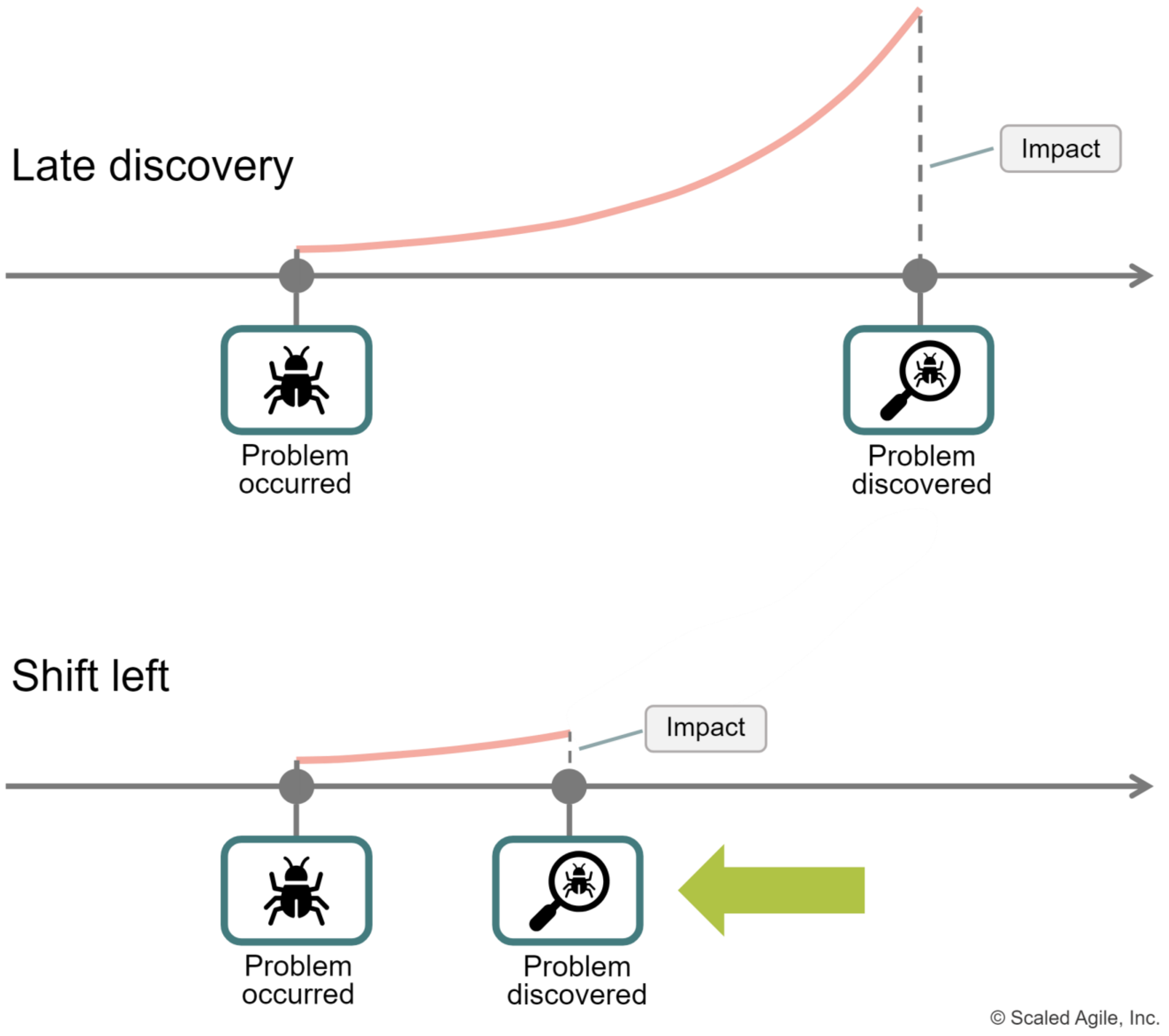
Programų kūrimo procesas

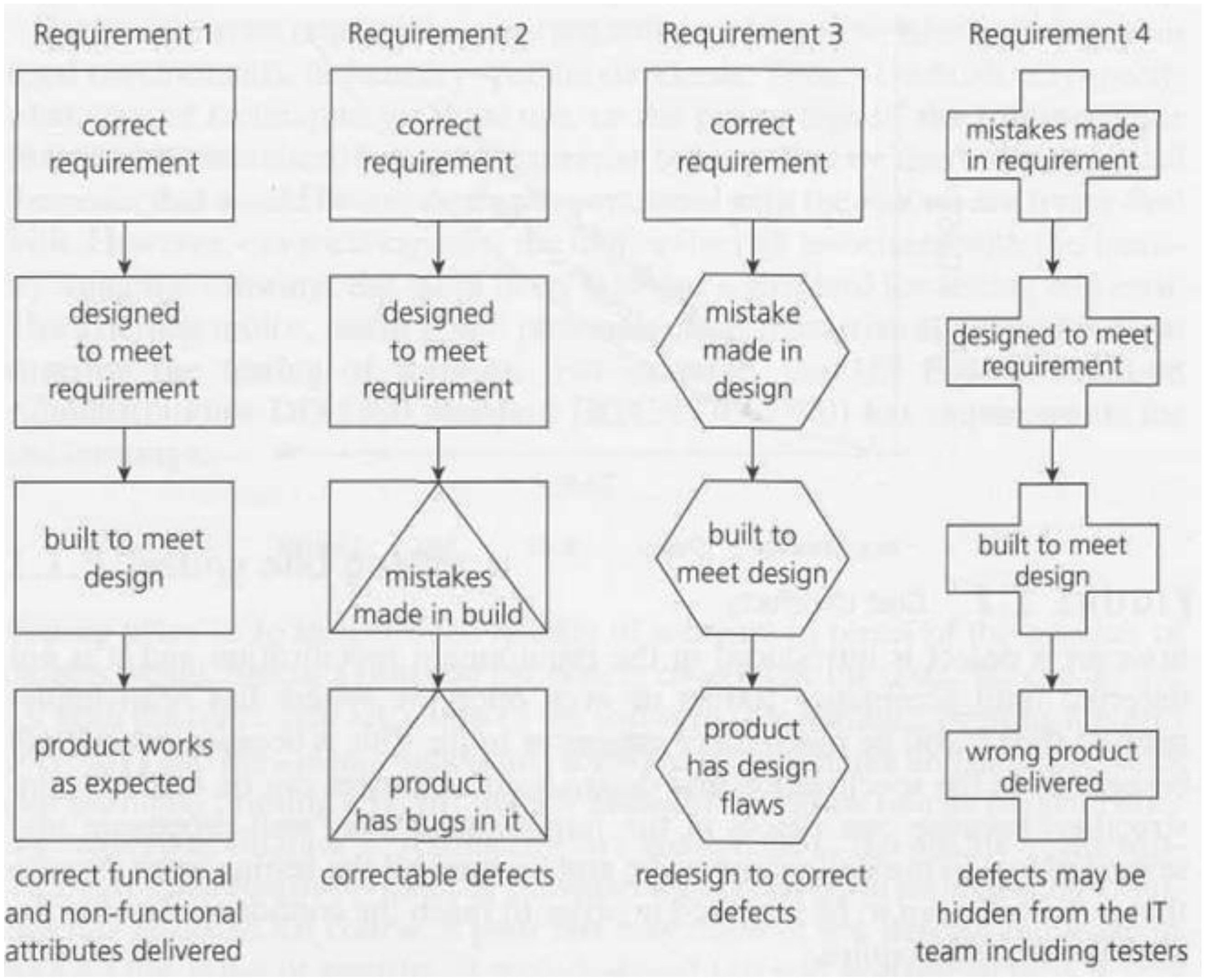
From Requirements to Test Cases

Dr. Asta Slotkiene

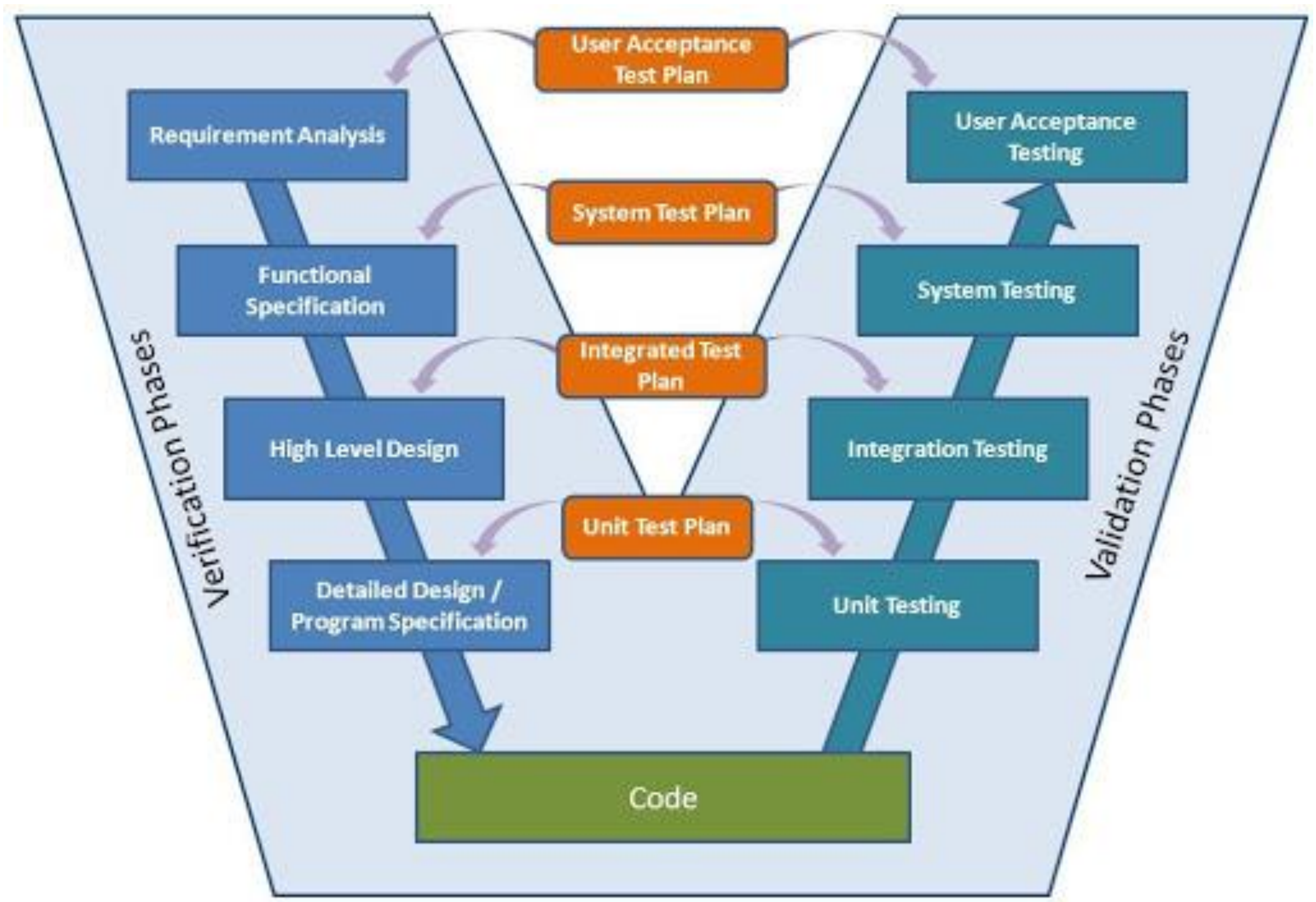
Agile Quality Practices



Types of error and defect

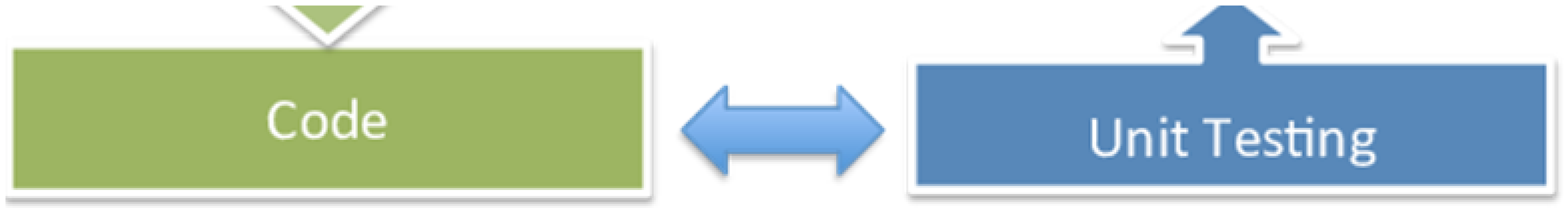


V model


User Acceptance Test Plan
Requirement Analysis
User Acceptance Testing
System Test Plan
Functional Specification
System Testing
Integrated Test Plan
High Level Design
Integration Testing
Unit Test Plan
Detailed Design / Program Specification
Unit Testing
Code
Verification Phases
Validation Phases

Testing Level: Unit Testing

- Who: Developers
- How:
  - White-Box Testing Method
  - UT frameworks (e.g., jUnit), drivers, stubs, and mock/fake objects are used



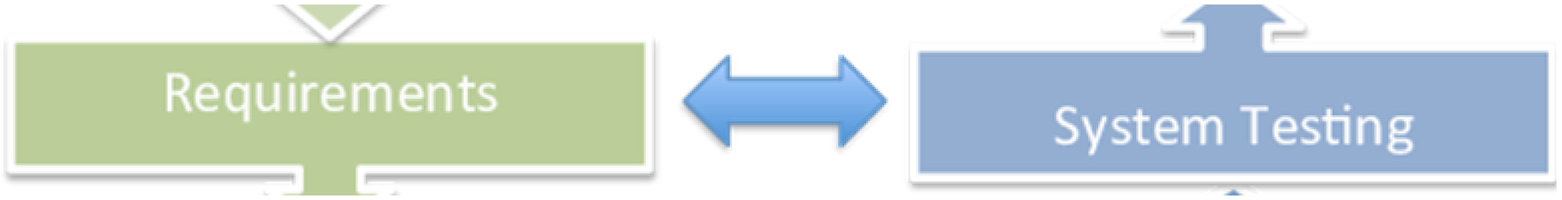
Testing Level: Integration Testing

- Who: Either Developers themselves or independent Testers
- How:
  - Any of Black Box, White Box, and Gray Box Testing methods can be used
  - Test drivers and test stubs are used to assist in Integration Testing.

