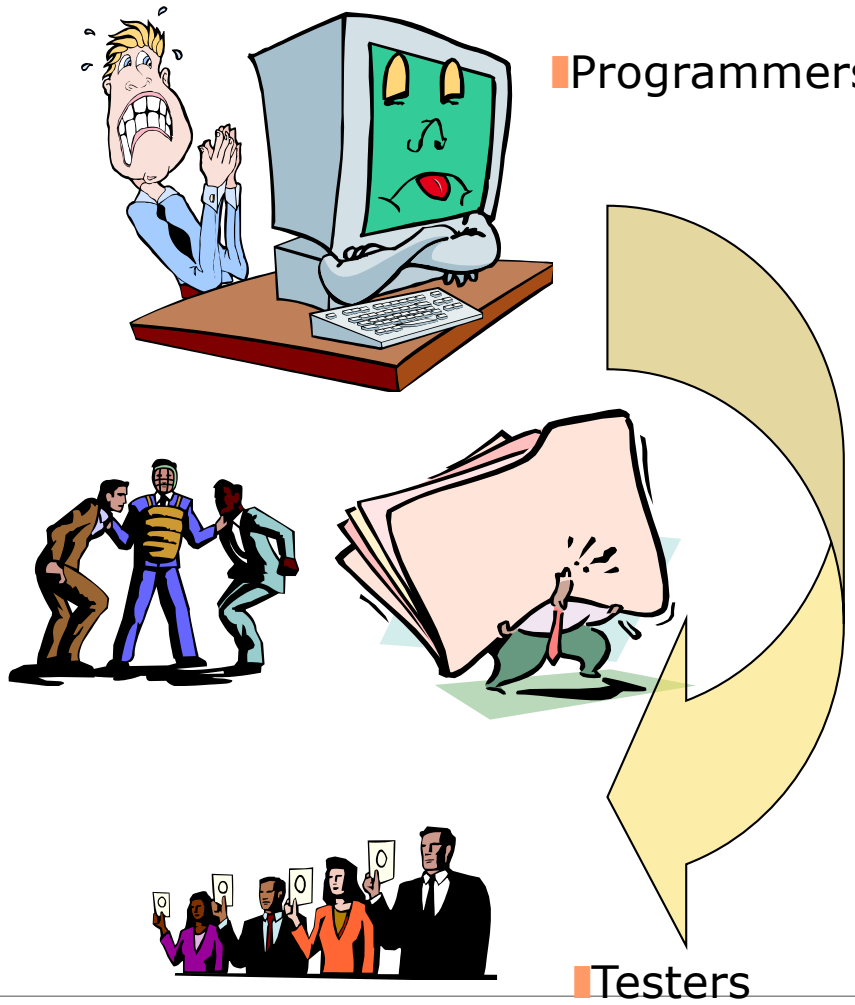


Other missing dimensions

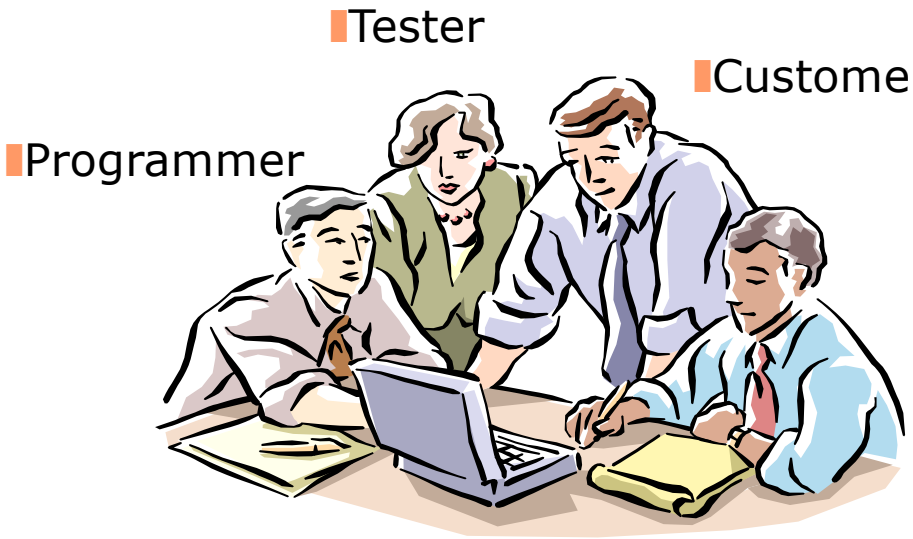
- Automated vs. Manual
- New test vs. regression test
- Test organization
- …

Organizing testing (waterfall vs. agile)

Waterfall model



Agile models

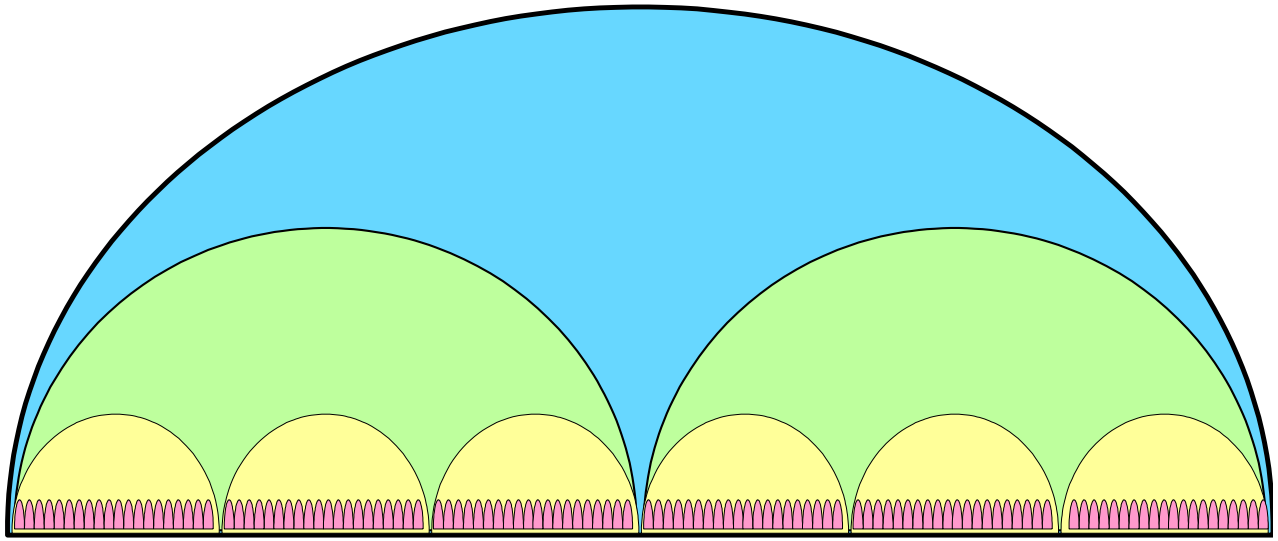


Idea:

Testing in collaboration

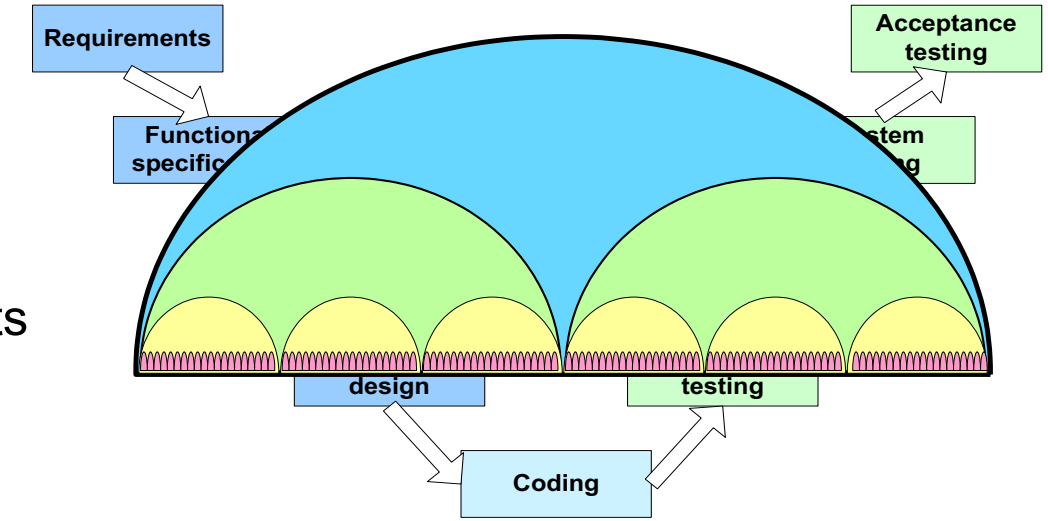
“What is time paced development?” – Time is fixed, scope changes, e.g. Scrum

- 30 days to complete iteration or sprint
- 90 days to complete alpha release
- 90 days to complete beta release
- 180 for whole projects



