Software quality and testing

- 1st course

Ádám István Szűcs | Budapest | February, 2017

Contents

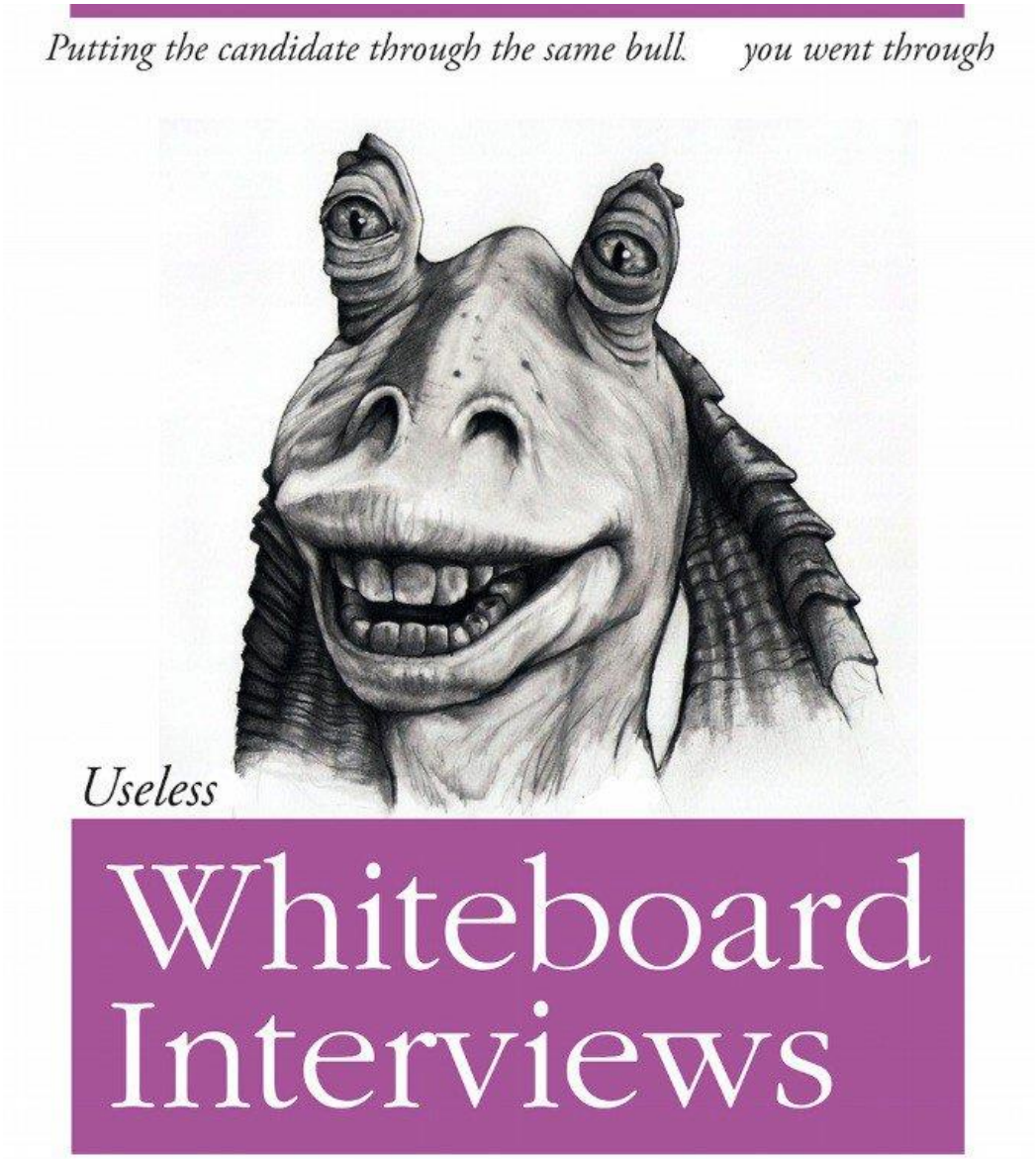
- Introduction, Agenda & Syllabus of the semester
- Tools
  - JetBrains IntelliJ IDEA
  - Coverage – metrics
- Sample testing project: uCoach @**federico.fiorini**
- Defining a Unit test
- Contrasting unit testing with integration testing
- Exploring a simple unit testing example
- Introduction and understanding test-driven development

