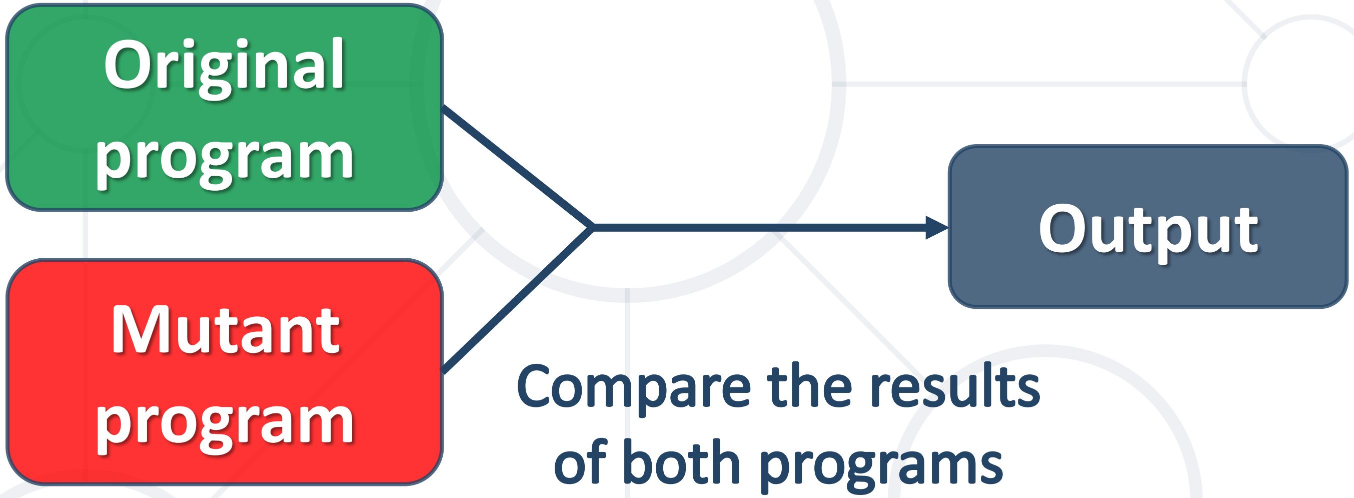
Mutation Testing

Evaluate the Quality of Existing Software Tests



Compare the results of both programs

SoftUni Team

Technical Trainers

Software University

Software University
https://softuni.bg

SoftUni

Table of Contents

1. Mutation Testing Introduction
2. Types of Mutation Testing
3. Tools for Mutation Testing
   - Stryker – Installation, Usings, Metrics
4. Mutation Testing Wrap-Up

Mutation Testing

Introduction

What is Mutation Testing?

- A **testing approach** in which **specific elements** of a software application's source **code** are **altered**
- Then, **tests are performed** to determine whether these **modifications lead** to **test failures**
- In simpler terms, **mutation testing** is making **minor changes** to the **code** and then **running unit tests** against the **modified code**, expecting the **tests to fail**
- If the **tests don't fail**, it suggests a necessity to **enhance the design of the tests** for better detection of potential issues

Brief History

- **Originates in the 1970s**:
  - Conceptualized by Richard Lipton as a **method to evaluate** the adequacy of **test data**
  - Devised as a **strategy** to **introduce faults intentionally** to test the effectiveness of test suites
- **Early Development**:
  - Further developed at Yale University
  - **Initially perceived** as **impractical** due to computational limitations at the time

Brief History

- **Advancements in the 1980s and 1990s**:
  - **Research** by Offutt and others expanded the **theoretical framework**
  - **Improved algorithms** and the advent of powerful computers made mutation testing **more feasible**
- **Growth in Academic Interest**:
  - Became a **popular topic** for **academic research**, leading to the development of various mutation operators and tools
  - **Studies highlighted** its **potential** for identifying subtle bugs not caught by other testing methods

Brief History

- **2000s - Tool Development**:
  - The creation of tools like Jester, PIT, and others for different programming languages
  - **Open-source projects** facilitated **community involvement** and **tool refinement**
- **Current State**:
  - **Modern tools** integrate with **CI pipelines**, supporting automated and periodic mutation testing
  - **Research continues** to **optimize performance**, reduce equivalent mutant generation, and improve result analysis

