SPE 2018 – Lecture 06

The busy student's introduction to testing

Dr. Daniel Schien

Daniel.schien@bristol.ac.uk

Recap

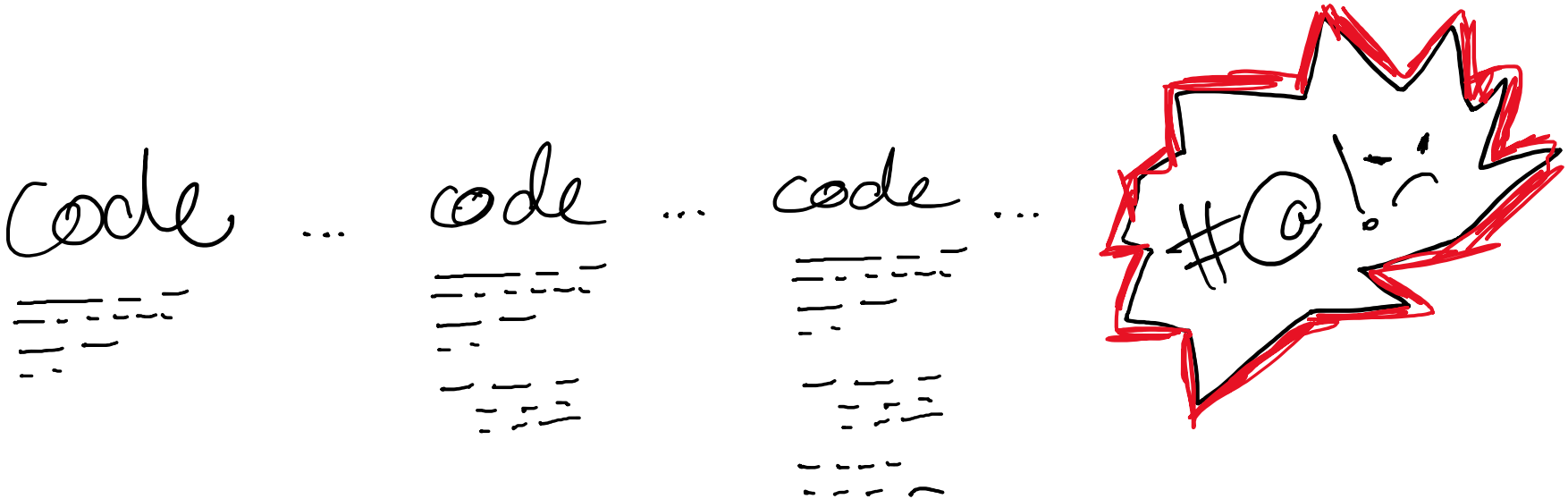
- Week 1 Introduction & The Open Project
- Week 2 Introduction to Agile & Agile Practices
- Week 3 CI & Validation and Verification
- Week 4 Requirements I & Requirements II

Le Menu

- What is verification and validation?
- What is testing in software engineering?
- Some practical tips to testing

/tɛst/

"A procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use." (Oxford Dictionary)



The Concepts

Subtle but Important Difference

Verification

"Are we building the product right"

Can we verify that a system is correct ?

