TEST DESIGN TECHNIQUES

TEST

PBA SOFTWAREUDVIKLING/
BSC SOFTWARE DEVELOPMENT

Christian Nielsen cnls@cphbusiness.dk

Tine Marbjerg tm@cphbusiness.dk

SPRING 2019

TODAY'S TOPICS

- **Test design techniques**
  - (Black box testing)
  - White box testing
  - Test coverage
- **Static techniques**
- **Testability (chap 4-6)**

Learning Objectives from Curriculum

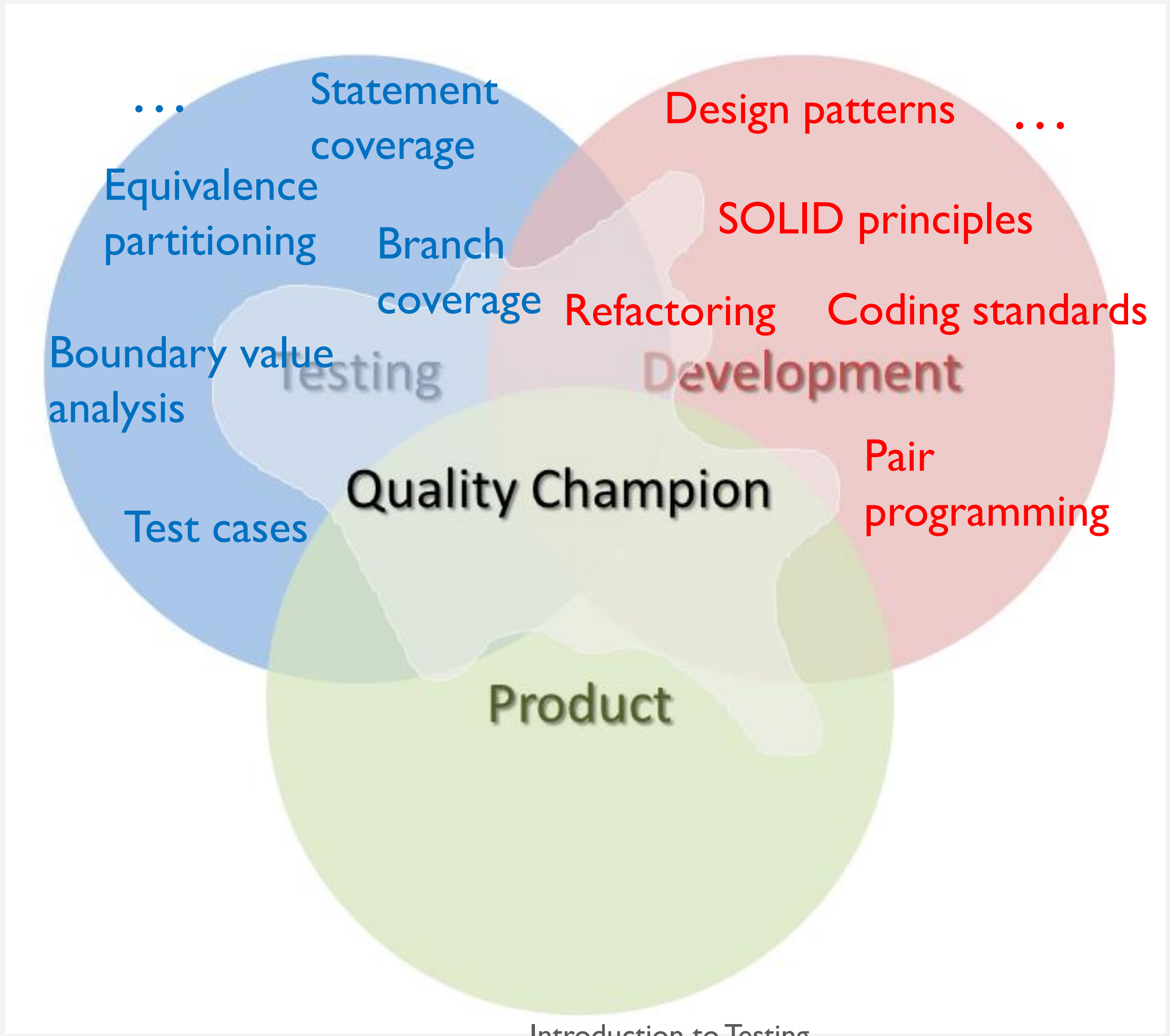
Skills

The student can
- apply black-box and white-box testing techniques
- apply various criteria for test coverage