Software Development and Testing

- **Software apps** are becoming increasingly **complex**
- High demand for **rapid feature development**
- **Ensuring robustness** and **reliability** of software is critical
- **Minor bugs** can lead to **significant consequences** /financial losses, security breaches, and compromised user experiences/
- Traditional methods focus on **improving software quality**
- They often have **limited capacity** to uncover **every potential defect**
- There's a need for **more advanced testing approaches**

Why Mutation Testing?

- Mutation testing **addresses** the **deficiencies** of traditional methods
- Entails **minor modifications** to **specific elements** of a software application's source code, such as:
  - Altering lines of code
  - Changing true/false expressions
  - Modifying variable values
- These intentional **changes** are **minimal** and do not significantly alter the software's primary functionality

Effectiveness, Purpose and Application

- The **effectiveness** of a test suite is **measured** by its ability to "**kill mutants**"
- The **objective** is to evaluate the **robustness** and **thoroughness** of test cases
- It is a method predominantly **used** in **white box** testing, particularly within unit testing frameworks
- By **testing** the **mutated code** against the **original** unaltered code, the **quality** and **coverage** of testing can be assessed

Types of Mutation Testing

Understanding Variations

Value Mutation

- Involves **changing** the **values** of **constants**, **method parameters**, or **loop variables**
- Aimed at **testing program behavior** under **varied conditions** and identifying potential weaknesses
- **Original Code:**

```
int originalValue = 10;
if (originalValue > 5) {
    Console.WriteLine("Original code: Value is greater than 5.");
}
```

Value Mutation

- **Mutant Code (Value Mutated):**

```
int originalValue = 10;
int mutantValue = 2; // Changed from 10 to 2
if (mutantValue > 5) {
    Console.WriteLine("Mutant code: Value is greater
  than 5.");
}
```

Decision Mutation

- Modifies **logical** and **arithmetic operators** within a program
- **Changes** impact the application's **decision-making** processes and subsequently **alter its results**
- **Original code:**

```
int a = 10;
int b = 5;
if (a > b) {
    Console.WriteLine("Original code: a is greater than b.");
}
```

Decision Mutation

- **Mutant Code (Decision Mutated):**

```
int a = 10;
int b = 5;
if (a < b) {  // Changed from a > b
    Console.WriteLine("Mutant code: a is less than b.");  // Changed message
}
```

Statement Mutation

- Involves **changing complete code statements**
- Modifications **include deleting** an **entire statement**, **re-ordering** statements within the code, **copying** and **pasting** statements to different locations, or **replicating** certain statements
- **Original Code:**

```
int x = 5;
int y = 10;
int result = x + y;
```

Statement Mutation

- **Mutant Code (Statement Mutated):**

```
int x = 5;
int y = 10;
// int result = x + y; // Mutated: Statement removed
int result = x - y; // Changed operation from addition
  to subtraction
```

Mutation Testing Tools

Mutation Testing Tools

- **Purpose**:
  - Automate the process of applying mutations to the software codebase
  - Evaluate the effectiveness of a test suite in detecting these introduced faults
- **Functionality**:
  - Introduce controlled faults or "mutations" into code
  - Run the existing tests to see if they "kill" the mutants
  - Generate reports indicating the mutation coverage and detection

Mutation Testing Tools

- PIT (Pitest) - Highly performant mutation testing tool for Java
- Jumble - Works by modifying Java bytecode
- LittleDarwin - Lightweight and easy to use (for Java)
- Cosmic Ray - wide range of mutation operators for Python
- Mutmut - Straightforward command-line interface for Python
- Mutode - Simplicity and ease of use for Node.js applications
- Stryker.NET - Mutation testing tool for C#, JS and Scala

