# Integration Testing

Interactions Between Software System Modules

SoftUni Team

Technical Trainers

Software University

https://softuni.bg/

# Have a Question?

Software University

[sli.do](sli.do)

# #QA-Auto-Backend

# Table of Contents

1. Introduction to Integration Testing
2. Importance of Integration Testing
3. Types of Integration Testing
   - Approaches and strategies
4. Planning the Integration Testing
5. Writing Integration Tests - Exercise

# Introduction to Integration Testing

## Definition and Types

# What is Integration Testing?

- **Integration Testing** is defined as a type of testing where **modules are integrated logically** and **tested as a group**
- A typical software project consists of **multiple software modules**, created by **different programmers**
- The **purpose** of this testing is to **expose errors** during the **interaction between** these **software modules**

# Significance in Software Development

- **Integration Testing** ensures that **different modules** or **services** used by an application **work well together**

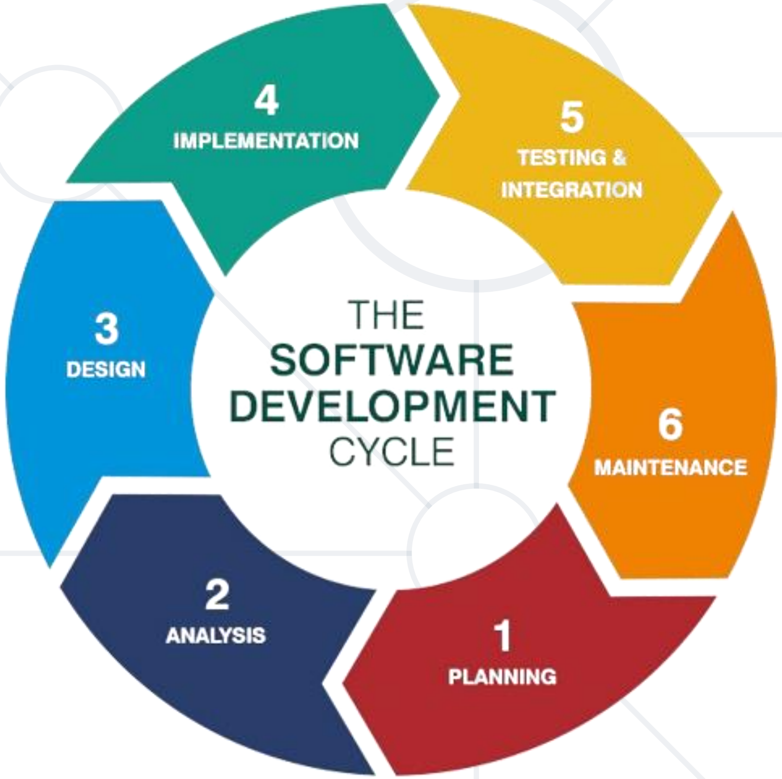
- Also termed as **"I & T" (Integration and Testing)**, it focuses on **checking data communication amongst these modules**

- **Integration Testing** is also known as **String Testing** or **Thread Testing**

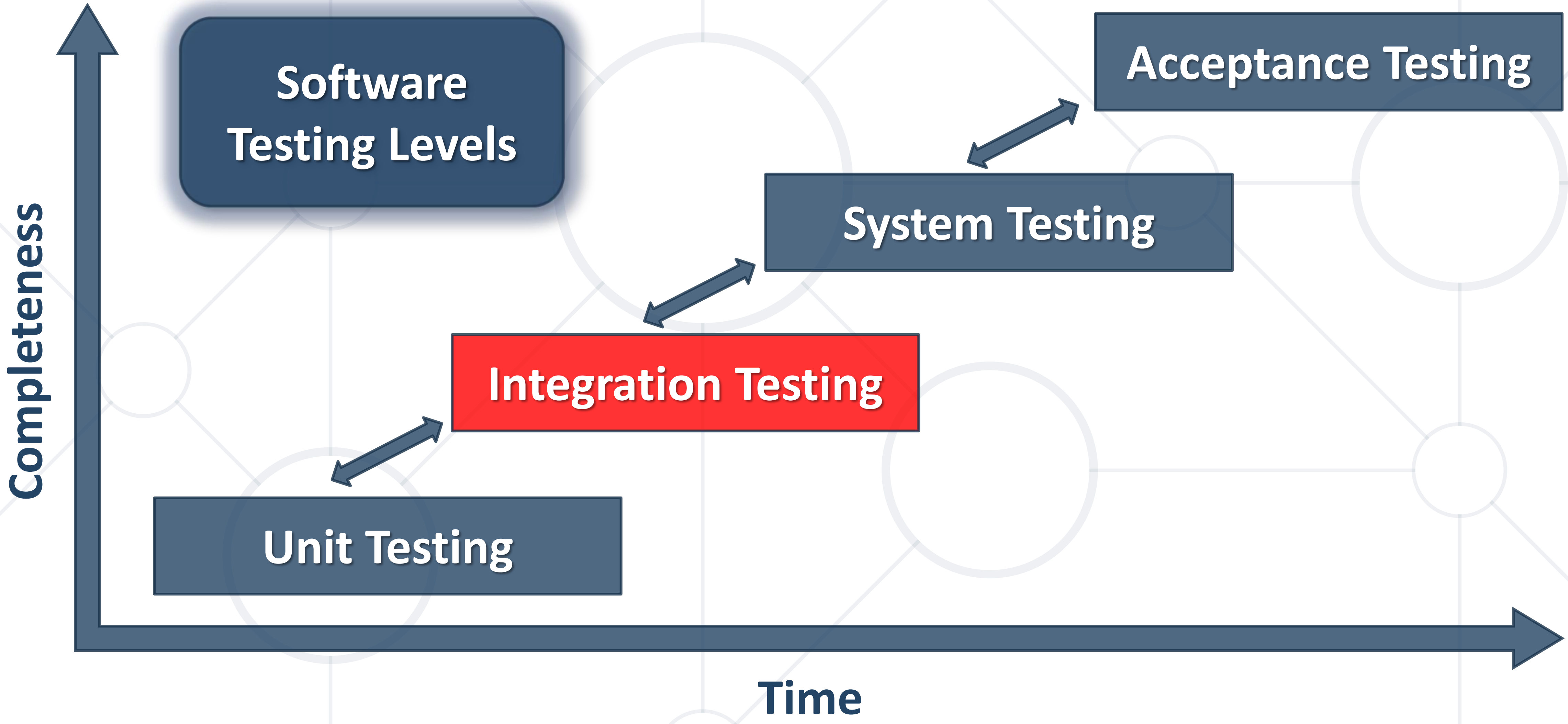
- **Integration Testing** is typically performed **after unit testing** and **before system testing**

