Exercise: Analyzing the Triangle Problem

(Allow 15 to 20 minutes for this exercise.)

Consider the following software program. It reads three input data values. These values represent the three lengths of the sides of a triangle. The purpose of this program is to display a message which states whether the triangle is scalene (i.e., no two sides are equal), isosceles (two sides equal) or equilateral (all sides equal). As software goes, this is very low in complexity. How do we test it? This challenge was posited by Jerry Weinberg, and described by Glen Myers in his classic book, "The Art of Software Testing".

Develop a list of test cases for this triangle problem. The table format below is convenient for listing your test cases, though you do not have to follow it. I have provided the first few test cases to help you get started:

Test Case #	**Condition Being Tested**[**]	**Input Data Values**[*]			**Expected Result**
		i	**j**	**k**	
1.	Three valid equal sides	7	7	7	Equilateral triangle
2.	Two equal valid sides	5	5	3	Isosceles triangle
3.	Two equal valid sides; third side invalid	8	8	%	Error message: invalid input data value

[*] Where i, j, and k are the names of the three sides of the triangle.

[**] You may find the word "condition" to be vague. I mean the test requirement; the circumstance or situation which we want the test case to check.

Lessons and Conclusions from the Triangle Problem

In experiments at IBM and other organizations, researchers asked experienced systems professionals to identify an adequate set of test cases for the triangle problem. The researchers gave the participants in these experiments unlimited time for this task. The results of these experiments have been judged to be deplorable: on average, people listed only about one third of the test cases needed for an adequate test. In his book, "*Software Testing: A Craftsman's Approach*", Paul Jorgensen lists about 185 test cases for this problem. How many did you identify?

I have problems with this "deplorable" interpretation. Adequacy must be assessed with regard to context. For the functionality specified in the exercise, I could argue that these four test cases are sufficient to test adequately:

Test Case	Input Values	Expected Results
1.	3, 3, 3	Equilateral Triangle
2.	3, 3, 2	Isosceles Triangle
3.	3, 4, 5	Scalene Triangle
4.	3, 3, ?	Invalid Input

There are possible defects that would not be detected by this set of four test cases, however, so other people may advocate a more thorough test. Kent Beck has written that he thinks that six test cases would be adequate for his implementation of the triangle problem, and compares this to the 65 test cases that Bob Binder describes. This sounds like a bad joke – ask four different experts, get four different answers which range from four to 165 test cases – and four consulting bills.

What's the right answer? The answer is that there is no single answer for all circumstances, as the most appropriate answer depends on the context. There is a long history of disservice to business and industry by testers who claim that "adequate" equals "thorough". As we saw in Chapter 1, the question: "What is the correct set of test cases?", can only be answered in terms of the perceived risk in the situation and the amount of effort we are willing to expend to avoid that risk. In other words, the right amount of testing depends on the trade-off between the risk and economics. If the risk is sufficiently low or the cost of testing is sufficiently high, the minimal set of four test cases is adequate. You also may feel that I am playing games by disclosing the context after I asked you to solve the problem. You first need to know the context before you can solve the triangle problem.

The triangle experiments, where the intuitive approach identified only about one third of the test cases, do *not* prove that the intuitive approach doesn't work. The yield of test cases from systematic analysis can be just as low if the information, time and skills are not available for effective analysis. The experiments do show that test coverage may be lower than we realize and we need: important test cases are often missed with the intuitive approach.

What test cases do people miss with the triangle problem? In my observation, the most overlooked test case is one with a long side and a couple of short flaps, such as an input combination of [1, 3 and 1]. These three lines do not close to form a triangle: to qualify, the sum of any two sides must be greater than the third. If the system concludes that [1, 3 and 1] is an isosceles triangle instead of rejecting this combination as invalid, the overlooked test case would fail. More than half of the participants in the experiments do not identify this test case.

The mis-design of test cases is another source of danger. The following example of a mis-designed test case is based on an ambiguity about triangles. Dictionaries define a triangle as a *body* which is *enclosed* by three straight lines. The word "enclosed" implies that all six ends of the three lines must meet. So this definition addresses whether the triangle closes and helps us see the [1, 3 and 1] scenario which was previously overlooked. However, there is an ambiguity in the word "body". If a body is allowed to have no area, then a test case with an input combination of [3, 6 and 3] should produce a message saying this triangle is isosceles, and inputs of [1, 2 and 3] produce one saying it is scalene.

