University of Nottingham
UK | CHINA | MALAYSIA

Lecture 10 – Important Miscellaneous

COMP2013 (AUT1 23-24)

Dr Marjahan Begum and Dr Horia A. Maior

Register your attendance

COMP2013: Developing Maintainable Software
Week 11 – 4:00pm Monday – 04 December 2023

valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

Content

Module Feedback

Can all of you please participate …

- https://bluecastle-uk-surveys.nottingham.ac.uk
- Log in using username and password
- All views are welcome
  - Positive
  - Negative
  - Neural
- Survey starts : 04/12/2023 09:00
- Survey ends : 08/12/2023 17:00

Coursework Feedback

High level Feedback

Milestone1 High Level Feedback

- Provisional marks will be released with feedback
  - Friday 8th December

- Any queries regarding the coursework
  - Please ask us during Friday's lab session

Git

- Most people set up git and number of commits and useful messages
- Some only showed that they have updated the repo with the current code base
- Very good use of branches
  - Main branch
  - Different development branches
- Too much time on Git

UML Class Diagrams

- Initial diagram and automatic class diagram
  - Clear explanations of the individual classes and details about the relationships
  - Brief comparison is made with automatic and self created
- Revised class diagram
  - OO principles applied
  - Clear explanations of the individual classes and details about the relationships
  - Rationalized why it’s a better class diagram
  - Some discussed about patterns they will employ

Understanding of current software structure and refactoring

- Main classes in the source code are explained
  - In relation to the class diagram
- Running the game
- Initial refactoring were shown
- Some ideas about future refactoring in relation to OO principles and design patterns

Video quality

- Dipropionate time spent on git
- Speaking too fast
- Image poor quality
- Not explaining what you are showing just say this is initial and this is revised
- No audio

EXAM Revision

Important topics

Fowler Defined Different Categories of Refactoring

- *Composing Methods:* refactoring within a method or within an existing class
- *Moving Features Between Objects:* refactoring changing the responsibility of a class
- *Organizing Data:* refactoring to improve data structures and object linking
- *Simplifying Conditional Expressions:* encapsulation of conditions by replacing with polymorphism
- *Making Method Calls Simpler:* refactorings that make interfaces simpler
- *Dealing with Generalization:* moving methods up and down the hierarchy of inheritance by (often) applying the Factory or Template design pattern
- *Big Refactorings:* architectural refactoring to promote loose coupling or to realise a redesign via object orientation (Note: this is extremely difficult for most projects).

Software Maintenance (SWM)

- What is it?
- Why is it important?
- Different types?
- What are the challenges?

Maintainability is not an afterthought, but should be addressed from the very beginning of a development project

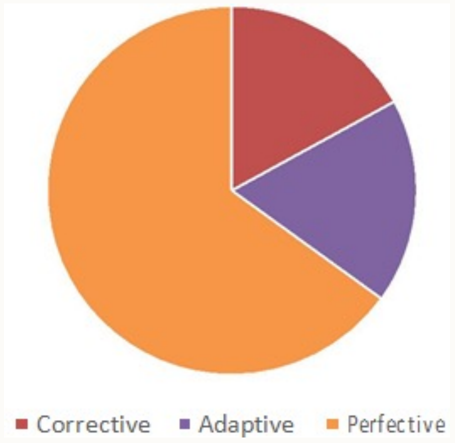
Maintainable Software

- Overview of generic maintainability guidelines:
  - Write short units (constructors/methods) of code
  - Write simple units of code
  - Write code once
  - Keep unit interfaces small
  - Separate concerns in modules (classes)
  - Couple architecture components loosely
  - Keep your codebase small
  - Automate development pipeline and tests
  - Write clean code

Visser et al (2016)

