Agile Software Development

L02 - Test-Driven Development (TDD)

Mariana Falco
mariana.falco@ut.ee
Agenda

1. Quick recap + What is Agile Software Development?
2. Current status of ASD
3. Test Driven Development (TDD)
Quick recap!
Learning goals

✔️ To introduce basic concepts of Agile Software Development and current practices

✔️ The course allows students to implement agile practices during the development of a web application
The 3 P’s in Software Development Projects

Software project

PRODUCT

PEOPLE

PROCESSES

"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

"MTAT.03.295 | Agile Software Development | © Mariana Falco 2022"
Software process

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

• A **software process** is a goal-oriented activity in the context of engineering-style software development
  • systematic approach
  • examples are creation of a product, testing of a system, measuring of a code module...

Software processes models

Waterfall

V Model

RUP

Scrum

Spiral
What is Agile Software Development (ASD)?
The Agile Umbrella

Agile is the ability to create and respond to change

ASD 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 Development.
The Agile Umbrella

Agile is the ability to create and respond to change

ASD 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
The Agile Umbrella

ASD 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 such as Scrum, XP, or FDD.
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/
The Agile Mindset

[established set of attitudes and habits towards] succeeding when there is uncertainty

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

BEING AGILE

https://www.slideshare.net/AgileNZ/ahmed-sidky-keynote-agilenz

MTAT.03.295 | Agile Software Development | © Mariana Falco 2022
Group activity

• Based on experience, write down success factors of project development
• Reach a consensus
• Timebox: 10/15 minutes
**PRINCIPLE 1**
Our highest priority is to **satisfy the customer** through **early and continuous delivery** of valuable software.

**PRINCIPLE 2**
Welcome **changing requirements**, even late in development. Agile processes **harness change** for the customer’s competitive advantage.

**PRINCIPLE 3**
Deliver working software **frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

**PRINCIPLE 4**
Business people and developers must **work together** daily throughout the project.

**PRINCIPLE 5**
Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.

**PRINCIPLE 6**
The most efficient and effective method of conveying information to and within a **development team** is **face-to-face conversation**.
Principles

PRINCIPLE 7
Working software is the primary measure of progress

PRINCIPLE 8
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

PRINCIPLE 9
Continuous attention to technical excellence and good design enhances agility

PRINCIPLE 10
Simplicity—the art of maximizing the amount of work not done—is essential

PRINCIPLE 11
The best architectures, requirements, and designs emerge from self-organizing teams

PRINCIPLE 12
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly
Current status of ADS
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

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.”
“… 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

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)

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

- Scrum: 58%
- Iterative Development: 17% 8% 8% 8%
- Kanban: 17% 8% 8% 8%
- DevOps: 67% 25% 8% 8%
- Classic Waterfall Process: 67% 25% 8% 8%
- eXtreme Programming (XP): 56% 25% 8% 8%
- Lean Software Development: 50% 25% 8% 8%
- Domain-Driven Design: 33% 25% 8% 8%
- ScrumBan: 67% 25% 8% 8%
- Feature Driven Development (FDD): 25% 17% 33% 8% 8%
- V-shaped Process (V-Model): 75% 25% 8% 8%
- Phase / Stage-gate model: 83% 8% 8% 8%
- Model-Driven Architecture (MDA): 50% 8% 42% 8% 8%
- Scaled Agile Framework (SAFe): 67% 25% 8% 8%
- Team Software Process: 83% 8% 8% 8%
- Personal Software Process: 75% 3% 17% 8%
- Nexus: 83% 8% 17% 8%
- Large-Scale Scrum (LESS): 17% 8% 67% 6% 8%
- SSADM: 83% 17% 8% 8%
- Spiral Model: 67% 25% 8% 8%
- Dynamic Systems Development Method: 67% 33% 8% 8%
- Crystal Family: 67% 33% 8% 8%
- PRINCE2: 58% 42% 8% 8%
- Rational Unified Process: 17% 75% 8% 8%

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?

What were the most important reasons for adopting Agile within your team or organization?

- 64% Enhance ability to manage changing priorities
- 64% Accelerate software delivery
- 47% Increase team productivity
- 47% Improve business and IT alignment
- 42% Enhance software quality
- 41% Enhance delivery predictability
- 40% Improve project visibility
- 39% Reduce project risk
- 39% Better respond to volatile market conditions
- 35% Improve team morale
- 24% Improve engineering discipline
- 24% Better manage distributed teams
- 23% Reduce project cost
- 20% Increase software maintainability
- 5% Other

STATE OF AGILE LOOK BACK

The reasons organizations want to adopt Agile remain unchanged for several years. However, we have seen an increase in their ability to meet those goals, particularly as DevOps and agile practices expand across the organization.
Does agile work?

