APPENDIX 1.

ANALYZING THE TRIANGLE PROBLEM

ANALYZING THE TRIANGLE PROBLEM

"No book on software testing is complete without a discussion of the triangle problem." Ed Kit. (This is meant as a joke, because this problem has been thoroughly worked and re-worked over the years. Nevertheless, this problem still retains its educational value.)

At least six self-proclaimed experts have developed answers for the triangle problem (including this self-proclaimed expert, the author), and each answer is different. When experts do not agree on the answer to a seemingly simple problem, perhaps this tells us something about the state of the art of testing.

The triangle problem was mentioned in Chapter III. Using intuitive, off-the-cuff techniques, systems professionals were able to identify only about half the test cases for the triangle problem. This problem provides an opportunity to see how the functional analysis, path analysis, boundary value and risk assessment methods can be applied.

Re-Statement of the Problem

Develop a set of test cases to adequately test a program, which works as follows:

> The program reads three input numbers that represent the lengths of the three sides of a triangle.
>
> Based on these three input values, the program determines whether the triangle is scalene (that is, it has three unequal sides), isosceles (two equal sides), or equilateral (three equal sides).
>
> The program displays the result on the screen.

Glen Meyers originally described this problem in his book, "The Art of Software Testing".

Exercise (Allow 15 to 20 minutes for this exercise.)

There is no one correct theoretical answer to any testing problem. The answer depends on the assumptions and judgement calls which the test planner makes.

Review and critique the following set of test cases for the triangle problem. How do you suggest that we improve them?

Suggested Answer

(A) Minimal Test

For the functionality specified in the exercise, some people may argue that a total of four test cases like the following are sufficient to test adequately. For example:

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.

The question: "which one is the correct or more appropriate 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.

The right amount of testing depends on the trade-off between the risk and economics.

If the risk is fairly low, the above minimal set of four test cases may be adequate.

If the risk is high, a more extensive test will be needed. Two more extensive test plans follow on the next several pages.

Suggested Answer

(B) Automated Test

Given that hardware today is so fast and cheap it is virtually free, an innovative tester has written the following test program to exhaustively and automatically test the triangle program.

Review this test program and form an opinion whether this automated testing is a good idea. If automation is worthwhile, is the test program acceptable as written?

```
for i = minval to maxval by increment do
        for j = minval to maxval by increment do
                for k = minval to maxval by increment do
                        enter { i, j, k }
                        capture { result }
                        if [ i = j = k ] and result = equilateral
                        or if [ ( i = j ) and ( i not = k ) and result = isosceles ]
                        or if [ ( j = k ) and ( j not = i ) and result = isosceles ]
                        or if [ ( k = i ) and ( k not = j ) and result = isosceles ]
                        or if [ ( i not = j ) and ( j not = k ) and ( k not = i )
                                                        and result = scalene ]
                                then ok
                                else write error_log [ i, j, k, result ]
                end
        end
end
```

In this test code, minval and maxval are set to the minimum and maximum valid input data values, and increment is the smallest numerical increment recognized by the triangle program (e.g., 0.01). enter is the name of a function which drives the triangle software by entering values i, j and k. capture is the name of a function which captures the result of the triangle software computation. error_log is the accumulated list of defects found in testing.

ANALYZING THE TRIANGLE PROBLEM
(Continued)

Is the above automated testing program effective? No -- for two reasons. First, many equivalent (and thus redundant) test cases are mindlessly included. For example, if we are willing to assume that the system behaves in the same if the input data is (2, 2, 2) or is (3, 3, 3), then we do not have to test both of these situations. Second, there are many negative test cases (i.e., ones based on invalid input data) which have not been identified and included in this automated test program. See the next sections for examples of negative test cases.

ANALYZING THE TRIANGLE PROBLEM
(Continued)

Do you consider the following automated Monte Carlo test or monkey test to be worthwhile?

```
do while we_want_to_keep_testing
        i = random ( i )
        j = random ( j )
        k = random ( k )
        enter { i, j, k }
        capture { result }
        if [ i = j = k ] and result = equilateral
        or if [ ( i = j ) and ( i not = k ) and result = isosceles ]
        or if [ ( j = k ) and ( j not = i ) and result = isosceles ]
        or if [ ( k = i ) and ( k not = j ) and result = isosceles ]
        or if [ ( i not = j ) and ( j not = k ) and ( k not = i )
                                                and result = scalene ]
                then ok
                else write error_log [ i, j, k, result ]
end
```

where random is a random number generator which is used in each iteration to create new random values of i, j and k, with each value within the range from minval to maxval.

