Week 8

[W8.1] [Revisiting] Drawing Class/Object Diagrams

[W8.2] [Revisiting] Drawing Sequence Diagrams

[W8.3] Testing: Types

[W8.4] Testing: Intermediate Concepts

▼ [W8.1] [Revisiting] Drawing Class/Object Diagrams

- See Week 4 Topics

▼ [W8.2] [Revisiting] Drawing Sequence Diagrams

- See Week 6 Topics

▼ [W8.3] Testing: Types

Integration Testing

▼ W8.3a - Quality Assurance → Testing → Integration Testing → What

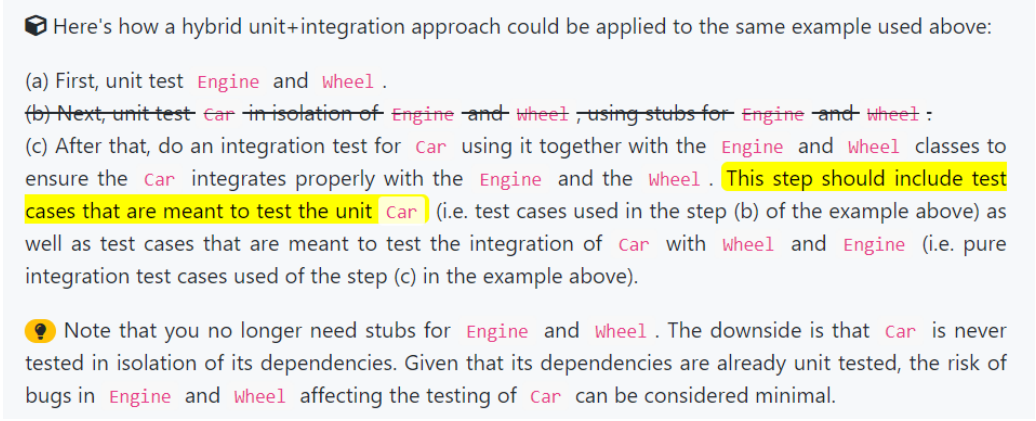
▼ Integration testing? (definition)

- Testing whether different parts of the software *work together* (i.e. integrates) as expected
- Aims to discover bugs in the 'glue code' related to how components interact with each other

▼ W8.3b - Quality Assurance → Testing → Integration Testing → How

▼ How? (3 + example)

- Not simply a case of repeating the unit test cases using the actual dependencies
- Additional test cases that focus on the interactions between the parts
- In practice, developers often use a hybrid of unit + integration tests to minimize the need for stubs



System Testing

▼ W8.3c - Quality Assurance → Testing → System Testing → What

▼ System testing? (definition)

- Take the *whole system* and test it against the *system specification*
- Typically done by a testing team (aka QA team)

▼ Basis of system test cases?

- Based on the specified external behavior of the system
- Sometimes, system tests go beyond the bounds defined in the specification (testing that the system fails 'gracefully' having pushed beyond its limits

Suppose the SUT is a browser supposedly capable of handling web pages containing up to 5000 characters. Given below is a test case to test if the SUT fails gracefully if pushed beyond its limits.

```
Test case: load a web page that is too big
* Input: load a web page containing more than 5000 characters.
* Expected behavior: abort the loading of the page and show a meaningful error message.
```

This test case would fail if the browser attempted to load the large file anyway and crashed.

▼ Examples of system testing against non-functional requirements? (6)

- *Performance testing* — to ensure the system responds quickly
- *Load testing* (aka *stress testing* or *scalability testing*) — to ensure the system can work under heavy load
- *Security testing* — to test how secure the system is
- *Compatibility testing*, *interoperability testing* — to check whether the system can work with other systems
- *Usability testing* — to test how easy it is to use the system
- *Portability testing* — to test whether the system works on different platforms

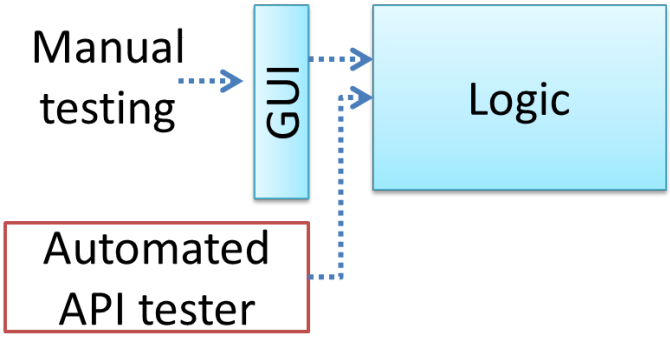
▼ W8.3d - Quality Assurance → Testing → Test Automation → Automated testing of GUIs

▼ Testing the GUI is much harder than testing the CLI/API. Why? (3)

