Software Testing

David Bishop

To Thoroughly test the new credit limit increase functionality for WonderCard, Ultd. We need to consider a variety of test cases to ensure we cover all or nearly all the possible scenarios that are described in the functional requirements. I would use four different types of test cases to ensure this, Boundary testing, Input space partitioning, Decision table testing, and invalid input data.

Boundary Testing:

This will help us test the extremes of the boundaries put forth by the new credit limit increase functionality.

Test Cases: There are eight different test cases I can think of to push these boundaries. These can be separated into 5 different categories.

1. Customer Tenure at the extreme ends
    a. A customer with exactly 5 years as a customer
    b. A customer with just a day less than the 2 year mark and another with a day more than 2 years
2. Credit usage at the thresholds
    a. A customer with exactly $1,000 credit usage
    b. A customer with credit usage at slightly below the credit usage ($999.99) and a customer with a credit usage just above the threshold ($1000.01)
3. Combination of Tenure and Usage
    a. A customer who has been with WonderCard, Ultd. For exactly 5 years and has a credit usage of exactly $1000
4. Credit score edge cases
    a. A customer with borderline credit scores (just transitioning from one credit score level to another)
5. Uncommon scenarios
    a. A customer with extremely high credit usage (like $100,000)
    b. A customer with a very short tenure, just a few weeks

All these test cases will ensure that there are no miscalculations in the end made by the system. While most of these scenarios are unlikely to happen it is important to ensure that a unlikely case is still handled correctly.

Input Space Partitioning:

This will ensure that the software is behaving as expectedly with normal cases.

Test Cases: This can be split into 10 different sections to ensure full coverage.

1. Credit score P

2. Credit Score G, <=5 years as a customer, $0-$1,000 usage
3. Credit score G, >5 years, $0-$1,000 usage
4. Credit score G, >2 years but <= 5 years, >$1,000 usage
5. Credit score E, <=2 years, >$1,000 usage
6. Credit Score E or G, <= 2 years, >$1,000 usage
7. Credit score G, >5 years, >$1,000 usage
8. Credit score E, >5 years, <=$1,000 usage
9. Credit score E, >2 years but <=5 years, >$1,000 usage
10. Credit score E, >5 years, >$1,000 usage

These test cases should encapsulate the entire coverage that the system should have.

Decision Table Testing:

This decision table should also help ensure that every case is accounted for in the new system. While similar to input space partitioning, its not to actually perform testing but to ensure the rest of the testing actually encapsulates the entire system.

Cases that should go on the table:

1. Years as a customer:
    a. Less than 2
    b. Equal to 2
    c. More than 2 but less than 5
    d. Equal to 5
    e. More than 5
2. Credit scores
    a. P
    b. G
    c. E
3. Monthly Usage
    a. Between 0 and 1000
    b. More than 1000

A table with these inputs should help account for any possible case and the expected outcome

Invalid Input data:

While invalid data should never be entered into the system it can ensure that accidental or nefarious attempts do not affect the system.

Test Cases:

1. Credit scores that are not P, G, or E
2. Negative or extremely high values of years
3. Negative or extremely high credit usage

4. Non-numeric input for years as a customer
5. Non-numeric input for credit usage

Much like the Boundary testing these are inputs that wont likely happen, but a system that accepts them without question may cause major problems.

Summary:

In summation, testing needs to be both broad and focused. Each of these 4 types of tests will achieve a high coverage of the system that should encapsulate any potential scenarios of input including ones that should never happen. While there is overlap between many of these testing types that is not an issue in my opinion. The order in which these tests should happen is not the order in which they are listed. The first thing performed should be the decision table, this is because it will ensure we understand what the outcome of any type of input should be. I would then move to input space partitioning testing; this is to ensure that the normal test cases run without issue. There is no point fixing extreme cases first if the normal cases cause issues. After that I would move to Boundary testing. This will ensure that extreme cases that don't commonly happen are accounted for and perform as expected. Finally invalid input testing should be performed. Invalid input testing will ensure that even inputting data that should never be inputted is correctly handled. This should be the final step as ensuring correctly entered data is handled before dealing with data that may never be inputted.