

SENG 275

# SOFTWARE TESTING

DR. NAVNEET KAUR POPLI

DEPT. OF ELECTRICAL AND COMPUTER  
ENGINEERING



# TDD & BDD

(READ CHAP 8 OF TEXTBOOK)



University  
of Victoria



TDD

DEVELOP CODE  
DRIVEN BY TESTS



University  
of Victoria

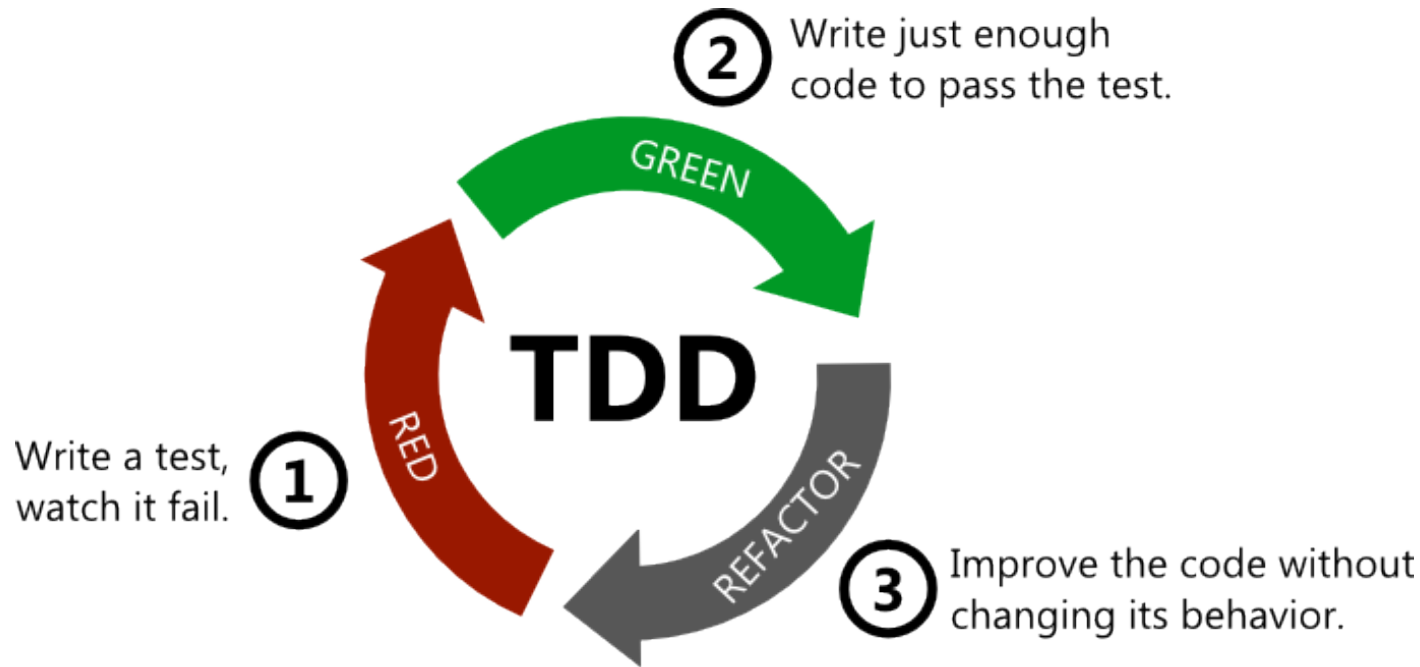
# Rules of the TDD game

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

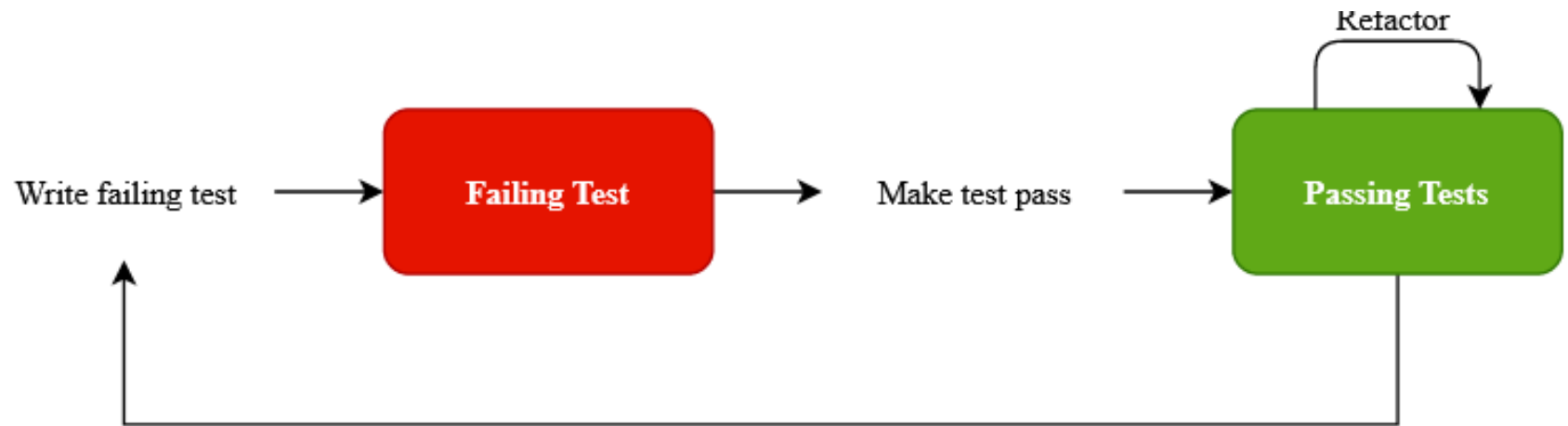


# TDD-Test then Code

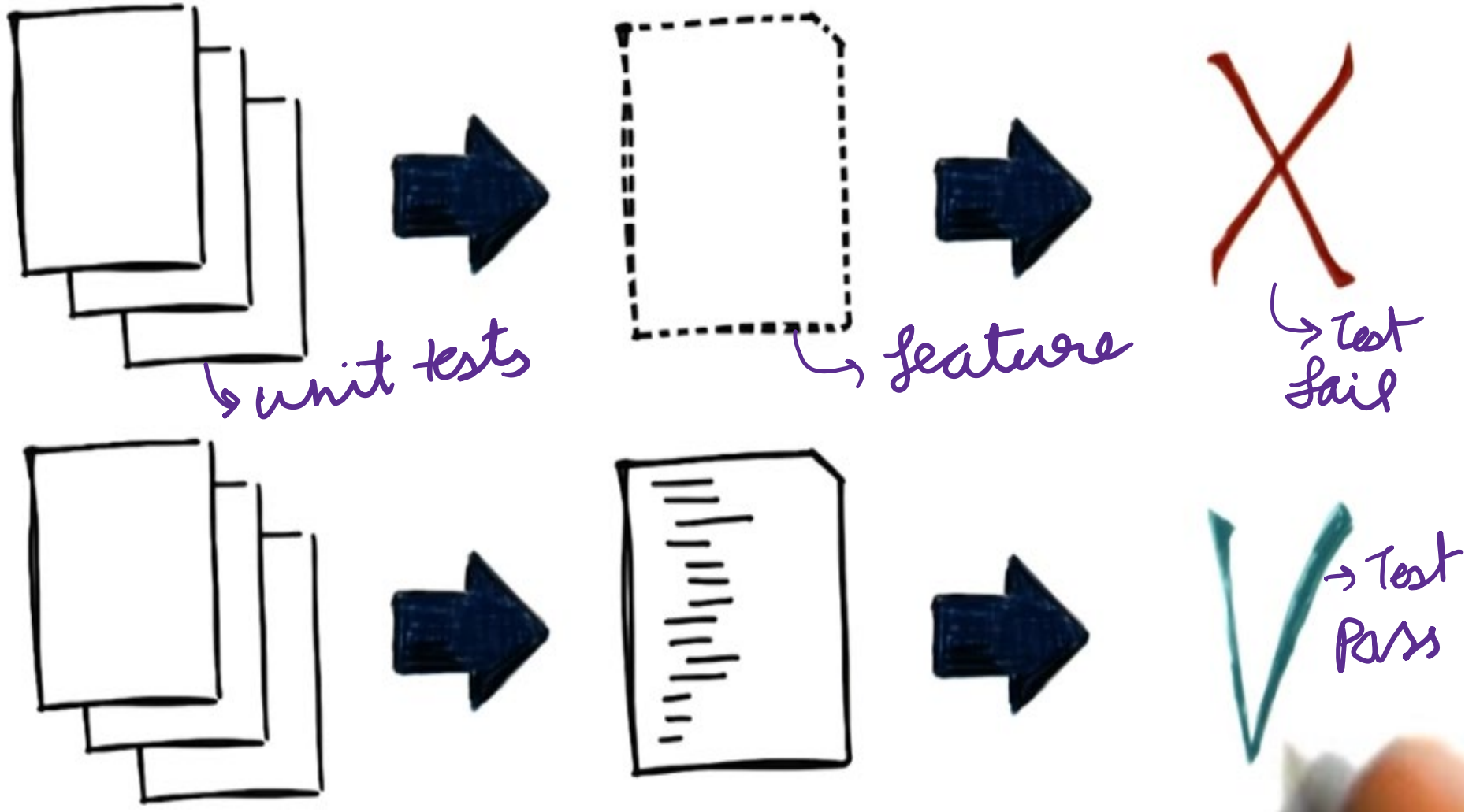
- **TDD (Test-driven development)** is a software development technique that entails writing automated test cases earlier than writing functional pieces of the code.



# TDD



# TEST-FIRST DEVELOPMENT



# Let's play the TDD game

## Returning Customer

I am a returning customer

E-Mail Address

Password


[Forgotten Password](#)

Login






## Play the TDD game



Create test case to verify that if the correct email id and password is added and Login button is clicked, the user is navigated to the home page. Execute the test. Watch it fail.



Implement just enough code to pass this test, no more.



Refactor and improve the code without changing the functional behaviour. For e.g. making the navigation speed better.



# Play the TDD game

1. First, you act like you're a **demanding user** who wants to use the code that's about to be written in the **simplest** possible way. **You must write a test that uses a piece of code as if it were already implemented.**
2. Do not write a bunch of functions/classes that you think you may need. Concentrate on the feature you are implementing and on what is really needed. Writing something the feature doesn't require is **over-engineering** (always beware of it ).



# Advantages

1



• Code Quality

2



• Code Coverage



# Why TDD?

- TDD developers use test cases before they write a single line of code.
- This approach encourages them to consider **how the software will be used** and what design it needs to have to provide the expected usability.
- Once a test fails, developers understand what needs to be changed and they refactor the code, that is, they rewrite it to improve it without altering its function.
- TDD focuses on the code necessary to pass the test, **reducing it to essentials**.
- It takes the **test unit**, or the smallest bit of functionality, as its basis.
- It's a bit like **building a house brick by brick**—no brick is laid down before it's tested, to make sure it's sound and that it's an integral part of the design.
- In TDD, you achieve **100% coverage test**. Every single line of code is tested, unlike traditional testing.



## Other advantages

1. Focus on the requirements
2. Controlling your pace
3. Testable code
4. Faster feedback
5. Revealing design problems early



# Focus on the requirements

Starting by the test means **starting by the requirements**. It makes us think more about:

- what we expect from the class.
- how the class should behave in specific cases.