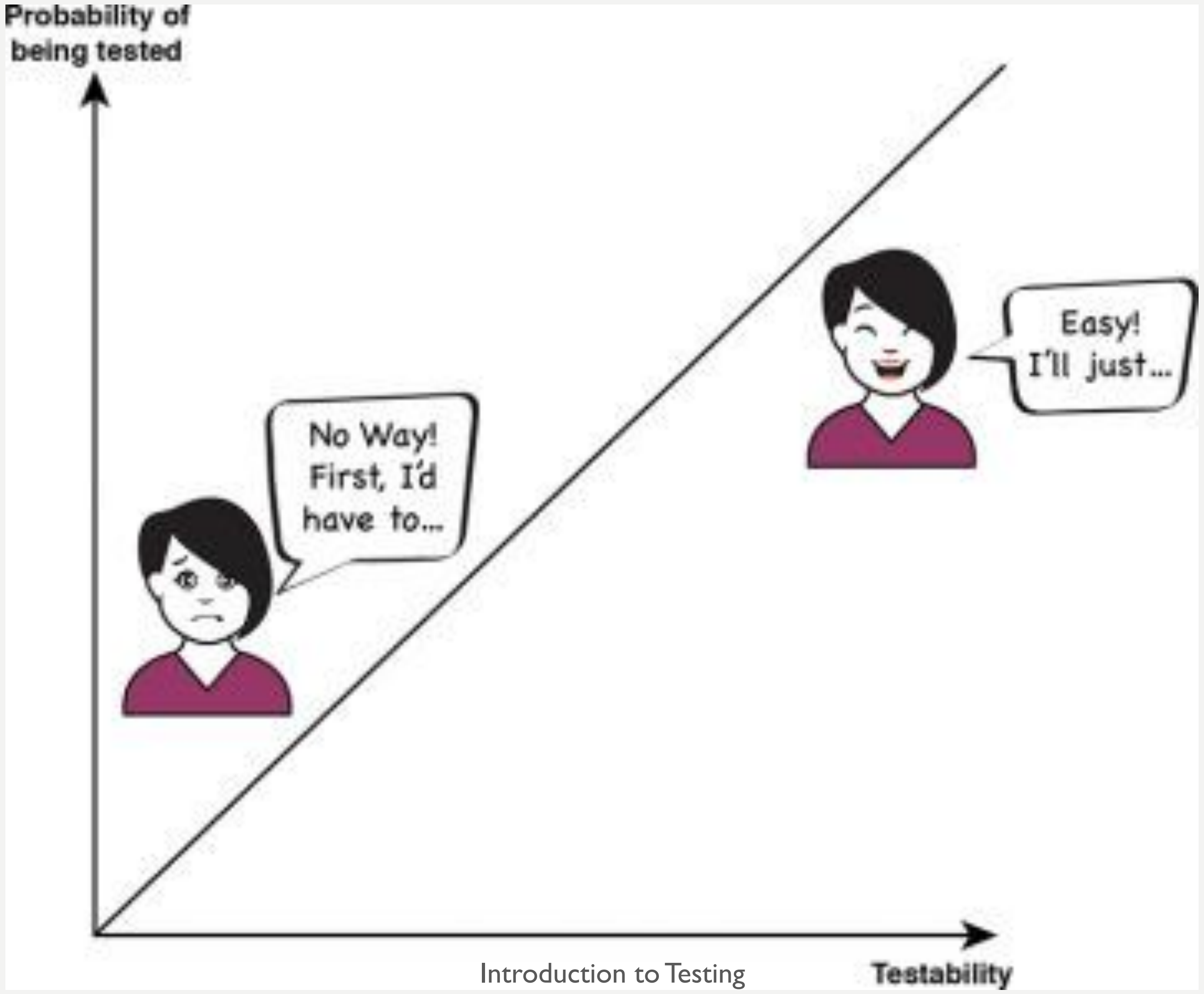
Competences

The student can
- design testable systems

DESIGN SKILLS



TESTABLE SOFTWARE



TEST DESIGN TECHNIQUES

- Static testing techniques
  - Generally used before any tests are executed on the software

Introduction today

- Specification-based (black-box) techniques
  - Input/output driven
  - Focus on the functional externals
  - Applied on all test levels where specification exists

Last week and more next week

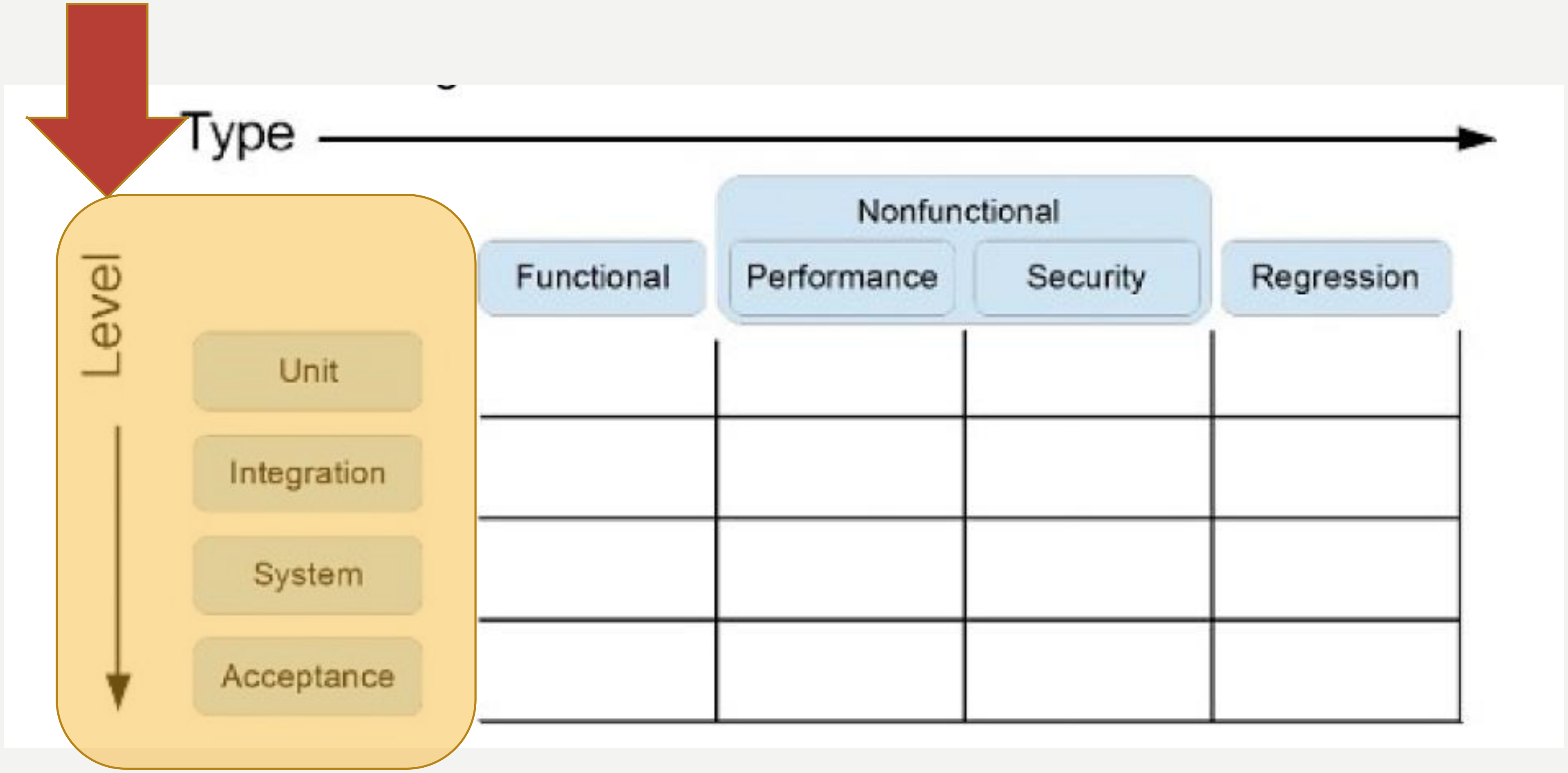
- Structure-based (white-box) techniques
  - Logic driven
  - Focus on internal structure of the software
  - Primarily unit and integration test level (good tool support)

