COURSEWARE

- Professional Skills
- Agile Fundamentals
- Jira
- Git
- Databases Introduction
- Java Beginner
- Maven
- Testing (Foundation)
- Java Intermediate
- HTML
- CSS
- Javascript
- Spring Boot
- Selenium
- Sonarqube
- Advanced Testing (Theory)
  - Integration Testing
  - System Testing
  - Acceptance Testing
  - Behaviour-Driven Development (BDD)
  - Non-Functional Testing
- Cucumber
- MongoDB
- Express
- NodeJS
- React
- Express-Testing
- Networking
- Security
- Cloud Fundamentals

System Testing

Contents

- Overview
  - What is System Testing
    - End-to-End Testing
  - Why do we perform System Testing
- Tutorial
  - How to perform System Testing
    - White box approach
    - Black box approach
- Exercises

Overview

Below we will be covering the basics of System testing, exploring how and why we use this testing approach.

What is System Testing

When covering integration testing, System testing is generally considered the next logical step. System testing is about testing the full data flow of interaction with the system.

But what are we saying precisely?

Well, where integration was looking to identify success between communication of `unit`/`class` objects, System testing is about testing the full software path.

While System testing is a strategy, it suffers from the choice to run in isolation (with stubs/mocking) as well as being run with dependencies, while we can run system testing without dependencies such as databases, we could instead opt to perform an End-to-End Test.

End-to-End Testing

We may want to prove that our database or external services perform as expected. Dependencies on this level, however, require some more consideration as our dependencies may be other services or servers such as databases.

Why Perform End-to-End Testing?

When performing an End-to-End test, we might have an environment that specifically creates a test database, servers or resources that are utilised by the system in order to function correctly. Rather than interjecting results in the testing, we can instead provide a pseudo-environment that models and replicates the behaviour with the purpose of localised testing.

The argument here is that if our system runs correctly with its resources in a simulated environment, it **should** functionally run in a production setting.

- The Production Setting
  > This is another expression for a "live system", it is not wise to perform testing on a live system; creating false bookings using tests might completely undermine the purpose and clarify of your application or service, cluttering and disrupting customer experience and data!!

AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

Why do we perform System Testing

Much like a fresh view of testing, we perform system testing in order to test if the application works correctly, that is to say, that when we call the highest level object in the system, the functionality runs as expected and returns the values we are looking for.

So, when System testing our application we are primarily concerned with:

- Validating code system/software works as intended.
- Simulating code execution in a `production setting`.
- Provide testing with example data.

Communication across the application differs from `unit` to `unit`, whilst the parts might work in conjunction with each-other, how does the sum of the application fair from highest to lowest level?

Tutorial

How to perform System Testing

Let's look at our repository model:

- `Controller` -> `Service` -> `DAO`

Where as integration might test the relationship of one of these arrows, System testing is about using functionality from `Controller` and allowing the test to run all the subsequent code dependencies to ensure that the end result has been achieved!

White box approach

This approach overlaps and reflects a integration test approach:

Observe the interaction between the classes below in terms of pathways, we know that unit testing can often encounter branching, but here we may have many particularly reasons for failure.

Each branch may result in a different reason for failure, assuming that path 1 is our success condition, we can we observe that all three methods perform as expected.

Testing pathways:

```
Controller.update() +
                    |
   Service.update() +--+--+
                    |  |  |
                    |  V  V
                    |  3  4
      DAO.update()  +--+
                    |  |
                    |  V
                    V  2
                    1
          Success path
```

▶ Diagram in code format

In this case we might wish to target branches 2, 3 and 4 in addition to 1. However this is still somewhat regarded as an integration test, as system integration tests are very much a valid approach of integration testing.

Black box approach

This approach has an overlap with acceptance testing styled approach; we interact with the system from the highest level without any knowledge of the inner code.

- `Front-end (Html/JS)` -> `Back-end (Java)` -> `Database (mySQL)`

The method of testing would be to use a tool such as Selenium to interact with the front-end in conjunction with the back-end in an isolated test environment, proving that communication; both successful and not are appropriately tested.

This is not to be confused with acceptance testing as we are trying to prove that all available functionality performs as expected without meeting any expectations that might be considered a requirement from the client; it is our opportunity to test all available points of entry into our and throughout our system.

Considerations:

> Whilst the scope of this approach is reduced, there are risks that additional failure cases maybe missed due to the focus on using the interface over the knowledge of the white box approach.

Certain frameworks/technologies can help us support running resources and environments, for example, `Spring` and `Spring boot` is a technology developed by Pivotal Team, it provides a context associated for hosting a server that will host your application locally.

Using a context or testing environment can help you better determine how dependencies will support your code in a production setting,

Exercises

There are no Exercises with this module.