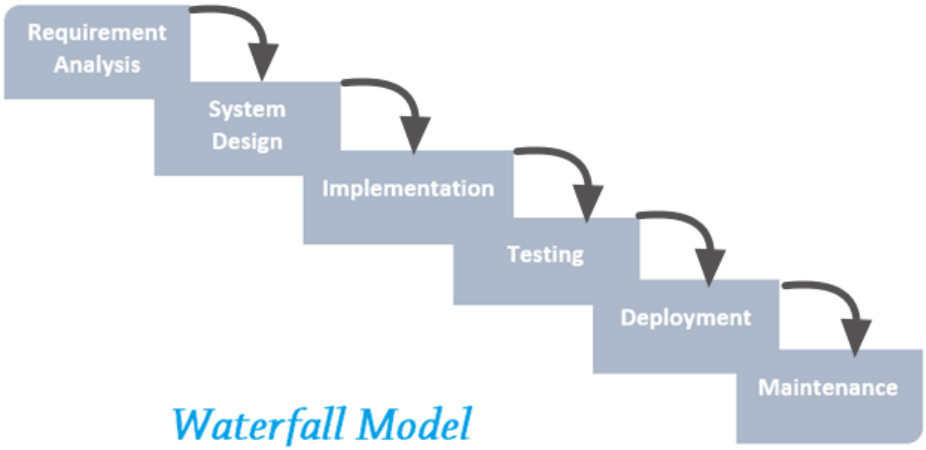
Software System Testing

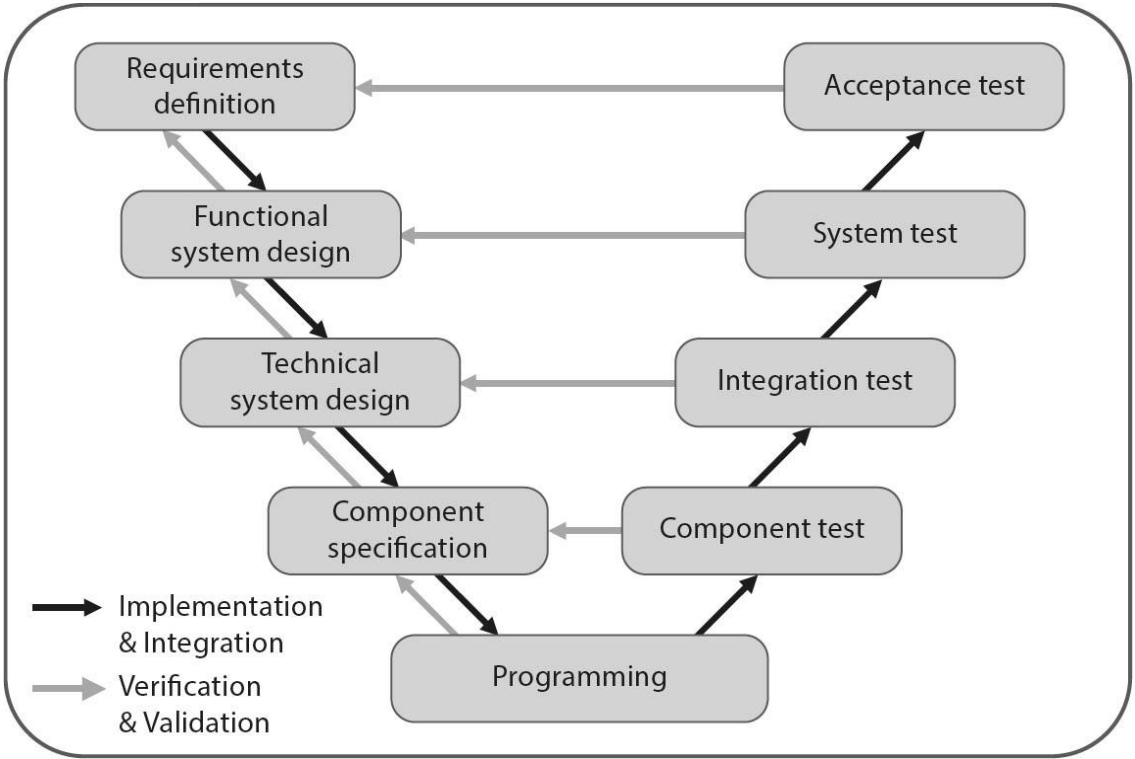
5th lecture

Assoc. Prof. Dr. Asta Slotkiene

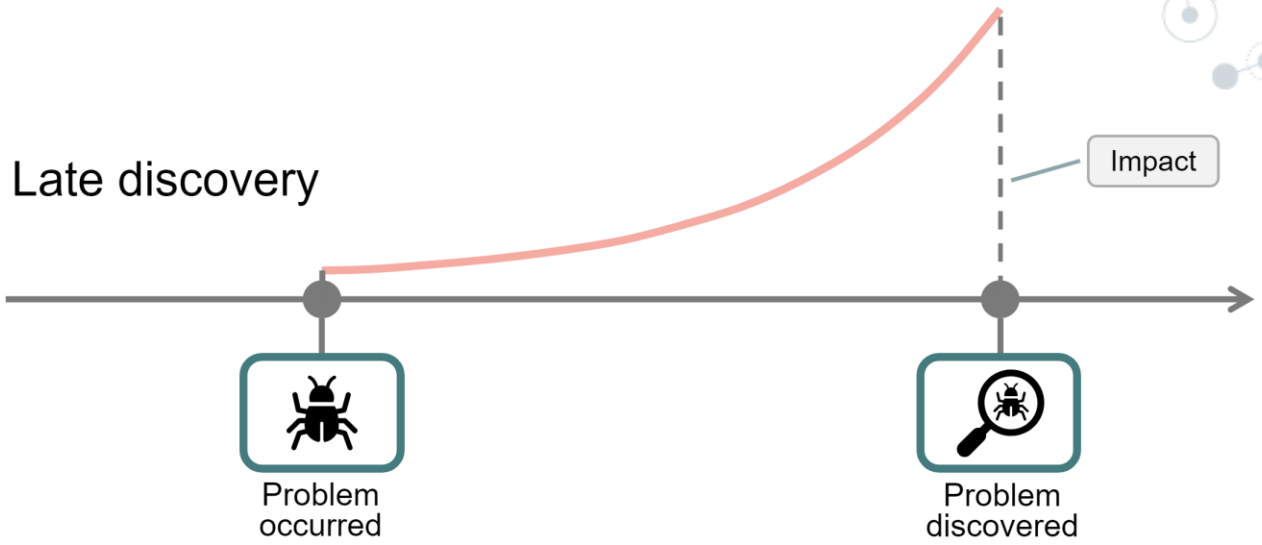
Software Development Models

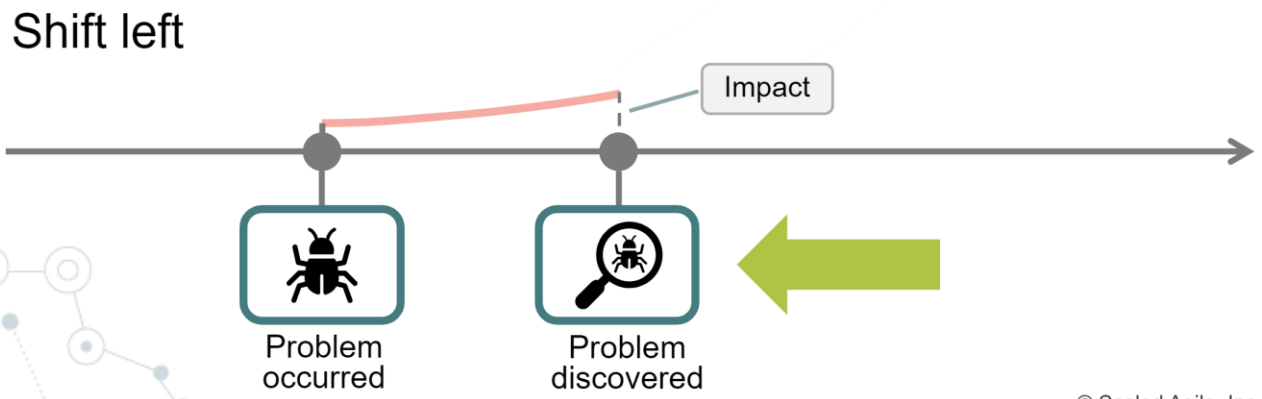


Waterfall Model



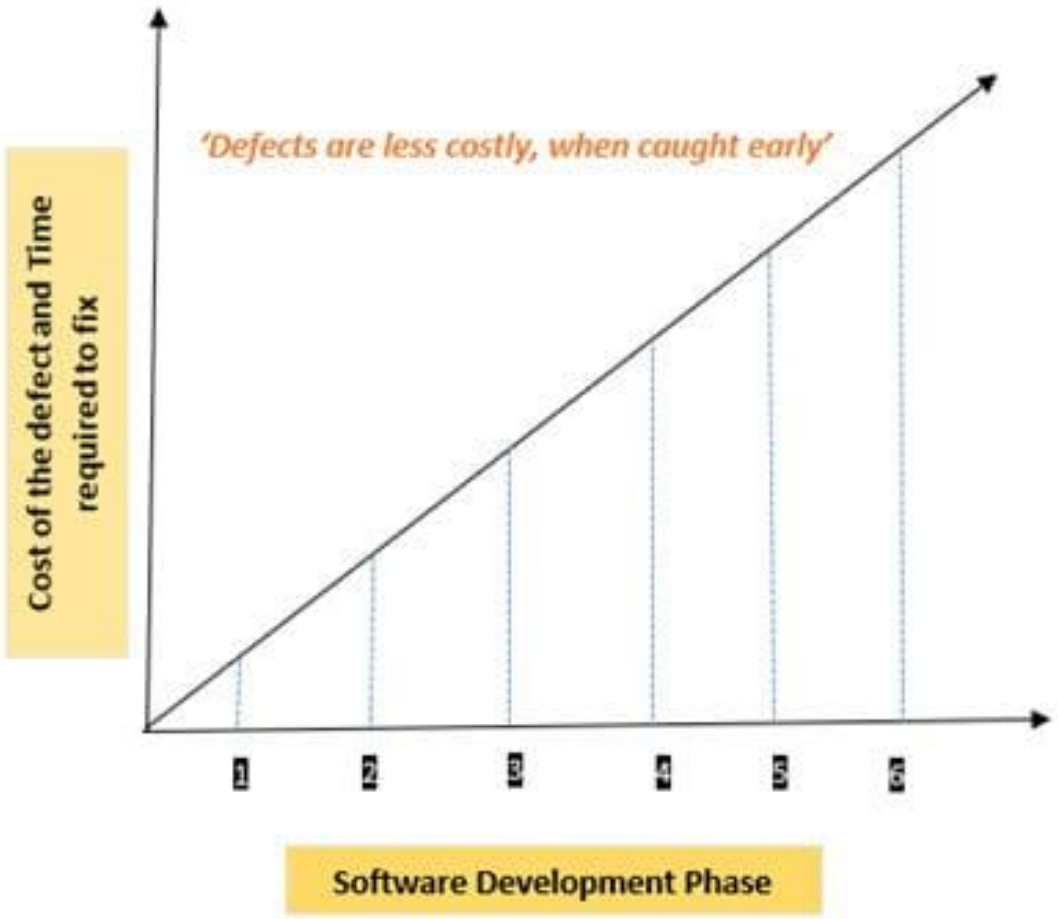
Tradicional vs Agile Quality Practices





© Scaled Agile, Inc.

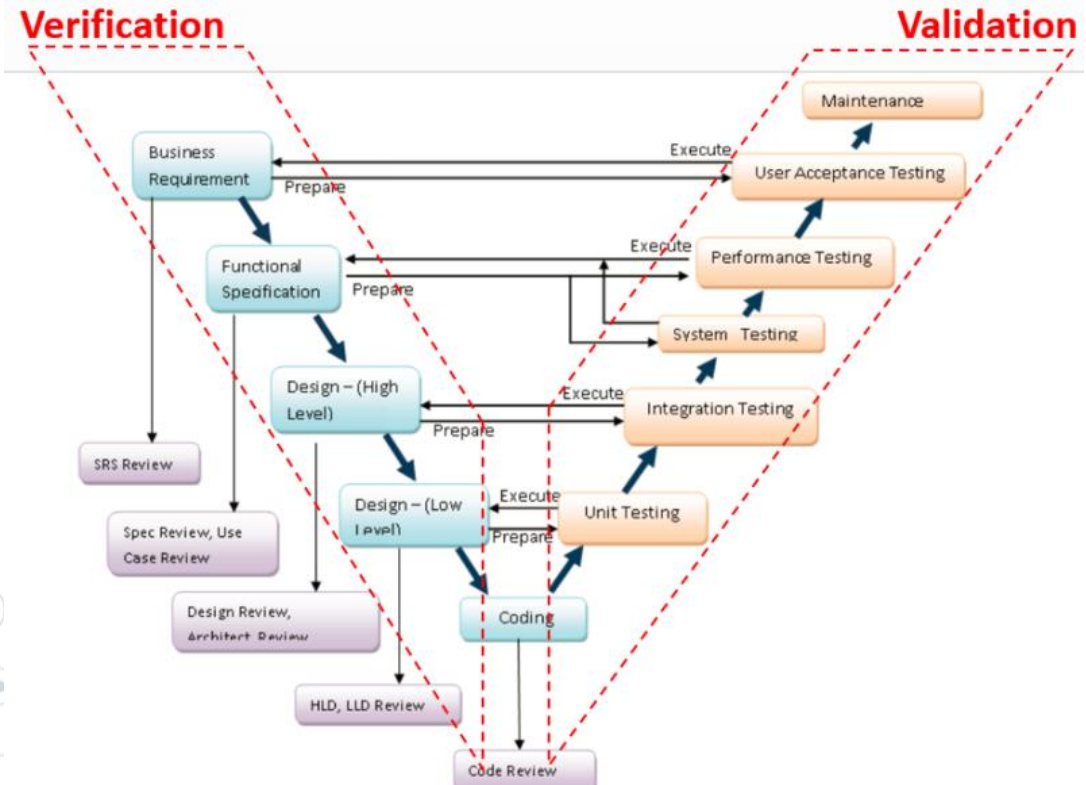
Why Shift Left Testing?