Introduction today

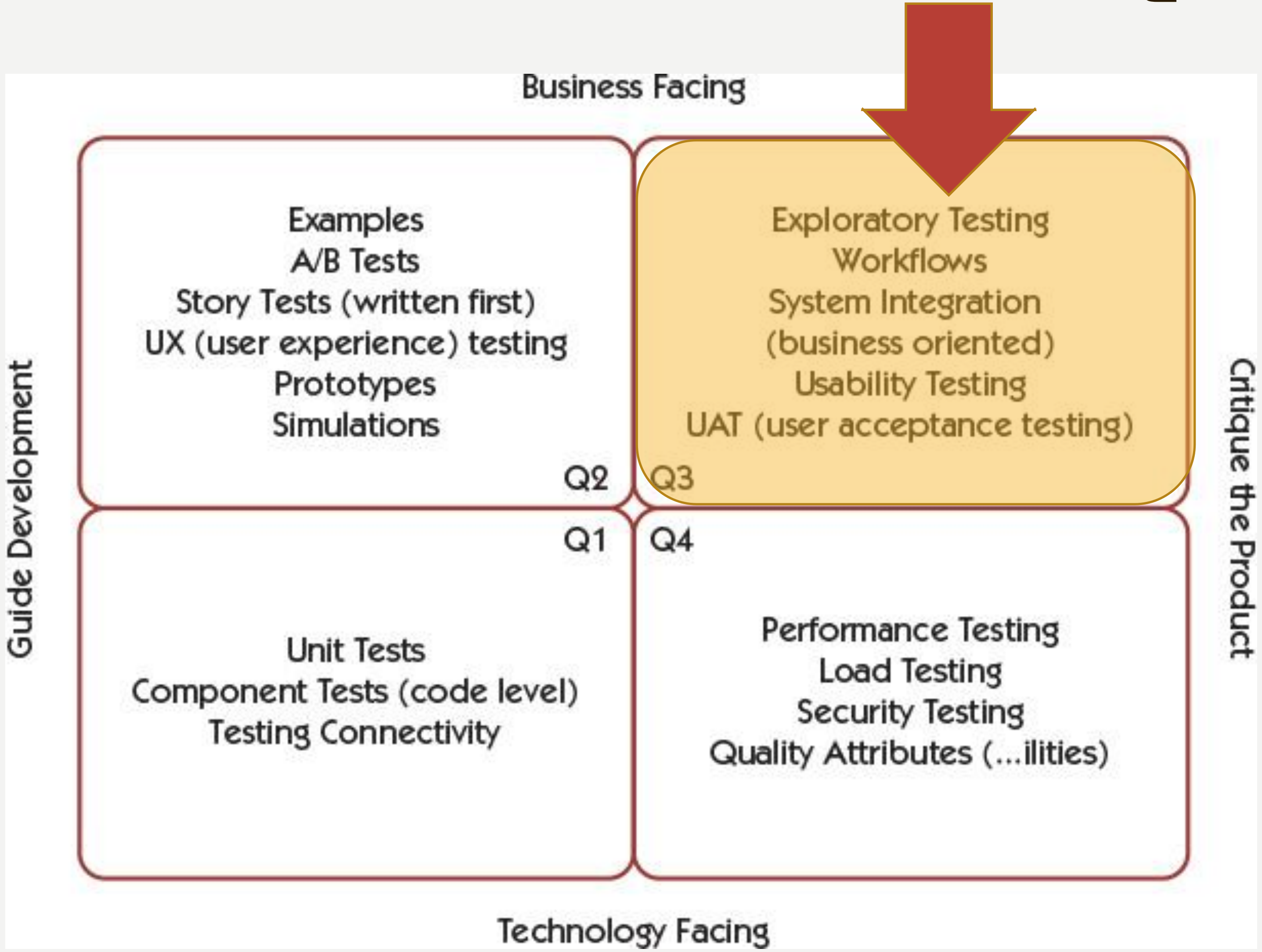
- Experience-based techniques
  - People's knowledge and skills are prime contributor to test case design
  - Complements the above, especially if no (or inadequate) spec.
  - Test cases are derived less systematic, but may be more effective ☺

Brief introduction later in semester

TEST LEVELS


Type
Level
Unit
Integration
System
Acceptance
Functional
Nonfunctional
Performance
Security
Regression

EXPERIENCE BASED TECHNIQUES



BLACK-BOX TECHNIQUES

- Equivalence partitioning
- Boundary value analysis
- Decision table testing
- State transition testing
- Pairwise testing

Last week (Equivalence partitioning, Boundary value analysis)

Next week (Decision table testing, State transition testing, Pairwise testing)

EQUIVALENCE PARTITIONING - EXAMPLE

```
// pre: 0 < age
// post: returns true if age >= 18, otherwise false

public boolean legalAge(int age)
```

What are the equivalence classes for this method?

Which test cases will the equivalence classes result in?

BOUNDARY VALUE ANALYSIS

- Boundary Value Analysis is a heuristic
  - errors happen at the edges
  - watch out for off by one errors

- Boundary Value Analysis is applied to input fields or anywhere you have ranges of input with some maximum and minimum validation.

BVA EXAMPLE

- Input field which accepts values from 10 to 100
  - 10 would be the minimum, 100 would be the maximum

- Modelled that as a Set of Ordered sets:

{ x <= 9}{10 -> 100}{101 to infinity}

- The first set
  - all the values less than 10
- The second set:
  - values from 10 to 100 inclusive
- The third set
  - values greater than 100

The values on the boundaries between the sets are (9,10) and (100,101)

BVA EXAMPLE - 3 VALUE APPROACH

- Another way to consider this is as -1,0 and +1 for the valid boundary minimum and maximum range values.

{ x <= 9}{10 -> 100}{101 to infinity}

- 10 -1 = 9
- 10 + 0 = 10
- 10 + 1 = 11 (optional extension)
- 100 -1 = 99 (optional extension)
- 100 + 0 = 100
- 100 + 1 = 101

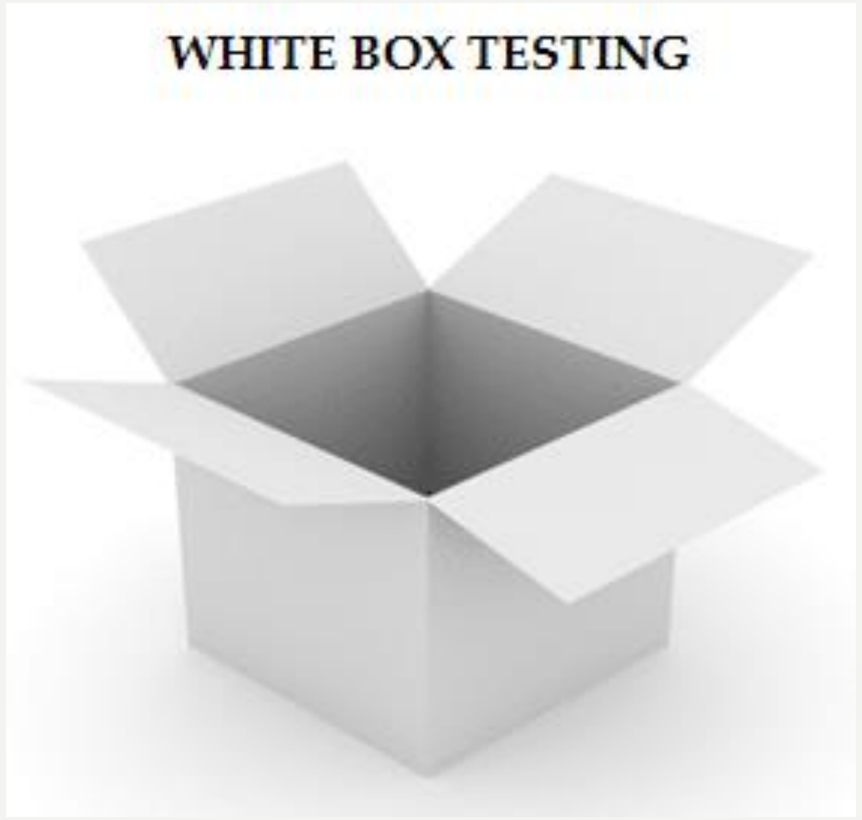
Named three value BVA approach
The additional values within the same equivalence class are optional and are an extension
11 (+1 on the minimum range value)
99 (-1 on the maximum range value).

PROBLEM: OPEN BOUNDARIES

- Open boundaries are more difficult to test, but there are ways to approach them.
- The best solution is to find out what the boundary should be specified as ☺
  - Look in spec
  - Ask product owner
  - Investigate other related areas of the system. Ex.:
    - The field that holds the account balance figure may be only six figures plus two decimal figures. This would give a maximum account balance of $999 999.99 so we could use that as our maximum boundary value.

