Agile Software Development

L02 - Test-Driven Development (TDD)

Ezequiel Scott
ezequiel.scott@ut.ee
Agenda

• Recap
• What is Agile Software Development (ASD)?
• Current status of ASD
• Test Driven Development (TDD)
Learning goals

To introduce some of the **practices on agile software development**, taking as an **example** the development of applications labelled as “software as a service”
Approach

• Learn the bases of ASD
  • Definitions, TDD, Scrum...

• Learn a technology stack (for SaaS)
  • Elixir, Phoenix, Vue.js...

• Put them into practice (guided during the labs)

• Develop a project (team work) from conception to deployment
  • Applying agile practices
  • Using a technology stack
How to pass the course

- **Homeworks** (submitted in pairs) 10 points
  - 2 lab assignments * 5 points
- **Project** (team-based, 4 members) 35 points
  - Evidence of the use of agile practices
  - Assessment of the delivered software (code + tests)
  - Sprint reviews (weekly)
    - commitment per sprint * completeness
    - Max: (5 sprints * 7 points) * 1.0
- **Final exam** 55 points
  - You need a mark of at least 23 points out of 55 to pass the course
  - Structure of the exam: Theory (20p) / Practice (35p)
The three P’s in Software Development Projects

Software project

“People, Process, and Product” by Dwayne Phillips
http://dwaynephillips.net/CutterPapers/ppp/ppp.htm

Adapted from Dietmar Pfahl’s course on Software Engineering Management at UT
Software process

• A process defines **who** does **what**, **when**, and **how** to reach a specific goal

• In **software engineering**, the goal is to build a software product or to enhance an existing one

• Software engineering is an engineering discipline that is applied to the development of software in a **systematic approach** (called a software process)

Software process examples

Waterfall

V Model

Scrum

RUP

Spiral
Software processes timeline

Figure 1. Forty years of information systems development methodologies

Software processes timeline

Figure 1. Forty years of information systems development methodologies

The Agile Manifesto

**Individuals and interactions** over processes and tools.

**Working software** over comprehensive documentation.

**Customer collaboration** over contract negotiation.

**Responding to change** over following a plan.

http://agilemanifesto.org/
What is Agile Software Development?

• Agile software development is more than frameworks such as Scrum, XP, or FDD.

• Agile software development is more than practices such as pair programming, TDD, stand-ups, planning sessions and sprints.

• Agile software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software.

https://www.agilealliance.org/agile101/
The Agile Umbrella

Scrum  FDD  ASD  Lean
Kanban  XP  DSDM  Crystal
Values, Principles, Practices, and Frameworks

• 4 values

• 12 principles

• A practice consists of a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis

• A framework conceptually organizes practices to facilitate the software development
Subway map to agile practices
https://www.agilealliance.org/agile101/subway-map-to-agile-practices/
Agenda

• Recap
• What is Agile Software Development (ASD)?
• Current status of ASD
• Test Driven Development (TDD)
Current status of Agile

The survey was conducted between August and December 2018. Sponsored by CollabNet VersionOne. Only 17% of the respondents (N=1319) were CollabNet VersionOne customers, indicating the range and diversity of respondents.
Agile methods and practices

Scrum and Scrum/XP Hybrid (64%) continue to be the most common agile methodologies used.
Engineering practices

The overall rank order of engineering practices employed remained the same this year with exception of one new addition to the survey: **Continuous delivery**.

<table>
<thead>
<tr>
<th>Engineering Practice</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit testing</td>
<td>69%</td>
</tr>
<tr>
<td>Coding standards</td>
<td>58%</td>
</tr>
<tr>
<td>Continuous integration</td>
<td>53%</td>
</tr>
<tr>
<td>Refactoring</td>
<td>41%</td>
</tr>
<tr>
<td>Continuous delivery</td>
<td>40%</td>
</tr>
<tr>
<td>Continuous deployment</td>
<td>35%</td>
</tr>
<tr>
<td>Pair programming</td>
<td>34%</td>
</tr>
<tr>
<td>Test-driven development (TDD)</td>
<td>33%</td>
</tr>
<tr>
<td>Automated acceptance testing</td>
<td>33%</td>
</tr>
<tr>
<td>Collective code ownership</td>
<td>31%</td>
</tr>
<tr>
<td>Sustainable pace</td>
<td>25%</td>
</tr>
<tr>
<td>Behavior-driven development (BDD)</td>
<td>22%</td>
</tr>
<tr>
<td>Emergent design</td>
<td>14%</td>
</tr>
</tbody>
</table>
Hybrid processes – HELENA survey

691 completed responses
55+ countries

Worldwide trend – HELENA survey


Frameworks/Methods
Which of the following frameworks and methods do you use?

