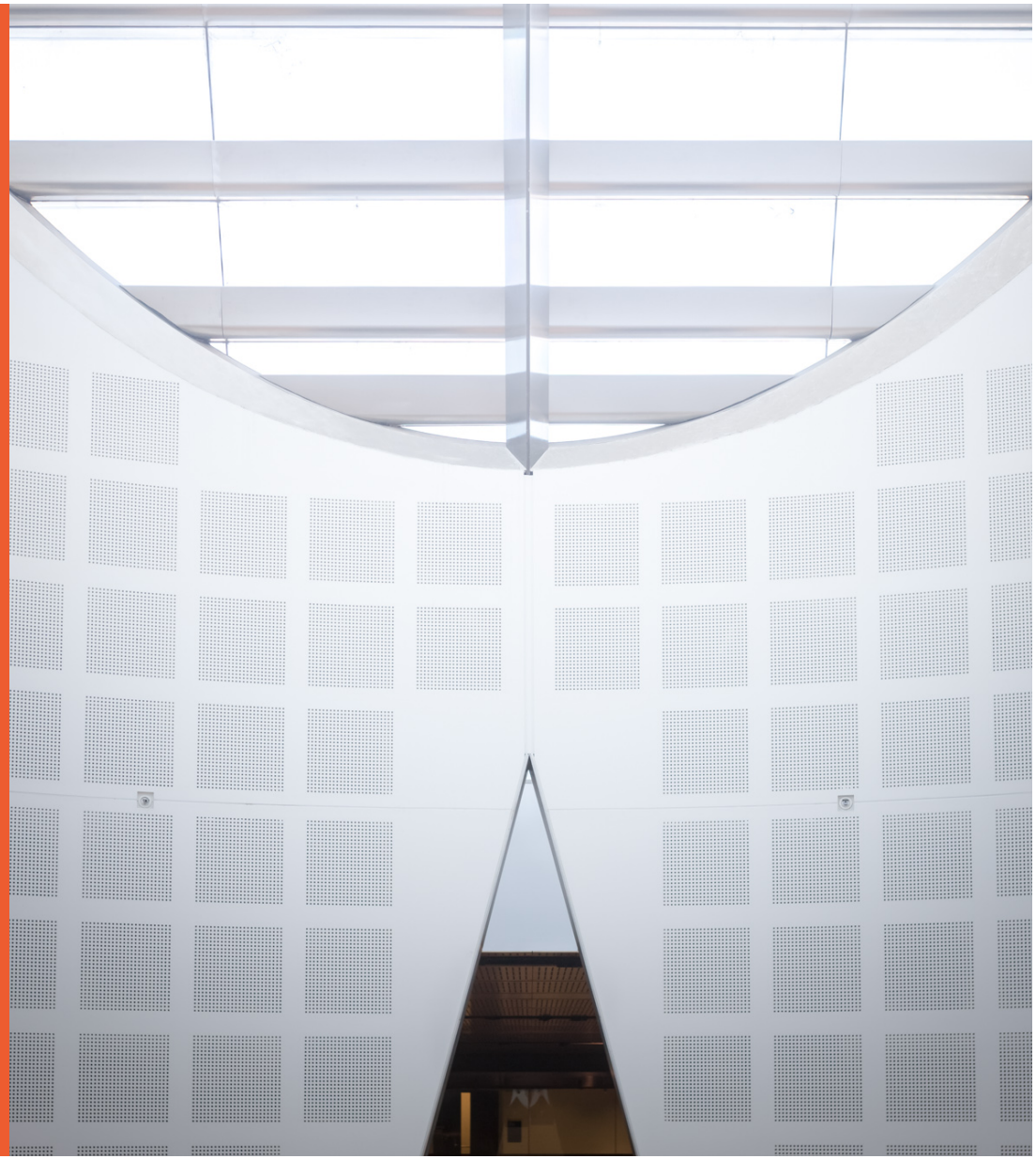
Software Design and Construction 2
SOFT3202 / COMP9202
Introduction
Software Testing

Prof Bernhard Scholz

School of Computer Science

THE UNIVERSITY OF SYDNEY



Agenda

- Workplace Health and Safety

- Unit/Course arrangements and Information
  - Unit details
  - Assessment
  - Expectations
  - Policies
- Assistance
- Advice

