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 practice sessions)

Then...

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

Processes can be explicit or implicit

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 processes models

**Waterfall**

- Requirements
- Design
- Implementation
- Verification
- Maintenance

**V Model**

- Requirement Design
- System Design
- Architecture Design
- Module Design
- Integration Test
- Unit Test
- System Test
- Acceptance Test
- Verification Phases
- Validation Phases

**RUP**

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Implementation</td>
<td>Test</td>
<td>Configuration Management</td>
</tr>
<tr>
<td>Deployment</td>
<td>Project Management</td>
<td>Environment</td>
<td></td>
</tr>
</tbody>
</table>

**Scrum**

- Scrum Master
- 24h daily meeting
- Sprint
- Product Owner
- Product Backlog

**Spiral**

- Determine Objectives, Alternatives, Constraints
- Requirements Validation
- Prototype
- Operate Prototype
- Evaluate Alternatives, Identify, Resolve Issues
- Plan Next Phases
- Design Evaluation and Verification
- Implement and Test
- Develop, Verify and Deliver Product
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)
Activity: Presto Manifesto

- Group activity
- Based on experience, write down success factors of project development
- Reach a consensus
- Let’s use this Miro board
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  Crystal
DSDM
Values, Principles, Practices, and Frameworks

• 4 values http://agilemanifesto.org/

• 12 principles 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

DOING AGILE
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

A survey to explore the current trends on ASD. Conducted between February and April 2021, online. Sponsored by Digital.ai (previously VersionOne). 4K responses, 1.3K complete.

<table>
<thead>
<tr>
<th>Where are you located?</th>
<th>Which role best describes your current position?</th>
</tr>
</thead>
<tbody>
<tr>
<td>North America</td>
<td>38% Scrum Master or Internal Coach</td>
</tr>
<tr>
<td>Europe</td>
<td>14% Development Leadership</td>
</tr>
<tr>
<td>Asia</td>
<td>13% Project/Program Manager</td>
</tr>
<tr>
<td>South America</td>
<td>8% External Consultant/Trainer</td>
</tr>
<tr>
<td>Australia/New Zealand</td>
<td>7% Product Manager/Product Owner</td>
</tr>
<tr>
<td>Africa</td>
<td>6% Development Team Member</td>
</tr>
<tr>
<td></td>
<td>4% C Level Exec</td>
</tr>
<tr>
<td></td>
<td>3% Business Analyst</td>
</tr>
<tr>
<td></td>
<td>2% DevOps</td>
</tr>
<tr>
<td></td>
<td>6% Other</td>
</tr>
</tbody>
</table>

[https://stateofagile.com/](https://stateofagile.com/)
Agile methods

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

  "... we have seen the use of Scrum increase from 40% of respondents in the first survey to 66% in the most recent."

- "Other approaches such as extreme programming (XP) dropped from use by almost a quarter of respondents to less than 1% today."
Engineering practices

“... the use of specific individual Agile techniques and practices remained relatively constant [since the first survey]... ”
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
- 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.
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.
HELENA results

✓ only a small number of participants operate their projects in a purely traditional or agile manner (under 15%)
✓ the selection of practices has an effect on the degree of agility of a given discipline
✓ there are no methods or practices that explicitly guarantee or prevent agility.
✓ agility cannot be defined solely at the process level
Does agile work?
Does agile work?

• It is not a silver bullet!
• There are barriers to its adoption*
• Despite these challenges, practitioners continue to see the value of broadening Agile adoption as a way to achieve critical business outcomes*

(* Digital.ai | 15th State of Agile Report

Barriers to adoption*:
- Inconsistencies in processes and practices 46%
- Cultural clashes 43%
- General organizational resistance to change 46%

Key issues in migrating to agile**:

<table>
<thead>
<tr>
<th>Management and organizational</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Organizational Culture</td>
</tr>
<tr>
<td>• Management Style</td>
</tr>
<tr>
<td>• Organizational Form</td>
</tr>
<tr>
<td>• Management of Software Development Knowledge</td>
</tr>
<tr>
<td>• Reward Systems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Working effectively in a team</td>
</tr>
<tr>
<td>• High level of competence</td>
</tr>
<tr>
<td>• Customer relationships—commitment, knowledge, proximity, trust, respect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Change from process-centric to a feature-driven, people-centric approach</td>
</tr>
<tr>
<td>• Short, iterative, test-driven development that emphasizes adaptability</td>
</tr>
<tr>
<td>• Managing large, scalable projects</td>
</tr>
<tr>
<td>• Selecting an appropriate agile method</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technology (Tools and Techniques)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Appropriateness of existing technology and tools</td>
</tr>
<tr>
<td>• New skill sets—refactoring, configuration management, JUnits</td>
</tr>
</tbody>
</table>
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>34%</td>
</tr>
<tr>
<td>Reduce project cost</td>
<td>32%</td>
</tr>
<tr>
<td>Improve engineering discipline</td>
<td>30%</td>
</tr>
<tr>
<td>Better manage distributed teams</td>
<td>29%</td>
</tr>
<tr>
<td>Increase software maintainability</td>
<td>28%</td>
</tr>
</tbody>
</table>

*Respondents were able to make multiple selections.