- Most GUIs can support a large number of different operations, many of which can be performed in any arbitrary order
- GUI operations are more difficult to automate than API testing. Reliably automating GUI operations and automatically verifying whether the GUI behaves as expected is harder than calling an operation and comparing its return value with an expected value. Therefore, *automated regression testing of GUIs is rather difficult*
- The appearance of a GUI (and sometimes even behavior) can be different across platforms and even environments
  - e.g. a GUI can behave differently based on whether it is:
    - minimized or maximized

- in focus or out of focus
- in a high resolution display or a low resolution display

▼ How to automate testing of GUIs? (3)



- Move as much logic as possible out of the GUI to make GUI testing easier
- Most of the system can be tested using automated API testing
- The GUI still requires to be tested manually

▼ Some tools for automating GUI testing? (3)

- **TestFx**: automated testing of JavaFX GUIs
- **VisualStudio**: supports 'record replay' type of GUI testg automation
- **Selenium**: automated testing of Web application UIs

Acceptance Testing

▼ W8.3e - Quality Assurance → Testing → Acceptance Testing → What

▼ Acceptance testing (aka User Acceptance Testing (UAT))? (definition + 2)

- Test the system to ensure it meets the user requirements
- Gives an assurance to the customer that the system does what it is intended to do
- Acceptance test cases often defined at the beginning of the project, usually based on the use case specification

▼ W8.3f - Quality Assurance → Testing → Acceptance Testing → Acceptance versus system testing

- Acceptance testing comes after system testing
- Both involve testing the whole system

▼ Differences? (4)

Aa System Testing	Acceptance Testing
Done against the system specification	Done against the requirements specification
Done by testers of the project team	Done by a team that represents the customer
Done on the development environment or a test bed	Done on the deployment site or on a close simulation of the deployment site
Both negative and positive test cases	More focus on positive test cases

-

▼ Differences between system specification and requirement specification? (3)

- In many cases, one document serves as both a system specification and requirement specification

Aa System Specification	Requirement Specification
Can also include details on how it will fail gracefully when pushed beyond limits, how to recover, etc. specification	Limited to how the system behaves in normal working conditions
Written in terms of how the system solve those problems (e.g. explain the email search feature)	Written in terms of problems that need to be solved (e.g. provide a method to locate an email quickly)
Could contain additional APIs not available for end-users (for the use of developers/testers)	Specifices the interface available for intended for end-users

▼ Passing system tests does not necessarily mean passing acceptance testing. Examples? (2)

- The system might work on the testbed environments but might not work the same way in the deployment environment, due to

subtle differences between the two environments

- The system might conform to the system specification but could fail to solve the problem it was supposed to solve for the user, due to flaws in the system design

Alpha/Beta Testing

▼ W8.3g - Quality Assurance → Testing → Alpha/Beta Testing → What

▼ Alpha testing? (definition)

- Performed by the users, under controlled conditions set by the software development team

▼ Beta testing? (definition)

- Performed by a selected subset of target users of the system in their natural work setting

▼ Open beta release? (definition)

- The release of not-yet-production-quality-but-almost-there software to the general population

Exploratory vs Scripted Testing

▼ W8.3h - Quality Assurance → Testing → Exploratory and Scripted Testing → What

▼ Scripted testing? (definition)

- First write a set of test cases based on the expected behavior of the SUT, and then perform testing based on that set of test cases

▼ Exploratory testing (aka reactive testing, error guessing technique, attack-based testing, bug hunting)? (definition + 3)

- Devise test cases *on-the-fly*, creating new test cases based on the results of the past test cases
- The 'simultaneous learning, test design, and test execution' whereby the nature of the follow-up test case is decided based on the behavior of the previous test cases
- Driven by observations during testing

- Usually starts with areas identified as error-prone, based on the tester's past experience with similar systems (conduct more tests for those operations where more faults are found)

▼ W8.3i - Quality Assurance → Testing → Exploratory and Scripted Testing → When

- A **mix** of both scripted and exploratory testing is better

▼ Comments about exploratory testing: (4)

- The success of exploratory testing depends on the tester's prior experience and intuition
  - Should be done by experienced testers, using a clear strategy/plan/framework
  - Ad-hoc exploratory testing by unskilled or inexperienced testers without a clear strategy is not recommended for real-world non-trivial systems
  - While exploratory testing may allow to detect some problems in a relatively short time, it is not prudent to use exploratory testing as the sole means of testing a critical system

▼ Comments about scripted testing: (2)

- More system, and hence, likely to discover more bugs given sufficient time
  - Exploratory testing would aid in quick error discovery, especially if the test has a lot of experience in testing similar systems

