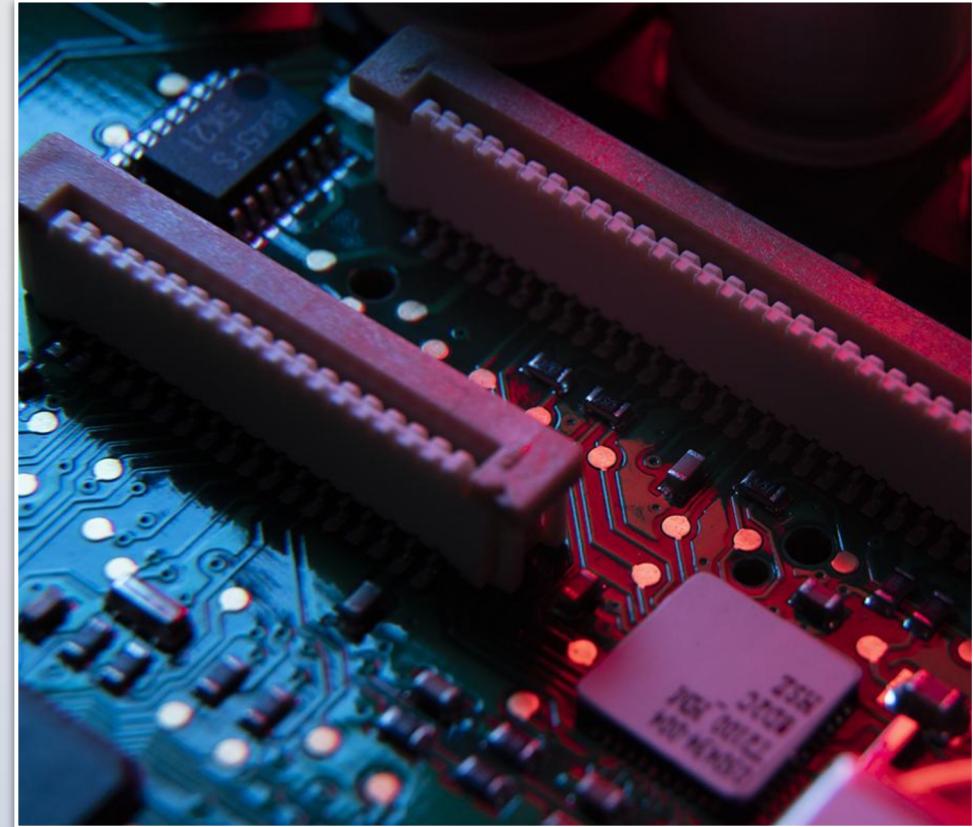




Extended Software Testing

Software Engineering

2023 - 2024





Introduction to Software Testing

- Verification & Validation
- Software Testing Types
- Software Testing Processes
- Unit Testing
- Performance Testing
- Record & Playback
- Designing a Testing Process
- Software Engineer & Testing





What is Software Testing?

- Software Testing is a **process** in which you **execute** your software with **simulated** data.
- If the test passes, the program is correct for the given data.
 - If the data is representative for a larger class of data, you can infer that the program is correct for any member of the data set.
 - If the test fails, we have **a bug**
- Two types of bugs:



...



Verification & Validation

“Testing can only show the presence of errors, not their absence[†]”

- Verification – Testing that the program runs correctly
 - $2 + 2 = 4$ – True
- Validation – Testing that the program does what it is supposed to do
 - You require a list of the top 10 IMDb movies, but you get a queue of the top 10 IMDb movies

[†]Dijkstra, E. W. 1972. “The Humble Programmer.” Comm. ACM 15 (10): 859–66. doi:10.1145/355604.361591

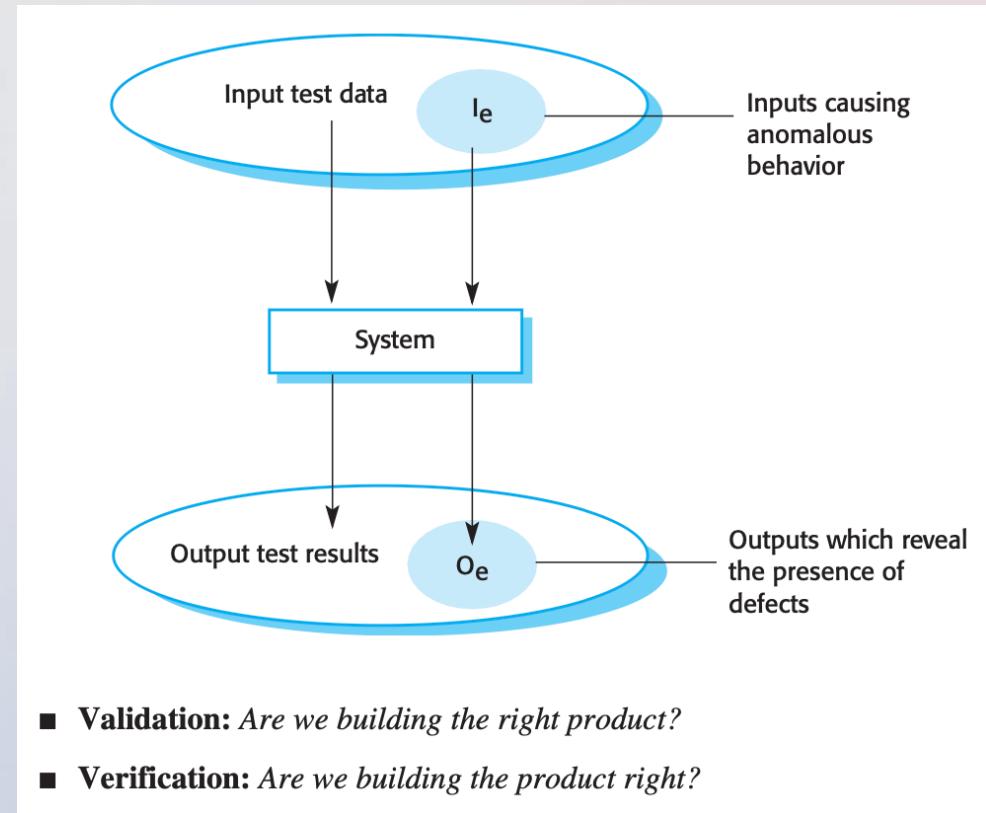




Verification & Validation

“Testing can only show the presence of errors, not their absence[†]”