WHITE-BOX TECHNIQUES

- Test coverage measurement
- Structural test case design



TEST COVERAGE

- Test coverage can be measured based on a number of different structural elements in a system or component
- Is the percentage to which a specified coverage item has been exercised by a test suite

$$\text{Coverage} = \frac{\text{Number of coverage items exercised}}{\text{Total number of coverage items}} \times 100\%$$

CODE COVERAGE

- Method coverage
- Statement coverage
- Decision coverage
- Branch coverage
- Path coverage

- White box testing requires *better programming skills* than black box testing

- Code coverage techniques are most often used on critical areas of the software, e.g.
  - Safety-critical code, vital code, complex logic

- Measurement of code coverage can be supported by *tools*

STATEMENT COVERAGE

You get 100 % statement coverage when test cases are designed so all statements in the program are traversed at least once

$$\text{Statement Coverage} = \frac{\text{No. of statements exercised}}{\text{Total no. of statements}} \times 100\ \%$$

Also known as line coverage (covers the true conditions)

STATEMENT COVERAGE EXAMPLE 1

100 % statement coverage:

Test case

A = 12, B = 10

```
READ A
READ B
IF A > B THEN C = 0
ENDIF
```

STATEMENT COVERAGE EXAMPLE 2

100 % statement coverage *:

Test cases

A = 2, B = 3 83 % statement coverage
PLUS

A = 20, B = 25

```
READ A
READ B
C = A + 2 * B
IF C > 50 THEN
   PRINT ’Large C’
ENDIF
```

* All lines in pseudo example are considered to be statements

DECISION COVERAGE

- Stronger logic-coverage criterion where both True and False outcome for each decision must be covered in a test case

$$\textbf{Decision Coverage} = \frac{\textbf{No. of decisions exercised}}{\textbf{Total no. of decisions}} \times 100\ \%$$

DECISION COVERAGE EXAMPLE

100 % decision coverage *:

Test cases

A = 20, B = 15   100 % statement coverage

PLUS

A = 10, B = 2

```
READ A
READ B
C = A - 2 * B
IF C < 0 THEN
   PRINT ’C negative’
ENDIF
```

* All lines in pseudo example are considered to be statements

TEST CASES DERIVED FROM WHITE-BOX TESTING

The resulting test suite includes enough input data sets to make sure that (decision coverage):

- all methods have been called,
- both the true and false branches have been executed in if statements,
- every loop has been executed zero, one, and more times,
- all branches of every switch statement have been executed.

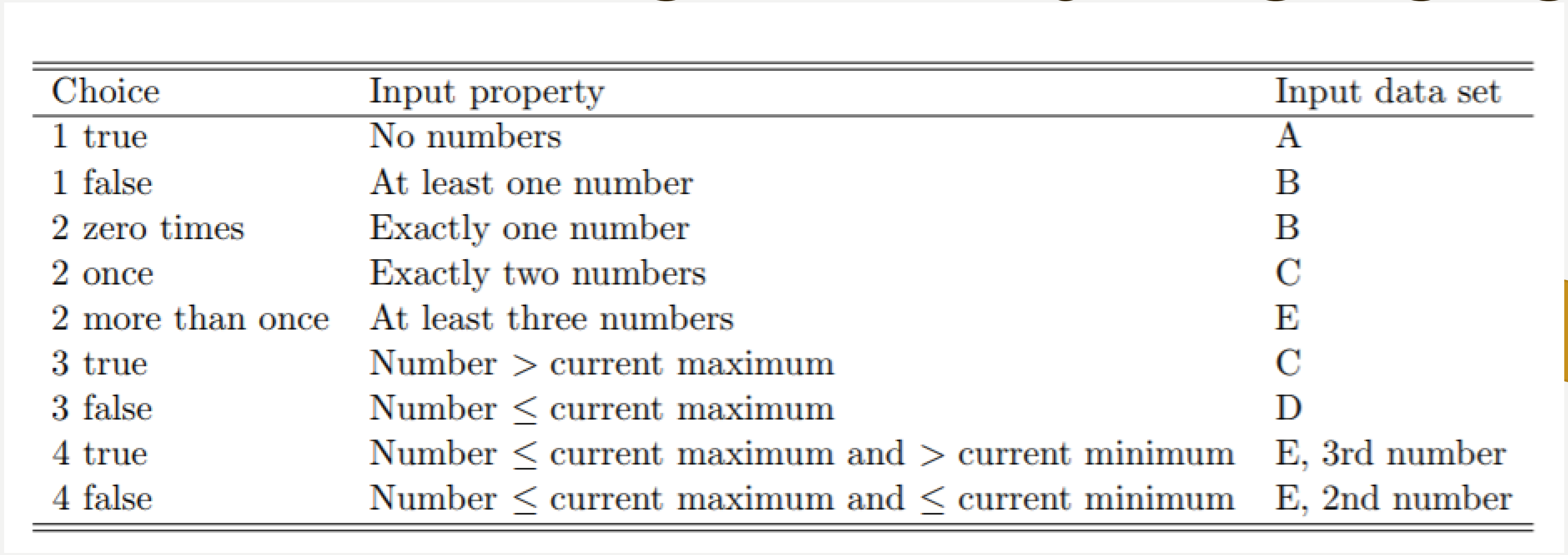
For every input data set, the expected output must be specified also.

WHITE-BOX TESTING EXAMPLE

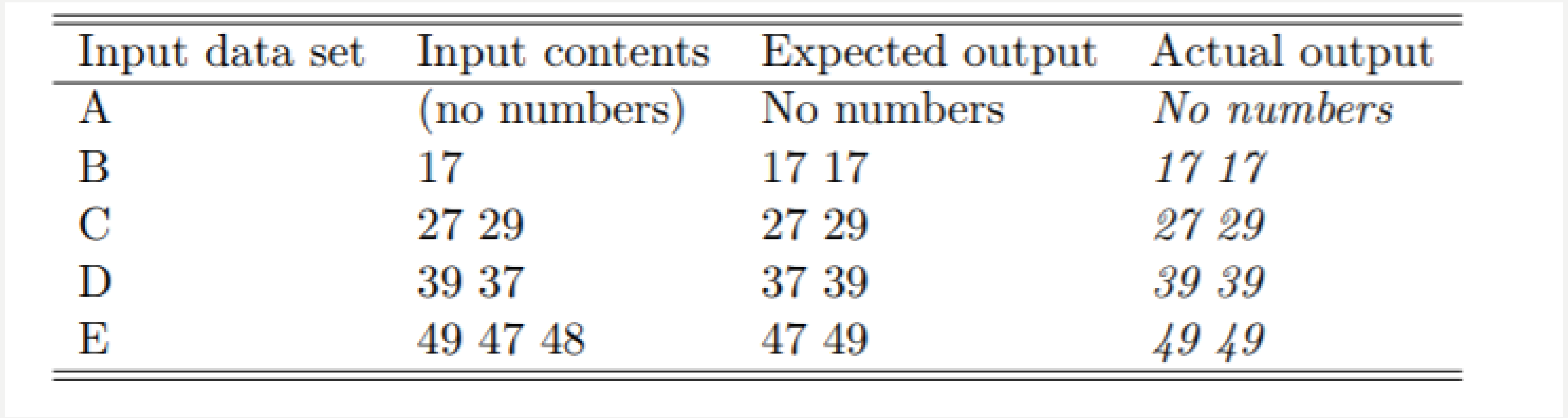
Source: Peter Sestoft - Systematic software testing

```
public static void main(String[] args) {
        int mi, ma;
        if (args.length == 0)                      /* 1 */
            System.out.println("No numbers");
        else
        {
            mi = ma = Integer.parseInt(args[0]);
            for (int i = 1; i < args.length; i++)   /* 2 */
            {
                int obs = Integer.parseInt(args[i]);
                if (obs > ma) ma = obs;             /* 3 */
                else if (mi < obs) mi = obs;        /* 4 */
            }
            System.out.println("Minimum = " + mi + "; maximum = " + ma);
        }
}
```

