AGILE SOFTWARE ENGINEERING TESTING

JOHN STOUBY PERSSON

AALBORG UNIVERSITY
DENMARK

Our learning from the guest lecture

Per Christian Møller

Ledelse
Transformation
Strategi
Organisations design
Produktudvikling
New ways of working
Digitalisering

PA

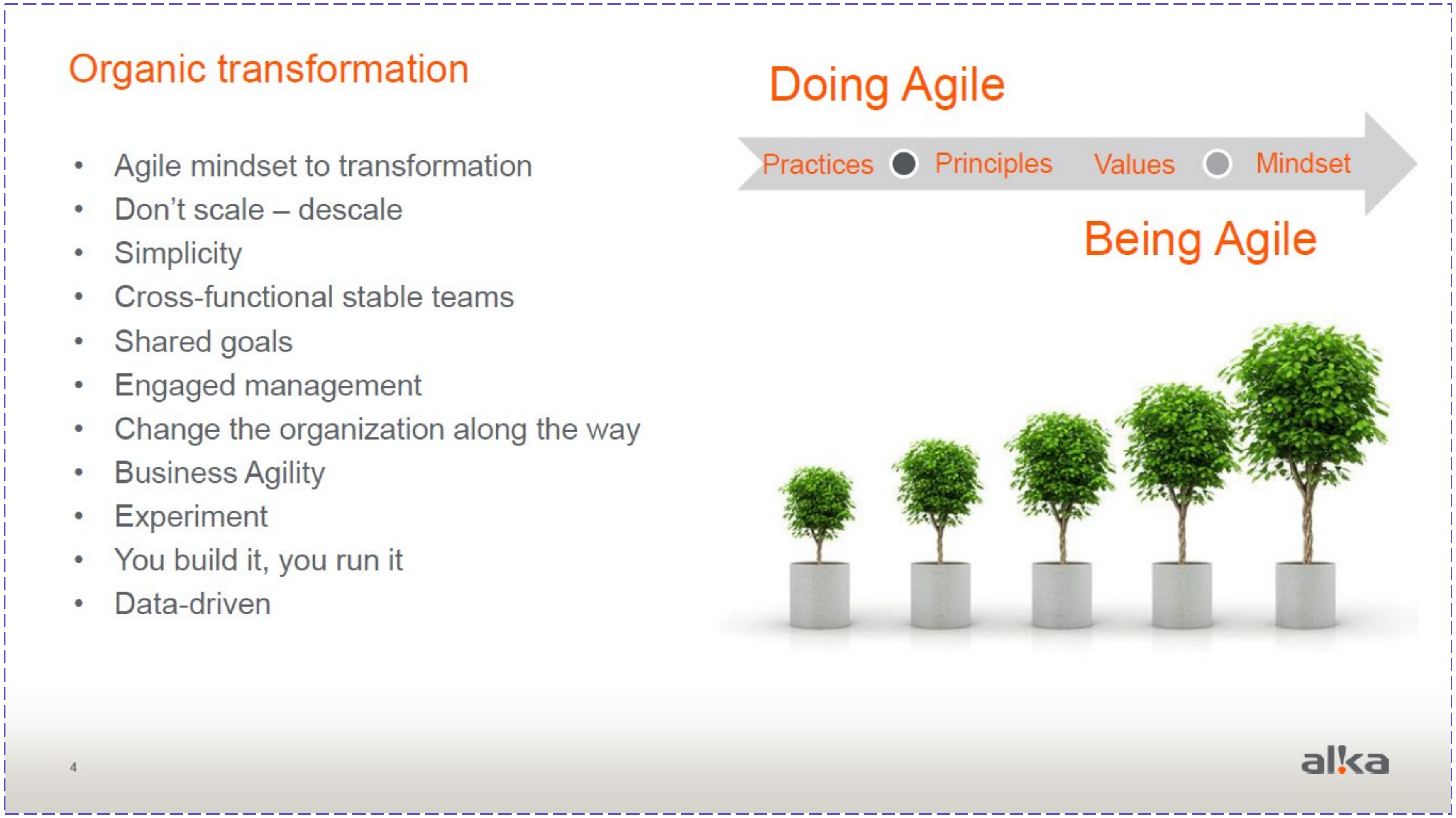
Nykredit

alka

VP SECURITIES

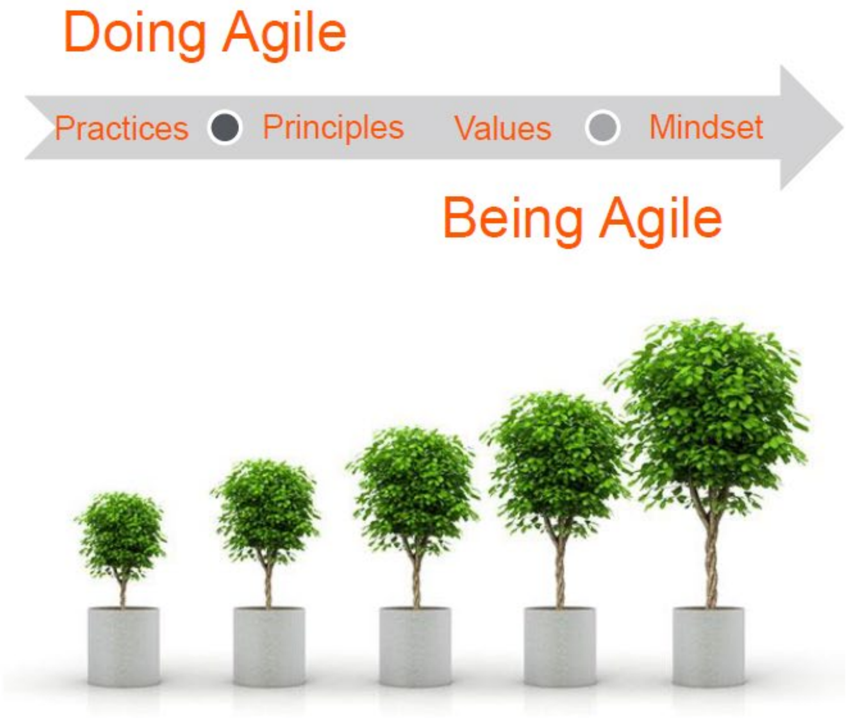
CBS COPENHAGEN BUSINESS SCHOOL

Our learning from the guest lecture



Organic transformation

- Agile mindset to transformation
- Don’t scale – descale
- Simplicity
- Cross-functional stable teams
- Shared goals
- Engaged management
- Change the organization along the way
- Business Agility
- Experiment
- You build it, you run it
- Data-driven



alka

Lecture objectives

Knowledge about testing in software engineering

Skills in organizing and conducting software test processes.

Competencies to manage testing activities in agile software engineering.

Software Testing

Purpose:

… show that a program does what is intended to do and to discover program defects before it is put to use …

Key activities:

1. Demonstrate the software meets it’s requirements
2. Find inputs or input sequences where the behaviour of the software is incorrect, undesirable or does notr conform to specification

Test versus inspection/review

<u>Software Test ties back to Quality Management and the concepts:</u>

- Verification (conform to requirements) and
- Validation (fit for use)

1. Test is dynamic and involves executing the system. If one error is found, you can never be sure that later anomalies are due to new anomalies or a result or a side effect of the first error
2. Inspection/review is static, and therefore no interactions between the errors found. Inspection can also consider broader quality issues: portability, maintainability, efficiency, …

Stages of testing

Development testing

- Unit testing
  - Partition testing, guidelines, overflow, invalid input
- Component
  - Interfaces misuse, misunderstanding
- System
  - Interactions, use-case based