Design
Integration Testing

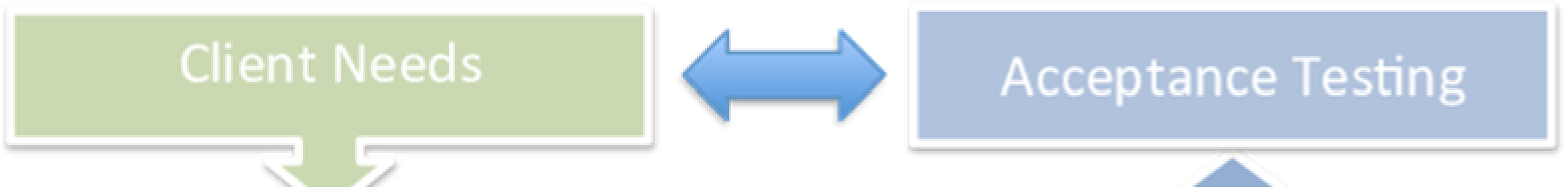
Testing Level: System Testing

- Who:
  - Normally, independent Testers perform System Testing
- How:
  - Usually, Black Box Testing method is used.



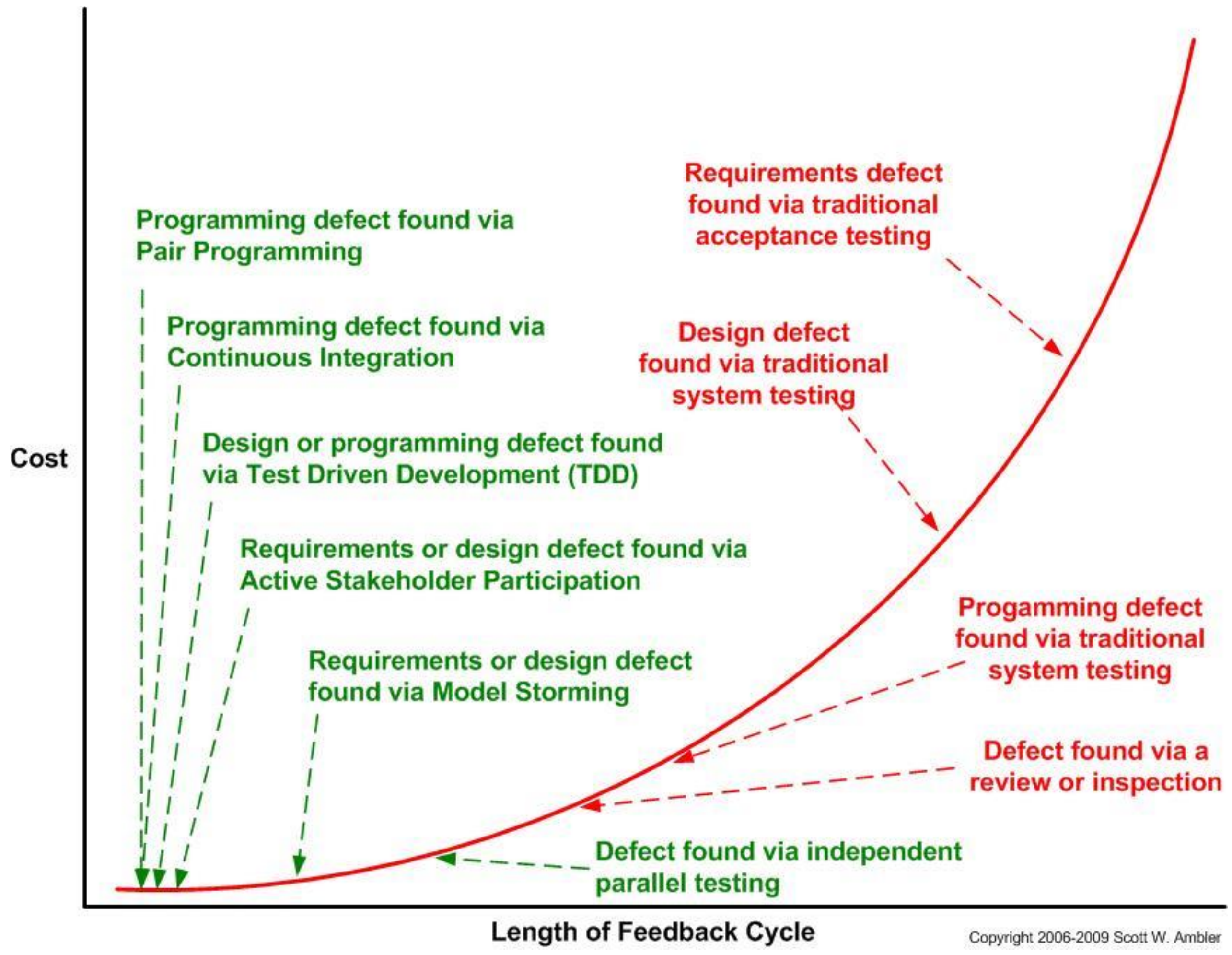
Testing Level: Acceptance Testing

- Who:
  - Product Management, Sales, Customer Support, Customers
- How:
  - Usually, Black Box Testing method is used; often the testing is done ad-hoc and non-scripted



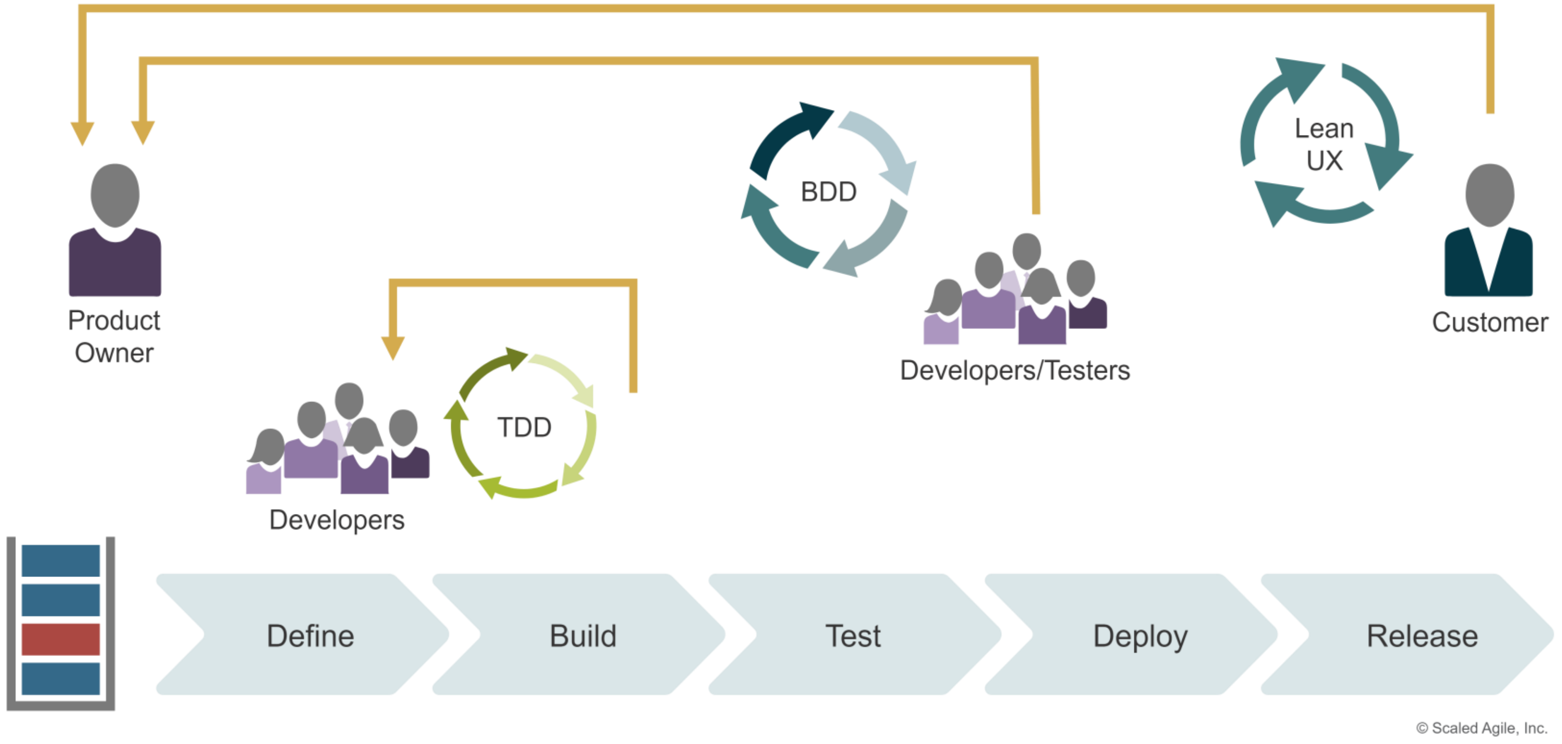
also called: Behavior-driven testing (BDD)

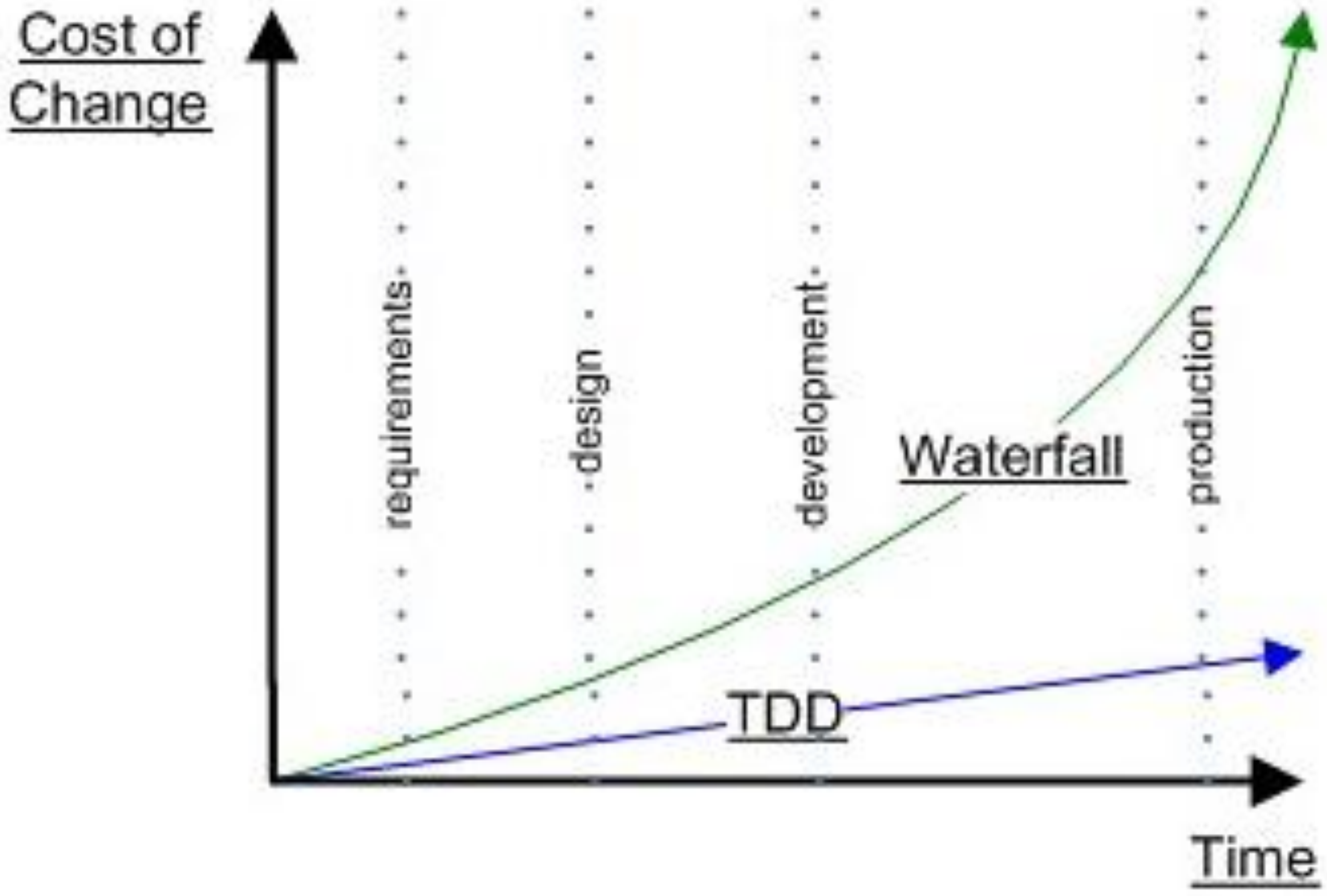
Comparing the feedback cycle of various development techniques