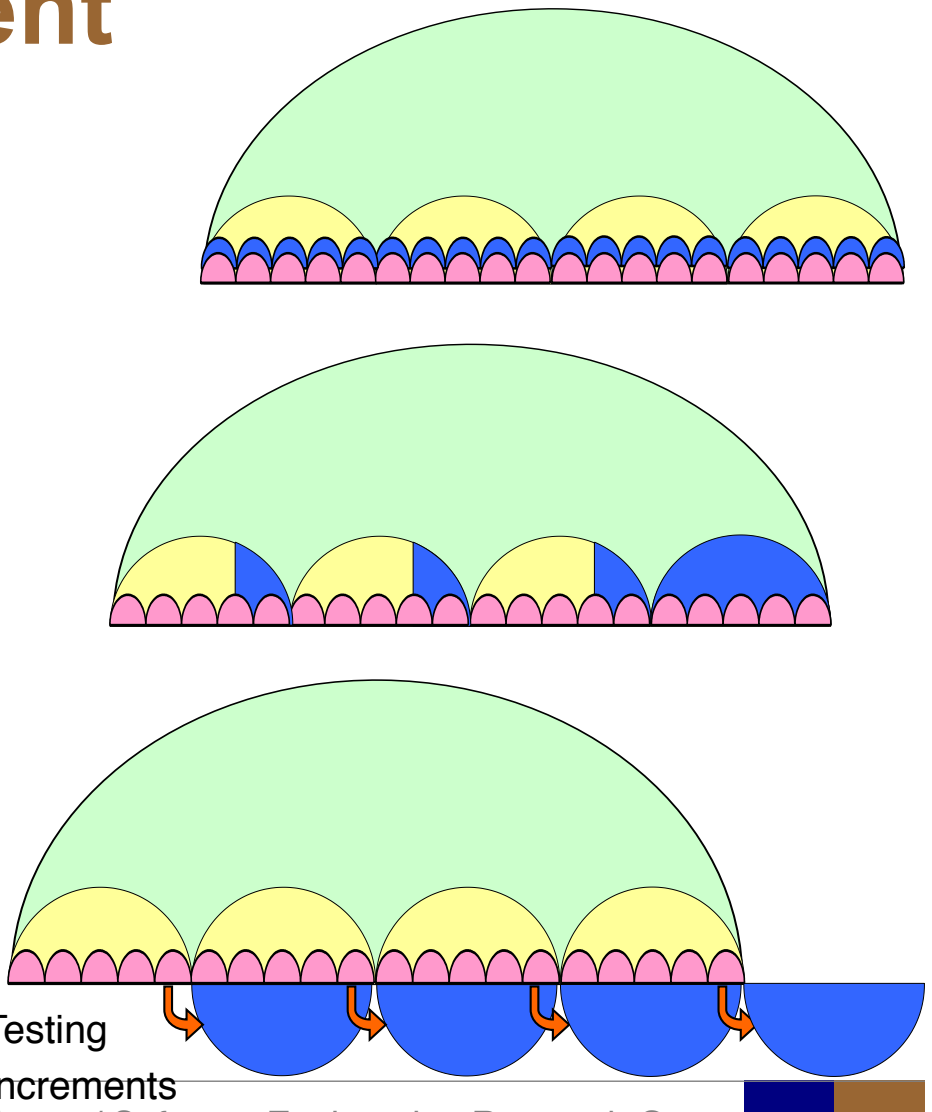
Testing in Time Paced Development

- Part of each development task
- Testing is not a phase
- Software is developed incrementally
  - It has to be tested incrementally
- Rapid, time paced development
  - Time boxed releases and increments
  - Deadlines are not flexible
- Scope is flexible
  - Testing provides information for scoping

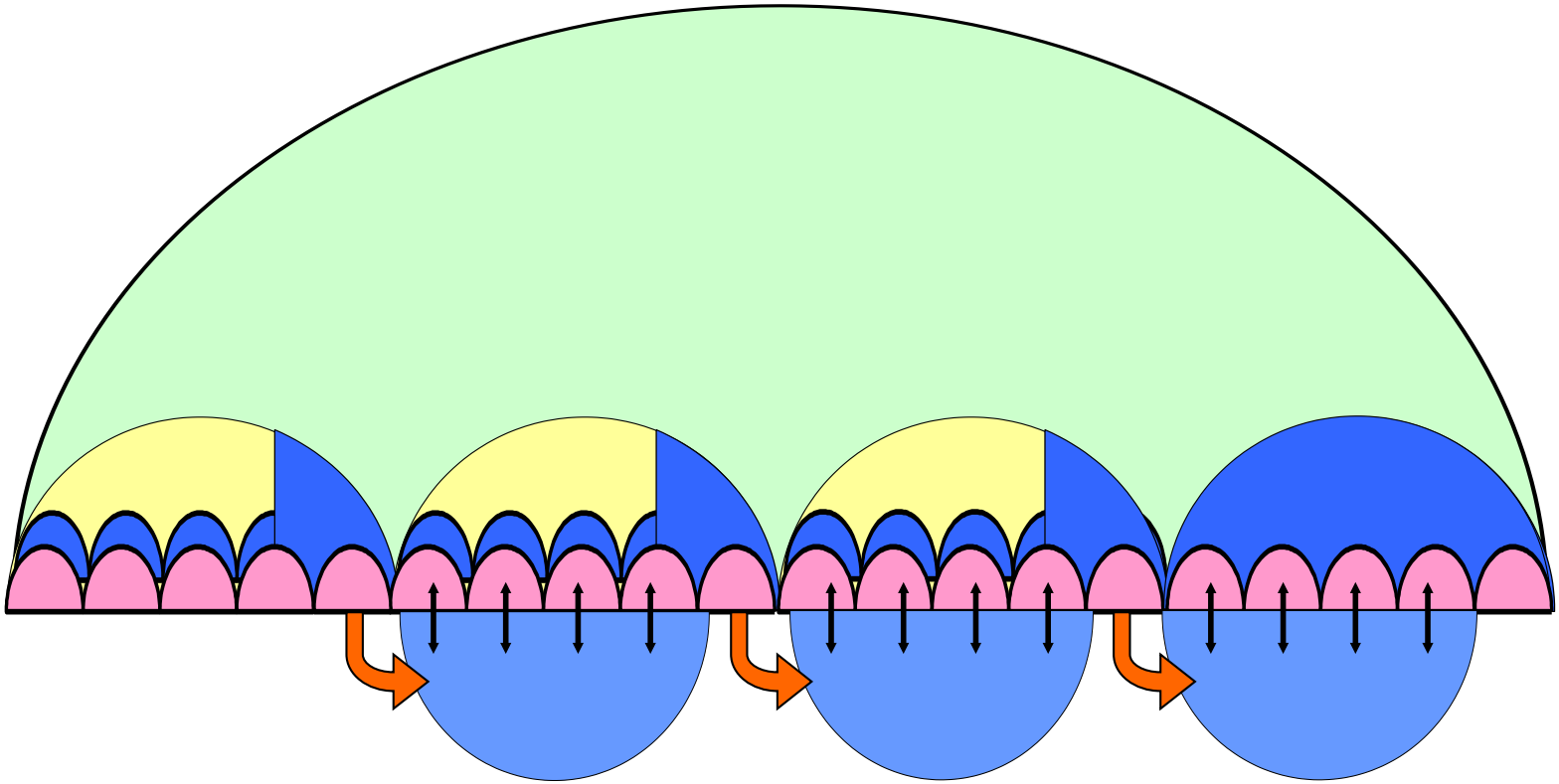


Approaches to Testing in Time Paced Development

- Automated regression testing
  - Automated unit testing
  - Test-driven development
  - Daily builds and automated tests
- Stabilisation phase or increment
  - Feature freeze
  - Testing and debugging at the end of the increment or release
- Separate system testing
  - Independent testing
  - Separate testers or testing team



A Combined Testing Approach



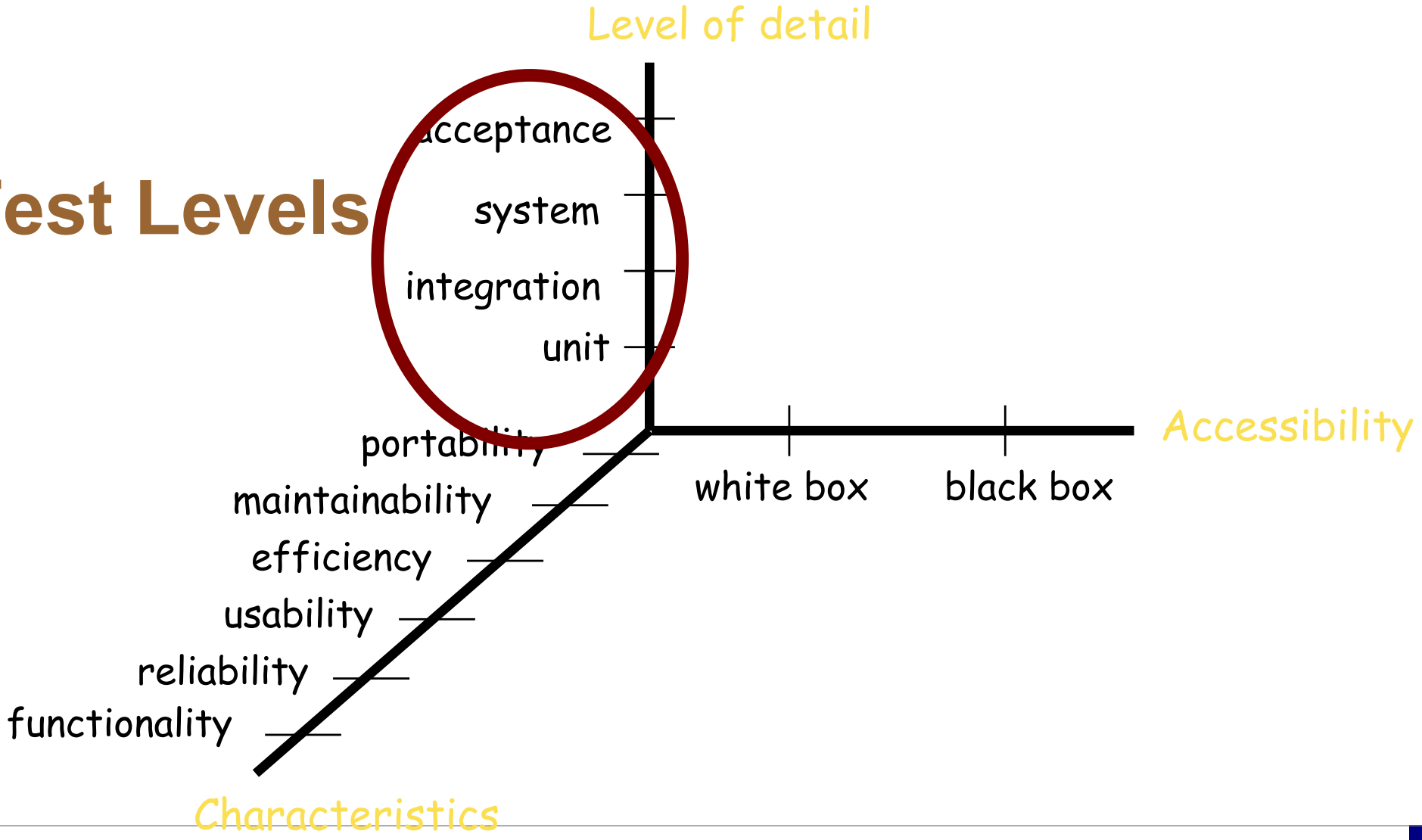
Coping with the Agile Challenges 1(2)