- Scrum: 34% use it, 14% use it sometimes, 11% use it rarely, 8% use it never.
- Iterative Development: 27% use it, 23% use it sometimes, 19% use it rarely, 12% use it never.
- Kanban: 23% use it, 16% use it sometimes, 14% use it rarely, 10% use it never.
- DevOps: 23% use it, 15% use it sometimes, 12% use it rarely, 10% use it never.
- Classic Waterfall Process: 20% use it, 16% use it sometimes, 14% use it rarely, 10% use it never.
- eXtreme Programming (XP): 19% use it, 15% use it sometimes, 12% use it rarely, 10% use it never.
- Lean Software Development: 18% use it, 15% use it sometimes, 12% use it rarely, 10% use it never.
- Domain-Driven Design: 17% use it, 15% use it sometimes, 12% use it rarely, 10% use it never.
- ScrumBan: 17% use it, 15% use it sometimes, 12% use it rarely, 10% use it never.
- Feature Driven Development (FDD): 16% use it, 14% use it sometimes, 12% use it rarely, 10% use it never.
- V-shaped Process (V-Model): 16% use it, 14% use it sometimes, 12% use it rarely, 10% use it never.
- Phase / Stage-gate model: 16% use it, 14% use it sometimes, 12% use it rarely, 10% use it never.
- Model-Driven Architecture (MDA): 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Scaled Agile Framework (SAFe): 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Team Software Process: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Personal Software Process: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Nexus: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Large-Scale Scrum (LESS): 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- SSADM: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Spiral Model: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Dynamic Systems Development Method: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Crystal Family: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- PRINCE2: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.
- Rational Unified Process: 15% use it, 13% use it sometimes, 11% use it rarely, 9% use it never.

Scott, E. et al. Initial results of the HELENA survey conducted in Estonia with comparison to results from Sweden and worldwide. In PROFES’17 (pp. 404-412). Springer, Cham.
What happens in Estonia? (HELENA)


Scott, E. et al. Initial results of the HELENA survey conducted in Estonia with comparison to results from Sweden and worldwide. In PROFES'17 (pp. 404-412). Springer, Cham.
What happens in Sweden? (HELENA)


Scott, E. et al. Initial results of the HELENA survey conducted in Estonia with comparison to results from Sweden and worldwide. In PROFES’17 (pp. 404-412). Springer, Cham.
Agenda

• Recap

• What is Agile Software Development (ASD)?

• Current status of ASD

• Test Driven Development (TDD)
### Testing Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Definition and Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance Testing (AT)</td>
<td>The level of the software testing process where a system is tested for acceptability. The purpose of AT is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.</td>
</tr>
<tr>
<td>System Testing (ST)</td>
<td>The level of the software testing process where a complete, integrated system/software is tested. The purpose of ST is to evaluate the system’s compliance with the specified requirements.</td>
</tr>
<tr>
<td>Integration Testing (IT)</td>
<td>The level of the software testing process where individual units are combined and tested as a group. The purpose of IT is to expose faults in the interaction between integrated units.</td>
</tr>
<tr>
<td>Unit Testing (UT)</td>
<td>The level of the software testing process where individual units/components of a software/system are tested. The purpose of UT is to validate that each unit of the software performs as designed.</td>
</tr>
</tbody>
</table>
Test Driven Development (TDD)

- TDD completely turns traditional development around
- Goals?
  - To think through your requirements/design before you write your functional code
  - To write clean code that works
Test Driven Development (TDD)

\[ \text{TDD} = \text{TFD} + \text{Refactoring} \]