http://www.agilemodeling.com/essays/costOfChange.htm

Agile Quality Practices



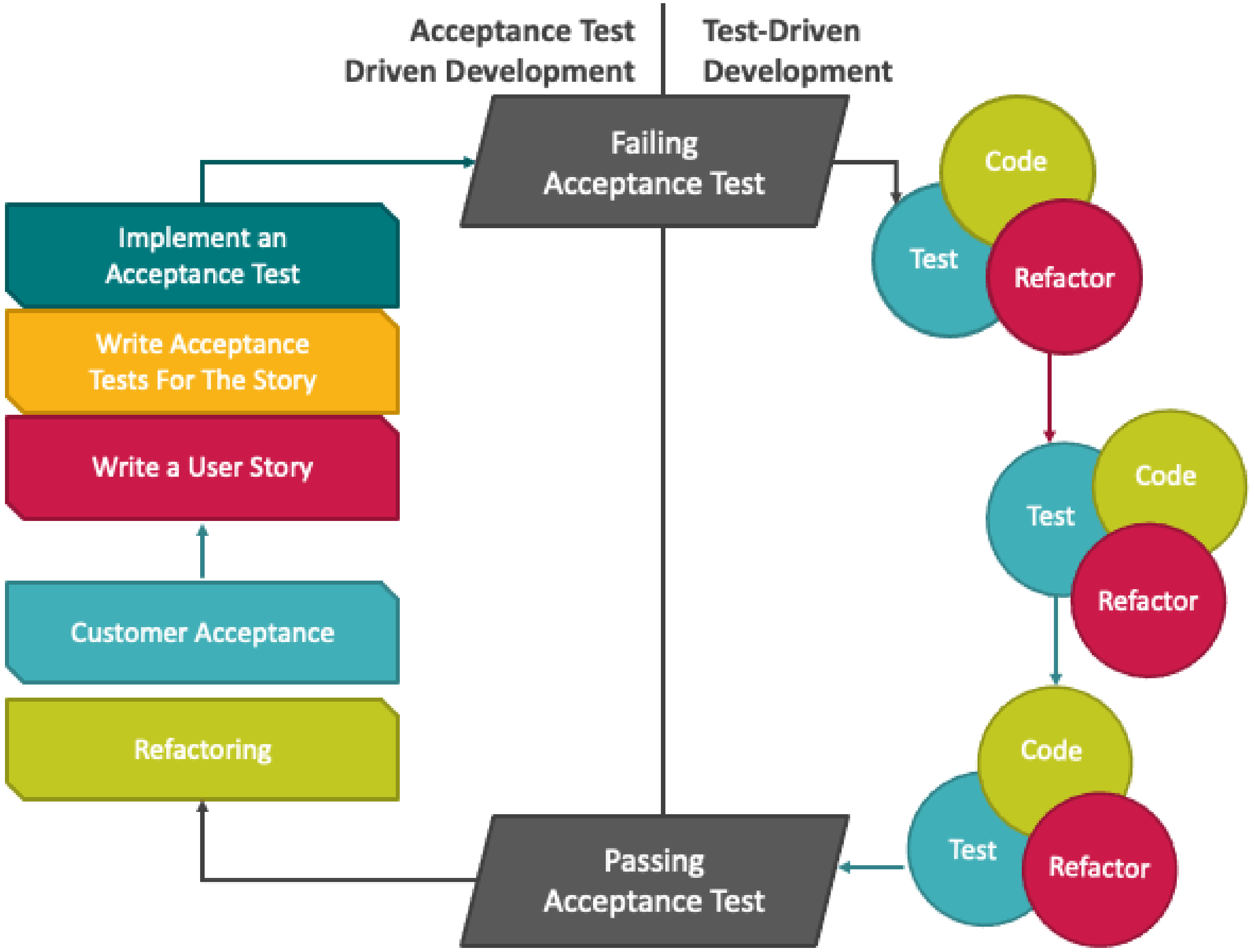

Cost of Change
requirements
design
development
production
Waterfall
TDD
Time

Agile Testing Methodology

- Test Driven Development (TDD)
- Acceptance Test Driven Development
- Behavior Driven Development (BDD)

http://tryqa.com/what-are-the-different-agile-testing-methodology-test-driven-development-behavior-driven-development/#more-1708

ATDD


Acceptance Test Driven Development
Test-Driven Development
Failing Acceptance Test
Implement an Acceptance Test
Write Acceptance Tests For The Story
Write a User Story
Customer Acceptance
Refactoring
Code
Test
Refactor
Code
Test
Refactor
Code
Test
Refactor
Passing Acceptance Test

Test-driven development (TDD)

- A software development technique in which the test cases are developed, and often automated, and then **the software is developed incrementally to pass those test cases.**

- ISTQB Glossary
  - ***https://glossary.istqb.org/search/***

Acceptance Test-driven development (ATDD)

- A collaborative approach to development in which **the team and customers are using the customers own domain language to understand their requirements**, which forms the basis for testing a component or system.

- ISTQB Glossary
  - ***https://glossary.istqb.org/search/***

Behavior driven development (BDD)

- A collaborative approach to development in **which the team is focusing on delivering expected behavior of a component or system for the customer,** which forms the basis for testing.

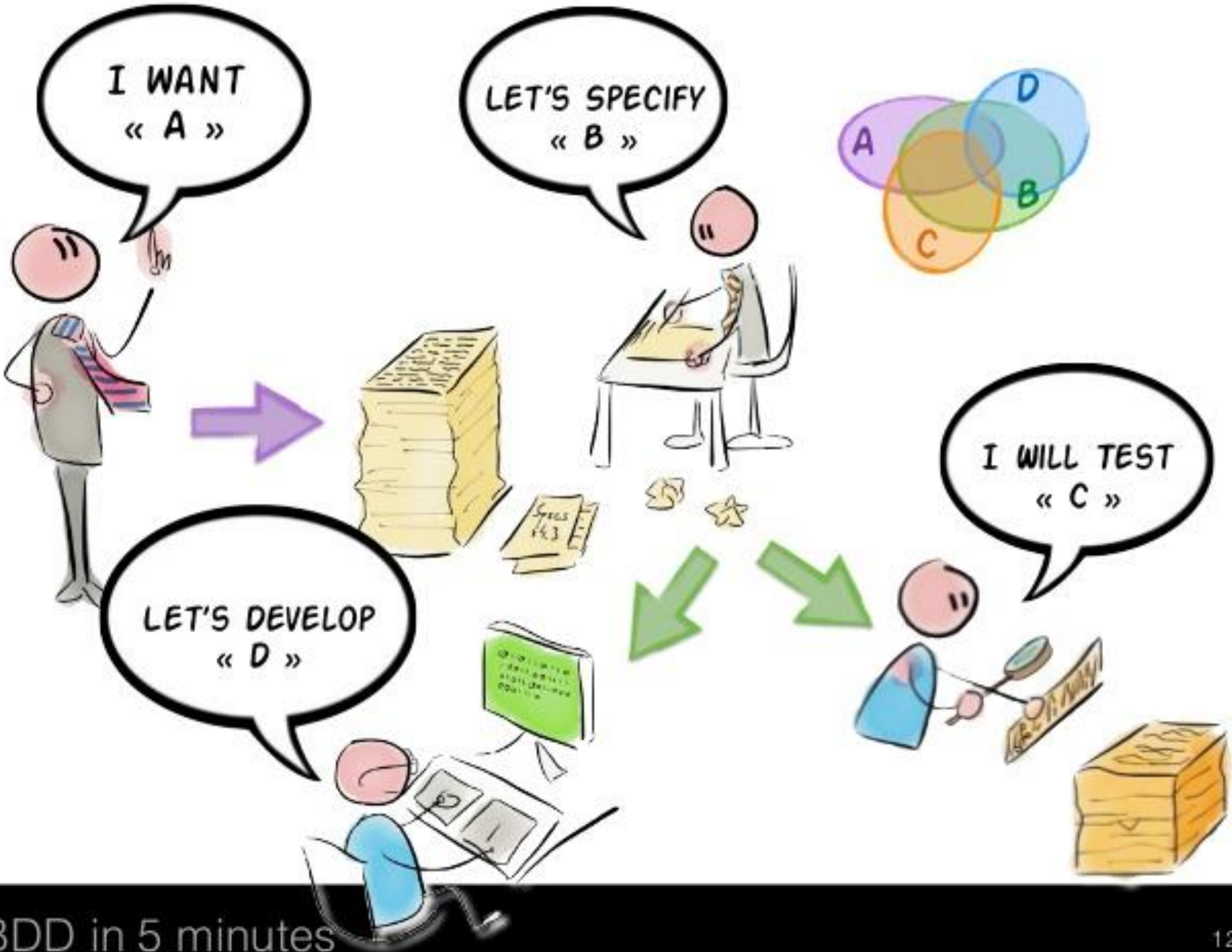
- ISTQB Glossary
  - ***https://glossary.istqb.org/search/***

Why is a focus on BEHAVIOUR so important?

Users usually don't care about **technical implementation**

they care about **BEHAVIOUR** of the software

*"...our clients **don't value the code as such;***
*they **value the things that the code does for them**."*
Michael Bolton


I WANT « A »
LET'S SPECIFY « B »
A
D
B
C
Specs v1.3
I WILL TEST « C »
LET'S DEVELOP « D »

Who should write acceptance tests?

- Answer: client???
  - Probably manual tests: OHHH

Step	Req	Pass Conditions	Pass?
1. Select the Biology program.	UIR-2	System displays biology classes w/ first class BIOLOGY 1150, Section 01, Title GENERAL BIOLOGY, Instructor Block, Anna, Filled/Seats 52/53, Class# 1311, Credits 5, Meets BOE 0221 MWF 8:00-8:52	P / F
2. Double-click on Class# 1330	UIR-1	System includes Class# 1330 in schedule at bottom	P / F
3. Scroll down to Class# 1331 (BIOLOGY 1650, Section 01)	UIR-9	System displays Class# 1331 with a pink background	P / F
4.	UIR-9	All sections listed between #1311 and #1331 have a white background	P / F
5. Select the GENENG program.	UIR-2	System displays general engineering courses	P / F

Who should write acceptance tests?

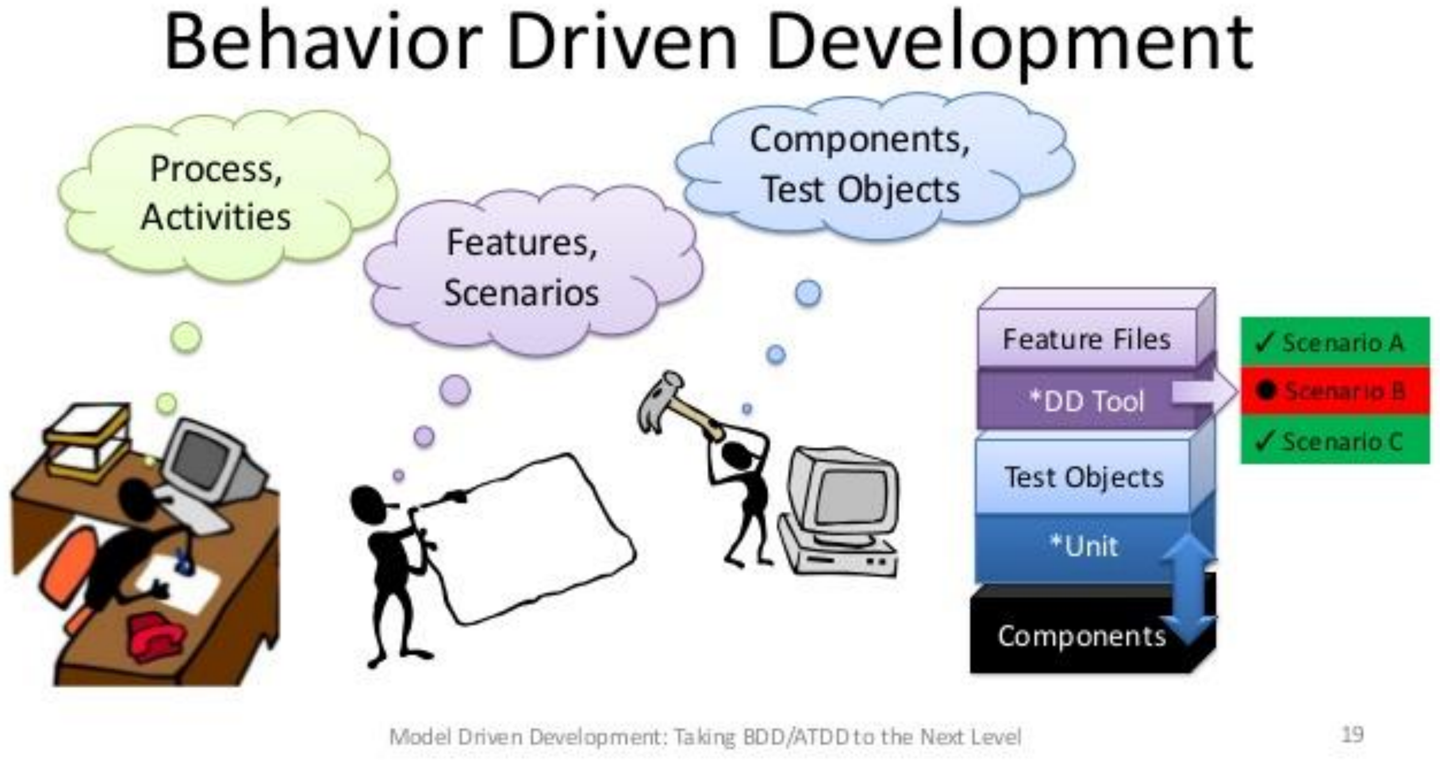
l client write test cases, Not much help to development team

Step	Req	Pass Conditions	Pass?
1. Select the Biology program.	UIR-2	System displays biology classes w/ first class BIOLOGY 1150, Section 01, Title GENERAL BIOLOGY, Instructor Block, Anna, Filled/Seats 52/53, Class# 1311, Credits 5, Meets BOE 0221 MWF 8:00-8:52	P / F
2. Double-click on Class# 1330	UIR-1	System includes Class# 1330 in schedule at bottom	P / F
3. Scroll down to Class# 1331 (BIOLOGY 1650, Section 01)	UIR-9	System displays Class# 1331 with a pink background	P / F
4.	UIR-9	All sections listed between #1311 and #1331 have a white background	P / F
5. Select the GENENG program.	UIR-2	System displays general engineering courses	P / F