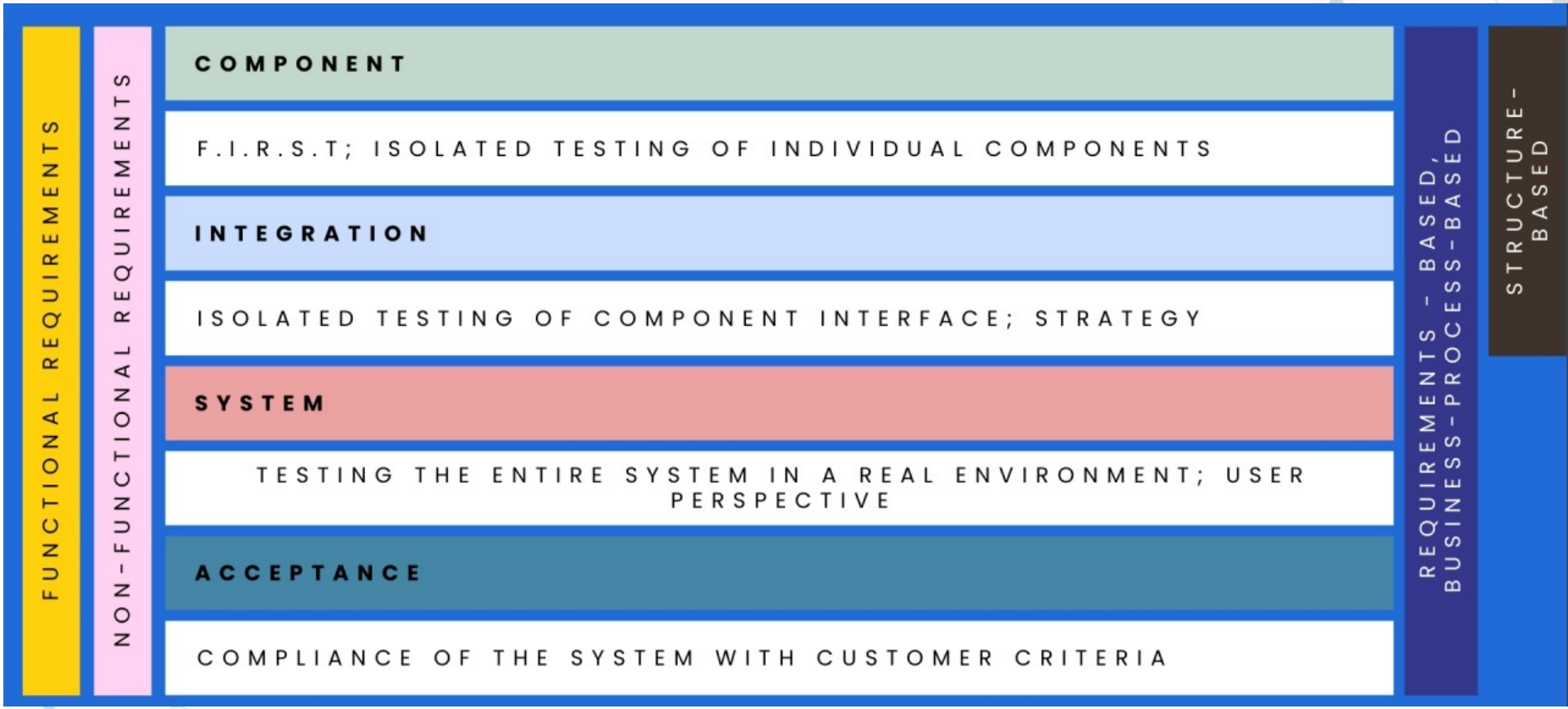
1. Requirement Stage
2. Design Stage
3. Coding Stage
4. Testing Stage
5. Production Deployment Stage
6. Post Go-Live Stage

Software development models

- Testing needs to begin as early as possible in the life cycle.
- Testing can be integrated into each phase of the life cycle.
- Within the V-model, validation testing takes place especially during
  - the early stages,i.e. reviewing the user requirements
  - and late in the life cycle, i.e.during user acceptance testing



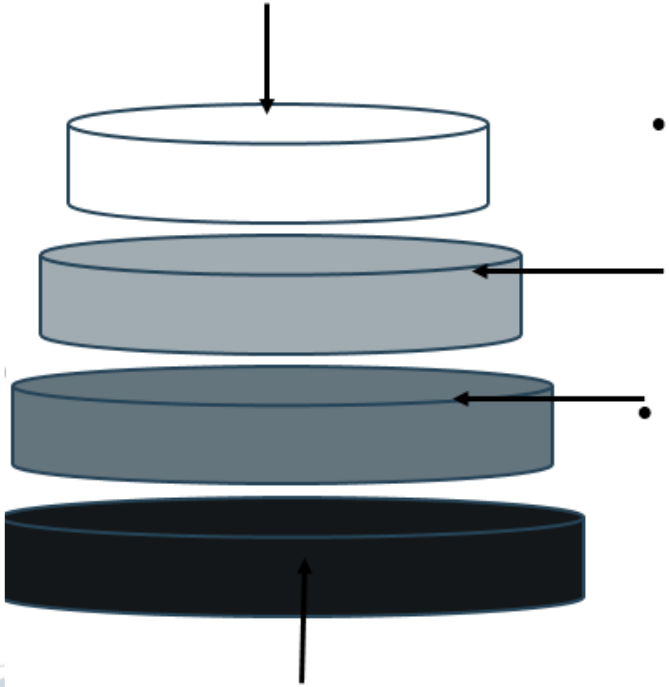
Testing Levels



Testing Levels

- **Acceptance**

  Is the responsibility of the customer – in general. The goal is to gain confidence in the system; especially in its non-functional characteristics



- **System**

  The behavior of the whole product(system) as defined by the scope of the project

- **Integration**

  Interface between components; interactions with other systems (OS, HW, …)

- **Unit**

  Any module, program, object separately testable

Testing Level: Unit/Component Testing

- **Unit/Component tests** ensure that each individual component fulfills its specified requirements.
- A unit is the smallest testable part of an application.
- 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
- Goal: ensure that groups of components interact as specified by the technical design
- More than one (tested) unit
  - Detecting defects: On the interfaces of units or Communication between units
- 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.

Testing Level: System Testing

- Who: Normally, independent Testers perform System Testing
- Goal: ensure that the system as a whole functions according to its specified requirements
- Types: Smoke Testing, Functional Testing, Usability Testing, Security Testing , Performance Testing, Regression Testing ,Compliance Testing and etc.
- How:
  - Usually, Black Box Testing method is used.

Testing Level: Acceptance Testing

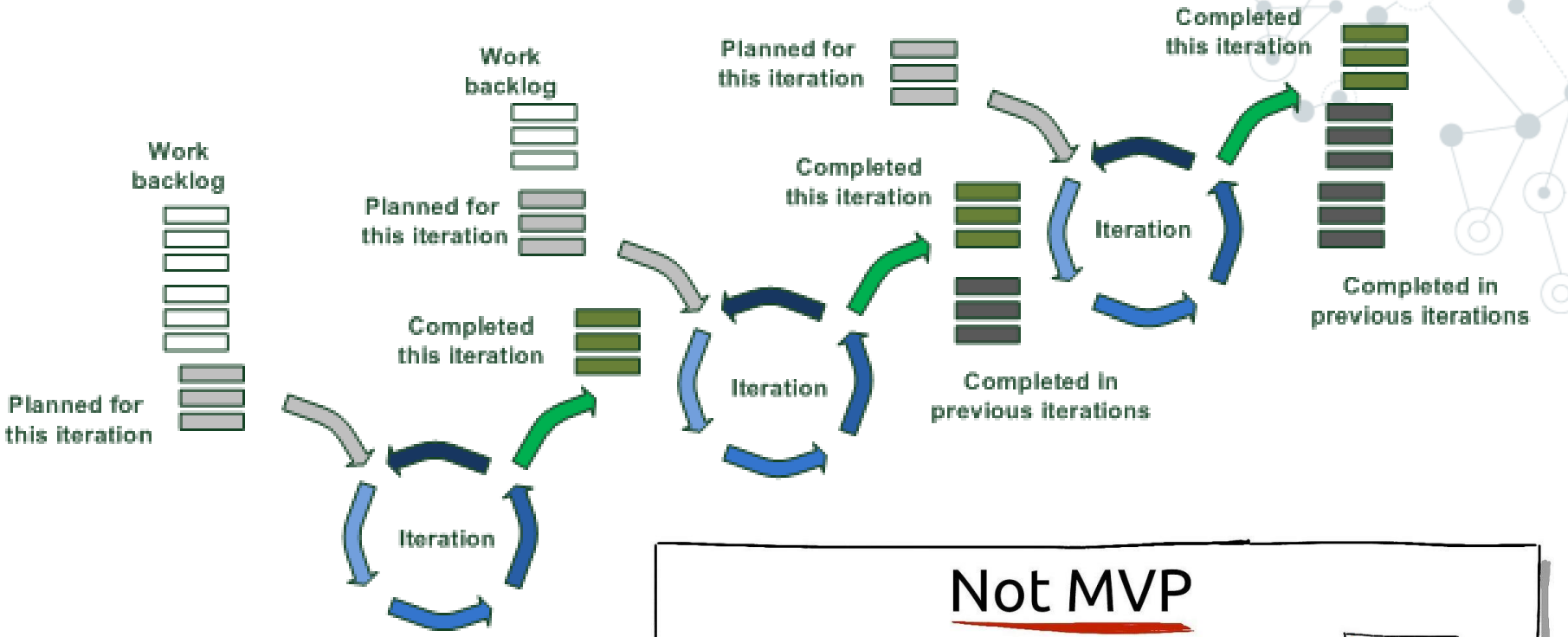
- Who: Product Management, Sales, Customer Support, Customers
- Goal: checks that the system as a whole adheres to the contractually agreed customer and end-user criteria.
- How:
  - Usually, Black Box Testing method is used;
  - often the testing is done ad-hoc and non-scripted

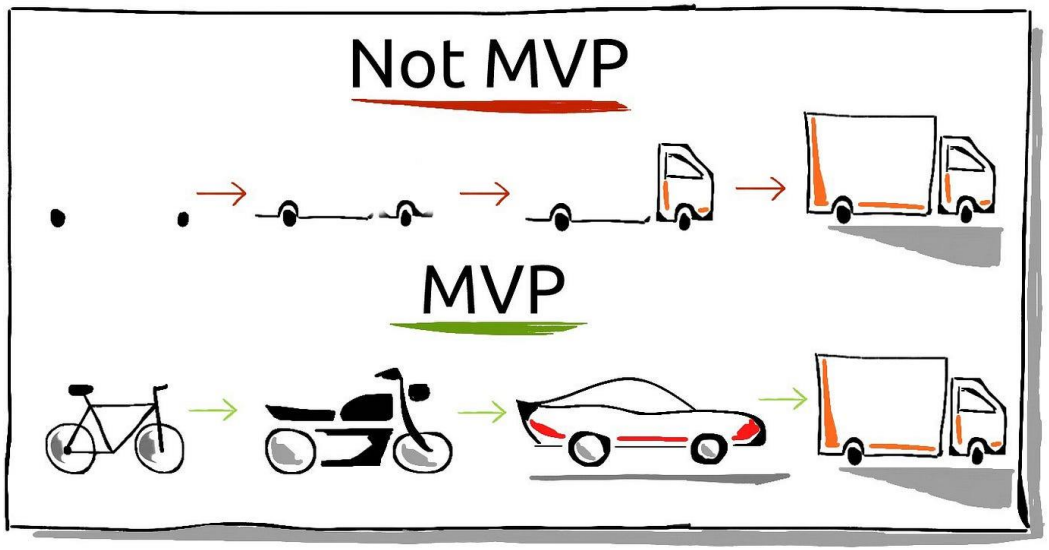
also called: Behavior-driven testing (BDD)

Testing Levels

- For each test level, please note:
  - The generic objectives
  - The test basis (docs/products used to derive test cases)
  - The test objects (what is being tested)
  - Typical defects and failures to be found
  - Specific approaches