- Verification – Testing that the program runs correctly
 - $2 + 2 = 4$ – True
- Validation – Testing that the program does what it is supposed to do
 - You require a list of the top 10 IMDb movies, but you get a queue of the top 10 IMDb movies

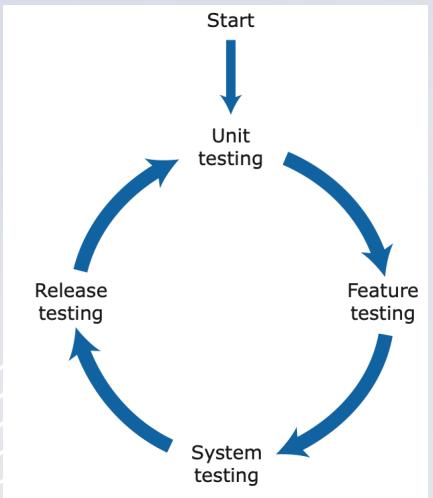


[†]Dijkstra, E. W. 1972. “The Humble Programmer.” Comm. ACM 15 (10): 859–66. doi:10.1145/355604.361591

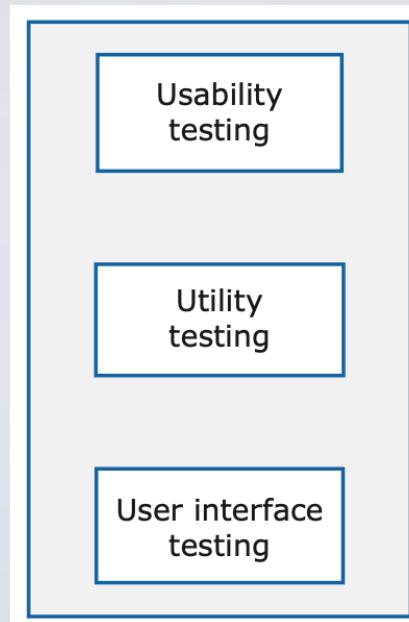


Testing Types

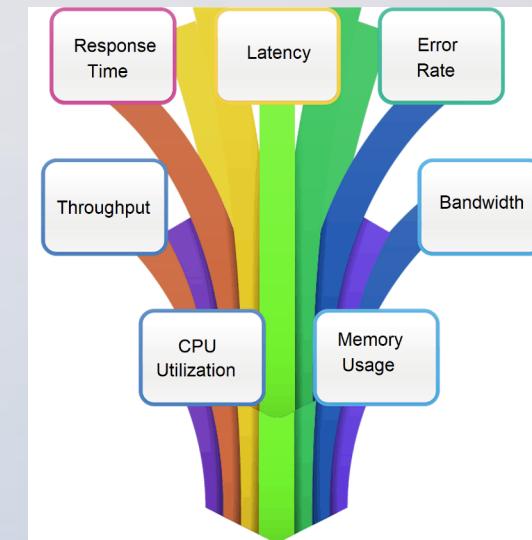
Functional testing



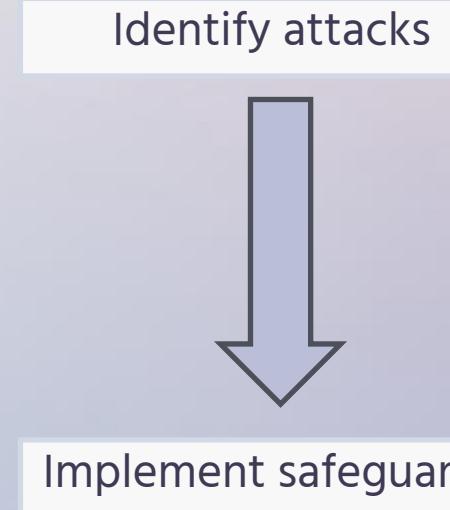
User testing



Performance testing

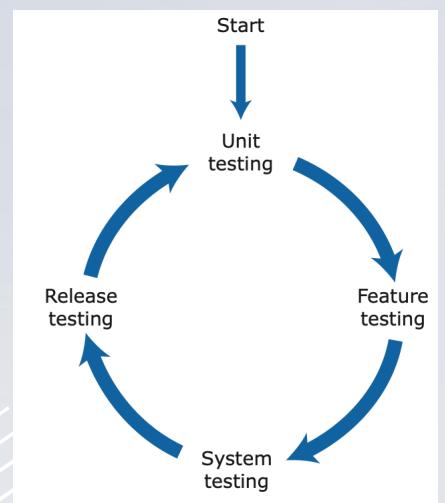


Security testing



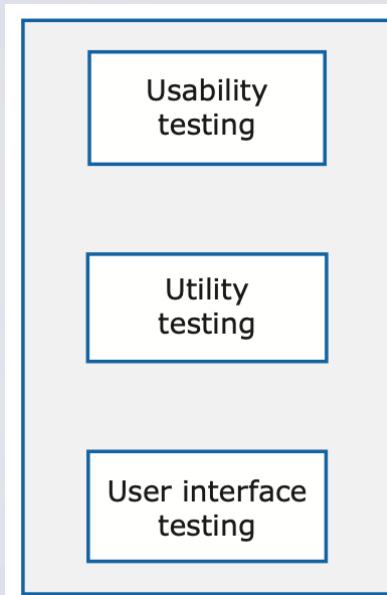
Testing Types

Functional testing



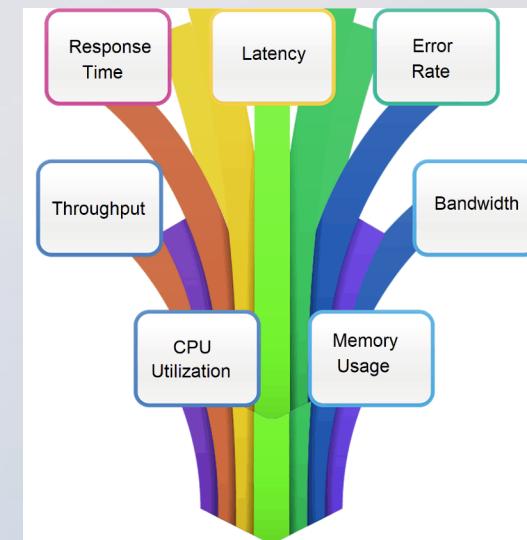
V & V

User testing



Validation

Performance testing



Verification

Security testing

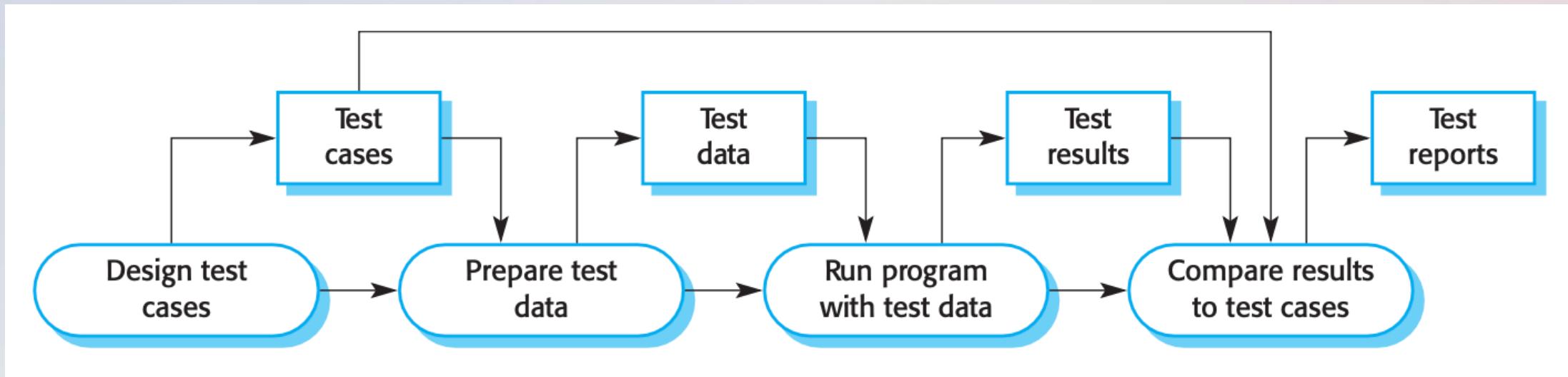
Identify attacks



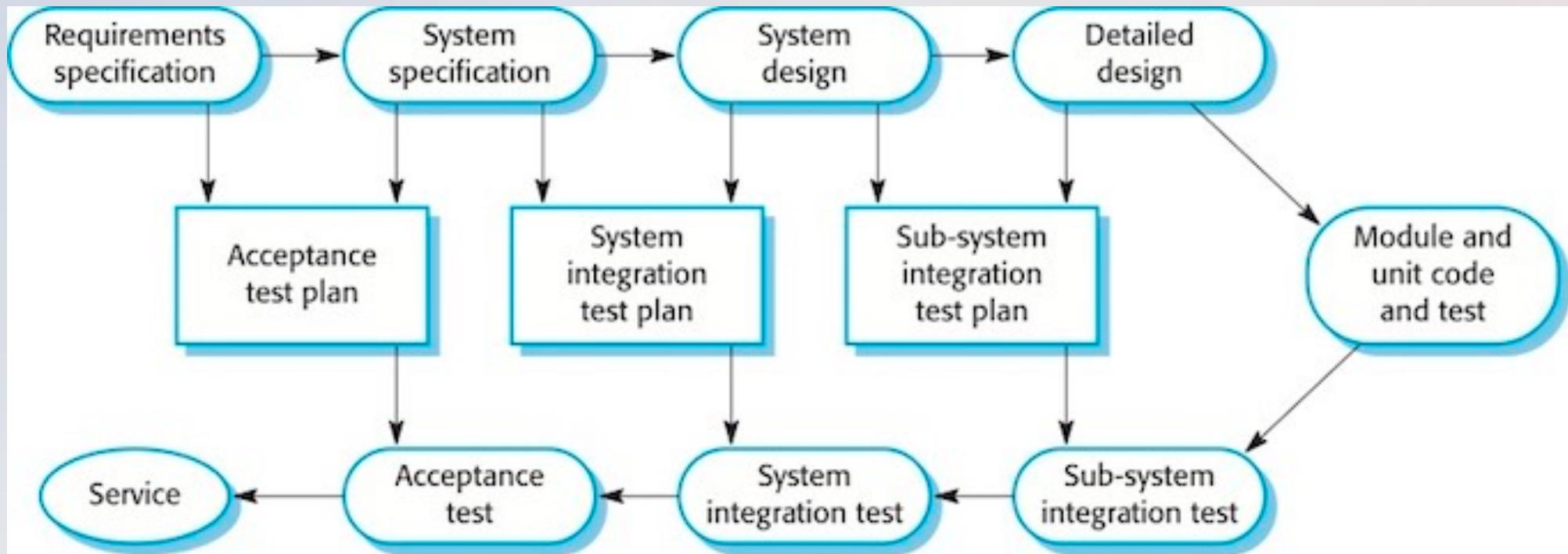
Implement safeguards

V & V

Basic Software Testing Process



V-model Testing Process





Test Driven Development



"We don't do that here."