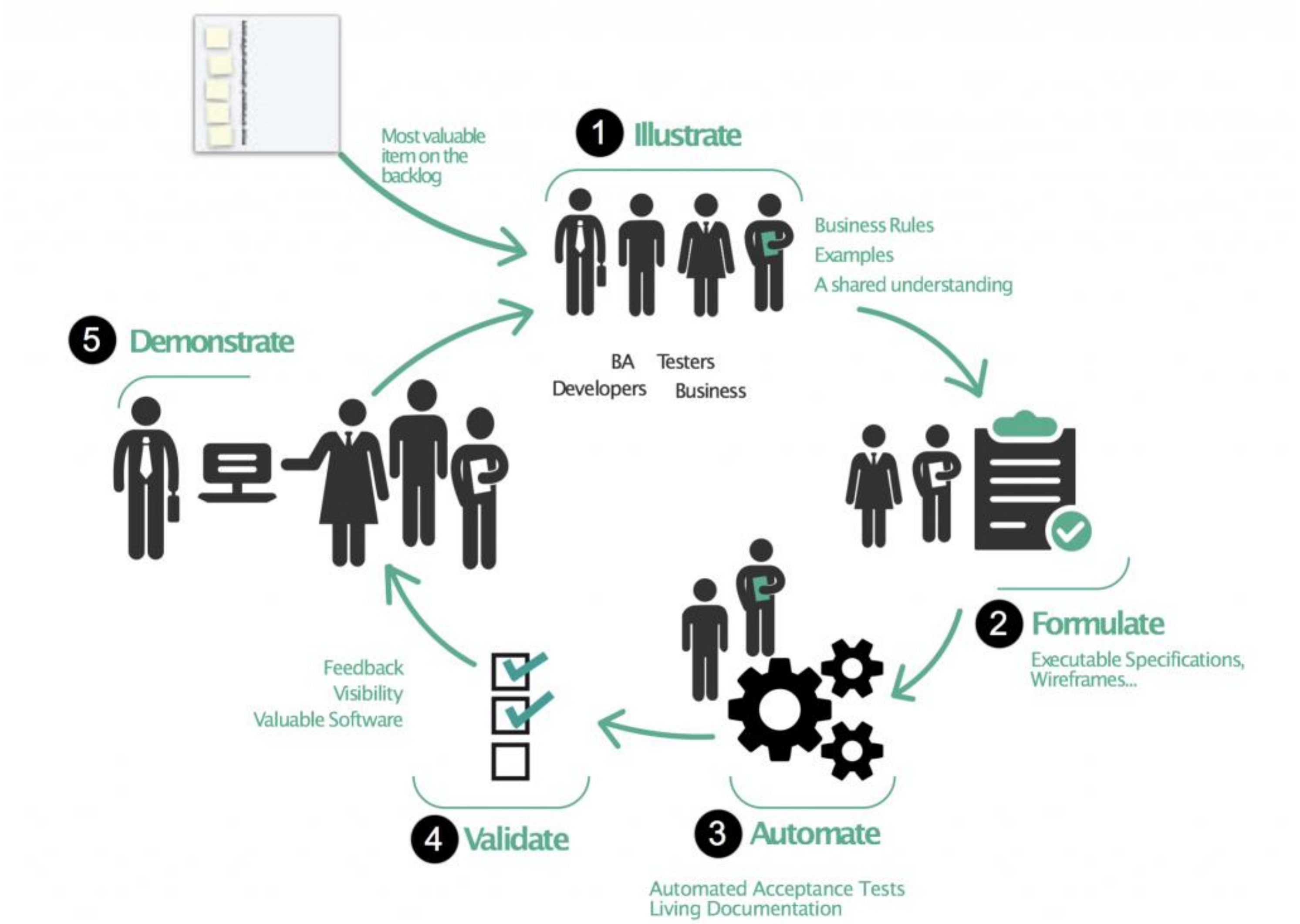
- Lots of detail
- Must document passes manually
- Maintenance issue when steps depend on each other

Behavior based development (BDD)

- **Software development methodology based on TDD**



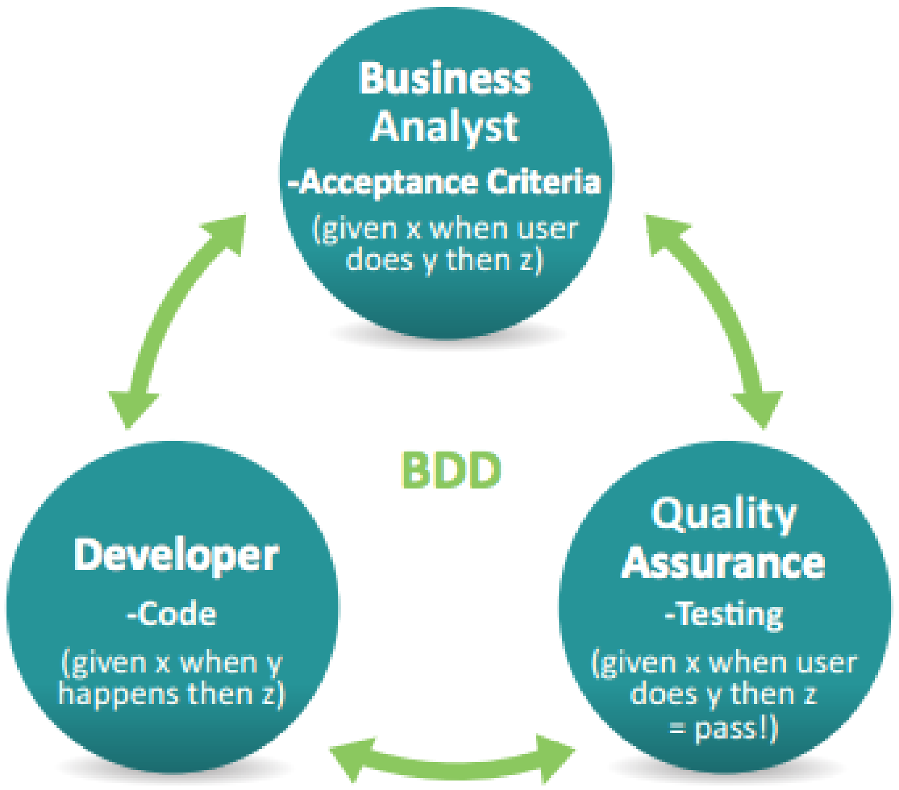
Behaviour Driven Development



https://johnfergusonsmart.com/behaviour-driven-development-3-minute-rundown/

Behavior based development (BDD)

- Simplify writing test cases is to use **behavior-driven development (BDD)**, which is an **extension of test-driven development** that encourages collaboration between:
  - developers,
  - QA testers
  - non-technical
  - business participants


Business Analyst
-Acceptance Criteria
(given x when user does y then z)
BDD
Developer
-Code
(given x when y happens then z)
Quality Assurance
-Testing
(given x when user does y then z = pass!)

Behavior Driven Development

- Basic model: test-driven development
- Developers code to tests written by POs, clients
  - Alternatively, clients review tests written in Gherkin by developers
- If only programmers reviewing or writing tests, Gherkin is probably not useful

Behavior Driven Development: Big Idea

- **Tests from customer-friendly user stories**
  - Acceptance: **ensure satisfied customer**
  - Integration: ensure interfaces between modules consistent assumptions, communicate correctly
- **Meet halfway between customer and developer**
  - User stories are not code, so clear to customer and can be used to reach agreement
  - Also not completely freeform, so can connect to real tests

Behavior based development (BDD)

1. Business analyst **writes a user story**
2. (Acceptance) tester writes **scenarios based on user story**
3. Business team **reviews scenarios**
4. Test engineer writes the step definitions for the scenario steps
5. QA team writes test scripts (to automate the scenarios)
6. The test scripts are run, issues analysed and bugs fixed
7. The test scripts are run as regression tests
8. End user accepts the software if tests pass (acceptance criteria met)

Focus on the requirements

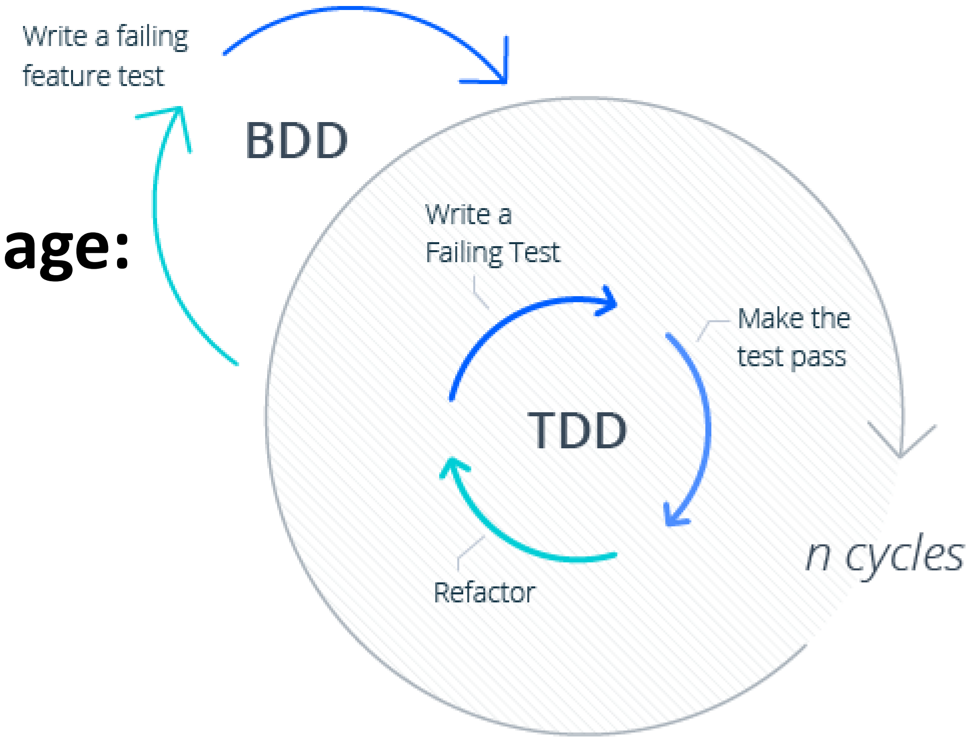
Starting by the test means starting by the requirements!!!

https://scaledagileframework.com/behavior-driven-development/

Behavior based development (BDD)

- Behavior-driven development should be focused on the business behaviors your code is implementing: **the “why” behind the code**

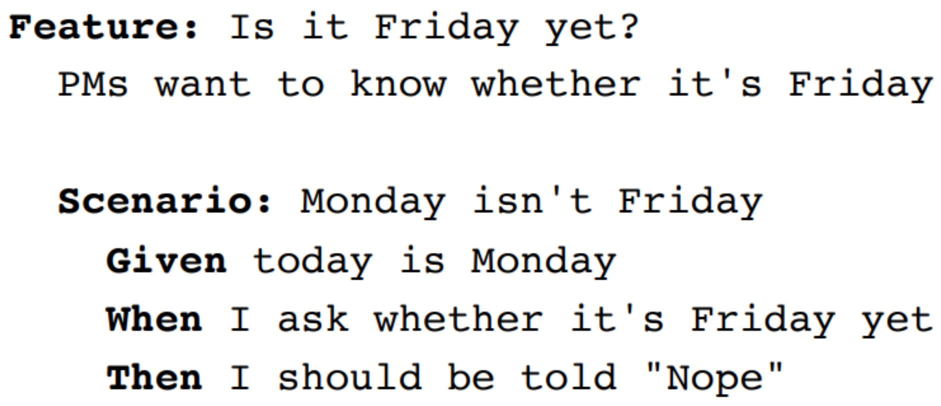
Scenario definition language: Gherkin (DSL)



https://cucumber.io/docs/gherkin/

Requirements based testing: Gherkin

- A Domain Specific Language (DSL) that **helps non-programmers express requirements (features)** in a structured manner
- **Requirements-based testing involves examining each requirement and developing a test or tests for it.**



```
Feature: Is it Friday yet?
  PMs want to know whether it's Friday

  Scenario: Monday isn't Friday
    Given today is Monday
    When I ask whether it's Friday yet
    Then I should be told "Nope"
```