**TFD** = test-first development → you write a test before you write just enough production code to fulfill that test.

**Refactoring** = a programming technique where the code is restructured through small changes (Martin Fowler, 1999)
TDD workflow

1. Add a test
2. Run the tests
   - [Fail]
   - Refactoring
3. Make a little change
4. Run the tests
   - [Fail]
5. [Pass, Development continues]
6. [Pass, Development stops]
TDD workflow

1. Write a “single” unit test describing an aspect of the program
2. Run the test, which should fail because the program lacks that feature
3. Write “just enough” code, the simplest possible, to make the test pass
4. “Refactor” the code until it conforms to the simplicity criteria
5. Repeat, “accumulating” unit tests over time
TDD Example

As a bank customer
I want to check the strength of my password
so that I don’t get hacked easily

* A password should have
between 5 and 10 characters

TDD Example

*The password should be between 5 to 10 characters.

```java
package Prac;
import org.testng.Assert;
import org.testng.annotations.Test;

public class TestPassword {
    @Test
    public void TestPasswordLength() {
        PasswordValidator pv = new PasswordValidator();
        Assert.assertEquals(true, pv.isValid("Abc123"));
    }
}
```

TDD Example

*The password should be between 5 to 10 characters.

```java
package Prac;

import org.testng.Assert;
import org.testng.annotations.Test;

public class TestPassword {
    @Test
    public void TestPasswordLength() {
        PasswordValidator pv = new PasswordValidator();
        Assert.assertEquals(true, pv.isValid("Abc123"));
    }
}
```

We can not run test because this class is not created yet

This is main validation test

TDD Example (2)

*The password should be between 5 to 10 characters.*

```java
package Prac;

class PasswordValidator {
    public boolean isValid(String Password) {
        if (Password.length() >= 5 && Password.length() <= 10) {
            return true;
        } else {
            return false;
        }
    }
}
```

This is main condition checking length of password. If meets return true otherwise false.

TDD Example (3)

*The password should be between 5 to 10 characters.

TDD Example (4)

*The password should be between 5 to 10 characters.

```java
package Prac;

import org.testng.Assert;
import org.testng.annotations.Test;

public class TestPassword {

    @Test
    public void TestPasswordLength() {
        //PasswordValidator pv = new PasswordValidator();
        Assert.assertEquals(true, pv.isValid("Abc123"));
    }
}
```

[Visit the source on Guru99](https://www.guru99.com/test-driven-development.html)
**TDD Example (4)**

*The password should be between 5 to 10 characters.*

```java
package Prac;

import org.testng.Assert;
import org.testng.annotations.Test;

public class TestPassword {

    @Test
    public void TestPasswordLength() {
        Assert.assertEquals(true, PasswordValidator.isValid("Abc123"));
    }
}
```

Re-factor code as there is no need of creating instance of class PasswordValidator().

TDD Example (5)

*The password should be between 5 to 10 characters.

If we removed instance creation statement compiler will give error. As we do not create instance it becomes non static method and there is no any reference to this method. Test results in Fail. To remove this error we have to make isValid() method of class PasswordValidator as static.

TDD Example (6)

*The password should be between 5 to 10 characters.

```java
package Prac;

public class PasswordValidator {
    public static boolean isValid(String Password) {
        if (Password.length()>=5 && Password.length()<=10) {
            return true;
        } else {
            return false;
        }
    }
}
```

TDD Example (7)

*The password should be between 5 to 10 characters.

Test results passed as we changed code in class PasswordValidator().

More examples?

• First practice session of ASD (using Elixir)
  https://orlenyslp.gitlab.io/ASD-19/labs/practical1/


• FizzBuzz by Davo
  https://learnitmyway.com/tdd-example/
Myths and Misconceptions

• You create a 100% regression test suite
  • Reusable components/frameworks and user interfaces are not usually covered by the tests

• The unit tests form 100% of your design specification
  • Design is much more than unit tests

• You only need to unit test
  • Complex systems need other testing techniques

• TDD is sufficient for testing
  • TDD is only part of your overall testing efforts.