- **Requirements change all the time**
- **Specifications are never final**
  - *Let them change, test design is part of each task*
- **Code is never 'finished', never ready for testing**
- **Not enough time to test**
  - *Focus on developing 'finished' increments, tracking at the task level*
  - *Testing is part of each development task*

Coping with the Agile Challenges 2(2)

- **Need, to regression test everything in each increment**
- **Developers always break things again**
- **How can we trust?**
  - *Trust comes from building in the quality, not from the external testing 'safety net'*
  - *Automation is critical*

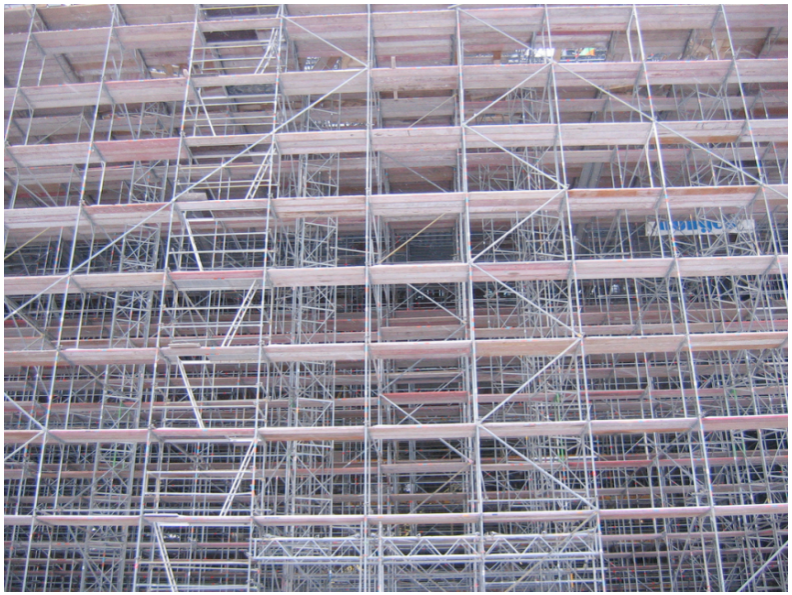
Test Levels


Level of detail
acceptance
system
integration
unit
Accessibility
white box
black box
portability
maintainability
efficiency
usability
reliability
functionality
Characteristics

Unit testing

- Unit = smallest testable software component (e.g. labs 1 and 2)
  - Objects
  - Procedures / functions
  - Reusable components
- Tested in isolation
- Usually done by programmer during development
  - A pair tester can help
  - Developer guide or coding guideline can require
- The book idealizes a heavy weight model for unit testing
  - “The unit should be tested by an independent tester (someone other than the developer) and the test results and defects found should be recorded as a part of the unit history”
- Also known as component or module testing

Scaffolding test harness



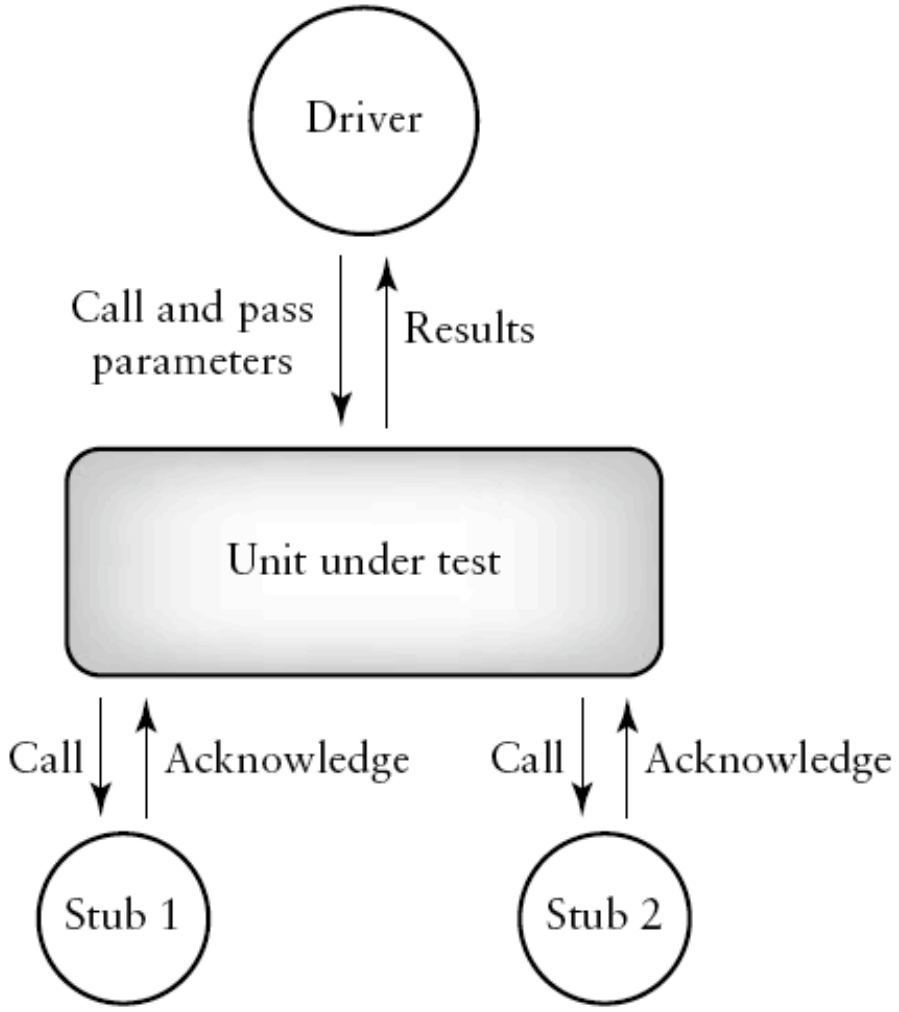
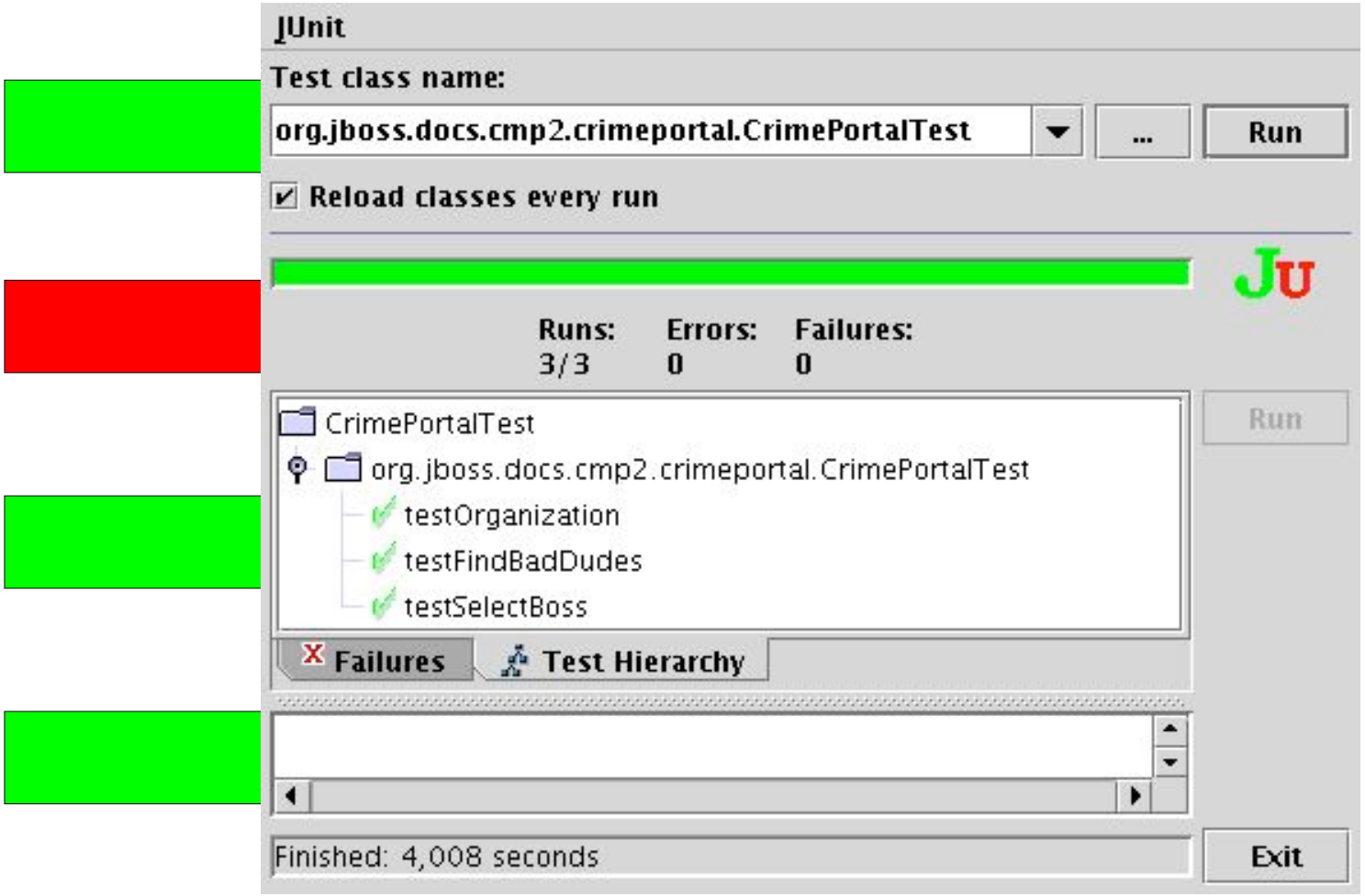
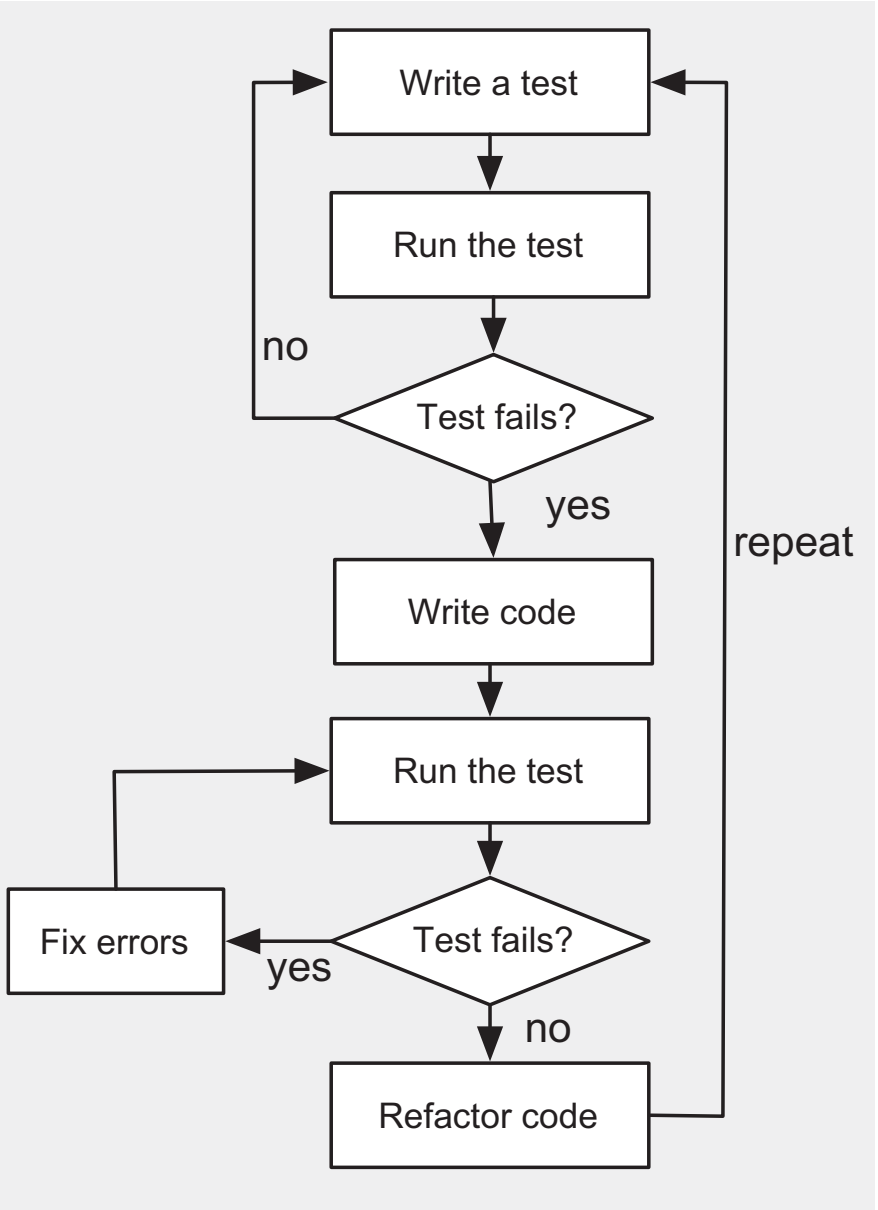