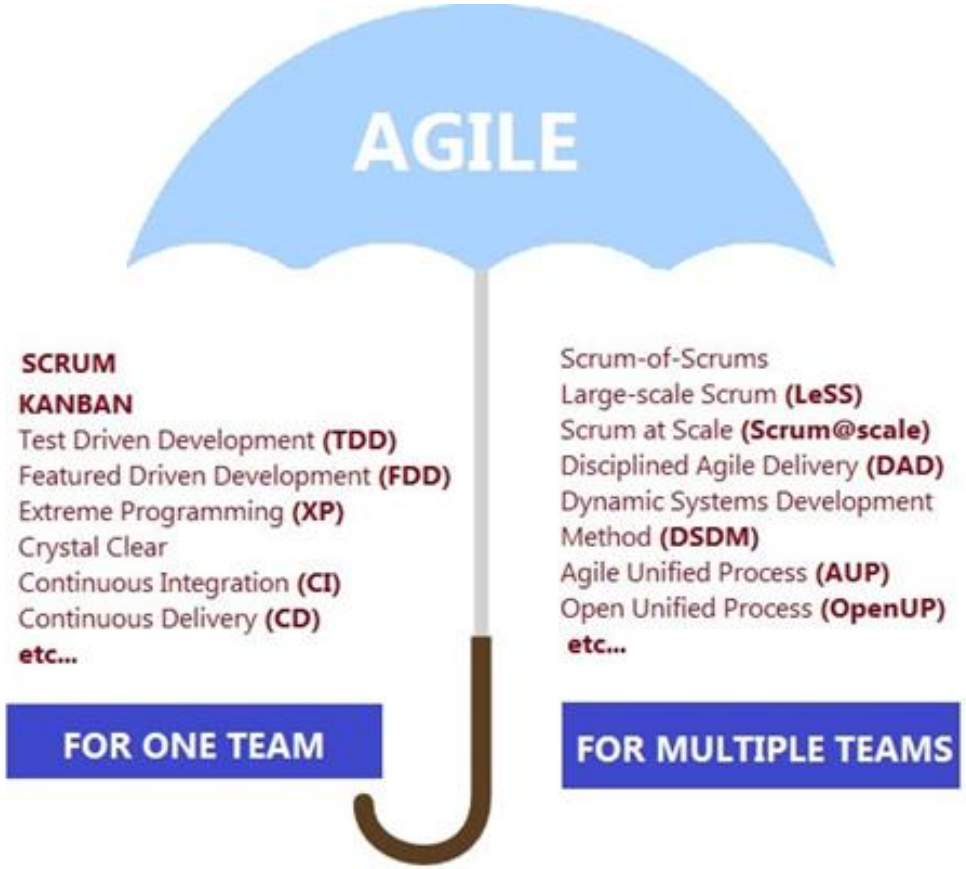
Iterative and Incremental Development Models





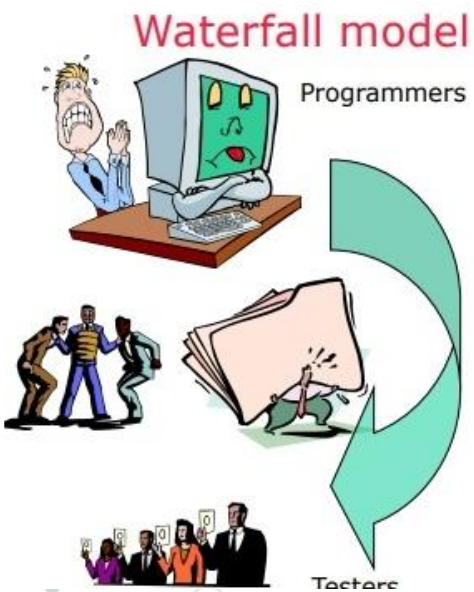
Iterative and Incremental Development Models

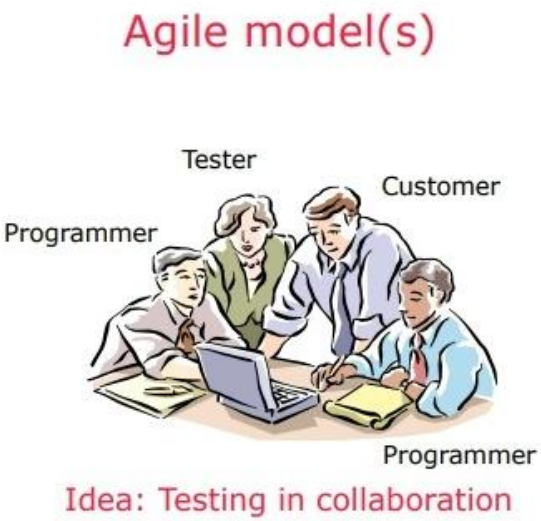
- All forms of agile software development are iterative-incremental development models.
- The best-known agile models are:

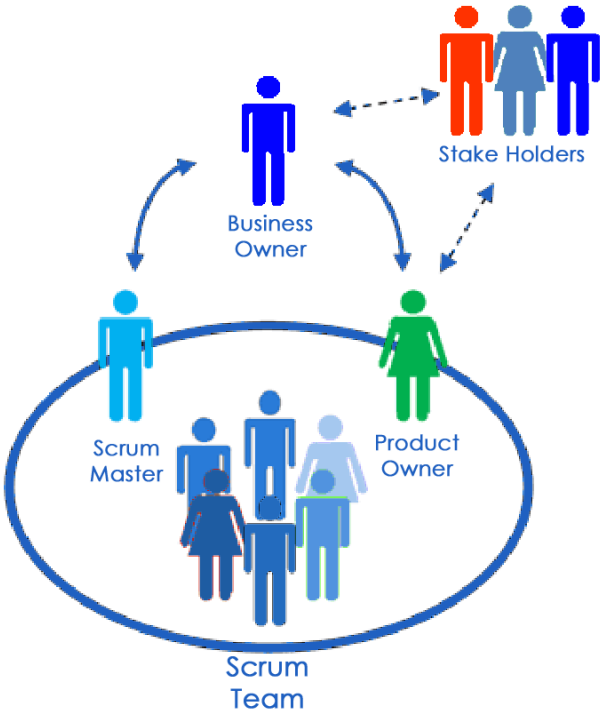


Test Planning and Test Management: Scrum

- Scrum Team **has no dedicated test manager** and distributes the responsibilities associated with this role within the team

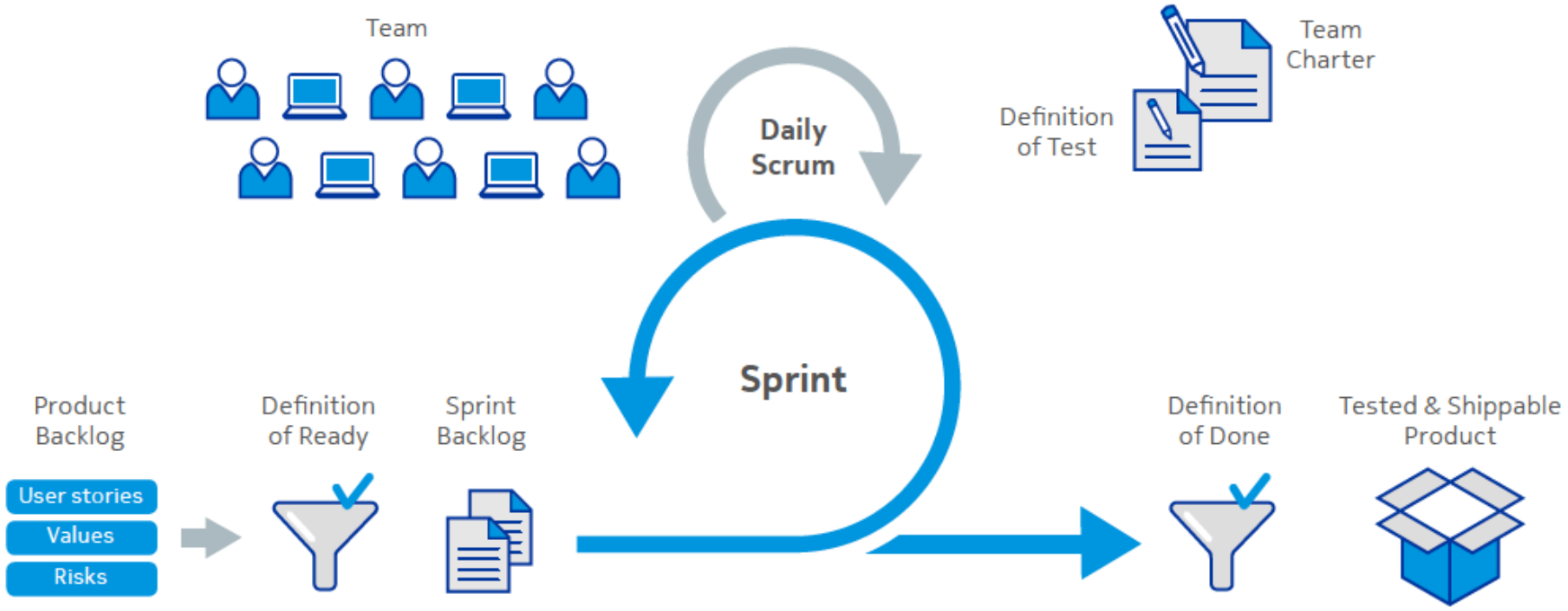






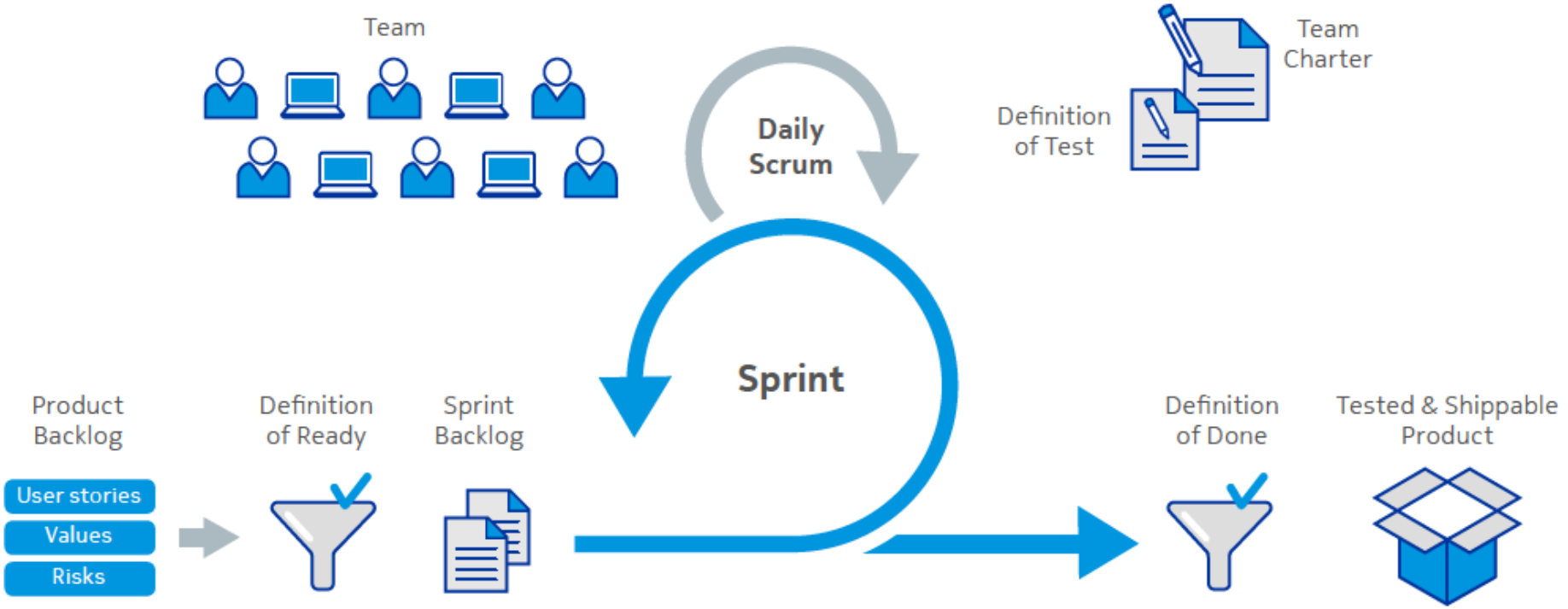
Test Planning and Test Management: Scrum

- Testing tasks are scheduled **explicitly as tasks within a Sprint** or **implicitly as part of the DoR and DoD criteria** of other tasks.