Requirements based testing: Gherkin

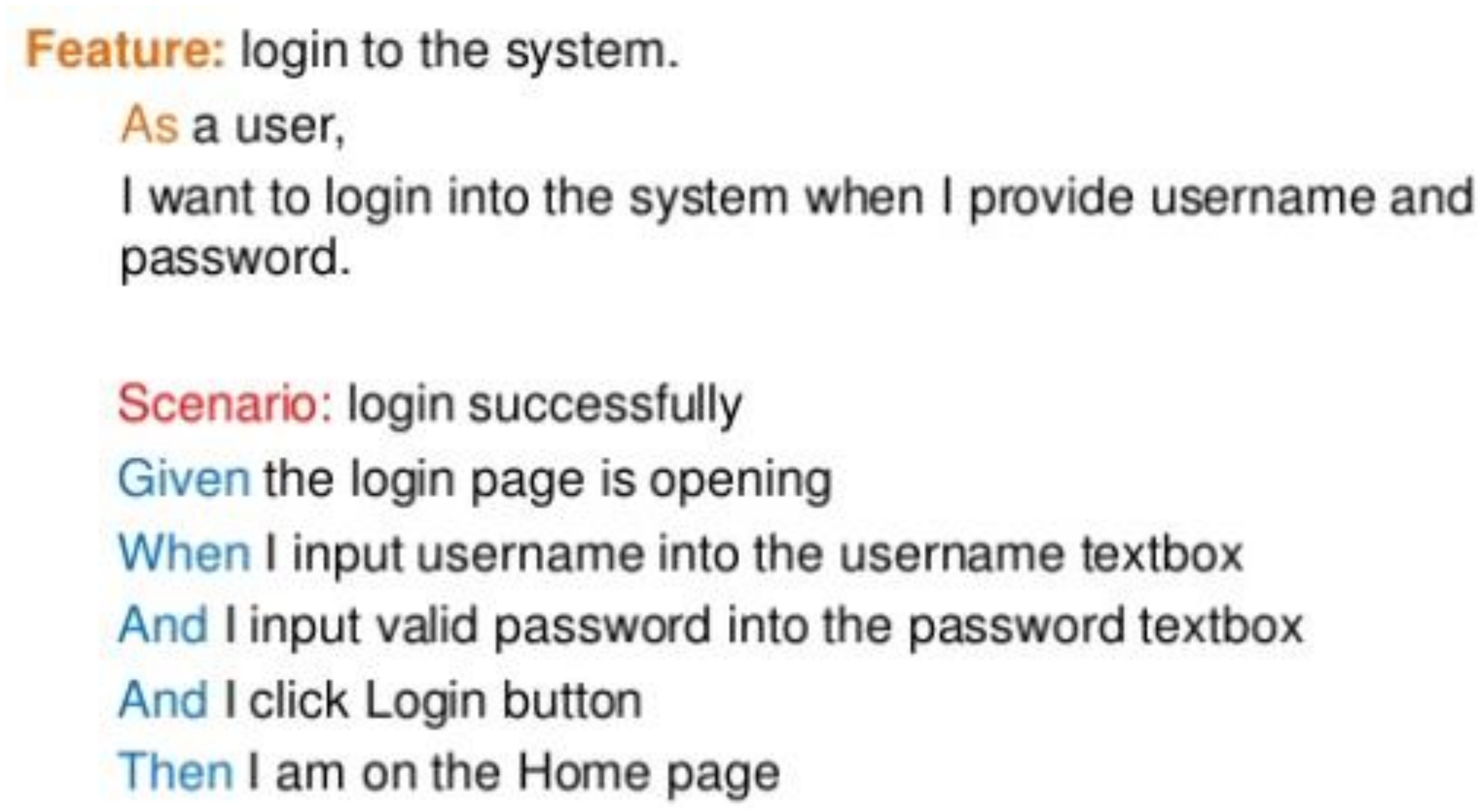
1. The first line of this file starts with the keyword **Feature**: followed by a name
   - Features will be saved in ***.feature files** in Cucumber.
2. The fourth line - **Scenario**
3. The last three lines starting with **Given, When and Then** are the steps of our scenario. This is what Cucumber will execute.

```
Feature:

Scenario:
  Given
  When
  Then
```

Cucumber Feature = Test Scenario

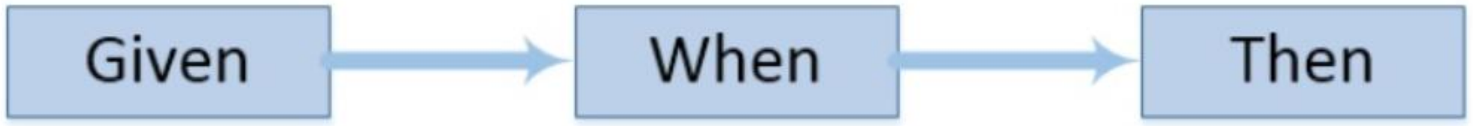
Cucumber Scenario = Test Case

Requirements based testing: Gherkin



```gherkin
Feature: login to the system.
    As a user,
    I want to login into the system when I provide username and
    password.

    Scenario: login successfully
    Given the login page is opening
    When I input username into the username textbox
    And I input valid password into the password textbox
    And I click Login button
    Then I am on the Home page
```

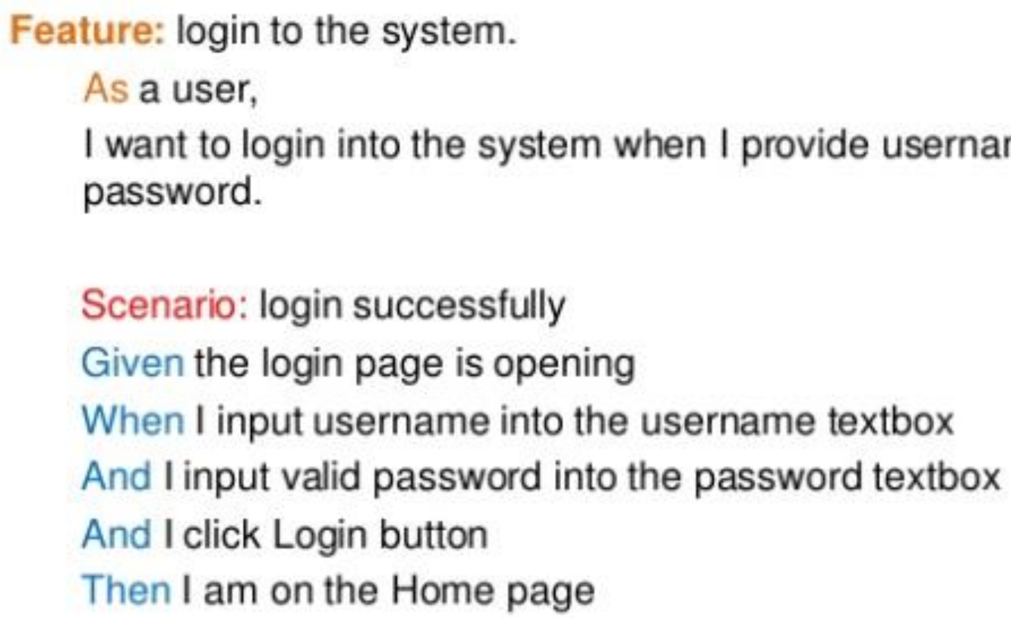
Requirements based testing: Gherkin



What software will look like to user

Things that the user will do

What the user should expect

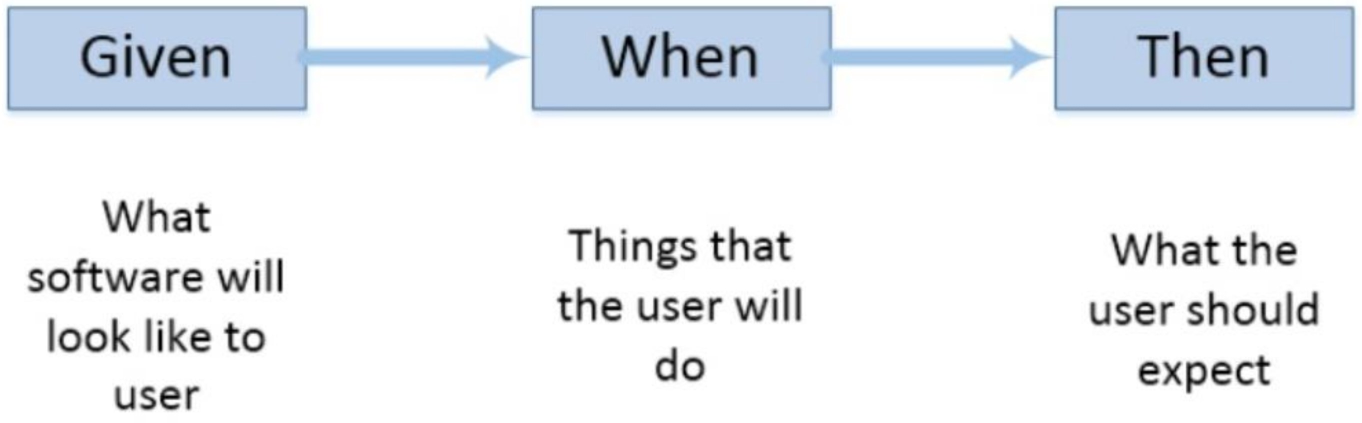


```
Feature: login to the system.
    As a user,
    I want to login into the system when I provide username and
    password.

    Scenario: login successfully
    Given the login page is opening
    When I input username into the username textbox
    And I input valid password into the password textbox
    And I click Login button
    Then I am on the Home page
```

Requirements based testing: Gherkin

- **Given**:
  - The purpose of **Given** steps is to put the system in a known **state before the user** (or external system) starts interacting with the system (in the When steps).

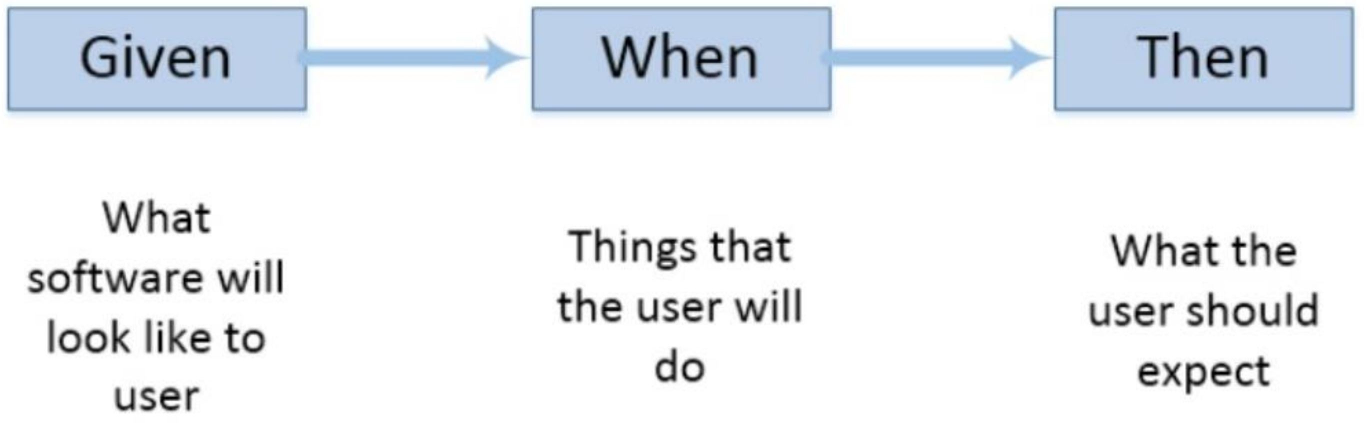
  - If you have worked with use cases, **givens are your preconditions.**


Given
When
Then
What software will look like to user
Things that the user will do
What the user should expect

https://cucumber.io/docs/guides/10-minute-tutorial/#write-a-scenario

Requirements based testing: Gherkin

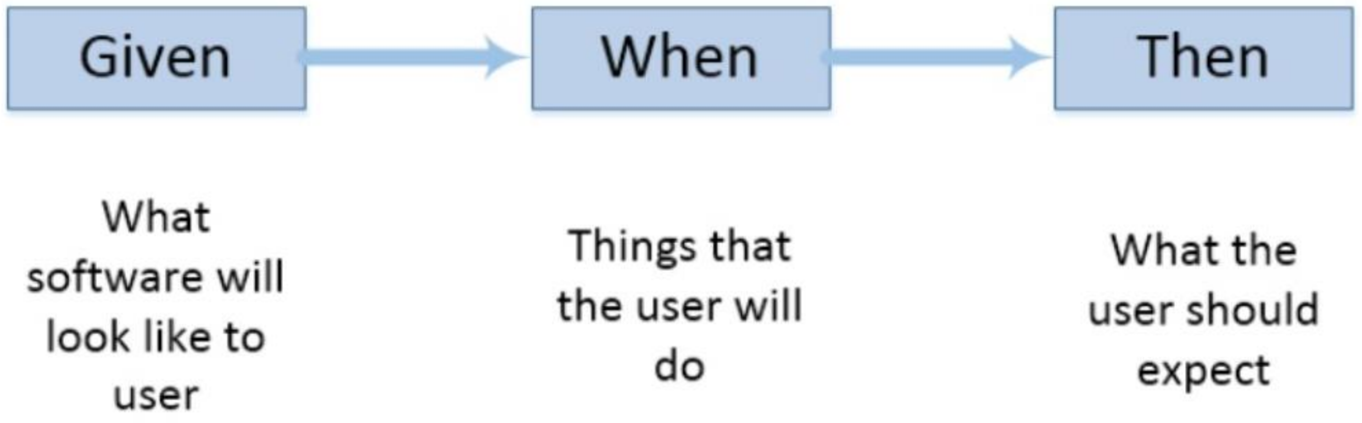
- **When**:
  - The purpose of **When** steps is to describe the key action the user performs.


Given
When
Then
What software will look like to user
Things that the user will do
What the user should expect

https://cucumber.io/docs/guides/10-minute-tutorial/#write-a-scenario

Requirements based testing: Gherkin

- **Then**:
  - The purpose of **Then** steps is to observe outcomes.
  - The observations should be related to **the business value/benefit** in your feature description.
  - Thus, it should be related to **something visible from the outside (behavior).**


Given
When
Then
What software will look like to user
Things that the user will do
What the user should expect

https://cucumber.io/docs/guides/10-minute-tutorial/#write-a-scenario

Gherkin: example

```
Feature: Multiple site support
 Only blog owners can post to a blog, except
administrators, who can post to all blogs.

  Background:
 Given a global administrator named "Greg"
 And a blog named "Greg's anti-tax rants"
 And a customer named "Dr. Bill"
 And a blog named "Expensive Therapy" owned by
     "Dr. Bill"
```

Gherkin: example

```gherkin
Scenario:
Dr. Bill posts to his own blog

Given I am logged in as Dr. Bill
When I try to post to "Expensive Therapy"
Then I should see "Your article was published.
```

Gherkin: example

```
Scenario:

Dr. Bill tries to post to somebody else's blog,
and fails

Given I am logged in as Dr. Bill
When I try to post to "Greg's anti-tax rants"
Then I should see "Hey! That's not your blog!"
```

Gherkin: example

```gherkin
Scenario:

Greg posts to a client's blog

Given I am logged in as Greg
When I try to post to "Expensive Therapy"
Then I should see "Your article was published."
```

Advantages of BBD

- **Better communication** between developers, testers and product owners.
- Being **non-technical** in nature, it can reach a wider audience
- The behavioral approach **defines acceptance criteria prior to development**.
- No defining ‘test’, but are defining ‘behavior’.

Advantages of BDD

- **Better communication** between developers, testers and product owners.
- Being **non-technical** in nature, it can reach a wider audience
- The behavioral approach **defines acceptance criteria prior to development**.
- No defining ‘test’, but are defining ‘behavior’.