A decade of agile methodologies: Towards explaining agile software development

ARTICLE INFO

Keywords:
Agile software development
Theory
Software engineering
Information systems
eXtreme programming, XP
Scrum
Lean software development
Crystal method
Feature-driven 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>
How do you do testing?

A. testing software with unit and integration tests on multiple machines

B. testing software with tests, using different techniques (white box, black box)

C. testing if software compiles

D. no testing, client reports bugs
Unit testing

Example class: a calculator that only multiplies two integers

```java
public class Calculator {
    public int multiply(int a, int b) {
        return a * b;
    }
}
```
Unit testing

Base class: a calculator that only multiplies two integers

- We check the functional correctness of the base class by using another class
- We must define the expected behaviour of the system
  - “Multiplying by zero makes the product equal zero”
- Our tests verify actual vs. expected outputs
  - If actual == expected, **PASS**
  - If actual != expected, **FAIL**

- **There are several software testing techniques!!**
  https://courses.cs.ut.ee/2021/SWT2021/spring

```java
public class Calculator {
    public int multiply(int a, int b) {
        return a * b;
    }
}
```

```java
class CalculatorTest {
    Calculator calculator;

    @BeforeEach
    void setUp() {
        calculator = new Calculator();
    }

    @Test
    @DisplayName("Simple multiplication should work")
    void testMultiply() {
        assertEquals(20, calculator.multiply(4, 5), "Regular multiplication should work");
    }

    @DisplayName("Ensure correct handling of zero")
    void testMultiplyWithZero() {
        assertEquals(0, calculator.multiply(0, 5), "Multiple with zero should be zero");
        assertEquals(0, calculator.multiply(5, 0), "Multiple with zero should be zero");
    }
}
```
Unit testing (Elixir)

Base class module:
a calculator that only multiplies

- We check the functional correctness of the base class by using another class of the base module by using another module
- We must define the expected behaviour of the system
  - “Multiplying by zero makes the product equal zero”
- Our tests verify actual vs. expected outputs
  - If actual == expected, PASS
  - If actual != expected, FAIL

```
// lib/calculator.ex
defmodule Calculator do
  def multiply(a, b) do
    a * b
  end
end

// test/calculatorTest.exs
defmodule CalculatorTest do
  use ExUnit.Case
doctest Calculator

  test "Simple multiplication should work" do
    assert Calculator.multiply(4, 5) == 20
  end

  test "Ensure correct handling of zero" do
    assert Calculator.multiply(0, 5) == 0
    assert Calculator.multiply(5, 0) == 0
  end
end
```

```bash
ezequiel@MacEze % mix test test/calculatorTest.exs
.
Finished in 0.03 seconds (0.00s async, 0.03s sync)
2 test, 0 failures```
Test Driven Development (TDD)

- 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, Development continues]
   - [Fail]
5. 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 have 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;

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.

---

<terminated> TestPassword [TestNG] C:\Program Files\Java\jre1.8.0_77\bin\javaw.exe (Jul 25, 2016, 2:10:22 PM)
[TestMethod] Running:
C:\Users\kanchan\AppData\Local\Temp\testng-eclipse--571370159\testng-customsuite.xml

PASSED: TestPasswordLength

Default test
Tests run: 1, Failures: 0, Skips: 0

Default suite
Total tests run: 1, Failures: 0, Skips: 0

---

[TestNG] Time taken by org.testng.reporters.EmailableReporter2@1b40d5f0: 202 ms
[TestNG] Time taken by org.testng.reporters.XMLReporter@28f67ac7: 63 ms
[TestNG] Time taken by org.testng.reporters.jq.Main@546a03af: 78 ms
[TestNG] Time taken by org.testng.reportersJUnitReportReporter@5a01c3a: 2 ms
[TestNG] Time taken by [FailedReporter passed=0 failed=0 skipped=0]: 1 ms
[TestNG] Time taken by org.testng.reporters.SuiteHTMLReporter@2b80d80f: 10 ms

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"));
    }
}
```

Refactor code as there is no need of creating instance of class PasswordValidator().

TDD Example (5)

*The password should be between 5 to 10 characters.

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)


• FizzBuzz by Davo (javascript)
  https://learnitmyway.com/tdd-example/

• Practice session (12:15 – 14:00) (Orlenys)