Testing

Software Testing

Software Testing involves much more than just a check on whether something works or not.

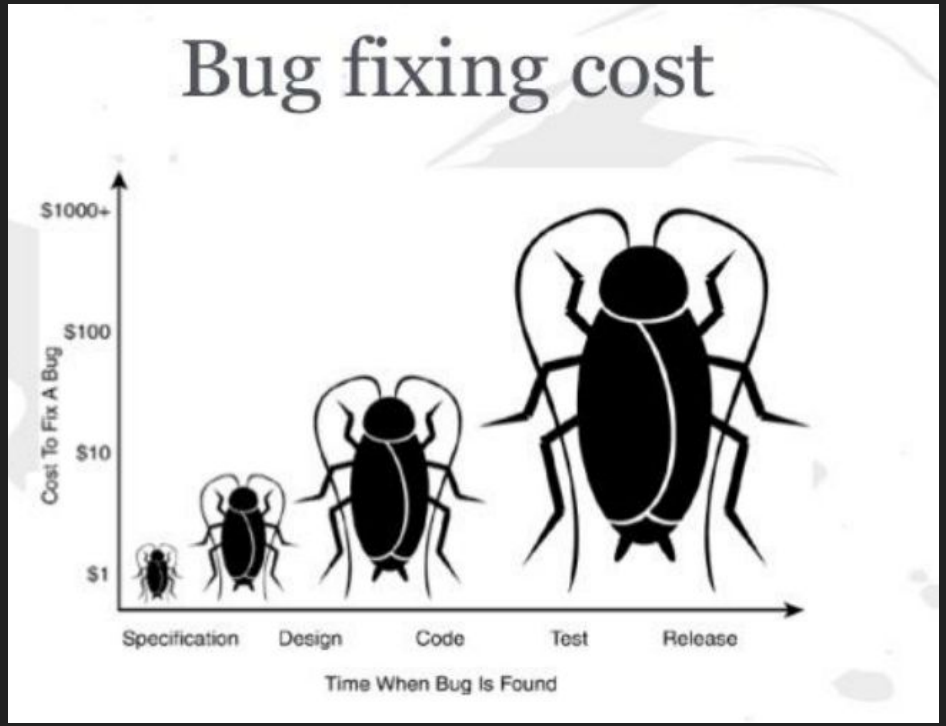
The testing process can involve

- Evaluating software
- Making suggestions for future design
- Finding inefficiencies in code
- Discovering unusual situations where code doesn’t work
- Considering the user’s experience
- Making suggestions for look and feel of a program or app

Software Testing

The software tester has a lot of responsibility.

The cost of fixing bugs increases hugely depending on the stage of software development



Software Testing

Recent changes to software development methodologies are reflected in a changing role for software testers.

In the traditional Waterfall model the testing process takes place at a particular point in the software development cycle.

In Agile development testers are involved from the start, in constant dialogue with clients, designers and project managers.

Unit testing is often carried out during the create stage of the development process. Other tests must be carried out after the software has been completed.

There is also Test Driven Development (TDD) where test cases are written before any code.

Windows XP - there were 65,000 bugs found before the product went to market

Boeing - problem where nose of aircraft went to high was overcorrected in a solution which was not properly tested. Two planes crashed.

Test Data

Test data are conditions/inputs that produce a known result and will be used to test each part of the software.

Test data is prepared in advance of testing and includes normal, incorrect, extreme and boundary values.

The software must be tested to ensure that it responds correctly to these types of test data.

Unit Testing

A unit refers to the smallest component of a program that can be tested e.g. a loop or a function.

Unit tests are typically automated tests written and run by software developers. Automated testing is a much faster process than manually testing and so more tests can be carried out.

Unit tests use test cases (similar to those you looked at in Evaluating Models).

Unit Testing

A unit test is another piece of code (usually a function itself), which passes arguments to the function to be tested. The unit test calculates the value that the function expected to return, the “expected” value. The function returns the “actual” value. The unit test then checks whether the actual value is equal to the expected value and records the result.

Code from the function should never be copied and pasted into the unit test. The code in the unit test should be written independently. This is to ensure errors in the code are identified using the test.

Benefits of Unit Testing

Compared to manual testing, the benefits of unit testing include:

- It is fast
- It allows errors to be identified easily
- It makes it easy for members of a team to check each others code
- It provides documentation
- It provides confidence that the code works correctly

Unit Testing v Integration Testing

Unit testing tests the smallest element of the system. It could be testing a single function.