Fig. 6.5

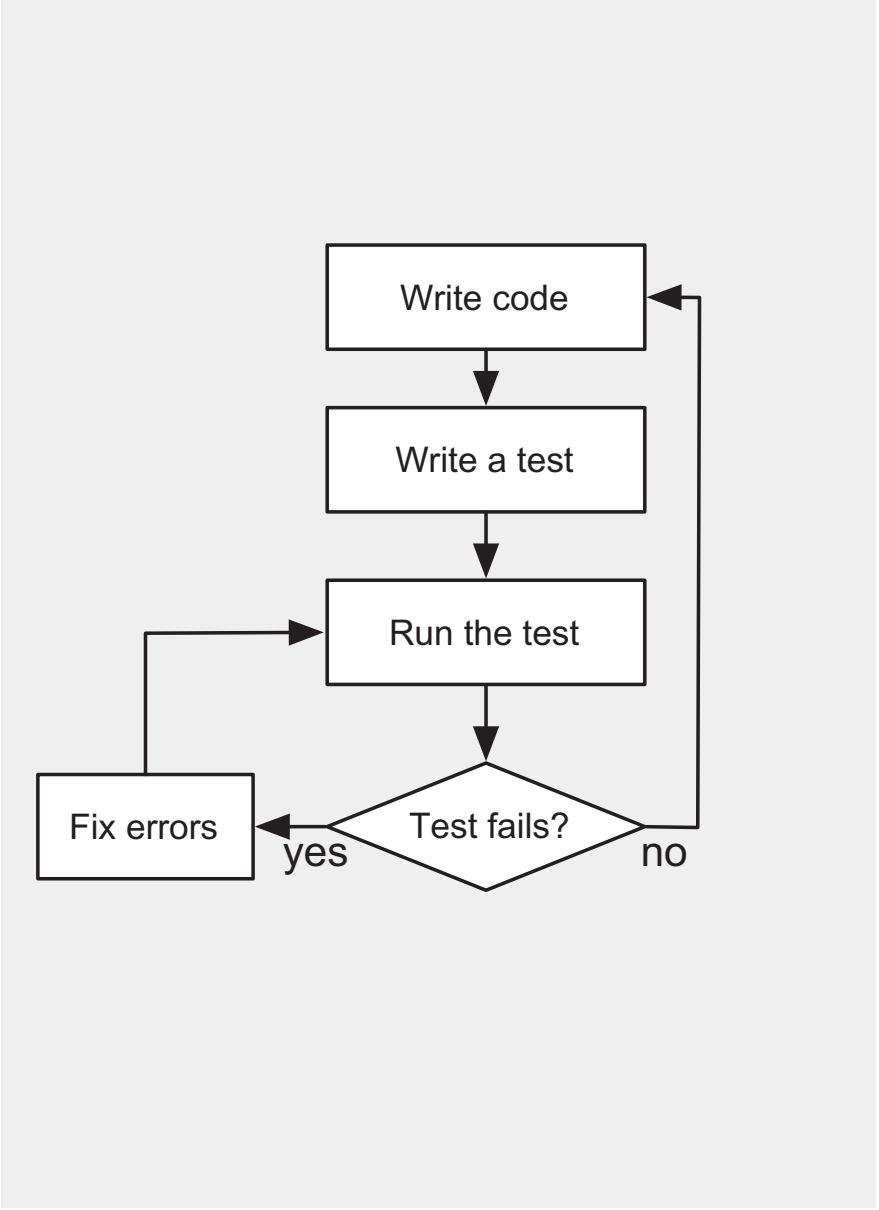
Test-driven development

1. Write a test
2. See it fail
3. Make it run
4. Make it right



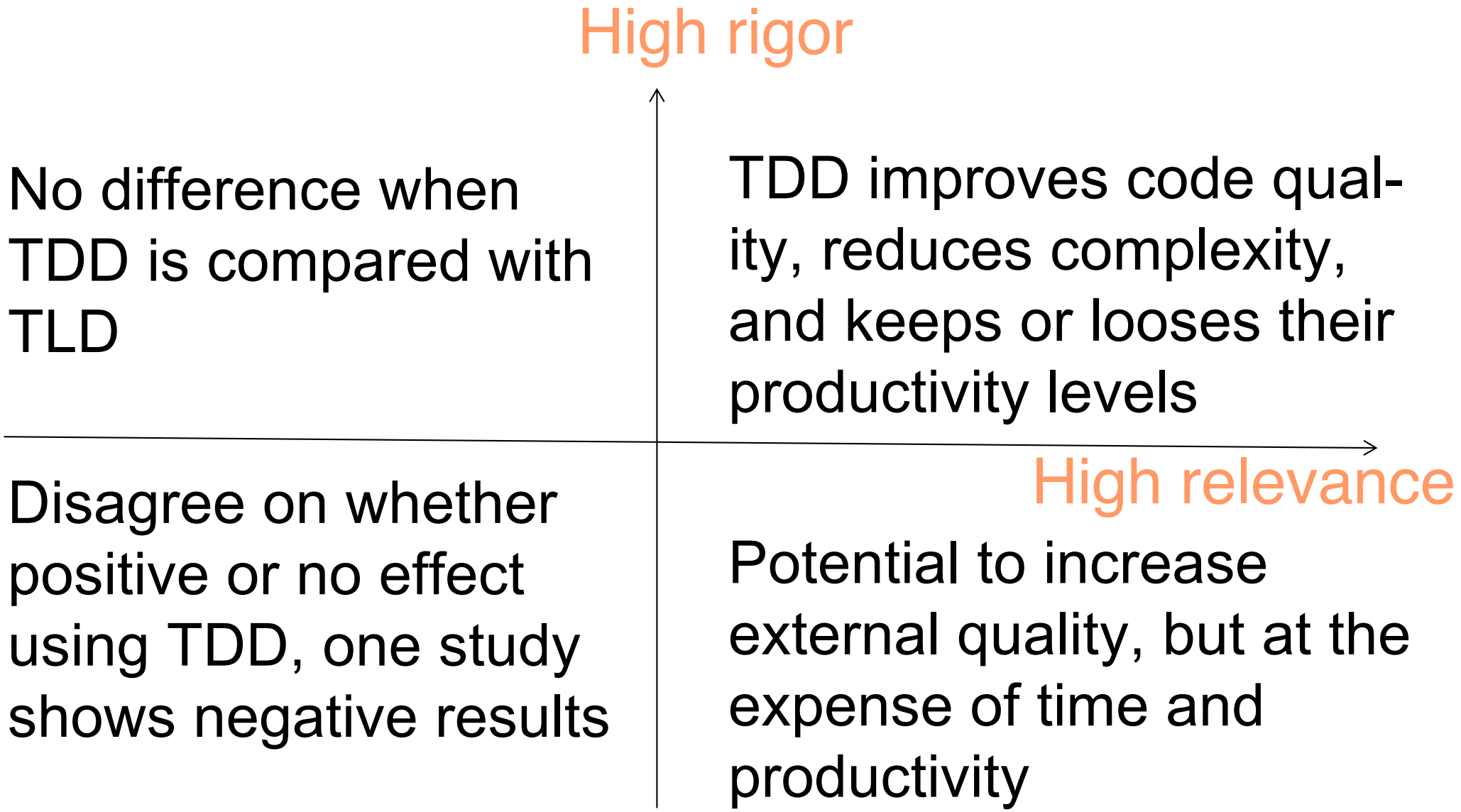

Write a test
Run the test
no
Test fails?
yes
repeat
Write code
Run the test
Fix errors
yes
Test fails?
no
Refactor code

TDD


Write code
Write a test
Run the test
Fix errors
yes
Test fails?
no

TLD

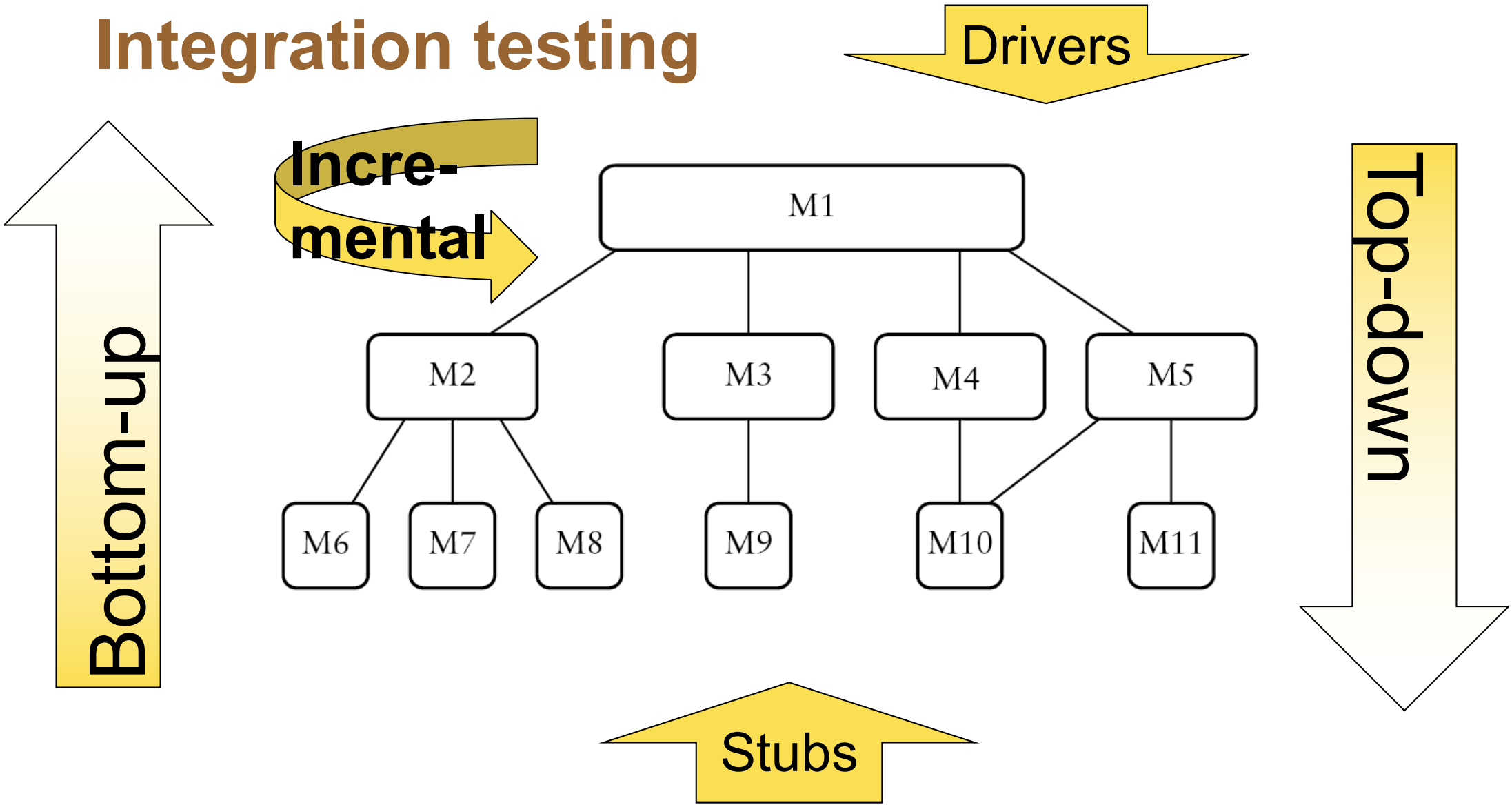
Test first or last? [Munir et al 2014]



Integration testing

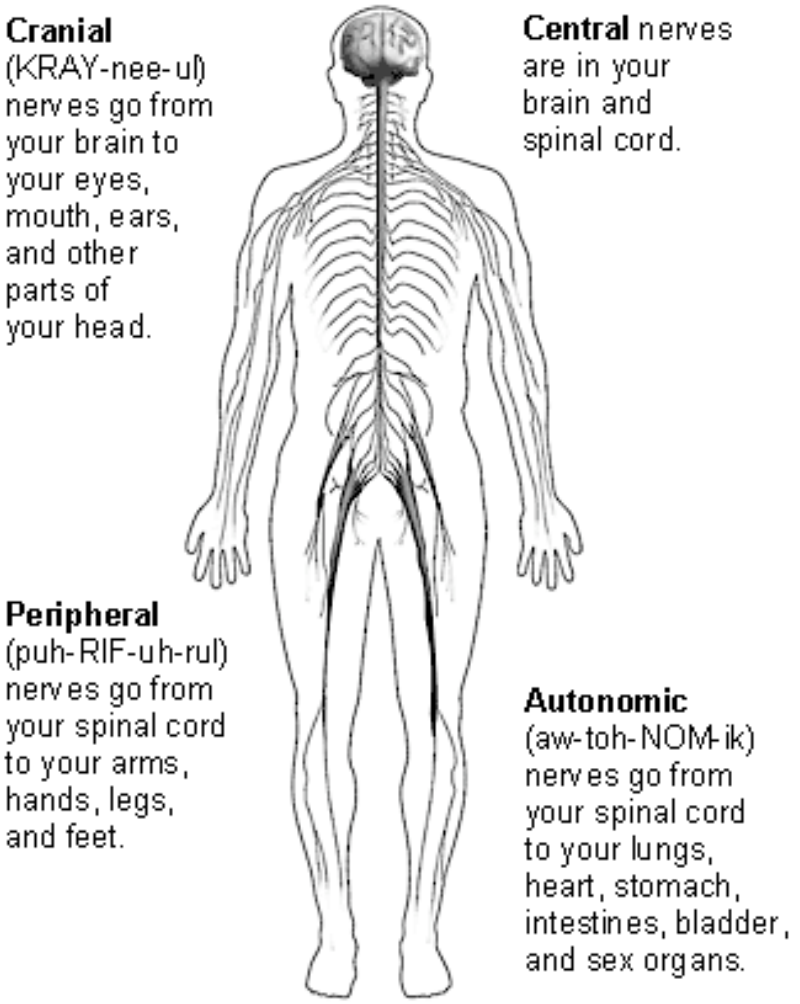
- More than one (tested) unit
- Detecting defects
  - On the interfaces of units
  - Communication between units
- Helps assembling incrementally a whole system
- Done by developers/designers or independent testers
  - Preferably developers and testers in collaboration
- Often omitted due to setup difficulties
  - Time is more efficiently spent on unit and system tests

Integration testing



System testing

- Testing the system as a whole
- Functional
  - Functional requirements and requirements-based testing
- Non-functional
  - Performance, stress, configuration, security, ...
  - As important as functional requirements
  - Often poorly specified'
  - Often done by specialists
  - Must be tested
- Often done by independent test group
  - Collaborating developers and testers