We do not write "useless code"

- Go to your codebase right now. How much code have you written that is never used in real world?



# Controlling your pace

- Having a failing test, give us a clear focus: **make the test pass.**
- I can write whichever test I want:
  - If I feel insecure, I can write a simpler test.
  - If I feel safe, I can write a more complicated test.
  - If there's something I do not understand, I can take a tiny baby step.
  - If I understand completely, I can take a larger step.



## Testable code

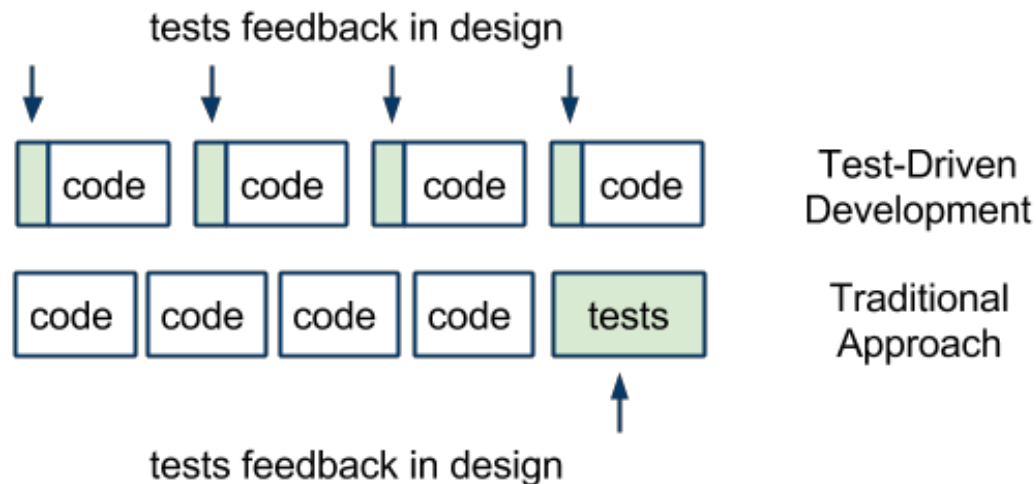
- TDD makes you think about tests from the beginning.
  - This means you will be enforced to **write testable classes**.
- A testable class is also an easy-to-use class.
- Some people call TDD as *Test-Driven Design*.





## Faster feedback

- You are writing tests frequently. This means you will find the problem sooner.
- Tests at the end also work. But maybe the feedback is just too late.



# Listen to your test- reveal design problems early

- The test may reveal design problems.
- You should **"listen to it"**.
- **Too many tests?**
  - Maybe your class **does too much**.
- **Too many mocks?**
  - Maybe your class is **too coupled**.
- **Complex set up** before calling the desired behavior?
  - Maybe rethink the **pre-conditions**.



# UNDERSTANDING TDD USING A CALCULATOR PROGRAM



# Create a junit test



```
import static org.junit.Assert.*;
import org.junit.Test;
// REQUIREMENTS
//0+0=0
//100+1=101
//1+100=101
//-19+19=0
//-19-19=-38
//2147483647+1=2147483648
```

```
public class CalculatorTest {

@Test
public void addTwoNumbers() {
    Calculator obj=new Calculator();
    assertEquals(0, obj.add(0,0));

}

}
```

This test fails because nothing is implemented till now.



University  
of Victoria

When you implement the functionality, the test passes.



```
public class Calculator {  
    public int add(int a, int b)  
    {  
        return a+b;  
    }  
}
```



# I only need to demonstrate that the acceptance criterion have been met.

```
import static org.junit.Assert.*;
import org.junit.Test;
// REQUIREMENTS
//0+0=0
//100+1=101
//1+100=101
//-19+19=0
//-19-19=-38
//2147483647+1=2147483648
```

```
public class CalculatorTest {
```

```
@Test
public void addTwoNumbers() {
    Calculator obj=new Calculator();
    assertEquals(0, obj.add(0,0));
    assertEquals(101, obj.add(100,1));
    assertEquals(101, obj.add(1,100));
}

}
```



University  
of Victoria

I am checking all my acceptance criterion and my tests are passing without further changes to my code.

```
import static org.junit.Assert.*;
import org.junit.Test;
// REQUIREMENTS
//0+0=0
//100+1=101
//1+100=101
//-19+19=0
//-19-1=-20
//2147483647+1=2147483648

public class CalculatorTest {
@Test
public void addTwoNumbers() {
Calculator obj=new Calculator();
assertEquals(0, obj.add(0,0));
assertEquals(101, obj.add(100,1));
assertEquals(101, obj.add(1,100));
assertEquals(0, obj.add(-19,19));
assertEquals(-20, obj.add(-19,-1));}}
```



Checking the domain boundary now. Integers can have max value 2147483647. The test fails here.



```
public class CalculatorTest {  
  
    @Test  
    public void addTwoNumbers() {  
        Calculator obj=new Calculator();  
        assertEquals(0, obj.add(0,0));  
        assertEquals(101, obj.add(100,1));  
        assertEquals(101, obj.add(1,100));  
        assertEquals(0, obj.add(-19,19));  
        assertEquals(-20, obj.add(-19,-1));  
        assertEquals(2147483648, obj.add(2147483647,1));  
    }  
}
```

Markers Properties Servers Data Source Explorer Snippets Console JUnit Coverage

ed after 0.038 seconds

s: 1/1 Errors: 1 Failures: 0

CalculatorTest [Runner: JUnit 4] (0.000 s)

addTwoNumbers (0.000 s)

Failure Trace

java.lang.Error: Unresolved compilation problem:



Now I have to refactor the code and all the tests pass.



```
public class Calculator {  
    public long add(long a, long b)  
    {  
        return a+b;  
    }  
  
}
```