Agenda and Syllabus

- Throughout the 12 – 13 weeks you will learn about software quality and testing where
  - You will need to write tests ☺
  - You'll get homework projects for(topic : topics)
  - Final exam during exam period
  - You will have fun!

eit Digital

What we wont do…



Quick Ref. of the IDE(A)

IntelliJ IDEA

DEFAULT KEYMAP

Remember these Shortcuts

Smart code completion	**Ctrl + Shift + Space**
Search everywhere	**Double Shift**
Show intention actions and quick-fixes	**Alt + Enter**
Generate code	**Alt + Ins**
Parameter info	**Ctrl + P**
Extend selection	**Ctrl + W**
Shrink selection	**Ctrl + Shift + W**
Recent files popup	**Ctrl + E**
Rename	**Shift + F6**

General

Open corresponding tool window	**Alt + #[0-9]**
Save all	**Ctrl + S**
Synchronize	**Ctrl + Alt + Y**
Toggle maximizing editor	**Ctrl + Shift + F12**
Inspect current file with current profile	**Alt + Shift + I**
Quick switch current scheme	**Ctrl + BackQuote (`)**
Open Settings dialog	**Ctrl + Alt + S**
Open Project Structure dialog	**Ctrl + Alt + Shift + S**
Find Action	**Ctrl + Shift + A**

Debugging

Step over / into	**F8 / F7**
Smart step into / Step out	**Shift + F7 / Shift + F8**
Run to cursor	**Alt + F9**
Evaluate expression	**Alt + F8**
Resume program	**F9**
Toggle breakpoint	**Ctrl + F8**
View breakpoints	**Ctrl + Shift + F8**

Search / Replace

Search everywhere	**Double Shift**
Find	**Ctrl + F**
Find next / previous	**F3 / Shift + F3**
Replace	**Ctrl + R**
Find in path	**Ctrl + Shift + F**
Replace in path	**Ctrl + Shift + R**
Select next occurrence	**Alt + J**
Select all occurrences	**Ctrl + Alt + Shift + J**
Unselect occurrence	**Alt + Shift + J**

—**Productivity Boosters**

eit Digital

Editing

Basic code completion	**Ctrl + Space**
Smart code completion	**Ctrl + Shift + Space**
Complete statement	**Ctrl + Shift + Enter**
Parameter info (within method call arguments)	**Ctrl + P**
Quick documentation lookup	**Ctrl + Q**
External Doc	**Shift + F1**
Brief Info	**Ctrl +mouse**
Show descriptions of error at caret	**Ctrl + F1**
Generate code...	**Alt + Insert**
Override methods	**Ctrl + O**
Implement methods	**Ctrl + I**
Surround with...	**Ctrl + Alt + T**
Comment / uncomment with line comment	**Ctrl + /**
Comment / uncomment with block comment	**Ctrl + Shift + /**
Extend selection	**Ctrl + W**
Shrink selection	**Ctrl + Shift + W**
Context info	**Alt + Q**
Show intention actions and quick-fixes	**Alt + Enter**
Reformat code	**Ctrl + Alt + L**
Optimize imports	**Ctrl + Alt + O**
Auto-indent line(s)	**Ctrl + Alt + I**
Indent / unindent selected lines	**Tab / Shift + Tab**
Cut current line to clipboard	**Ctrl+X,Shift+Delete**
Copy current line to clipboard	**Ctrl+C,Ctrl+Insert**
Paste from clipboard	**Ctrl+V,Shift+Insert**
Paste from recent buffers...	**Ctrl+Shift + V**
Duplicate current line	**Ctrl+D**
Delete line at caret	**Ctrl+Y**
Smart line join	**Ctrl+Shift + J**
Smart line split	**Ctrl+Enter**
Start new line	**Shift + Enter**
Toggle case for word at caret or selected block	**Ctrl + Shift + U**
Select till code block end / start	**Ctrl + Shift +] / [**
Delete to word end	**Ctrl + Delete**
Delete to word start	**Ctrl + Backspace**
Expand / collapse code block	**Ctrl + NumPad+ / -**
Expand all	**Ctrl+Shift+NumPad+**
Collapse all	**Ctrl+Shift+NumPad-**
Close active editor tab	**Ctrl + F4**