Types of system testing

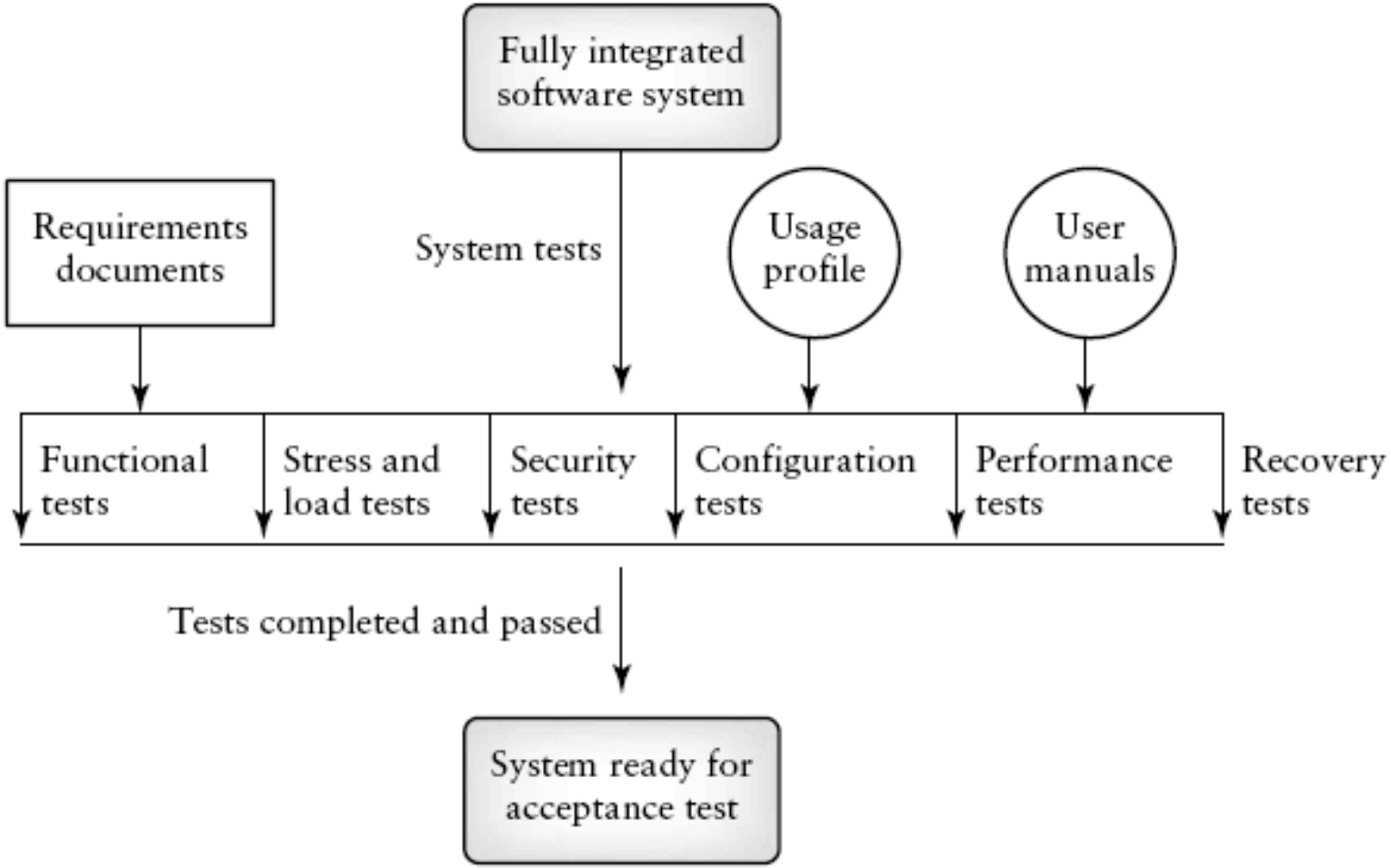
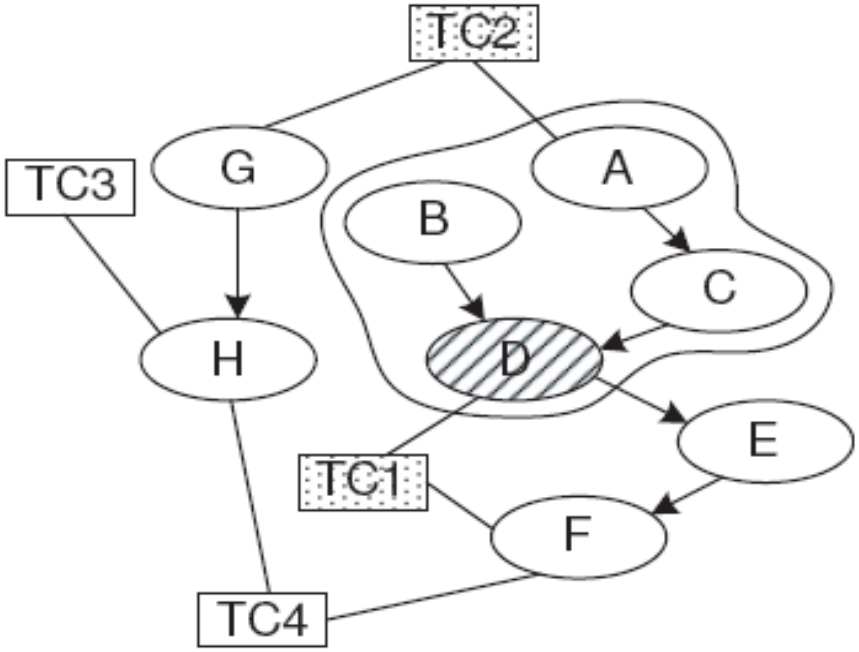


Fig. 6.10

Regression testing

- Testing changed software
  - Retest all
  - Selective regression testing
- Regression test suite
- Types of changes
  - Defect fixes
  - Enhancements
  - Adaptations
  - Perfective maintenance

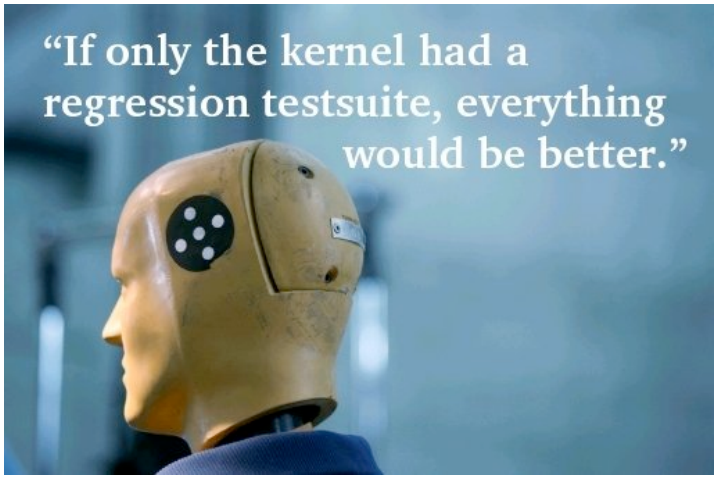


[Skoglund, Runeson, ISESE05]

Retest all

- Assumption:
  - Changes may introduce errors anywhere in the code
- Expensive, prohibitive for large systems
- Reuse existing test suite
- Add new tests as needed
- Remove obsolete tests (discrepancies between expected and actual output)

Selective regression testing



- Impact analysis
- Only code impacted by change needs to be retested
- Select tests that exercise such code
- Add new tests if needed
- Remove obsolete tests

Information and Software Technology 52 (2010) 14–30

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A systematic review on regression test selection techniques

Emelie Engström, Per Runeson *, Mats Skoglund

Department of Computer Science, Lund University, SE-221 00 Lund, Sweden

ARTICLE INFO

Article history:
Received 4 February 2009
Received in revised form 24 June 2009
Accepted 10 July 2009
Available online 18 July 2009

Keywords:
Regression testing

ABSTRACT

Regression testing is verifying that previously functioning software remains after a change. With the goal of finding a basis for further research in a joint industry-academia research project, we conducted a systematic review of empirical evaluations of regression test selection techniques. We identified 27 papers reporting 36 empirical studies, 21 experiments and 15 case studies. In total 28 techniques for regression test selection are evaluated. We present a qualitative analysis of the findings, an overview of techniques for regression test selection and related empirical evidence. No technique was found clearly superior since the results depend on many varying factors. We identified a need for empirical studies where con-

Industry practice of Regression Testing

What?

- Retesting, different levels

When?

- Daily – Before release

What?

- Functional focus
- “Smoke test” = selection

Challenges

- Cost/benefit of automation
- Automated test environment
- Presentation of test results

Good practices

- Automated daily module tests
- Automation below UI
- Visualize test progress

E. Engström and P. Runeson. A qualitative survey of regression testing practices. PROFES, LNCS6156, pp 3–16. Springer, 2010

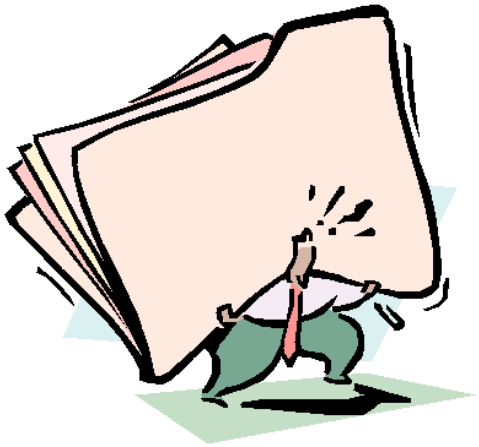
Acceptance testing (α, β)

- Final stage of validation
  - Customer (user) should perform or be closely involved
  - Customer can perform any test they wish
  - Final user sign-off
- Approach
  - Mixture of scripted and unscripted testing
  - Performed in real operation environment
- Project
  - Contract acceptance
  - Customer viewpoint
  - Validates that the right system was built
- Product
  - Final checks on releases
  - User viewpoint throughout the development; validation

To discuss

- What is the difference between test **levels** and test **phases**?
- Which of the test **types** are more/less feasible for different test **levels**?

