Test Background

1. A software test adequacy criterion defines what properties of a program must be exercised to constitute a test whose successful execution implies no error in a tested program. These days the software testing literature contains two different notations related with test data adequacy criteria: test adequacy criteria as stopping rules(C1), and test data adequacy criteria as measurements(C2). These two notations of test data adequacy criteria are closely related. Applying only one of them will lead to some implication. C1 shows if whether enough testing has been done that it can be stopped. It only tells us if the test set (T) is adequate for testing the program (P) or not. This will not guarantee if all the statements are executed. C1 would not give any further information regarding the quality and the percentage of the program that is covered by the test set. Therefore, using only C1 is not enough. C2 shows the percentage of the code covered during the testing, which indicates the testing quality we need to measure. It provides more information about the testing. However, when we use only C2, we don't have indication on when to stop since there is no stopping criteria. Therefore, using only C2 is not sufficiently practical. As a result, having only one of C1 or C2 have their own effect, and is not good enough.

   We can construct a stopping rule Mr where M is an adequacy measurement, and r is a degree of adequacy. Mr states that a test set (T) is adequate if and only if the adequacy degree is greater than or equal to r. It is not easy to write such rules specially when P (set of programs) is large. As the program is larger, the complexity degree and the amount of the detail will take more time to write a stopping rule function. Adequate measurement M is a function of M (P x S x T) where S is a specification, and even the testing process focus on that specific part of the project, as the project is large most part will be related and connected with each other. In addition, selecting the right measurement criterion for large project will be a bit complicated. Therefore, writing the Mr function will be more complicated as the project is larger.

2. Statement coverage is a white box testing technique, which involves the execution of all the statements at least once in the source code. For the scenario given, I would have an objection. First thing, it might not be easy to achieve 100% coverage for a large project. Even when it is achievable, the most significant is to cover the most important methods and their scenarios. Statement coverage doesn't consider that many statements involve branching and decision-making, and due to this, a bug can easily occur. Statement coverage does not call for testing simple if statements, logical operators, loop termination decisions, and so on. we can always achieve a full statement coverage for if statement when the condition is only true, but not with false outcome. We can see the example below:

```
Int * p=NULL;
If(condition) {
        P=&variable;
        *P=1;
}
```

```
*p=0;
```

In the code above from [1], If condition evaluate false; the code deference null pointer. But without a test case that causes condition to evaluate false, statement coverage declares this code fully covered. The other is in a loop that stops with a Break statement, test cases needed to expose bugs related to boundary checking can be hidden, for example:

```
char output[100];
 for (int i = 0; i <= sizeof(output); i++) {
 output[i] = input[i];
 if (input[i] == '\0') {
   break;

  }
}
```

In the code above from [1], we will get full statement coverage with any input less than size of the array without exposing the bug. This doesn't test the condition where main loop termination decision, i <= sizeof(output), intends to prevent overflowing the output buffer.

Therefore, even if we achieve 100% statement coverage, the following can go wrong: Simple IF statements, Exception Handling, Incorrect test cases that are just making the statement covered, Missing of functionality, Logical Operators, and Loop termination. 100% statement coverage is not at all adequate for testing criteria and does not imply that all the functionality has been addressed, there may be a missing functionality. I think we need branch testing such as decision coverage with statement coverage to achieve better test coverage.

Reference

1. (n.d.). Retrieved from https://www.bullseye.com/statementCoverage.html#b1
2. Zhu, H., Hall, P. A. V., & May, J. H. R. (1997). Software unit test coverage and adequacy. *ACM Computing Surveys*, *29*(4), 366–427. doi: 10.1145/267580.267590