**2. TEST EXECUTION, RESULT CHECKING, AND MEASUREMENT**

The key to the overall test execution is the smooth transition from one test run to another, which also requires us to allocate all the required resources to ensure that individual test runs can be started, executed, and finished, and related problems can be handled seamlessly.

General steps in test execution include:

1. Allocating test time and resources;

2. Invoking and running tests, and collecting execution information and measurements;

**3.** Checking testing results and identifying system failures.

The step to invoke and run tests is fairly straightforward with well-prepared test cases or already sensitized test cases. We can simply provide the input variable values over the whole execution duration as required and as already precisely specified in these test cases. The sequence of test runs can follow the pre-planned sequence.

In the case where test cases are generated dynamically, such as in various usage-based statistical testing approaches, much of the work we described in terms of test sensitization needs to be done at this stage. The key in handling failed test runs is to ensure that they will not block the execution of other test cases. In addition, there will be test runs related to the re-verification of fixed problems, which can be treated much the same way as other planned test cases except the newly added dependency and its impact on test sequencing: Before an integrated fix becomes available, the test case that triggered the failure observation in the first place and other closely related test cases should be suspended to avoid repeatedly observing the same failure, which adds little new information to what is already known. The same test case can be re-run after the fix is in, and closely related test cases can also continue at this point. By doing so, we avoided unnecessary repetitions/re-runs, thus improving the overall test efficiency.

Test time and resources allocation is most closely related to the test planning and preparation activities described in the previous section. Although the allocation could be planned or even carried out at the previous stage, the monitoring, adjustment, and management of these resources need to be carried out during test execution. Test time allocation and management are closely related to people’s roles and responsibility in carrying out specific testing activities. Managing other test resources primarily involves the environmental set up and related facility management. For pure software systems, this is fairly straightforward, with the environment setup to include the hardware configuration and software environment that the finished product will operate within. Sometimes, limited number of simulation programs or hardware simulators can be used for testing some product components, but the overall system testing would very much resemble the actual operational environment. Once the general system configuration is decided, the facility management is mainly the allocation and monitoring of testing time on these facilities.

For embedded software systems or for heterogeneous systems with important software components, the environment and facility management issues involve the so-called “super system”. Coordination between different branches is a major issue where people have different perspectives and concerns. In addition, various techniques, such as simulation and prototyping techniques, will be used to aid testing or sometimes to replace part of the testing. We will also see some specific techniques to deal with interface, interaction, and interoperability problems among different sub-systems as part of the safety assurance program.

**Result checking: The oracle problem**

Result checking, or the ***oracle problem,*** and the related failure identification is a difficult task, because of both the theoretical difficulties and practical limitations. In this book, we use the term ***test oracle*** to indicate any means to check the testing result. Long standing theoretical results state that result checking for testing in general is an undecidable problem.

In other words, there is no hope for algorithmic or fully automated solution to the general test oracle problem. On the practical side, the expected behavior can hardly be precisely described so that the observed behavior can be compared against. Combined with the fact that software can fail in innumerable variations, the unexpected behavior can happen in truly unexpected ways, thus making result checking difficult or nearly impossible. However, there are cases where specific types of system failures, such as irresponsive behavior or system crash, are easy to identify. In other cases, various other means, such and/or execution states, etc., can be used to help us find approximate solutions to the oracle problem, as described below:

* Sometimes heuristics guesses can be used based on product domain knowledge, for example, what other similar products would do under similar situations. Consequently, similar products, such as previous releases of the same product or competitors’ products with similar functionality, can often be used as the test oracle, to check execution results and to identify system failures.
* Knowledge of implementation details can also be used to link specific behavior to specific program units. We can also examine various product internal information and dynamic execution state to help solve the oracle problem. For example, if an external function is supported by some internal components, and these internal components were not invoked when we test for this external function, we can be almost certain there is something wrong with this test run. In addition, product experts or developers themselves can also help testers to perform this difficult task when some important problem is suspected, making effective use of these people’s product knowledge.
* Various types of consistency checking during execution, such as checking for the database consistency, can also help us determine the execution failures. Such consistency checking can usually be done through sampling of some dynamic states and related product internal information, which could be analyzed either on-line during the test execution, or off-line with the detailed dynamic execution information.

**Test measurement**

Observed failures need to be recorded and tracked until their resolution. This defect handling or defect tracking process is typically considered part of the testing process itself, and the reporting of the observed failures is consider part the test execution activity. However, the handling of defects discovered during testing is not fundamentally different from that of defects discovered during other QA activities, as we described in the general context of QA activities.

Detailed information about failure observations and the related activities is needed for problem diagnosis and defect fixing. Some specific information for failures and faults also includes various generic information about defects, covering defect type, severity, impact areas, possible cause, when-injected, etc. This information could be collected either when the failure were observed and recorded or when the faults were fixed, or even afterward. When failures are not observed, the measurement of related test runs can be used to demonstrate product reliability or correct handling of input and dynamic situations.

**Table1. A** template for test execution measurements

|  |
| --- |
| * *rid* - run identification, consisting of   - **sc** - scenario class,  - *sn* - scenario number,  - *vn* - variation number with a particular scenario,  - *an* - attempt number for the specific scenario variation   * *timing* - start time *tO* and end time *tl* * *tester* - the tester who attempted the test run * *trans* - transactions handled by the test run * ***result*** - result of the test run (1 indicates success and 0 for failure) |

Various other measurements can be taken during test execution for further analysis and follow-up actions. Successful executions also need to be recorded for various purposes, including documentation of test activities, possible use as oracle to check future execution results, etc. This is particularly important for regression testing and for legacy products that are expected to change and evolve over the whole product lifespan. The timing and other related information can be important, when it can be used as input in analysis and follow-up activities and in reliability analysis. In addition to the “on-line” measurement of the dynamic test runs and related failure information, the corresponding static test cases can be measured “off-line” to avoid interference with normal test execution. Various other information could also be collected, such as testing personnel, environment, configuration, test object, etc**. Table1** is an example template for test execution information collected for an IBM product during system testing (Tian, 1995). Notice that a test run here corresponds to a specific *attempt* in the hierarchically organized test suite.Each attempt or test run, numbered *an,* is drawn from a specific variation, with variation number *vn,* of a scenario numbered as *sn* that belong to a specific scenario class sc. Other information about individual test runs, such as timing, tester, workload measured in transactions, and the run result, is also recorded.