Test artifacts – Chapter 12



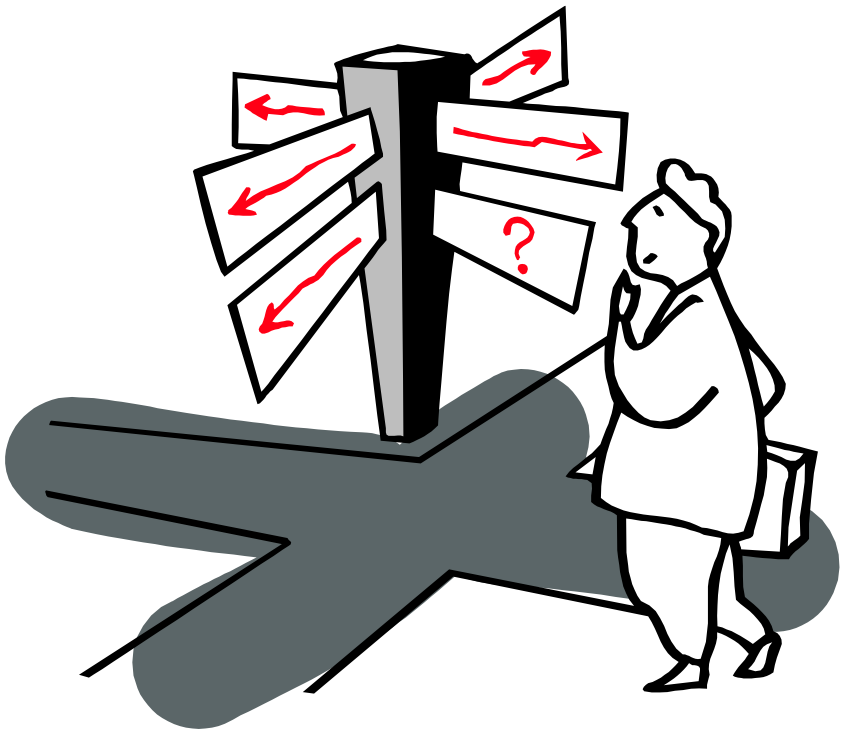
Test management

- Goal
- Policy
- Plan
- Follow-up

- Reach India
- Go west
- Get three ships
- …

Test planning

- Objectives
- What to test
- Who will test
- When to test
- How to test
- When to stop



Test Planning – different levels

Test Policy

Test Strategy

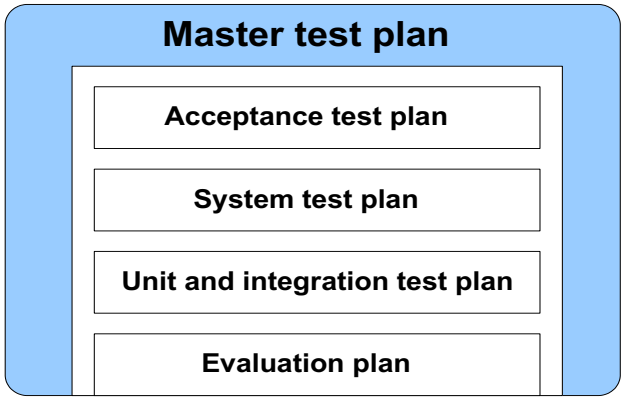
Master Test Plan

Detailed Test Plan

Company or product level

Project level

Number of needed levels depends on context (product, project, company, domain, development process, regulations, …)



Test levels (one for each within a project, e.g. unit, system, ...)

Software Project Management Plan

Goals

- Business
- Technical
- Business/technical
- Political

Quantitative/qualitative

Policies

High-level statement of principle or course of action that is used to govern a set of activities in an organization.

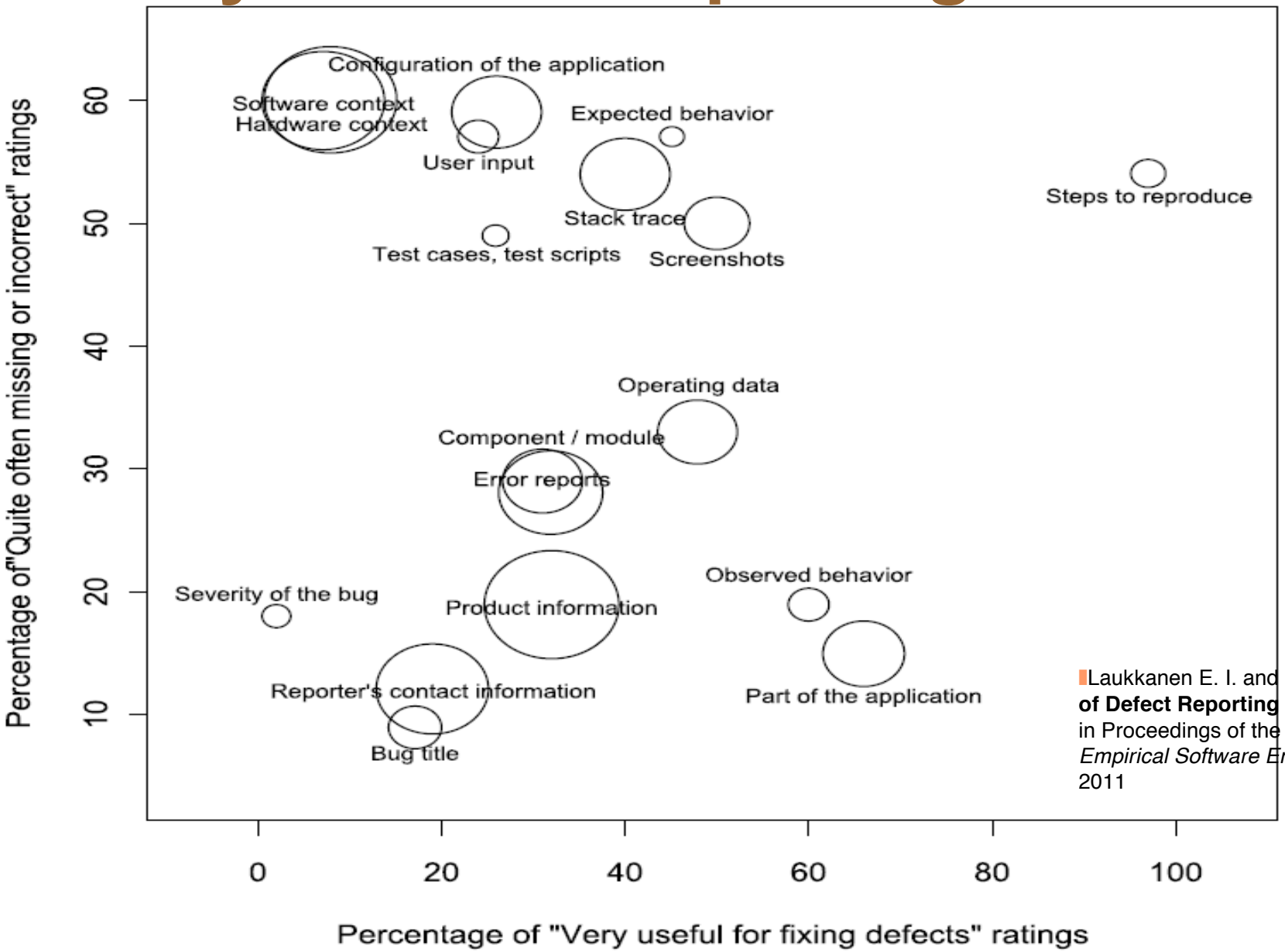
System test plan

1. Introduction
2. Feature description
3. Assumptions
4. Test approach
5. Test suite structure
6. Test environment
7. Test execution strategy
8. Test effort estimation
9. Scheduling and milestones

