Object Oriented Programming with Applications
Lecture 8

Witold Gawlikowicz

School of Mathematics, University of Edinburgh

2018-19[1]

[1] Last updated 31st October 2018

Lecture 8

- Testing
- Problem Sheet 5 overview

Testing

- Writing robust and maintainable code requires testing
- It assures the required functionality is implemented correctly and is adequate to end-users' needs
- We can distinguish two types of tests:
  - Manual - carried out by users by interacting with the software
  - Automatic - Separate test software is used to interact with the software in predetermined ways and to report the results
- There are different levels of testing, we can distinguish:
  - Unit tests
  - Integration tests
  - System tests
  - Acceptance tests

Unit tests

- Unit tests focus on the lowest and most granular level of software
- They are aimed at individual methods and classes
- Written with knowledge of internal implementation detail of the code under test (**white box testing**)
- Very useful for refactoring (changing the design of the code without affecting it's behaviour) as a well written test suite can quickly identify which parts of the code exhibit unintended changes in behaviour after the latest change
- Aimed at frequent execution - say every commit - therefore need to be fast

Integration tests

- Integration tests focus on interaction between components of the program
- They are aimed at program and solution level - test if a few classes combined together provide the expected functionality
- Aimed at fairly frequent execution - say every push - don't need to be as fast as unit tests
- Often run by a build agent - a separate computer monitoring the code repository and triggering a build and test execution (and possibly more tasks) every time the repository is changed

System tests

- Test system as a whole
- Designed without knowledge of the internal implementation detail (**black box testing**)
- Often carried out by a separate test team not involved in the development process at all
- Carried out in an environment resembling production as closely as possible
- Assures that software delivers the intended set of functionality, meets the technical requirements and is of appropriate quality

Acceptance tests

- Final stage of testing before the release
- Often carried out by the actual stakeholders - e.g. people that ordered the software
- Confirms that all the agreed features are available and work as expected
- Sometimes combined with User Experience testing - observing the user interact with the program and gathering his feedback in order to assess how intuitive the user interface is

General remarks

- The fact that all the tests are passing doesn't guarantee that the code is flawless - there may still be some classes, methods, branches of the code or specific inputs which may trigger unwanted behaviour - they just didn't get covered by the tests.
- There are tools that allow to analyze test coverage (% of classes or methods that get covered by your test cases). Even 100% test coverage doesn't guarantee that there aren't any bugs in the program, but it certainly helps and is worth aiming for.
- When writing tests, always try to outsmart yourself and find a case where the test would fail.
  - Don't waste too much time writing mundane test cases that you're sure are going to pass - these are still needed, but 1 or 2 is enough.
  - Focus on cases more likely to lead to some problems: inputs out of the usual range, null values for reference-type inputs, try trigger division by 0 etc.

Arrange - Act - Assert

- Most commonly used approach to writing unit tests, where a test code is split into 3 logical parts
- Arrange - Part of the test code where all the necessary inputs are defined. More advanced testing techniques would often use Mocks here - objects that are defined to act as if they are implemnting a given interface, but they only provide the rudimentary implementation needed for test execution
- Act - invoke method under test
- Assert - compare output of the method under test with the expected result

Arrange - Act - Assert

See `https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2017` for more details

Test Driven Development (TDD)

- One approach to software development is TDD
- In simplest terms, it assumes that a test code code is written first
- It's meant to help with the design of software by thinking from the point of view of someone using the resulting code
- Once a test is written, a skeleton implementation will need to be created so that the code compiles and test can run (and fail)
- A developer then proceeds by following a Red-Refactor-Green pattern
  - Run the tests
  - Provide the basic, minimal implementation so that the tests pass
  - Refactor the code and if the tests fail fix it until they pass
  - Carry on for as many iterations as needed
- See `https://www.codecademy.com/articles/tdd-red-green-refactor` for more details

Writing test code is still coding

- Just because you're writing test code doesn't mean you should forget all the techniques you've learned
- **Encapsulation** is still important, tests should have a very limited scope and focus on one thing
- Code still needs to be **robust** as you count on it to make judgments about the quality of your business (non-test) code based upon it
- **Code duplication** can still be avoided by using private methods and inheritance
- **Readability** is still extremely important - people trying to understand a new codebase (the entirety of code making up a program) would often look through it's tests first to see how different pieces of functionality are invoked, what are the interactions between classes, what are the outputs etc.

Popular unit testing frameworks for .NET

While you can write your own test code from scrtach, testing frameworks are usually used to help with the task. The most popular ones for .NET are:

- NUnit - `https://nunit.org/`
- Xunit - `https://xunit.github.io/`
- MSTest (build into the Windows distribution of .NET) - `https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest`

Problem Sheet 5

Available at: `https://github.com/OOPA2018/Problem-sheets/blob/master/ProblemSheet5.pdf`