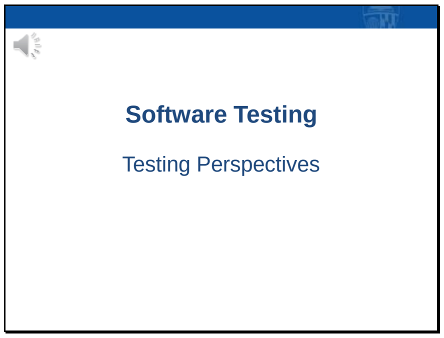
1

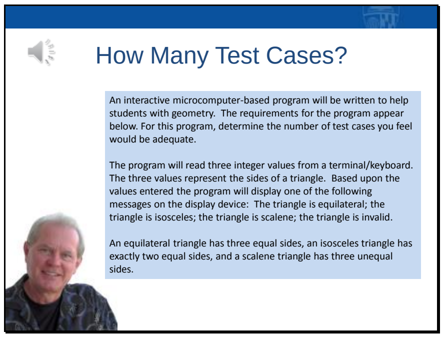


Software Testing

Testing Perspectives

In this lecture, I'm going to discuss several things about the overall topic of software testing.

2



How Many Test Cases?

An interactive microcomputer-based program will be written to help students with geometry. The requirements for the program appear below. For this program, determine the number of test cases you feel would be adequate.

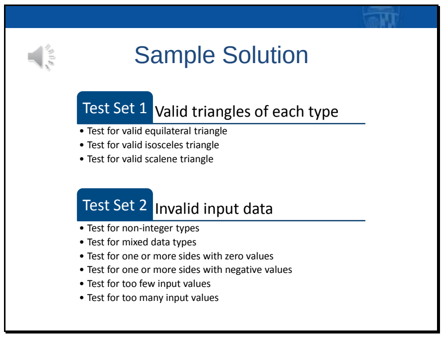
The program will read three integer values from a terminal/keyboard. The three values represent the sides of a triangle. Based upon the values entered the program will display one of the following messages on the display device: The triangle is equilateral; the triangle is isosceles; the triangle is scalene; the triangle is invalid.

An equilateral triangle has three equal sides, an isosceles triangle has exactly two equal sides, and a scalene triangle has three unequal sides.

And…before I even get started…I'm going to ask you to do a little testing exercise.

This describes requirements for a simple software product. What I'd like you to do is to tell me how many test cases you would use to test this product. Please pause the lecture now, and resume it when you have completed the exercise.

3



Sample Solution

Test Set 1 Valid triangles of each type

- Test for valid equilateral triangle
- Test for valid isosceles triangle
- Test for valid scalene triangle

Test Set 2 Invalid input data

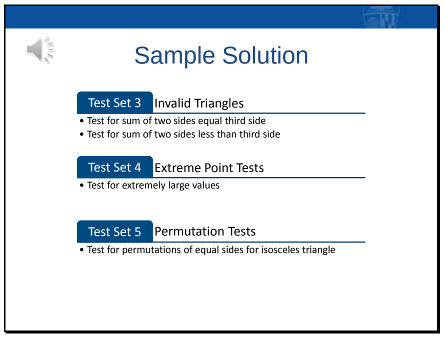
- Test for non-integer types
- Test for mixed data types
- Test for one or more sides with zero values
- Test for one or more sides with negative values
- Test for too few input values
- Test for too many input values

The next two slides contain a possible solution. When I give this exercise in my testing seminars, the rough average number of test cases that people come up with is between five and seven. I come up with a lot more.

Now…I've grouped my test cases into test sets in order to illustrate the concept of the test set. Not everybody uses the idea of a test set, but I have found it to be very helpful.

My first test set consists of tests associated with valid triangles of each type…so, I'll have three test cases. The second test set consists of test cases that will have invalid input data. Each bullet here consists of one or more test cases. I'll have one or more test cases for non-integer types…maybe a test case that has floating point numbers, a test case that has alphabetic values, and a test case that uses various symbols. I'll also have at least one test case with mixed data types…and so forth. The last two items will depend somewhat on the user interface…so they may change or be eliminated as a function of that…but I want to include them in my planning.

4



Sample Solution

Test Set 3 Invalid Triangles
- Test for sum of two sides equal third side
- Test for sum of two sides less than third side

Test Set 4 Extreme Point Tests
- Test for extremely large values

Test Set 5 Permutation Tests
- Test for permutations of equal sides for isosceles triangle

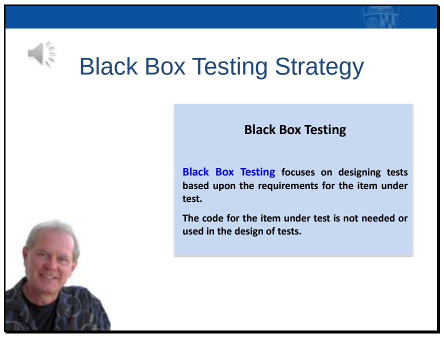
Here are three more test sets. Again…each bullet corresponds to at least one test case.