Stryker Mutator

Kill the Mutants

Striker Mutator Overview

- Provides **intelligible reports** that help identify surviving mutants, improving test suite effectiveness
- **Features**:
  - Supports over **30 mutation types**
  - Utilizes **code analysis** and **parallel test runners** for speed
  - **Works** seamlessly with **various test runners**
  - **Maintained** by the **open-source community** on GitHub
  - Compatible with **JavaScript**, **TypeScript**, **C#**, and **Scala**

Installing Stryker

- You are given a Class Library project named **ArrayTools**
- You also have a Test project named **ArrayToolsTests**
- Execute the test within the VS IDE to ensure it passes and the setup is correct
- To install **Stryker,** open **Package Manager Console** and run the following command:

```
dotnet tool install --global dotnet-stryker
```

- You should see messages in the Package Manager Console indicating that the tool is being installed

Invoke Stryker

- To invoke Stryker, first you need to **navigate** to your **Test project directory**

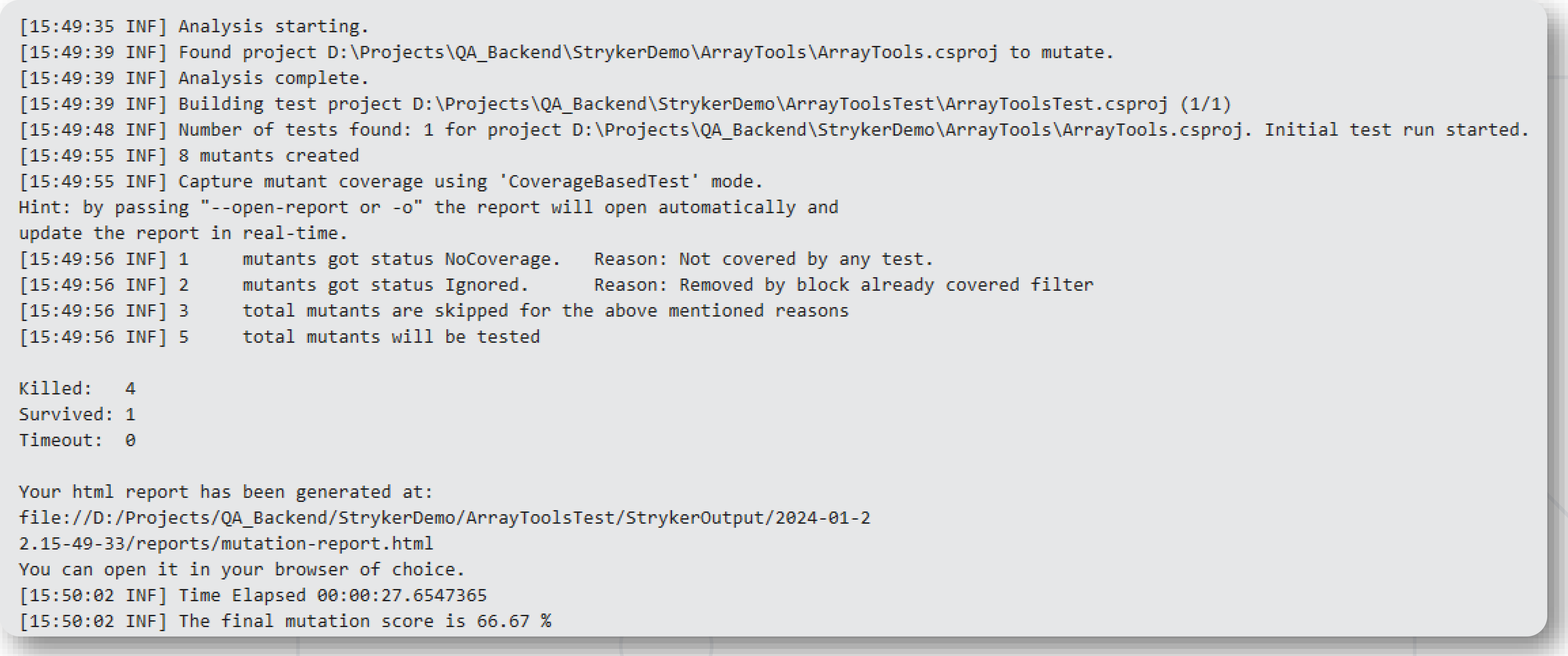
```
cd path\to\ArrayToolsTest
```

- If you are unsure in which directory you're currently at, you can run **pwd** in PM Console
- Now you **can invoke Stryker** with the following command:

```
dotnet-stryker
```