---

▼ [W8.4] Testing: Intermediate Concepts

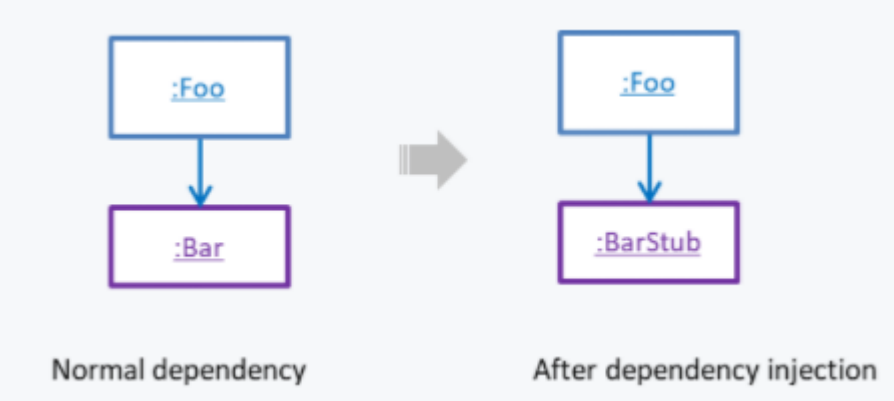
Dependency Injection

▼ W8.4a - Quality Assurance → Testing → Dependency Injection → What

▼ Dependency injection? (definition + use)

- The process of 'injecting' objects to replace current dependencies with a different object

- Often used to inject stubs to isolate the SUT from its dependencies so that it can be tested in isolation



▼ W8.4b - Quality Assurance → Testing → Dependency Injection → How

- Polymorphism can be used to implement dependency injection

Testability

▼ W8.4c - Quality Assurance → Testing → Introduction → Testability

▼ Testability? (definition)

- An indication of how easy it is to test an SUT
- The higher the testability, the easier it is to achieve a better quality software

Test Coverage

▼ W8.4d - Quality Assurance → Testing → Test Coverage → What

▼ Test coverage? (definition)

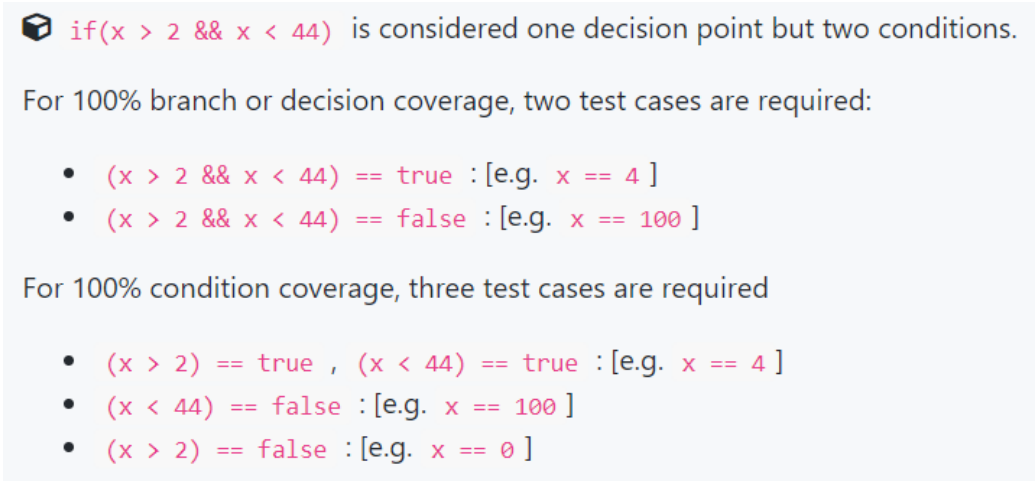
- A metric used to measure the extent to which testing exercises the code (i.e. how much of the code is 'covered' by the tests

▼ Some examples of different coverage criteria? (6)

- **Function/method coverage**: based on functions executed (e.g. testing executed 90 out of 100 functions)
- **Statement coverage**: based on the number of lines of code executed (e.g. testing executed 23k out of 25k LOC)
- **Decision/branch coverage**: based on the decision points exercised (e.g. an `if` statement evaluted to both `true` and

`false` with separate test cases during testing is considered 'covered')

- **Condition coverage**: based on the boolean sub-expressions, each evaluated to both true and false with different test cases (different from decision coverage)



- **Path coverage**: measures coverage in terms of possible paths through a given part of the code executed
  - A commonly used notation for path analysis is called the *Control Flow Graph (CFG)*
- **Entry/exit coverage**: measures coverage in terms of possible *calls to* and *exits* from the operations in the SUT

▼ W8.4e - Quality Assurance → Testing → Test Coverage → How

▼ How is measuring coverage done? (2)

- Using *coverage analysis tools*
- Most IDEs have inbuilt supports for measuring test coverage, or at least have plugins that can measure test coverage

▼ Usefulness of coverage analysis? (1)

- Useful in improving the quality of testing (e.g. if a set of test cases does not achieve 100% branch coverage, more test cases can be added to cover missed branches)

TDD

▼ W8.4f - Quality Assurance → Testing → Test-Driven Development → What

▼ Test-driven development (TDD)? (definition + 2)

- Advocates writing the tests before writing the SUT, while evolving functionality and tests in small increments
- You first define the precise behavior of the SUT using test cases, and then write the SUt to match the specified behavior
- One big advantage of TDD is that it guarantees the code is testable