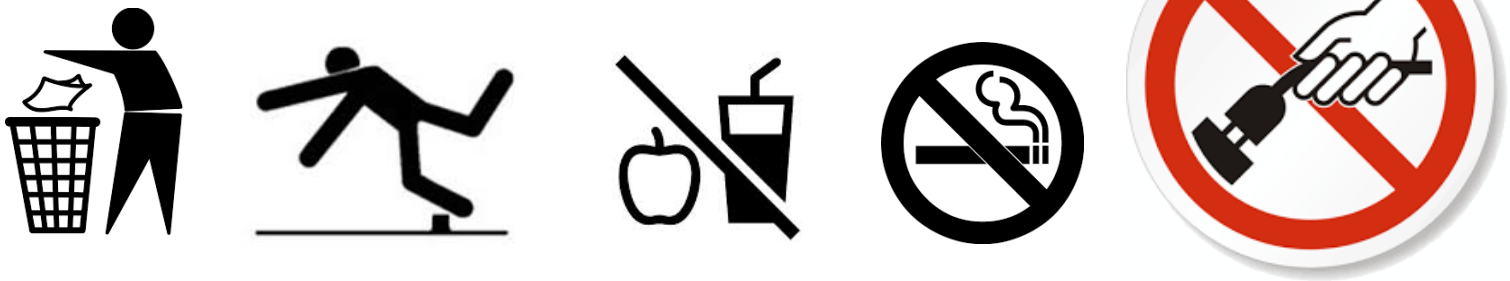
WHS INDUCTION

School of Computer Science

THE UNIVERSITY OF SYDNEY

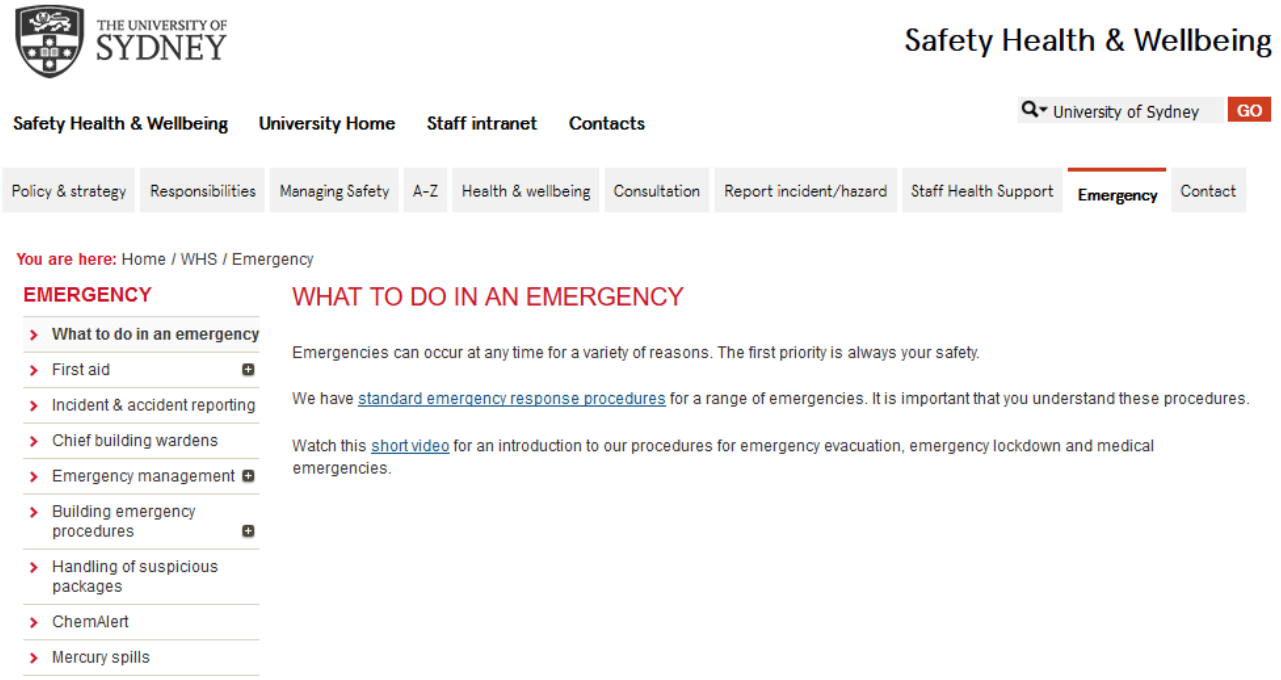
General Housekeeping – Use of Labs

- Keep work area clean and orderly
- Remove trip hazards around desk area
- No food and drink near machines
- No smoking permitted within University buildings
- Do not unplug or move equipment without permission



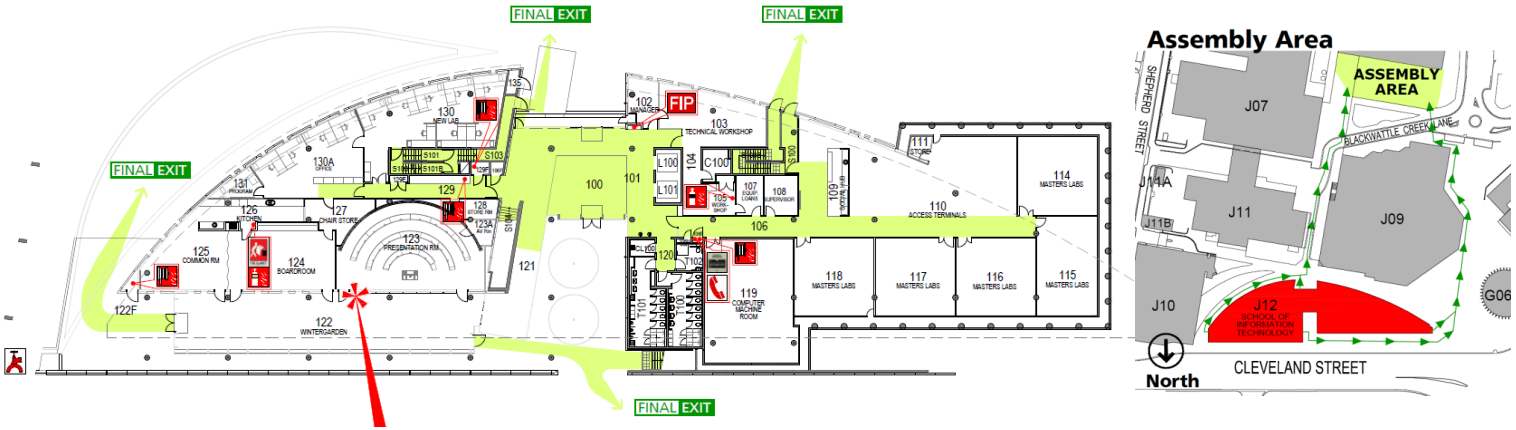
EMERGENCIES – Be prepared

www.sydney.edu.au/whs/emergency



EMERGENCIES

WHERE IS YOUR CLOSEST SAFE EXIT ?



EMERGENCIES

Evacuation Procedures

ALARMS

BEEP... BEEP... Prepare to evacuate

1. Check for any signs of immediate danger.
2. Shut Down equipment / processes.
3. Collect any nearby personal items.

WHOOP... WHOOP... Evacuate the building

1. Follow the EXIT exit signs.
2. Escort visitors & those who require assistance.
3. DO NOT use lifts.
4. Proceed to the assembly area.

EMERGENCY RESPONSE

1. Warn anyone in immediate danger.
2. Fight the fire or contain the emergency, if safe & trained to do so.

If necessary...

3. Close the door, if safe to do so.
4. Activate the "Break Glass" Alarm  or 
5. Evacuate via your closest safe exit. EXIT 
6. Report the emergency to 0-000 & 9351-3333

MEDICAL EMERGENCY

- If a person is seriously ill/injured:

1. **call an ambulance 0-000**
2. **notify the closest Nominated First Aid Officer**

If unconscious– send for Automated External Defibrillator (AED) AED **locations**.

NEAREST to CS Building (J12)
- Electrical Engineering Building, L2 (ground) near lifts
- Seymour Centre, left of box office
- Carried by all Security Patrol vehicles

3. **call Security -** 9351-3333
4. **Facilitate the arrival of Ambulance Staff (via Security)**

Nearest Medical Facility
University Health Service in Level 3, Wentworth Building

First Aid kit – SIT Building (J12)
kitchen area adjacent to Lab 110

School of Computer Science Safety Contacts

CHIEF WARDEN

Greg Ryan
Level 1W 103
9351 4360
0411 406 322

FIRST AID OFFICERS

Julia Ashworth
Level 2E Reception
9351 3423

Will Calleja
Level 1W 103
9036 9706
0422 001 964

Katie Yang
Level 2E 237
9351 4918

Orally REPORT all INCIDENTS & HAZARDS to your SUPERVISOR

Undergraduates: to Katie Yang
9351 4918

Coursework Postgraduates: to Cecille Faraizi
9351 6060

CS School Manager:
9351 4158

Do you have a disability?