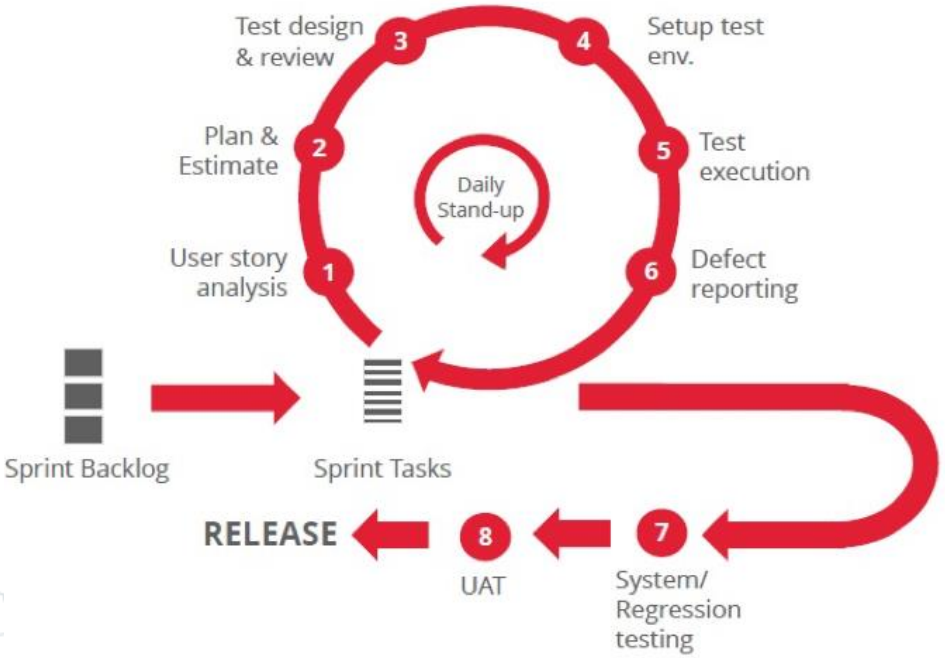
Test Planning and Test Management: Scrum

- Team requires a member with dedicated testing expertise.
- **Responsible for designing appropriate risk-oriented tests and implementing those in every Sprint.**

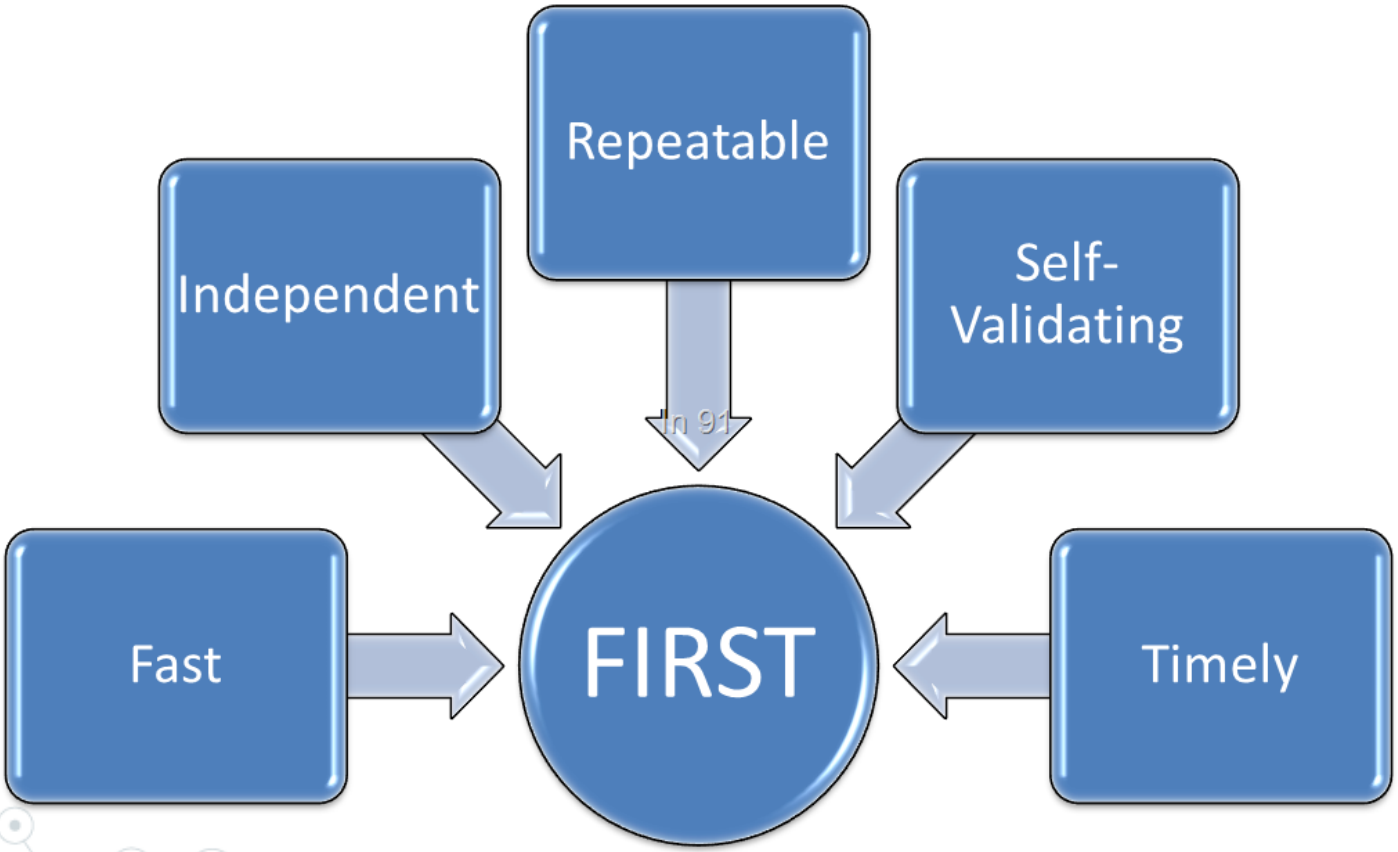


Challenges of Testing in Agile Development

- Requirements change all the time
- Specification documents are never final
- Code is never ‘finished’, never ‘ready for testing’
- Limited time to test
- Need for **regression testing** in each increment



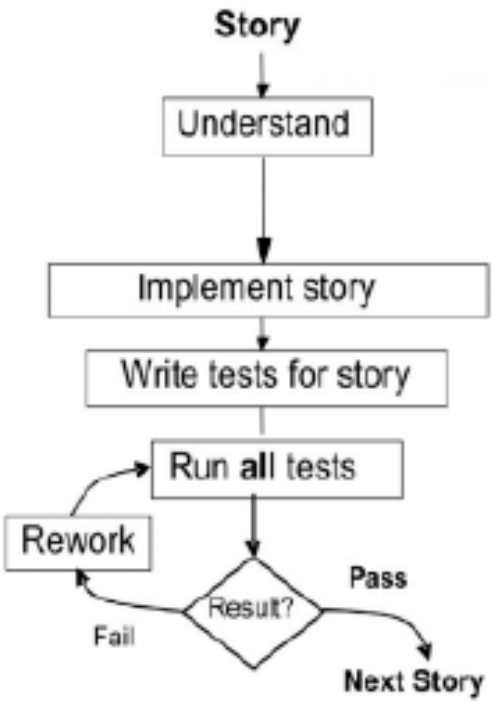
F.I.R.S.T. test principles



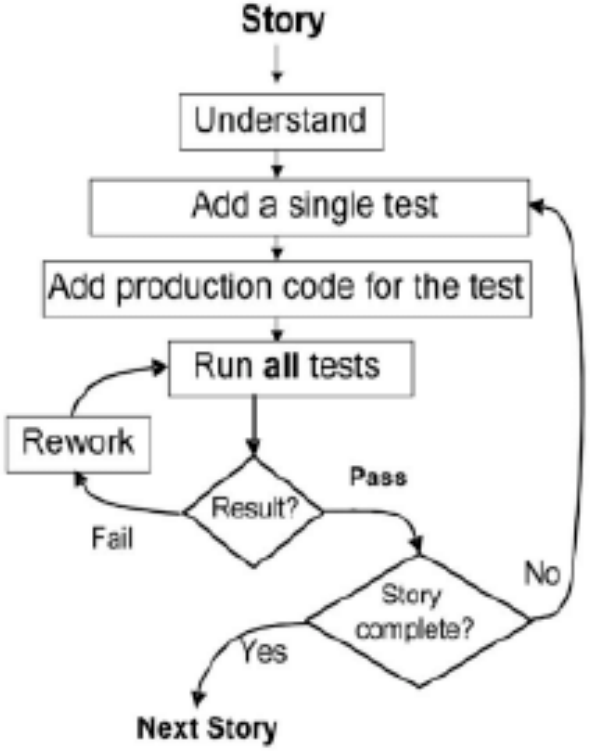
https://agileinaflash.blogspot.com/2009/02/first.html

Test-first versus test-last development

- Test First means considering, which tests will be necessary to show that the software actually fulfills any new specifications before any changes in the code itself are performed.
  - Before you alter your code, write an automated test that fails. (Test Driven Development).



Test - Last



Test - First

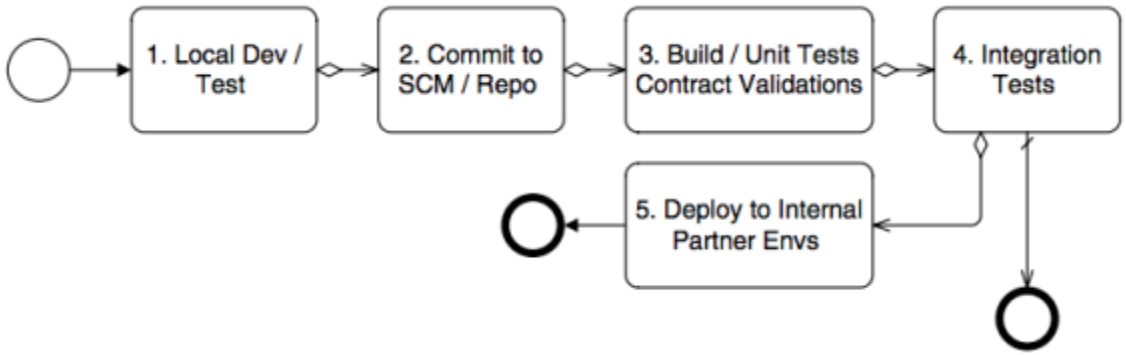
Test-driven development (TDD)

- Test-driven development - 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/***

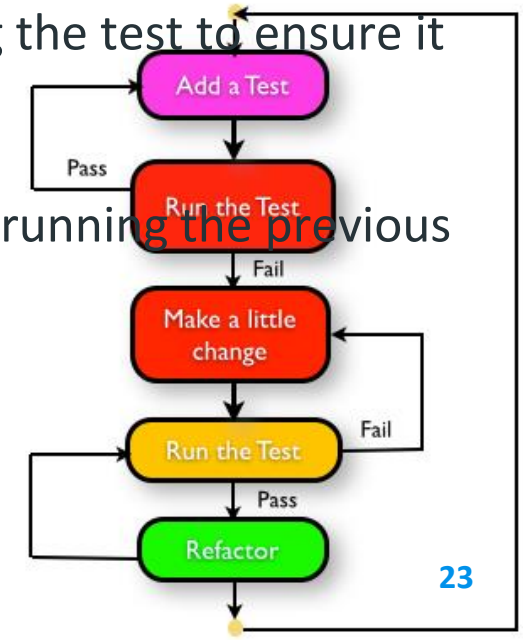
Unit testing and Scrum

- Scrum itself doesn't demand the use of Test First and works just as well with traditional unit tests.
- However, the team can benefit significantly on a technical level and through the acceleration of the feedback loop it provides:
  - **For every change in the code, the programmer receives immediate feedback about its success or failure.**



Unit testing and Scrum:TDD

- The process for test-driven development is:
  1. Add a test that captures the programmer's concept of the desired functioning of a small piece of code
  2. Run the test, which should fail since the code doesn't exist
  3. Write the code and run the test in a tight loop until the test passes
  4. Refactor the code after the test is passed, re-running the test to ensure it continues to pass against the refactored code
  5. Repeat this process for the next small piece of code, running the previous tests as well as the added tests



TDD cycle: 1 step-> *Add a test that captures the programmer's concept of the desired functioning of a small piece of code*

The user entered "I", the program must return 1.

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

TDD cycle: 2 step->*Run the test, which should fail since the code doesn't exist*

The user entered "I", the program must return 1.

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

TDD cycle: 3 step-> *write the code and run the test in a tight loop until the test passes*

The user entered "I", the program must return 1.