Integration testing is meant to show that the system's components work with one another.

The goal is to find problems when components that work correctly are put together.

Functional Testing

Functional testing involves testing each of the functions or features of the system. It includes:

- Whether the function produces the expected result
- Usability - how easy it is for the user to navigate
- Accessibility - how easily a user can access the function
- Error handling - whether appropriate error messages are displayed when they are needed.
- Security - if the function is as secure as it is meant to be.

Functional Testing

Functional Testing begins with identifying the functions in the system and then for each function:

- Creating test data
- Determining the expected output from test data
- Carrying out the test
- Comparing actual result with expected result
- Checking if the function worked correctly
- Documenting results of the test

Functional / Non Functional Testing

Functional testing tests the system’s functionality and behaviour.

Non - functional testing tests aspects of the system such as usability, reliability, maintainability and performance.

System Testing

System testing tests the entire system. The aim of system testing is to ensure that the software meets the specification agreed with the client and the users.

System testing can be broken into alpha testing and beta testing.

Alpha testing is carried out by a test team in house. It aims to test as many paths through the system as possible. The purpose is to discover and fix problems before the system is tested by users.

System Testing

Beta testing is when an early (beta version) version of the software is sent to potential users who agree to use the software and report any faults.

Users perform tests that are typical of what would take place in real life conditions. They record problems and notify the developers so that they can be fixed.

Real users may try to do something that a developer did not anticipate.

Black box v White box Testing

Black box testing is carried out independently of the code used in the program.

It looks at the program specification and creates a set of test data that covers all the inputs, outputs and program functions.

It basically checks if the program works or doesn’t.

System testing is a form of Black box testing.

White box testing depends on the code logic.

Tests are devised which test each path through the code at least once.

Unit Testing is a form of White box testing.

Regression Testing

Regression Testing is testing to confirm that code changes have not affecting existing features.

It is carried out using test cases that worked before the changes to ensure existing functionality still works as expected.

This type of testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works after changes are made.

Evaluation

Once a project is completed, the user now needs to test every aspect of the software to make sure it does what it is supposed to do.

It will be evaluated against the original specification document

This stage can be called **Acceptance Testing.**

Worth a read

https://www.brighthubpm.com/monitoring-projects/104432-software-testing-a-simple-five-stage-model/

Creating your own Test Cases

When we create our own code, we should always test it.

We do this by testing a range of values or inputs ro cover all of the possible values that could be entered.

Test Cases

A test case uses a set of stated conditions/inputs that are tested. A typical test case consists of:

Test ID - unique reference

Test Description - purpose of the test

Expected Result - what the output should be

Actual Result - what the output is

Pass/Fail - If the expected result is the same as the actual result, the test passed.

Example:Program to calculate the volume of a sphere

We will use the formula

$V = (4/3)\pi r^3$

(For this problem $\pi = 3.14$.)

Here is the code:

```
radius = float(input("Enter radius: "))
volume = (4 /3) * 3.14 * radius ** 3
print("The volume is: ", volume)
```

ID	Description	Test data	Expected result	Actual result	Passed Y or N
1	To test a regular value for a radius of a sphere.	4	267.9466	267.9466	Y
2	To test a very large value for a radius of a sphere.	4,000,000	2.6794... e+20	2.6794... e+20	Y
3	To test a very small value.	0.00000000001	4.1866... e-33	4.1866... e-33	Y
4	To test a negative number.	-1	Error	-4.1866	N

Example: Driver License

We previously created a program to work out whether a user is eligible to apply for a driving license.

Here is the code:

```
age=int(input("Please enter your age: "))
if age >= 17:
    print("You are eligible")
else:
    print("You are not eligible")
```

This code uses selection (if/elif/else) so each branch should be tested.

We should test extreme values (very high and very low)

Most importantly, we should check boundary values - in this case numbers near 17.

Task:

Create a program to find the average of five heights.

Your program should:

- Ask the user to input five heights (one at a time)
- Calculate the average of the five heights
- Display the answer to the user

Test your program using the following test table

Input Values (cm)					Expected Output	Actual Output
h1	**h2**	**h3**	**h4**	**h5**		
150	160	170	180	190	170.0 cm	
190	172	172	178	187	179.8 cm	
171	175	169	185	178	175.0 cm	

The users height is 173 cm

Modify your program to compare the users height to the average height.

Output to the user if they are greater than, less than or equal to the average height.

Test each of these branches.