DP2 2023-2024

Report on WIS testing knowledge

Acme Software Factory

Repository: https://github.com/rafcasceb/Acme-Software_Factory-C1.049

Members:

- Castillo Cebolla, Rafael rafcasceb@alum.us.es
- Flores de Francisco, Daniel danflode@alum.us.es
- Heras Pérez, Raúl rauherper@alum.us.es
- Mellado Díaz, Luis luimeldia@alum.us.es
- Vento Conesa, Adriana adrvencon@alum.us.es

GROUP C1.049

Version 1.0

15-02-24

Content Table

Executive Summary

This report provides a comprehensive analysis of the current state of our testing knowledge respect to a Web Information System (WIS) as of the start of this subject. It examines various aspects related to functionality, performance, security, and usability testing of the WIS. Additionally, it explores the most suitable tools and methodologies and known best practices for conducting an effective testing process.

Revision Table

Date	Version	Description of the changes	Sprint
15/02/2024	1.0	• Executive summary • Introduction • Content • Conclusion • Bibliography	1

Introduction

We were first introduced to software testing in the subject of *Fundamentos de la Programación* (FP) and since then, we have been progressively expanding our knowledge in this area thanks to the subjects *Análisis de Datos y Diseño de Algoritmos* (ADDA), *Introducción a la Ingeniería del Software y a los Sistemas de la Información* (IISSI) and *Diseño y Pruebas 1* (DP1), culminating in independently conducting testing tasks.

In this report, we will expose our knowledge on the testing matter. We will start by discussing the fundamental importance of testing, exploring some concepts and approaches such as automated testing and test-driven development. Next, we will delve into service, controller, and transactional testing, reviewing different tools available for testing purposes. And, lastly, a brief comment of test data management.

Contents

Concept and importance of testing

Testing plays a pivotal role in software development by subjecting a software component, known as the Subject Under Testing (SUT), to rigorous analysis. Its primary objective is to verify whether the SUT performs its designated function accurately and efficiently, thereby identifying any potential failures and their root causes.

Typically, a test consists of three fundamental steps: initializing the necessary data (arrange/fixture), executing the SUT (act), and verifying that the expected outcomes are achieved (assert).

We understand the vital role of thorough testing in ensuring the quality and reliability of a system, as well as in enhancing the user experience. Additionally, we recognize the benefits of automated testing, which reduces human error, minimizes monotony for workers, and saves time. Automation also allows for the reuse of tests, improving long-term efficiency.

In software testing, there's often a choice between adhering to the DRY (Don't Repeat Yourself) principle or embracing DAMP (Descriptive And Meaningful Phrases). While effective testing is essential, it's equally important for tests to be understandable, especially since they may be reviewed by other developers in the future. Therefore, prioritizing DAMP principles is generally recommended for clearer and more maintainable testing practices.

Test classification

Tests can be classified based on their scope and granularity, providing insights into their specific focus within the testing process. This classification ranges from less to more granularity, encompassing unit tests, integration tests, end-to-end tests, acceptance tests, and exploratory tests.

Another classification criterion considers how the subject under test interacts with other components during testing. In this context, tests are categorized as sociable or solitary. Although these terms are frequently associated with unit tests, they can apply to testing in various contexts.

Furthermore, tests can be categorized as positive or negative based on the expected outcome of the method execution within the test. Positive testing aims to validate that the system behaves as expected under normal conditions, while negative testing seeks to identify and address potential failure scenarios.

Testing services

Throughout this degree, our focus on testing has primarily centered around unit service testing. This approach involves executing unit tests for each implemented service method

using a controlled sample database. Desired results were clear, the method would be independently executed and actual results were asserted.

Most cases we have developed have been unit tests. The main advantage of these type of tests is that, since they are independent, side effects of one scenario don't accidentally invalidate others.

In DP1 we would make use of the tools JUnit and AssertJ to carry out these tests. The service test classes would be annotated with @DataJpaTest. These tools also allowed for parameterized testing. In IISSI 1, we made use of the Thunder Client extension for Visual Studio Code.

Testing controllers

With controller testing, we delved into the concept of test doubles, which serve as substitutes for real objects during testing, effectively isolating the behaviour of the Subject Under Test (SUT) from its collaborators. This approach promotes solitary testing, allowing for focused evaluation of the SUT's functionality. The most common types of test doubles include stubs, which set the SUT into a specific state; mocks, used to verify interactions within a fixed state; and fakes, providing lightweight implementations. Additionally, we can encounter dummies and spies.

For controllers we used JUnit and Mockito. The controller test classes would be annotated with @WebMvcController.

Testing frontend

We were very briefly introduced to frontend testing but we didn't dive any deep, neither theoretically nor practically. We have focused primarily on backend testing but we acknowledge the importance of these type of tests.

Testing transactions

Throughout various subjects, we've had the opportunity to explore and test transactional behavior, particularly concerning methods that involve database modifications. The process typically involved ensuring the database was in a safe state, executing a modification query, intentionally interrupting it with a controlled error, and verifying that all changes were effectively rolled back to their previous safe state.

Test data management

In our testing practices, we've primarily focused on ensuring that test cases pass successfully, without delving into the management of test data or processing test results. While our tests typically halt program execution upon failure, we haven't explored further analysis or interpretation of the results. We recognize its importance in simulating real-world scenarios and validating software functionality and performance.

Conclusions

Through our academic journey, we have gained a deep understanding of the importance of software testing and a decent, both, theoretical and practical knowledge of its implementation. However, we acknowledge out mastery currently is far from ideal. Despite being well-versed on backend knowledge and the most common approaches at a starter level, for example, our frontend testing knowledge is very limited, our results management simple and we don't fully know yet how testing is really managed in the industry .

Concluding this report, we affirm our commitment to furthering our understanding and honing our skills of software testing.

Bibliography

Intentionally blank.