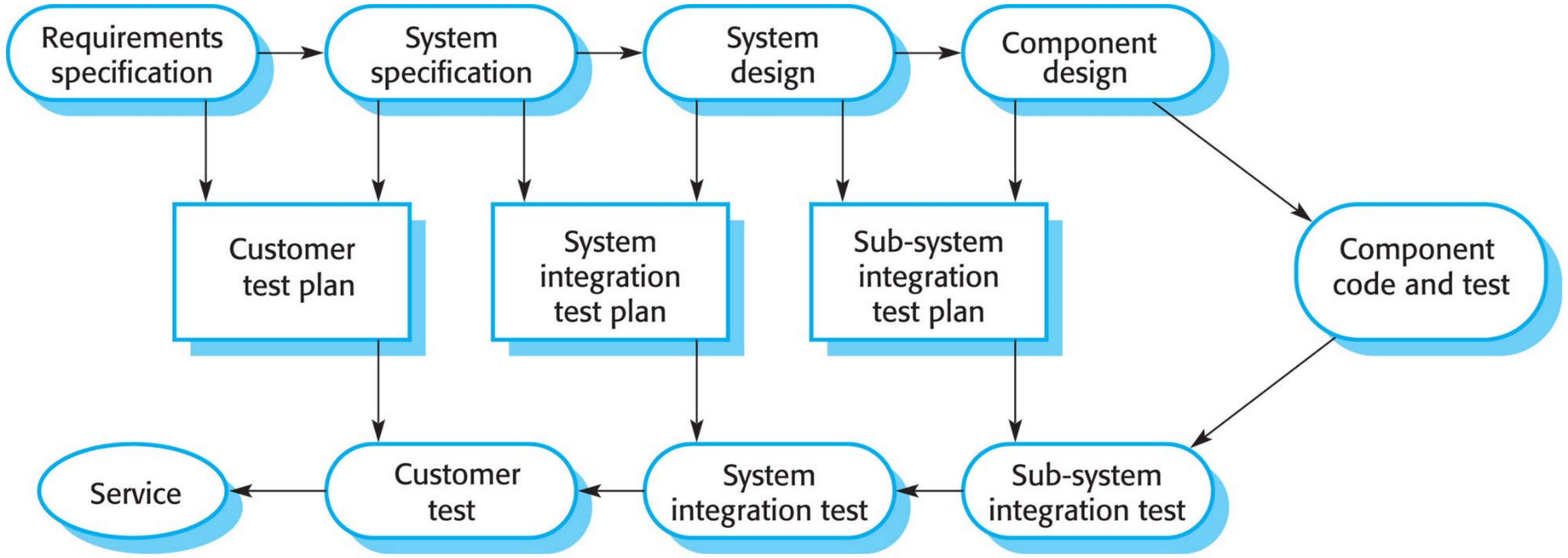
Release testing

- Requirements testing
  - All requirements have been satisfied
- Scenario testing
  - Good enough for use, functionality,
- Performance testing

User testing

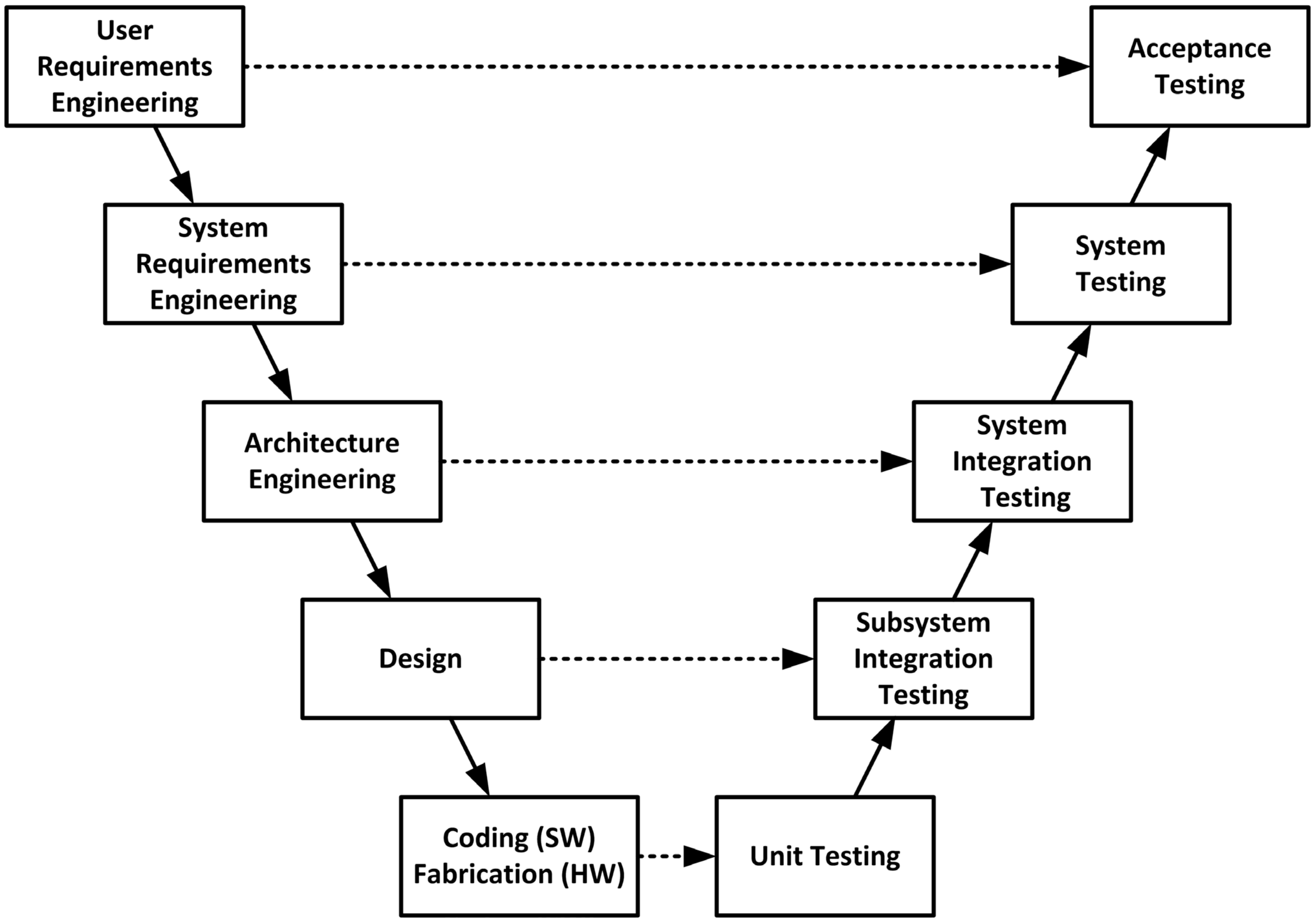
- Fit for use?, meet expectations?, exploratory, accept