The software must conform to its specification

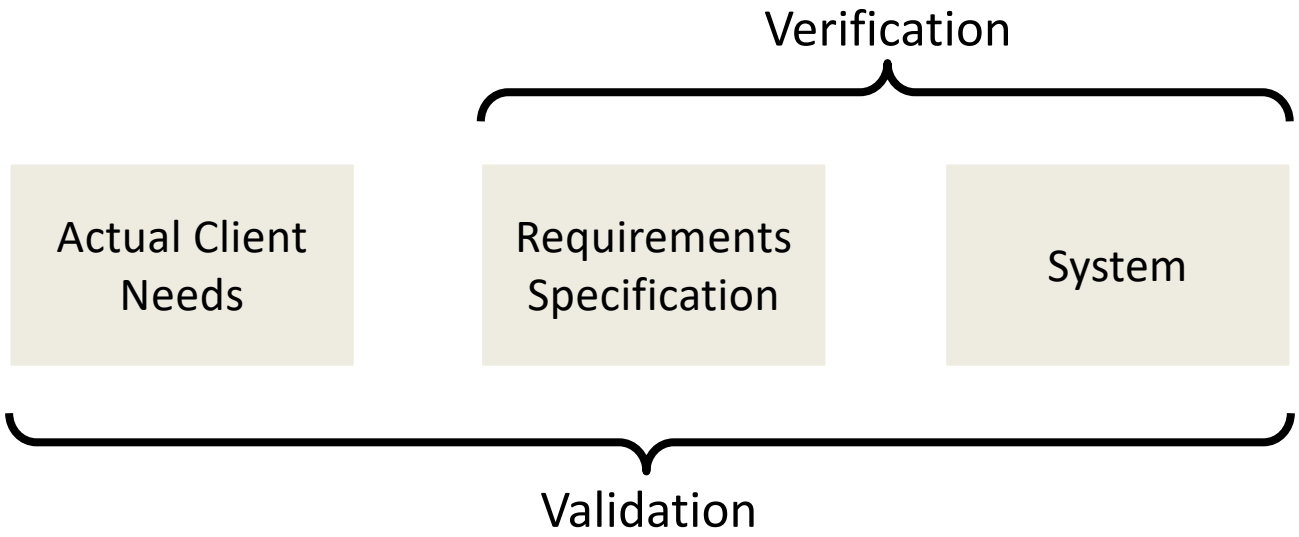
Validation

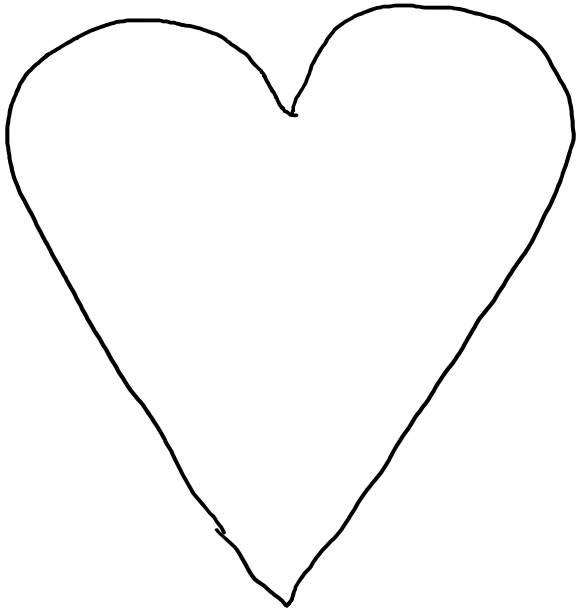
"Are we building the right product"

Is our system a valid solution ?