Refactoring

Copy	**F5**
Move	**F6**
Safe Delete	**Alt + Delete**
Rename	**Shift + F6**
Refactor this	**Ctrl + Alt + Shift + T**
Change Signature	**Ctrl + F6**
Inline	**Ctrl + Alt + N**
Extract Method	**Ctrl + Alt + M**
Extract Variable	**Ctrl + Alt + V**
Extract Field	**Ctrl + Alt + F**
Extract Constant	**Ctrl + Alt + C**
Extract Parameter	**Ctrl + Alt + P**

Navigation

Go to class	**Ctrl + N**
Go to file	**Ctrl + Shift + N**
Go to symbol	**Ctrl + Alt + Shift + N**
Go to next / previous editor tab	**Alt + Right/Left**
Go back to previous tool window	**F12**
Go to editor (from tool window)	**Esc**
Hide active or last active window	**Shift + Esc**
Go to line	**Ctrl+G**
Recent files popup	**Ctrl+E**
Navigate back / forward	**Ctrl+Alt + Left/Right**
Navigate to last edit location	**Ctrl+Shift+Backspace**
Select current file or symbol in any view	**Alt + F1**
Go to declaration	**Ctrl + B , Ctrl + Click**
Go to implementation(s)	**Ctrl + Alt + B**
Open quick definition lookup	**Ctrl + Shift + I**
Go to type declaration	**Ctrl + Shift + B**
Go to super-method / super-class	**Ctrl + U**
Go to previous / next method	**Alt + Up/Down**
Move to code block end / start	**Ctrl +]/[**
File structure popup	**Ctrl + F12**
Type hierarchy	**Ctrl + H**
Method hierarchy	**Ctrl + Shift + H**
Call hierarchy	**Ctrl + Alt + H**
Next / previous highlighted error	**F2 / Shift + F2**
Edit source / View source	**F4 / Ctrl + Enter**
Show navigation bar	**Alt + Home**
Toggle bookmark	**F11**
Toggle bookmark with mnemonic	**Ctrl + F11**
Go to numbered bookmark	**Ctrl + #[0-9]**
Show bookmarks	**Shift + F11**

Compile and Run

Make project	**Ctrl + F9**
Compile selected file, package or module	**Ctrl + Shift + F9**
Select configuration and run / debug	**Alt + Shift + F10/F9**
Run / Debug	**Shift + F10 / F9**
Run context configuration from editor	**Ctrl + Shift + F10**

Usage Search

Find usages / Find usages in file	**Alt + F7 / Ctrl + F7**
Highlight usages in file	**Ctrl + Shift + F7**
Show usages	**Ctrl + Alt + F7**