# Software Development Lifecycle

- **SDLC** or **Software Development Life Cycle**
  - A process that produces software with the **highest quality** and **lowest cost** in the **shortest time** possible
  - Provides a **well-structured flow** of phases that help an organization to quickly produce **high-quality software** which is **well-tested** and ready for production use



# Integration Testing in SDLC



# Importance of Integration Testing

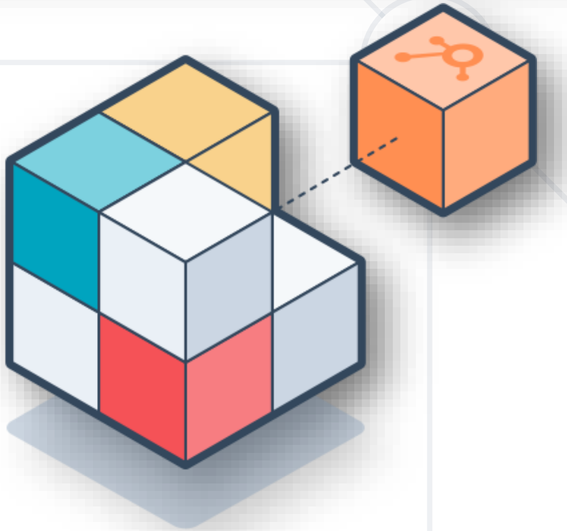
## Evaluating the Compliance of a System

# Necessity of Integration Testing

- Although each software module **is unit tested**, **defects still exist** for various reasons like:
  - A module is **designed by an individual software developer**
  - At the time of module development, there are chances of **change in requirements**
  - **Interfaces** of the software modules could be **improper** and/or **faulty**
  - **External Hardware** interfaces, if any, could be erroneous
  - Incomplete or inadequate **Exception Handling**

# Units and Developers

- Every separate software developer has **its own understanding and programming logic**, that may differ from other programmers'
- Integration Testing becomes necessary to **verify the software modules work in unity**



# Dynamic Business Environment

- **Fast-paced industries** and **market demands** can lead to changes in software requirements, even **in the middle of the development cycle**
- **Stakeholders** might no fully understand their needs at the outset or may gain new insights as they see the **project evolve**

# Common Interface Issues

- **Data type** mismatches
- Incorrect handling of **database connections**
- Faulty **error handling** and exception swallowing
- Inadequate **transaction** management

# Types of Integration Testing

Execution Strategies

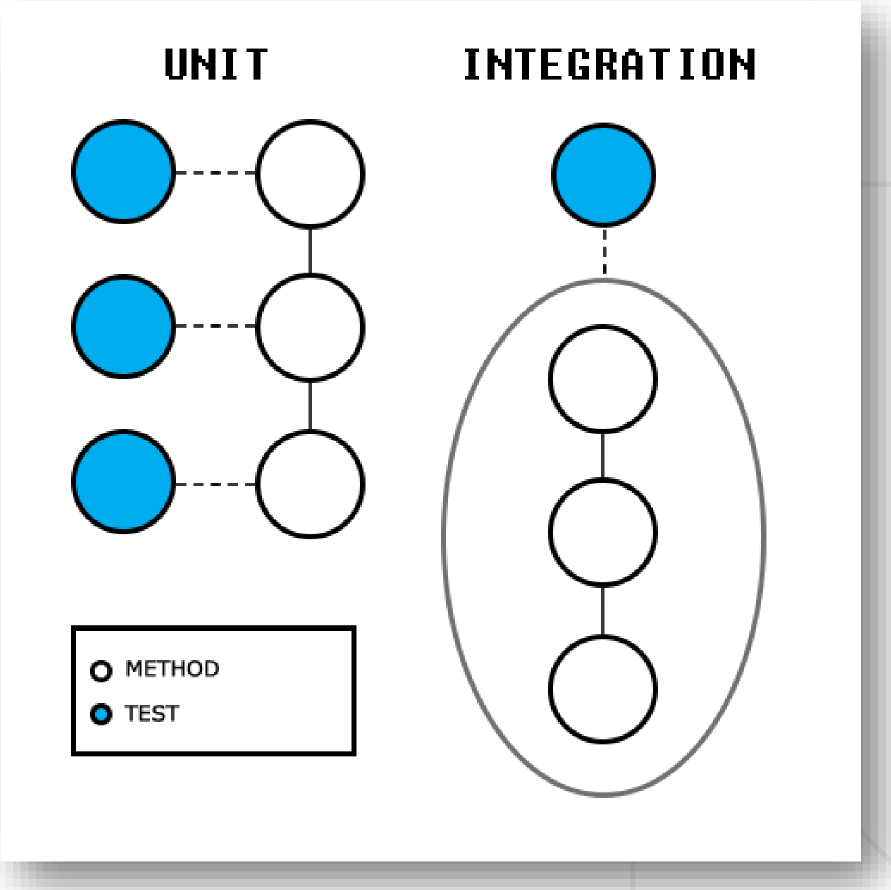
# Unit vs. Integration Testing

- In unit testing, **each module** of the software is **tested separately**
- Tester **knows the internal design** of the software
- Unit testing is performed **first of all testing processes**