Plan-Driven: V Model



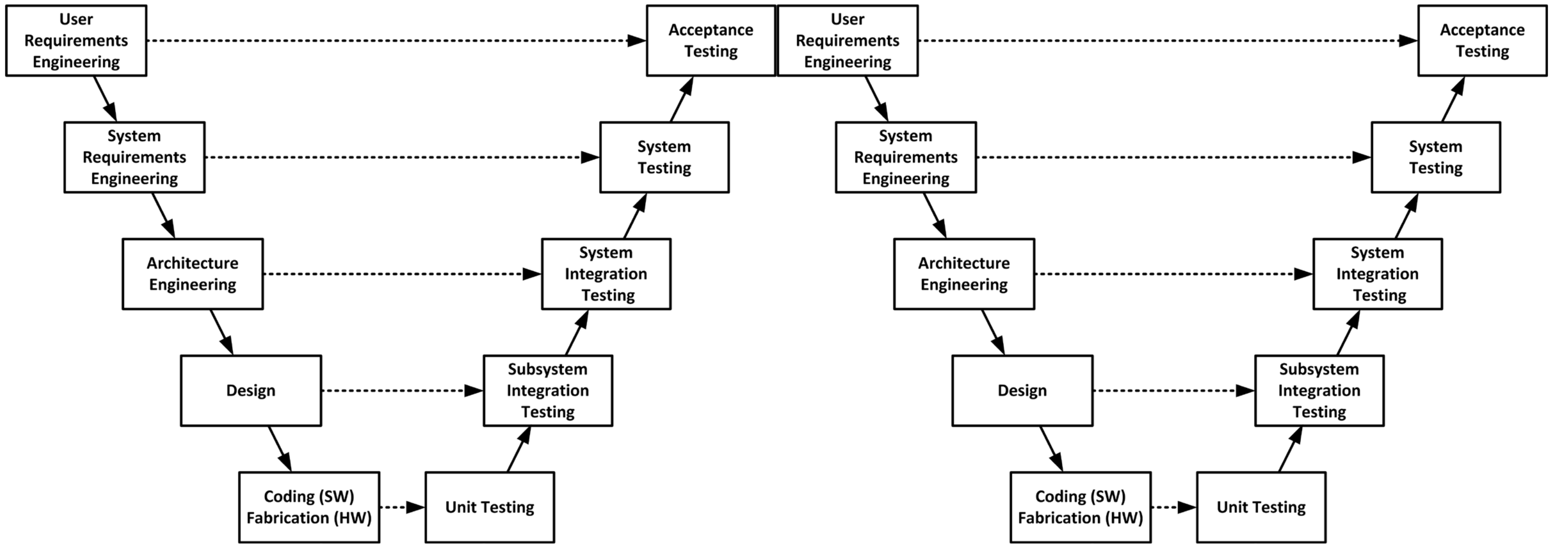
Source:
Sommerville, ch. 2

Plan-Driven: V Model


User Requirements Engineering
System Requirements Engineering
Architecture Engineering
Design
Coding (SW) Fabrication (HW)
Unit Testing
Subsystem Integration Testing
System Integration Testing
System Testing
Acceptance Testing