The answer is that this monkey test may be useful, provided that we_want_to_keep_testing is not set to a pointlessly high value, as an insurance policy in case the assumption of equivalence is not correct. I.e., the system does not in fact behave in the same if the input data is (2, 2, 2) or (3, 3, 3), or does not behave in the same way if the input data is (3, 4, 5) or (5, 12, 13), and so on.

In addition, negative test cases are still inadequately represented. This lack of negative test cases could be partly covered by extending the range of randomly generated numbers to below minval and above maxval.

Suggested Answer

(C) Extensive Test

Validation of the Specifications

The first step in any black-box (i.e., specification-based) test planning is to validate the specification itself. At least 50% of system errors originate in the specifications, and we cannot simply assume that the specs. are complete and correct as they are presented.

In this case, there is at least one omission in the specification -- it does not state whether or not negative triangles are acceptable. For example, should the expected valid response to a test case of a triangle with sides of -3, -5, and -3 respectively be "isosceles" or "error"?

If this system is designed for Ph.D. mathematicians, it might allow triangles with negative sides or even unreal (complex number) sides. If it is designed for fourth graders, though, negative triangles could be banned. This uncertainty needs to be resolved before we can complete our test case planning.

The Need for Assumptions

Assumptions are critical to test planning, since the set of test cases which are needed will vary based on which assumptions are made.

Assumptions need to be documented. The most dangerous assumptions are those which are not documented but which are simply assumed to be true. Simply assuming that "the world has to work this way" gets testers into trouble.

Assumptions also need to be reviewed and confirmed with the subject matter expert or the final authority on the application functionality. Undocumented assumptions tend to be embedded invisibly inside test plans and are not reviewed with the subject matter expert.

Definition of a Triangle

The most important assumption to be made is the definition of the word “triangle”. The client for the software may be a PhD mathematician (or a pre-school child), whose definition is different from our everyday, “common sense” definition.

Assumptions (continued)

The Oxford English Dictionary (OED) defines a triangle as a "figure (especially plane) bounded by three (especially straight) lines". Webster's Dictionary defines a triangle as "a polygon having three sides", and it further defines a polygon as "a closed plane figure bound by straight lines".

The Webster's definition is stricter and more limiting than the OED's, because Webster's requires a plane (flat) surface and straight lines. We will use the Webster's definition.

Other Important Assumptions

The other assumptions that have been made in developing the following list of test cases are:

- The input numbers may not have been previously edited and cannot be assumed to be "clean", i.e., invalid values such as "?" or "A" may occur in the input lengths of the sides of a triangle.

- Negative lengths are not valid for the sides of a triangle.

- Scientific notation is not acceptable. (E.g., lengths cannot be entered in the form 1.234E6, which stands for 1,234,000 in scientific notation, or in the form 1.234D6, which stands for the same number in double-precision arithmetic.)

- If three input numbers are valid lengths for the sides of a triangle, but do not together describe a triangle that can close, the input is invalid. (By closing, we mean a triangle where all six ends on the three sides can meet.)

  For example, the input numbers 1, 10 and 1 may each be valid lengths in themselves, but together they do not describe a triangle. Instead, they make up one long side (the one with length 10) and two short flaps.

ANALYZING THE TRIANGLE PROBLEM
(Continued)

Assumptions (continued)

- The smallest allowable input length is assumed to be 0.0001, and the largest allowable input length is assumed to be 99999.9999. (Ideally, the functional spec. in the exercise should have defined the smallest and largest allowable input numbers, so we would not need to make an assumption.) Note: sometimes the size limits are language- and machine dependent, for example, the limits may be based on the size of a "short" word on a particular platform.

- The precision of measurement is assumed to be 0.00005. In other words, differences between two numbers which are less than this cannot be recognized. All the numbers between 3.329951 and 3.330050, for example, are assumed to be rounded to the same value, 3.3300. Thus a triangle with side lengths of 3.329951, 3.330023 and 3.330050 will be accepted as equilateral.

- The three input numbers are expressed in the same units of measurement. For example, if the first number was expressed in feet but the last two were expressed in inches, the input set of numbers 1, 12 and 12 would define an equilateral triangle.