Stryker Report

- **Results** in PM Console:



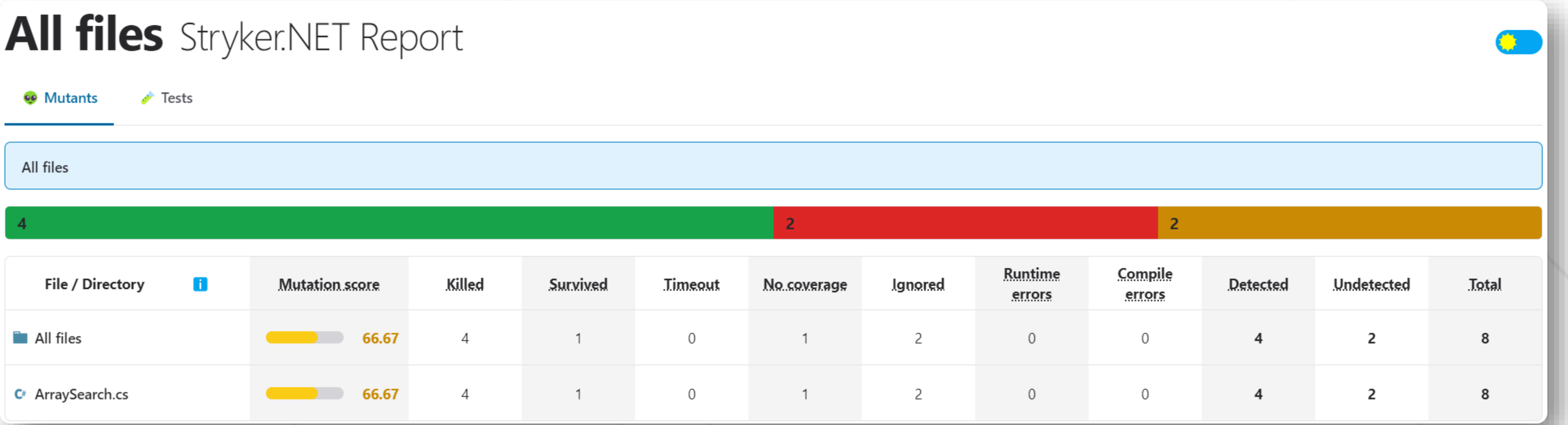
```
[15:49:35 INF] Analysis starting.
[15:49:39 INF] Found project D:\Projects\QA_Backend\StrykerDemo\ArrayTools\ArrayTools.csproj to mutate.
[15:49:39 INF] Analysis complete.
[15:49:39 INF] Building test project D:\Projects\QA_Backend\StrykerDemo\ArrayToolsTest\ArrayToolsTest.csproj (1/1)
[15:49:48 INF] Number of tests found: 1 for project D:\Projects\QA_Backend\StrykerDemo\ArrayTools\ArrayTools.csproj. Initial test run started.
[15:49:55 INF] 8 mutants created
[15:49:55 INF] Capture mutant coverage using 'CoverageBasedTest' mode.
Hint: by passing "--open-report or -o" the report will open automatically and
update the report in real-time.
[15:49:56 INF] 1     mutants got status NoCoverage.   Reason: Not covered by any test.
[15:49:56 INF] 2     mutants got status Ignored.      Reason: Removed by block already covered filter
[15:49:56 INF] 3     total mutants are skipped for the above mentioned reasons
[15:49:56 INF] 5     total mutants will be tested

Killed:   4
Survived: 1
Timeout:  0

Your html report has been generated at:
file://D:/Projects/QA_Backend/StrykerDemo/ArrayToolsTest/StrykerOutput/2024-01-2
2.15-49-33/reports/mutation-report.html
You can open it in your browser of choice.
[15:50:02 INF] Time Elapsed 00:00:27.6547365
[15:50:02 INF] The final mutation score is 66.67 %
```