The software should do what users really want

Quality Control

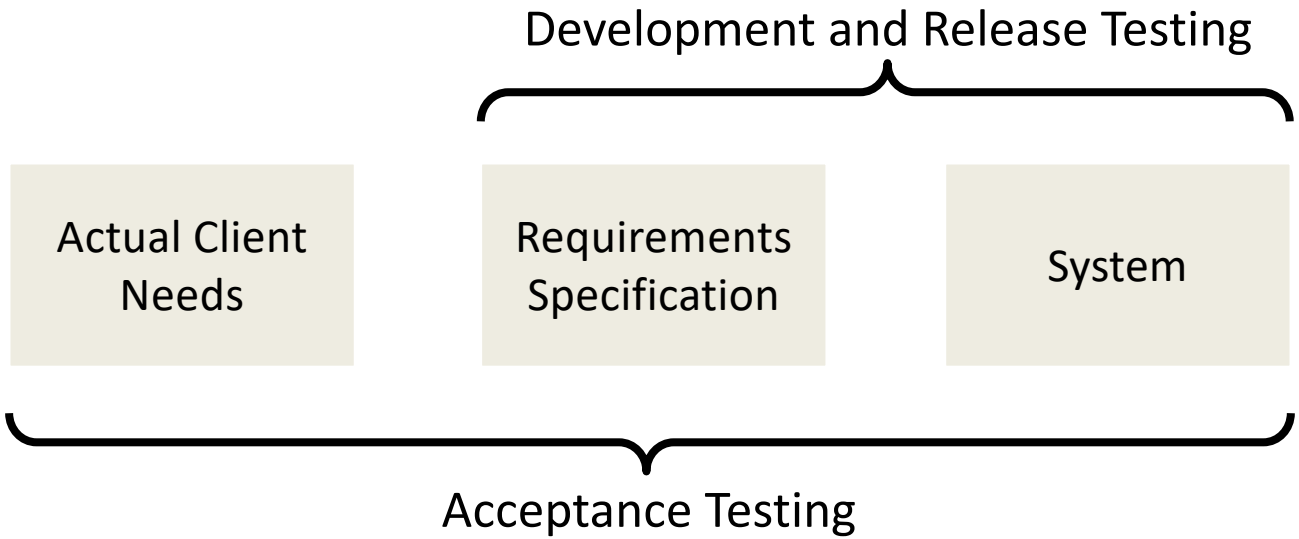


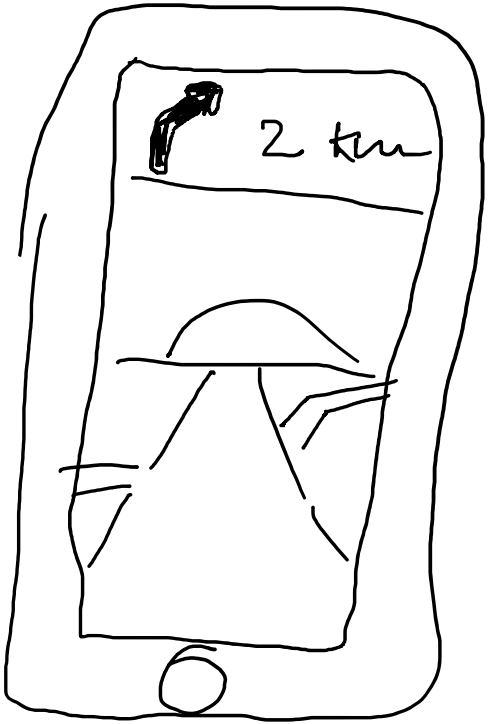


Testing as an approach to V&V

- Development testing: the system is tested during development to discover bugs and defects
- Release testing: a separate testing team test a complete version of the system before it is released to users.
- Acceptance testing: client and users of a system assess it within in their own environment.

Software Testing





Three Phases of Testing (Revisited)

- **Development testing**: the system is tested during development to discover bugs and defects
- Release testing: a separate testing team test a complete version of the system before it is released to users.
- Acceptance testing: client and users of a system assess it within in their own environment.

Nature of Development Testing

The following might seem a bit obvious (to such experienced devs) but worth stating anyway…

- When you test, you execute a program
- You “exercise” it using synthetic data
- Checking the results for errors or anomalies

- Aim of testing is to identify presence of defects
- A successful test is one that finds a defect !
- Absence of evidence is not evidence of absence !!!

Example

Consider a function that,

if given a day, month and year,

would tell you what day of the week it was

(Mon, Tues, Weds, Thurs, Fri, Sat, Sun)

How do we know if it is ***completely*** correct ?

monkey testing

Correctness & Completeness

- We could run program with a few arbitrary values
- But that probably won't reveal many errors
- It isn't methodical or systematic enough

- How to ensure complete & comprehensive test ?
- What we need is a proper, organised strategy…

Input-output specification

Black Box Testing

- Components are viewed as black boxes
- We can't see their internal implementation

- From reading the system specification
- We know the range of acceptable inputs
- And the corresponding correct outputs

- An actual output that doesn't match the expected output indicates existence of a defect !

Coverage versus Practicallity

- Functions have finite number of input parameters
- Complete black-box testing would try every possible combination of such import values

- This obviously isn't always practical…
- For the previous "day of the week" function, 100 years would require around 37k combinations !

- We need to identify a ***sample set*** of test cases
- Ensures coverage, without having to exhaustively try all combinations

inferred from the spec

Equivalence Partitioning

A technique to help systematic selection of test cases

An equivalence partition is:

"A cluster of input values for which a program should behave in the same way"

An example partition might be set of all +ve numbers
(up to max allowable in the programming language)

The number 54 is likely to have the same effect as 55 !

Selecting Test Data

For each partition, select upper & lower boundary

(Boundary values can be overlooked by coders)

Also choose a data value from middle of partition

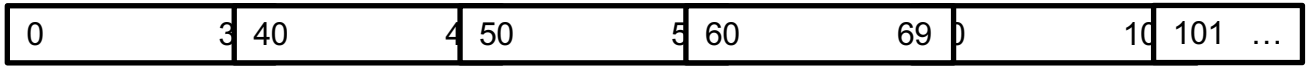
Should be representative of main body of values

In this way, we end up testing each partition, rather than every single possible value or combination

Example Equivalence Partitions

Imagine we had a function to convert a grade (as a percentage) into a degree classification

(1st, 2.1, 2.2. 3rd, Fail)



The equivalence partitions might be as follows:

The test case values might then be:

Exercise

For the “Day of the Week” program

What test cases would you use ?

Don’t forget to consider all 3 parameters

(day, month, year)

Interesting Test Cases

Day: -30, -1, 0, 1, 15, 27, 28, 29, 30, 31, 32, 40

Month: -12, -1, 0, 1, 6, 12, 13, 30

Year: -2017, -1, 0, 00, 1, 17, 69, 70, 1969, 1970, 2000

That's still 12 x 8 x 11 (over 1000) combinations !