Unit Testing

- Defining and testing the smallest unit of your software.
 - A unit is small enough to be a unit when you can't reduce the complexity of the test any further.
- Stop over-using it ☺
- When to use it?
 - When you are able to define a unit (equivalent sizes)
 - When the unit si complex or critical.
- When not to use it?
 - When the "unit" depends on many other services (for example if you have to mock multiple dependencies – DB connection, DTO TranslationService, etc.)
 - When test assertion depends on too many variables (but then again, probably your unit could be broken down) – see next slides





Unit Testing (Bad Example)

Depends on a complex(?) Service function

```
[HttpGet]  
👤 Rares Cristea  
public async Task<ActionResult<List<Teacher>>> GetAllTeachers()  
{  
    var result:List<TeacherSendDTO>? = await _teacherService.GetAllTeachers();  
    if (result is null)  
    {  
        return ⚡NotFound("Didn't find the list of teachers.");  
    }  
    return ⚡Ok(result);  
}
```

Not a lot of actual functionality

Heavily dependent on Database Scenario



Unit Testing (Better Example)

It's good to make a Unit Test for this function, because:

- Single, well defined purpose
- Can easily define homogenous test cases (departments, and corresponding DTOs)

```
public class DepartmentSendDTO
{
    public long Id { get; set; }

    public string FullName { get; set; }

    public FacultySendDTO Faculty { get; set; }
}
```

```
public DepartmentSendDTO DepartmentToDepartmentSendDto(Department department)
{
    return new DepartmentSendDTO()
    {
        Id = department.Id,
        FullName = department.FullName,
        Faculty = _facultyService.GetSingleFaculty(department.FacultyId).Result
    };
}
```

Still not ideal due to
this dependency



Unit test breakdown

```
class Program
{
    static void Main(string[] args)
    {
        using (SqlConnection connection = new SqlConnection(
            "Server=[SQ_SIXTEEN];Database=[PocketCentral];User ID=[un];Password=[pw];Trusted_Connection=true"))
        {
            connection.Open();
            using (SqlCommand command = new SqlCommand("SELECT * FROM tbl_Terminal", connection))
            {
                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        for (int i = 0; i < reader.FieldCount; i++)
                        {
                            Console.WriteLine(reader.GetValue(i));
                        }
                        Console.WriteLine();
                    }
                }
            }
        }
    }
}
```

- Let's assume the code above.
- It opens a **DB connection**, **reads** some data, and **displays** it in the console.



Unit test breakdown

What can go wrong?

Database may
not connect

Parsing a non-
iterable object

SQL malformed

Trying to write
an object without
writing overload

```
class Program
{
    static void Main(string[] args)
    {
        using (SqlConnection connection = new SqlConnection(
            "Server=[SQ_SIXTEEN];Database=[PocketCentral];User ID=[un];Password=[pw];Trusted_Connection=true"))
        {
            connection.Open();
            using (SqlCommand command = new SqlCommand("SELECT * FROM tbl_Terminal", connection))
            {
                {
                    for (int i = 0; i < reader.FieldCount; i++)
                    {
                        Console.WriteLine(reader.GetValue(i));
                    }
                    Console.WriteLine();
                }
            }
        }
    }
}
```

Are these versions of the same problem?



Unit test breakdown

+ Connecting to
the database

+ Executing an
SQL command

+ Reading, and
parsing an object

+ Writing

4 Unit Tests

(each with multiple test values)

```
class Program
{
    static void Main(string[] args)
    {
        using (SqlConnection connection = new SqlConnection(
            "Server=[SQ_SIXTEEN];Database=[PocketCentral];User ID=[un];Password=[pw];Trusted_Connection=true"))
        {
            connection.Open();
            using (SqlCommand command = new SqlCommand("SELECT * FROM tbl_Terminal", connection))
            {
                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        for (int i = 0; i < reader.FieldCount; i++)
                        {
                            Console.WriteLine(reader.GetValue(i));
                        }
                        Console.WriteLine();
                    }
                }
            }
        }
    }
}
```





Unit Testing

Join at menti.com | use code 2545 1554



Waiting for players...



Start quiz





Visual “Unit Testing”

The screenshot shows the Storybook interface with the "Canvas" tab selected. On the left, a sidebar lists various components under the "Components" heading, including "DatePicker", "ActivityList", "Footer", "Header", "Sidebar", "NewAppForm", "Pagination", "PageLayout", and "Notification". The "DatePicker" component is currently selected, indicated by a blue background. The main canvas area displays a date picker interface. The calendar shows the month of August (AUG) and the beginning of September (SEP). The date "31" is highlighted with a green circle, indicating it is the active or selected date. The calendar grid includes columns for Sunday through Saturday and rows for specific dates.

Visual test in a glance





Performance Testing

- Load test
- Stress Test
- Scalability Test
- Endurance Testing
- Great explanation: (1) Performance, Load, Stress or Endurance Test? Which do you want? | LinkedIn