You may not think of yourself as having a ‘disability’ but the definition under the **Disability Discrimination Act (1992)** is broad and includes temporary or chronic medical conditions, physical or sensory disabilities, psychological conditions and learning disabilities.

The types of disabilities we see include:
Anxiety // Arthritis // Asthma // Autism // ADHD
Bipolar disorder // Broken bones // Cancer
Cerebral palsy // Chronic fatigue syndrome
Crohn’s disease // Cystic fibrosis // Depression Diabetes // Dyslexia // Epilepsy // Hearing impairment // Learning disability // Mobility impairment // Multiple sclerosis // Post-traumatic stress // Schizophrenia // Vision impairment
and much more.

Students needing assistance must register with Disability Services. It is advisable to do this as early as possible. Please contact us or review our website to find out more.

THE UNIVERSITY OF SYDNEY

Disability Services Office
sydney.edu.au/disability
02-8627-8422

Other Support

- Learning support
  - http://sydney.edu.au/study/academic-support/learning-support.html
- International students
  - http://sydney.edu.au/study/academic-support/support-for-international-students.html
- Aboriginal and Torres Strait Islanders
  - http://sydney.edu.au/study/academic-support/aboriginal-and-torres-strait-islander-support.html
- Student organization (can represent you in academic appeals etc)
  - http://srcusyd.net.au/ or http://www.supra.net.au/
- Please make contact, and get help
- You are not required to tell anyone else about this
- If you are willing to inform the unit coordinator, they may be able to work with other support to reduce the impact on this unit
  - eg provide advice on which tasks are most significant

Assistance

- There are a wide range of support services available for students
- Please make contact, and get help
- You are not required to tell anyone else about this
- If you are willing to inform the unit coordinator, they may be able to work with other support to reduce the impact on this unit
  - E.g., provide advice on which tasks are most significant

Special Consideration (University policy)

- If your performance on assessments is affected by illness or misadventure
- Follow proper bureaucratic procedures
  - Have professional practitioner sign special USyd form
  - Submit application for special consideration online, upload scans
  - Note you have only a quite short deadline for applying
  - http://sydney.edu.au/current_students/special_consideration/
- Also, notify coordinator by email *as soon as anything begins to go wrong*
- There is a similar process if you need special arrangements, e.g., for religious observance, military service, representative sports

Academic Integrity (University policy)

- "The University of Sydney is unequivocally opposed to, and intolerant of, plagiarism and academic dishonesty."
  - Academic dishonesty means seeking to obtain or obtaining academic advantage for oneself or for others (including in the assessment or publication of work) by dishonest or unfair means.
  - Plagiarism means presenting another person's work as one's own work by presenting, copying or reproducing it without appropriate acknowledgement of the source." [from site below]
- http://sydney.edu.au/elearning/student/EI/index.shtml
- Submitted work is compared against other work (from students, the internet, etc)
  - Turnitin for textual tasks (through eLearning), other systems for code
- Penalties for academic dishonesty or plagiarism can be severe
- Complete self-education AHEM1001 (required)

Unit/Course Overview

THE UNIVERSITY OF SYDNEY

SOFT3202/COMP9202: Lectures and Tutorials

- Lecture
  - Wednesday, 3-5pm, online
- Labs/Tutorials
  - SOFT3202: 2h tutorial, slots between 10am-4pm on Mondays
  - COMP9202: 2h tutorial, 10am-noon, Fridays
  - Check your timetable
    - Attend ONE (go to the lab you're scheduled for)
  - Lab Schedule and material on Canvas
    - Starting from week 2
  - Hands-on practical exercises
    - Try it, debug and fix issues (experience)
    - Tutors to supervise your learning and provide guidance

SOFT3202/COMP9202: Staff

Course Coordinator and Lecturer:

- Prof Bernhard Scholz (bernhard.scholz@sydney.edu.au)
  - Consultation: Monday, 4-5 PM by arrangement
  - School of Computer Science (J12), Room 352
- Teaching Assistant
  - Joshua Burridge (jbur2821@uni.sydney.edu.au)
- Tutors
  - Martin McGrane (mmcg5982@uni.sydney.edu.au)
  - Tyson Thomas (ttho6664@uni.sydney.edu.au)

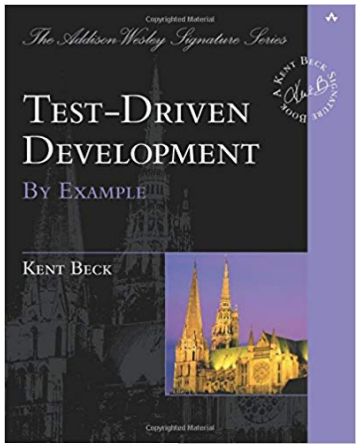
SOFT3202/COMP9202: Resources

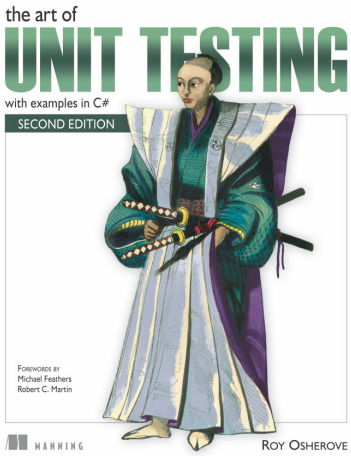
- Canvas (eLearning)
  - https://canvas.sydney.edu.au/
  - Copies of lecture slides
  - Lab instructions
  - Assignment instructions
  - Lecture recordings
    - We intend to record the lectures, but the technology may not be reliable
  - *Submit official assignment work here*;
  - Gradebook
  - Discussion (Q&As) on Edstem