Test set three is a very powerful one…because any three sides does not a triangle make. There's a rule in mathematics that says for three sides to form a valid triangle, the sum of any two sides must be strictly greater than the third side. So…if I used values 1, 2, and 3, they would not make a valid triangle. Similarly, values in which the sum of any two sides is less than the third would also make an invalid triangle.

I'd also test for extremely large numbers. For example…an equilateral triangle with side values of 999,999. Depending on the computing platform, programming language, and coding decisions that were made for the software product…the largest integer value that could be handled might be smaller than these values.

And…I'd also probably test for different permutations of the equal sides for isosceles triangles.

So…as you can see…I come up with a lot more than 5-7 test cases.

5 

Black Box Testing Strategy

Black Box Testing

Black Box Testing focuses on designing tests based upon the requirements for the item under test.

The code for the item under test is not needed or used in the design of tests.

Now…the type of testing we just did is called black box testing. Black box testing is an approach or strategy that focuses on designing test cases based on the requirements for a product. The code isn't used in the design of the tests…in fact, the code may not yet even be developed.

There are three key drivers to how effective a black box testing effort will be: the test design techniques used, the quality of the requirements, and the level of subject matter expertise there is on the test design team.

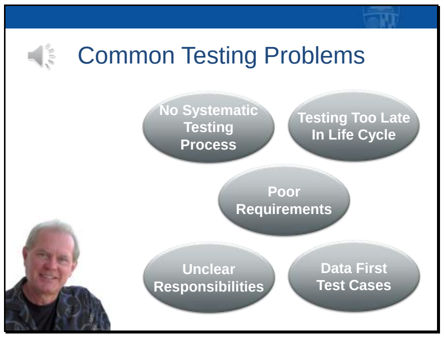
In the exercise, I used the concept of test sets in conjunction with a test design technique in which I started with conformance-directed tests, then fault-directed tests. The fault-directed tests were organized into invalid data, business rules, and extreme point tests. This is what helped me to define such a rich set of test cases.

The quality of the requirements in the first place is another driver that impacts the potential quality of the test effort. The requirements were incomplete. They didn't specify any upper bounds on the side values, and a critical business rule was missing…so there's a risk that testers may miss some very important tests.

In my seminars, I always ask participants if testers should be expected to know the business rules…in our example the business rules would be that the sum of any two sides is strictly greater than the third. Participant responses to this question vary quite a bit. Then I ask them to think about the testers who test the software in their organization and whether they have significant knowledge of the business rules. The response to this question is usually a lot narrower…the majority say no. Which brings us to the third driver.

The third driver is the level of subject matter expertise on the part of the test designers. A high level of subject matter expertise would likely test the missing business rule…a low level of expertise would not.

So…the combination of incomplete requirements and lack of subject matter expertise present a significant project risk when it comes to testing.

6 

Common Testing Problems

- No Systematic Testing Process
- Testing Too Late In Life Cycle
- Poor Requirements
- Unclear Responsibilities
- Data First Test Cases

I'd like to wrap up this lecture by summarizing some common testing problems that organizations experience in practice.

One very common problem I see quite often is the lack of a systematic testing process. Testing is often ad hoc. When testers are challenged to defend how they came up with number of test cases and overall testing effort required to test a product, they often don't have a reasonable response because they didn't use a systematic method for estimating how many tests are required and why. One of the things it's important to strive for is repeatability and defensibility.

A second problem that's also quite prevalent is that testing activities often take place too late in the life cycle…and some testing efforts don't get started until after the software is coded. That's an expensive mistake…and it can lead to more rework than is necessary.

I've already talked about poor requirements in the context of that little exercise you did. Now, if we combine this with the scenario that we didn't do that testing exercise until after the software was constructed, you can see the rework impact very clearly. If the business rule was missing from the requirement there's an excellent chance it's missing from the code.

There's another problem that I see a lot in organizations that have independent test groups. In those situations, it’s often unclear what responsibilities the developers should have for testing and what responsibilities the independent testers should have. The result is often a less thorough level of testing than there should be.

Last but not least, is what I like to call data-first or bottom-up test case development. When I give that testing exercise in my in-person seminars, I often see participants start by writing down data values for the sides of a triangle. Some even try to sketch out what the code would look like. This is not the best way to design tests. The data should come last. It’s much more effective to estimate how many test cases we might need, and what types of test cases, and then come up with the data as a last step. Recall that I didn’t use a single data value in my sample exercise solution. My experience has shown that the bottom-up approach misses a lot of potentially powerful tests, and also can result in what I like to call the over-test, under-test syndrome. That means that a large number of tests may be generated, but many are redundant and don’t add any value.