Source: Software Engineering Institute

Agile: W Model

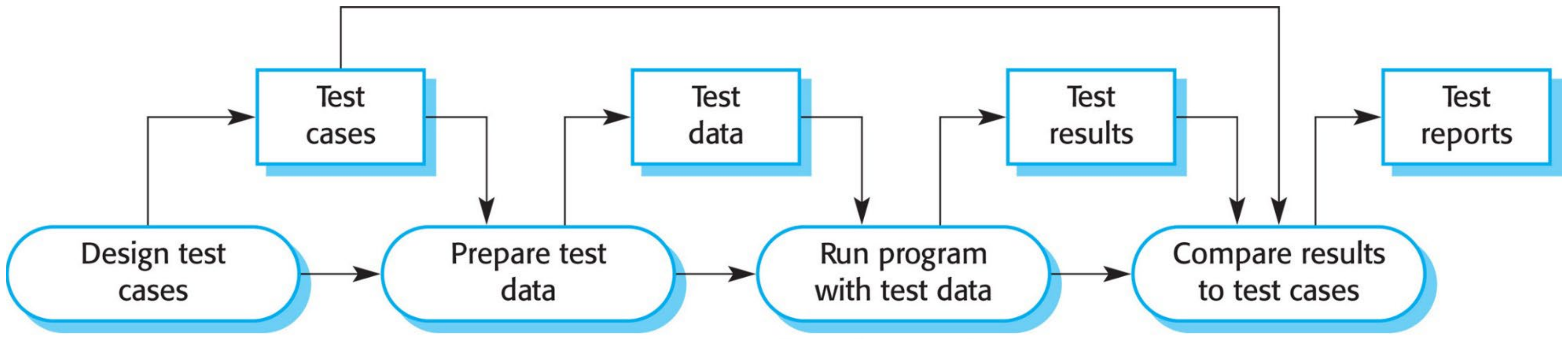

User Requirements Engineering
System Requirements Engineering
Architecture Engineering
Design
Coding (SW) Fabrication (HW)
Unit Testing
Subsystem Integration Testing
System Integration Testing
System Testing
Acceptance Testing
User Requirements Engineering
System Requirements Engineering
Architecture Engineering
Design
Coding (SW) Fabrication (HW)
Unit Testing
Subsystem Integration Testing