Stryker Report

- The Report also creates a directory in your Test Project, called **StrykerOutput**
- You can find there the **mutation-report.html** and open it with your browser



All files Stryker.NET Report

File / Directory	Mutation score	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
All files	66.67	4	1	0	1	2	0	0	**4**	**2**	**8**
ArraySearch.cs	66.67	4	1	0	1	2	0	0	**4**	**2**	**8**

Stryker Report – Mutant States

- **Pending**: Mutant yet to be tested. Temporary state
- **Killed**: A test failed; the mutant is eliminated. Ideal outcome
- **Survived**: All tests passed; a test is likely missing for this mutant
- **No Coverage**: No test covers this mutant; it survived as a result

Stryker Report – Mutant States

- **Timeout**: Test run exceeded the time limit, possibly due to issues like infinite loops; it's counted as detected
- **Runtime Error**: An error occurred during test execution, not reflected in the mutation score
- **Compile Error**: Mutant caused a build failure, not counted in the mutation score
- **Ignored**: Mutant was not tested due to being ignored or another reason; doesn't impact the mutation score

Stryker Report – Metrics

- **Based** on the **states, metrics** are **calculated**:
  - Detected (**killed** + **timeout**) - The number of mutants detected by the tests
  - Undetected (**survived** + **no coverage**) - The number of mutants that are not detected by the tests
  - Covered (**detected** + **survived**) - The number of mutants that the tests produce code coverage for
  - Valid (**detected** + **undetected**) - The number of valid mutants. They didn't result in a compile error or runtime error

Stryker Report – Metrics

- **Invalid** (**runtime errors** + **compile errors**) - The number of invalid mutants. They couldn't be tested because they produce either a compile error or a runtime error
- **Total mutants** (**valid** + **invalid** + **ignored** + **pending**) - All mutants
- **Mutation score** (**detected / valid * 100**) - The total percentage of mutants that were detected. **The higher, the better!**
- **Mutation score based on covered code** (**detected / covered * 100**) - The total percentage of mutants that were detected based on the code coverage results

Stryker Report - Test States and Metrics

- A test can also have state with regards to mutation testing
  - **Killing**: The test is killing at least one mutant. This is what you want
  - **Covering**: The test is covering mutants, but not killing any of them. The coverage information should be available per test to provide this test state
  - **Not covering**: The test is not even covering any mutants (and thus not killing any of them)
  - **Total** (not covering + covering + killing) - Total number of tests

More with Stryker

- **Custom Mutations**: You can define custom mutation operators specific to your codebase for more targeted testing
- **Stryker Dashboard**: Visualizes mutation testing reports and provides an aggregated view of the mutation score over time
- Stryker can be **integrated** into **GitHub workflows**. You can configure automated mutation testing as part of your CI/CD pipeline
- Utilize Stryker's **parallel execution** and other performance features to optimize the mutation testing process for large projects

Mutation Testing Wrap-Up

Advantages, Disadvantages, When to Use it

Advantages of Mutation Testing

- Achieves **Extensive Coverage**
- Mimics errors to **enhance** test suite **detection capabilities**
- Leads to the creation of **comprehensive test cases**
- Subjects the test suite to **various scenarios**, including **edge cases**
- Uncovers **potential issues** that traditional testing might miss
- Enhances **Error Detection**
- Helps identify **undetected gaps** in test coverage
- **Early detection and fixing of issues** by software developers

Disadvantages of Mutation Testing

- **Costly** and **Time-Consuming**
- Generating numerous mutants can be **resource-intensive**
- Requires **automation tools** for efficient execution
- **Extensive Testing** Required
- Each mutation might need as many tests as the original program, **increasing testing efforts**
- **Unsuitable** for **Black Box Testing**

When to Perform Mutation Testing

- **Early in the Test Process**
  - Conducted during the unit testing phase for timely improvements
- **For Various Software Types**
  - Suitable for web, mobile, and desktop applications, ideally added early in development