Three (or four, depending on authors) Main Categories of SWM

Maintenance effort

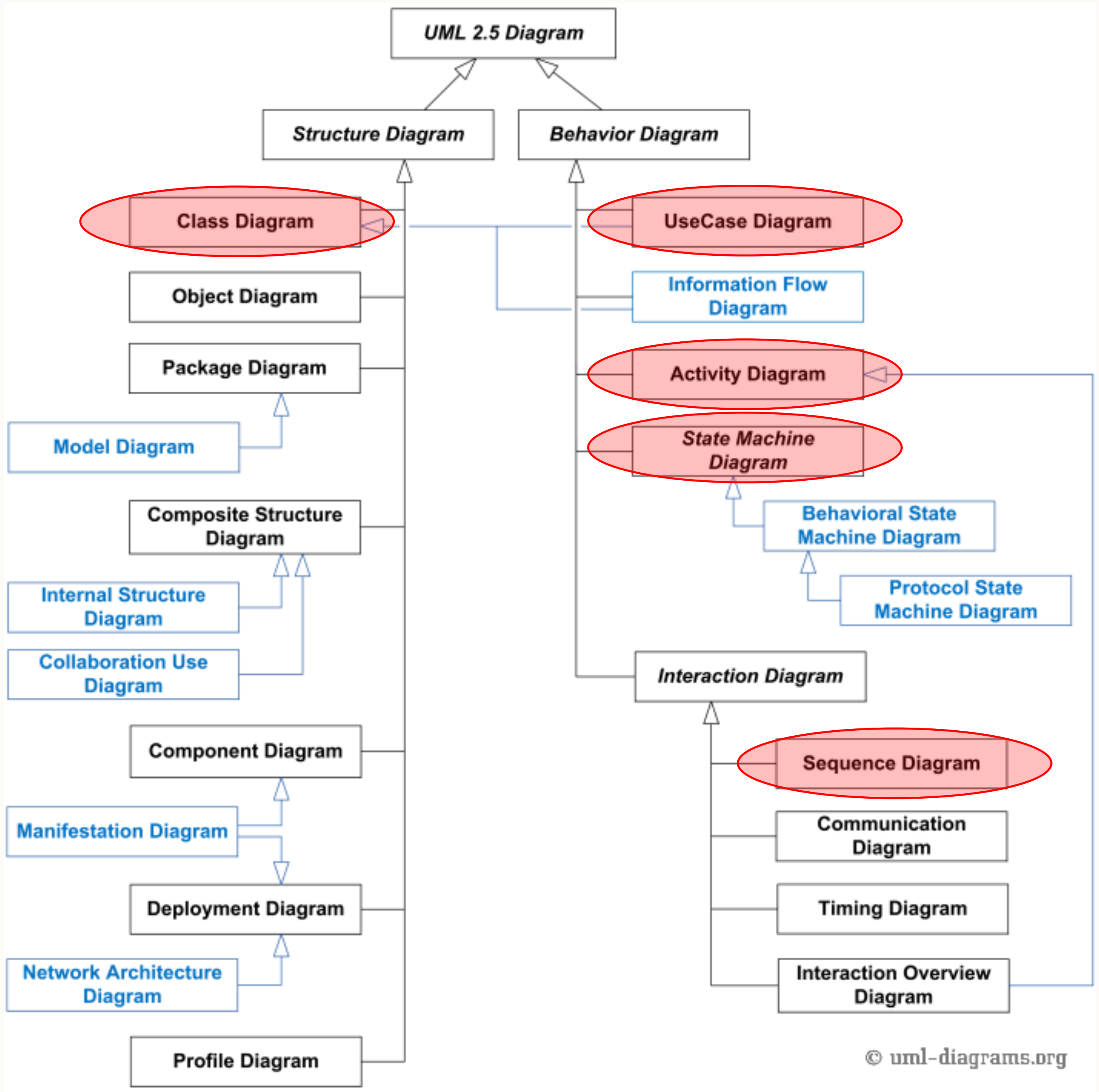


- Corrective Maintenance
  - Finding and fixing errors in the system
    - e.g. bugs
- Adaptive Maintenance
  - The system has to be adapted to changes in the environment in which it operates
    - e.g. VAT change, bank offers new mortgage product
- Perfective + Preventive Maintenance
  - Users of the system (and/or other stakeholders) have new or changed requirements
  - Ways are identified to increase quality or prevent future bugs from occurring

UML and its Java Implementation

- UML diagrams
  - You only need to consider the ones we discussed in the lecture
  - What are they used for?

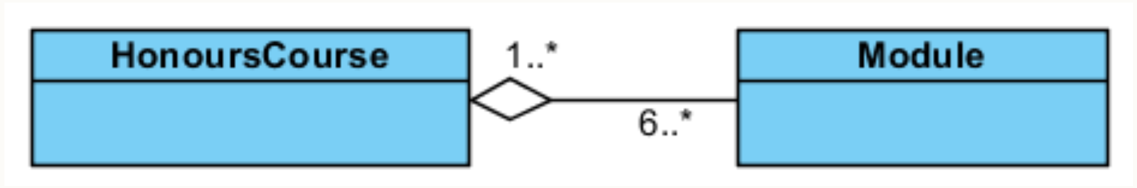
- Class diagrams
  - Relationship types and multiplicity
    - Design
    - Implementation



UML 2.5 Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Object Diagram
Package Diagram
Model Diagram
Composite Structure Diagram
Internal Structure Diagram
Collaboration Use Diagram
Component Diagram
Manifestation Diagram
Deployment Diagram
Network Architecture Diagram
Profile Diagram
UseCase Diagram
Information Flow Diagram
Activity Diagram
State Machine Diagram
Behavioral State Machine Diagram
Protocol State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Timing Diagram
Interaction Overview Diagram
© uml-diagrams.org

Class Diagram: Design

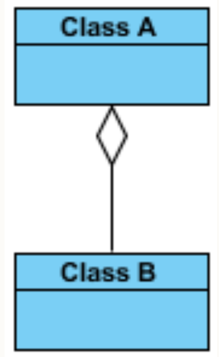
- Aggregation ("is part of" relationship)

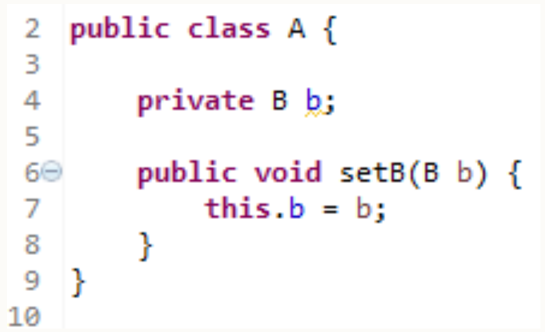


Each Honours Course consists of 6 or more Modules; each Module could be part of one or more Honours Courses

Class Diagram: Implementation

- Class B is part of class A (semantically) but class B can be shared and if class A is deleted, class B is not deleted.
  - Class A stores the reference to class B for later use
    - Often setter injection is used





```java
public class A {

    private B b;

    public void setB(B b) {
        this.b = b;
    }
}
```