# The last test also passes



```
assertEquals(2147483648L, obj.add(2147483647,1));
```

```
}
```

```
}
```



Markers Properties Servers Data Source Explorer Snippets Console JUnit Coverage

shed after 0.033 seconds

ns: 1/1

Errors: 0

Failures: 0



CalculatorTest [Runner: JUnit 4] (0.000 s)

Failure Trace



University  
of Victoria

# HOW TDD FITS INTO DEVOPS, AGILE AND CI/CD



## WATERFALL

# DEPLOYMENT FREQUENCY



## AGILE

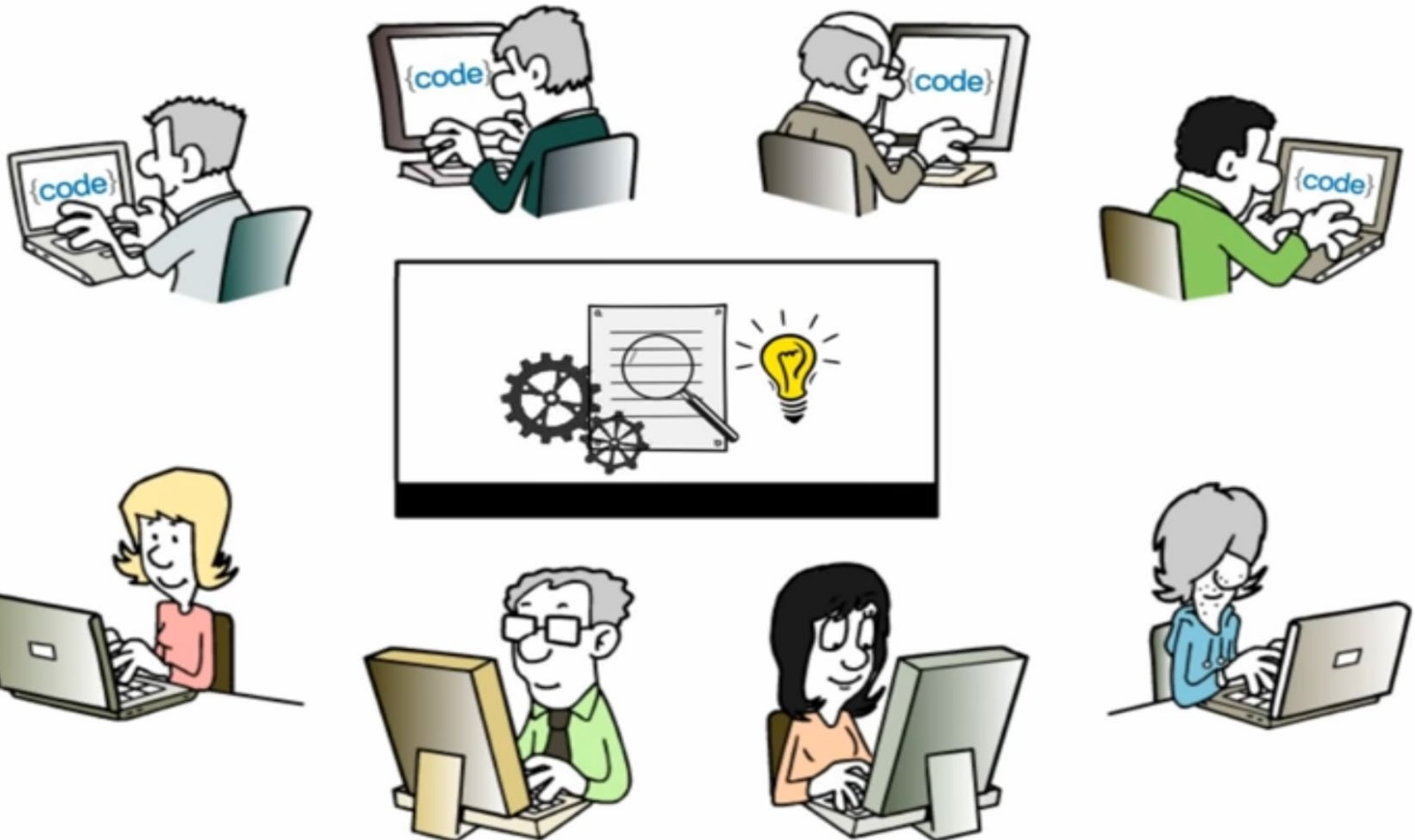


## DEVOPS

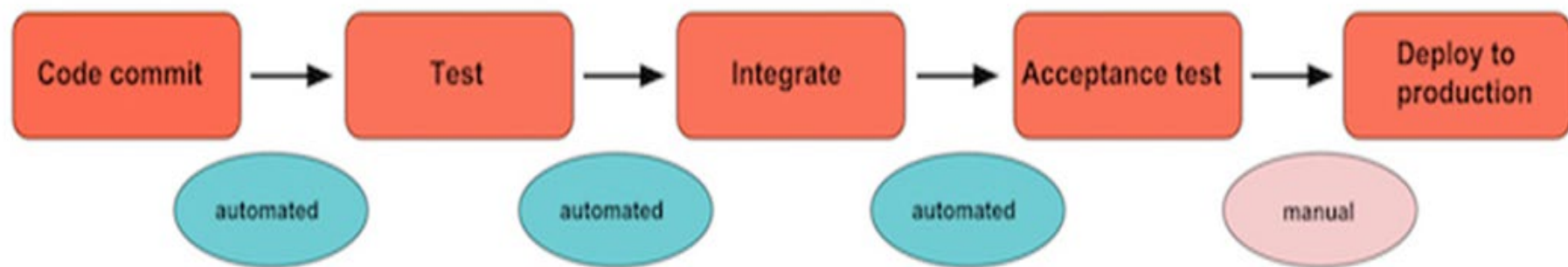


Company	Deployment Frequency
Amazon	23,000 per day
Google	5,500 per day
Netflix	500 per day
Facebook	1 per day
Twitter	3 per week
Typical enterprise	1 every 9 months