Defect report

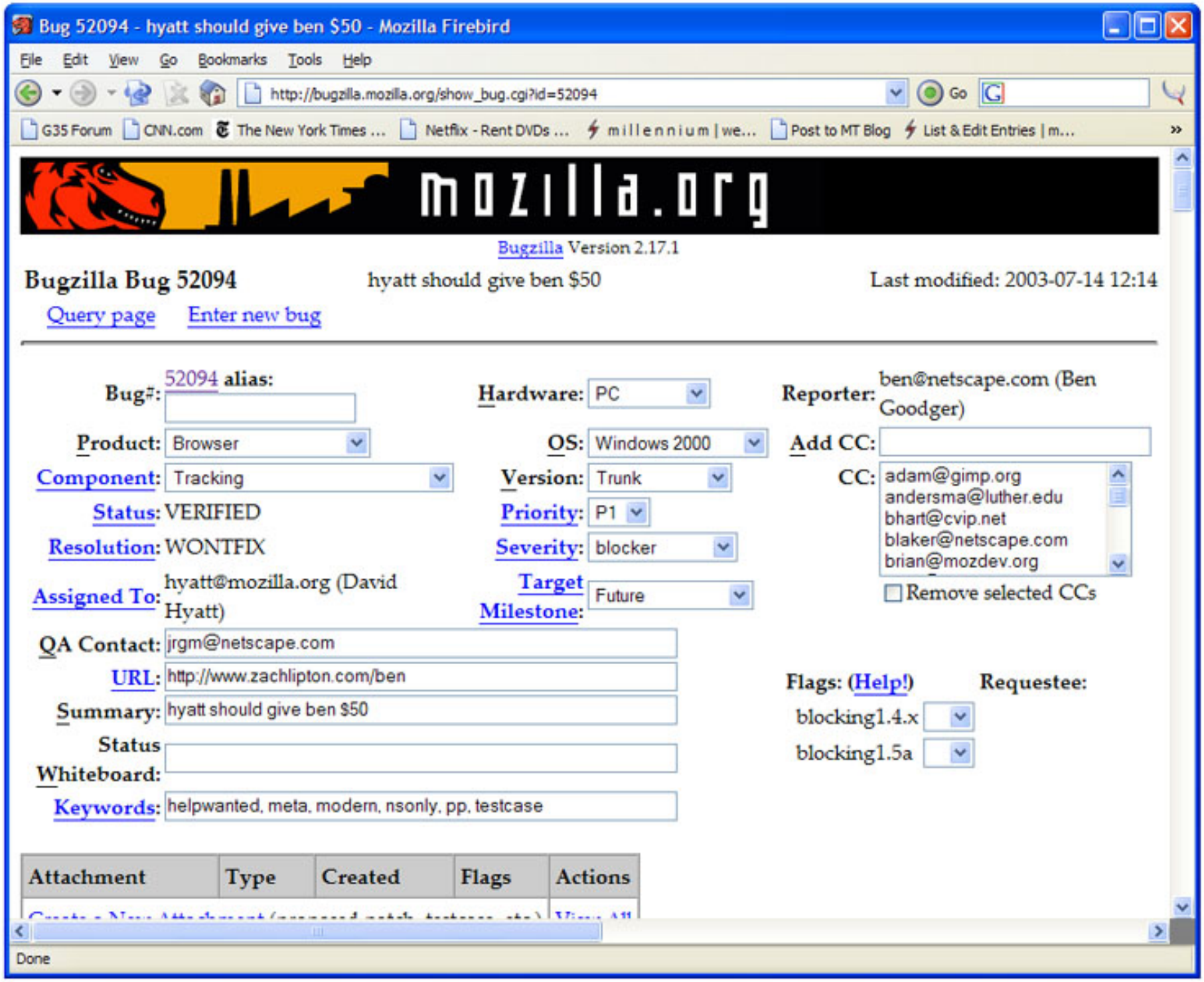
- Defect ID
- Date detected
- Project ID
- Defect type
- Location
- Symptoms
- Tester/inspector
- Defect origin
- Phase injected
- Phase detected
- Defect repair time
- Related problem report

Survey of defect reporting in SW industry



Laukkanen E. I. and Mäntylä M. V., "**Survey Reproduction of Defect Reporting in Industrial Software Development**" in Proceedings of the 5th *International Symposium on Empirical Software Engineering and Measurement* (ESEM), 2011

Figure 1. Summary of The Results. Size of circles is determined by difficulty of automatic collection (larger circles are easier).


Bug 52094 - hyatt should give ben $50 - Mozilla Firebird
File Edit View Go Bookmarks Tools Help
http://bugzilla.mozilla.org/show_bug.cgi?id=52094
Go
G35 Forum
CNN.com
The New York Times ...
Netflix - Rent DVDs ...
millennium | we...
Post to MT Blog
List & Edit Entries | m...
mozilla.org
Bugzilla Version 2.17.1
Bugzilla Bug 52094
hyatt should give ben $50
Last modified: 2003-07-14 12:14
Query page
Enter new bug
Bug#: 52094 alias:
Hardware: PC
Reporter: ben@netscape.com (Ben Goodger)
Product: Browser
OS: Windows 2000
Add CC:
Component: Tracking
Version: Trunk
CC: adam@gimp.org andersma@luther.edu bhart@cvip.net blaker@netscape.com brian@mozdev.org
Status: VERIFIED
Priority: P1
Resolution: WONTFIX
Severity: blocker
Remove selected CCs
Assigned To: hyatt@mozilla.org (David Hyatt)
Target Milestone: Future
QA Contact: jrgm@netscape.com
URL: http://www.zachlipton.com/ben
Summary: hyatt should give ben $50
Flags: (Help!)
Requestee:
blocking1.4.x
blocking1.5a
Status Whiteboard:
Keywords: helpwanted, meta, modern, nsonly, pp, testcase
Attachment
Type
Created
Flags
Actions
Done

Test results report

- Test cases executed
- Versions tested
- Defects found and reported

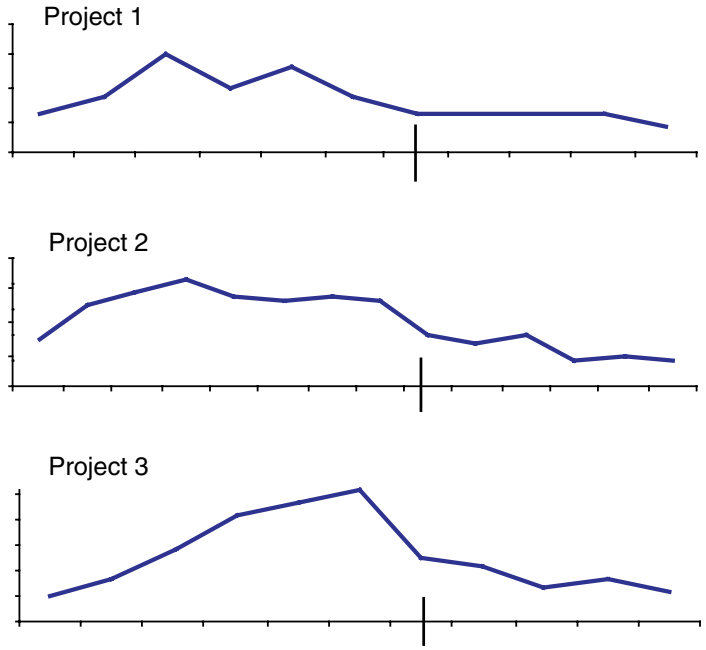


Figure 6. Defect distributions, showing number of defects detected over time, for the three studied projects. Ship dates are indicated with a vertical bar

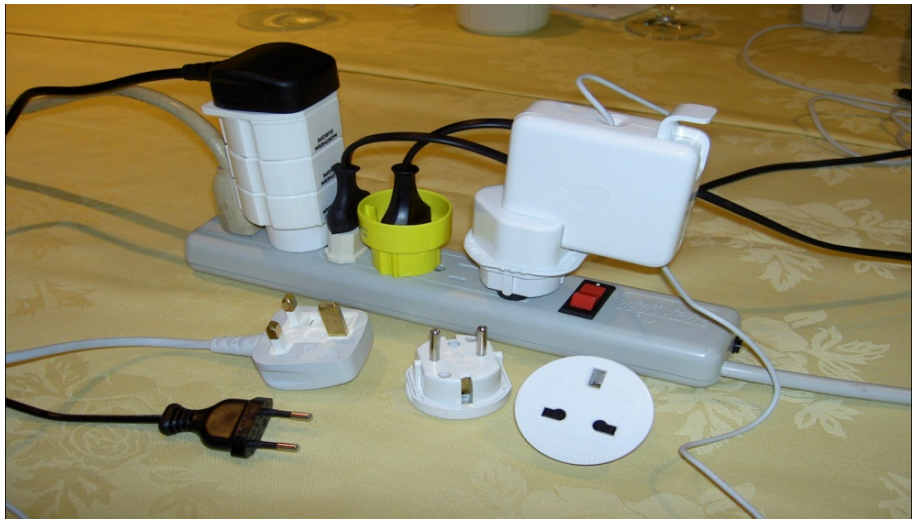
$$X_{Total,Ship} = X_{FT,\alpha} \times \frac{1}{Share_{FT,\alpha}} \times \frac{1}{Distr_{FT,Ship}}$$

where

$X_{FT,\alpha}$ = Total number of defects detected by FT at Alpha,
$Share_{FT,\alpha}$ = Percentage of defects detected by FT at Alpha compared to ship date,
$Distr_{FT,Ship}$ = Percentage of defects detected by FT compared to ST, CAT and Misc. at ship date

C. Andersson and P. Runeson. A Spiral Process Model for Case Studies on Software Quality Monitoring - Method and Metrics. SPIP, 12(2):125–140, 2007.

Standards



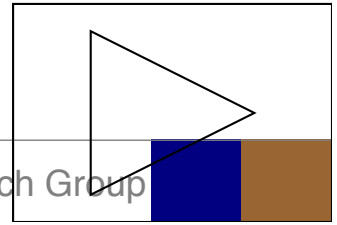
- IEEE 829-1998
  Standard for Software Test Documentation
- IEEE 1012-1998
  Standard for Software Verification and Validation
- IEEE 1008-1993
  Standard for Software Unit Testing

->

- ISO 29119
  Software Testing Concepts

Test plan according to IEEE Std 829-1998

a) Test plan identifier
b) Introduction
c) Test items
d) Features to be tested
e) Features not to be tested
f) Approach
g) Item pass/fail criteria
h) Suspension criteria and resumption requirements
i) Test deliverables
j) Testing tasks
k) Environmental needs
l) Responsibilities
m) Staffing and training needs
n) Schedule
o) Risks and contingencies
p) Approvals



Test plan quality criteria

Usefulness – Will the test plan effectively serve its intended functions?

Accuracy – Is the test plan document accurate with respect to any statements of fact?

Efficiency – Does it make efficient use of available resources?

Adaptability – Will it tolerate reasonable change and unpredictability in the project?

Clarity – Is the test plan self-consistent and sufficiently unambiguous?

Usability – Is the test plan document concise, maintainable, and helpfully organized?

Compliance – Does the test plan meet externally imposed requirements?

Foundation – Is the test plan the product of an effective test planning process?

Feasibility – Is the test plan within the capability of the organization that must use it?

A fresh contrast

- http://googletesting.blogspot.se/2011/09/10-minute-test-plan.html
- https://www.youtube.com/watch?v=QEu3wmgTLqo

To discuss

- What are the pros and cons of test standards?
- Which is most important, the test plan or the planning?

Recommended exercises

- Chapter 7
  - 1, 2, 3, 4, 5, 6, 7
- Chapter 12
  - 1, 2, 3, 5, 6, 7, 8, 13