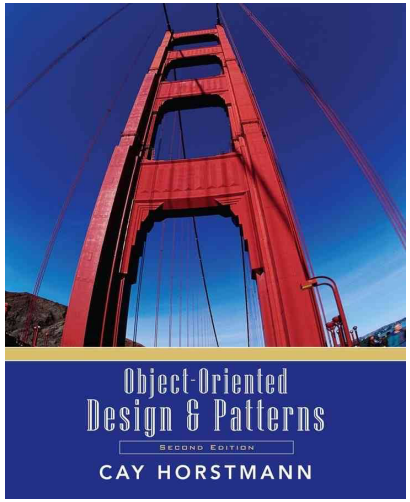
Prerequisites

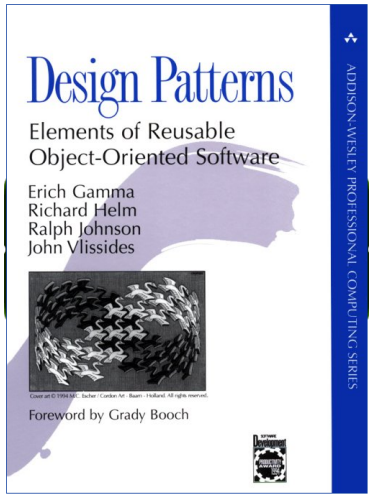
- This course has the following prerequisites:
  - SOFT2201
- This means that we expect students who enroll in this course to be:
  - Confident Java programmers
  - Good with OO modeling (UML)
  - Good with design patterns
- Prohibitions
  - INFO3220 OR COMP9202

Resources/References









Topics Overview

Week	Contents
1	Introduction; Software Testing, Software Verification and Validation
2	Theory of Testing: Design of tests, test coverage metrics and tools
3	Advanced testing techniques 1
4	Advanced testing techniques 2
5	Software verification theory
5-12	Review of GoF design patterns, advanced design patterns (GoF creational, structural, and behavioral), enterprise design patterns
13	Review, exam structure

Check Unit of Study outline for latest version

Lab / Tutorial Work

- Lab work will be available on Canvas
- 2-hour lab/tutorial work!
  - Check your schedule and allocation on the timetable
- Great opportunity for interactive and hands-on learning experience
- Weekly quizzes/exercises (week 2-11)
- Respect your tutors and value their feedback
- Respect your classmates
- Tutors will supervise your learning, provide you guidance
  - Not to debug your code, or solve the problems for you

Language and Tools

- Your coding will be in Java 11
- You will use IntelliJ for writing your code
- You will use supporting tools such as Junit, Gradle and Mockito
- Other?

- Note: feel free to use other IDEs

Feedback

- Talk to us (e-mail) if
  - You have problems or are struggling,
  - You can't understand the contents,
  - You become ill and can't make a tutorial, or
  - You think there's something else wrong

- A discussion forum is setup:
  - We are using ED for technical discussions only
  - Please don’t use ED for personal problems/matters!
  - Inappropriate posts will be removed

Assessment

What (Assessment)*	When (due)	How	Value
Weekly Quizzes/exercises	Weekly (2-11)	Online Quizzes (Canvas)	10%
Testing Assignment A1	Week 4	Individual submission using github	10%
Testing Assignment A2	Week 8	Individual submission using github	20%
Exam	Exam period	Individual exam	60%

Weekly Quiz/exercise

- Individual work/exercise demonstration and explanation
- When: multiple weeks, during your lab/tutorial
  - Week 2-11 *unless otherwise announced*
  - You need to stay in your assigned lab/tutorial
- Closed book: No notes/other teaching material are permitted
- Must present your student card to your tutors
- Covers recent previous week lecture and current tutorial
- Total marks: 10%

Assignments

- Two individual assignments (30% marks)

    1. Software Testing Assignments A1 (10%)
    2. Software Testing Assignment A2 (20%)

Assignments Feedback

- Testing assignments
  - Test component(s) of an ERP application (A1)
  - Extend your test to cover additional components of the ERP application (A2)
  - Automated marking – strictly follow the submission requirements
  - Tutor's feedback

- Design Pattern Assignment / Exam
  - Exam will be a longer coding / essay style exercise
  - Instructions will be handed out in the second-half of the semester

Late Assessments

- We cannot accept late submissions for A1 and A2
- Special consideration possible up to a week
- Beyond this we can only do mark adjustments

Passing this unit*

- To pass this unit you must do all of these:
  - Get a total mark of at least 50%
  - Get at least 40% for your exam mark

SOFT3202/COMP9202: Expectations

- Students attend scheduled classes, and devote an *extra* 6-8 hours per week
  - Prepare and review lecture lab materials
  - Revise and integrate the ideas
  - Carry on the required assessments
  - Practice and self-assess
- Students are responsible learners
  - Participate in classes, constructively
    - Respect for one another (criticize ideas, not people)
    - Humility: none of us knows it all; each of us knows valuable things
  - Check eLearning site very frequently!
  - Notify academics whenever there are difficulties
  - Notify group partners honestly and promptly about difficulties
- Know the uni/school policies

Feedback to you!

- When you submit work, we have to mark it;
- We try to make this feedback as fast as possible
- Progressive marks will be recorded on Canvas

Feedback to you will take many forms:

- *Verbally by your tutor*
- *As comments accompanying hand marking of your assignment work and automated quiz answers*

- *Do pay attention to this feedback, it's expensive stuff.*

Special Consideration (University Policy)

- If your performance on assessments is affected by illness or misadventure
- Follow proper bureaucratic procedures
  - Have professional practitioner sign special USyd form
  - Submit application for special consideration online, upload scans
  - Note you have only a quite short deadline for applying
  - http://sydney.edu.au/current_students/special_consideration/
- Also, notify the coordinator by email *as soon as anything begins to go wrong*
- There is a similar process if you need special arrangements e.g., for religious observance, military service, representative sports

Advice for doing well in this unit

- To do *well* in this unit you should
  - Organize your time well
  - Devote 6-8 hours a week in total to this unit
  - Read.
  - Think.
  - Practice.