Disadvantages of BDD

- To work in BDD, prior experience of TDD is required.
- BDD is incompatible with the **waterfall approach.**
- If the requirements are not properly specified, BDD may not be effective.
- **Testers using BDD need to have sufficient technical skills.**

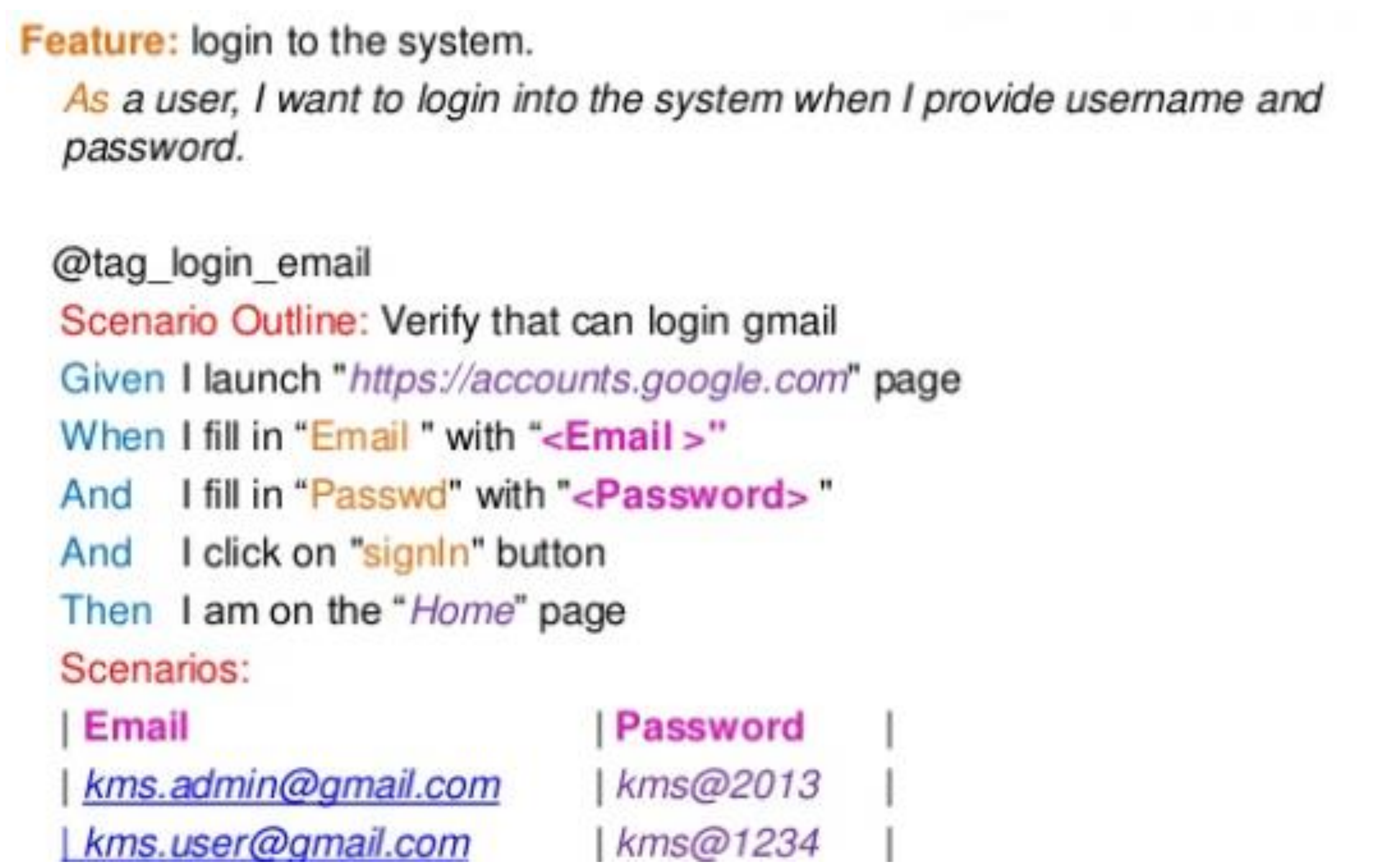
Gherkin Format and Syntax

The keywords are:

- `Feature`
- `Rule (as of Gherkin 6)`
- `Example (or Scenario)`
- `Given, When, Then, And, But for steps (or *)`
- `Background`
- `Scenario Outline (or Scenario Template)`
- `Examples (or Scenarios)`
- `""" (Doc Strings)`
- `| (Data Tables)`
- `@ (Tags)`
- `# (Comments)`

https://cucumber.io/docs/gherkin/reference/

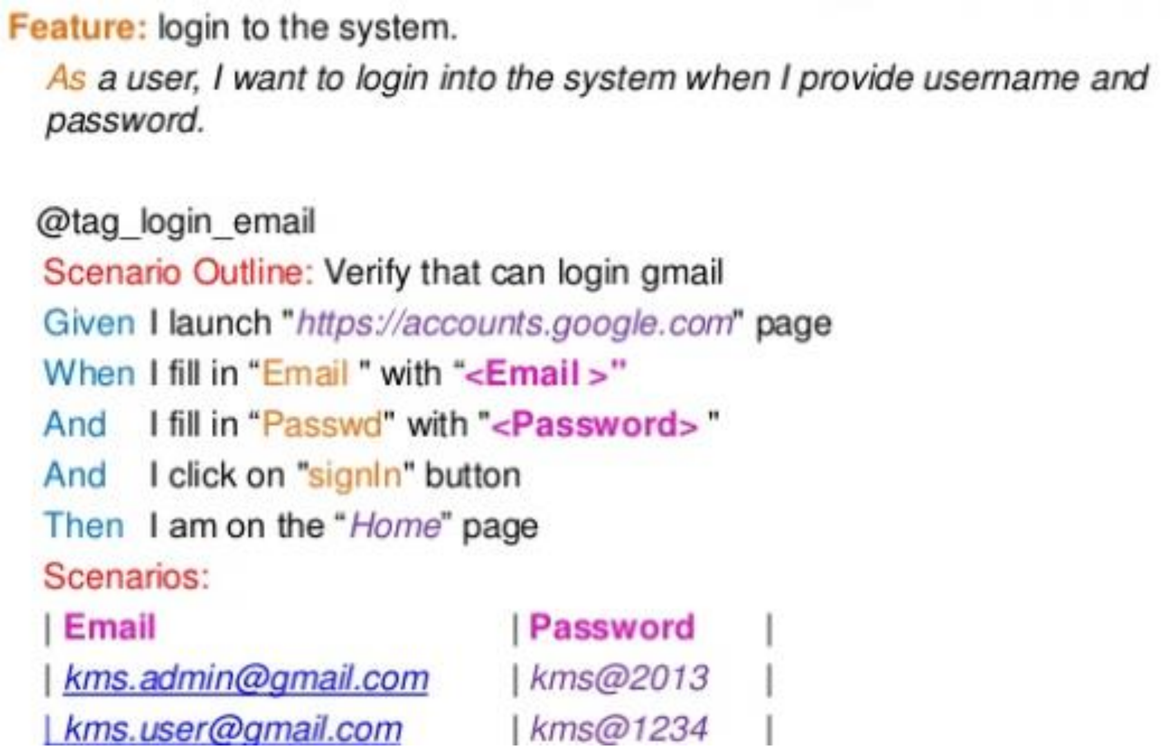
BDD example: scenario outline



```gherkin
Feature: login to the system.
  As a user, I want to login into the system when I provide username and
  password.

  @tag_login_email
  Scenario Outline: Verify that can login gmail
  Given I launch "https://accounts.google.com" page
  When I fill in "Email " with "<Email >"
  And  I fill in "Passwd" with "<Password> "
  And  I click on "signIn" button
  Then I am on the "Home" page
  Scenarios:
  | Email                    | Password   |
  | kms.admin@gmail.com      | kms@2013   |
  | kms.user@gmail.com       | kms@1234   |
```

BDD example: scenario outline

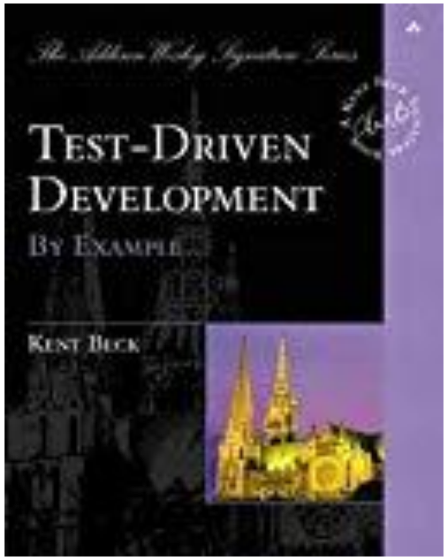
- **Tables as arguments** to steps are handy for specifying a larger data set - usually as input to a Given or as expected output from a Then.



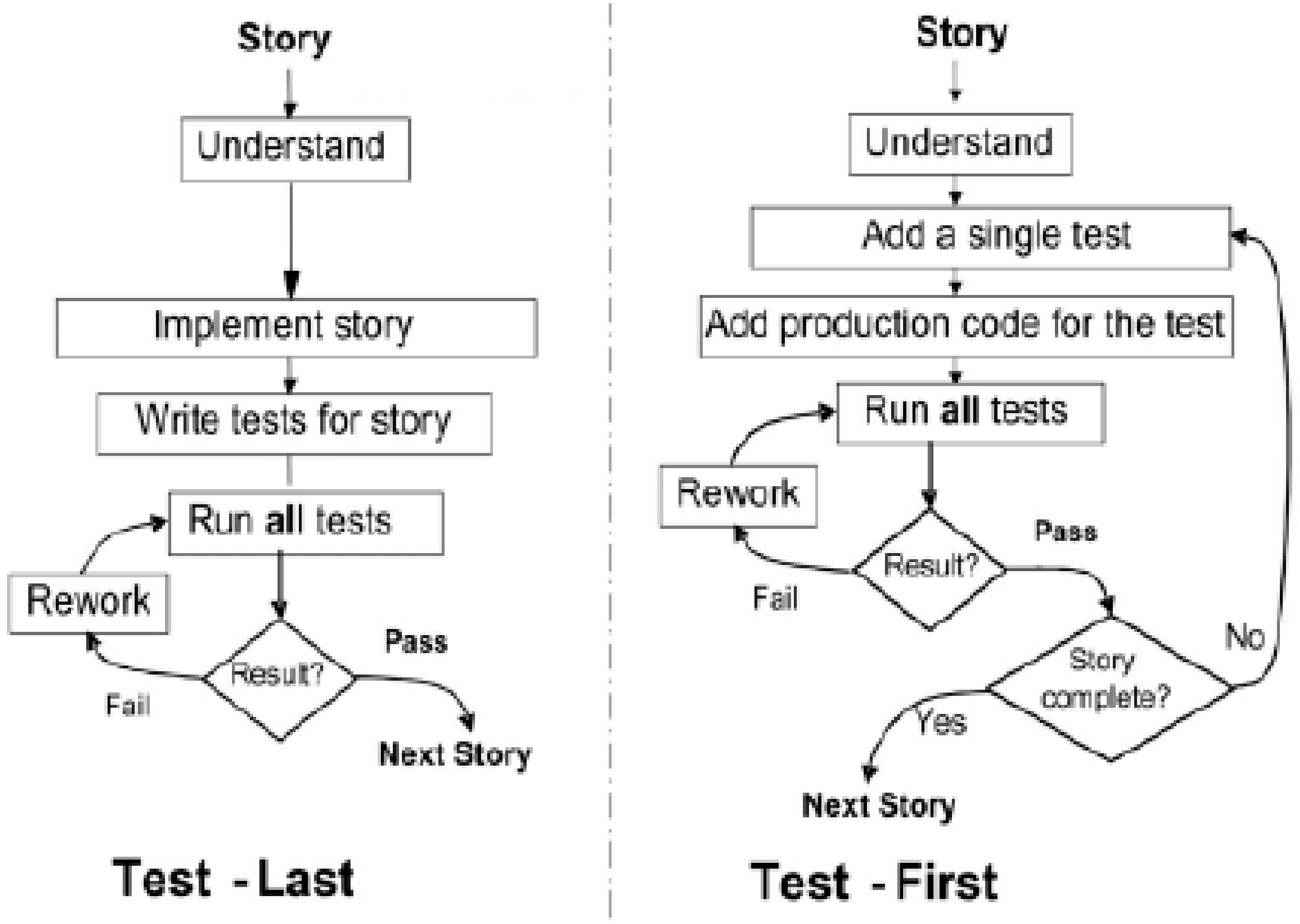
```
Feature: login to the system.
  As a user, I want to login into the system when I provide username and
  password.

 @tag_login_email
 Scenario Outline: Verify that can login gmail
 Given I launch "https://accounts.google.com" page
 When I fill in "Email " with "<Email >"
 And   I fill in "Passwd" with "<Password> "
 And   I click on "signIn" button
 Then  I am on the "Home" page
 Scenarios:
 | Email                  | Password    |
 | kms.admin@gmail.com    | kms@2013    |
 | kms.user@gmail.com     | kms@1234    |
```

Test-Driven-Development

- **Popularized by Kent Beck (2003)**
- TDD completely turns traditional development around