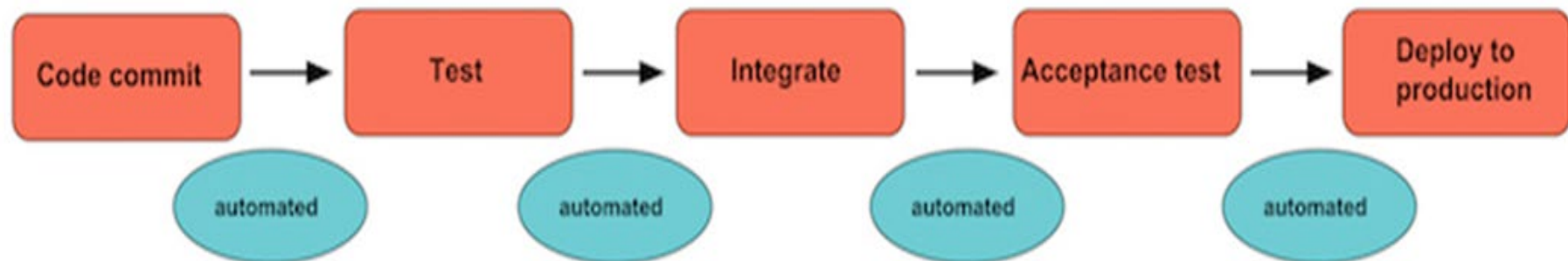
# CI-commit your code often



# Continuous delivery



# Continuous deployment



# WRITING TESTS IN BDD

**“We can’t just start building features before:**

- we have actually articulated how we expect people to use them,**
- what purpose they are meant to serve, and**
- what value they are intended to help us deliver.”**



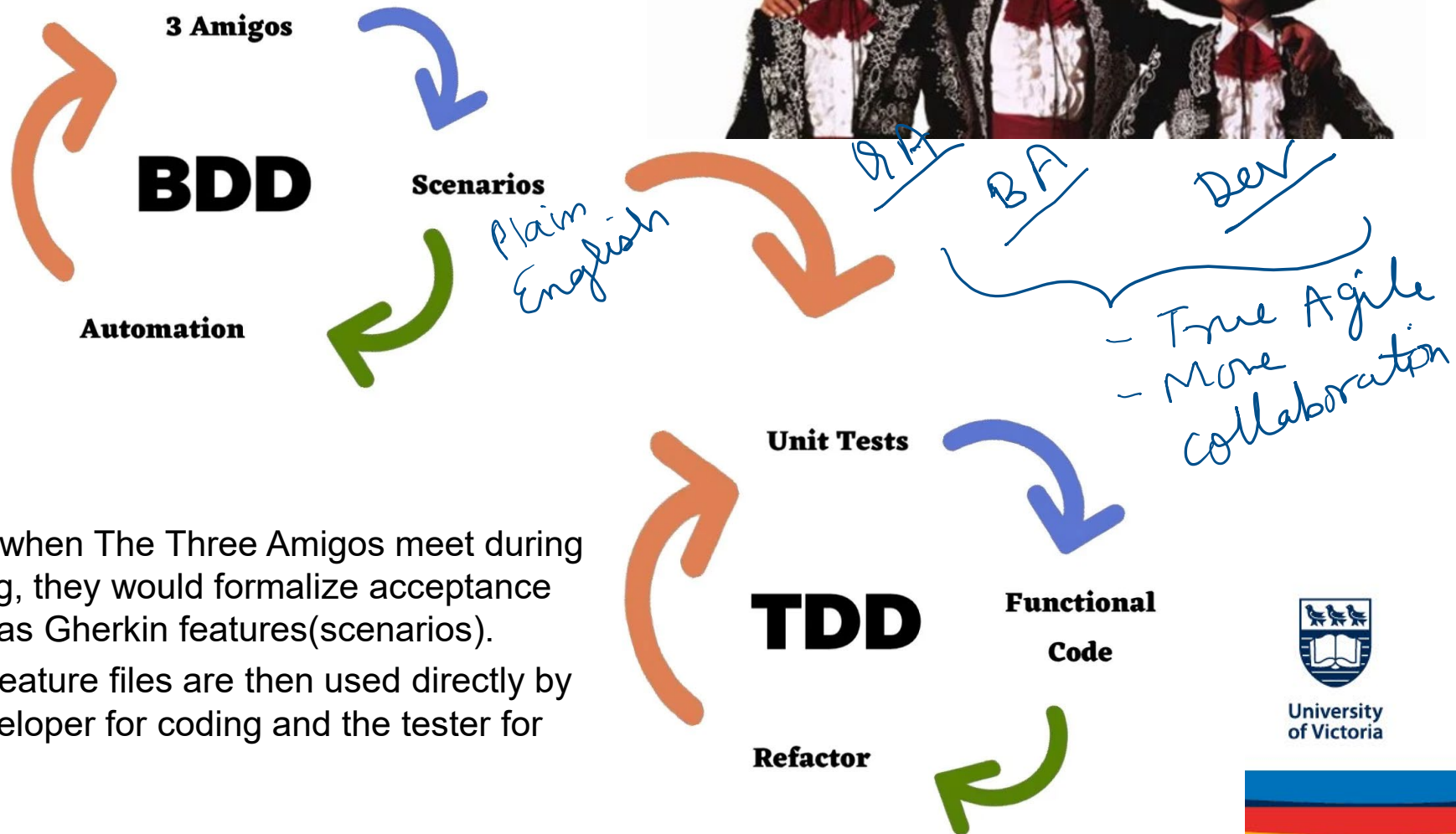


# BDD

- **Behavior Driven Development (BDD)** is a branch of Test-Driven Development (TDD).
- BDD uses **human-readable descriptions of software user requirements** as the basis for software tests.
- An early step in BDD is the definition of **a shared vocabulary between stakeholders, domain experts, and engineers**.
- This process involves the definition of entities, events, and outputs that the **users care about**, and giving them names that everybody can agree on.