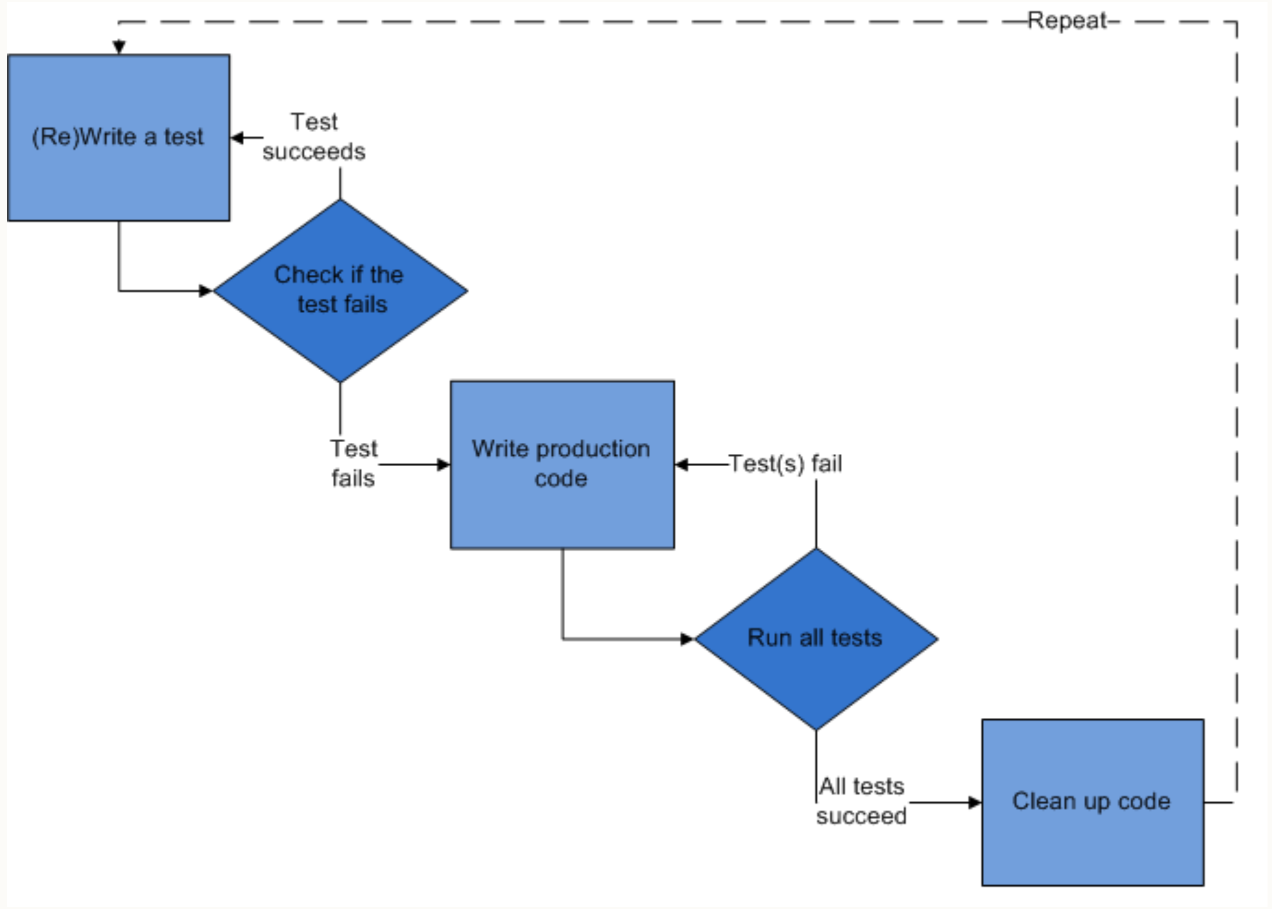
Coding Tools for SWM

- Documenting code
  - Javadocs
    - Make sure you remember the common block tags (e.g. @param)
- Build files (build scripts)
  - Common tools (e.g.; Maven; Gradle)
  - Why build files are useful
  - Typical tasks accomplished with build files
- Testing
  - Unit and Regression Testing: What's the difference?
  - Unit Testing
  - Test-Driven Development (TDD)

