Debugging
Integration and System Testing

Debugging

Once errors are identified in a program code, it is necessary to first identify the precise program statements responsible for the errors and then to fix them. Identifying errors in a program code and then fix them up are known as debugging.

Debugging Approaches

Brute Force Method:

- This is the most common method, but the least efficient.
- The program is loaded with print statements to print the intermediate values with the hope that some of the printed values will help to identify the statement in error.
- Becomes more systematic with the use of a symbolic debugger (also called a source code debugger), because values of different variables can be easily checked and break points and watch points can be easily set to test the values of variables effortlessly.

Debugging

Backtracking:

- A fairly common approach.
- Beginning from the statement at which an error symptom has been observed, the source code is traced backwards until the error is discovered.
- As the number of source lines to be traced back increases, the number of potential backward paths increases and may become unmanageably large thus limiting the use of this approach.

Cause Elimination Method:

A list of causes which could possibly have contributed to the error symptom is developed and tests are conducted to eliminate each. A related technique of identification of the error from the error symptom is the software fault tree analysis.

Program Slicing:

This technique is similar to back tracking. Here the search space is reduced by defining slices. A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence the value of that variable.

Debugging

Debugging Guidelines:

Debugging is often carried out by programmers based on their ingenuity. The following are some general guidelines for effective debugging:

- Many times debugging requires a thorough understanding of the program design. Trying to debug based on a partial understanding of the system design and implementation may require an inordinate amount of effort to be put into debugging even simple problems.
- Debugging may sometimes even require full redesign of the system. In such cases, a common mistake that novice programmers often make is attempting not to fix the error but its symptoms.
- One must be beware of the possibility that an error correction may introduce new errors.
- Therefore after every round of error-fixing, regression testing must be carried out.

Integration Testing

- Integration testing is carried out after all (or at least some of) the modules have been unit tested.
- Successful completion of unit testing, to a large extent, ensures that the unit (or module) as a whole works satisfactorily.
- The objective of integration testing is to detect the errors at the module interfaces (call parameters).
- For example, it is checked that no parameter mismatch occurs when one module invokes the functionality of another module.
- Thus, the primary objective of integration testing is to test the module interfaces, that is, there are no errors in parameter passing, when one module invokes the functionality of another module

Integration Testing

- During integration testing, different modules of a system are integrated in a planned manner using an integration plan.
- The integration plan specifies the steps and the order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested.
- An important factor that guides the integration plan is the module dependency graph or structure chart which specifies the order in which different modules call each other.
- By examining the structure chart, the integration plan can be developed.

Any one (or a mixture) of the following approaches can be used to develop the test plan:

- ❑ Big-bang approach to integration testing
- ❑ Top-down approach to integration testing
- ❑ Bottom-up approach to integration testing
- ❑ Mixed (also called sandwiched) approach to integration testing

Big-bang approach to integration testing

- Big-bang testing is the most obvious approach to integration testing. In this approach, all the modules making up a system are integrated in a single step.
- All the unit tested modules of the system are simply linked together and tested.
- This technique can meaningfully be used only for very small systems.
- The main problem with this approach is that once a failure has been detected during integration testing, it is very difficult to localize the error as the error may potentially exist in any of the modules.
- Therefore, debugging errors reported during big-bang integration testing are very expensive to fix.
- Big-bang integration testing is almost never used for large programs.

Bottom-up approach to integration testing

- Large software products are often made up of several subsystems. In bottom-up integration testing, first the modules for the each subsystem are integrated. Thus, the subsystems can be integrated separately and independently.
- Large software systems normally require several levels of subsystem testing, lower-level subsystems are successively combined to form higher-level subsystems.
- The principal advantage of bottom-up integration testing is that several disjoint subsystems can be tested simultaneously. Another advantage of bottom-up testing is that the low-level modules get tested thoroughly, since they are exercised in each integration step.
- Since the low-level modules do I/O and other critical functions, testing the low-level modules thoroughly increases the reliability of the system.
- A disadvantage of bottom-up testing is the complexity that occurs when the system is made up of a large number of small subsystems that are at the same level. This extreme case corresponds to the big-bang approach.

Top-down approach to integration testing

Top-down integration testing starts with the root module in the structure chart and one or two subordinate modules of the root module.

After the top-level 'skeleton' has been tested, the modules that are at the immediately lower layer of the 'skeleton' are combined with it and tested.

Top-down integration testing approach requires the use of program stubs to simulate the effect of lower-level routines that are called by the routines under test. A pure top-down integration does not require any driver routines.

An advantage of top-down integration testing is that it requires writing only stubs, and stubs are simpler to write compared to drivers.

A disadvantage of the top-down integration testing approach is that in the absence of lower-level routines, it becomes difficult to exercise the top-level routines in the desired manner since the lower level routines usually perform input/output (I/O) operations.