For extra credit: can you find any hidden assumptions that are buried in the following set of test cases for the triangle problem? Are any of these assumptions not likely to be obvious to the general reader? (If we are confident that an assumption will be obvious to everyone, it is not too dangerous for this assumption to be undocumented. The dangerous ones are those assumptions which are both undocumented and not obvious.)

Suggested Test Cases:

I. INPUT VALIDATION

1. Verify that invalid inputs are rejected as expected.

Confirm that only three valid numbers can be entered as the lengths of the three sides of the triangle. (These three numbers will be referred to here as i, j, and k.)

Three series of tests are needed, for i, j and k respectively:

(a) i is not valid

(i) i is not numeric

alpha; upper and lower case

numeric-alpha mix (e.g., "9A9")

non-alphanumeric (e.g., "?", blank)

characters which may have special meaning in the system environment (e.g., Escape, Enter, Backspace)

scientific notation (e.g., 1.2E6 or 4.56D7)

(ii) i is negative (i.e., less than zero)

(iii) i is zero

(iv) i is above the largest allowable input number (e.g., 999,999,999.99)

(v) the number of decimal places input for i exceeds the maximum allowed (e.g., i = 99.999 when only 2 places are allowed)

(vi) common or likely keyboard data entry "flubs" (errors), such as a double decimal point (i.e., "..")

(vii) i contains leading and embedded blanks (e.g., " 1 2 7" instead of "127")

(Note: in each of the above cases, an appropriate, clear error message should be displayed.)

The above test cases are negative test cases (i.e., test cases with unacceptable input values), for the first input number, i.

How many negative tests are enough? For example, many negative data values are not included in the above test cases, such as an input number containing an embedded blank like "1 0". Alternatively, are the above set of test cases too many, i.e., overkill? The answer to the question (of how many negative test cases are appropriate for the input length i), depends on two factors:

(i) The degree of risk inherent in the situation: if the triangle function carries a high level of averse consequences, logically more test cases are justified.

(ii) The perceived hostility of the input environment. If the quantity of incoming "garbage" is likely to be high, more tests are justified. Conversely, if the input lengths are not likely to contain many errors, then less tests are justified.

Can we utilize the concept of equivalence to trim the number of test cases to be executed, without materially increasing the risk that defects will not be detected?

If there is a front-end filter or edit which is reliable, then it is physically impossible for invalid values of i to be received. If we are willing to assume that the input has been filtered reliably, then the number of negative test cases needed here is zero.

If there is not a reliable front-end edit, but we are willing to assume that all inputs fall into only two equivalence classes (numeric and non-numeric), then only one negative test is needed. In other words, if we are willing to assume that if a non-numeric value for i such as "" is detected and rejected, then the system should behave in the same way for any other non-numeric input value (e.g., "X", "?"). In this situation (where there is one equivalence class for all possible non-numeric values), only one negative test case would be needed.*

Is the ratio of negative to positive test cases appropriate? Note that the number of negative test cases listed (in item 1(a) above, and in the remainder of items (1), (2) and (3) below), total more than 50% of all the test cases. The test cases listed in items 4 through 9 are primarily positive test cases, i.e., have acceptable input values.

The answer is the right mix of negative and positive test cases depends on the situation.

Only if the triangle software must have high reliability and robustness is the number of negative test cases likely to exceed the positive ones. Otherwise the positive test cases should outnumber the negative ones.

1. Verify that invalid inputs are rejected as expected (continued).

(b) j is not valid

The same types of tests are needed as for j as were needed for i.

(c) k is not valid

The same types of tests are needed as for k as were needed for i.

How much testing of the second and third input lengths, j and k, is appropriate?

If we can assume that the software engineer who developed the triangle software is reasonably competent, we can also assume with confidence that the software engineer would have provided one common subroutine to edit all of i, j and k at the time they are input.

In this situation, a comprehensive set of test cases for j and k, following the comprehensive set of tests for i, is unnecessary. The first set of test cases for i will already have extensively exercised the common edit routine, so re-testing for several values of j and k would be redundant. If the software works for i, it should also work for j and for k.

Only one additional test case is needed for j, and also one for k, to double-check that the software engineer at least remembered to call or access the common edit routine for j and k as well as for i.

A better approach (instead of many negative test cases for i, followed by only one each for j and k), is to spread the set of negative test cases equally among i, j and k.