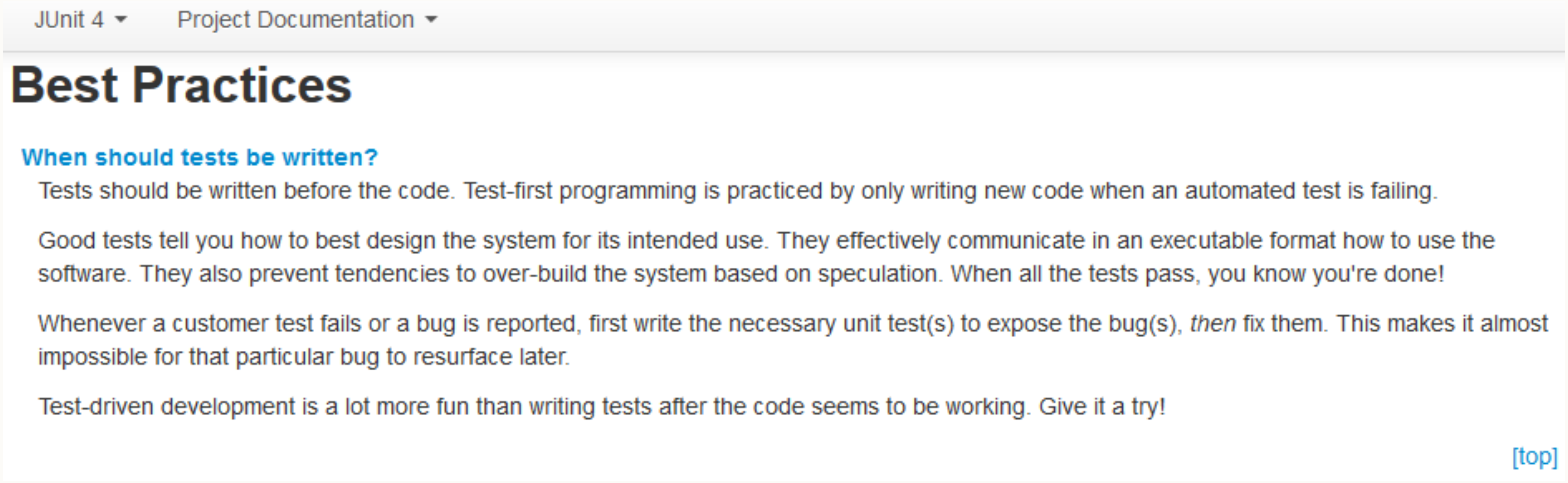
Test-Driven Development

- How does TDD work?
  - Write a test
  - Check if test fails
  - Test fails > Write production code
  - Run all tests
  - All tests succeed > Clean up code

- What does writing the test first allow us to do?
  - Test can be derived from requirements, so it makes sure we meet the specs
  - We write the minimum amount of code to pass (following XP's YAGNI principle)
  - We write maintainable code



Test-Driven Development



Best Practices

When should tests be written?

Tests should be written before the code. Test-first programming is practiced by only writing new code when an automated test is failing.

Good tests tell you how to best design the system for its intended use. They effectively communicate in an executable format how to use the software. They also prevent tendencies to over-build the system based on speculation. When all the tests pass, you know you're done!

Whenever a customer test fails or a bug is reported, first write the necessary unit test(s) to expose the bug(s), *then* fix them. This makes it almost impossible for that particular bug to resurface later.

Test-driven development is a lot more fun than writing tests after the code seems to be working. Give it a try!

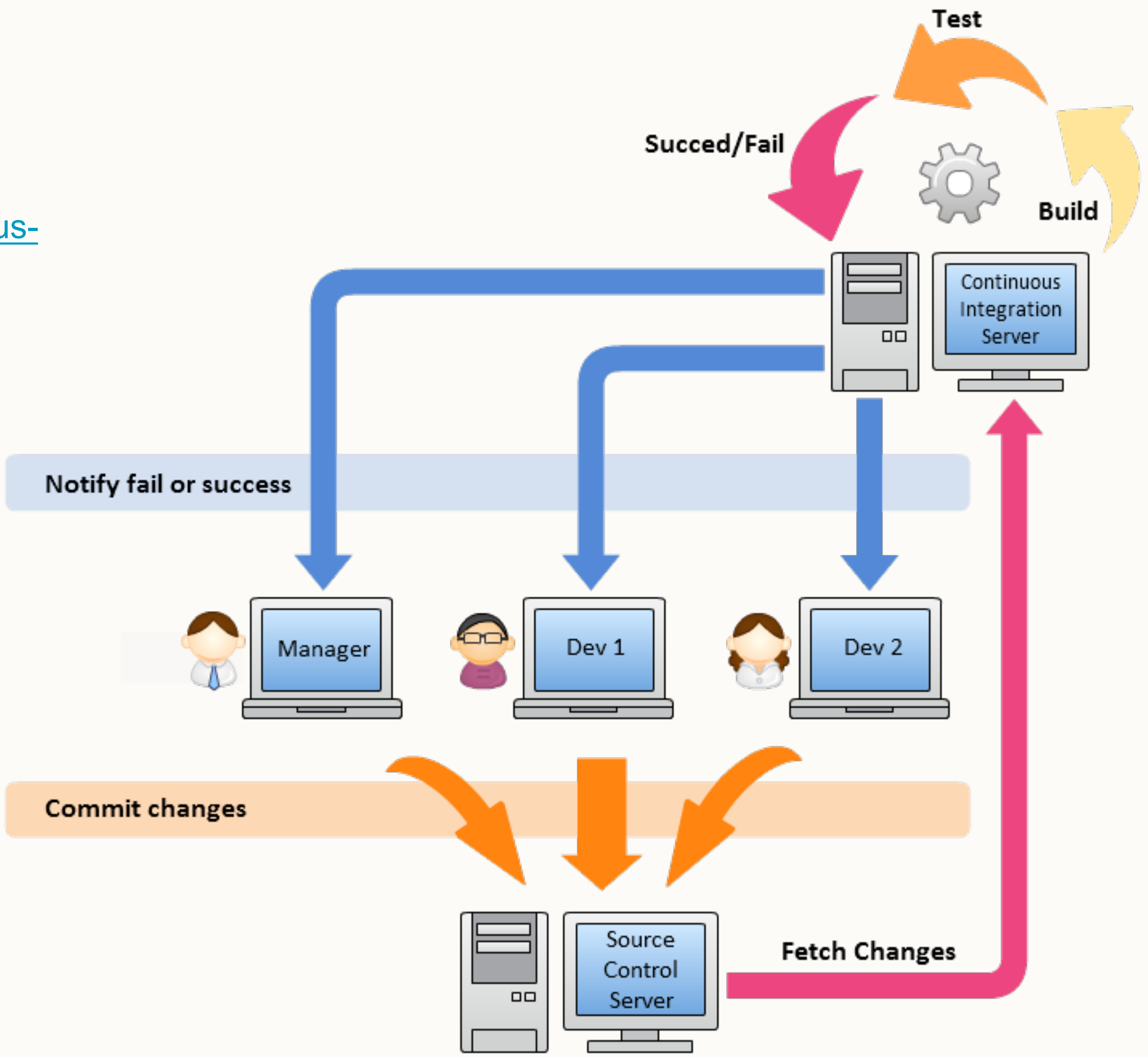
[top]

https://junit.org/junit4/faq.html#best_1

Continuous Integration Lifecycle

https://code-maze.com/what-is-continuous-integration/

Integrate new code often and early



Design Principles and Patterns

- From Concepts to Patterns
  - OO Concepts
    - Object model (abstraction, encapsulation, modularity, hierarchy); data abstraction; inheritance; polymorphism; interfaces
  - OO Design Principles
    - Encapsulate what varies; favour composition over inheritance (object aggregation vs class inheritance); program to interfaces, not implementations; ...
  - OO Design Patterns:
    - Show how to build systems with good OO design qualities (reusable; extensible; maintainable)