- In integration testing, **all modules** of the software are **tested combined**
- Integration testing **doesn't know the  internal design** of the software
- Performed **after unit testing** and **before system testing**

# Example of Integration Test Case

- **Integration Test Case** differs from other test cases in the sense it **focuses mainly on the interfaces and flow of data** between the modules
  - Priority is to be given for the **integrating links** rather than the unit functions which are already tested



# Example of Integration Test Case

- Simple Integration Test Cases for **the following scenario**:
  - Application has **3 modules** – **Login Page**, **Mailbox**, **Delete Mails**
  - **Each of them is integrated logically**
- Do not concentrate much on the **Login Page** testing as it's already been done in Unit Testing
  - **Check how it's linked to the Mail Box page**
- Similarly for **Mail Box** – **Check its integration** to the **Delete Mails** module

# Example of Integration Test Case

- Agree on a test case strategy to prepare and execute test cases in conjunction with the test data:

| TC ID | Test Case Objective | Test Case Description | Expected Result |
|---|---|---|---|
| **1** | Check the interface link between the Login and Mailbox module | Enter login credentials and click in the Login button | To be directed to the Mailbox |
| **2** | Check the interface link between the Mailbox and Delete Mails module | From Mailbox select the email and click a delete button | Selected email should appear in the Deleted/Trash folder |

# Types of Integration Testing

- Software Engineering defines **variety of strategies to execute** Integration testing.
  - **Big Bang** Approach
  - **Incremental** Approach:
    - **Top Down** Approach
    - **Bottom Up** Approach
    - **Sandwich Approach**
      - **Combination** of Top Down and Bottom Up

# Big Bang Testing

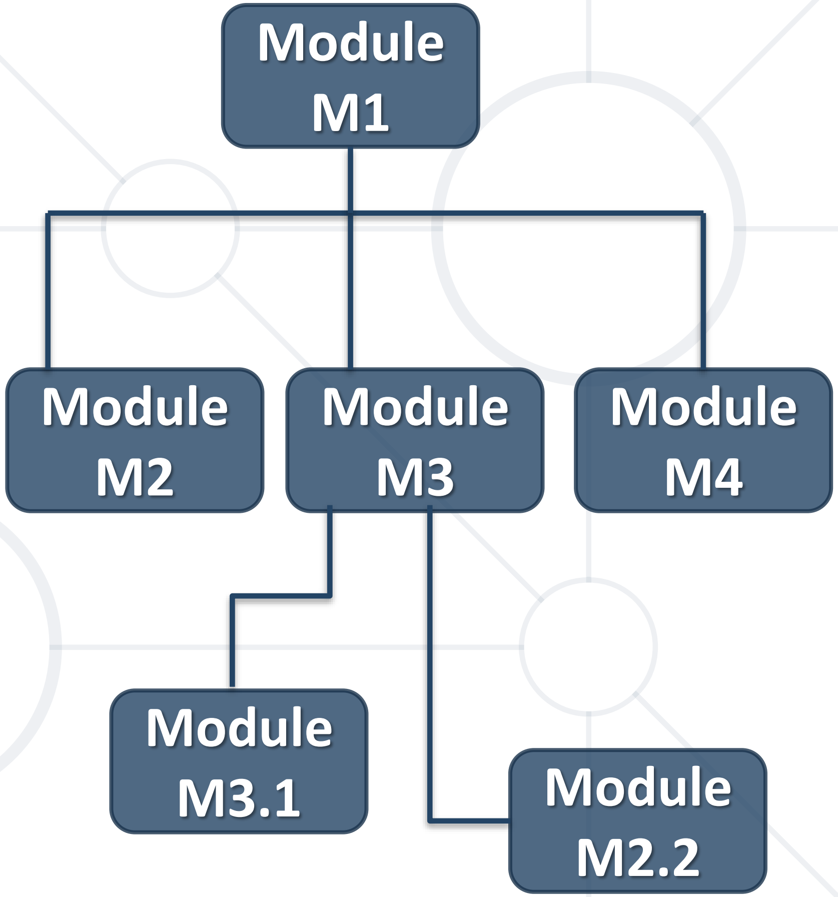
- **Big Bang Testing** is an Integration testing approach in which all the components or **modules are integrated together at once** and then **tested as a unit**
- This combined **set of components is considered as an entity** while testing
- If all of the **components in the unit are not completed**, the **integration process will not execute**

# Big Bang Testing

- Advantages – Convenient for **small systems**
- Disadvantages:
  - Fault Localization is **difficult**
  - Some interfaces link to be tested **could be missed easily**
  - Since the Integration testing can commence only after "all" the modules are designed, the **testing team will have less time** for execution in the testing phase
  - Since all modules are tested at once, **high-risk critical modules are not isolated and tested on priority**. Peripheral modules which deal with user **interfaces are also not isolated** and tested on priority