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

TDD cycle: *4 steps->refactor the code after the test is passed, re-running the test to ensure it continues to pass against the refactored code*

The user entered "I", the program must return 1.

```
public class RomanNumeralConverter {

      public int convert(String numberInRoman) {

         if(numberInRoman.equals("I"))

            return 1;
            return 0;
}
}
```

TDD cycle: 5 step ->*repeat this process for the next small piece of code, running the previous tests as well as the added tests*

The user entered „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;
} }
```

Why 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

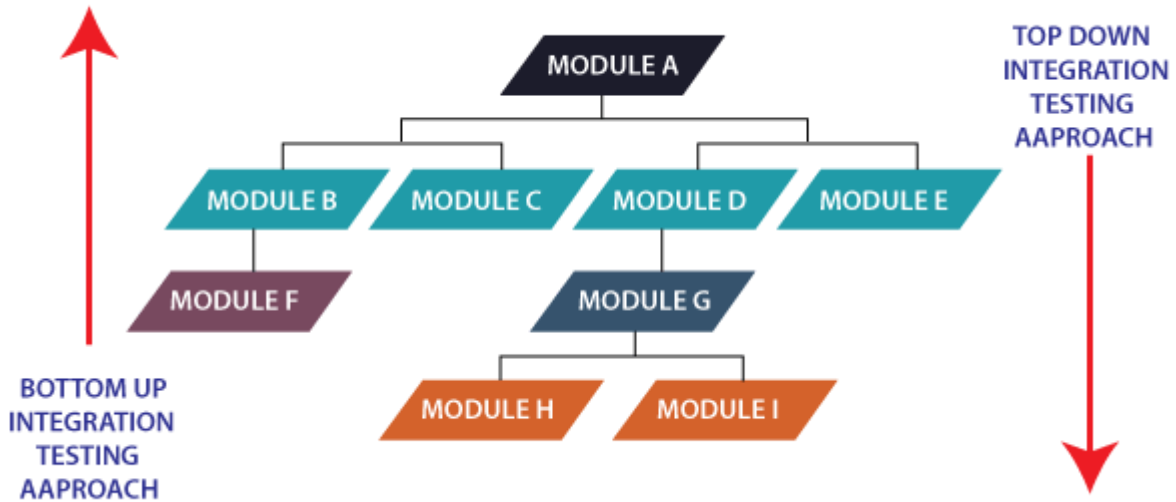
- Difficult in Some Situations
  - **GUIs, Relational Databases, Web Service**
  - Requires Mock objects
- TDD does not often include an upfront design
  - Focus is on implementation and **less on the logical structure**
- Difficult to write test cases for hard-to-test code
  - **Requires a higher level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

What need to do Scrum Master in the testing process?

- Make sure that all team members write similarly structured tests.
  - Tests have to be stored centrally on the team's development network in a file system with an agreed structure.
- **Ensure that test coverage is measured reliably** and its Sprint-by-Sprint development analyzed regularly.
  - The necessary coverage limits should be established in accordance with the risk analysis performed for each unit.
- Ensure **static code analysis**
- Ensure that regular program code reviews take place.
- Ensure that the test code is **reviewed** regularly.
- Define Unit testing **guidelines**

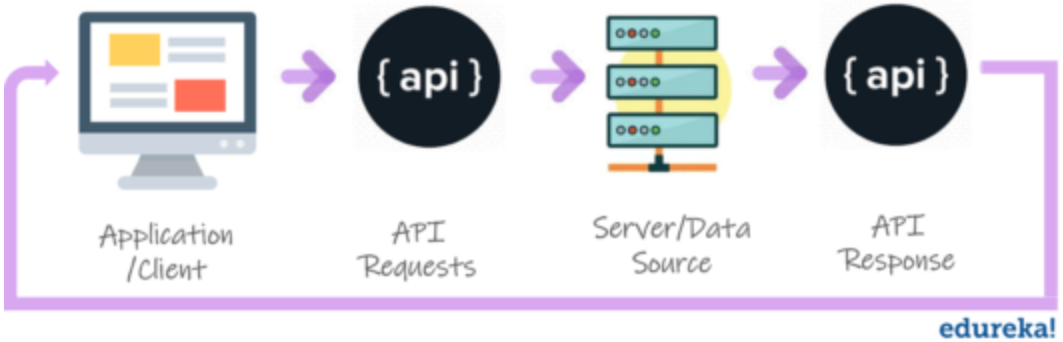
Integration Testing

- Integration tests are designed to discover **potential defects in the interaction** of the individual components and their interfaces.
- Even if just a single component is altered, **always should re-run not only its unit test, but also all the integration tests** that cover components that it interacts with.

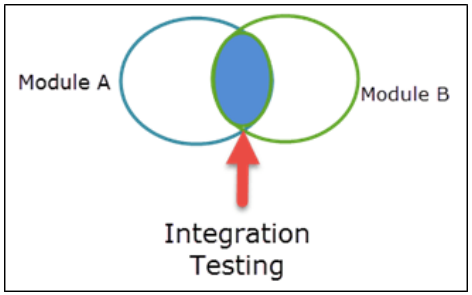


Integration Testing inludes:

- Dependencies:
  - **Explicit**. If a component A calls a component B directly via its API.



  - **Implicit**. If multiple components share a single resource (global variable, same database, etc.)



Integration Testing in the Scrum: CI/CD

- The CI Process:
  - Code repository (check in as soon as possible)
  - Automated integration runs
  - Compilation
  - Static code analysis
  - Deployment to the test environment
  - Initialization Unit Testing
  - Integration testing
  - System testing
  - Feedback and dashboard

Continuous Integration

- Each check-in is then verified by an automated build,
  - allowing teams to **detect problems early**.

Cyclic delivery
of earlier times

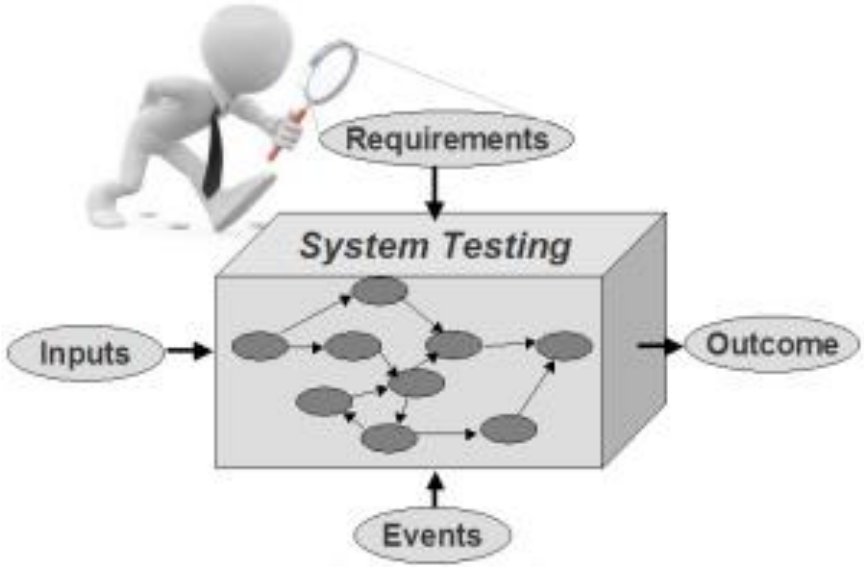
continuous delivery
of modern times

Integration Test Management in Scrum

- **Ensure that a sufficient number of appropriate test cases are written and run**
- Can and should be designed using **Test First principles**
- Take integration testing effort into account during Sprint Planning
- Check whether **additional integration-related code analysis is possible**
- Attention to sorting automated integration tests into batches and the continuous optimization of the speed of the CI process.

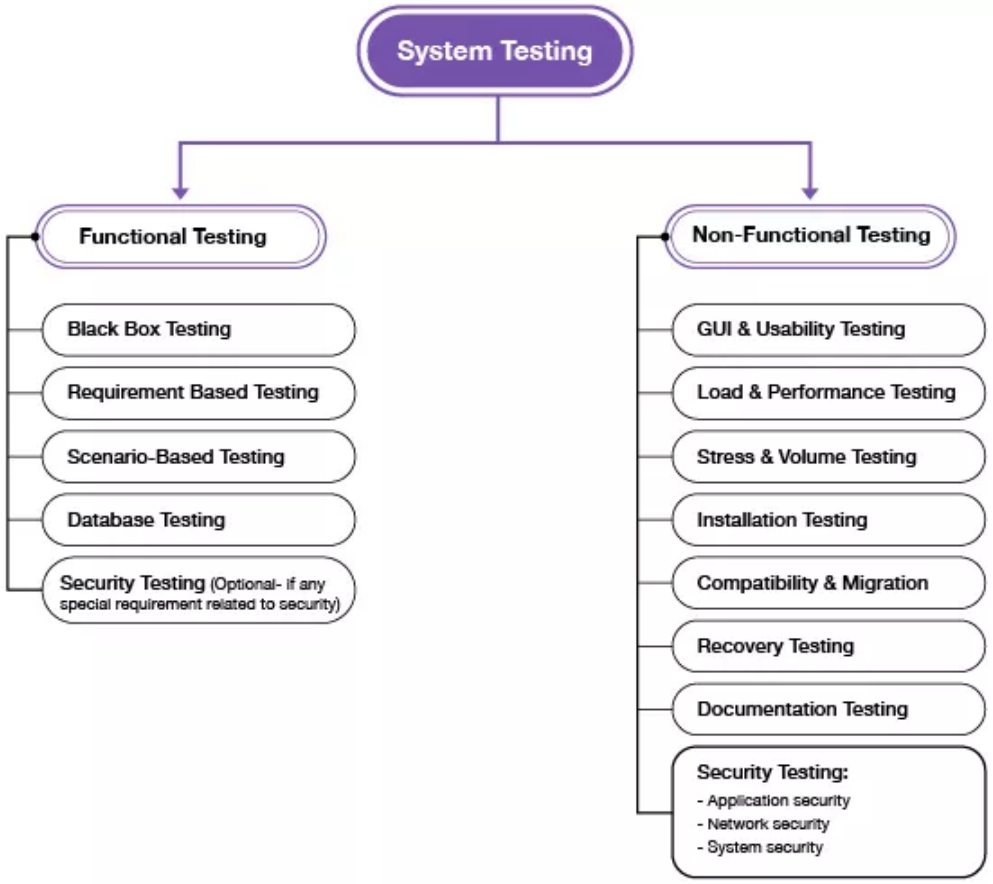
System Testing in the Scrum

- **System tests check that the system works from the user's point** of view using the customer's interfaces.
- It can be derived directly **from the requirements and acceptance criteria** listed in the Product Backlog or from the corresponding use case description.
- The dialog with the Product Owner should be used to clarify when and whether requirements listed in the Backlog need **to be more precisely defined** and weather additional requirements need to be drafted and added to the Backlog.



System testing in the Scrum

- The effort involved in system testing **does not scale according to the number of system test cases**, but rather **with the number of different test environments and quality criteria that need to be checked for functionality.**



Behavior driven development (BDD)

- Behavior driven development - a collaborative approach to development in **which the team focuses on delivering the 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

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. Show the list	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

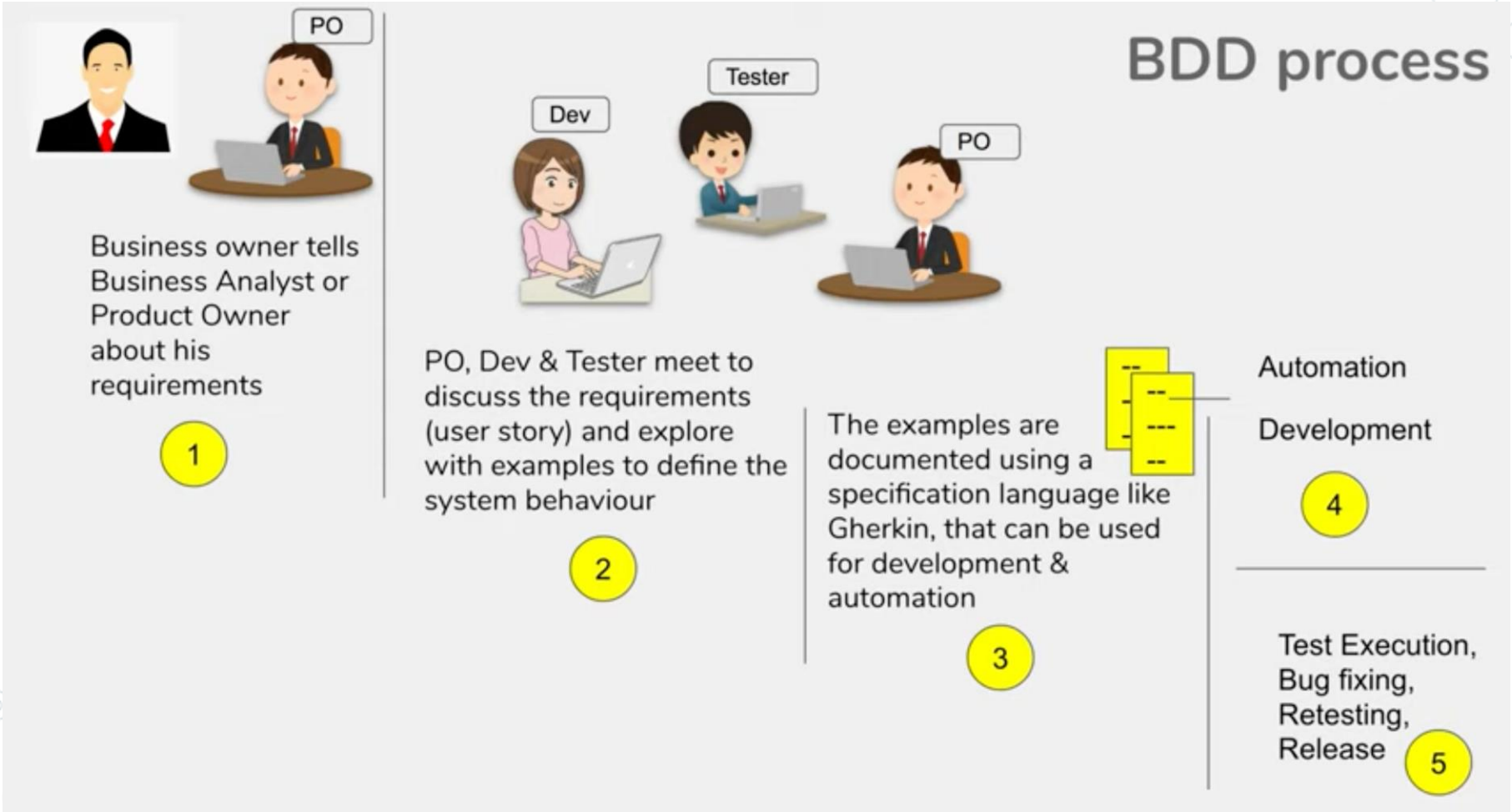
Who should write acceptance tests?

I 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. Show the list	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

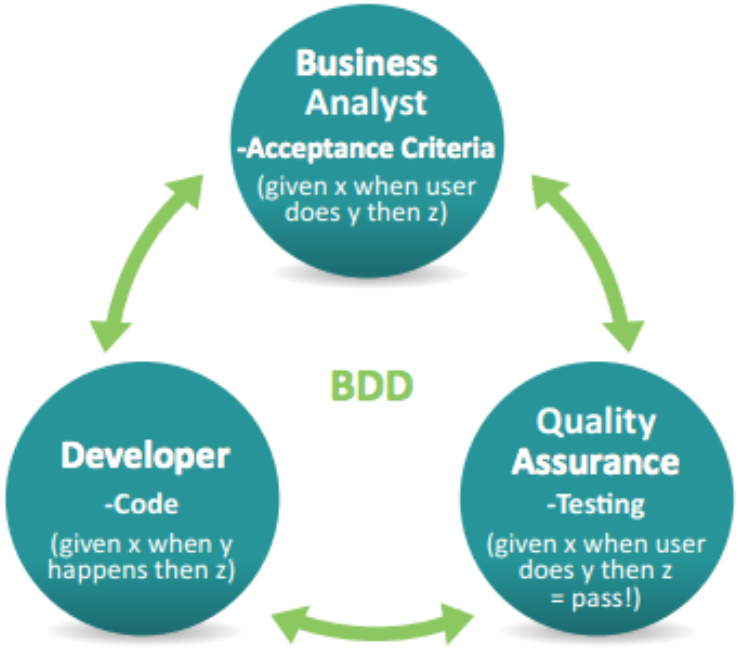
- Lots of detail
- Must document passes manually
- Maintenance issue when steps depend on each other

Behavior based development (BDD)



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**



Behavior Driven Development: Big Idea

- **Tests from customer-friendly user stories**
  - Acceptance: **ensure satisfied customer**
  - Integration: ensure interfaces between modules have consistent assumptions, and communicate correctly
- **Meet halfway between the customer and developer**
  - User stories are not code, so clear to the customer and can be used to reach an 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

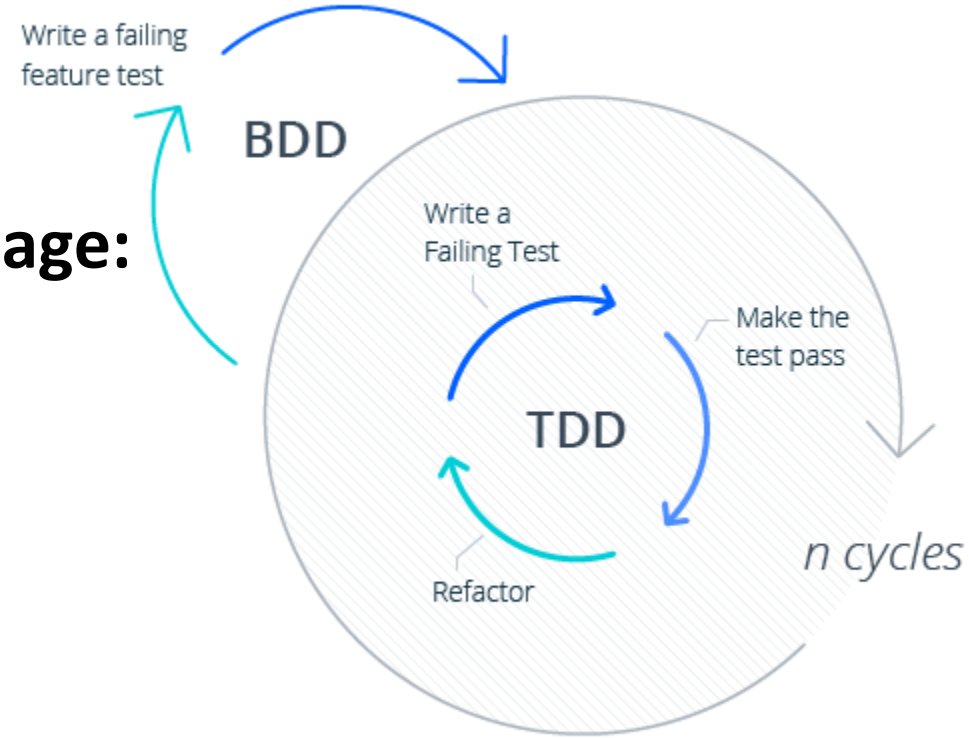
Starting by the test means starting by the requirements!!!

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.**