Principles and Patterns

- SOLID Principles
  - You should understand and be able to explain the idea behind the principles
- Design Patterns
  - You should understand the ones we discussed in the lectures
    - Singleton; Abstract Factory; Factory Method; Adapter; Observer; State; MVC
    - You should be able to identify core design patterns in class diagrams
    - Only the ones we have gone through
  - You don't need to be able to code them

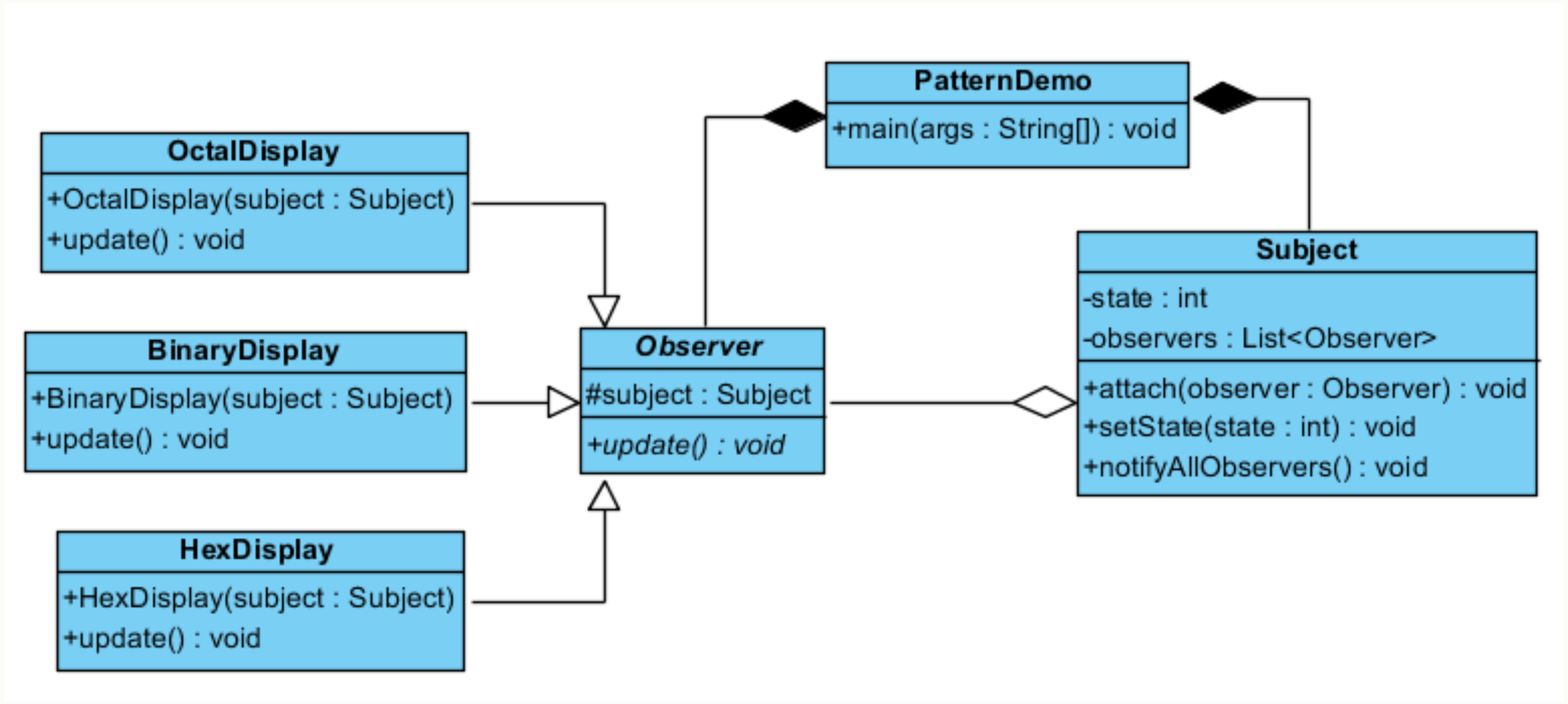
Example: Design Principle

- UB says "High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions."

- Dependency Inversion Principle
  - Goal is to reduce coupling between different pieces of code by adding a layer of abstraction

Example: Design Pattern

- What pattern? Benefits? Example for application?