Mixed approach to integration testing

- The mixed (also called sandwiched) integration testing follows a combination of top-down and bottom-up testing approaches.
- In top-down approach, testing can start only after the top-level modules have been coded and unit tested.
- Similarly, bottom-up testing can start only after the bottom level modules are ready.
- The mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. In the mixed testing approach, testing can start as and when modules become available after unit testing.
- Therefore, this is one of the most commonly used integration testing approaches.
- In this approach, both stubs and drivers are required to be designed.

System Testing

After all the units of a program have been integrated together and tested, system testing is taken up.

The test cases for system testing are designed solely based on the SRS document and the actual implementation (procedural or object-oriented) is immaterial.

There are three main kinds of system testing.

Alpha Testing: Alpha testing refers to the system testing carried out by the test team within the developing organization.

Beta Testing: Beta testing is the system testing performed by a select group of friendly customers.

Acceptance Testing: Acceptance testing is the system testing performed by the customer to determine whether to accept the delivery of the system

Smoke Testing

- Smoke testing is performed to check whether at least the main functionalities of the software are working properly. Unless the software is stable and at least the main functionalities are working satisfactorily, system testing is not undertaken.
- Smoke testing is carried out before initiating system testing to ensure that system testing would be meaningful, or whether many parts of the software would fail.
- The idea behind smoke testing is that if the integrated program cannot pass even the basic tests, it is not ready for a vigorous testing.
- For smoke testing, a few test cases are designed to check whether the basic functionalities are working.
- For example, for a library automation system, the smoke tests may check whether books can be created and deleted, whether member records can be created and deleted, and whether books can be loaned and returned.

Performance Testing

Performance testing is an important type of system testing. For a specific system, the types of performance testing to be carried out on a system depends on the different non-functional requirements of the system documented in its SRS document. All performance tests can be considered as black-box tests.

There are several types of performance testing corresponding to various types of non-functional requirements-

- **Stress testing**
- **Volume testing**
- **Configuration testing**
- **Compatibility testing**
- **Regression testing**
- **Recovery testing**
- **Maintenance testing**
- **Documentation testing**
- **Usability testing**
- **Security testing**

Performance Testing

Stress testing

- Stress testing is also known as *endurance testing*. Stress testing evaluates system performance when it is stressed for short periods of time. Stress tests are black-box tests which are designed to impose a range of abnormal and even illegal input conditions so as to stress the capabilities of the software. Input data volume, input data rate, processing time, utilization of memory, etc., are tested beyond the designed capacity.
- Stress testing is especially important for systems that under normal circumstances operate below their maximum capacity but may be severely stressed at some peak demand hours.

Performance Testing

Volume testing

- ❖ Volume testing checks whether the data structures (buffers, arrays, queues, stacks, etc.) have been designed to successfully handle extraordinary situations. For example, the volume testing for a compiler might be to check whether the symbol table overflows when a very large program is compiled.

Configuration testing

- ❖ Configuration testing is used to test system behavior in various hardware and software configurations specified in the requirements. Sometimes systems are built to work in different configurations for different users. For instance, a minimal system might be required to serve a single user, and other extended configurations may be required to serve additional users. During configuration testing, the system is configured in each of the required configurations and it is checked if the system behaves correctly in all required configurations.

Performance Testing

Compatibility testing

- This type of testing aims to check whether the interfaces with the external systems are performing as required. For instance, if the system needs to communicate with a large database system to retrieve information, compatibility testing is required to test the speed and accuracy of data retrieval.

Regression testing

- This type of testing is required when a software is maintained to fix some bugs or enhance functionality, performance, etc.

Recovery testing

- Recovery testing tests the response of the system to the presence of faults, or loss of power, devices, services, data, etc. The system is subjected to the loss of the mentioned resources (as discussed in the SRS document) and it is checked if the system recovers satisfactorily.

Performance Testing

Maintenance testing

- ❖ This addresses testing the diagnostic programs, and other procedures that are required to help maintenance of the system. It is verified that the artifacts exist and they perform properly.

Documentation testing

- ❖ It is checked whether the required user manual, maintenance manuals, and technical manuals exist and are consistent. If the requirements specify the types of audience for which a specific manual should be designed, then the manual is checked for compliance of this requirement.

Performance Testing

Usability testing

- Usability testing concerns checking the user interface to see if it meets all user requirements concerning the user interface. During usability testing, the display screens, messages, report formats, and other aspects relating to the user interface requirements are tested.

Security testing

- Security testing is essential for software that handle or process confidential data that is to be guarded against pilfering. It needs to be tested whether the system is fool-proof from security attacks such as intrusion by hackers. A large number of security testing techniques have been proposed, and these include password cracking, penetration testing, and attacks on specific ports, etc.