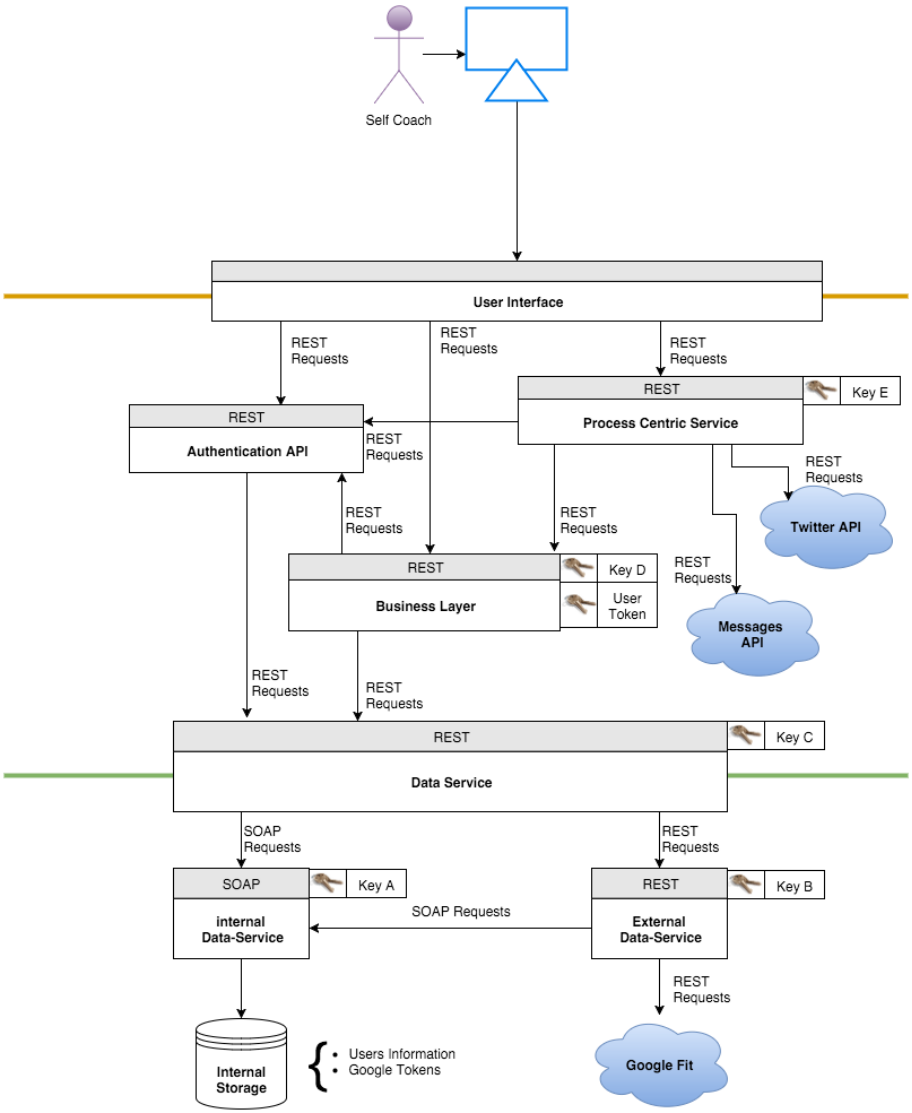
VCS / Local History

Commit project to VCS	**Ctrl + K**
Update project from VCS	**Ctrl + T**
Push commits	**Ctrl + Shift + K**
'VCS' quick popup	**Alt + BackQuote (`)**

Live Templates

Surround with Live Template	**Ctrl + Alt + J**
Insert Live Template	**Ctrl + J**

The project


Self Coach
User Interface
REST Requests
REST Requests
REST Requests
REST
Key E
Process Centric Service
REST
Authentication API
REST Requests
REST Requests
REST Requests
Twitter API
REST Requests
REST Requests
REST
Key D
Business Layer
User Token
REST Requests
Messages API
REST Requests
REST Requests
REST
Key C
Data Service
SOAP Requests
REST Requests
SOAP
Key A
internal Data-Service
SOAP Requests
REST
Key B
External Data-Service
REST Requests
Internal Storage
Users Information
Google Tokens
Google Fit

Unit test

- Def: A unit test is a piece of a code (usually a method) that invokes another piece of code and checks the correctness of some assumption afterward. If the assumptions turn out to be wrong, the unit test has failed. A unit is a method or function.
- Def: SUT stand for system under test, and some people like to use CUT (class under test or code under test). When you test something, you refer to the thing you're testing as the SUT.

Unit test cont.

- Def: A unit of work is the sum of actions that take place between the invocation of a public method in the system and a single noticeable end result by a test of that system. A noticeable end result can be observed without looking at the internal state of the system and only through its public APIs and behavior. An end result is any of the following:
- The invoked public method returns a value
- There is a noticeable change in the state of behavior, that can be determined by interrogating the internal state
- There is a callout to a 3rd – party system where over the test has no control

Unit test cont.

