Software Testing Strategies

Mrs. Bhumika S. Zalavadia[1]
[1]HOD
[1]Computer Department
[1]Atmiya Institute of Technology and Science for Diploma Studies-Rajkot, India

Abstract---Software has increased tremendously over the past decades which lead to inventions of more and more efficient software development techniques. It is always of interest that software must be developed with high degree of accuracy in minimum time period with the fact that software must satisfy all the desires of the customer. After the software is successfully developed by the software engineer, different efficient testing techniques are applied to test whether it produces the desired outputs or not. Testing techniques not only focus on the desired output, but it also focuses on the fact that output must be generated in easiest way. All the customers are not the computer experts. So software must be designed in such a way that any technical or non-technical person can operate the software in most convenient way.

I. Introduction

Software development is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and enhance specific software. The life cycle of software development defines a method for improving the quality of software and the overall development process. Any software development process goes through following phases [3].

(1) Planning
(2) Defining
(3) Designing
(4) Building (Implementing)
(5) Testing
(6) Deployment

II. Stategic Approach To Software Testing

Testing is a set of activities that can be planned in advance and conducted systematically. For this reason a template for software testing which is a set of steps into which we can place specific test case design techniques and testing methods, should be defined for the software process. In testing, system behavior is observed, and from that we determine whether or not there is a failure. By observing failures of the system we can assume the presence of faults. The actual faults are identified by separate activities, commonly known as debugging. In other words, for identifying faults, after testing the expensive task of debugging has to be performed. This is one of the reasons why testing is an expensive method for identification of faults, compared to methods that directly observe faults [2].

III. Generic Charactristics Of Software Testing

A number of software testing strategies have been proposed till now. All provide the software developer with a template for testing and all have the following generic characteristics [3].

- Testing begins at the component level and works toward the integration of the entire computer-based system,
- Different testing techniques are appropriate at different points in time.
- Testing is conducted by the developer of the software or by an independent test group for large projects.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.
- A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. A strategy must provide guidance for the practitioner and a set of highlights for the manager.

IV. Strategies Of Software Testing

The software engineering process may be viewed as the spiral process of each of the steps listed above. Initially, system engineering defines the role of software and leads to software requirements analysis where the information domain, function, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral we come to design and finally to coding. To develop computer software, we spiral inward along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral.

- Unit testing begins at the vortex of the spiral and concentrates on each unit of the software as implemented in source code.
- Taking another turn outward on the spiral, we come to integration testing where the focus is on design and the construction of the software architecture.
- Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed.
- Finally, we arrive at system testing, where the software and other system elements are tested as a whole. To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.

A. *Unit Testing*

Unit testing focuses verification effort on the smallest unit of software design that is software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components [4].

1) Considerations in Unit Testing

The module interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested [1].

Tests of data flow across a module interface are required before any other test is initiated. If data do not enter and exit properly, all other tests are unresolved. In addition, local data structures should be exercised and the local impact on global data should be ascertained during unit testing.

Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons, and improper control flow. Basis path and loop testing are effective techniques for uncovering a broad array of path errors.

Among the more common errors in computation are followings [2].

- Misunderstood or incorrect arithmetic precedence,
- Mixed mode operations,
- Incorrect initialization
- Precision inaccuracy
- Incorrect symbolic representation of an expression.
- Comparison of different data types
- Incorrect logical operators or precedence
- Expectation of equality when precision error makes equality unlikely
- Incorrect comparison of variables
- Improper or nonexistent loop termination
- Failure to exit when divergent iteration is encountered
- Improperly modified loop variables.

2) Procedures in Unit Testing

Unit testing is normally considered as an adjunct to the coding step. After source level code has been developed, reviewed, and verified for correspondence to component-level design, unit test case design begins. A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories discussed earlier. Each test case should be coupled with a set of expected results. Because a component is not a stand-alone program, driver or stub software must be developed for each unit test. In most applications a driver is a main program that accepts test case data, passes such data to the component to be tested and prints relevant results. Stubs serve to replace modules that are subordinate the component to be tested. A stub or dummy subprogram uses the subordinate modules interface, may do minimal data manipulation, prints verification of entry and returns control to the module undergoing testing [2].

Drivers and stubs represent overhead that is, both are software that must be developed to test the module but it will not be delivered to the customer. Unit testing is simplified when a component with high consistency is designed. When only one function is addressed by the component, the number of test cases is reduced and errors can be more easily predicted and uncovered [2].

B. *Integration Testing*

Sometimes it happens that when all modules work individually, desired result can be obtained. But when we put them together using common interface the output may be incorrect or different from our expectation. Data can be lost across an interface. One module can effect on another module or sub-functions, when combined, may not produce the desired result [5].

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. All components are combined in advance. The entire program is tested as a whole. When a set of errors is encountered, correction is difficult because isolation or causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues endlessly [5].

1) Incremental Integration

Incremental integration is the approach to solve above program. The program is constructed and tested in small increments, where errors are easier to locate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied. In the sections that follow a number of different incremental integration strategies are followings [3].