However, if we are paranoid, we will be willing to assume nothing. Since this is a black box test, we do not know for sure how the software was constructed internally. Although it may be an incompetent practice, the software engineer may have written three separate input edit routines, one each for i, j and k. In this circumstance, the fact that the first set of tests for i work correctly will predict nothing about the behavior of j and k.

In other words, the answer to the question (of how much testing is enough for j and k), depends on what assumptions we are willing to make.

2. Verify that only and exactly three numbers can be entered successfully.

(a) Verify that no more than three input lengths can be entered, or if they can be, that the system queries as to which three of multiple input numbers represent the actual three sides of the triangle.

(b). Verify that a triangle computation cannot be performed if only two values are entered (representing only two sides of the triangle).

The user interface may well be designed in a way that makes it physically impossible to test for only two numbers or for four numbers to be entered. In this circumstance, the above tests are unnecessary. There is no point in testing for situations which we know cannot occur.

However, in the functional spec. (i.e., in the exercise narrative), we are not told anything about the user interface. So the prudent approach is to include these test cases, then delete them later if and when they are found to be unnecessary.

3. Verify that the triangle must close.

Verify that invalid triangles, where the sum of any two sides is not large enough to exceed the length of the third side, are detected and identified. In other words, check that the input is rejected with an appropriate error message if any of the three following conditions do <u>not</u> occur:

(a) i + j => k (and all three are numeric and positive)
  (i) i + j = k
  (ii) i + j > k

(b) j + k => i (and all three are numeric and positive)

  (The same two tests are needed as for i + j => k, above)

ANALYZING THE TRIANGLE PROBLEM
(Continued)

(c) k + i => j (and all three are numeric and positive)

(The same two tests are needed as for i + j => k, above)

Of all the test cases in this answer, the most overlooked by people who work through this triangle problem is the possibility that three numbers are entered which are independently valid but which together do not form a triangle. (For example, the numbers (1, 10 and 1) do not form a triangle.)

People miss this possibility either because they are mildly math-phobic, they fell asleep in high school geometry class, or do not work with triangles as part of their day-to-day work.

In any situation, there is a background level of knowledge that is needed in order to identify the test cases competently. In this particular situation, people do not test for a triangle that does not close simply because they do not recognize that the possibility exists.

What about the situation where a triangle does close, but is flat with zero area? This occurs with sets of input numbers like (3, 3 and 6), or (1, 2 and 3). Are these legitimate triangles or not?

Like the citizens of Lilliput (who went to war over which end of their boiled eggs to break open, the pointed end or the round end), we probably will have some people who say yes and others who say no.

In addition, we are assuming traditional Euclidean geometry, the kind we learned in high school, where triangles are drawn on surfaces which are flat. Ph. D. mathematicians also utilize other types of geometry, with names like Lobatchevsky and Riemannian geometries, which can have curved surfaces. With curved surfaces, a triangle could have lengths of (1, 10 (on a wrap-around surface), and 1), and still have all the ends meet so that the triangle closes.

ANALYZING THE TRIANGLE PROBLEM
(Continued)

If we wish to become really esoteric, there is another unstated assumption that we need to challenge. So far, we have assumed, without really thinking about it, a two-dimensional (flat) or three-dimensional universe. Physicists say that the physical universe has four dimensions, including time (and maybe more than four). A triangle exists in any two or three of these four dimensions, which means we need to ask not only if the edges of a triangle meet, but also when *they meet.*

II. TYPE OF TRIANGLE (internal flow logic)

The following tests are based on an assumed internal logical software design, as shown by the paths on the following graph. While there is no guarantee the internal software design actually conforms to this graph, the test cases based on the assumed internal flow logic are as follows:

4. Verify that equilateral triangles are processed correctly.

Verify that the displayed result is "Equilateral" when the three sides are equal (i = j = k).

5. Verify that isosceles triangles are processed correctly.

Verify that the displayed result is "Isosceles" when any two sides are equal. This requires three tests:

(a) i = j, and i not equal to k (and all three inputs are numeric and positive)

(b) j = k, and j not equal to i (and all three are numeric and positive)

(c) k = i, and k not equal to j (and all three are numeric and positive)

Do we need to test all three of these conditions, 5(a), 5(b) and 5(c). If we assume equivalence, in the sense that the order of data entry does not matter, then any one of these three will suffice. For example, we are willing to assume that if the input sequence [3,3,2] is recognized as isosceles, then so will be the sequences [3,2,3] and [2,3,3].