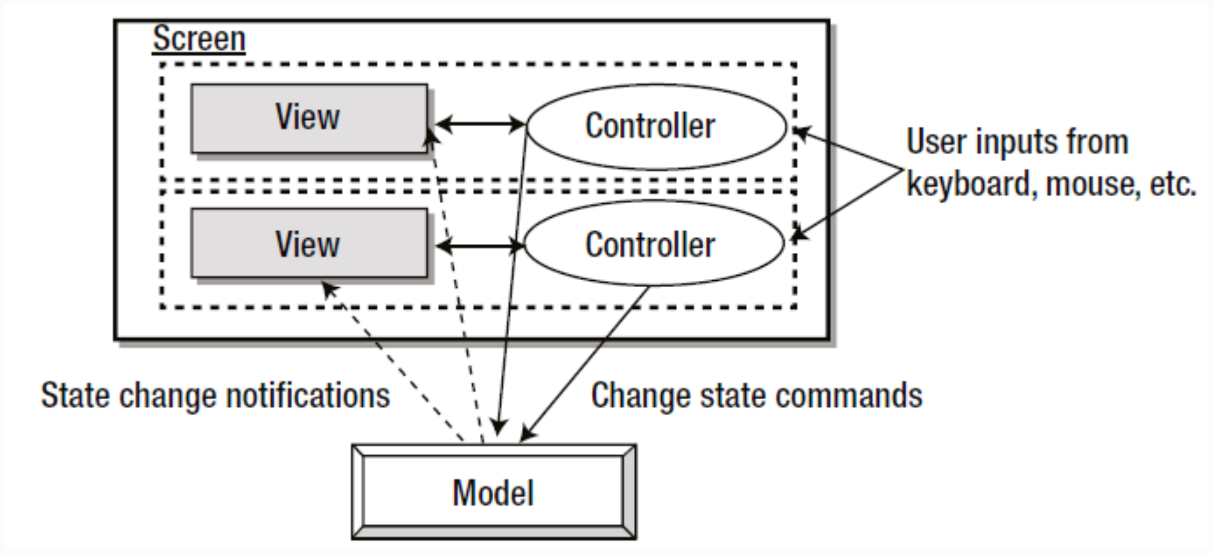

PatternDemo
+main(args : String[]) : void
OctalDisplay
+OctalDisplay(subject : Subject)
+update() : void
BinaryDisplay
+BinaryDisplay(subject : Subject)
+update() : void
HexDisplay
+HexDisplay(subject : Subject)
+update() : void
Observer
#subject : Subject
+update() : void
Subject
-state : int
-observers : List<Observer>
+attach(observer : Observer) : void
+setState(state : int) : void
+notifyAllObservers() : void

Maintainable GUI Development

- GUIs in Java
  - Swing vs. JavaFX
  - Concept of FXML and SceneBuilder
  - Event driven programming in JavaFX
  - Simple usage of CSS to style interfaces
  - MVC design pattern
- Threading principles
  - Stopping GUIs becoming unresponsive
  - The idea behind why we want to do this…

MVC Design Pattern

- MVC
  - The ***model*** provides a way for ***views*** to subscribe to its state change notifications
  - Any interested ***views*** subscribe to the ***model*** to receive state change notifications
  - The ***model*** notifies all ***views*** that had subscribed whenever a ***model's*** state changes


Screen
View
Controller
View
Controller
User inputs from keyboard, mouse, etc.
State change notifications
Change state commands
Model

Threading

- Stopping GUIs Becoming Unresponsive

  - JavaFX launches the UI on a JavaFX Application thread
    - This thread should be left handling the UI interaction
    - Heavy computation should be done elsewhere to prevent freezing!

  - JavaFX provides a solution: The "javafx.concurrent" package
    - A way of creating and communicating with other threads with a JavaFX interface

Libraries and Communal Software Development

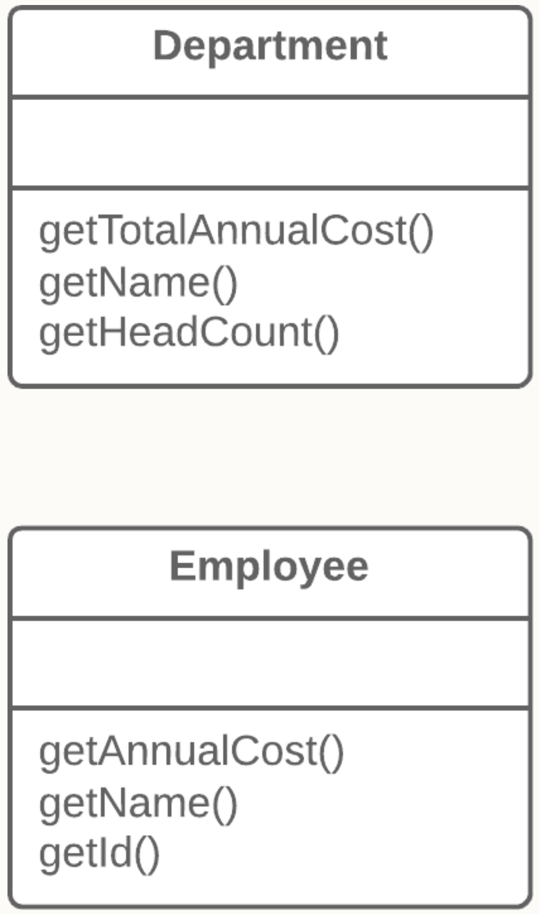
- Software deployment through libraries
  - Example: Java JRE > *.jar file + API + License