# BDD



- Ideally, when The Three Amigos meet during planning, they would formalize acceptance criteria as Gherkin features(scenarios).
- Those feature files are then used directly by the developer for coding and the tester for testing.



University of Victoria

# BDD Acceptance Criteria

**Scenario:** Log in valid credentials

**Given** I have entered correct values for both username and password

UserName	Password
abc	123

**When** I click on login

**Then** User is navigated to dashboard page

**Scenario:** Log in Invalid credentials validation empty fields

**Given** I have left username or password fields empty

UserName	Password

**When** I click on login

**Then** error message should be displayed

**And** Login attempt should fail

**Scenario:** Log in Incorrect username

**Given** I have entered an invalid value for username

UserName	Password
ert	123

**When** I click on login

**Then** error message should be displayed

**And** Login attempt should fail

**Scenario:** Log in Correct username and incorrect password

**Given** I have entered correct username and incorrect password

UserName	Password
abc	456

**When** I click on login

**Then** error message should be displayed

**And** Login attempt should fail

Inputs From End-Users,  
Customers, Team and other  
Stakeholders



Product Owner



Team

1.

2.

3.

4.

5.

6.

7.

Product  
Backlog

Team Selects How  
Much To Commit  
To Do By The End  
Of The Sprint

Sprint Planning  
Meeting

TASKS

Sprint  
Backlog

Product  
Backlog  
Refinement

Scrum Master



Daily Scrum  
Meeting and  
Artifacts Update



Sprint

1-4 Weeks



Sprint Review

Potentially Shippable  
Product Increment



Retrospective

# BDD and TDD in Agile SCRUM, combined with CI/CD/CD and DevOps

1. The business Administrator(BA) sits with the client and gather requirements from them.
2. All requirements are put in the product backlog in a collaborative platform like Jira or Trello.
3. The product owner decides which requirements would be taken up in the first sprint(2-4 week long).
4. The three amigos (PO, Dev, QA) sit together to work on these requirements.
5. Feasibility study is done on those requirements and all ambiguities are removed by clarifying with the client by presenting the requirements as user stories(As a, I want to, So that format) , UML use case diagrams, scenarios and/or mock-ups.



# BDD and TDD in Agile SCRUM, combined with CI/CD/CD and DevOps

6. The requirements are then prioritized according to user specifications, budgets, market demands etc. by giving them story points and applying other prioritization techniques.
7. These requirements are then converted into detailed scenarios. Gherkin frameworks can be used here. Gherkin scenarios can be directly used by tools like Cucumber to generate automated test cases and scripts. At least one positive and one negative scenario has to be created.
8. BDD ensures that a shared vocabulary is used between all stakeholders, domain experts and engineers. Entities, function names, inputs, outputs, data types and other specifications are decided.



# BDD and TDD in Agile SCRUM, combined with CI/CD/CD and DevOps

9. Test Driven Development is applied according to which testers write test cases before even a single piece of development code is written. The test fails. Only enough code is written which is required to pass a single failing test. Test passes. We can refactor.
10. Agile ensures constant feedback from the client. A daily stand-up meeting of about 15 minutes is carried out among the team to discuss what went good, what went wrong, what are the plans ahead etc. Client may or may not be part of these meetings.
11. When all tests pass and all acceptance criteria are met a feature is released. This may be a MVP(Minimum viable Product) or a feature addition.





# BDD and TDD in Agile SCRUM, combined with CI/CD/CD and DevOps

12. The release may be a part of continuous deployment (automated last test) or continuous delivery (manual last test) if a CI/CD/CD pipeline (like Jenkins) is being used.
13. A sprint retrospective meeting involves the 3 amigos to review the sprint and if there was a requirement which was not completed in this sprint, it is taken to the next.