Many people argue that these test results are ridiculous. In their view, test cases with input data values of [1, 2 and 3] and [3, 6 and 3] represent flat lines, not triangles, and thus the system is working correctly if these input combinations are rejected. These people claim that the word "body" implies a contained area which must always be greater than zero. Since testing is a trade-off of risk and effort, validating this requirement (that the area must exceed zero) is justified only if the risk associated with the triangle is sufficiently high. If the system includes the capability to generate a "Not a triangle" error message in response to invalid inputs, then the corresponding test cases are less likely to be overlooked: the presence of the error message prompts us to ask what circumstances trigger it.

Another example of an overlooked test case is more esoteric, but shows the lengths we go to in test case design if the risk justifies the effort. The software for the NASA space shuttle computes the shape of triangles as part of its orbital navigation. (Most image processing software works with polygons, of which triangles are the simplest form.) The NASA mission is life-critical: if the orbital navigation is wrong the consequences could be severe. The triangles computed on the space shuttle are curved—they are based on the shape of the Earth's surface. Most participants in the triangle experiments assume a flat surface without really thinking about it.

On a curved surface like the Earth's, we could have an input combination of [1, 3 and 1] or even [1, 300 and 1], with the long side curving back on itself as it wraps around the world. It helps to visualize the shape of this "triangle": it looks like a long narrow strip of peel cut from the top of an orange. Starting at the point where the stem connects on top of the orange, the boundary of the piece of peel follows a longitudinal line vertically down the side of the orange for one unit of length. Then the second side of the triangle proceeds horizontally and circles the orange for another ten units. The third side proceeds

vertically back to the top of the orange, closing and completing the triangle. If the three lines are all “straight” within their geometry (which is called Riemannian geometry), and if all six ends of the lines meet, then by definition this [1, 3 and 1] combination is a valid isosceles triangle on the curved surface.

If we allow for the possibility of triangles on a curved surface such as a sphere, then the only correct result for any possible set of only three inputs is: "Indeterminate -- not enough data to compute." We will need additional information which indicates whether the six ends meet or not: not every [1, 3 and 1] combination of three “straight” lines forms a triangle on the curved surface. To determine if the six ends meet, we might also need to know the shape of the surface – not all curved surfaces are spheres, and the rules which determine if a triangle closes on a spherical surface do not hold true for all curved surfaces. Now you know why NASA employs lots of PhDs in mathematics.

GRAPH GOES HERE TO SHOW A GLOBE-ENCIRCLING [1, 3, 1] TRIANGLE.

If you are not feeling a sense of “gotcha” by now, you should be. I could have introduced the last section by adding the possibility of triangles on curved surfaces to the previously stated context, and then asking how the new information affects your test cases. Instead I implied that you were derelict if you overlooked the nuances of curved surfaces. Not telling you the full context of a problem, and then expecting you to see every context-dependent nuance, is unrealistic and unfair. We cannot expect a tester to be omniscient and find all the significant test cases regardless of the information she has. Especially when the mission is life-critical, if the triangles can occur on curved surfaces then the project manager is responsible for ensuring the tester has this information. Nevertheless, the tester needs to prompt for information by asking the right questions to uncover it. Rightly or wrongly, usually she is held accountable for any inadvertent omissions in testing (not including deliberate omissions to meet time and resource limitations).

The ineffective testing of the triangle software arises from incomplete analysis, unrecognized assumptions and vague requirements. The main lesson from the triangle testing experiments is that intuitive testing usually is not enough. In the face of poorly defined requirements, though, intuitive testing may be the only approach we’re left with. We need to be able to recognize when intuitive testing is not sufficient to deliver good results. Not only do people see only about one third of the test cases in the triangle example using intuitive, unstructured techniques, most of these people feel that they had uncovered 80% or more of the test cases. In other words, an over-confidence in the completeness of the testing accompanies the lack of coverage.

However, mindlessly following the test techniques described in this book, and applying them to typical imperfect specifications, may not be any better than intuitive testing. The test coverage for the triangle software (and by extension, any software), increases significantly with test case design, but only if three conditions are met:

1. The testers have a clear, correct, complete and unambiguous feature specification, plus access to one or more knowledgeable persons who can authoritatively answer questions about the feature.
2. The testers are methodically trained to analyze features and derive test cases, using the techniques which are explained in this book.
3. The testers have sufficient time to do the work. The testers in the triangle experiment were given unlimited time, so that having the time available is not sufficient by itself to ensure adequate coverage. And in reality, many testers are pressured by tight deadlines and resource limits.

When these conditions are met, the test coverage, the count of detected bugs (weighted by severity), and the test efficiency (bugs found per hour) often increase dramatically. Admittedly, these benefits are not free -- the test preparation effort also increases. But my experience is that the extra effort pays off on most projects.