> "*Tell me and I forget, teach me and I may remember, involve me and I learn.*" – Benjamin Franklin

Advice

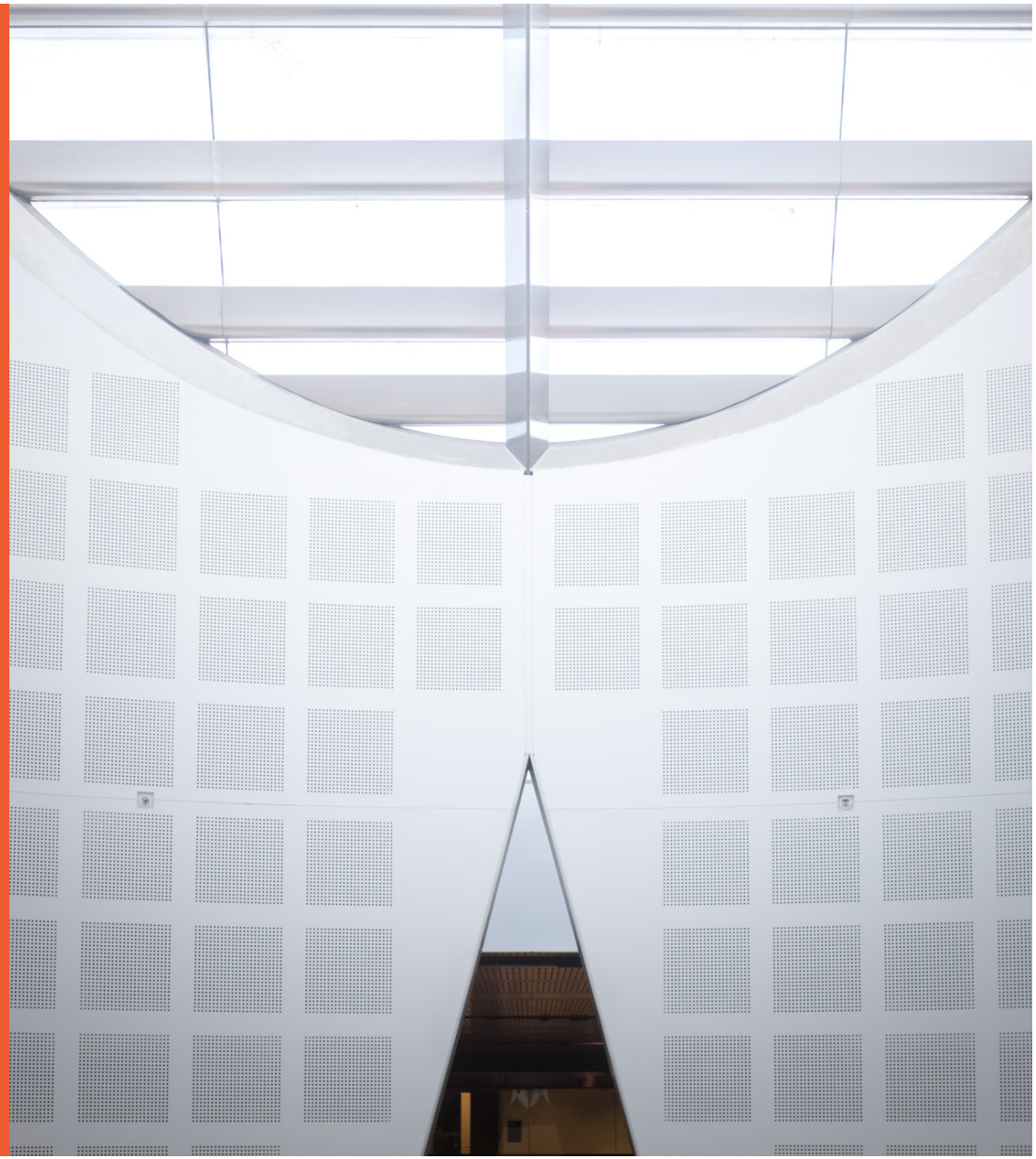
- Metacognition
  - Pay attention to the learning outcomes
  - Self-check that you are achieving each one
  - Think how each assessment task relates to these
- Time management
  - Watch the due dates
  - Start work early, submit early
- Networking and community-formation
  - Make friends and discuss ideas with them
  - Know your tutor, lecturer, coordinator
  - Keep them informed, especially if you fall behind
    - Don't wait to get help
- Enjoy the learning!

Software Design and Construction 2
SOFT3202 / COMP9202
Introduction
Software Testing

Prof Bernhard Scholz

School of Computer Science

THE UNIVERSITY OF SYDNEY



Agenda

- Software Engineering
- Software Testing
- Unit Testing

Testing in Software Engineering

THE UNIVERSITY OF SYDNEY

Software is Everywhere!

- Societies, businesses and governments dependent on SW systems
  - Power, Telecommunication, Education, Government, Transport, Finance, Health
  - Work automation, communication, control of complex systems
- Large software economies in developed countries
  - IT application development expenditure in the US more than $250bn/year[1]
  - Total value added GDP in the US[2]: $1.07 trillion
- Emerging challenges
  - Security, robustness, human user-interface, and new computational platforms

[1] Chaos Report, Standish group Report, 2014
[2] softwareimpact.bsa.org

Software Engineering Body of Knowledge

- Software Requirements
- **Software Design / Modelling**
- **Software Construction**
- **Software Testing**
- Software Maintenance
- Software Configuration Management
- Software Engineering Process
- Software Engineering Tools and Methods
- Software Quality

Software Engineering Body of Knowledge (SWEBOK) https://www.computer.org/web/swebok/

Why Software Engineering?

Need to build high quality software systems under resource constraints

- Social
  - Satisfy user needs (e.g., functional, reliable, trustworthy)
  - Impact on people's lives (e.g., software failure, data protection)
- Economical
  - Reduce cost; open up new opportunities
  - Average cost of IT development ~$2.3m, ~$1.3m and ~$434k for large, medium and small companies respectively[3]
- Time to market
  - Deliver software on-time

[3] Chaos Report, Standish group Report, 2014

Software Quality Assurance

- Quality (products)
  - "Fitness for use"
- Software quality
  - Satisfying end use's needs; correct behaviour, easy to use, does not crash, etc.
  - Easy to the developers to debug and enhance
- Quality Assurance (QA)
  - Processes and standards that lead to manufacturing high quality products
- Software Quality Assurance
  - Ensuring software under development have high quality and creating processes and standards in organization that lead to high quality software
  - Software quality is often determined through Testing

Juran and Gryna 1998

Software Testing

Why software testing?

THE UNIVERSITY OF SYDNEY

Therac-25 Overdose*

- **What happened?**
  - Therac-25 radiation therapy machine
  - Patients exposed to overdose of radiation (100 times more than intended)
  - Cost three lives!!

- **Why Did happen?**
  - Particular nonstandard sequence of keystrokes was entered within 8 seconds
  - Operator override a warning message with error code ("MALFUNCTION" followed by a number from 1 to 64) - not explained in the user manual
  - Software checks safety replacing hardware interlocks in previous Therac versions
  - Absence of independent software code review
  - software and hardware integration has never been tested until assembled in the hospital (*Big Bang Testing*')

*https://en.wikipedia.org/wiki/Therac-25#Problem_description

Software Failure - Ariane 5 Disaster[5]

What happened?