However, what about an input sequence like [2,2,10]? A paranoid person might wonder whether the software first determines that this triangle must be at least an isosceles one, based on i = j = 2, but then simply assumes that the same triangle cannot be equilateral because the third side is not equal (it has a length of 10). Thus, the final determination is that [2,2,10] is by default isosceles, which is an incorrect conclusion.

6. Verify that scalene triangles are processed correctly.

Verify that if a triangle has three unequal sides, the displayed result is "Scalene".

In addition to the above set of test cases for equilateral, isosceles and scalene triangles, do we need to test for any special types or shapes of triangles?

For example, do we need to test specifically for a right-angled triangle, or can we just assume that this is already covered by the earlier, more general test cases? Similarly, do we need to test for any special shape (for example a triangle with angles of 30^o, 60^o and 90^o)?

The answer to this question lies in an analysis of commonality versus differences. Is there anything distinct or unusual about a right-angled triangle, for example, that means the results of any test of a isosceles or scalene triangle would not also apply to the right-angled triangle? There is nothing apparently different -- if the software works for any samples of scalene and isosceles triangles, logically it should also work for the right-angled one.

If we have a hunch, though, that something strange may occur with the right-angled triangle, it is a good idea still to test it, no matter what rational logic tells us.

III. BOUNDARY VALUES

7. Verify that extreme lengths are processed correctly.

Verify that both very large and very small triangles are processed correctly. For example, if the smallest and longest allowable lengths of the sides are 0.0001 and 99999.9999 respectively, then try the following boundary values:

(a) i = j = k = 0.0001 (should be accepted as equilateral).

(b) i = j = k = 99999.9999 (should be accepted as equilateral).

(c) i = 0.0001; j = k = 99999.9999 (should be accepted as isosceles).

(d) i = 3.329951, j = 3.330023 and k = 3.330050 (should be accepted as equilateral).

These test cases may reveal defects caused by internal arithmetic overflows or rounding errors in the computations.

IV. ASSUMED LOGIC

8. Verify that the process exits and shuts down without any problems.

The functional specification provided in the exercise says nothing about the possibility of processing another triangle after the first one has been analyzed. It is likely, though, that this processing is intended to occur in an iterative cycle.

Presumably, the software should inquire at the end of processing one triangle whether the user wants to do another one or to shut down. This assumption can be politically dangerous, because we are now testing for features beyond what the functional spec. calls for. This assumption adds another test case (number 8 above).

For example, if after processing one triangle the system prompts: "Do you want to analyze another triangle?" and a response of "N" or "Escape" causes the system to freeze, this is a defect. If a response of "Y" or "y" does not initialize the system ready for the next set of inputs, this is a defect.

Also, test the result when the query is answered with an input the is not any of "Y", "y", "N" and "n".

9. Verify that the result is displayed on the screen in an acceptable manner.

The exercise states that the result of the process must be displayed "on the screen". If the program prints the right result but cannot correctly display it on the screen, it does not meet the required functionality. To test the screen display, we need additional information which is not provided in the exercise itself:

- *What GUI display mechanisms must be supported? (E.g., Windows, Mac, X-Windows, OS/2, CICS?)*
- *What screen technologies must be supported? (E.g., VGA, Super VGA, monochrome vs. color, different types of screen sizes?)*
- *What display format is acceptable?*

- *If a diagram of the triangle is displayed, calculated from the three numbers entered on the keyboard (which may be suitable for a children's math tutorial package), is its shape correct?*

- *What if the rapidity of the input data entry exceeds the system's ability to receive and process the data?*

- *What about testing for usability? For example, are the instructions and error messages meaningful to the typical user?*

- *If the delimiter among the input data fields is the Enter key, does it work as expected? What if it is hit multiple times? What if the tab is used instead of the enter key?*

- *What about testing for international users? What if the system messages are in German but most users speak only English?*

- *What about the software documentation? It is not good practice to deliver software without adequate documentation. The documentation test addresses these types of questions:*
  - *Does the documentation describe correctly how the software works?*
  - *Is the documentation readable and usable?*

* * * * * * * * * * * * *

There are other tests that may be appropriate too. For example, if the data entry of the three numbers can be performed either by keystrokes or mouse clicks, a reasonable test is to check that both input methods work correctly. *What other test cases can you find?*