Performance Testing

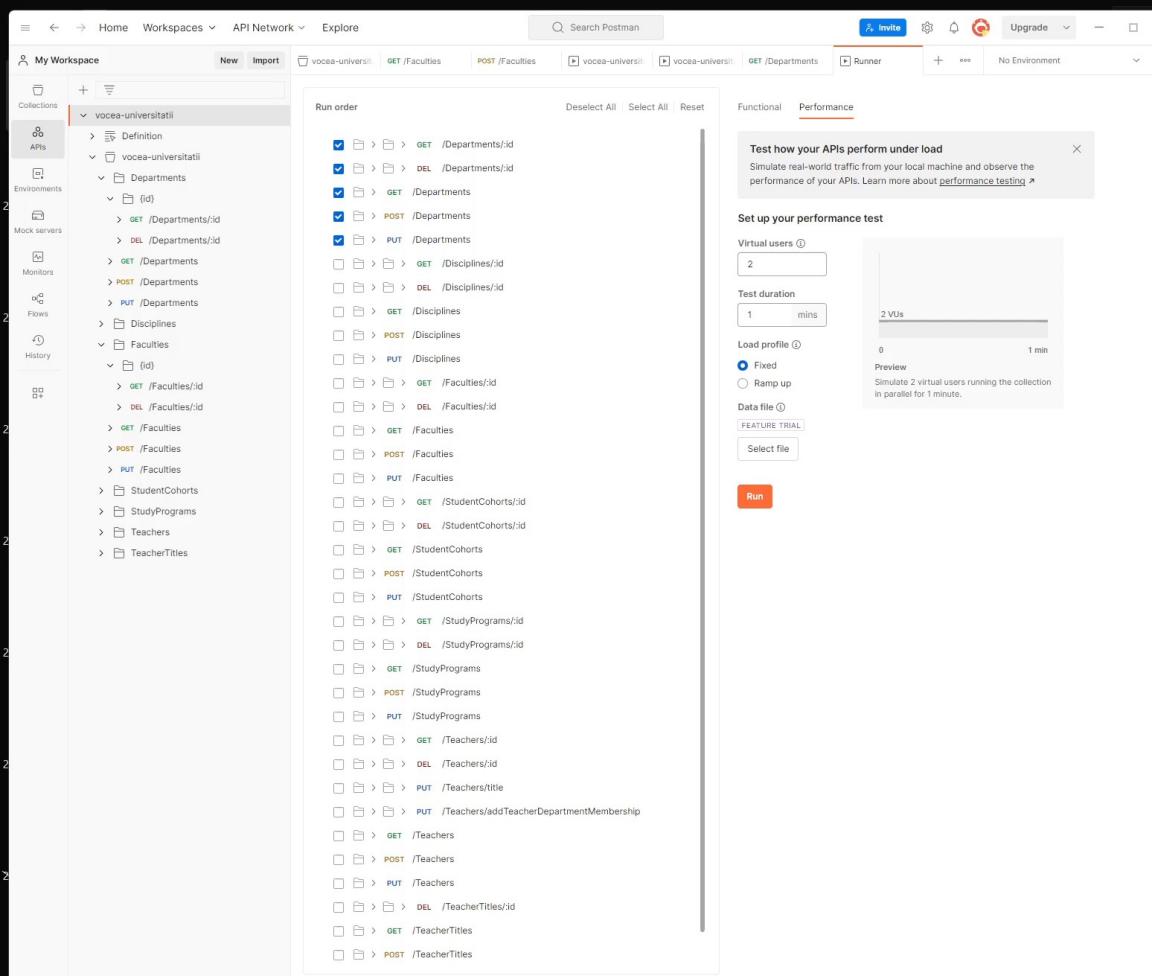
The screenshot displays two windows side-by-side. On the left is the Postman interface, specifically the 'Performance' tab for a collection named 'vocea-universitati'. The table shows one run with the following details:

Run	Start time	VUs	Duration	Total requests	Requests/s	Resp. time (Avg ms)	Error %
#1	Dec 18, 2023, 13:08:37	20	1m	251	10.64	1,159	84.06

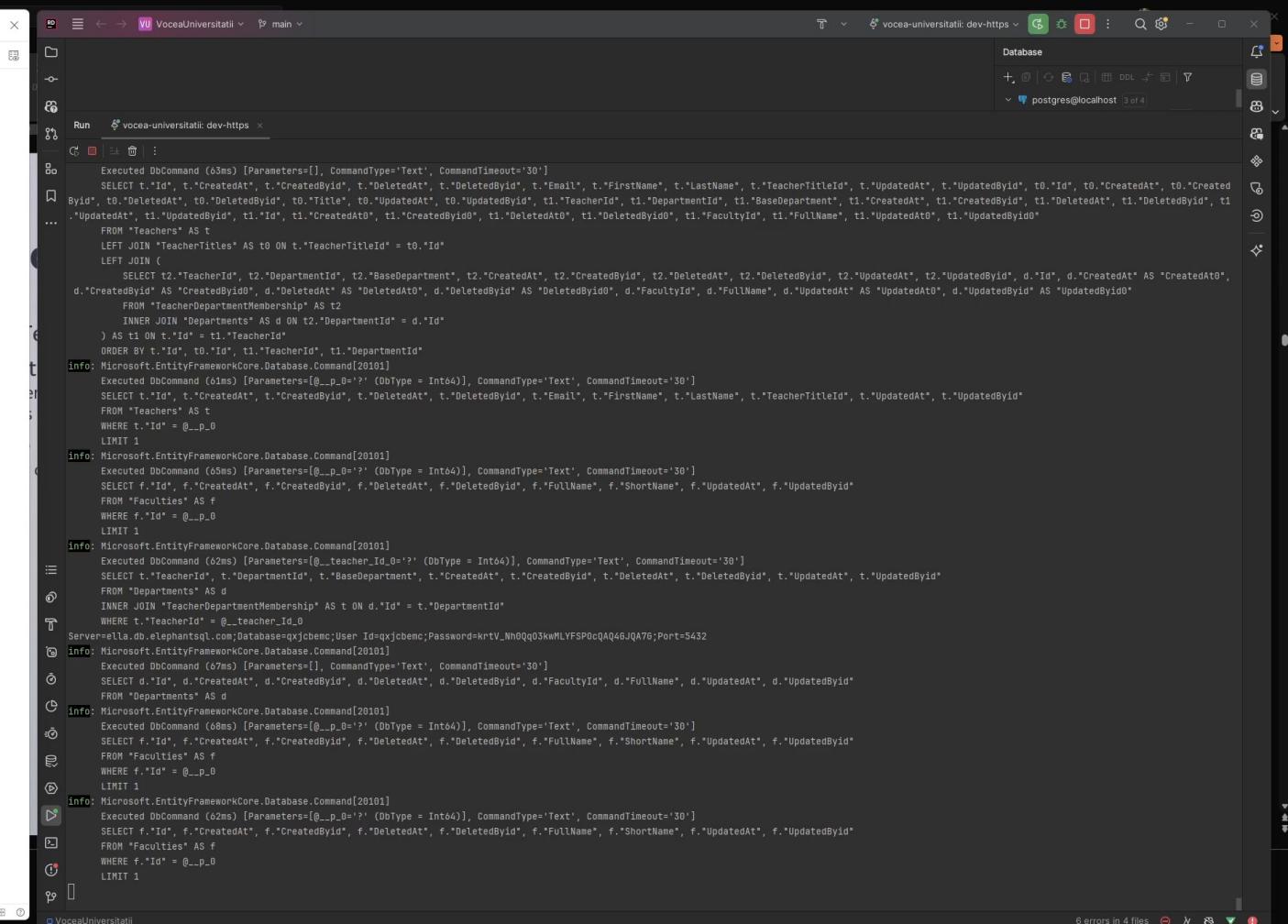
On the right is a detailed error stack trace for a failed database connection attempt. The error message indicates a transient failure to connect to PostgreSQL at localhost:

```
System.InvalidOperationException: An exception has occurred while executing the request.  
System.InvalidOperationException: An exception has been raised that is likely due to a transient failure.  
-> Npgsql.NpgsqlException (0x80004005): Failed to connect to 13.93.108.05:5432  
-> System.Net.Sockets.SocketException (10061): No connection could be made because the target machine actively refused it.  
at System.Net.Sockets.Socket.AwaitableSocketAsyncEventArgs.ThrowException(SocketError error, CancellationToken cancellationToken)  
at System.Net.Sockets.Socket.AwaitableSocketAsyncEventArgs.System.Threading.Tasks.ValueTaskSource.GetResult(Int32 token)  
at System.Net.Sockets.Socket.<ConnectAsync>g__WaitForConnectWithCancellation|281_0(AwaitableSocketAsyncEventArgs saea, ValueTask connectTask, CancellationToken cancellationToken)  
at Npgsql.TaskTimeoutAndCancellation.ExecuteSync(Func<ValueTask> getTaskFunc, NpgsqlTimeout timeout, CancellationToken cancellationToken)  
at Npgsql.Internal.NpgsqlConnector.ConnectAsync(NpgsqlTimeout timeout, CancellationToken cancellationToken)  
at Npgsql.Internal.NpgsqlConnector.ConnectAsync(NpgsqlTimeout timeout, CancellationToken cancellationToken)  
at Npgsql.Internal.NpgsqlConnector.ConnectAsync(NpgsqlTimeout timeout, Boolean isFirstAttempt)  