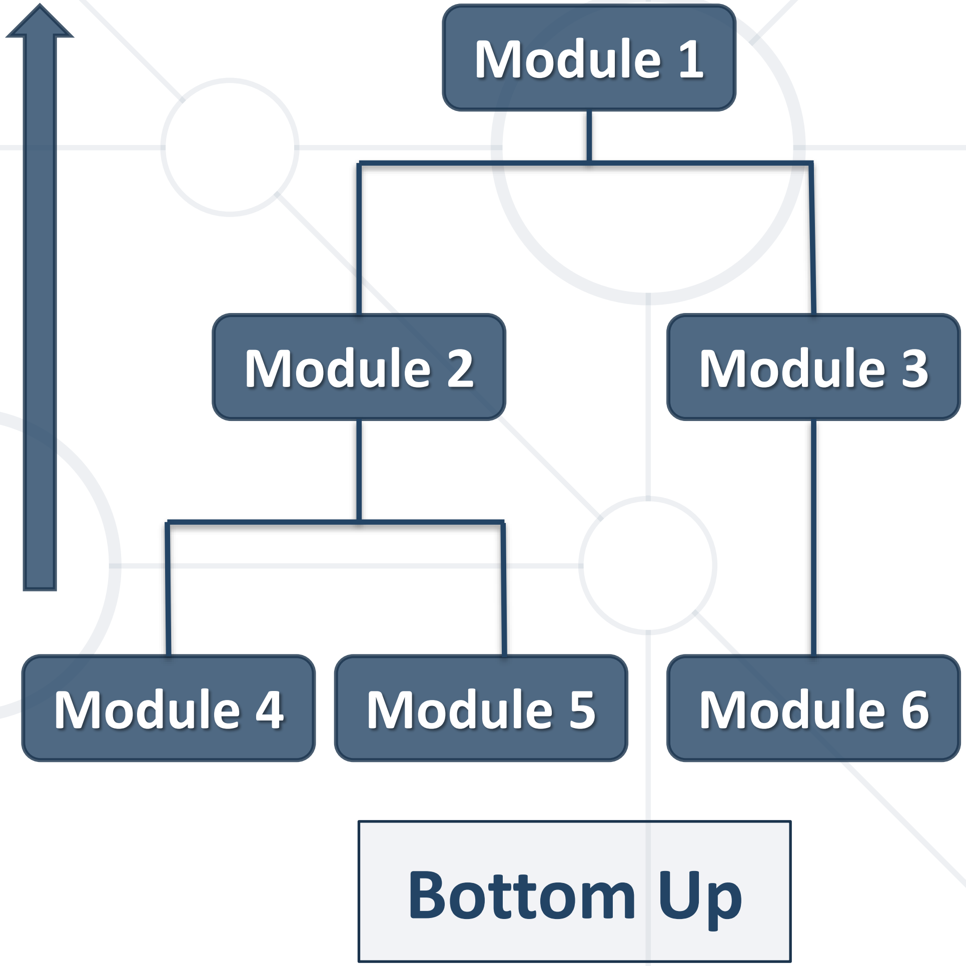
# Incremental Testing

- Testing is done by **integrating two or more modules** that are **logically related to each other** and then **tested for proper functioning of the application**
- The other modules are **integrated incrementally** and the **process continues until all the logically related modules are integrated and tested successfully**
- **Incremental Approach** is carried out by **two different Methods**:
  - Bottom Up
  - Top Down



# Bottom Up Integration Testing

- **Bottom Up Integration Testing**:
  - A strategy in which the **lower level modules are tested first**
  - These tested modules are further used to support the testing of higher level modules
  - The process continues **until all modules at top level are tested**
  - Once the lower level modules are tested and integrated, then the next level of modules are formed

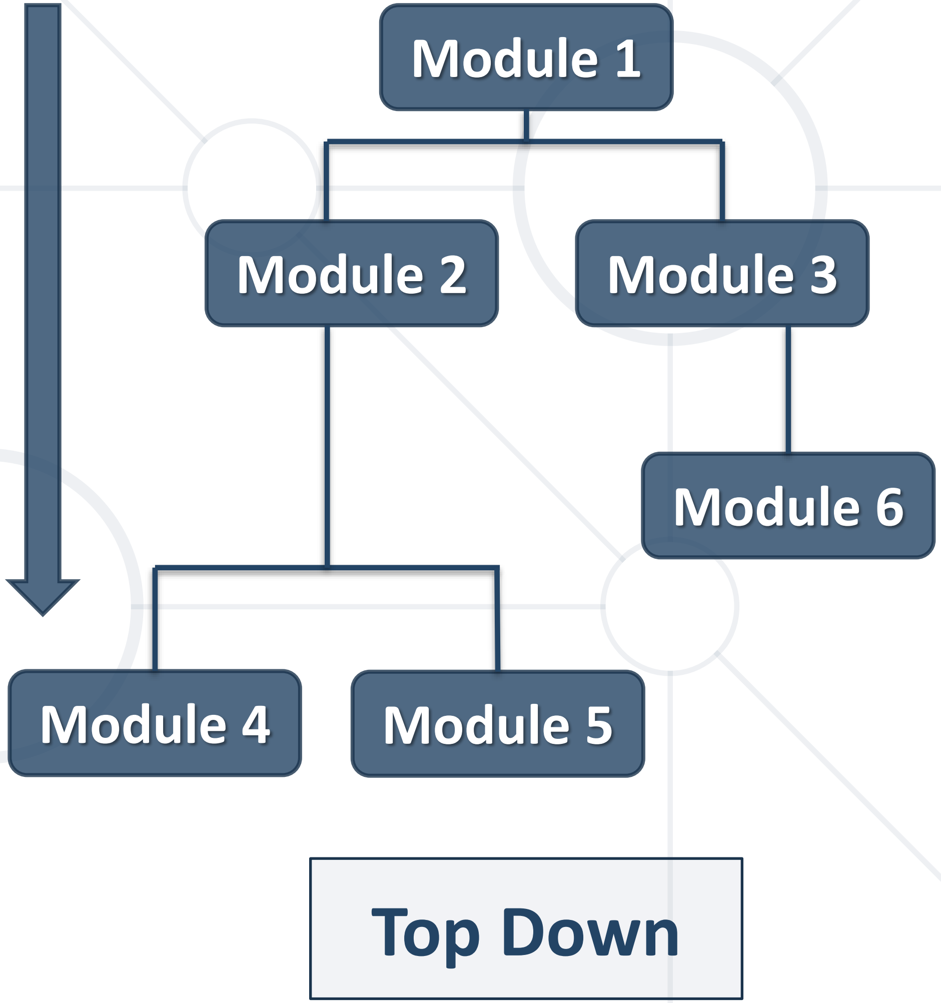


# Bottom Up Integration Testing

- **Advantages**:
  - **Fault localization** is easier.
  - No **time** is wasted waiting for all the modules to be developed
- **Disadvantages**:
  - **Critical modules** (at the top level of software architecture) which control the flow of application **are tested last**
  - An **early prototype** is not possible

