Software Quality

Tags

Overview:

- [x] ~~Software quality and how to get to it~~
- [x] ~~Test Driven Development~~
  - [x] ~~White box testing~~
  - [ ] Black box testing

Relevance

- Leads to fewer mistakes during the software development process
- A buggy or poorly tested software can lead to a loss of trust and money
- Software Quality is important because:
  - Reputation
  - Cost of Production and Maintenance
  - Potential loss of Software Certification
  - Potential loss of Organisational Certification
  - Legality

    - Moral/ethical codes of practise

There are two main things to consider when testing your software for quality :

1. **Objective:** What we discussed above, which is the conforming to industry standards and protocols for certifications such as ISO / IES25010
2. **Subjective:** Is the software meeting your clients / users needs, e.g. in a game is the game actually fun to play?

Steps Towards Software Quality

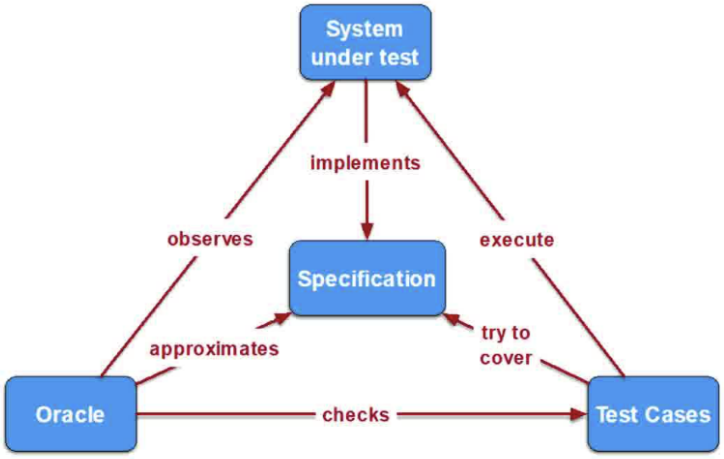
- Use a standard development process
- Use a coding standard
    - Compliance with industry standard (ISO)
    - Consistent code quality
    - Secure from the start
    - Reduce development cost and accelerate time to market
- Define and monitor metrics
    - High complexity leads to higher number of defects
- Identify and remove defects
    - Conduct manual reviews
    - Use Testing

Testing

- You want to write the minimum amounts of tests which cover the maximum number of requirements of your program

The testing process:

- A system under test needs to meet its specifications. A specification can be specific i.e. a function that either works or doesn't or it can be a user story of how the user would interact with the system.
- How do you know if your system is passing these specifications? The **Oracle;** the standard setting entity observes where the system is passing tests and where it is failing



White Box Testing :

- Useful for when you have full access Software internals:
  - The source code
  - runtime state
  - you can keep track of executions

- Unlike Black Box testing which focuses on input and outflows and tests functionality of the system without knowledge of the internal software, white box testing requires knowledge of the code, its structure and implementation

- The core principles of white box testing are:

1. **Code Coverage:**

- Statement Coverage: Ensures that every statements in the code has been executed at least once. **example:** If a function adds two numbers together then the test case should include lines that ensures all statements of the function are executed.
- Branch Coverage: Ensures all branches (if-else statements) has been executed at least once.
- Path Coverage: Involves testing all possible paths through the code. This can be much more extensive tan branch coverage because it considers multiple branches.

2. **Logical Paths:**

- Testing should cover all logical paths through the code, **example :** If a function checks for the correct combination of the username and password, it is essential to test all paths, where the user enters the correct username, but wrong password and vice versa

3. **Unit Testing:**

- White box testing is often preformed at the unit level where each individual component can be tested. This can include input validation, error handling checks

4. **Loop Testing:**

- Special attention is given to loops during white box testing, to ensure all Paths are covered. Loops can be tested at different iterations:

- **Zero iterations:** To ensure that the loop exits properly and is not stuck in an infinite loop
- **One Iteration:** ensuring the loop performs a single pass
- **Multiple iterations**

Example:

- Take a look at the code below, which calculates the factorial for an integer, which tests would ensure complete coverage :

```
def factorial(n):
    if n < 0:
        return "Error: Negative input not allowed"
    elif n == 0:
        return 1
    else:
        result = 1
        for i in range(1, n + 1):
            result = result * i
        return result
```

Answer: Tests when n =

- less than 0 (negative number)
- n = 0
- A range of positive integers, e.g. pass 5 and verify the output is 120

Black Box Testing:

What is it?

Black box testing aims to test the program without knowing its internal code structure, it focuses on the outputs generated in response to specific inputs. Black box testing ensures system requirements are met given a range of inputs.

Functional Testing

- This involves testing the business requirements of the system, given valid inputs.
- **example:** A system used to book flights, will have functional tests that test a valid output (successful flight booking) given that the user enters correct card details and passport information. These tests will also test the production of an error message if the user does not provide all the required information

Non-Functional Testing

- Non functional testing can involve testing the components related other than just the correct output of the system, Non-functional tests involve:
  - Performance Testing
  - Usability Testing
  - Reliability testing

Regression Testing

- This is used to confirm a recent change has not significantly altered the overall use of the system, such as adding a new feature

Boundary Value Analysis

- This involves testing the extreme ranges of the inputs because these extreme ranges are where most errors occur
- **example:** If a function takes inputs from ranges 1 - 100, then the tests should test inputs 0, 1, 100 and 101

Equivalence Partitioning Method

- Systems where there can be an infinite number of ways things can interact are hard to write tests for. **example**: Imagine the different apps on your phone, you cannot test how the phone will react to each different combination of apps, that's just too many variables
- Here we can use **Equivalence Partitioning Method **
- The basis of Equivalence Partitioning is to split data into valid and invalid partitions.
- **Example:** For a field that accepts month values (1 - 12)
  - valid inputs: 1, 5, 7,
  - invalid inputs : 13, -1, 1.5

1. **Decompose the program to its functional units**

- Divide into smaller units
  - Possible to generate more rigorous test cases
  - Easier to debug

  example:
  - Imagine splitting a large function to smaller functions, it is much easier to test the mini functions then to test the the large functions as a whole
  - Or dividing a large JAVA application into its core modules / packages

2. **Identify input and output parameters for these units**

- For some systems this can be straight forward e.g. :
    - A function which takes in a finite number of inputs, and

3. **For each input:**
    a. **Identify its limits**
        - What are the valid inputs and invalid inputs, e.g. in a function that takes integers as its parameters, Strings or Floats will be invalid inputs
    b. **Define partitions - value categories**
        - In the example above where we calculated the factorial of a number n; Invalid inputs were anything below 0 and if we had set an upper limit such that the loop does not run for a very long time for large numbers, anything above this number would be invalid. This gives us the range of invalid inputs
    c. **identify constraints between categories**
    d. **Write test specifications**

Example of Black Box Testing

Let's consider a simple online form that accepts user information for registration. The form includes fields for name, age, and email.

Test cases might include:

- **Valid Input Test**: Input "John Doe", 25, "john@example.com" and expect successful registration.
- **Invalid Email Test**: Input "John Doe", 25, "john" and expect a message that the email is invalid.
- **Boundary Age Test**: Input "John Doe", 0, "john@example.com" and expect an error that age must be at least 1.

- **Usability Test**: Check if the form provides explanatory error messages and verifies if the fields have helpful placeholders or tips.