# Gherkin

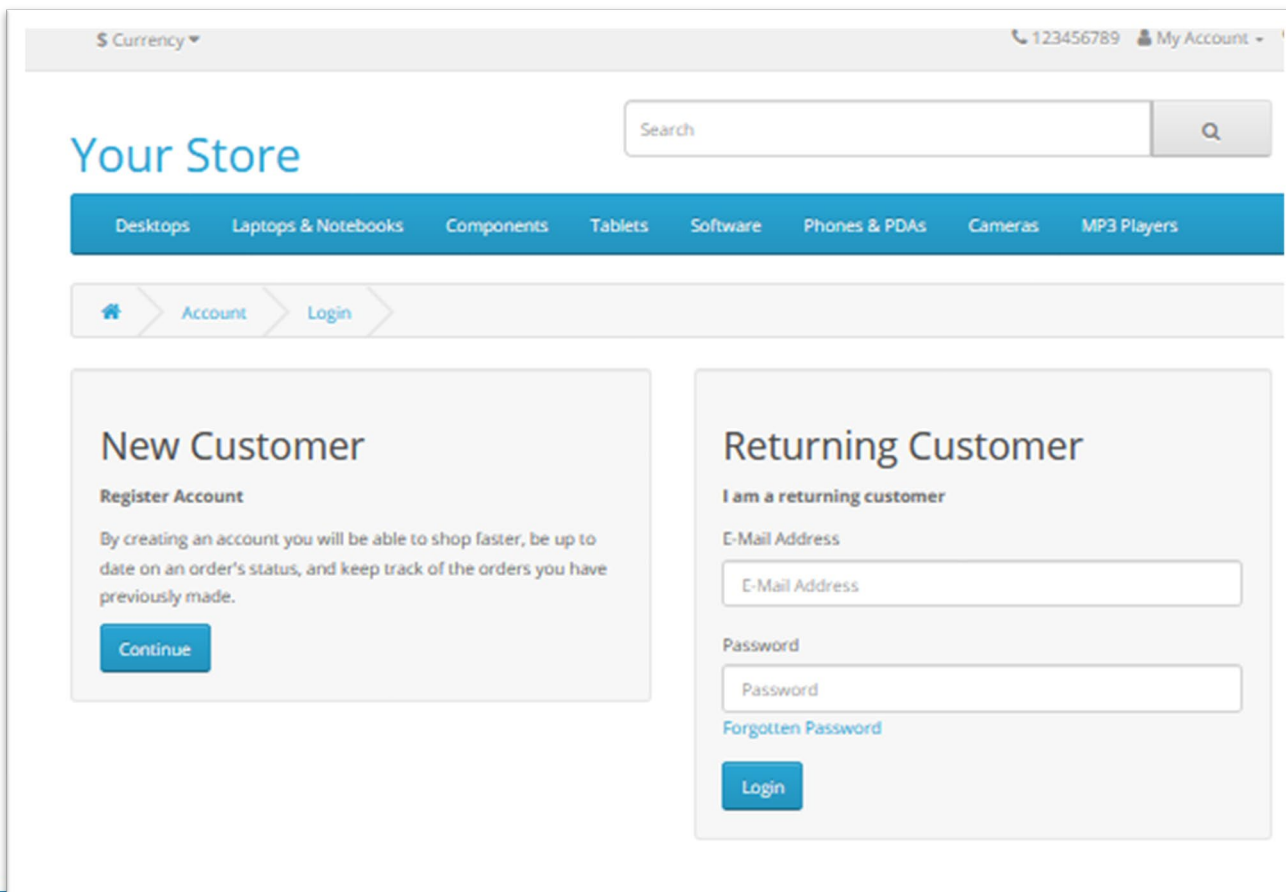
- **Gherkin** is a domain specific language that captures the conversations in a human readable format.
- We can define our acceptance criteria for a user story (the most atomic form of requirement) is Gherkin format.
- **BDD tools like Cucumber** are frameworks that parse Gherkin syntax to generate automation tests.



# Example-Writing test scenarios in BDD

**Feature:** Login

**User Story:** **As a** returning customer, **I want** to login the application, **So that** I can go to the home page



The screenshot shows a web application interface for a store. At the top, there's a header with 'Currency' and a phone number '123456789'. Below the header is a search bar. A navigation bar lists categories: Desktops, Laptops & Notebooks, Components, Tablets, Software, Phones & PDAs, Cameras, and MP3 Players. A breadcrumb trail shows 'Home' > 'Account' > 'Login'. The main content area has two columns. The left column is for 'New Customer' with a 'Register Account' link and a 'Continue' button. The right column is for 'Returning Customer' with a 'Login' button and a 'Forgotten Password' link.

Currency 123456789 My Account

Search

## Your Store

Desktops Laptops & Notebooks Components Tablets Software Phones & PDAs Cameras MP3 Players

Home Account Login

### New Customer

Register Account

By creating an account you will be able to shop faster, be up to date on an order's status, and keep track of the orders you have previously made.

Continue

### Returning Customer

I am a returning customer

E-Mail Address

E-Mail Address

Password

Password

[Forgotten Password](#)

Login



University  
of Victoria

# Gherkin Test Scenario from user story

- **Feature:** Login Functionality
- **User Story:**

**As a** returning customer  
**I want** to login the application  
**So that** I can go to the home page

- **Given** : prerequisite
- **When**: action
  - **And**: more action
- **Then**: expected output

**Scenario – also called 3 A's of testing: Arrange, Act, Assert**



# Gherkin Test Scenario from user story

- **Feature**: Login Functionality

- **User Story**:

**As a** returning customer

**I want** to login the application

**So that** I can go to the home page

- **Scenario**: Logging with **valid** credentials

**Given** user is on Login screen

**When** user enters valid username and password

**And** clicks on Login button

**Then** user is navigated to the home page.



# Gherkin Test Scenario from user story

- **Feature**: Login Functionality

- **User Story**:

**As a** returning customer

**I want** to login the application

**So that** I can go to the home page

- **Scenario**: Logging with **invalid** credentials

**Given** user is on Login screen

**When** user enters invalid username and password

**And** clicks on Login button

**Then** user is not navigated to the home page.



University  
of Victoria

# Advantages of BDD

- Easy to understand by all business stakeholders. Everybody can read and understand feature files and the resulting test cases.
- The given, when, then flow is very detailed and unambiguous.
- Easy to automate tests.
- Tests written in Cucumber directly interact with the development code.



# Here is a sample Gherkin document for withdrawal of money from account

**Feature:** Account Holder withdraws cash (Account has sufficient funds)

**User story:**

**As an** account holder

**I want** to request money

**So that** I am able to withdraw money

**Scenario1: The card is valid, and the machine contains enough money**

**Given** the account balance is \$100

**And** the card is valid

**And** the machine contains enough money

**When** the Account Holder requests \$20

**Then** the ATM should dispense \$20

**And** the account balance should be \$80

**And** the card should be returned



## Other test scenarios

**Scenario 2:** The card is valid, and the machine does not contain enough money

**Scenario 3:** The card is invalid, and the machine contains enough money.

**Scenario 4:** The card is valid, and the machine does not contain enough money





## Scenario: Signup process in Facebook

- **Given** Fred is signing up for Facebook,  
**When** he enters the required details,  
**And** submits his request,  
**Then** a Facebook account is created,  
**And** account name is set as Fred's email address,  
**And** a confirmation email is sent to Fred.