- Updated def.: A unit test is a piece of code that invokes a unit of work and checks one specific end result of that unit of work, whereas it is fully isolated. If the assumptions on the end result turn out to be wrong, the unit test has failed. A unit test's scope can span as little as a method or as much as multiple classes.
- The primary goal of unit testing is to
  - Take the smallest piece of testable software in the application
  - Isolate it from the remainder of the code and
  - Determine whether it behaves exactly as you expected

eit Digital

Unit test cont.

- Practically, a unit test is a piece of code and
  - Invokes another piece of code
  - Checks the correctness of some assumption afterward.
  - If the assumption turn out to be wrong, the unit test has failed. A unit is a method or function.
- THE IMPORTANCE OF A GOOD UNIT TEST

Properties of a good unit test

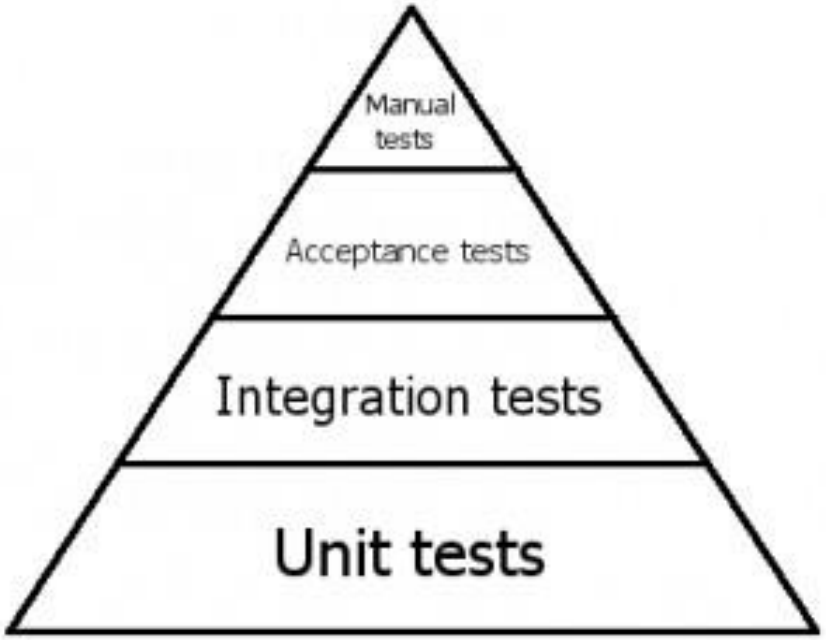
- A unit test should have the following props.
  - It should be automated and maintainable
  - It should be easy to implement
  - It should be relevant tomorrow
  - Anyone should be able to run it at the push of a button
  - It should run quickly
  - It should be consistent in its results
  - It should be fully isolated
  - When it fails, it should be easy to detect what was expected and determine how to pinpoint the problem

Integration tests

- Def.: Integration testing is testing a unit of work without having full control over all of it and using one or more of its real dependencies, such as time, network, database, threads, random number generators, and so on.
- The complete Guide to Software testing by Bill Hetzel (Wiley, 1993), integration testing is “an orderly progression of testing in which software and/or hardware elements are combined and tested until the entire system has been integrated.”

Unite test cont.

- Def.:(final) A unit test is an automated piece of code that invokes the unit of work being tested, and then checks some assumptions about a single end result of that unit. A unit test is almost always written using a unit testing framework. It can be written easily and runs quickly. It's trustworthy, readable, and maintainable. It's consistent in it's results as long as production code hasn't changed.
- A **Simple** unit testing example