System Integration Testing
System Testing
Acceptance Testing

Regression testing

- Regression testing is testing the system to check that changes have not ‘broken’ previously working code.

- In a manual testing process, regression testing is expensive but, with automated regression testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.

Plan-Driven Testing



Agile Testing Quadrants

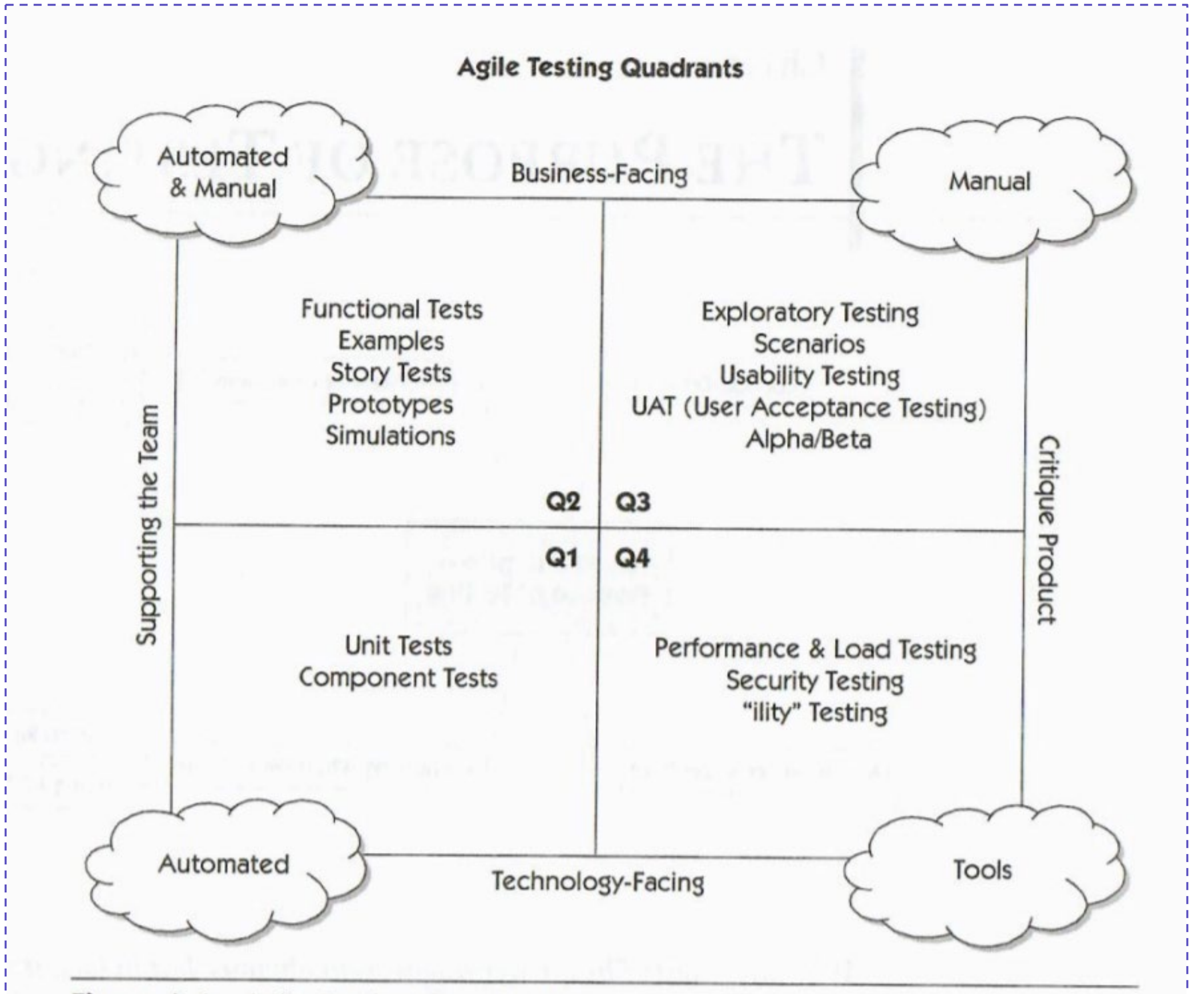
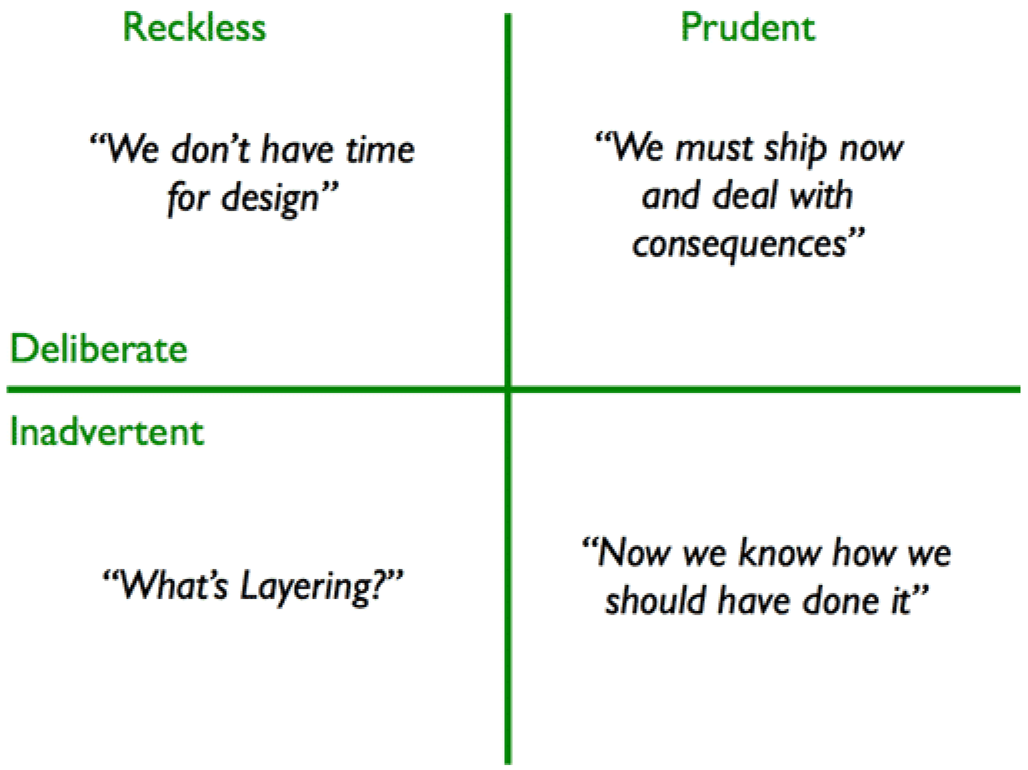