# Write test scenarios for these user stories using given, when, then format

1. As a customer of Air Canada I should be able to buy a flight ticket so that I should be able to fly to my destination.
2. As a customer of Air Canada I should be able to cancel my flight ticket so that I should be able to able to get my refund.
3. As a user of onedrive.com I should be able to upload my documents so that my team members may be able to see them.



# JPacman game

- The JPacman game is played on a rectangular board. A square on the board can be empty, or can contain the Pacman itself, one of the several ghosts, a pellet (worth 10 points), or a wall. Moveable characters such as the Pacman and the ghosts can make single-step horizontal or vertical moves. Tunnels on the border make it possible to move from one border to the opposite border. When the Pacman moves over a square containing a pellet, the player earns points, and the pellet disappears. If a player and a ghost meet at the same square, the the game is over. The player wins the game once he or she has eaten all pellets.

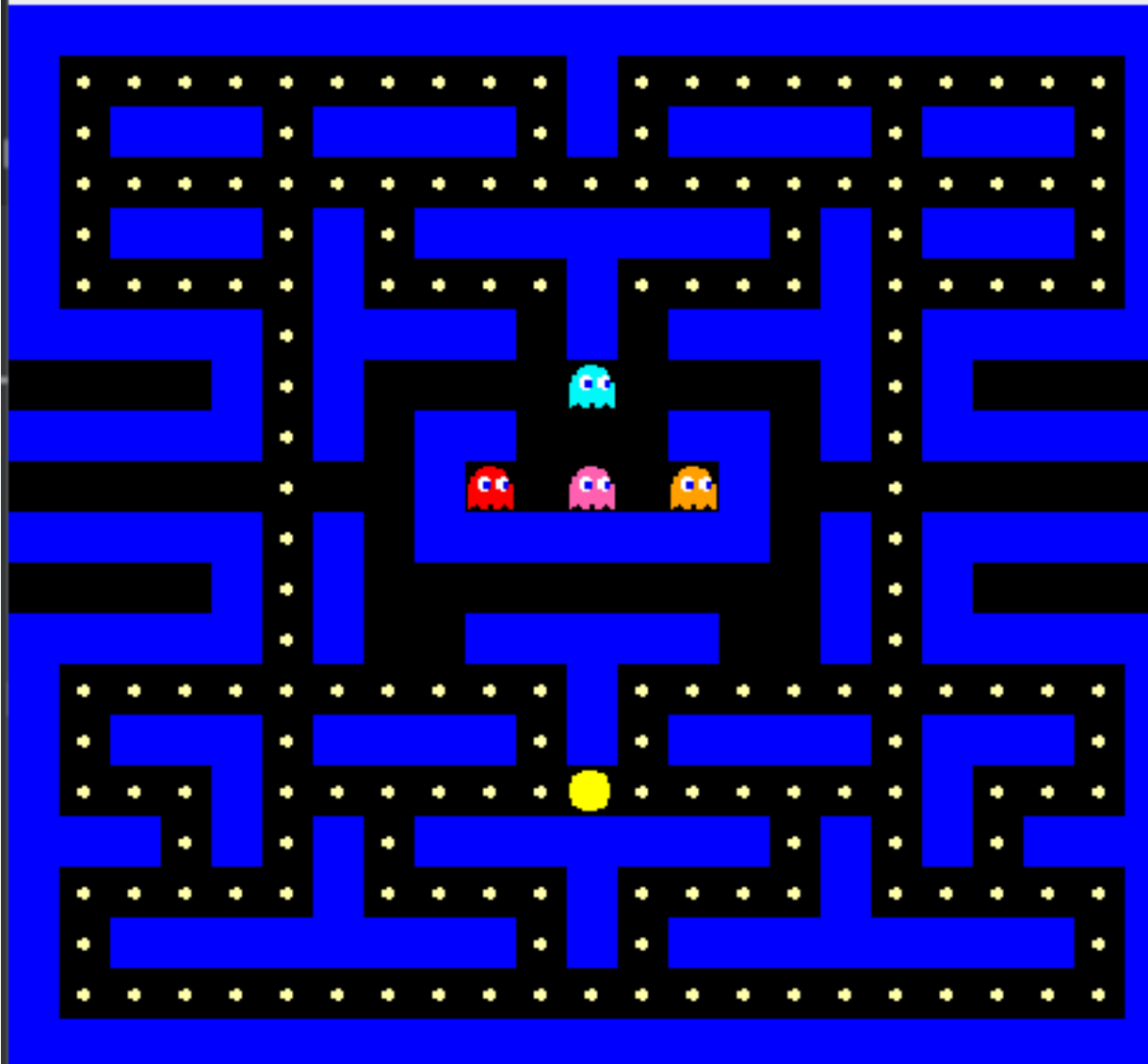




JPacman



Player 1  
Score: 0



Start

Stop



University  
of Victoria

# Play the game and write stories and scenarios

- Start the game
- Move the JPacman
- Consume the pellet
- Think about 3 more scenarios.....



# Story 1 and Scenario1 : Startup

As a player

I want to start the game  
so that I can actually play

Given the user has launched the JPacman GUI;  
When the user presses the "Start" button;  
Then the game should start.



University  
of Victoria

## Story 2: Move the Player

- As a player,
- I want to move my Pacman around on the board;
- So that I can earn all points and win the game.



## Scenario S2.1: The player consumes the pellet

- Given the game has started, and my Pacman is next to a square containing a pellet;
- When I press an arrow key towards that square;
- Then my Pacman can move to that square, and I earn the points for the pellet, and the pellet disappears from that square.





## Scenario S2.2: The player moves on empty square

- Given the game has started, and my Pacman is next to an empty square;
- When I press an arrow key towards that square;
- Then my Pacman can move to that square and my points remain the same.



# Tools for BDD-creating and automating BDD user stories

- Cucumber
- JBehave
- Behat