# Top Down Integration Testing

- **Top Down Integration Testing**:
  - Integration Testing takes place **from top to bottom**, following the **control flow of software system**
  - The **higher level modules are tested first** and **then lower modules are tested and integrated** in order to check the software functionality



# Top Down Integration Testing

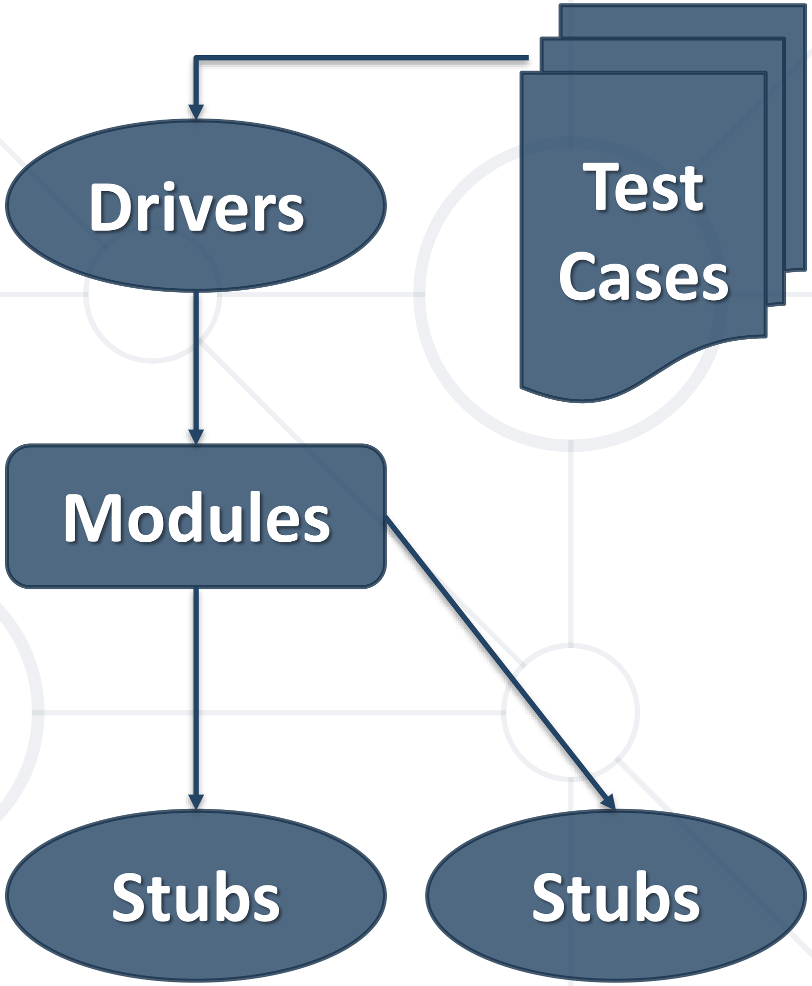
- **Advantages**:
  - It can help to **identify potential risks early on**
  - Possibility to **obtain an early prototype**
  - **Critical Modules** are **tested on priority**
- **Disadvantages**:
  - It can be **challenging to implement** for **large and complex systems**
  - **Modules at lower levels** are **tested inadequately**

# Sandwich Testing

- **Sandwich Testing**:
  - A strategy in which top level modules are tested with lower level modules **at the same time**
  - Lower modules are integrated with top modules and are **tested as a system**
  - It is a **combination of Top Down and Bottom Up approaches**
  - It is called **Hybrid Integration Testing**

# Stubs and Drivers

- These elements are **dummy programs** used in integration testing to facilitate software testing activity, **acting as substitutes for any missing models in the testing process**
- These programs **don't implement the missing software module's entire programming logic**, but they do simulate the **everyday data communication with the calling module**



# Planning the Integration Testing

Testing Procedure

# Define the Scope and Objectives

- The first step is to **define the scope**:
  - What components or modules are included in the testing
  - The boundaries and dependencies of the system
- The objectives define **what you want to achieve or verify** through the testing:
  - Functionality, compatibility, security, reliability
- Specify the **criteria for starting and ending the testing**, as well as **expected outcomes**

# Design the Integration Test Strategy

- The integration test strategy describes **the approach to the testing**:
  - Which **methods**, **tools** and **frameworks** will be used
  - Which **levels**, **types** and **phases** of integration testing will be performed
- Should be defined:
  - The **roles and responsibility** of the **testing team**
  - The **communication and reporting mechanisms**
  - The **risk management**