Write Test Cases Before Coding

- You have already seen that “Tests first” is a key element of Agile and Test Driven development
- Act of creating test cases forces us to think about problem from a different (lower level) perspective
- This can help identify deficiencies in specification

White Box Testing

- Use knowledge of the code structure to define test cases
- Branch structure

- Has the potential to miss parts of the requirements

Model Driven Testing

- Part of model drive software engineering
- Test the model (domain experts)
- Test the model interpreter/converter

Typical Tabular Representation

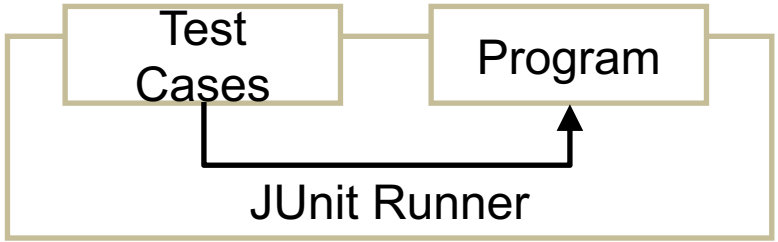
Test ID	Description	Input	Expected output

We can document test cases using a table

Where each row represents a particular test case

JUnit

- JUnit is a simple tool to aid unit testing
- Provides a framework which developers use to write and run their own test cases:



- When using JUnit, you write test cases in Java!
- Various user interfaces exist to support running of test cases and reporting their outcomes

Assertions

- JUnit uses “Assertions” to test the code
- Allow us to state what should be the case
- If assertions do not hold, JUnit’s logging mechanisms reports failed test cases

- Various assertions are available
- The most commonly used are:
  - assertEquals(expected, actual)
  - assertTrue(condition)
  - assertFalse(condition)

Examples

Given three integer variables:

int x = 1; int y = 10; int z = 100;

What are the outcomes of:

assertEquals(1, x);	PASS
assertEquals(x, z);	FAIL
assertTrue(Math.sqrt(y) > x);	PASS
assertFalse(x > y);	PASS
assertFalse(z != y);	FAIL

Magic ?

- There is nothing particularly magical about JUnit
- It doesn't do anything amazing - you could do it all manually by just writing and running test classes
- It does however provide a set of standards and conventions for doing things
- As well as some classes and support tools for automated and systematic checking
- It is also built into many IDEs

Three Phases of Testing (Revisited)

- Development testing: the system is tested during development to discover bugs & defects

- **Release testing**: a separate testing team test a complete version of the system before it is released to users.

- Acceptance testing: client and users of a system assess it within in their own environment.

Release Testing

- Release testing is the process of testing a particular release of a system that is intended for actual use
- Primary goal is to convince the customer that the system is good enough for use
- Clearly, required functions need to be present
- But we must also show that system delivers specified performance and dependability
- Most importantly we need to demonstrate that the system

Use-Case Driven Testing

- Use-cases identified during requirements & design phases can be used as a basis for release testing
- Each goal may involve several components
- Their execution often involves complex interactions
- Which we should really be attempting to test !
- Testing all goals also helps to ensure full coverage

Performance Testing

- ***Release testing*** may involve testing of emergent properties such as reliability & performance
- Tests ***normally*** reflect the likely usage profile…
- ***Performance testing*** involves steadily increasing load to identify point where system cannot cope
- ***Stress testing*** deliberately overloads system to test how gracefully it handles failure

Nature of Test Cases

- Release testing is higher-level than unit testing
- Test cases are likely to involve complex data
- Running test cases is likely to be a manual activity
- However automate wherever possible !
- Tools like CURL and Python “Requests” can help

Three Phases of Testing (Revisited)

- Development testing: the system is tested during development to discover bugs and defects
- Release testing: a separate testing team test a complete version of the system before it is released to users.
- **Acceptance testing**: client and users of a system assess it within in their own environment.

Acceptance Testing

- Essential, even after comprehensive release testing
- Client tests a system to decide if it is ready for use
- MUST be done by REAL users in REAL environment
- Influences from user environment have an effect on reliability, performance, usability, robustness etc
- These cannot be replicated by testing “in the lab”

Demo Time

A TemplateEmailService

Dear **{0}**,

Thank you for your application to the **{1}** degree program at the Department of Computer Science at the University of Bristol.

I am happy to be able to tell you that you your application was successful.