IDENTIFY INPUT DATA & TEST CASES



Choice	Input property	Input data set
1 true	No numbers	A
1 false	At least one number	B
2 zero times	Exactly one number	B
2 once	Exactly two numbers	C
2 more than once	At least three numbers	E
3 true	Number $>$ current maximum	C
3 false	Number $\leq$ current maximum	D
4 true	Number $\leq$ current maximum and $>$ current minimum	E, 3rd number
4 false	Number $\leq$ current maximum and $\leq$ current minimum	E, 2nd number

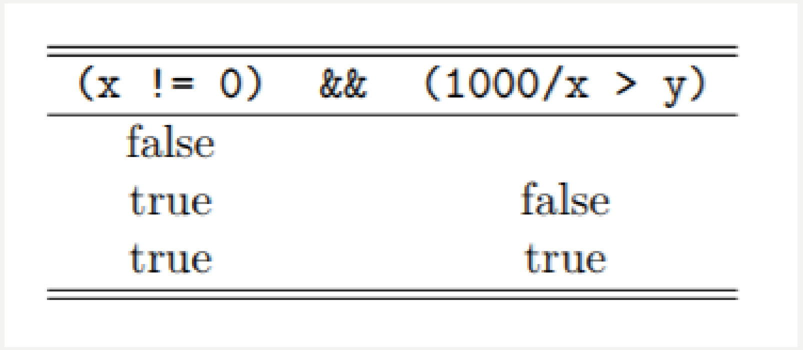


Input data set	Input contents	Expected output	Actual output
A	(no numbers)	No numbers	*No numbers*
B	17	17 17	*17 17*
C	27 29	27 29	*27 29*
D	39 37	37 39	*39 39*
E	49 47 48	47 49	*49 49*

SHORT-CUT LOGICAL OPERATORS

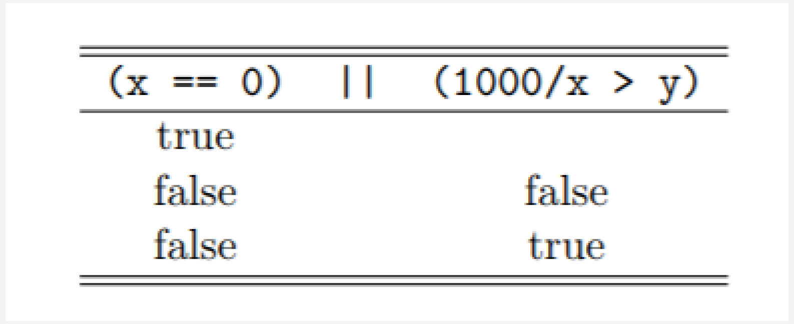
- Must be tested for all possible combinations, e.g.

AND operator



(x != 0)	&&	(1000/x > y)
false		
true		false
true		true

OR operator



(x == 0)	||	(1000/x > y)
true		
false		false
false		true

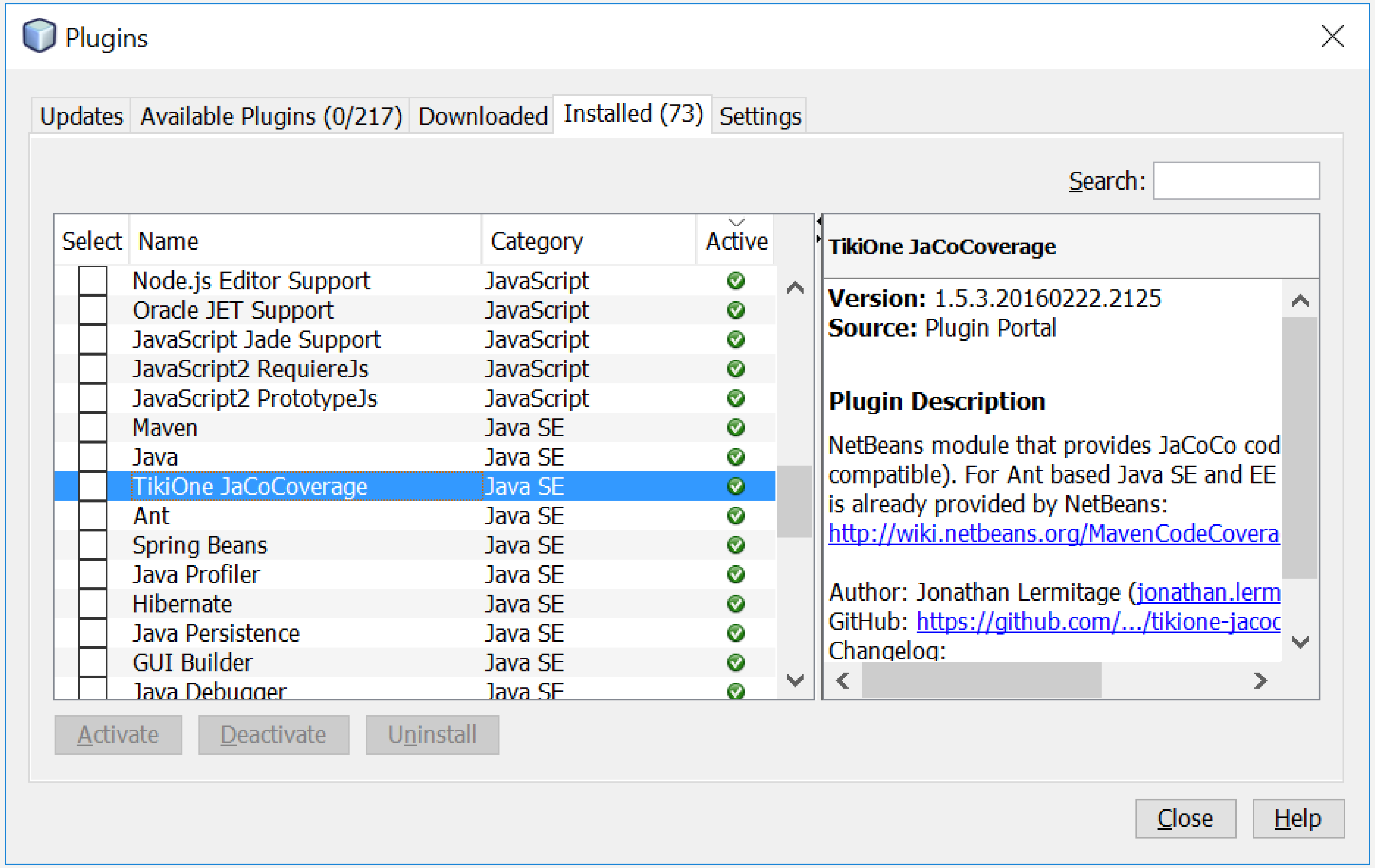
CODE COVERAGE TOOL

JaCoCo – a code coverage reports generator for Java projects

We need to declare this maven plugin in our *pom.xml* file:

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.3</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

PLUGIN – WITHOUT MAVEN



Select	Name	Category	Active
☐	Node.js Editor Support	JavaScript	✓
☐	Oracle JET Support	JavaScript	✓
☐	JavaScript Jade Support	JavaScript	✓
☐	JavaScript2 RequireJs	JavaScript	✓
☐	JavaScript2 PrototypeJs	JavaScript	✓
☐	Maven	Java SE	✓
☐	Java	Java SE	✓
☐	TikiOne JaCoCoverage	Java SE	✓
☐	Ant	Java SE	✓
☐	Spring Beans	Java SE	✓
☐	Java Profiler	Java SE	✓
☐	Hibernate	Java SE	✓
☐	Java Persistence	Java SE	✓
☐	GUI Builder	Java SE	✓
☐	Java Debugger	Java SE	✓

TikiOne JaCoCoverage

Version: 1.5.3.20160222.2125
Source: Plugin Portal

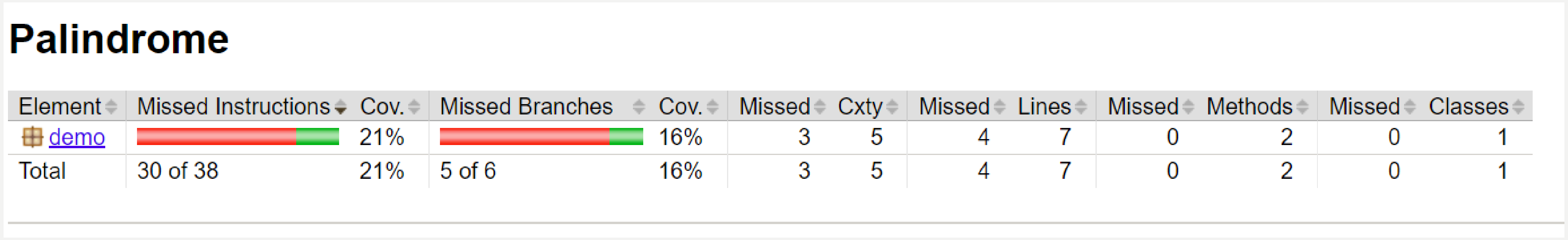
Plugin Description

NetBeans module that provides JaCoCo cod
compatible). For Ant based Java SE and EE
is already provided by NetBeans:
http://wiki.netbeans.org/MavenCodeCovera

Author: Jonathan Lermitage (jonathan.lerm
GitHub: https://github.com/.../tikione-jacoc
Changelog:

CODE COVERAGE REPORT

- Running the test using JUnit will automatically activate the JaCoCo agent
- Report page at *target/site/jacoco/index.html :*



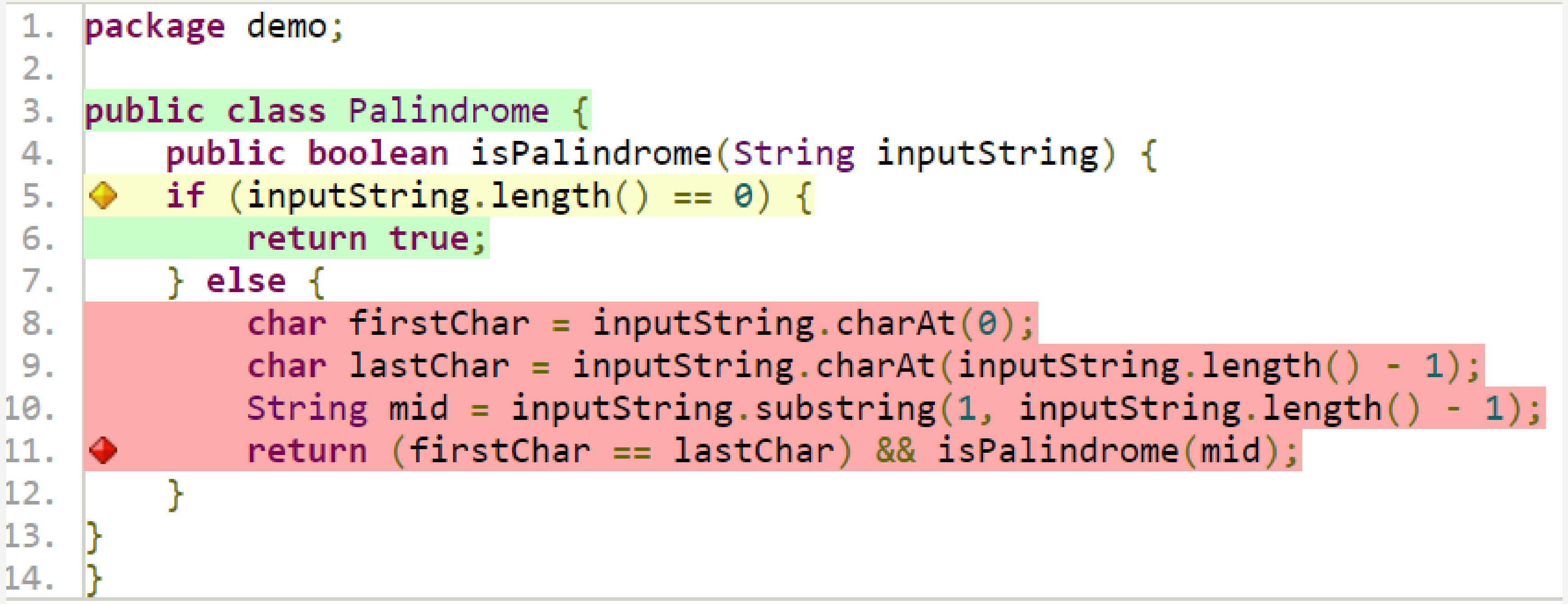
Palindrome

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
demo	🟥🟩	21%	🟥🟩	16%	3	5	4	7	0	2	0	1
Total	30 of 38	21%	5 of 6	16%	3	5	4	7	0	2	0	1

- Note:
  - The 38 instructions refer to the **bytecode instructions** (as opposed to ordinary Java statements)

DETAILED VIEW OF THE CODE

Following the link in the report we arrive at a more detailed view for each Java class:

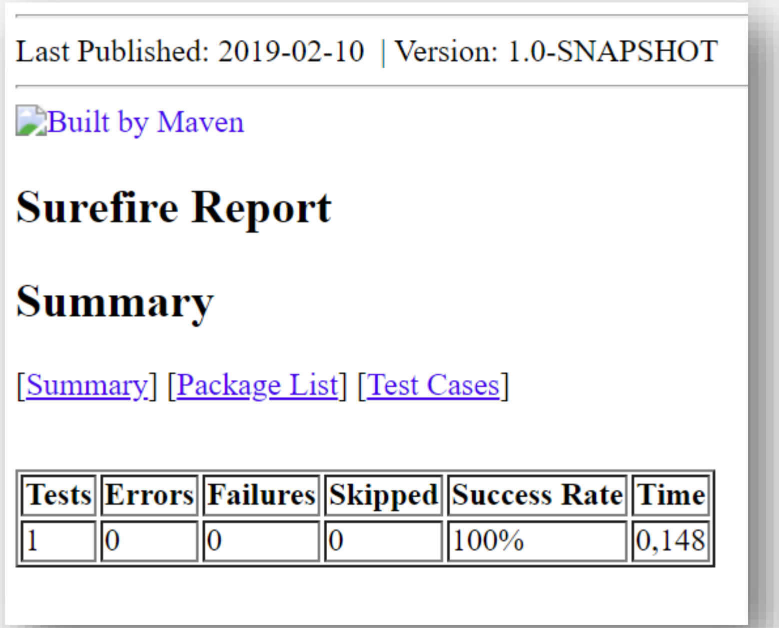


```java
package demo;

public class Palindrome {
    public boolean isPalindrome(String inputString) {
    if (inputString.length() == 0) {
        return true;
    } else {
        char firstChar = inputString.charAt(0);
        char lastChar = inputString.charAt(inputString.length() - 1);
        String mid = inputString.substring(1, inputString.length() - 1);
        return (firstChar == lastChar) && isPalindrome(mid);
    }
}
}
```

Red diamond means that no branches have been exercised during test
Yellow diamond shows that the code is partially covered
Green diamond means that all branches have been exercised during test
The same color code applies to the background color, but for lines coverage.

HTML JUNIT REPORT


Last Published: 2019-02-10 | Version: 1.0-SNAPSHOT

Built by Maven

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
1	0	0	0	100%	0,148

- HTML JUnit report with Maven is done by plugin

  `maven-surefire-report-plugin`

- By default, it is not attached to any core phases (validate, compile, test, package, integration test, … deploy)
- Instead we have to call it directly from command line:

```
mvn surefire-report:report
```

- Result is in the target/site directory (project documentation directory)

EXERCISE 1

Make a JUnit report with Maven
Make a code coverage report using JaCoCo

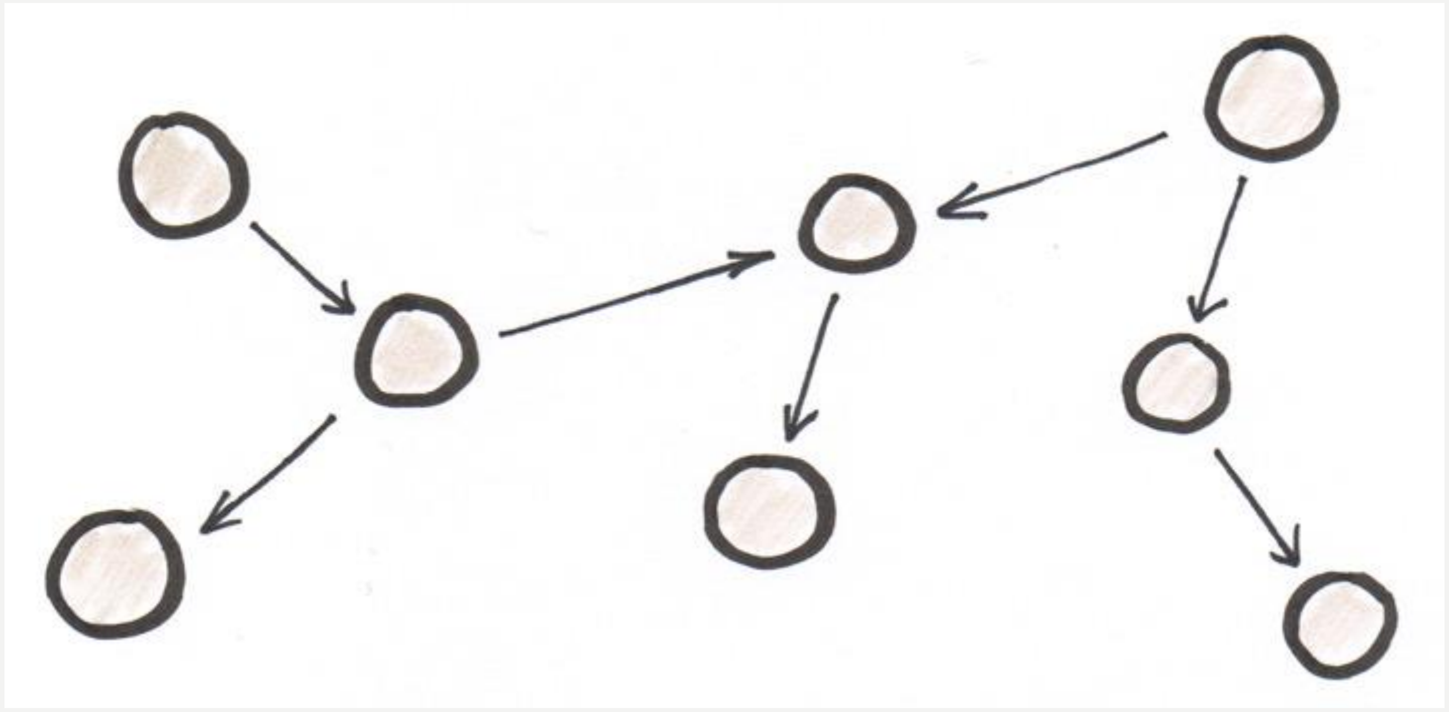
Use some arbitrary example project you have (e.g. your Triangle program) – we just need technical proof of concept. i.e. that you can make code coverage tool work on your own code

PAINFUL RESULTS FROM ACCIDENTLY LEAVING GAPS IN COVERAGE

HOW HIGH TEST COVERAGE?

80 %   ?