# Develop Test Cases and Scenarios

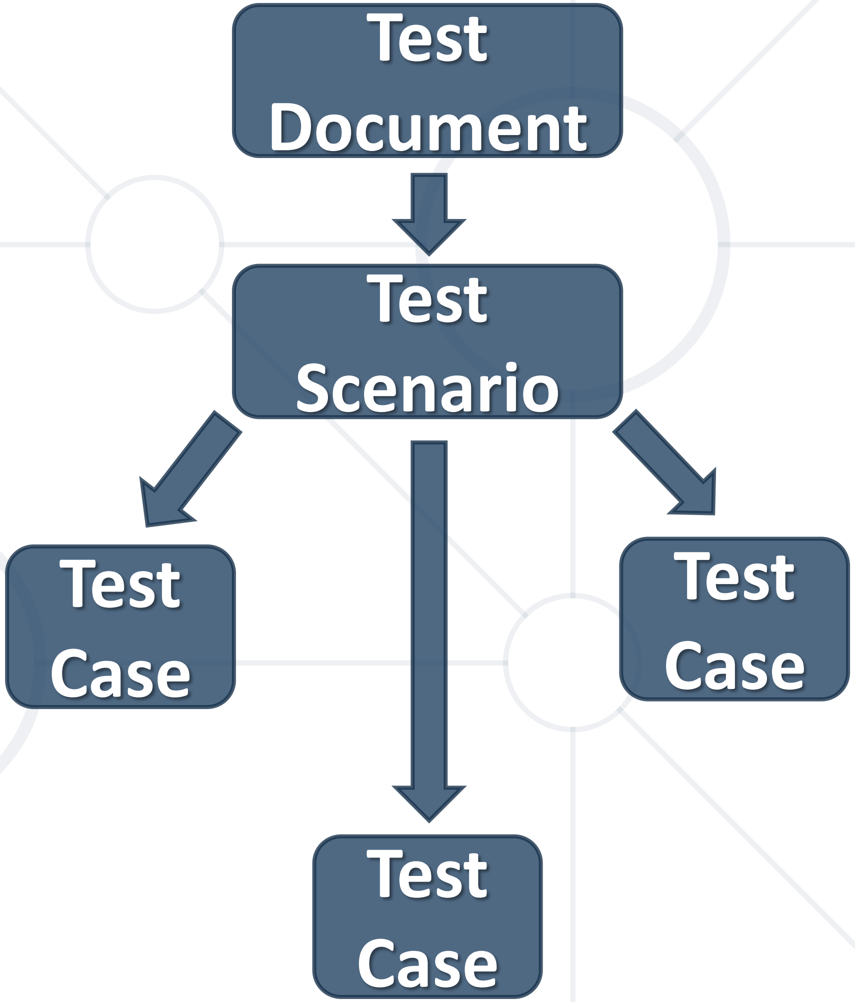
- The integration **test cases** and **scenarios** are the detailed specification of:
  - **What** will be tested?
  - **How** it will be tested?
  - **What is expected** as a result?
- Integration test cases and scenarios should be:
  - **Traceable**
  - **Reusable**
  - **Maintainable**

# Prepare Environment and Data

- The integration test **Environment** and **Data** are the resources and conditions that **will be used to execute the integration test cases and scenarios**
- Ensure that the integration test environment and data are **consistent**, **realistic**, and **representative** of the actual system and its users
- Also ensure the integration test environment and data are **properly configured**, **secured**, and **managed**

# Execute the Test Cases and Scenarios

- The **execution** of the integration test cases and scenarios is the actual process of **running the tests** and **verifying the results**
- Follow the **integration test strategy** and **schedule**, and use the integration test tools and frameworks to **automate**, **monitor**, and **record the test execution**
- Make sure the test execution is **reliable**, **efficient**, and **accurate**



# Writing Integration Tests - Exercise

Implementing Test Methods

# Download Skeleton and Extract Solution

# Test Project Configuration

- The test project for integration testing has been **created and configured**
- All needed **dependencies** and **NuGet packages** have been installed
- The project is prepared and **ready for the implementation** of test methods
- We are equipped to **start writing integration tests** to ensure the reliability of our C# console application

