Project Two (Revised)

Eduardo Rodrigues

SNHU (Southern New Hampshire University)

CS-320-J8975

Rob Tuft

10/18/2023

Summary

Our tests may have covered 80 percent of the deliverable content; but it still has a number of edge cases and functionality to account for now and in future development. There is a need to fashion the code to be more visible by including more exception handling mechanisms and their respective tests.

Unit testing approach

A bottom to top approach development is being followed here; and we have made several assertions that pertain to functionality between the integration of three service classes with their respective Java object classes. The tests were split into six individual test classes and automated the use of Junit using Maven. Essentially, directed strategy was used to test for what the specification intended the team to filter for when populating the POJOs, including tests for known edge cases for the data structures we employed and the accessor and setter framework that we needed to fashion (Morgan, 2019).

Alignment to the requirements

The unit testing was modeled against the software design description. The program had exceptions thrown out for each requirement noted in the specification. Each constraint had a sole assertion modeled against it. We created equivalence partitions with respect to each parameter that would throw an exception given a deviation from the specification.

Effectiveness

Edge cases for each parameter were accounted for with respect to the deliverable requirements. There are however edge cases that can be drawn from the Appointment service and the implementation of the id. The appointment service deals with the Date data type and should have exceptions that account for the use of an incorrect format being delivered to the Appointment classes. There is also the case for keeping track of the upper bounds of our unique identifier, there is no means to account for an exception should the inline Integer data type within the code increment past 10^{10}–1. (Oracle,2023)

Technically Sound

At this point in testing, we are ready to adapt to further integration and system testing. New features can be tested as each class grows and changes. A focus now would be on future implementation and integration to adapt tests to account for use cases and mock environments that will reflect actual working conditions to further harden our stance.

Efficiency

The deliverables themselves work to decouple the passing of demographic data from a service interface into their respective data structures. This sort of organization allows revisions and edge cases to be accounted for when moving forward with a bottom-up development stream and further integration testing.

Reflection

Most of our reflections come from a need to expand the software design template, the specification, to continue development to a more complex platform later in the software life cycle.

Software Testing Techniques

This document is an excellent means of black box testing by code review and reflection on the current state of the project. It solidifies our initial white box approach of unit and integration testing that paves the path for new features and integration to the end-product. Our current unit tests simply apply boundary analysis and equivalence partitions to isolate edge cases bound by our project's list of specifications.

Other Testing Techniques Not used

Future development may result in the addition of various test harnesses that focus on manual testing and testing specific to integration and system testing that form feedback loops with regression testing. Testing will reflect any errors due to changes in the code basis.

Uses and Implications

Future efforts may involve specific tools that pertain to either Design or System specific testing. This testing is relevant to the use case for the software itself. It may involve third-party, new, or legacy code that will be used.

Caution

The directed and test-driven workflow allows the developer and other stakeholders to reflect on how our modularized code can be vetted against the requirements to date. This reduces the inherent risk that our code will hold as the code basis grows over time. Overall risk is reduced because of the tighter feedback loop the team can draw from.

Bias

Bias is also in turn reduced with the transparency and quantifiable resources that we provide to the team as part of the testing process. We decentralize the coding process here by allowing other contributors to be more involved in the development process.

Discipline

The automation and testing methods that we employ lead us to be more considerate of the level of visibility and decoupling that we aim to achieve to enable collaboration and instill a greater level of flexibility in our code and testing basis. Our methodology is to prepare our code to receive and be received by the work of others. As we perform this summary, we see that we are already thinking ahead to the complexities of integration, system, and acceptance testing.

References

Morgan, P., Samaroo, A., Thompson, G., Williams, P. (2019). Software Testing - An ISTQB-BCS Certified Tester Foundation Guide, Chapter 1: The Fundamentals of Testing. Swindon, UK: BCS.

Oracle. Java Platform Standard Edition 8. Class Integer. https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html . Accessed 2023