- European large rocket - 10 years development, ~$7 billion
- Unmanaged software exception resulted from a data conversion from 64-bit floating point to a 16-bit signed integer
- Backup processor failed straight after using the same software
- Exploded 37 seconds after lift-off

Why did it happen?

- *Inadequate validation and verification, testing and reviews*, design error, incorrect analysis of changing requirements, ….

[5] http://iansommerville.com/software-engineering-book/files/2014/07/Bashar-Ariane5.pdf

Nissan Recall - Airbag Defect*

- **What happened?**
  - ~ 3.53 million vehicles recall of various models 2013-2017
  - Front passenger airbag may not deploy in an accident
- **Why Did happen?**
  - Software that activates airbags deployment improperly classify occupied passenger seat as empty in case of accident
  - Software defect that could lead to improper airbag function (failure)
  - No warning that the airbag may not function properly
  - Software sensitivity calibration due to combination of factors (high engine vibration and changing seat status)

http://www.reuters.com/article/us-autos-nissan-recall/nissan-to-recall-3-53-million-vehicles-air-bags-may-not-deploy-idUSKCN0XQ2A8
https://www.nytimes.com/2014/03/27/automobiles/nissan-recalls-990000-cars-and-trucks-for-air-bag-malfunction.html

Software Project Failures

Project	Duration	Cost	Failure/Status
Pust Siebel - Swedish Police case management (Swedish Police)	2011 - 2014	$53m (actual)	Permanent failure – scraped due to poor functioning, inefficient in work environments
US Federal Government Health Care Exchange Web application	2013 – ongoing	$93.7m (expected), $1.5bn (actual)	Ongoing problems - too slow, poor performance, people get stuck in the application process (frustrated users)

https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects

Why Software Testing?

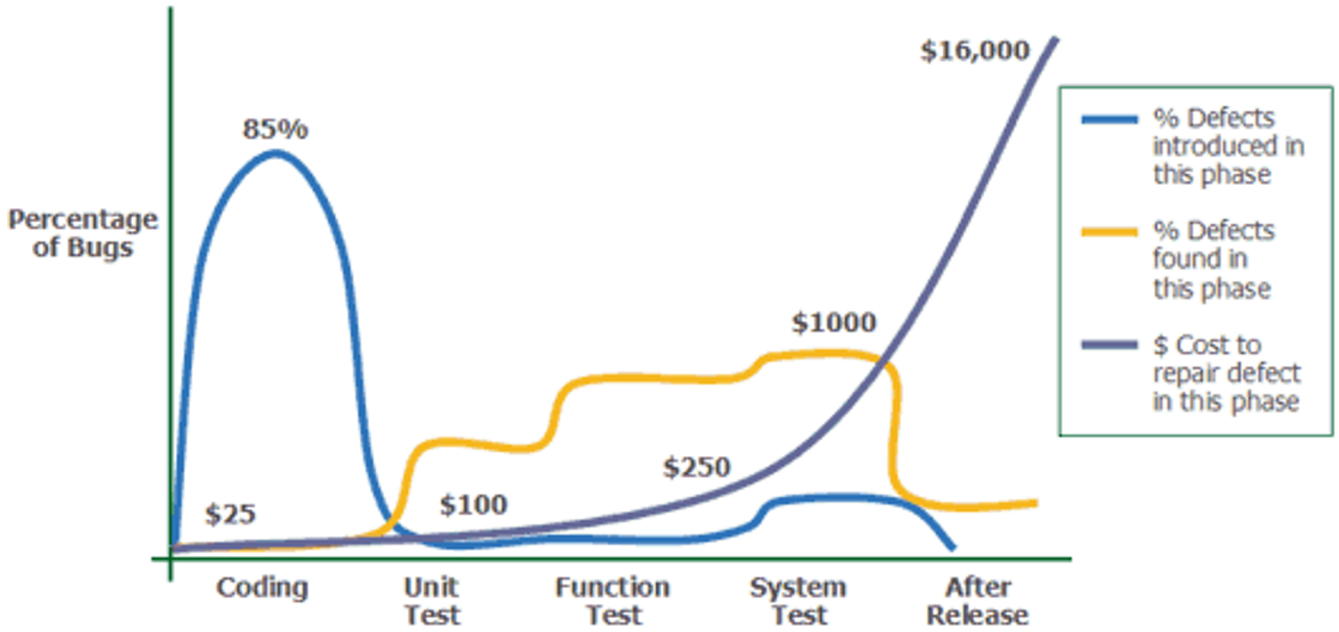
- Software development and maintenance costs
  - Big financial burden

- Total costs of inadequate software testing on the US economy is $59.5bn
  - NIST study 2002*
  - One-third of the cost could be eliminated by *improved software testing*

- Need to develop functional, robust and reliable software systems
  - Human/social factor - society dependency on software in every aspect of their lives
    - Critical software systems - medical devices, flight control, traffic control
  - Meet user needs and solve their problems
  - Small software errors could lead to disasters

* https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf

Software Testing - Costs



after release it is hard to fix bugs

Capers Jones, Applied software measurement (2nd ed.): assuring productivity and quality, (1997), McGraw-Hill

Software Testing

What is software testing?

THE UNIVERSITY OF SYDNEY

Software Testing

- Software process to
  - Demonstrate that software meets its requirements (*validation testing*)
  - Find incorrect or undesired behavior caused by defects/bugs (defect testing)
    - E.g., System crashes, incorrect computations, unnecessary interactions and data corruptions
- Part of software verification and validation (V&V) process

Testing (Levels)

Testing level	Description
Unit / Functional Testing	The process of verifying functionality of software components (functional units, subprograms) independently from the whole system
Integration Testing	The process of verifying interactions/communications among software components. Incremental integration testing vs. "Big Bang" testing
System Testing	The process of verifying the functionality and behaviour of the entire software system including security, performance, reliability, and external interfaces to other applications
Acceptance Testing	The process of verifying desired acceptance criteria are met in the system (functional and non-functional) from the user point of view

Unit Testing

THE UNIVERSITY OF SYDNEY

Unit Testing

- The process of verifying functionality of software components independently
  - System is broken down in small units
  - Unit → methods, functions or object classes
  - Validate that each functional unit behaves as expected (defect testing)
    - No formal verification required
    - Check for given input / output pairs only whether a unit test works
  - Carried out by developers / SW testers
    - Implemented in a test framework
  - First level of testing / defence
    - Assumption that if all units work it is more likely that the whole system works

Why Unit Testing?