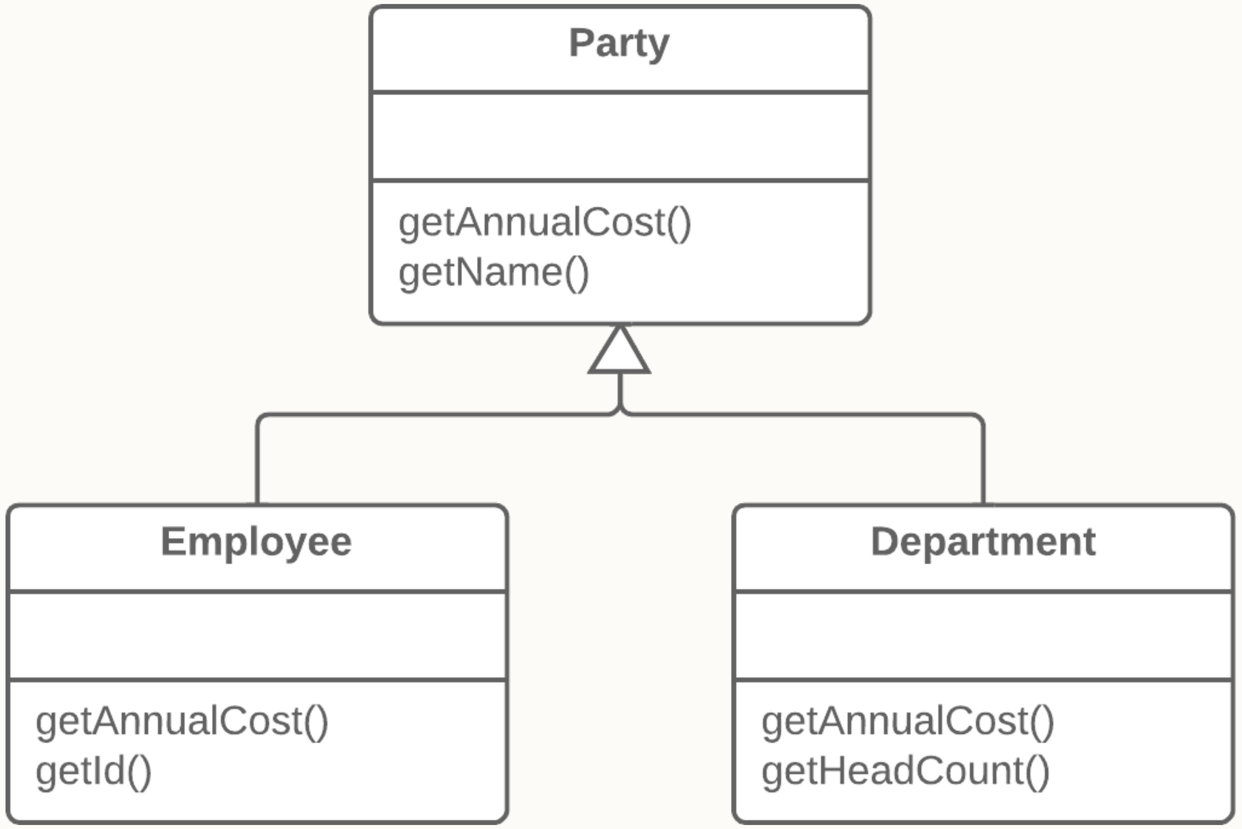
- Open source software development
  - Why go open source?
  - Principle of open source development
    - Open source criteria
    - Common licenses / types
    - BSD; GNU; etc.
    - Permissive; CopyLeft; etc.
  - Tracking issue and bugs
    - Mantis and Bugzilla

Replace type code with polymorphism

- **Switch** statements are very rare in properly designed object-oriented code, much more common in procedural programs
  - Therefore, a switch statement is a simple and easily detected “bad smell”
  - Of course, not all uses of switch are bad
  - A switch statement should *not* be used to distinguish between various kinds of object
  - Lengthy to test all of the test cases
- There are several well-defined refactorings for this case
  - The simplest is the creation of subclasses

Calculating annual cost examples





Exam Formalities

Format of Exam

- The exam (worth 25%) will be using the UoN Rogo system
  - Check your exam timetable to find out which room you have to go to!

- Mock exam (will be announced)
  - Get an idea of the types of questions you are going to get in the actual exam

- Different question styles (multiple choice; text answers; diagram labelling; etc.)
  - You will not be asked to write big code sections from scratch but you will be asked questions about code and perhaps to improve given code

Time and Location of the Exam

- Time and location
  - Time
    - Friday XX/XX/2024; Xam
  - Location
    - No room yet
    - Will be multiple rooms
    - Make sure you go to the correct one!

Rogo Mock Exam

High level Feedback

Rogo Mock Exam

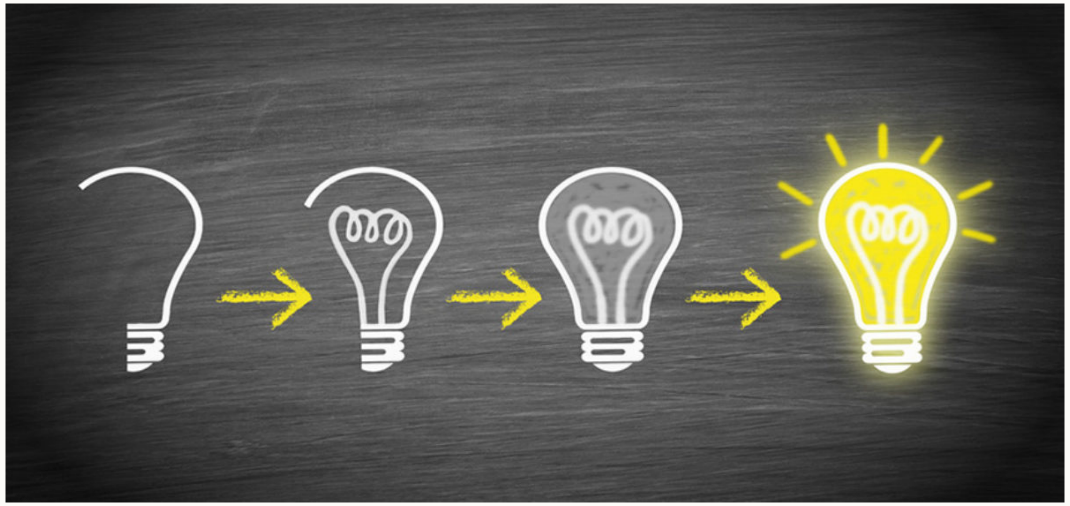
- Real exam
  - 100 marks of material to be completed in 60 minutes
- Mock exam
  - 35 marks of material to be completed in 21 minutes
- Exam will be on multiple; you can always go back to previous questions
- Some parts of the exam will be automatically marked (e.g. multiple choice)
- Some parts of exam will be manually marked (e.g. text)

Rogo Mock Exam to Try

- Mock exam
  - As well as seeing the kind of questions you will encounter, it is also a chance to experience the Rogo online system which is what you will use in the exam.
  - The system will give you feedback, but obviously some of the questions need manual marking, so you will have to look at the answer and judge for yourself if you have scored.