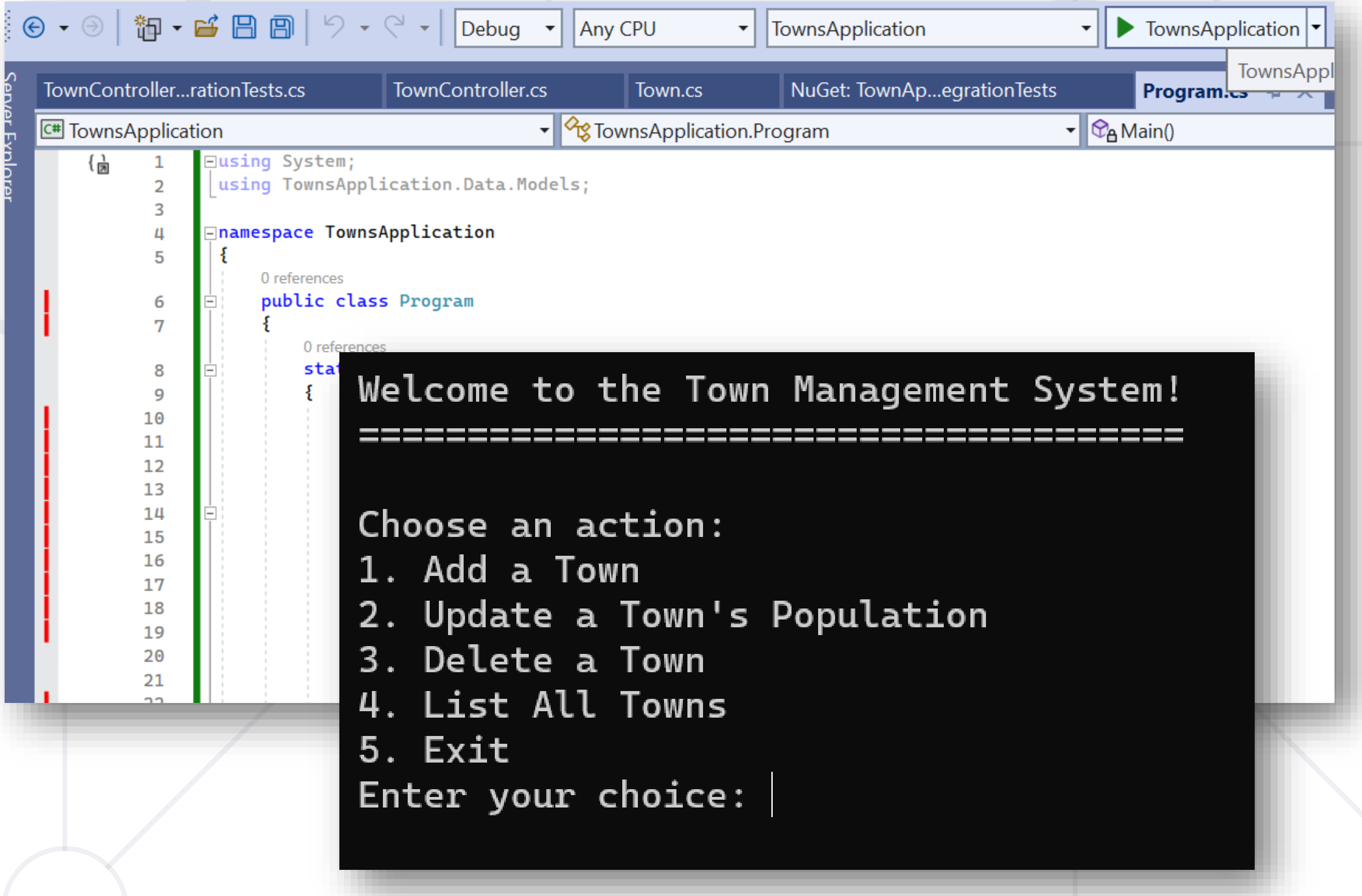
# The "AAA" Testing Pattern

- Automated tests usually follow the "**AAA**" **pattern**
  - **Arrange**: prepare the **input** data and entrance conditions
  - **Act**: invoke the **action** for testing
  - **Assert**: check the **output** and exit conditions

```
[Test]
public void Test_SumNumbers()
{
  // Arrange
  var nums = new int[] {3, 5};

  // Act
  var sum = Sum(nums);

  // Assert
  Assert.AreEqual(8, sum);
}
```

# Explore the Application First

- Fully Functional in Console Mode:
- Features to discover:
  - Add a Town
  - Update Population
  - Delete a Town
  - List All Towns
  - Exit



```
using System;
using TownsApplication.Data.Models;

namespace TownsApplication
{
    public class Program
    {
        stat
        {
```

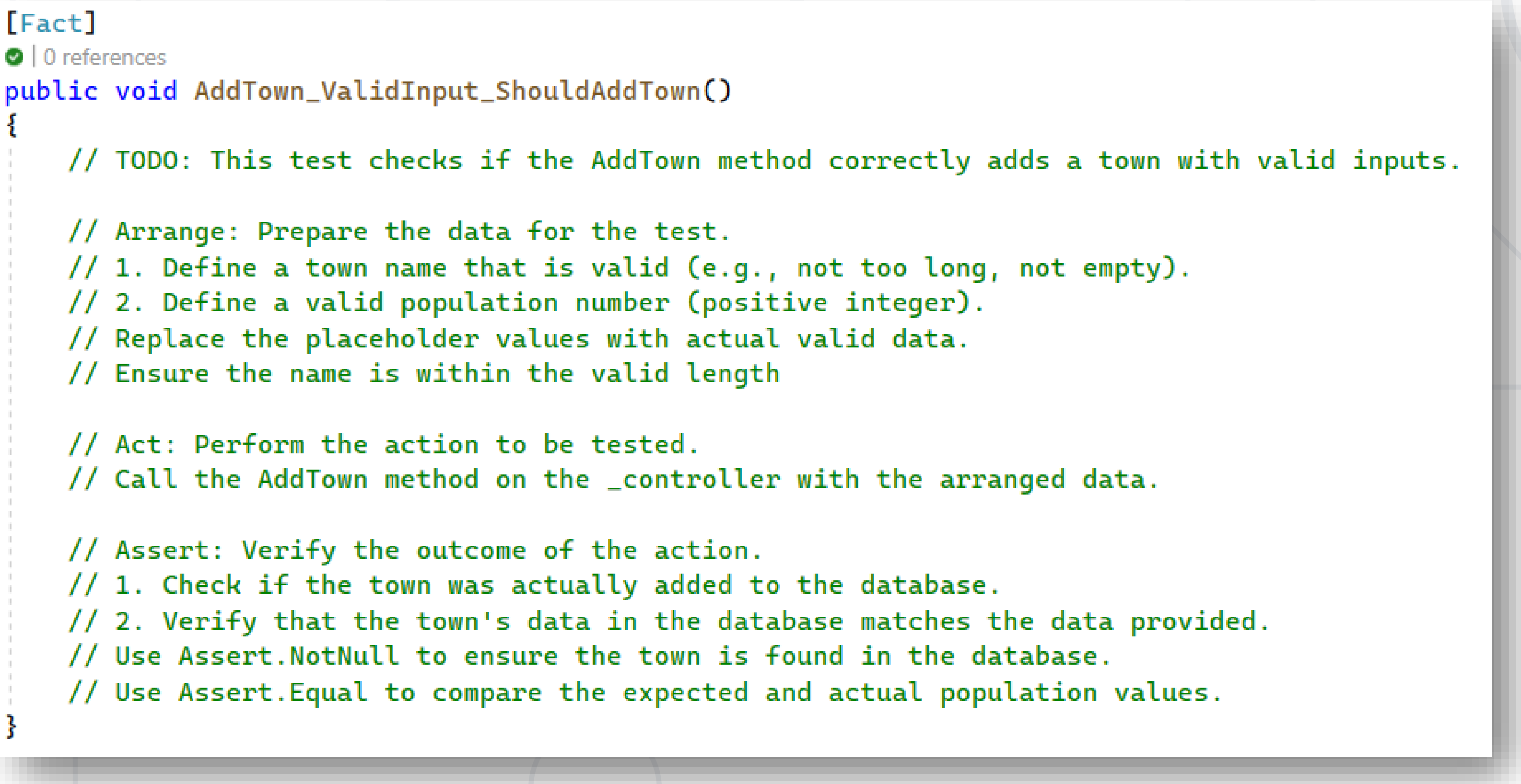
```
Welcome to the Town Management System!
======================================

Choose an action:
1. Add a Town
2. Update a Town's Population
3. Delete a Town
4. List All Towns
5. Exit
Enter your choice:
```