Forty-four percent of respondents report agile transformations at their organizations, but only 14 percent are enterprise-wide.

State of agile transformation, % of respondents (n = 2,190)

- No plans to transform: 25%
- Preparing to launch: 19%
- Currently in progress: 37%
- Completed: 7%
- Born agile\(^1\): 12%

Breakdown of in-progress and completed transformations by scope, %

- Entire organization: 10%
- Excluding operations: 4%
- Multiple units: 22%
- Multiple teams: 8%

\(^1\)Organizations in which agile practices and concepts have always been core to how they work.

Agile for all is coming—but how fast?
Does agile work?

Sectors are transforming at different paces, with telecom and financial services leading the way.

Transformation progress by sector

% of respondents reporting an in-progress or completed transformation

Perceived speed of change in sector

Agile for all is coming—but how fast?
Does agile work?

Cross-functional teams, self-managing teams, and lean are the most commonly applied agile concepts.

Top agile concepts applied, % of respondents reporting an in-progress or completed agile transformation

- Cross-functional teams: 74%
- Self-managing teams: 49%
- Lean: 44%
- Scrum/kanban: 40%
- Capability or competency lead/chapter lead: 34%

Agile for all is coming—but how fast?
Does agile work?

• 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

Barriers to adoption*:

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

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

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>32%</td>
</tr>
<tr>
<td>Reduce project cost</td>
<td>29%</td>
</tr>
<tr>
<td>Improve engineering discipline</td>
<td>27%</td>
</tr>
<tr>
<td>Better manage distributed teams</td>
<td>25%</td>
</tr>
<tr>
<td>Increase software maintainability</td>
<td>24%</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.
Test Driven Development (TDD)
Testing levels

**ACCEPTANCE TESTING**
To evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery

**SYSTEM TESTING**
To evaluate the system’s compliance with the specified requirements

**INTEGRATION TESTING**
To expose faults in the interaction between integrated units

**UNIT TESTING**
To validate that each unit of the software performs as designed
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

E.g.: a calculator that only multiplies two integers

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

E.g.: a calculator that only multiplies two integers
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!!

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

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");
    }
}
```

https://courses.cs.ut.ee/2021/SWT2021/spring
Unit testing (Elixir)

Base module: a calculator that only multiplies

• We check the functional correctness 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
```
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)

• A software development approach in which test cases are developed to specify and validate what the code will do.

• In simple terms, test cases for each functionality are created and tested first and

• if the test fails then the new code is written in order to pass the test and making code simple and bug-free.
Test Driven Development (TDD)

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

- **TFD** = test-first development \(\rightarrow\) 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
   - If pass, go to step 3
   - If fail, go to step 4
3. Make a little change
4. Run the tests
   - If fail, go to step 2
   - If pass, go to step 5
5. [Pass, Development stops]

Refactoring
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

Condition for password acceptance:
- 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

Condition for password acceptance:
- the password should have between 5 to 10 characters.

TDD Example (2)

Condition for password acceptance:
- the password should have 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)

Condition for password acceptance:
- the password should have between 5 to 10 characters.

TDD Example (4)

Condition for password acceptance:
- 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"));
    }
}
```

in method TestPasswordLength (), there is no need of creating an instance of class PasswordValidator. Instance means creating an object of class to refer the members (variables/methods) of that class

[Diagram showing TDD process]

TDD Example (4)

Condition for password acceptance:
- 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() {
        Assert.assertEquals(true, PasswordValidator.isValid("Abc123");
    }
}
```

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

TDD Example (5)

Condition for password acceptance:
- the password should have between 5 to 10 characters.

After refactoring the output shows failed status (see image below) this is because we have removed the instance. So there is no reference to non–static method `isValid()`.
TDD Example (6)

Condition for password acceptance:
- the password should have 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)

Condition for password acceptance:
- the password should have between 5 to 10 characters.

<terminated> TestPassword [TestNG] C:\Program Files\Java\jre1.8.0_77\bin\javaw.exe (Jul 25, 2016, 3:02:16 PM)
[TestNG] Running:
C:\Users\kanchan\AppData\Local\Temp\testng-eclipse--1385484104\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: 19 ms
[TestNG] Time taken by org.testng.reporters.XMLReporter@28f67ac7: 10 ms
[TestNG] Time taken by org.testng.reporters.jq.Main@546a03af: 34 ms

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

Advantages of TDD

• Early bug notification
• Better designed, cleaner and more extensible code
• Confidence to refactor
• Good for teamwork