at Npgsql.Internal.NpgsqlConnector.<OpenCore>g__OpenCore|216_1(NpgsqlConnector conn, SslMode sslMode, NpgsqlTimeout timeout, Boolean async, CancellationToken cancellationToken, Boolean isFirstAttempt)  
at Npgsql.Internal.NpgsqlConnector.Open(NpgsqlTimeout timeout, Boolean async, CancellationToken cancellationToken)  
at Npgsql.PoolingDataSource.<Gettg_RentAsync>g__OpenpgsqlConnection|28_0(NpgsqlConnection conn, NpgsqlTimeout timeout, Boolean async, CancellationToken cancellationToken)  
at Npgsql.PoolingDataSource.<Gettg_RentAsync>g__OpenpgsqlConnection|28_0(NpgsqlConnection conn, NpgsqlTimeout timeout, Boolean async, CancellationToken cancellationToken)  
at Npgsql.NpgsqlConnection.<Open>g__OpenAsync|45_0(Boolean errorsExpected, CancellationToken cancellationToken)  
at Microsoft.EntityFrameworkCore.Storage.RelationalConnection.OpenInternalAsync(Boolean errorsExpected, CancellationToken cancellationToken)  
at Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteReaderAsync(RelationalCommandParameterObject parameterObject, CancellationToken cancellationToken)  
at Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteNonQuery(CancellationToken cancellationToken)  
at Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteReaderAsync(RelationalCommandParameterObject parameterObject, CancellationToken cancellationToken)  
at Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteNonQuery(CancellationToken cancellationToken)  
at Npgsql.EntityFrameworkCore.PostgreSQL.Storage.Internal.NpgsqlExecutionStrategy.ExecuteAsync(IState, TResult) [TState state, Func<T> operation, Func<T> verifySucceeded, CancellationToken cancellationToken]  
--- End of inner exception stack trace ---  
at Npgsql.EntityFrameworkCore.PostgreSQL.Storage.Internal.NpgsqlExecutionStrategy.ExecuteAsync(IState, TResult) [TState state, Func<T> operation, Func<T> verifySucceeded, CancellationToken cancellationToken]
```