Best regards,

HoS MVSE

+

Email	Name	Program
crazybot@gmail.com	Fritz	Meng
walker89@hotmail.com	Hans	MSc

Dear **Fritz**,

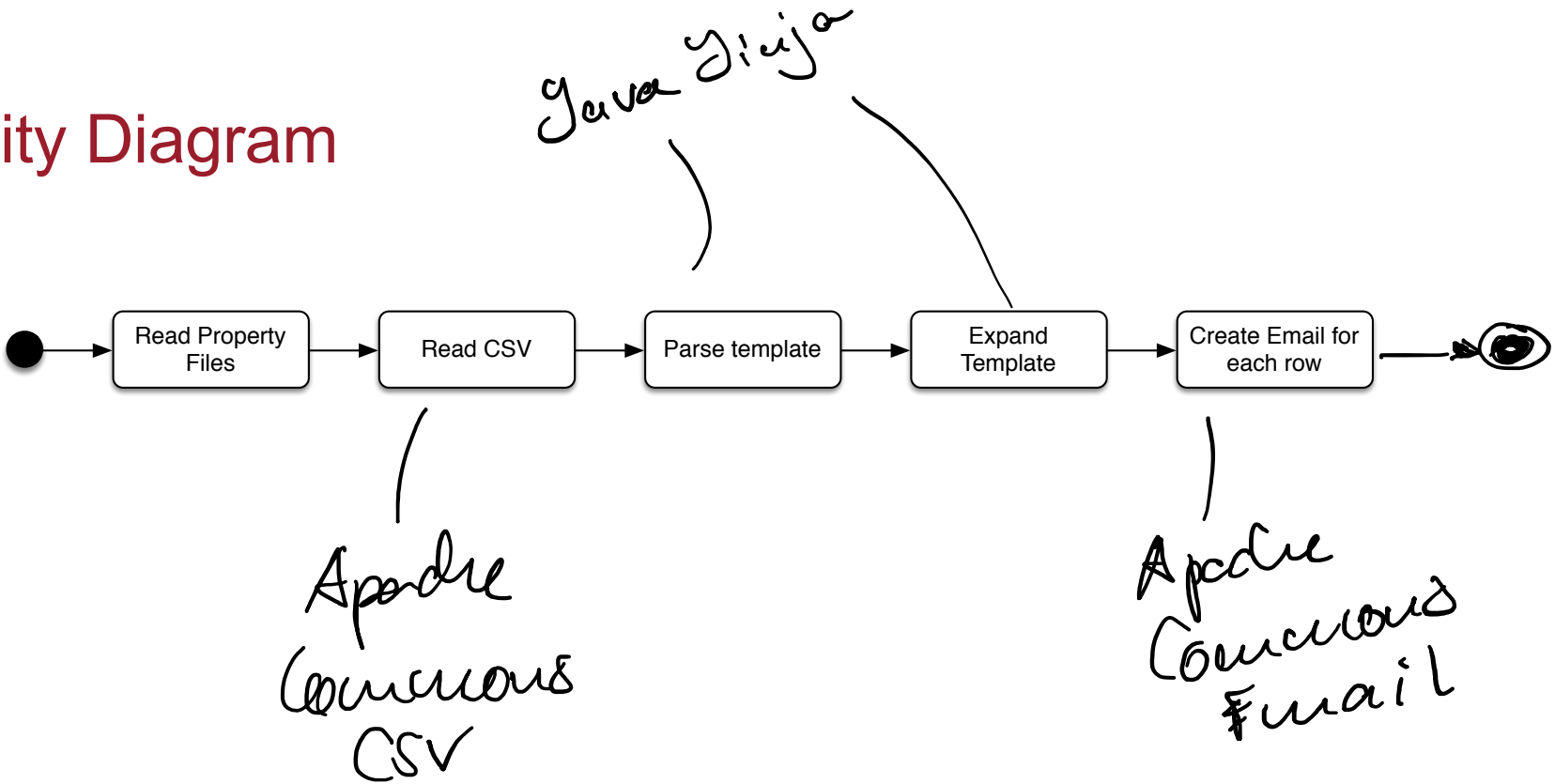
Thank you for your application to the **MEng** degree program at the Department of Computer Science at the University of Bristol.

I am happy to be able to tell you that you your application was successful.

Best regards,

HoS MVSE

Activity Diagram



Mockito

- *dummy object – not used, e.g. parameter not relevant in the test*
- *Fake – simplified implementations*
- *Stub – partial implementation*
- *Mock – a dummy that provides responses to certain calls*

Thank you for your attention