Figure 6-1 Agile Testing Quadrants

Lisa Crispin - agile Testing ch. 6

Technical Debt


Reckless
Prudent
"We don't have time for design"
"We must ship now and deal with consequences"
Deliberate
Inadvertent
"What's Layering?"
"Now we know how we should have done it"

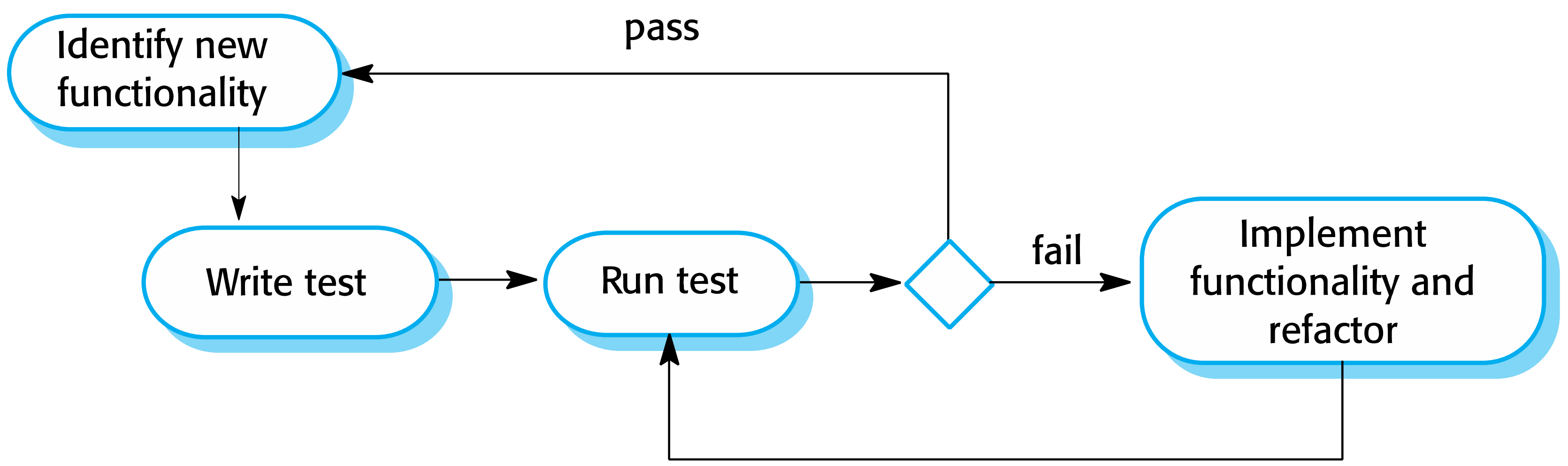
Fowler, M.: Technical debt quadrant.
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Test-driven development

- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- Tests are written before code and ‘passing’ the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don’t move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

Source: Sommerville, ch. 8

Test-driven development



Identify new functionality
pass
Write test
Run test
fail
Implement functionality and refactor

Test-driven development: process activities

- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality, so the new test will fail.
- Implement the functionality and re-run the test.
- Once all tests run successfully, you implement the next chunk of functionality.

Benefits of test-driven development

Code coverage

- Every code segment you write has at least one associated test, so all code written has at least one test.

Regression testing

- A regression test suite is developed incrementally as a program is developed.

Simplified debugging

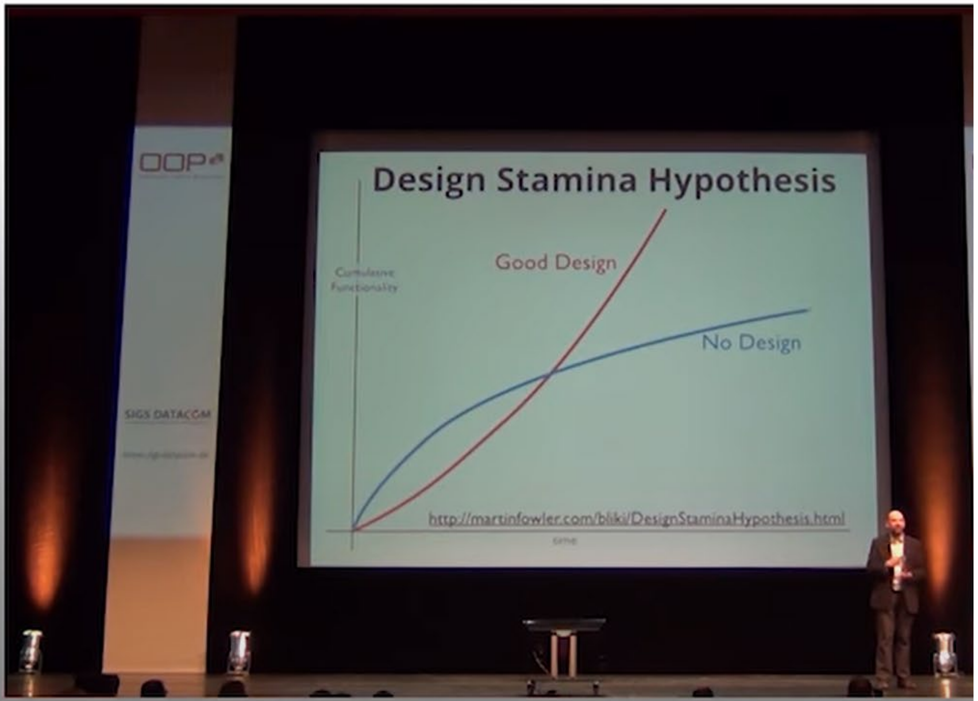
- When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.

System documentation

- The tests themselves are a form of documentation that describes what the code should be doing.

OOP 2014
SOFTWARE MEETS BUSINESS

"Workflows of Refactoring"