- Change and maintain code at smaller scale
  - Isolation of the smallest piece of testable software
  - Programmer: maintain unit code, unit tests, and unit test infrastructure
- Provides a notion of documentation 
  - functionality of unit can be derived from unit test
- Discover defects early and fix it at cheaper costs
  - Identifies large percentages of defects
- Automation of testing processes
  - Use test infrastructures for unit testing
  - Continuous integration; delta debugging

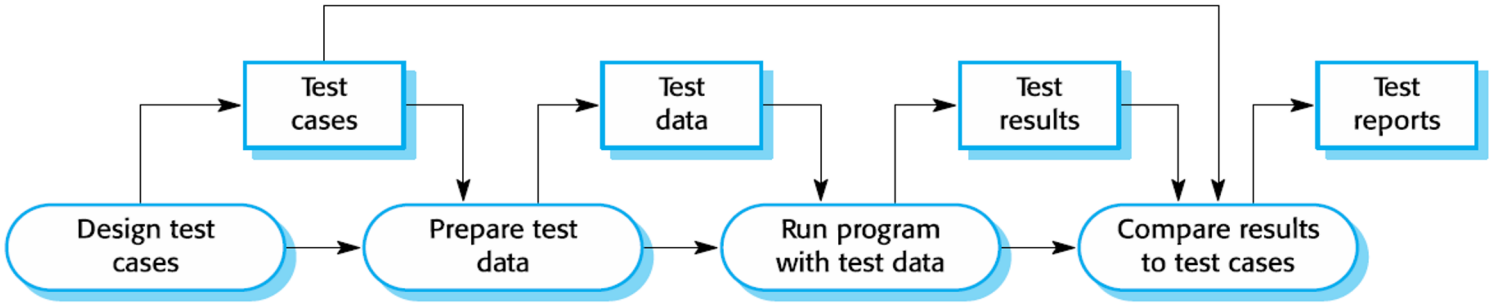
Why Unit Testing? (cont'd)

- Ease debugging
  - Reduces difficulty of discovering errors
    - Unit dependencies are avoided; isolation of bugs
  - Enhances test-coverage
    - Isolated achieve high coverage
    - Ideally, a unit test covers all statements in unit
- Code reusability
  - Makes programming agile
  - Improves code quality because of clear interfaces
- Simplifies integration testing
- **Reduces costs** of SW-Development and **increases productivity**

Unit Testing – How?

1. Design test cases
2. Prepare test data
3. Run test cases using test data
4. Compare results to test cases
5. Prepare test reports

Software Testing Process



Ian Sommerville. 2016. *Software Engineering* (10th ed.). Addison-Wesley, USA.

Designing Test Cases

- Effective test cases show:
  - The unit under test does what it supposed to do
  - Reveal defects in the unit, if there is any
  - Reflect the design intent of the unit

- Design two types of test cases
  - Test normal operation of the unit (positive test)
  - Test abnormality: common problems/defects (negative test)

Choosing Test Cases – Techniques

- Partition testing (aka. equivalence partitioning)
  - Identify groups of inputs that have common characteristics
  - From within each of these groups, choose a representative
  - Use program specifications, documentation and/or experience

- Guideline-based testing
  - Use testing guidelines based on previous experience of the kinds of errors often made
  - Cookbook-recipe style testing

Equivalence Partitioning

- Different groups of input with common characteristics
- Example code fragment

```
min(a,b): if (a < b)
              return a;
          else
              return b;
```

- Program behave in a comparable way for all members of a group, e.g.,
  - Group 1: min(1, 2); min(2,10), …
  - Group 2: min(10,1); min (5,2), …
- Choose test cases from each of the groups

Test Case Selection 

- Understanding developers thinking
  - Developers tend to think of typical values of input
  - Developers sometimes overlook atypical values of input, i.e., corner-cases

- Strategy for choosing test cases that are:
  - on the boundaries of the partitions, e.g.,

    min(10,10)

    min(-2^128,2^128)

  - close to the midpoint of the partition

General Testing Guidelines – Examples

- Choose inputs that force the system to generate all error messages;
- Design inputs that cause input buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated
- Force computation results to be too large or too small

J. A. Wittaker (2009) Exploratory Software Testing, Adison-Wesley

Unit Testing

THE UNIVERSITY OF SYDNEY

Unit Testing – Terminology

- Code under test
- Unit test
  - a piece of code written by a developer that executes a specific functionality in the code under test and asserts a certain behavior or state
  - Small unit of code, e.g., a method or class
  - External dependencies are removed (mocks can be used)
  - Not suitable for complex user interface or component interaction
- Test Fixture
  - The context for testing
    - Usually, shared set of testing data
    - Methods for setup those data
    - E.g., a fixed string (test fixture), which is used as input for a method

Unit Testing Frameworks for Java

- Instead of writing your own test-harness use testing tools!

- Tasks of a unit test harness
  - Facilitates test classes / boiler plate code
  - Running tests
  - Reporting test results

- Many different testing frameworks for Java
  - Junit, TestNG, Jtest (commercial), …

https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#Java

Unit Testing Frameworks – Junit

- An open-source framework for writing and running tests for Java
  - Test cases and fixtures, test suites, test runner
- Junit is one of the unit testing frameworks
- Uses annotations to identify methods that specify a test
- Can be integrated with Eclipse/IntelliJ, and build automation tools (e.g., Ant, Maven, Gradle)

https://github.com/junit-team/junit4

Junit – Constructs

- **Junit test (Test class)**
  - A method contained in a class which is only used for testing (called *Test class*)
- **Test suite**
  - Contains several test classes which will be executed all in the specified order
- **Test Annotations**
  - To define/denote test methods (e.g., @Test, @Before)
  - Such methods execute the code under test
- **Assertion methods (assert)**
  - To check an expected result versus actual results
  - Variety of methods
  - Provide meaningful messages in assert statements

Junit – Annotations

JUnit 4*	Description
import org.junit.*	Import statement for using the following annotations
@Test	Identifies a method as a test method
@Before	Executed before each test. To prepare the test environment (e.g., read input data, initialize the class)
@After	Executed after each test. To cleanup the test environment (e.g., delete temporary data, restore defaults) and save memory
@BeforeClass	Executed once, before the start of all tests. To perform time intensive activities, e.g., to connect to a database. Need to be defined as static to work with Junit
@AfterClass	Executed once, after all tests have been finished. To perform clean-up activities, e.g., to disconnect from a database. Need to be defined as static to work with Junit