```
Scenario Outline: Newline before Examples
    Squashed together steps with no newlines to seperate them
    makes it more difficult to discern which information belongs together
    especially if tables are involved

    Given I add a new person

        And this person has the birthdate '<birthdate>'

    When I try to save this person

    Then I receive the error message for 'invalid birthdate'

    Examples:
        | birthdate  |
        | 01.01.1800 |
```

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. Below – **Scenario/s**
3. The last three lines starting with **Given, When, and Then** are the steps of our scenario.

Feature:

Scenario:
Given
When
Then

Cucumber Feature = Test Scenario

Cucumber Scenario = Test Case

Requirements based testing: Gherkin

What software will look like to user

Things that the user will do

What the user should expect

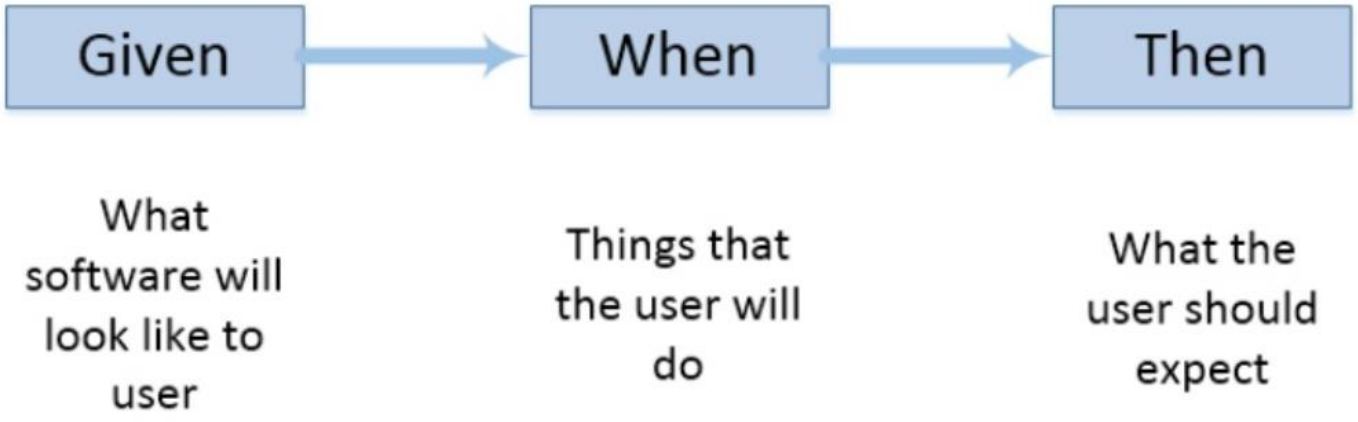
Feature: login to the system.

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.**



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

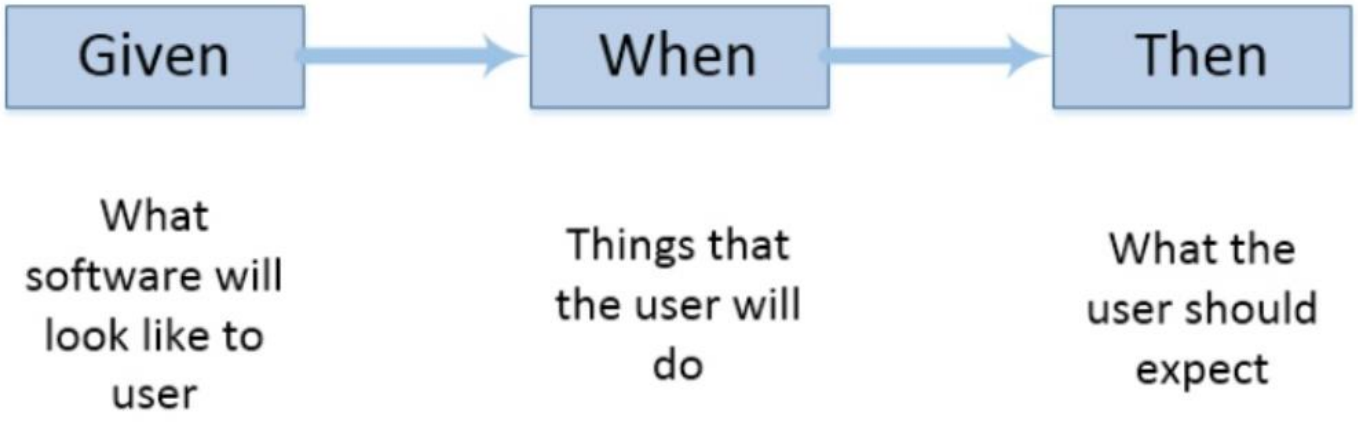
Requirements based testing: Gherkin

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

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).**



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

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/

Gherkin: example

Feature: Login to the System
As a user, I want to be able to log in to the system, So that I can access my account and perform various actions

Scenario Outline: Successful Login
Given the user is on the login page
When the user enters their valid username "<username>" and password "<password>"
And the user clicks the "Login" button
Then the user should be redirected to the dashboard page
And the user should see a welcome message "Welcome, <full_name>!"

Examples:

username	password	full_name
john_doe	Password123	John Doe
jane_smith	Passw0rd456	Jane Smith

Gherkin: example

Feature: Login to the System
As a user I want to be able to log in to the system So that I can access my account and perform various actions

Scenario Outline: Unsuccessful Login (Invalid Username)
Given the user is on the login page
When the user enters an invalid username "<username>" and valid password "<password>"
And the user clicks the "Login" button
Then the user should see an error message "Username not found, please check your username"
And the user should remain on the login page

Examples:

username	password
invalid_user	Password123
unknown_user	Passw0rd456

Gherkin: example

Feature: Login to the System
As a user I want to be able to log in to the system So that I can access my account and perform various actions

Scenario Outline: Unsuccessful Login (Invalid Password)
Given the user is on the login page
When the user enters their valid username "<username>" and invalid password "<password>"
And the user clicks the "Login" button
Then the user should see an error message "Invalid password, please try again"
And the user should remain on the login page

Examples:

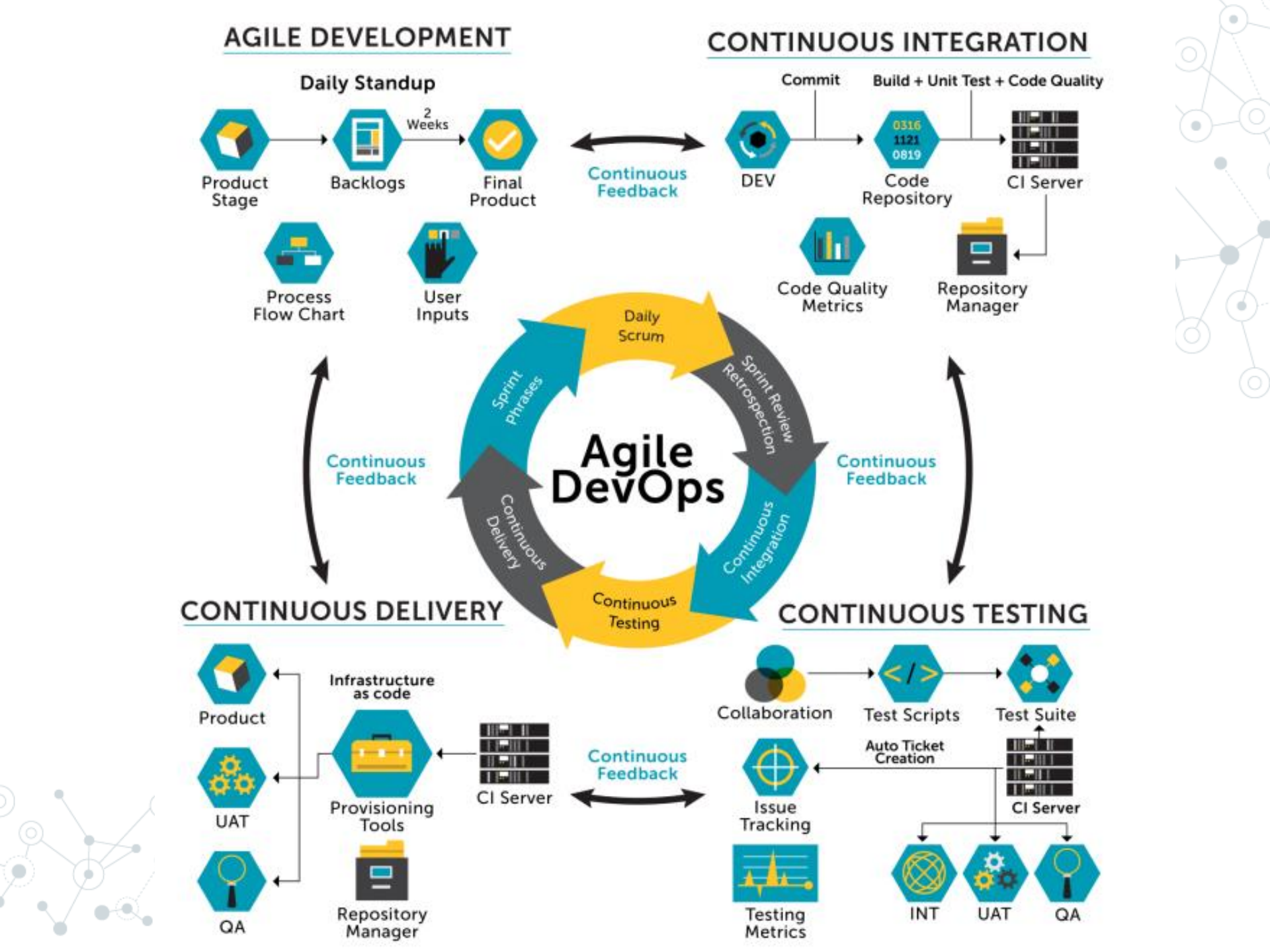
username	password
john_doe	InvalidPass
jane_smith	WrongPass

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’.

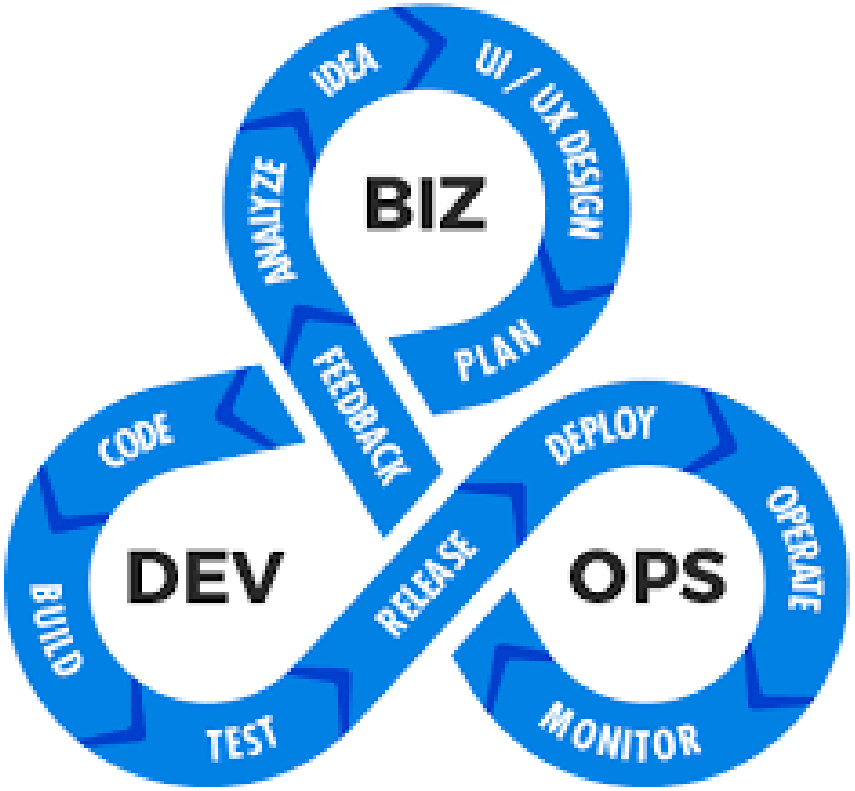
Disadvantages of BDD

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

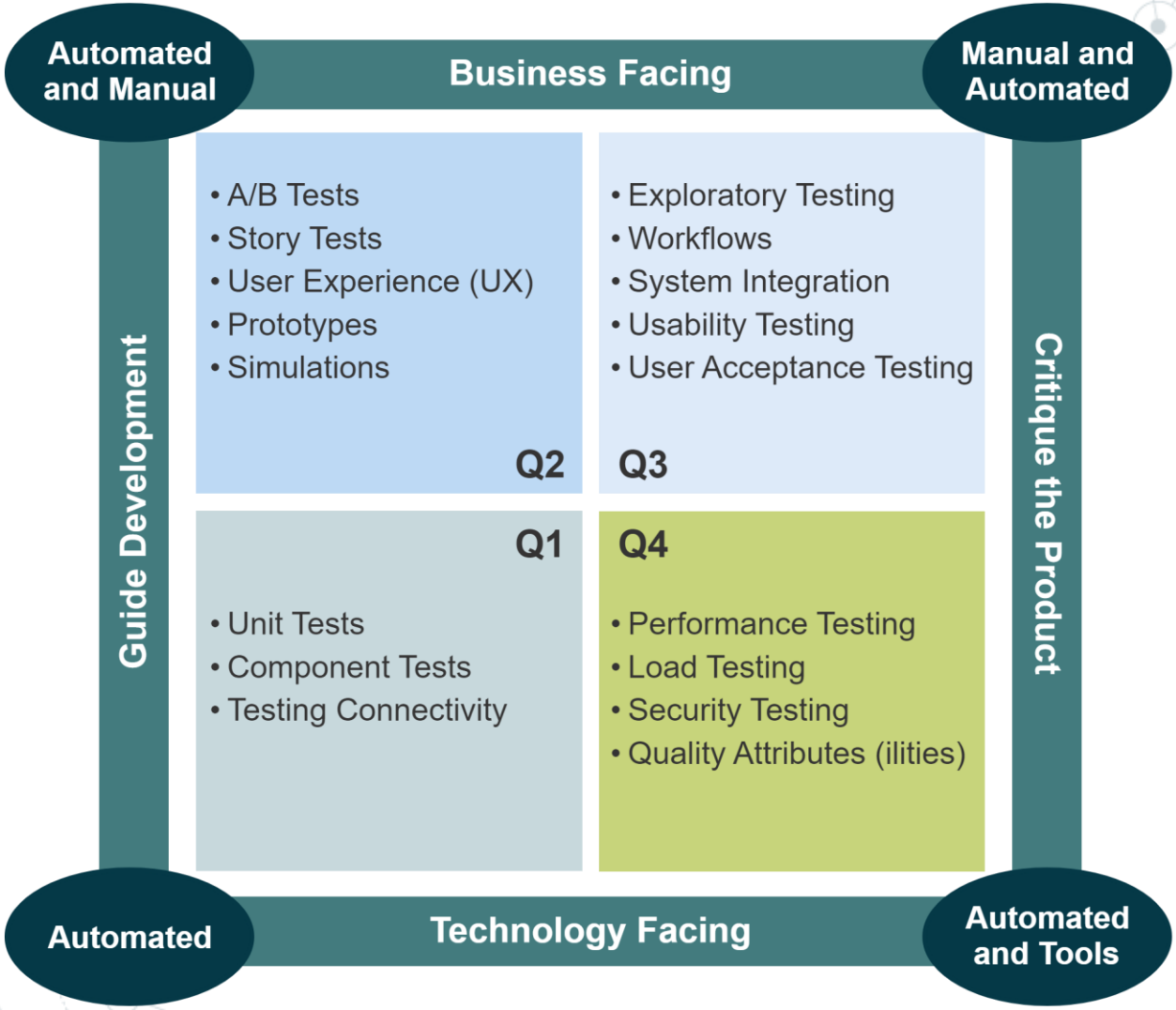

AGILE DEVELOPMENT
Daily Standup
Product Stage
Backlogs
2 Weeks
Final Product
Process Flow Chart