2) Top Down Integration Testing

Top down integration testing is an incremental approach to construction or program structure. Modules are integrated by moving downward through the control hierarchy beginning with the main control module. Modules subordinate to the main control module are incorporated into the structure in either a depth-first or breadth-first manner [3].

The integration process is performed in a series of five steps:

(1) The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
(2) Depending on the integration approach selected, subordinate stubs are replaced one at a time with actual components. Tests are conducted as each component is integrated. On completion of each set of tests, another stub is replaced with the real components.
(3) Regression testing may be conducted to ensure that new errors have not been introduced. The process continues until the entire program structure is built.
(4) The top-down integration strategy verifies major control or decision points early in the test process. In a well-factored program structure, decision making occurs at upper levels in the hierarchy and is therefore encountered first. If major control problems do exist, early recognition is essential. If depth-first integration is selected, a complete function of the software may be implemented and demonstrated.
(5) Top-down strategy sounds relatively uncomplicated, but in practice, logistical problems can arise. The most common of these problems occurs when processing at low levels in the hierarchy is required to adequately test

upper levels. Stubs replace low-level modules at the beginning of top down testing; therefore, no significant data can now upward in the program structure.

3) Bottom up Integration Testing

It begins construction and testing with atomic modules. Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated [3]. A bottom-up integration strategy may be implemented with the following steps:

- Low-level components are combined into clusters that perform a specific software sub-function.
- A driver which is a control program for testing is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.

C. *Validation Testing*

Software validation is the process of testing software to check whether it satisfies the customer needs or not. This testing is done during and/or at the end of the process of software development. Verification and validation testing are two important tests which are carried out on software before it has been handed over to the customer. The aim of both verification and validation is to ensure that the product is made according to the requirements of the client, and does indeed fulfill the intended purpose. While verification is a quality control process, the quality assurance process carried out before the software is ready for release is known as validation testing. Its goal is to validate and be confident about the product or system, and that it fulfills the requirements given by the customer. The acceptance of the software from the end customer is also its part. Often, testing activities are introduced early in the software development life cycle [5].

Validation testing provides answers to questions such as Does the software fulfill its real use? is the company building the right product?, can the project be properly implemented?, are the documents in line with the development process? Etc [4].

D. *System Testing*

The purpose of system testing is to identify defects that will only occur when a complete system is assembled. That is, defects that cannot be attributed to individual components or the interaction between two components. Testers, who are trained to plan, execute, and report on application and system code should perform system testing. They should be aware of scenarios that might not occur to the end user, like testing for null, negative, and format inconsistent values. A tester should be able to repeat the steps that caused an error. System testing includes testing of performance, security, configuration sensitivity, startup and recovery from failure modes.

System testing is available for different operating systems like Windows XP, Windows 2000, Windows ME, and Windows Server 2003, Windows Datacenter Server etc. for Desktop PC, Mobile PC and Server PC [4].

V. DEBUGGING

The process of debugging involves analyzing and possibly extending the given program that does not meet the specifications in order to find a new program that is close to the original and does satisfy the specifications

Thus it is the process of diagnosing the precise nature of a known error and then correcting it. The purpose of debugging is to locate and fix the offending code responsible for a symptom violating a known specification.

Debugging typically happens during three activities in software development, and the level of granularity of the analysis required for locating the defect differs in these three [6].

(1) The first is during the coding process, when the programmer translates the design into an executable code. During this process the errors made by the programmer in writing the code can lead to defects that need to be quickly detected and fixed before the code goes to the next stages of development. Most often, the developer also performs unit testing to expose any defects at the module or component level.

(2) The second place for debugging is during the later stages of testing, involving multiple components or a complete system, when unexpected behavior such as wrong return codes or abnormal program termination may be found. A certain amount of debugging of the test execution is necessary to conclude that the program under test is the cause of the unexpected behavior and not the result of a bad test case due to incorrect specification, inappropriate data, or changes in functional specification between different versions of the system. Once the defect is confirmed, debugging of the program follows and the misbehaving component and the required fix are determined.

(3) The third place for debugging is in production or deployment, when the software under test faces real operational conditions. Some undesirable aspects of software behavior, such as inadequate performance under a severe workload or unsatisfactory recovery from a failure, get exposed at this stage and the offending code needs to be found and fixed before large-scale deployment. This process may also be called problem determination, due to the enlarged scope of the analysis required before the defect can be localized.

REFERENCES

[1] http://en.wikipedia.org/wiki/Software_testing
[2] Software Engineering by K.K. Aggarwal, Yogesh Singh
[3] http://www.internetjournals.net/journals/tir/2009/January/Paper%2006.pdf
[4] Software engineering by Rojar S. Pressman
[5] Software testing research and techniques by Autonla Bertollno
[6] Introduction to software testing available at http://www.onestoptetsing.com/introduction
[7] Mohd. Ehmer Khan,"Different Forms of Software Testing Techniques for Finding" IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 1, May 2010.
[8] Guide to the Software Engineering Body of Knowledge, Swebok – A project of the IEEE Computer Society Professional Practices Committee, 2004.