*See Junit 5 annotations and compare them https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations

Junit – Assert Class

- Assert class provides *static* methods to test for certain conditions

- Assertion method compares the actual value returned by a test to the expected value
  - Specify the expected and actual results and the error message
  - Throw an *AssertionException* if the comparison fails

Junit – Methods to Assert Test Results

Method	Description
fail([message]) 	Let the method fail. E.g., to check that a certain part of the code not reached or to have a failing test before the test code implemented
assertTrue([message,] boolean condition) assertFalse([message,] boolean condition)	Checks that the Boolean condition is true Checks that the Boolean condition is false
assertEquals([message,] expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([message,] expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same
assertNull([message,] object) assertNotNull([message,] object)	Checks that the object is null Checks that the object is not null

*Also check assertions in Junit 5 – https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions

Junit Test – Example

```
1
2 //Class to be tested
3 public class MyClass{
4     public int multiply (int m, int n){
5         return m * n;
6     }
7
8 }
9
```

MyClass' multiply(int, int) method

```
1  import static org.junit.Assert.assertEquals;
2
3  import org.junit.Test;
4
5  public class MyClassTests {
6
7      @Test
8      public void multiplicationOfZeroIntegersShouldReturnZero() {
9          MyClass tester = new MyClass(); // MyClass is tested
10
11         // assert statements
12         assertEquals(0, tester.multiply(10, 0), "10 x 0 must be 0");
13         assertEquals(0, tester.multiply(0, 10), "0 x 10 must be 0");
14         assertEquals(0, tester.multiply(0, 0), "0 x 0 must be 0");
15     }
16 }
```

MyClassTests class for testing the method multiply(int, int)

Junit – Executing Tests

- Tests can be executed from the command line
  - *runClass()* of the *org.junit.runner.JUnitCore* class allows developers to run one or several test classes
  - Information about tests can be retrieved using the *org.junit.runner.**Result*** object

- Test automation
  - Build tools such as Maven or Gradle along with a Continuous Integration Server (e.g., Jekins) can be configured to run tests automatically on a regular basis
  - Essential for regular daily tests (agile development)

Junit – Executing Tests

- To run tests from the command line

```
1  import org.junit.runner.JUnitCore;
2  import org.junit.runner.Result;
3  import org.junit.runner.notification.Failure;
4
5  public class MyTestRunner {
6    public static void main(String[] args) {
7      Result result = JUnitCore.runClasses(MyClassTest.class);
8      for (Failure failure : result.getFailures()) {
9        System.out.println(failure.toString());
10     }
11   }
12 }
```

Junit – Test Suites

- **Test suite** contains several test classes which will be executed all in the specified order

```
1  import org.junit.runner.RunWith;
2  import org.junit.runners.Suite;
3  import org.junit.runners.Suite.SuiteClasses;
4
5  @RunWith(Suite.class)
6  @SuiteClasses({
7          MyClassTest.class,
8          MySecondClassTest.class })
9
10 public class AllTests {
11
12 }
```

Junit – Static Import

- Static import is a feature that allows fields and methods in a class as public static to be used without specifying the class in which the field s defined

```
// without static imports you have to write the following statement
Assert.assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));


// alternatively define assertEquals as static import
import static org.junit.Assert.assertEquals;

// more code

// use assertEquals directly because of the static import
assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
```

Junit – IntelliJ Support

- Create Junit tests via wizards or write them manually
- IntelliJ IDE also supports executing tests interactively
  - Run-as Junit Test will starts JUnit and execute all test methods in the selected class
- Extracting the failed tests and stack traces
- Create test suites

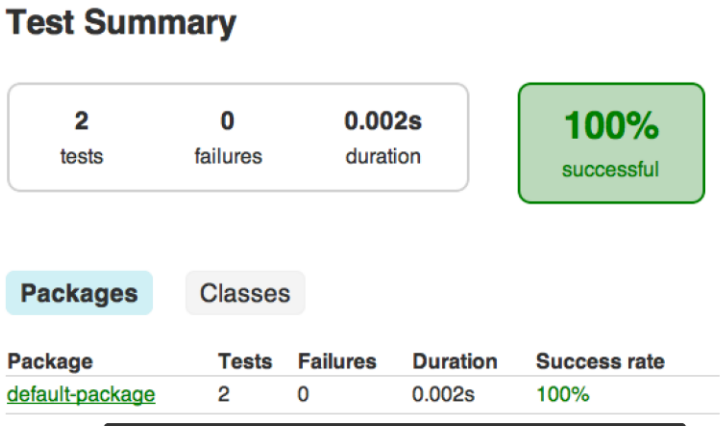
Automated Test Frameworks

- Good practice is that tests are kept as an asset of the process
  - and can be run automatically, and frequently
- Also, reports should be easy to understand
  - Big Green or Red Signal
- A framework is some software that allows test cases to be described in standard form and run automatically

Tests Automation – Junit with Gradle

- To use Junit in your Gradle build, add a testCompile dependency to your build file
- Gradle adds the test task to the build graph and needs only appropriate Junit JAR to be added to the classpath to fully activate the test execution

```
1 ...
2 apply plugin: 'java'
3
4 repositories {
5     mavenCentral()
6 }
7 dependencies {
8     testCompile 'junit:junit:4.8.2'
9 }
```



Junit with Gradle – Parallel Tests

```
1  ...
2  apply plugin: 'java'
3
4  repositories {
5      mavenCentral()
6  }
7  dependencies {
8      testCompile 'junit:junit:4.8.2'
9  }
10 test {
11     maxParallelForks = 5
12     forkEvery = 50
13 }
```

maximum simultaneous JVMs spawned

causes a test-running JVM to close and be replaced by a brand new one after the specified number of tests have run under an instance

Next week ….

Theory of Testing, Design of Test Cases and more unit testing

Tutorial - Unit Testing

THE UNIVERSITY OF SYDNEY

References

- Ian Sommerville. 2016. Software Engineering (10th ed.) Global Edition. Pearson, Essex England

- Junit 4, Project Documentation, https://junit.org/junit4/

- Vogella GmbH, JUnt Testing with Junit – Tutorial (Version 4.3,21.06.2016) http://www.vogella.com/tutorials/JUnit/article.html