And Finally …

- What have you learned in the module and how does it help you in the future?

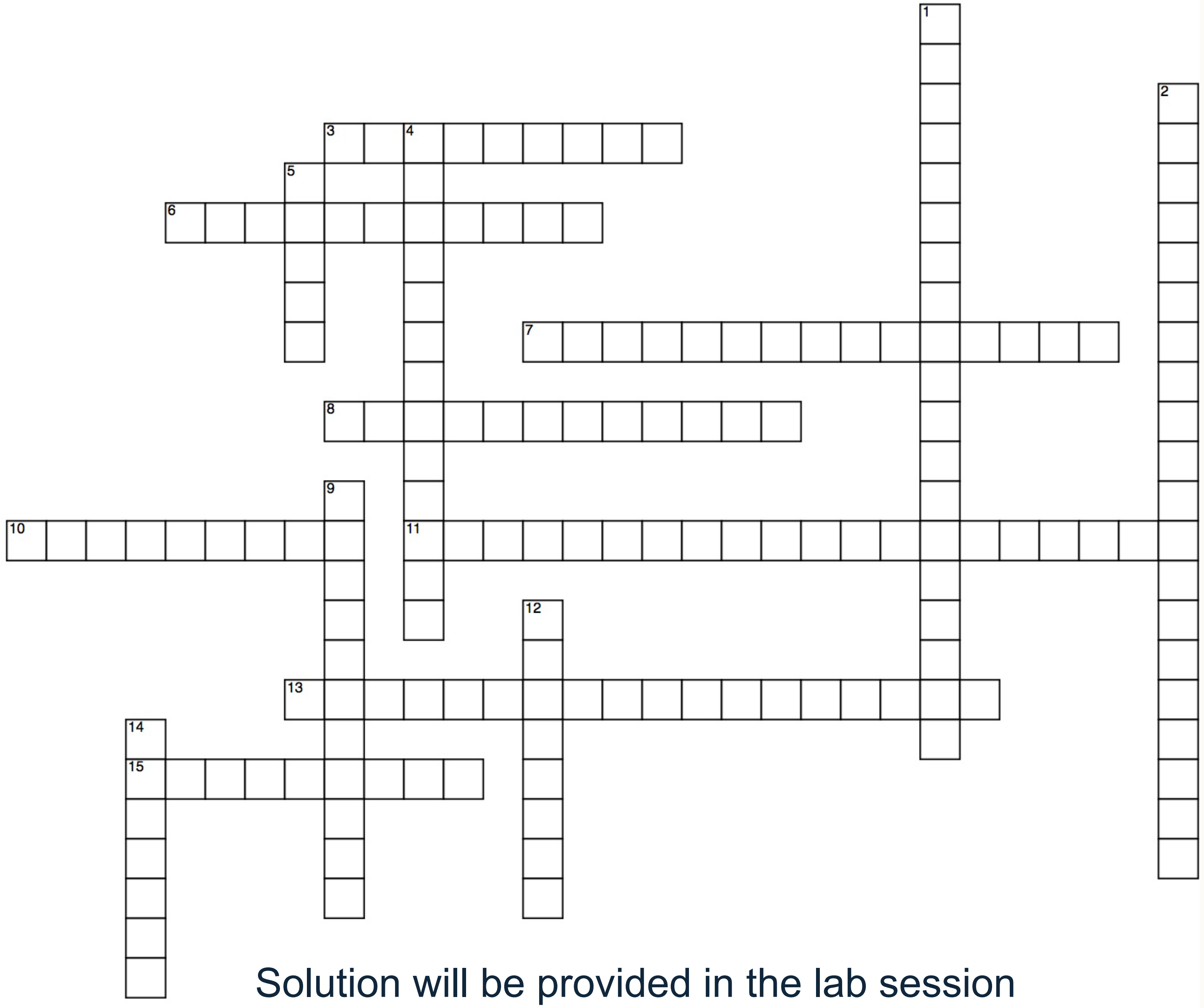


- And the last word for your revision:
  - Lecture recordings are not the most reliable source of information
  - You should use them only to complement other more reliable resources

Crossword Puzzle

SWM Design Principles and Patterns

Object orientation, SOLID and GOF Patterns Puzzle



Solution will be provided in the lab session

ACROSS

3 Describes the use of plugins and APIs
6 Expose essential features while hiding irrelevant detail
7 Uninstantiated creational pattern
8 Adaptation to a specific usage
10 For creating one and only one instance
11 Clients should not be forced to depend on methods that they themselves do not use
13 If it quacks like a duck and swims like a duck but needs batteries you are probably breaking this principle
15 Flexible alternative to using inheritance

DOWN

1 High and low level modules should depend on abstractions
2 One class should do one thing
4 To modularise class functionality
5 To allow object behaviour to change at run time
9 Passing on behaviours from super classes to subclasses
12 One-to-many behavioural pattern
14 Is a wrapper for interfaces