User Inputs
CONTINUOUS INTEGRATION
Commit
Build + Unit Test + Code Quality
DEV
0316
1121
0819
Code Repository
CI Server
Code Quality Metrics
Repository Manager
Continuous Feedback
Continuous Feedback
Continuous Feedback
Agile DevOps
Daily Scrum
Sprint Review Retrospection
Continuous Integration
Continuous Testing
Continuous Delivery
Sprint phrases
CONTINUOUS DELIVERY
Product
UAT
QA
Infrastructure as code
Provisioning Tools
Repository Manager
CI Server
Continuous Feedback
CONTINUOUS TESTING
Collaboration
Test Scripts
Test Suite
Auto Ticket Creation
Issue Tracking
Testing Metrics
CI Server
INT
UAT
QA

Continues testing

- **Continuous Testing** is a software testing type **that involves testing the software at every stage** of the software development life cycle



Agile Continuous Testing

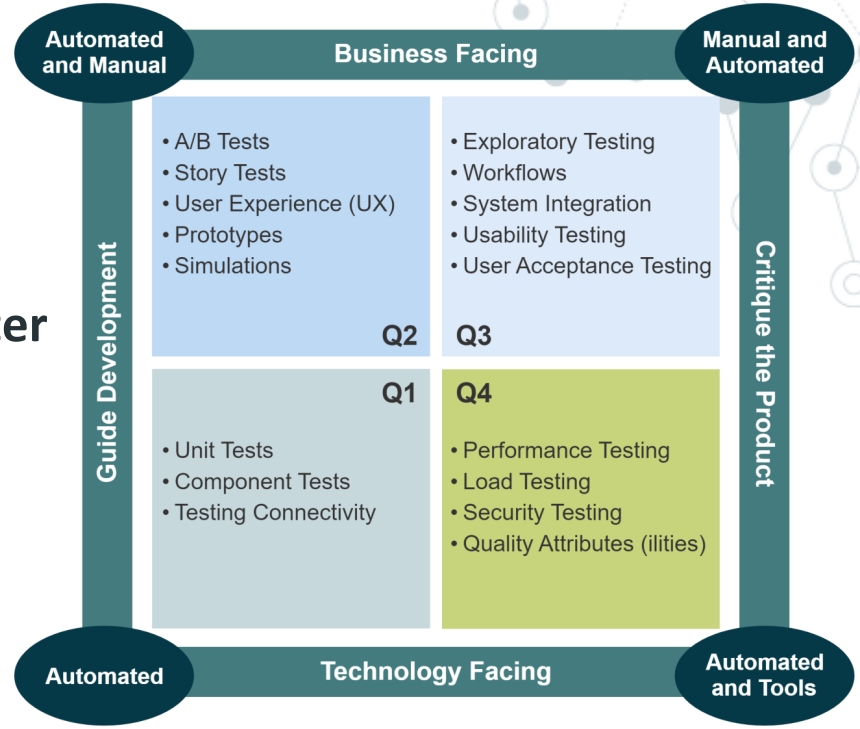


© Scaled Agile, Inc.

Agile Continuous Testing

Q1 – contains **unit and component tests.**

- **To confirm that the system works as agreed**
- **Tests are written to run before and after code changes.**
- In software, this is largely the home of TestDriven Development (TDD).

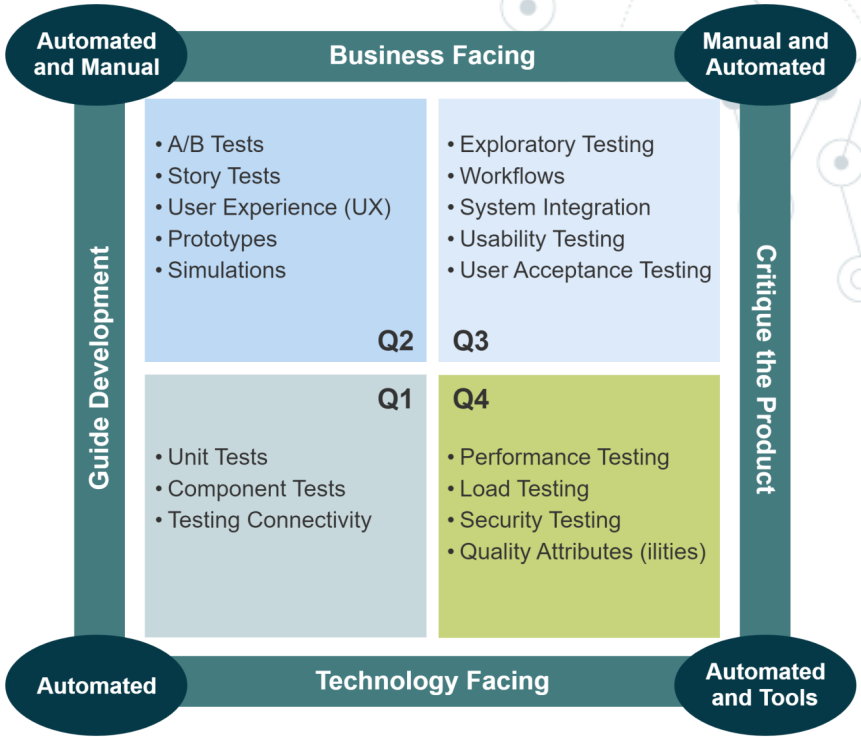


© Scaled Agile, Inc.

Agile Continuous Testing

Q2 – contains **functional tests**:

- **user acceptance tests** for stories, features, and capabilities, to validate that they work the way the Product Owner (or Customer/user) intended.
- **Feature-level and capability-level acceptance tests confirm the aggregate behavior of many user stories.**

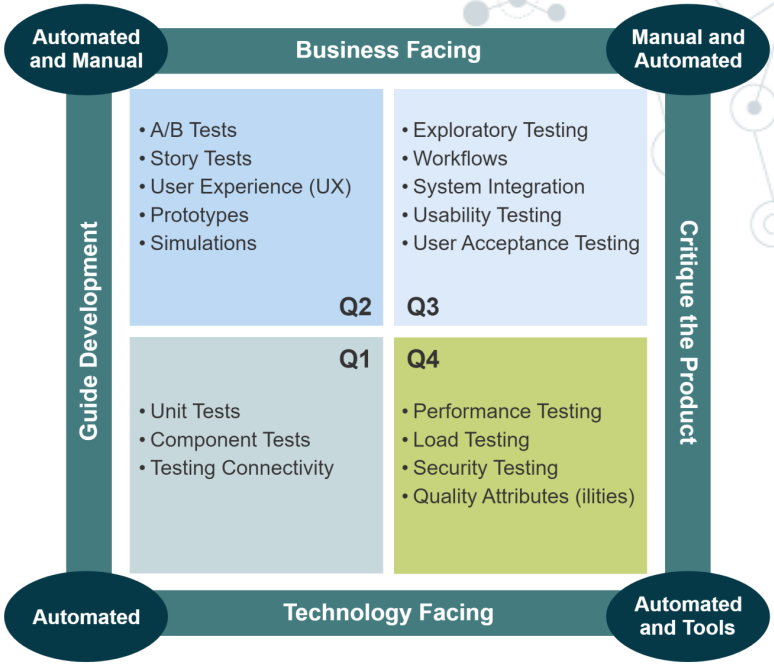


© Scaled Agile, Inc.

Agile Continuous Testing

Q3 – contains **System-level acceptance tests**

- Contains system-level acceptance tests **to validate that the behavior of the whole system meets usability and functionality requirements**, including scenarios that are often encountered during system use.
  - They involve users and testers engaged in real or simulated deployment scenarios, these tests are often manual.
- They're frequently **the final system validation** before delivery of the system to the end-user.

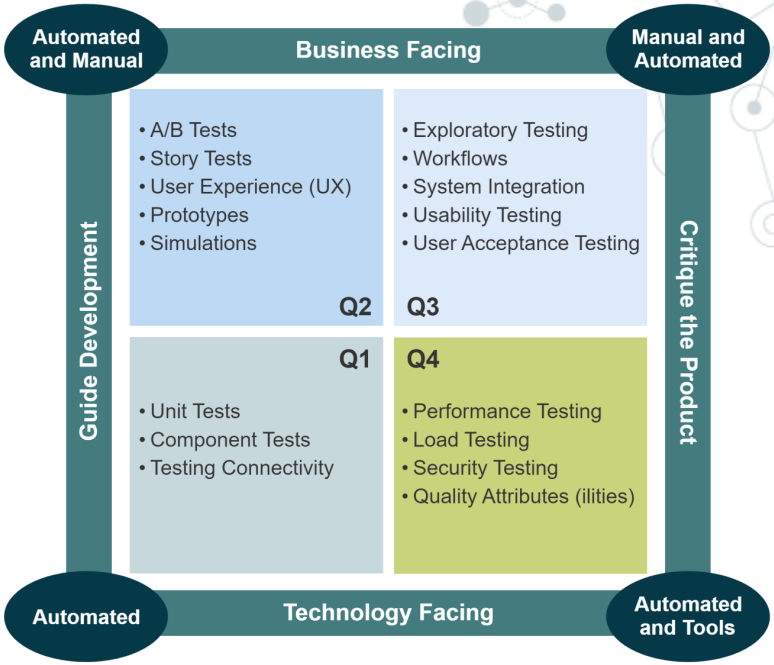


© Scaled Agile, Inc.

Agile Continuous Testing

Q4 contains **systems qualities test**

- Contains system qualities testing to verify **the system meets its Nonfunctional Requirements** (NFRs),
  - Typically, they're supported by a suite of **automated testing tools** (such as load and performance) designed specifically for this purpose.
- Since any system changes can violate conformance with NFRs, they must be run continuously, or at least whenever it's practical.



© Scaled Agile, Inc.

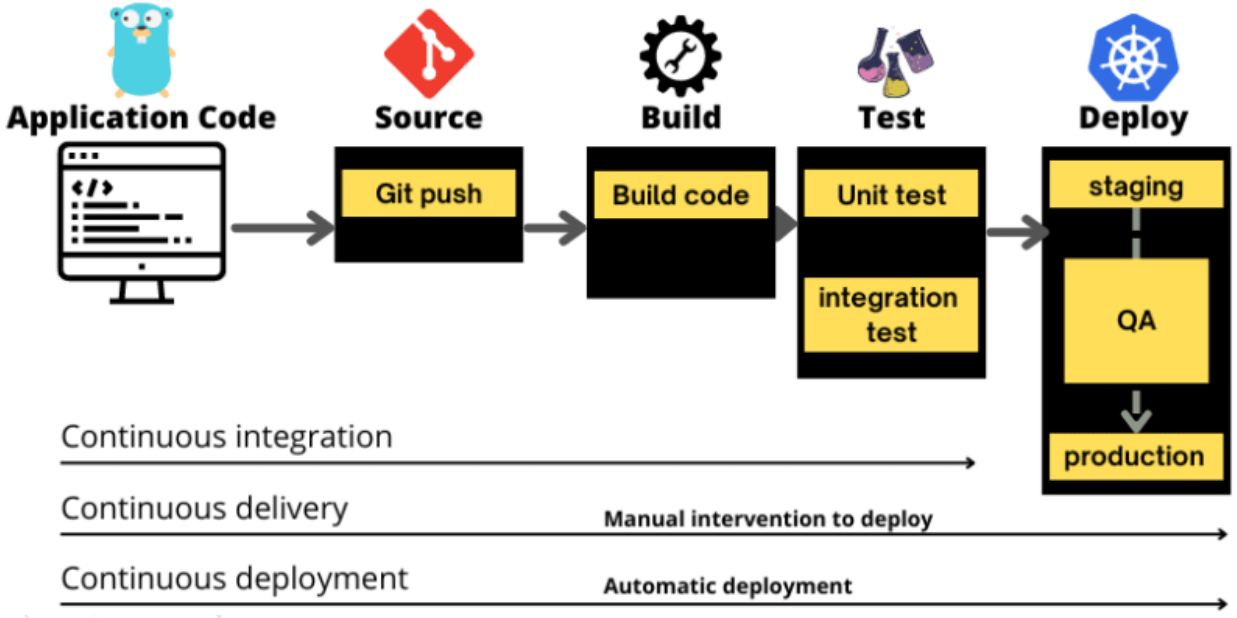
https://scaledagileframework.com/agile-testing/

Agile testing: **Manual** vs Automated

- **Exploratory Testing Manual System Testing** is a popular agile approach to testing and combines the investigation of the system that is to be tested with the design and performance of manual tests
  - **The tester does not require a detailed specification for the test object or the test**. The test is created while it is being performed and concentrates on suspect or defective components.
  - At the start of a session, the tester defines only the objectives of the test (which feature or User Story is to be tested).
  - **The tester tries to execute the user story or feature concerned and observes the system's behavior.**
  - This approach is perfect for taking a quick look at new or unknown features.