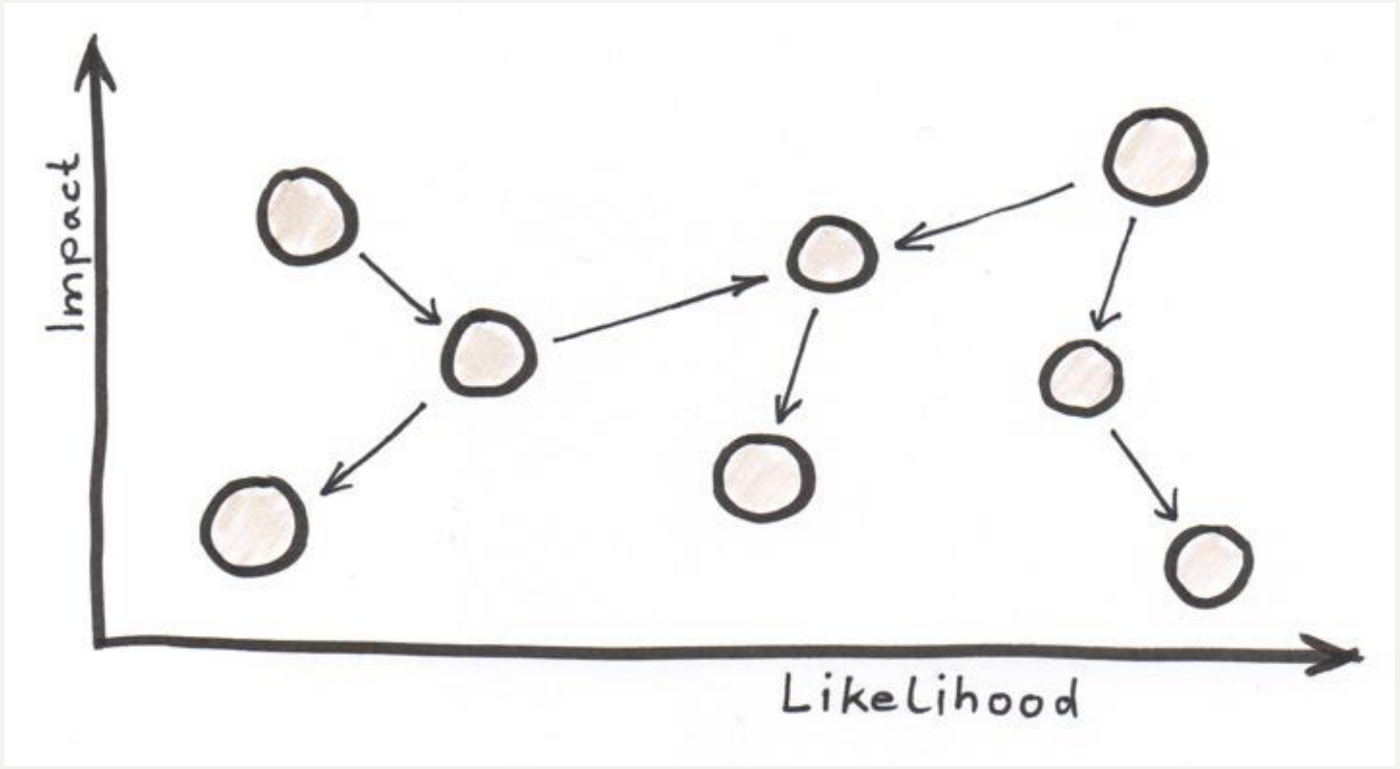
ILLUSTRATION OF SOFTWARE SYSTEM

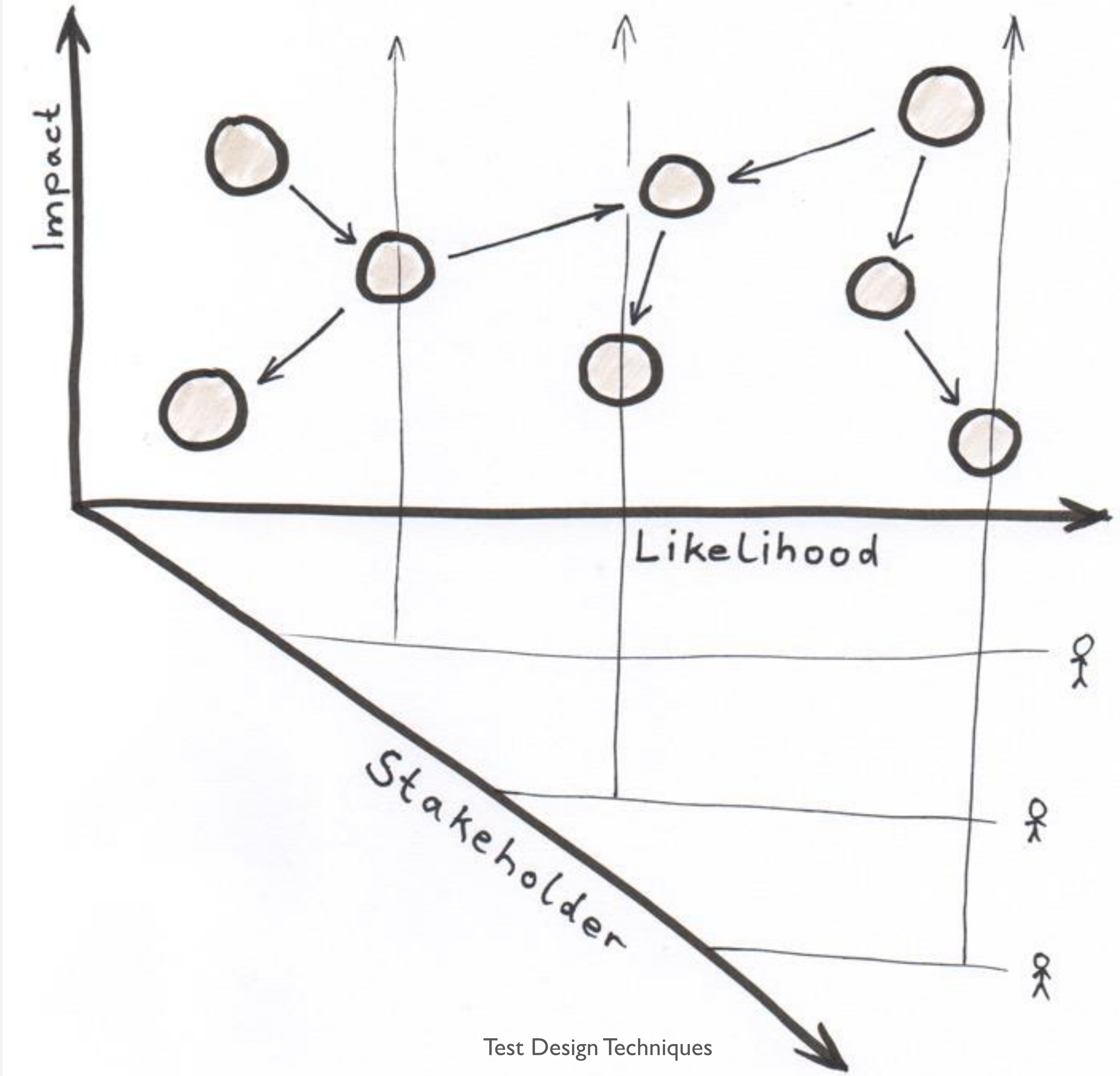


RISK PLANE

We add coordinate system of impact and likelihood

What do we want to test the most?




Impact
Likelihood
Stakeholder

COVERAGE ITEMS - EXAMPLES

- Typically code, but could be …
  - requirements, menu options or screens (system level)
  - interfaces (integration testing)
  - EP: percentage of equivalence partitions exercised (we could measure valid and invalid partition coverage separately if this makes sense).
  - BVA: percentage of boundaries exercised
  - Decision tables: percentage of business rules or decision table columns tested.
  - State transition testing: there are a number of possible coverage measures:
    - Percentage of states visited
    - Percentage of (valid) transitions exercised
    - …

CHOOSING TEST TECHNIQUES

- There are no best technique
- Specification-based testing can find missing things in the code (i.e. missing requirements)
- Structure-based can only test what is there already, i.e. test the quality of the existing code – including not used code, logic errors and variables initialized with the wrong values

EXERCISE 2

Experiment:
Try to run a test from Maven with different naming of the test class
Run both in IDE and from `mvn test`
Is there a difference in result (what is conclusion)?

Test at the start e.g. `TestNameClass`
Test at the end e.g. `NameClassTest`
Test in the middle e.g. `NameTestClass`
Without Test e.g. `NameClass`