Martin Fowler
ThoughtWorks

Bad Smells (the Yuck!) guiding refactoring

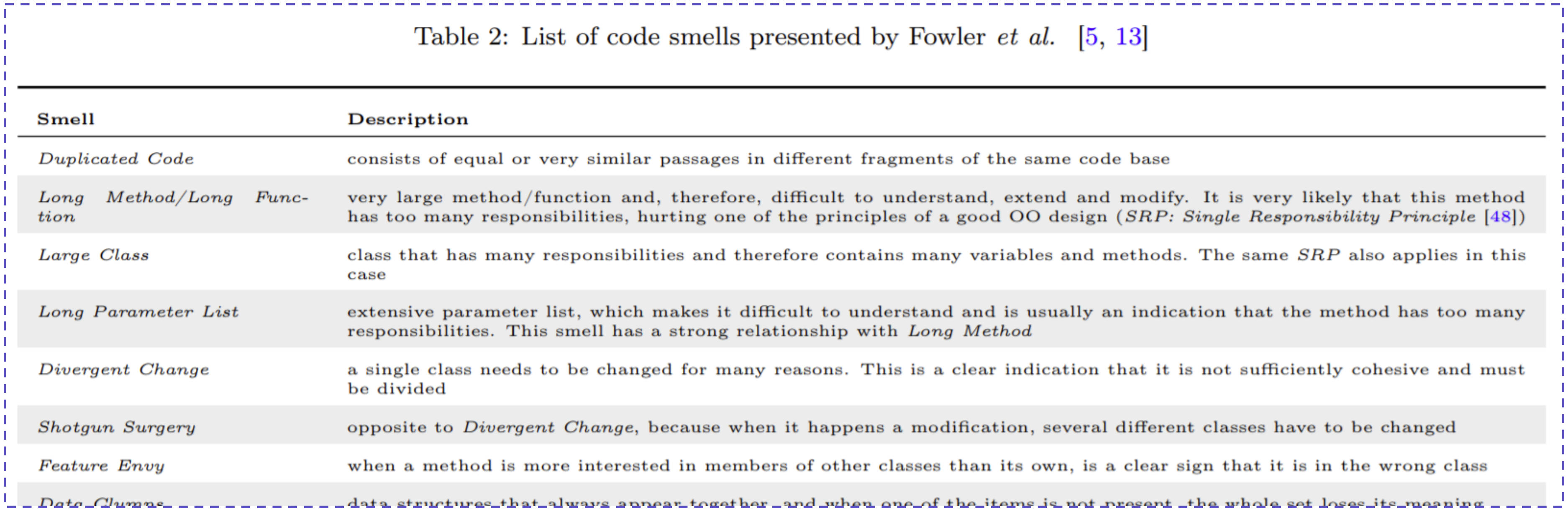


Table 2: List of code smells presented by Fowler *et al.* [5, 13]

Smell	Description
Duplicated Code	consists of equal or very similar passages in different fragments of the same code base
Long Method/Long Function	very large method/function and, therefore, difficult to understand, extend and modify. It is very likely that this method has too many responsibilities, hurting one of the principles of a good OO design (*SRP: Single Responsibility Principle* [48])
Large Class	class that has many responsibilities and therefore contains many variables and methods. The same *SRP* also applies in this case
Long Parameter List	extensive parameter list, which makes it difficult to understand and is usually an indication that the method has too many responsibilities. This smell has a strong relationship with *Long Method*
Divergent Change	a single class needs to be changed for many reasons. This is a clear indication that it is not sufficiently cohesive and must be divided
Shotgun Surgery	opposite to *Divergent Change*, because when it happens a modification, several different classes have to be changed
Feature Envy	when a method is more interested in members of other classes than its own, is a clear sign that it is in the wrong class
Data Clumps	data structures that always appear together, and when one of the items is not present, the whole set loses its meaning

https://arxiv.org/pdf/2004.10777.pdf

Lacerda, G., Petrillo, F., Pimenta, M., & Guéhéneuc, Y. G. (2020). Code smells and refactoring: A tertiary systematic review of challenges and observations. *Journal of Systems and Software*, *167*, 110610.

Group exercises

Exercise 1: Decide what bad smells (e.g., from Lacerda et al. (2020) or check your risk assessment from lecture 7) are important to your semester project and then refactor a part of it (preferable code) (50%)

Exercise 2: Make a test strategy for each quadrant for your semester project (or the imaginary product) (50%)

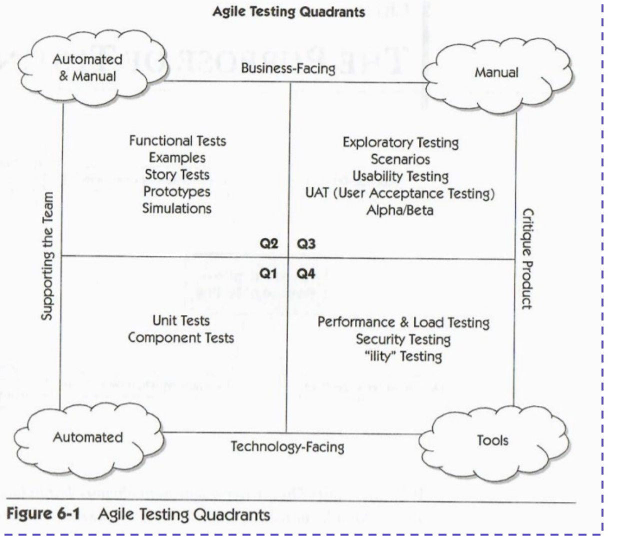


Figure 6-1 Agile Testing Quadrants

Lecture objectives

Knowledge about testing in software engineering

Skills in organizing and conducting software test processes.

Competencies to manage testing activities in agile software engineering.