WHAT'S WRONG WITH SOFTWARE TEST?

Herbert Hecht
SoHaR Incorporated
Beverly Hills, California

ABSTRACT

The short answer to the above question is "Nothing that couldn't be cured by a complete change in setting test objectives". Practically all testing today is aimed at demonstrating that the software meets its requirements. Testing aimed at establishing where it is likely not to meet its requirements can be much more effective.

Our interest arose from a study conducted at the European nuclear research facility at Halden, Norway, the key result of which is shown in Figure 1. The ordinate of this figure measures the branch coverage achieved by a number of test strategies, and the number of test cycles is plotted along the abscissa. Several European authors concluded that the results indicate that uniform random testing is superior to other methodologies (the original article included several additional methodologies that aren't shown here for the sake of clarity). Below we show that a different conclusion may also be possible. The acceptance test represents a conventional test approach in which the program is first subjected to the normal "user profile", then to some mild exceptions (starting at about cycle 30), and finally to severe exception conditions (about cycle 200). The plant simulator only permits inserting plant exception conditions (not those arising from abnormal program execution) and hence it never achieves more than 0.75 coverage.

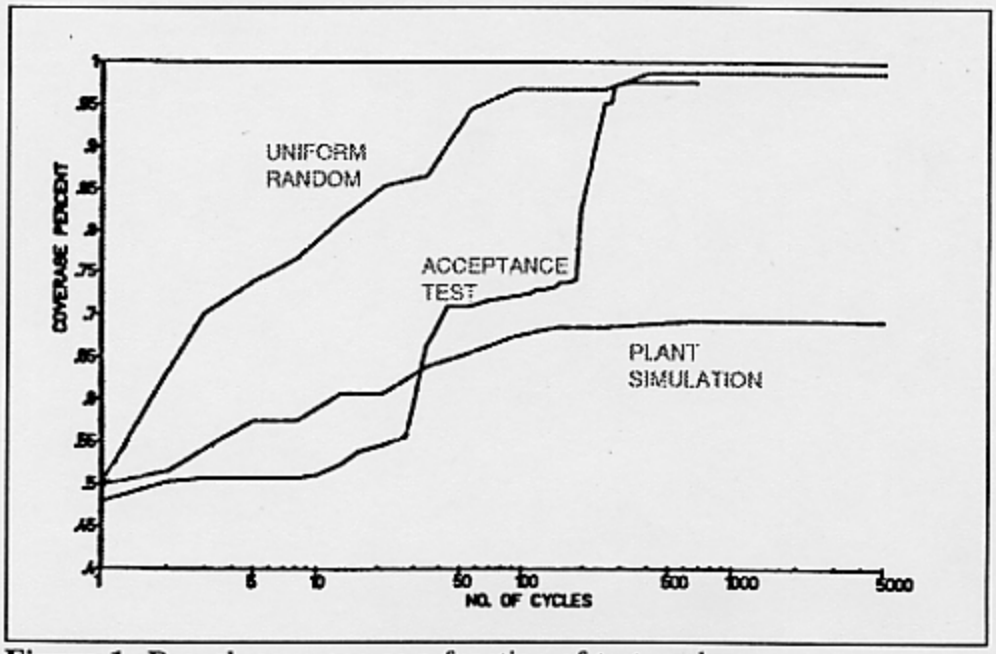

Figure 1 Branch coverage as a function of test cycles

The uniform random test methodology appears superior because it mixes normal and exception

conditions throughout the test. However, a restructuring of the acceptance test to transition to disturbed conditions earlier would have eliminated the prolonged plateaus in the curve and could quite conceivably have achieved higher coverage at a more rapid rate than is the case for the uniform random methodology. Other data supplied by this study indicate that the number of errors detected is roughly proportional to the coverage. The coverage curves were shown here because they have less random noise.

The number of test cycles required to achieve a structural coverage target is not often chronicled as well as it was in the Halden study. But our own experience, as well as that of colleagues whom we exposed to these findings, indicates that the standard approach to software testing is well modeled by the acceptance test curve in the figure. Further confirmation for this belief comes from results of the NASA multi-university experiment in N-version programming. Five programmer teams at each of four universities were asked to generate a program that provided a three-dimensional acceleration vector from the outputs of 6 non-orthogonal accelerometers. A valid output can be generated for up to three accelerometer failures. The test runs summarized in the following table observed the operation of this program by simulating an accelerometer anomaly during each test. Thus, if the run started with two prior anomalies and one anomaly was introduced during the test, this equates to the three accelerometer failures which were the maximum that the program could be expected to handle. Notice the very sharp increase in the failure fraction as a function of the number of prior anomalies. The procedure in this test, as in practically all tests, it to start with routine conditions and then gradually build up to test cases that impose a very high stress on the software. Thus, going from no prior anomalies to one and then two prior anomalies, is very similar to the increase in the severity of the exception conditions that was discussed in connection with Figure 1.

PRIOR ANOMALIES	TOTAL TESTS	FAILURE FRACTION
0	134,135	0.01
1	101,151	0.13
2	143,509	0.58

Table 1 Failures in redundancy management

As both of these examples show, if you want to determine whether a program contains errors (and we assume that is the main purpose of testing), you should go to high stress levels much earlier in the test than is the common practice. A few test runs under routine conditions may be an effective way to check out the test environment. Additional runs under routine or only slightly off-nominal conditions are not anywhere near as likely to find errors than runs at very high stress levels. What's wrong with software test? Nothing that couldn't be cured by planning and executing the test to find errors.