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)
Agenda

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

✓ To explain basic concepts of Agile Software Development and current practices.

✓ To implement agile practices during the development of a web application.
Approach

• Learn the basic concepts of ASD
  • Definitions, TDD, Scrum...

• Learn a technology stack
  • Elixir, Phoenix ...

• Put them into practice (guided during the labs)

Then...

• Develop a project (teamwork!)
  from conception to deployment
  • Applying agile practices
  • Using a technology stack
The three P’s in Software Development Projects

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

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

RUP

Scrum

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

Agenda

• Recap

• **What is Agile Software Development (ASD)?**

• Current status of ASD

• Test Driven Development (TDD)
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

Agile

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

• **4 values** [http://agilemanifesto.org/](http://agilemanifesto.org/)

• **12 principles** [http://agilemanifesto.org/principles.html](http://agilemanifesto.org/principles.html)

• 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
The Agile Mindset

“Agile is a mindset (established set of attitudes and habits towards) succeeding when there is uncertainty”

BEING AGILE

https://www.slideshare.net/AgileNZ/ahmed-sidky-keynote-agilenz
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 2019. Sponsored by CollabNet VersionOne. Only 14% of the respondents (N=1121) were CollabNet VersionOne customers, indicating the range and diversity of respondents.

https://stateofagile.com/
Agile methods and practices

Scrum and Scrum variants continue to be the most common agile methodologies used.
Engineering practices

ENGINEERING PRACTICES EMPLOYED
The overall rank order of engineering practices employed remained almost the same this year over last. Automated acceptance testing increased 3% while pair programming, test-driven development, and behavior-driven development each fell 3%.

- **UNIT TESTING**: 67%
- **CODING STANDARDS**: 58%
- **CONTINUOUS INTEGRATION**: 55%
- **REFACTORING**: 43%
- **CONTINUOUS DELIVERY**: 41%
- **AUTOMATED ACCEPTANCE TESTING**: 36%
- **CONTINUOUS DEPLOYMENT**: 36%
- **PAIR PROGRAMMING**: 31%
- **TEST-DRIVEN DEVELOPMENT**: 30%
- **COLLECTIVE CODE OWNERSHIP**: 29%
- **SUSTAINABLE PACE**: 23%
- **BEHAVIOR-DRIVEN DEVELOPMENT (BDD)**: 19%
- **EMERGENT DESIGN**: 13%

*Respondents were able to make multiple selections*
Hybrid processes – HELENA survey

691 completed responses
55+ countries

Worldwide trend – HELENA survey


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)


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

- Scrum
- Iterative Development
- Kanban
- DevOps
- Classic Waterfall Process
- eXtreme Programming (XP)
- Lean Software Development
- Domain-Driven Design
- ScrumBan
- Feature Driven Development (FDD)
- V-shaped Process (V-Model)
- Phase / Stage-gate model
- Model-Driven Architecture (MDA)
- Scaled Agile Framework (SAFe)
- Team Software Process
- Personal Software Process
- Nexus
- Large-Scale Scrum (LESS)
- SSADM
- Spiral Model
- Dynamic Systems Development Method
- Crystal Family
- PRINCE2
- Rational Unified Process

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.
Does agile work?
Does agile work?

• Why would you want to do Agile?
  ... see the facts (next slide)

• Agile does not work for big projects with set deadlines
  ... deadlines are important in agile (time-box)

• Agile Methodologies are not holistic
  ... see agile @ scale

• All those concepts were already around since long time ago
  ... holistic perspectives are not useful?

• Where is the evidence?
  ... have a look at Google Scholar (~377000 results)
REASONS FOR ADOPTING AGILE

Accelerating software delivery and enhancing ability to manage changing priorities remain the top reasons stated for adopting Agile. Respondents indicated this year that reasons for adoption were less about reducing project cost (26% compared to 41% last year), and more about reducing project risk (37% compared to 28% last year).

<table>
<thead>
<tr>
<th>Reason</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerate software delivery</td>
<td>71%</td>
</tr>
<tr>
<td>Enhance ability to manage changing priorities</td>
<td>63%</td>
</tr>
<tr>
<td>Increase productivity</td>
<td>51%</td>
</tr>
<tr>
<td>Improve business/IT alignment</td>
<td>47%</td>
</tr>
<tr>
<td>Enhance software quality</td>
<td>42%</td>
</tr>
<tr>
<td>Enhance delivery predictability</td>
<td>39%</td>
</tr>
<tr>
<td>Reduce project risk</td>
<td>37%</td>
</tr>
<tr>
<td>Improve project visibility</td>
<td>36%</td>
</tr>
<tr>
<td>Improve team morale</td>
<td>36%</td>
</tr>
<tr>
<td>Reduce project cost</td>
<td>36%</td>
</tr>
<tr>
<td>Improve engineering discipline</td>
<td>36%</td>
</tr>
<tr>
<td>Better manage distributed teams</td>
<td>36%</td>
</tr>
<tr>
<td>Increase software maintainability</td>
<td>36%</td>
</tr>
</tbody>
</table>

*Respondents were able to make multiple selections.

A decade of agile methodologies: Towards explaining agile software development

**Abstract**

Ever since the agile manifesto was created in 2001, the research community has devoted a great deal of attention to agile software development. This article examines publications and citations to illustrate how the research on agile has progressed in the 10 years following the articulation of the manifesto. Specifically, we delineate the conceptual structure underlying agile scholarship by performing an analysis of authors who have made notable contributions to the field. Further, we summarize prior research and introduce contributions in this special issue on agile software development. We conclude by discussing directions for future research and urging agile researchers to embrace a theory-based approach in their scholarship.

© 2012 Elsevier Inc. Open access under CC BY-NC-ND license.
The Rise and Evolution of Agile Software Development

Rashina Hoda, University of Auckland
Norsaremah Salleh, International Islamic University Malaysia
John Grundy, Monash University


FIGURE 2. The emergence of trends in agile software development, based on the first relevant publications in the IEEE and ACM digital libraries. SE = software engineering, RE = requirements engineering, and AR/VR = augmented reality or virtual reality.
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)

• Popularized by **Kent Beck (2003)**

• **TDD** completely turns traditional development around

• Goals?
  • To think through your requirements/design before your write your functional code
  • To write clean code that works
Test Driven Development (TDD)

TDD = TFD + Refactoring

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

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

1. Add a test
2. Run the tests
   - [Pass]
   - [Fail]
3. Make a little change
4. Run the tests
   - [Pass]
   - [Fail]

Refactoring

Copyright 2003-2006 Scott W. Ambler
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"));
    }
}
```

[Link to Test-Driven Development](https://www.guru99.com/test-driven-development.html)
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 (2)

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

```java
package Prac;

public 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"));
    }
}
```

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().
```

[Refactor code as there is no need of creating instance of class PasswordValidator.](https://www.guru99.com/test-driven-development.html)
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;
        }
    }
}
```

Re factor : Added static word to pass test.

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://courses.cs.ut.ee/2020/ASD/fall/Main/Practicals


- 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.