Performance Testing



The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists several API collections under 'vocea-universitati'. The 'vocea-universitati' collection is expanded, showing various endpoints for Departments, Disciplines, Faculties, StudentCohorts, StudyPrograms, Teachers, and TeacherTitles. In the center, the 'Performance' tab is selected in the 'Run order' section. A modal window titled 'Test how your APIs perform under load' provides instructions on simulating real-world traffic. Another modal titled 'Set up your performance test' allows setting virtual users (2), test duration (1 min), and a load profile (Fixed). A 'Run' button is at the bottom of this modal. At the top, there are tabs for 'Invite', 'Upgrade', and 'Runner', along with environment selection and other UI elements.



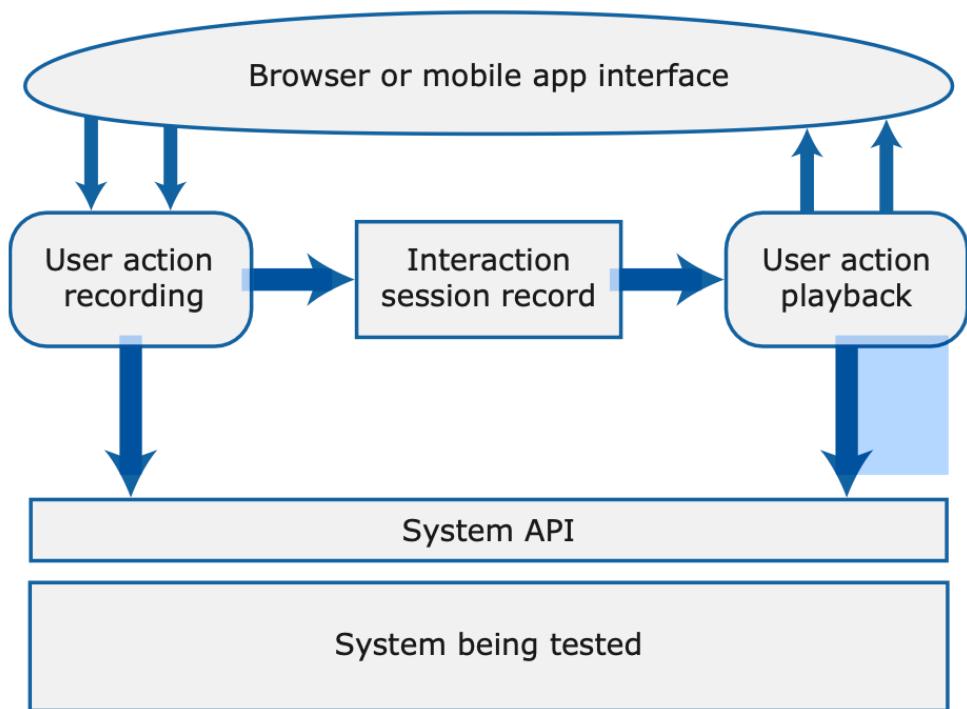
The right side of the screen displays a terminal or database viewer window titled 'vocea-universitati: dev-https'. It shows a series of SQL commands being executed, likely related to Entity Framework Core operations. The logs include details about DbCommands, EntityFrameworkCore.Database.Command, and Microsoft.EntityFrameworkCore.Database.Command, showing queries for selecting data from 'Teachers', 'Disciplines', 'Faculties', 'StudentCohorts', 'StudyPrograms', 'Teachers', and 'TeacherTitles' tables, as well as insertions and updates. The logs also mention 'DbCommand (65ms)' and 'DbCommand (62ms)' with various parameters and command types.

