Nick Forleo

SWENG 581

1. **Input Partitioning-Based Criteria Testing Design: Describe how you would identify relevant partitions for the input fields in the search criteria (e.g., departure city, date), and explain why this is important for efficient testing.**

I would identify relevant partitions for the inputs for search criteria using an interface-based approach. This is because it will ensure that each input parameter is tested and validated. For example, if we said that a partition of the input parameter of the departure date is “Relation of departure date to today”, we would have the following blocks: “before today”, “today”, and “after today”. Each of these values could then be tested. A functionality-based approach would not provide as robust tests. Also, since the result set is not a predefined set, it would be difficult to verify a functionality-based set of partitions for an infinite amount of results. This is important for efficient testing because it provides a set number of input values, whereas much more tests would be necessary to provide all functionally possible test cases.

1. **All Combinations (ACoC) Criterion: Discuss how you would apply the ACoC criterion to validate different combinations of input values (e.g., all possible combinations of departure cities and dates) and their impact on testing coverage.**

To apply All Combinations Criterion, each input parameter value would be partitioned and then create blocks for each parameter. For example, a partition for the departure city could be “value of string” and potential values (blocks) could be “BOS” (airport code), “Boston” (city name), “” (empty string), or “02128” (zip code). The same could also be applied to the arrival city. We could use the blocks for the date partitions as described above. The number of passenger could be partitioned as “relation of number of passengers” with blocks “equal to 0”, “equal to 1”, “equal to 2”, “equal to 3”, “more than 3” (Chose to stop at 3 because planes usually have at most 3 seats in a row). You would then generate test cases for each block value for each partition. To get the total number of test cases you would need, you have to take the number of blocks per partition and multiply them together. This would result in a lot more tests than are needed to achieve good test coverage.

1. **Pair-Wise (PWC) Criterion: Explain the concept of Pair-Wise testing and how it can be used to reduce the number of test cases while maintaining adequate coverage for filtering options (e.g., price, duration, airline).**

To apply Pair-Wise Coverage to the filter criteria, you would need to take a value from each block for each characteristic and combine it with a value from every block for all of the other characteristics. For example, this would mean you could take a value for a characteristic for price and apply it to each value for the blocks for a characteristic of duration and airline. It will reduce test cases because it will ensure the same test coverage without having to retest code just to satisfy a unique combination of input values. This results in less test cases while maintaining the necessary test coverage.

1. **Multiple Base Choice Coverage (MBCC): Describe how MBCC can be utilized to test various base choices, such as different combinations of search criteria, and how it contributes to robust testing.**

MBCC can be utilized to create a more robust test set. This is because it allows you to create multiple base tests and gives the tester the ability to craft these base tests based on domain knowledge. In the Search and Filter example, base tests that return the desired results can be created around specific search criteria. Then, as you rotate the non-base values, you can effectively execute stress testing. This helps with robust testing because you are covering your valid input parameters as well as verifying invalid input parameters are handled correctly. Testing both valid and invalid inputs is important to verify the quality and resilience of the code.

1. **Integration with Agile Practices: Discuss how these testing techniques align with Agile principles, including continuous testing and collaboration with the development team and Product Owner.**

These testing techniques work well with Agile principles. Each of the aforementioned techniques can help when writing automated testing. This is important in agile methodologies because you are constantly building and testing software. This means that the better the techniques are to generate tests and improve test coverage, the more accurate the automated testing can be in CI/CD pipelines - ensuring rapid deployments or deliverables. It may even force more, and more effective, communication between the product owner and development team because some of the techniques may require a better understanding of the domain. In my experience, the better understanding of the domain that I have, the better the quality of my code is. It will also mean that my testing can be more targeted and reflect real world possibilities. These techniques are essential to creating robust and comprehensive test coverage, which when automated, streamline the agile development process.