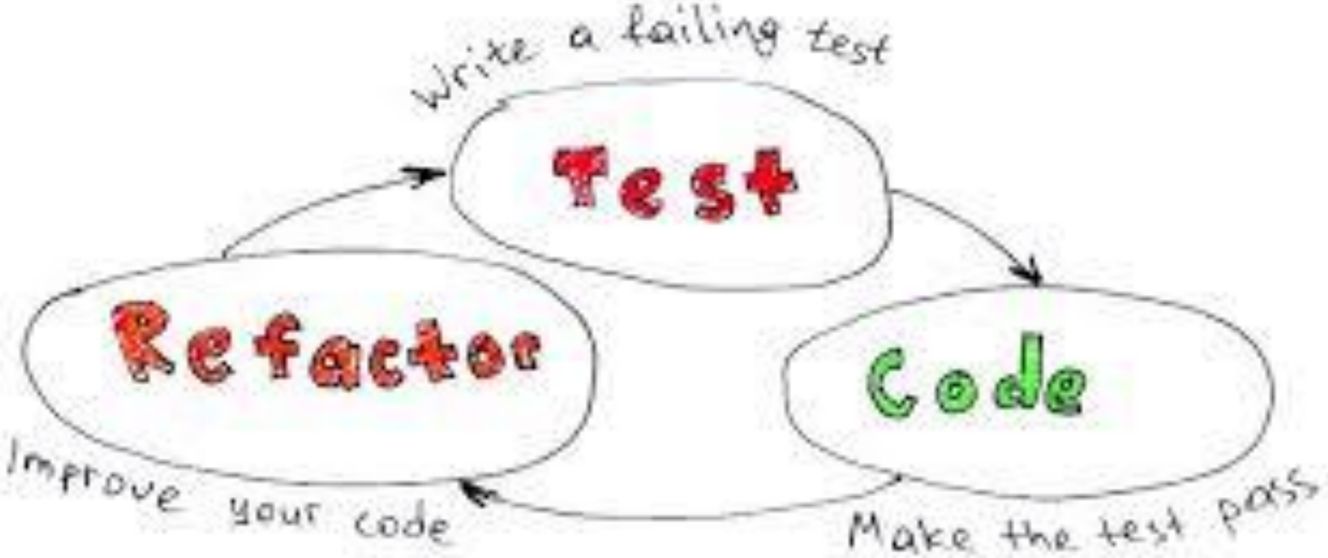


Test-first versus test-last development



Test-Driven-Development

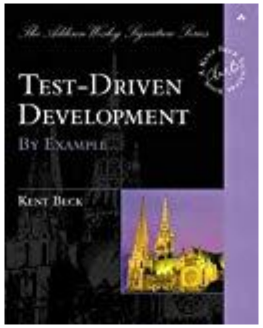
- **To think through your requirements/design before your write your functional code**



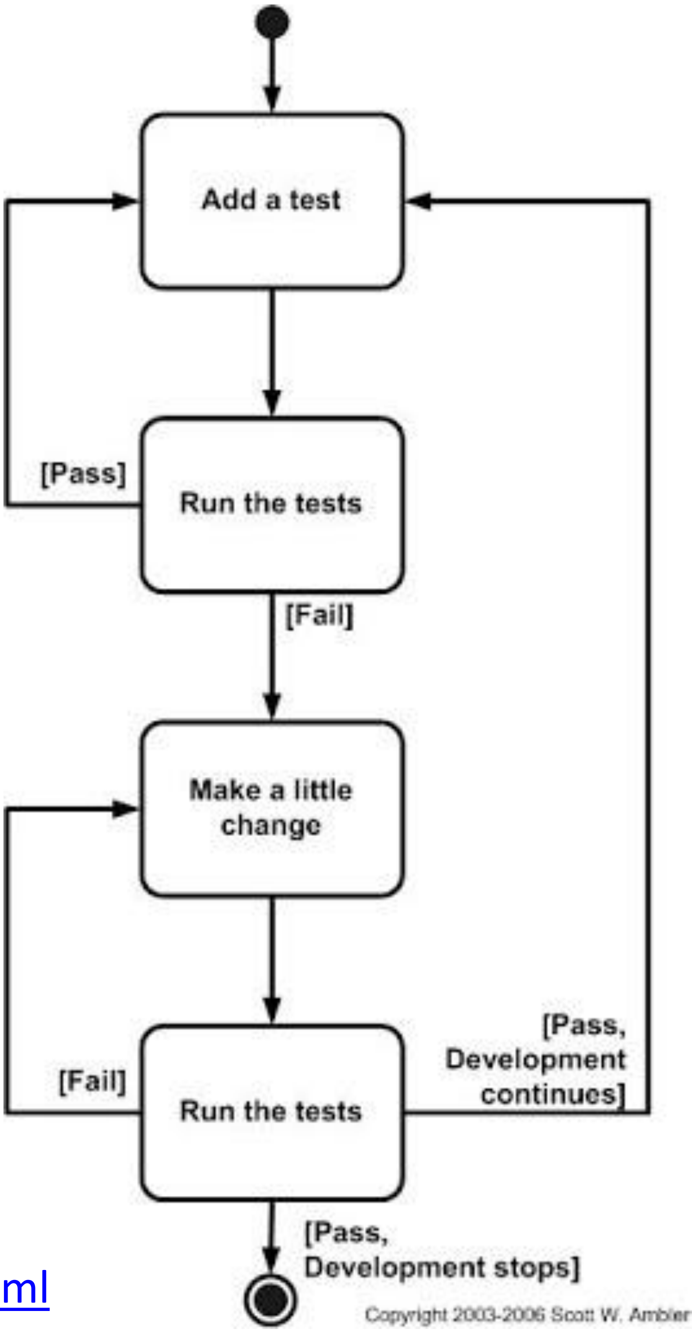
TDD cycle

TDD works in small iterations

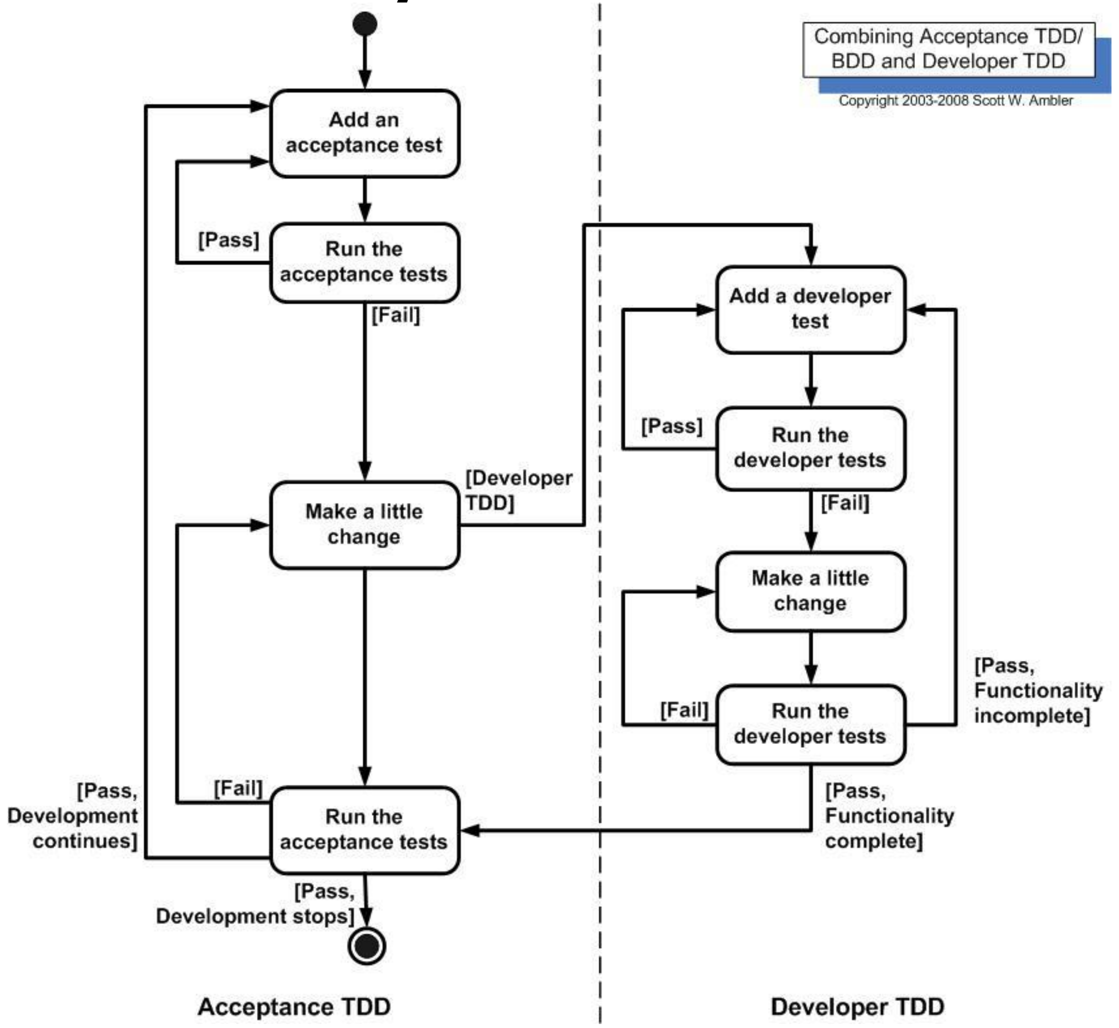
1. add a test
2. run all tests and watch the new one fail
3. make a small change
4. run all tests and see them all succeed
5. refactor (as needed)

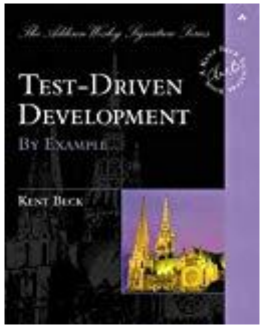


http://www.agiledata.org/essays/tdd.html


Add a test
[Pass]
Run the tests
[Fail]
Make a little change
[Fail]
Run the tests
[Pass, Development continues]
[Pass, Development stops]
Copyright 2003-2006 Scott W. Ambler

ATDD cycle

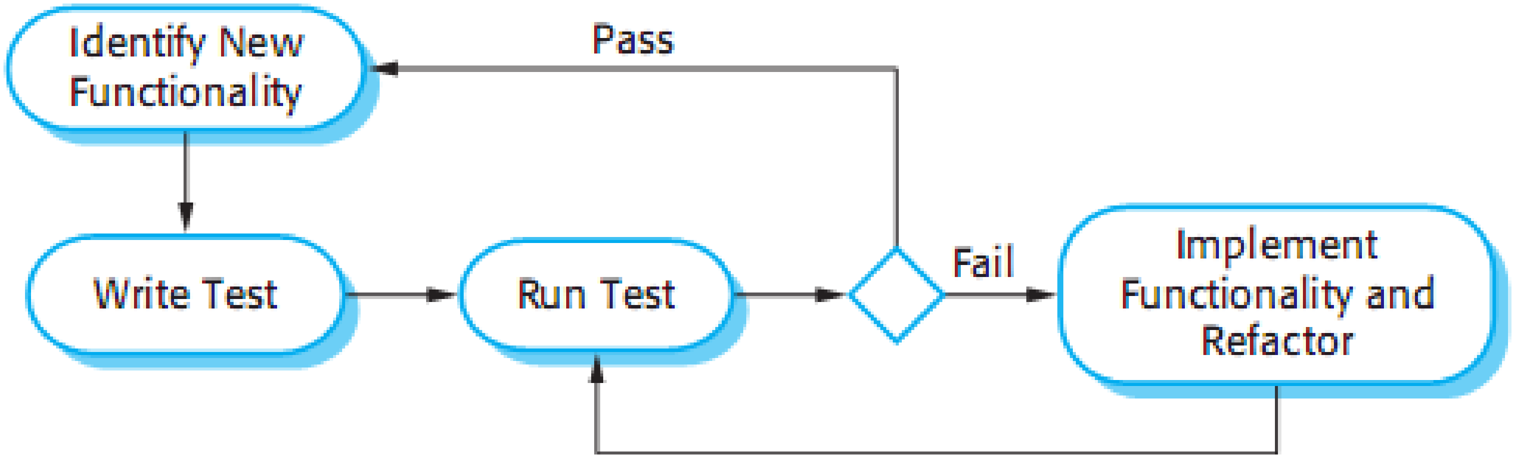




http://www.agiledata.org/essays/tdd.html

Test-driven development

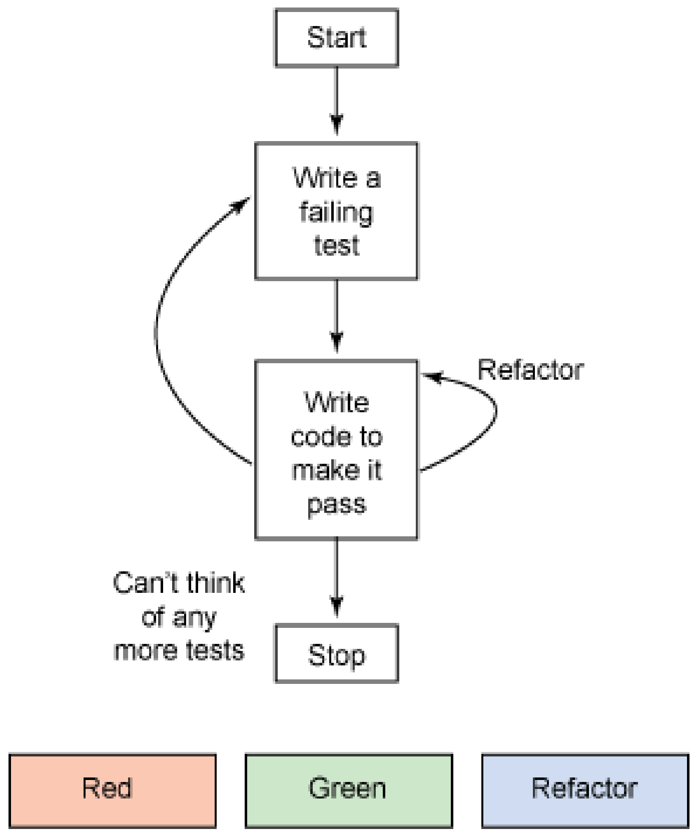
- Test-driven development (TDD) is an approach to program development in **which you inter-leave testing and code development.**
- You **develop code incrementally,** along with a test for that increment. You don't move on to the next increment until the code that you have developed passes its test



Identify New Functionality
Pass
Write Test
Run Test
Fail
Implement Functionality and Refactor

Test-driven development

- Tests are written before code and ‘passing’ the tests is the critical driver of development.