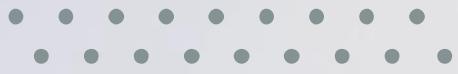


Testing Techniques – Record and Playback

- Suitable for End-To-End Testing
- It tests an entire functionality across the system
- End of the Cycle Technique
- Example: New Account Registration process from front-end.
- Tool:  Selenium IDE

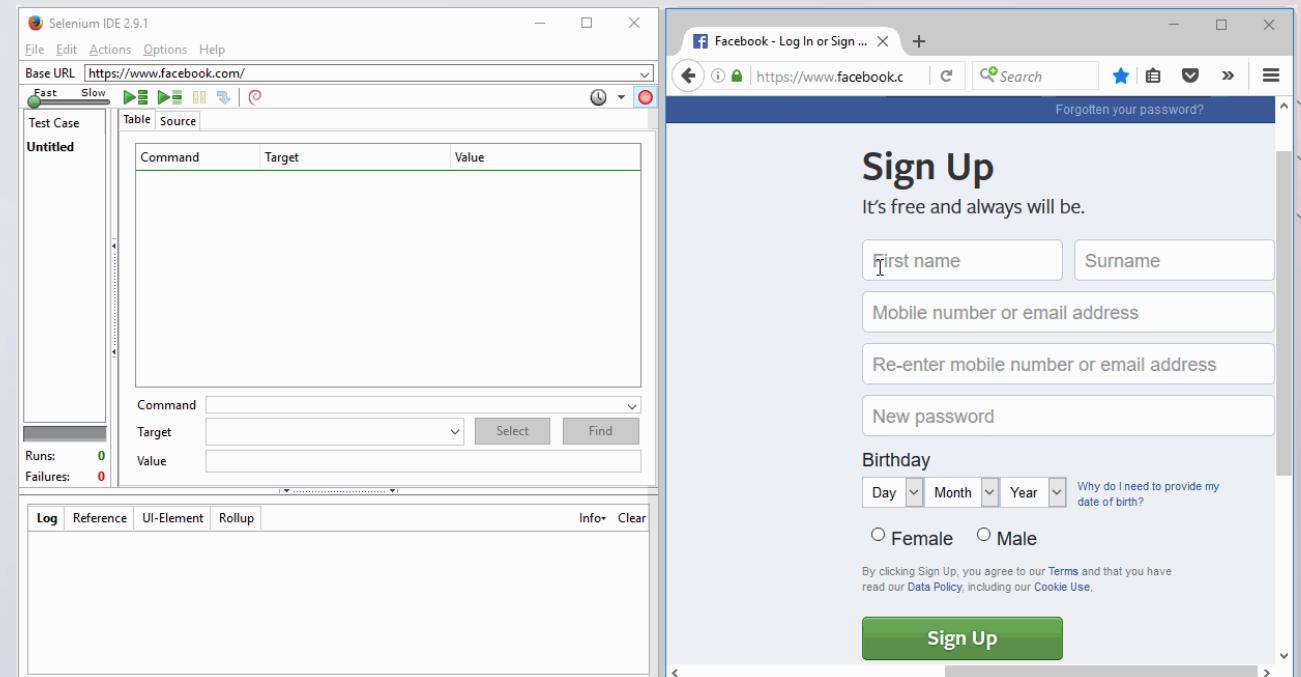
Figure 9.7 Interaction recording and playback





Testing Techniques – Record and Playback

- Suitable for End-To-End Testing
- It tests an entire functionality across the system
- End of the Cycle Technique
- Example: New Account Registration process from front-end.
- Tool:  Selenium IDE



The image displays two windows side-by-side. On the left is the Selenium IDE 2.9.1 interface, which includes a toolbar at the top, a 'Test Case' panel on the left showing 'Untitled', and a main area with a table for recording test steps. The table has columns for 'Command', 'Target', and 'Value'. On the right is a screenshot of a Facebook 'Sign Up' page. The URL 'https://www.facebook.com/' is visible in the browser's address bar. The page itself shows fields for 'First name', 'Surname', 'Mobile number or email address', 'Re-enter mobile number or email address', 'New password', and 'Birthday'. There are also gender selection buttons ('Female', 'Male') and a 'Sign Up' button at the bottom. A small note at the bottom of the page states: 'By clicking Sign Up, you agree to our [Terms](#) and that you have read our [Data Policy](#), including our [Cookie Use](#)'.

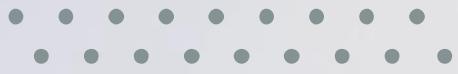




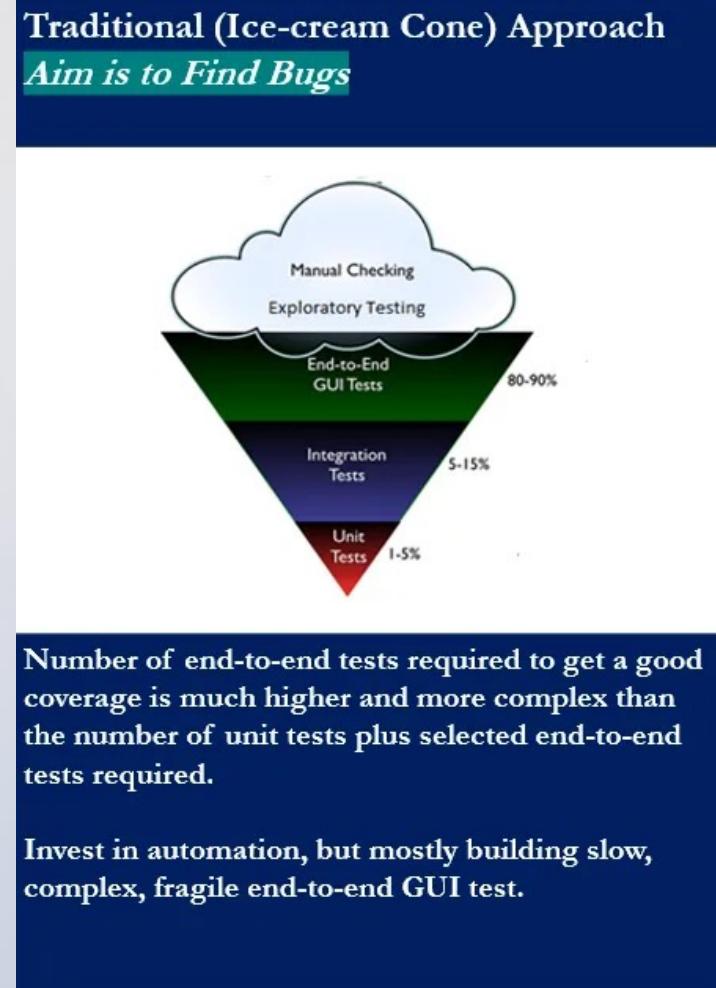
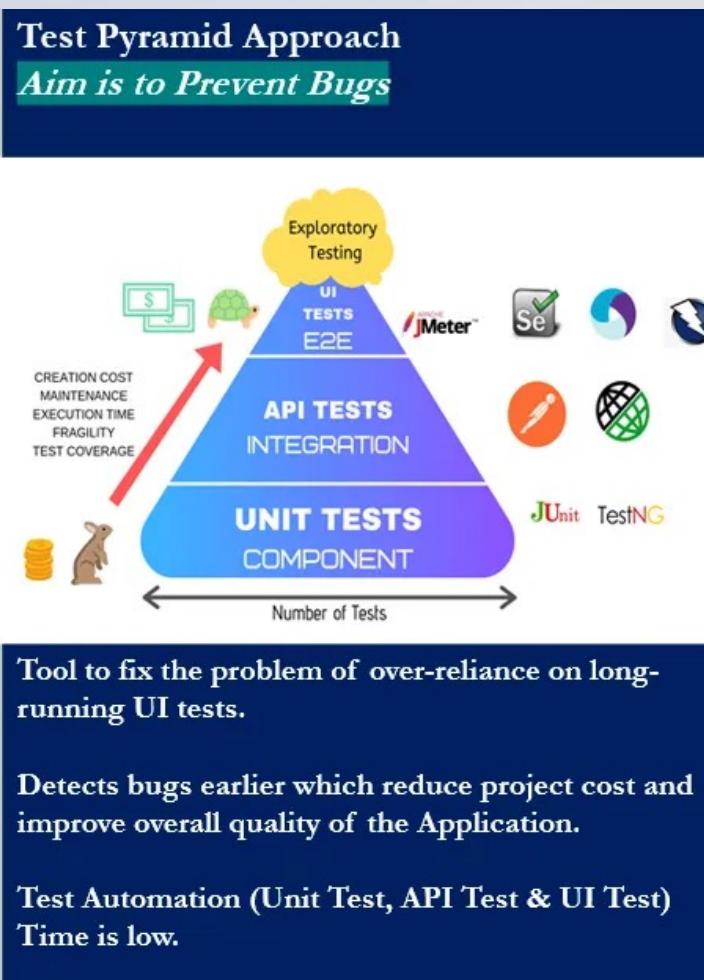
Matching testing types with levels

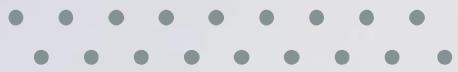
	Functional Testing	User Testing	Performance Testing	Security Testing
Unit Testing	Partition testing / Guideline based testing	Storybook / Visual Testing (A/B Testing)	Stress testing	Partition testing
Integration Testing	System Testing / Behaviour Driven Testing	Requirements-based testing (Acceptance testing)	Interface Testing	?
End-to-End testing	Release Testing / Behaviour Driven Testing	Manual Scenario Testing	Record and Replay	Third-party White-hat hacking





Designing a Testing Approach for your Software





Software Engineer's Role in Testing



Software Engineers

Design Documentation → Design
Data Structures → Design
Architecture →

Write Functional Code →
Review Code →
Unit Test + Integration Test →

Software Engineers in Test

Review the Design →
Check Code Quality & Risks →
Refactor the Code to make it more
testable →

Write Unit Testing Frameworks &
Automation →
Increase Quality & Test Coverage by
adding new features →
Work with Developers →

Test Engineers

Test with their deep Domain
knowledge & Product expertise →

Create / Execute Automation
Scripts →

Report Bugs →
Act as End User & Analyze Risks →





Software Testing Requirements

- Implement a Software Testing Process for your project.
- Use the most appropriate testing process
 - Explain why it was chosen (compare with other methodologies)
 - Describe its components
- Define tests on all three levels (end-to-end, integration, unit)
 - Alternatively, argue why on your project one level is not appropriate.
- The scenarios that you are going to present at the live demo, should be thoroughly tested.
 - Identify the most critical qualities of your software
 - Define test suites that are appropriate for those qualities

Your project must prove that you:

- Can differentiate between multiple testing types
- Know when to apply each
- Implement them, so that they are useful for the application.



Resources Used