Manual tests
Acceptance tests
Integration tests
Unit tests

eit Digital

Test – driven development

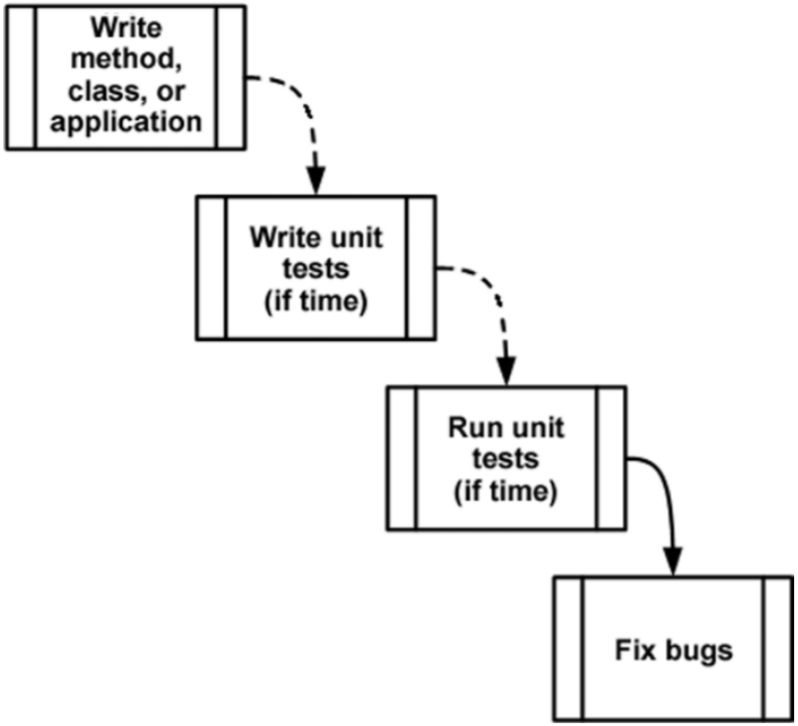


Figure 1.3 The traditional way of writing unit tests. The broken lines represent actions people treat as optional.

Test – driven development cont.

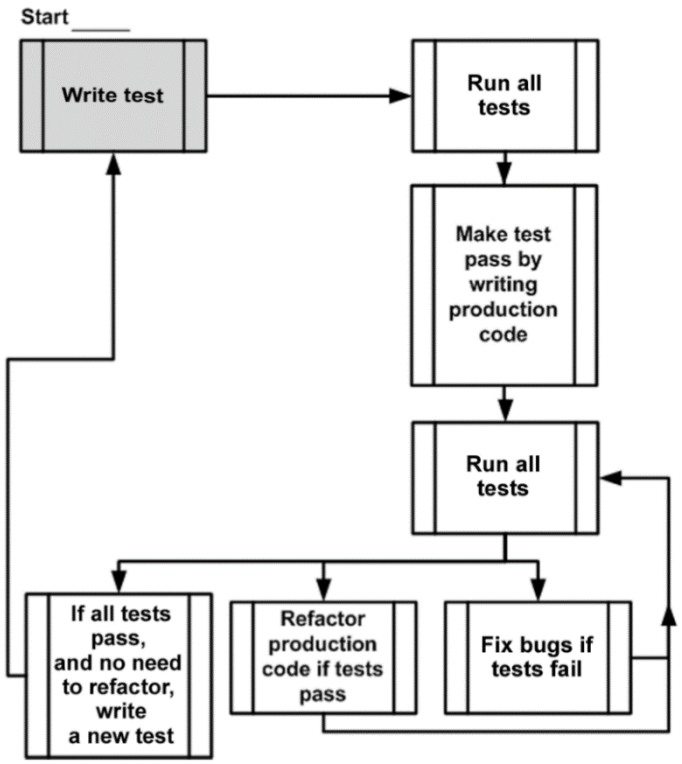


Figure 1.4 Test-driven development—a bird's-eye view. Notice the spiral nature of the process: write test, write code, refactor, write next test. It shows the incremental nature of TDD: small steps lead to a quality end result.

Technique of TDD in a nutshell

- Write failing test to prove code or functionality is missing from the end product.
- Make the test pass by writing production code that meets the expectations of your test.
- Refactor your code.

And don’t forget to make them fast

eit Digital

eitdigital.eu

http://www.doctoralschool.eitdigital.eu/doctoral-training-centres/dtc-budapest/