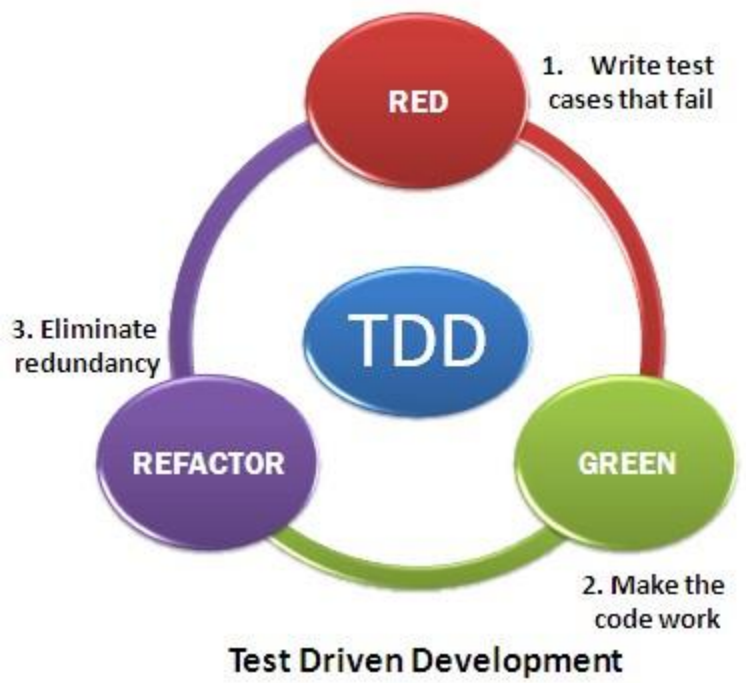


TDD

Red: Create a test and make it fail.

Green: Make the test pass by any means necessary.

Refactor: Change the code to remove duplication in your project and to improve the design while ensuring that all tests still pass.



TDD example

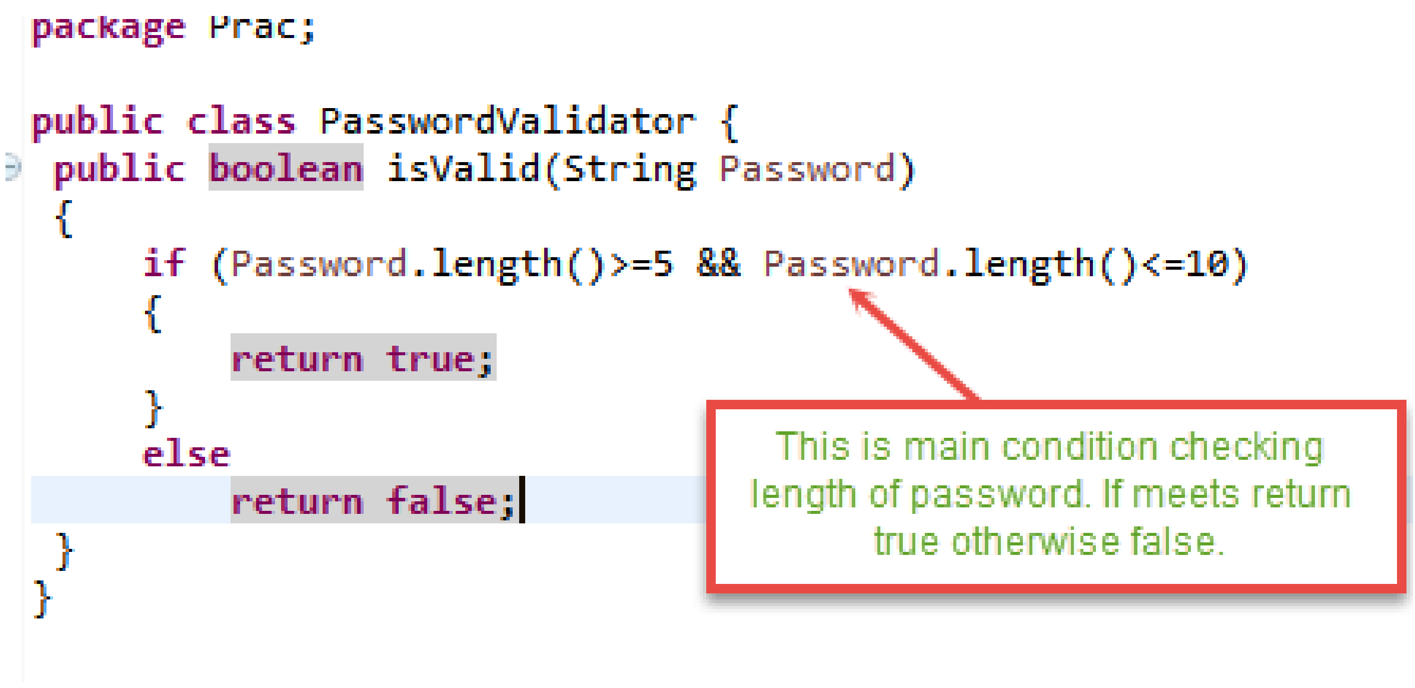
US: *As a bank customer I want to check the strength of my password so that I don't get hacked easily*

AC: *A password should have between 5 and 10 characters*

TDD example

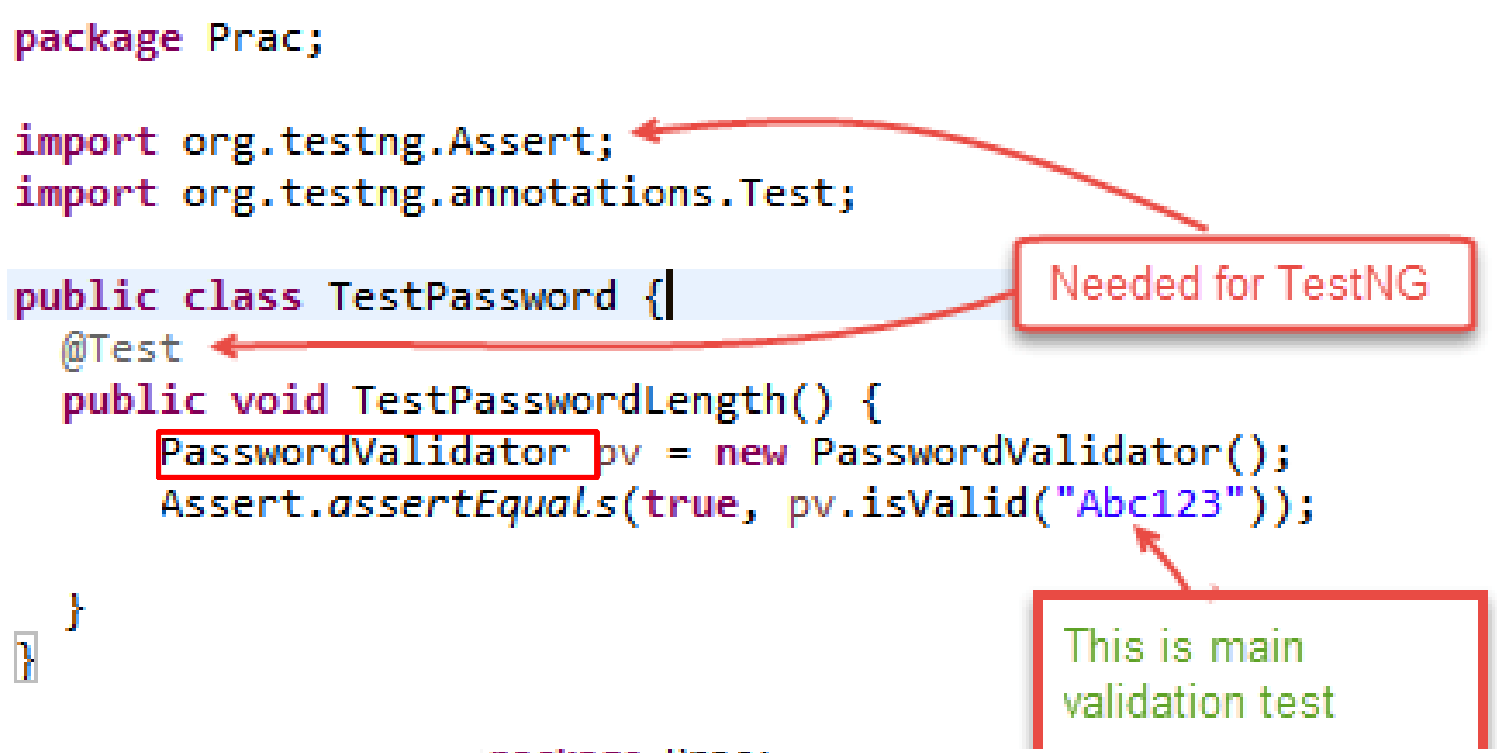
US: *As a bank customer I want to check the strength of my password so that I don't get hacked easily*

AC: ***A password should have between 5 and 10 characters***

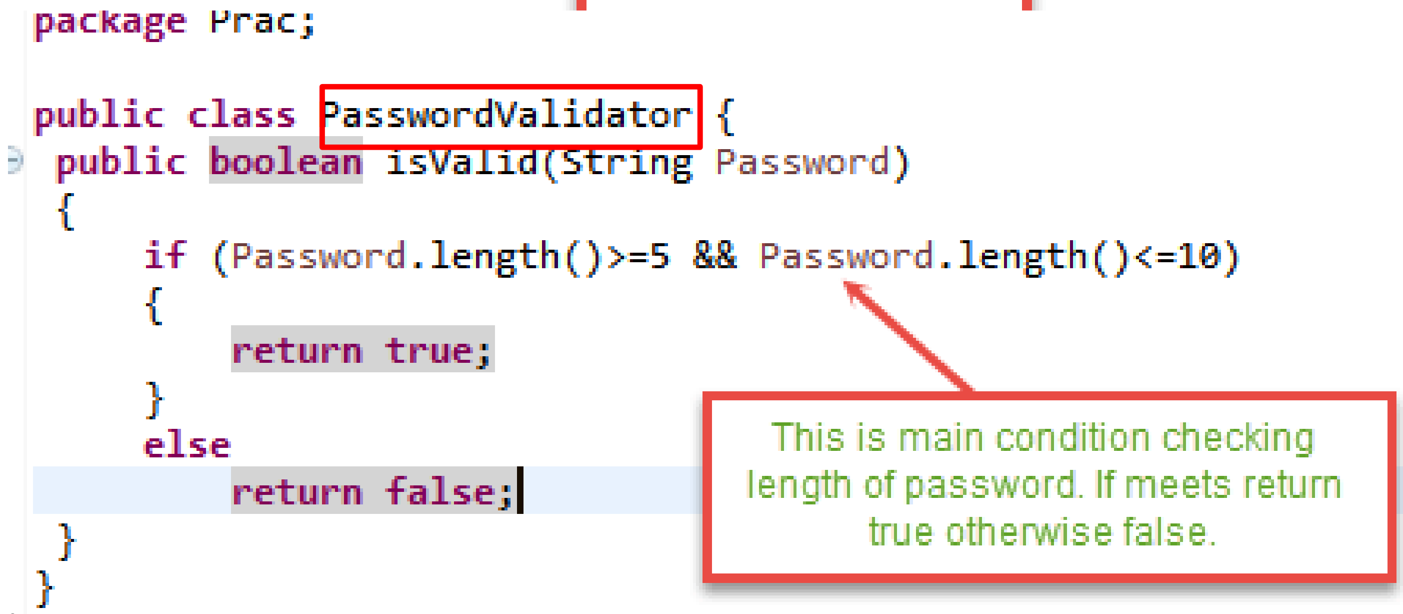


```
package Prac;

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.



```
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"));

  }
}
```



```
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.

https://www.guru99.com/test-driven-development.html

TDD cycle: 1 step: add a test

The input "I", the program must return 1.

```
public class RomanNumberConverterTest {
@Test
void shouldUnderstandSymbolI() {
 RomanNumeralConverter roman = new RomanNumeralConverter(); //fai
       int number = roman.convert("I");
       assertThat(number).isEqualTo(1);
} }
```

TDD cycle: step: write a code

The input "I", the program must return 1.

```
public class RomanNumeralConverter {
     public int convert(String numberInRoman)
{
     return 0; //expected 1 //fail
} }
```

TDD cycle: step: from fail to pass

The input "I", the program must return 1.

```
public class RomanNumeralConverter {

      public int convert(String numberInRoman) {

      if(numberInRoman.equals("I"))

      return 1;
      return 0;
} }
```

TDD cycle: step: add a new test

The input „V", the program must return 5.

```
@Test
void shouldUnderstandSymbolV() {
      RomanNumeralConverter roman = new
RomanNumeralConverter();
      int number = roman.convert("V");
      assertThat(number).isEqualTo(5);
}
```

TDD cycle: step: from fail to pass

The input „V", the program must return 5.

```
public class RomanNumeralConverter {

       public int convert(String numberInRoman) {

       if(numberInRoman.equals("I"))  return 1;
       if(numberInRoman.equals(„V"))  return 5;


       return 0;
} }
```

Whye need fail test?

1. it verifies the test works, including any testing harnesses,
2. demonstrates how the **system will behave if the code is incorrect.**

TDD process activities

- Start by **identifying the increment of functionality** that is required.
  - This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this **as an automated test.**
- Run the test, **along with all other tests** that have been implemented.
  - Initially, you have not implemented the functionality so the new test will fail.
- Implement the functionality and **re-run the test.**
- Once all tests run successfully, you move on to implementing the **next chunk of functionality.**

TDD Limitations

- Non-productive and **hard to learn**
- Difficult in Some Situations
  - **GUIs, Relational Databases, Web Service**
  - Requires mock objects
- TDD does not often include an upfront design
  - Focus is on implementation and **less on the logical structure**
- Difficult to write test cases for hard-to-test code
  - **Requires a higher level of experience from programmers**
- **TDD merge distinct phases of software development**
  - design, code and test