  - **The quality of an exploratory test depends heavily on the tester's degree of discipline, level of experience,** and feel for the software.
  - **Tests are difficult or impossible to reproduce** and have to be performed manually.
  - **Can not be managed or measured**

Agile testing: Manual vs **Automated**

- In Scrum Projects**, unit and integration tests are usually performed automatically**.
  - If these test scripts are embedded in the CI environment, they are automatically run every time the code is altered.



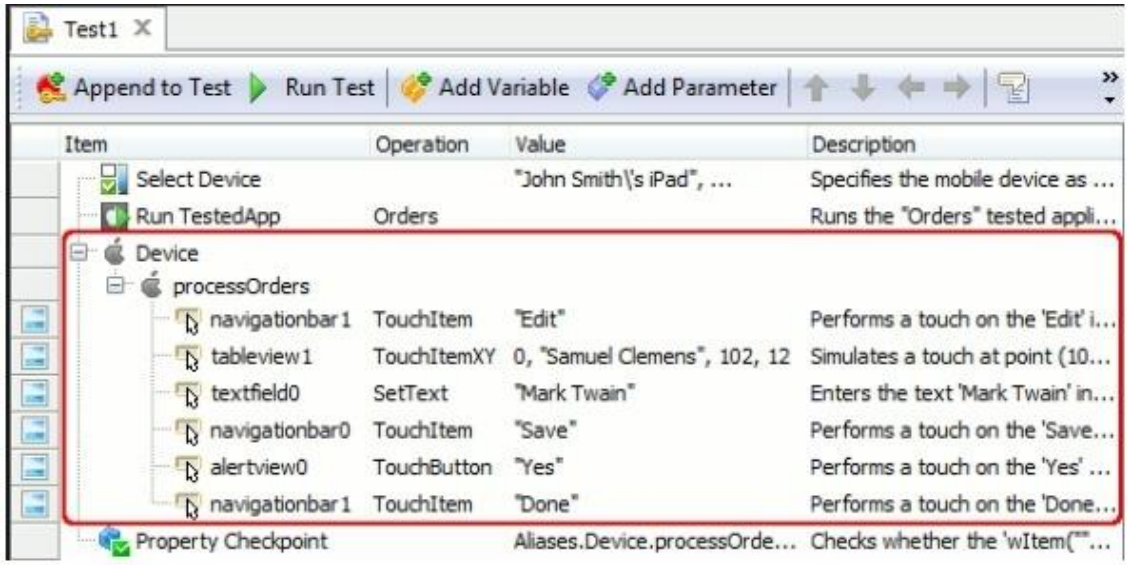
Agile testing: **Manual vs Automated**: System Testing

- To automate system tests is harder, because:
  - *The most important system test interface is the product's GUI,* which requires the use of **dedicated GUI test tools** or external systems.
  - **Test results often have to be analyzed manually** because it is too difficult to automatically compare expected behavior with actual behavior.
  - Some system test cases might require manual intervention.

Agile testing: Manual vs **Automated**

Record/playback tools record sequences of commands

- **A record/playback tool** records:
  1. all manually entered keyboard and mouse-driven commands
  2. the tester performs during a test session
  3. saves them in the form of a script.
- Running the script reproduces the recorded test sequence—an action that can be repeated as often as necessary.



Agile testing: Manual vs Automated

Record/playback tools record sequences of commands

- However**, the GUI is an element of the product that often changes** significantly during the course of the project's Sprints, so the corresponding tests have to be constantly altered to fit the changes.
- Generally, it is preferable to spend time during a Sprint writing new tests for new product functions rather than constantly maintaining and updating existing tests.

Data-driven scripting vs Keyword –driven scripting

Example: Simple Login Form

Test with different combinations of username and password

Problem: Necessary to write three scripts for three different combinations?

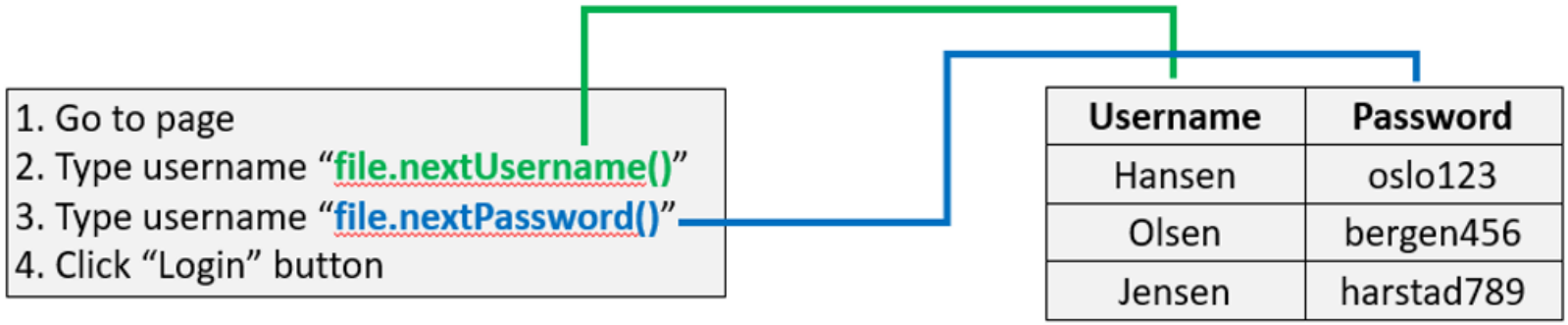
1. Go to login page 2. Type username “Hansen” 3. Type password“oslo123” 4. Click “Login” button	1. Go to login page 2. Type username “Olsen” 3. Type password“bergen456” 4. Click “Login” button	1. Go to login page 2. Type username “Jensen” 3. Type password“harstad789” 4. Click “Login” button

Data-driven scripting vs Keywrod – driven scripting

This test approach is time-consuming

Solution: Separate test script from data (username, password) → No hard-coding

One script retrieves different combinations of username and password



Data-driven scripting vs **Keywrod – driven scripting**

Keyword-driven scripting

Keywords symbolising actions (functionality)

"One level up" from data-driven scripting

Can write tests using keywords

"What to test, rather than how to test it"

Keyword	Script
Login	script1
CH_password	script2
Logout	script3

[script1]
1. Go to page
2. Type username "file.nextUsername()"
3. Type username "file.nextPassword()"
4. Click "Login" button

[script2]
1. Click on user avatar
2. Click "Change password"
3. Type current password
4. Type new password
5. Click "Confirm" button

[script3]
1. Click on user avatar
2. Click "Logout" button

Agile testing: Manual vs **Automated**

Regression Testing is testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made.

It is performed when the software or its environment is changed. [ISTQB]