When Not to Perform Mutation Testing

- During **Black Box Testing** Focus
- If **testing** is **limited to front-end** or user interface without delving into code internals
- When **Time** and **Resources** are limited
- May be skipped if considered **too resource-intensive**
- Can be omitted if **test cases** are thoroughly **vetted by QA professionals**

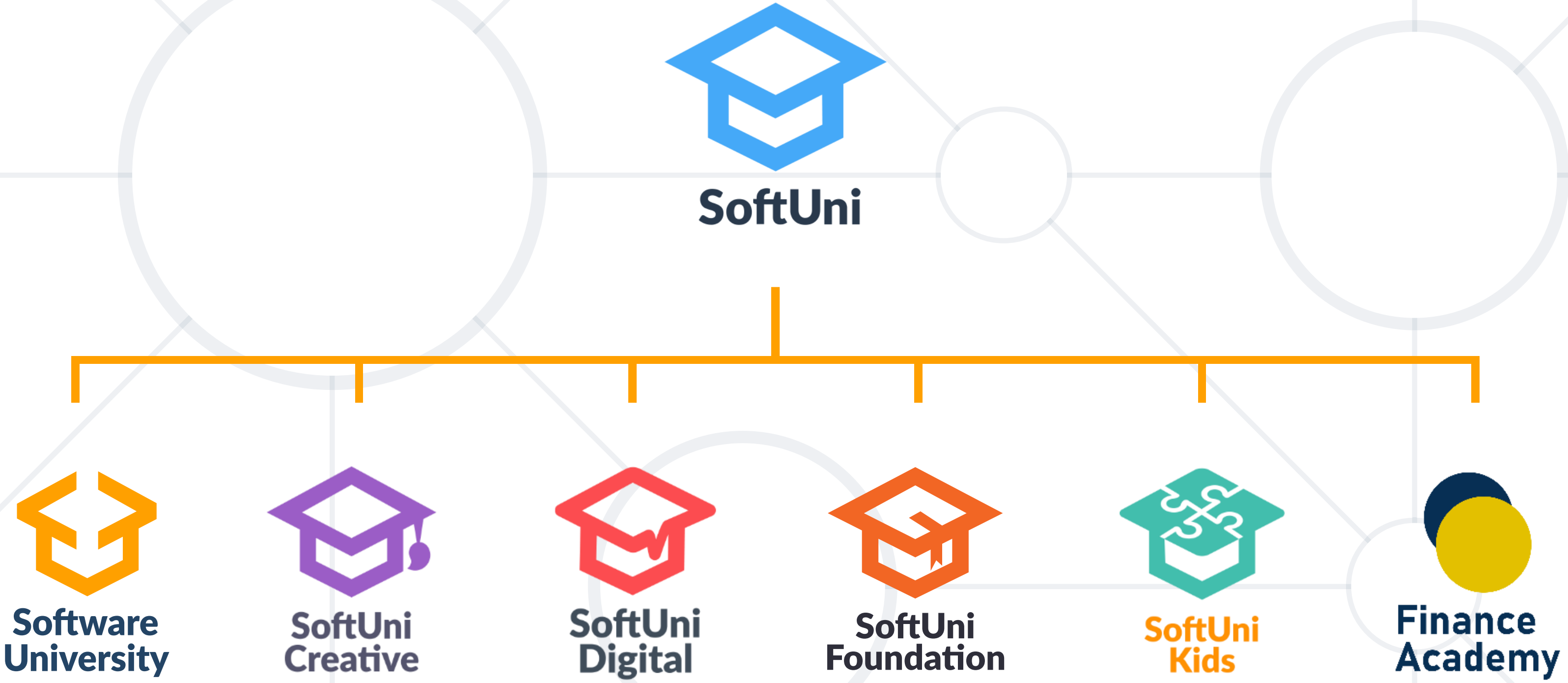
Summary

Software University

- Mutation Testing - Identifying Gaps in Tests
- Types of Mutation Testing - Value, Decision, Statement
- Tools for Mutation Testing - Automate and Analyze Mutations
- Stryker - Powerful Suite of Features
- Mutation Testing Wrap-up - Advantages vs. Disadvantages; When to use it

Software University

Questions?



SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

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