# xUnit.net: Overview

- **xUnit** == popular **C# testing framework**
  - Supports test suites, test cases, before & after code, startup & cleanup code, timeouts, expected errors, …
  - Like **JUnit** (for Java)
  - Free, open-source
  - Powerful and mature
  - Wide community
  - Built-in support in Visual Studio
  - Official site: xunit.net/

# Integration Tests Project

- The test project is called **TownApplication.IntegrationTests**
- All testing methods are empty and should be implemented:



```
[Fact]
✓ | 0 references
public void AddTown_ValidInput_ShouldAddTown()
{
    // TODO: This test checks if the AddTown method correctly adds a town with valid inputs.

    // Arrange: Prepare the data for the test.
    // 1. Define a town name that is valid (e.g., not too long, not empty).
    // 2. Define a valid population number (positive integer).
    // Replace the placeholder values with actual valid data.
    // Ensure the name is within the valid length

    // Act: Perform the action to be tested.
    // Call the AddTown method on the _controller with the arranged data.

    // Assert: Verify the outcome of the action.
    // 1. Check if the town was actually added to the database.
    // 2. Verify that the town's data in the database matches the data provided.
    // Use Assert.NotNull to ensure the town is found in the database.
    // Use Assert.Equal to compare the expected and actual population values.
}
```

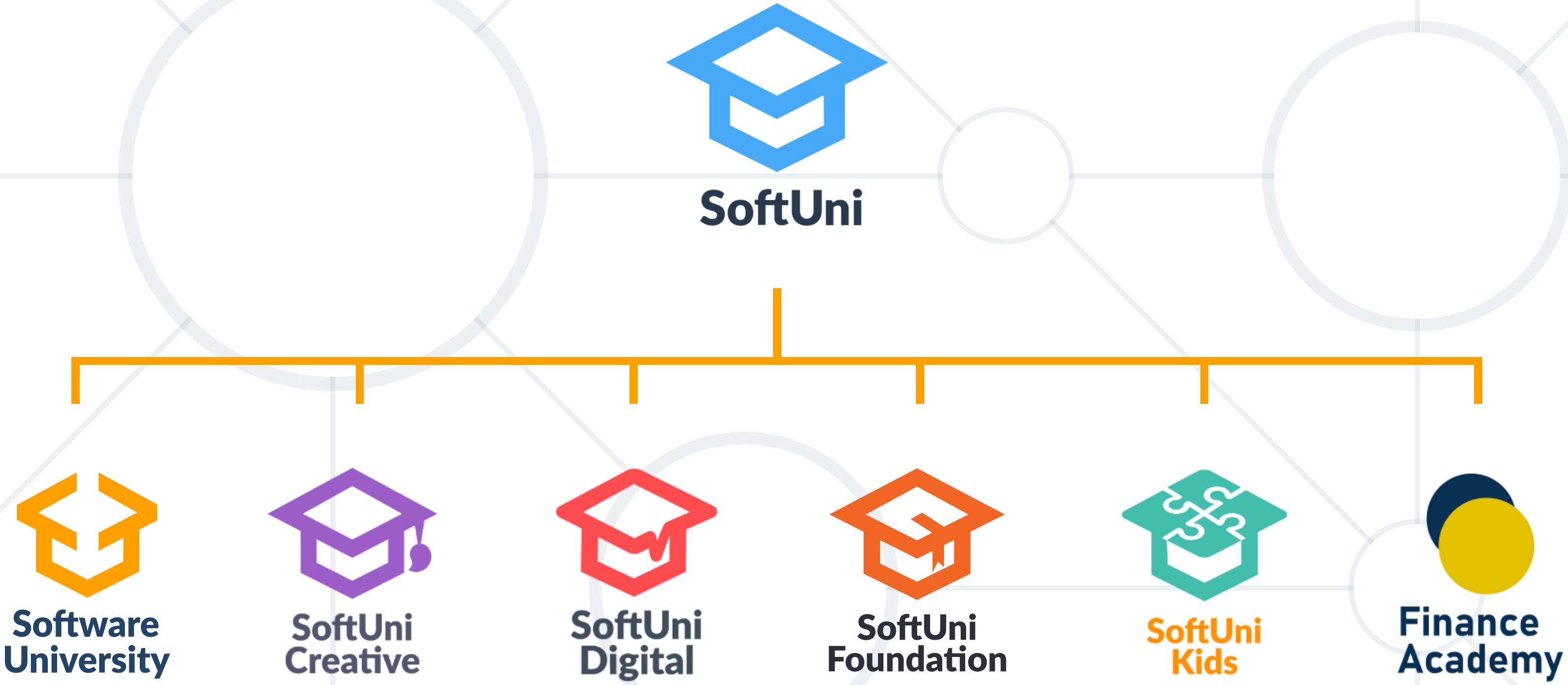
# Writing First Integration Test

```
// Arrange
string townName = "Rome"; // Should be within the valid length
int population = 2873545;

// Act
_controller.AddTown(townName, population);

// Assert
var townInDb = _controller.GetTownByName(townName);
Assert.NotNull(townInDb);
Assert.Equal(population, townInDb.Population);
```

# Summary

- Integration Testing Essentials
- Importance of Integration Testing
- Types, Approaches and Strategies
- Planning and Scenarios
- Implementing I&T Methods

# Questions?



SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

# SoftUni Diamond Partners

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank

Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

# License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg