Software Testing and Software Development Lifecycles

Ms Chitra Wasnik

Dr Chelpa Lingam

ABSTRACT

Software Testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software. What is Software Testing? Process of validating and verifying that a program does what it is expected to do. Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which it is intended to operate.

Software Testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs. It can also be stated as the process of validating and verifying that a software program/application/product meets the business and technical requirements that guided its design and development, so that it works as expected and can be implemented with the same characteristics. Software Development Life Cycle (SDLC) is a methodology that is typically used to develop, maintain and replace information systems for improving the quality of the software design and development process.

KEYWORDS

Software testing, software specifications, software quality plan, software development lifecycles, regression testing

1. INTRODUCTION

This paper provides an introduction to software testing and outlines some commonly used software development lifecycle models, with particular emphasis on the testing activities involved in each model. Topics covered include basic definitions of testing, validation and verification; the levels of testing from unit testing through to acceptance testing; the relationship with requirements and design specifications; and test documentation. There are a number of different models for software development lifecycles.

Irrespective of the lifecycle model used for software development, software has to be tested. Efficiency and quality are best served by testing software as early in the lifecycle as practical, with full regression testing whenever changes are made.

2. WHAT IS SOFTWARE TESTING?

Software testing is the process of executing software in a controlled manner, in order to answer the question "Does the software behave as specified?" Software testing is often used in association with the terms verification and validation. Verification is the checking or testing of items, including software, for conformance and consistency with an associated specification. Software testing is just one kind of verification, which also uses techniques such as reviews, analysis, inspections and walkthroughs. Validation is the process of checking that what has been specified is what the user actually wanted.

Validation: Are we doing the right job?

Verification: Are we doing the job right?

3. SOFTWARE SPECIFICATIONS AND TESTING

Depending on the size of the development and the development methods, specification of software can range from a single document to a complex hierarchy of documents. A hierarchy of software specifications will typically contain three or more levels of software specification documents. The Requirements Specification, which specifies what the software is required to do and may also specify constraints on how this may be achieved. The Architectural Design Specification, which describes the architecture of a design which implements the requirements. Components within the software and the relationship between them will be described in this document.

Detailed Design Specifications, which describe how each component in the software, down to individual units, is to be implemented.
The design of tests is subject to the same basic engineering principles as the design of software. Good design consists of a number of stages which progressively elaborate the design of tests from an initial high level strategy to detailed test procedures. These stages are: test strategy, test planning, test case design, and test procedure design. At the highest level this means that tests will be designed to verify that the software faithfully implements the requirements of the Requirements Specification. At lower levels tests will be designed to verify that items of software implement all design decisions made in the Architectural Design Specification and Detailed Design Specifications.

4.1. Test Strategy

The first stage is the formulation of a test strategy. A test strategy is a statement of the overall approach to testing, identifying what levels of testing are to be applied and the methods, techniques and tools to be used. A test strategy should ideally be organisation wide, being applicable to all of organisations software developments. The application of a test strategy to a software development project should be detailed in the projects software quality plan.

4.2 Test Plans

The next stage of test design, which is the first stage within a software development project, is the development of a test plan. A test plan states what the items to be tested are, at what level they will be tested, what sequence they are to be tested in, how the test strategy will be applied to the testing of each item, and describes the test environment. A test plan may be project wide, or may in fact be a hierarchy of plans relating to the various levels of specification and testing: An Acceptance Test Plan, describing the plan for acceptance testing of the software. A System Test Plan, describing the plan for system integration and testing. This would also usually be published as a separate document, but might be published with the acceptance test plan. A Software Integration Test Plan, describing the plan for integration of tested software components. This may form part of the Architectural Design Specification. Unit Test Plan(s), describing the plans for testing of individual units of software. These may form part of the Detailed Design Specifications. The objective of each test plan is to provide a plan for verification, by testing the software, that the software produced fulfils the requirements or design statements of the appropriate software specification. In the case of acceptance testing and system testing, this means the Requirements Specification.

4.3. Test Case Design

Once the test plan for a level of testing has been written, the next stage of test design is to specify a set of test cases or test paths for each item to be tested at that level. A number of test cases will be identified for each item to be tested at each level of testing. Each test case will specify how the implementation of a particular requirement or design decision is to be tested and the criteria for success of the test. The test cases may be documented with the test plan, as a section of a software specification, or in a separate document called a test specification or test description. An Acceptance Test Specification, specifying the test cases for acceptance testing of the software. This would usually be published as a separate document, but might be published with the acceptance test plan. A System Test Specification, specifying the test cases for system integration and testing. This would also usually be published as a separate document, but might be
published with the system test plan. Software Integration Test Specifications, specifying the test cases for each stage of integration of tested software components. These may form sections of the Architectural Design Specification. Unit Test Specifications, specifying the test cases for testing of individual units of software. These may form sections of the Detailed Design Specifications. System testing and acceptance testing involve an enormous number of individual test cases. In order to keep track of which requirements are tested by which test cases, an index which cross references between requirements and test cases often constructed. This is usually referred to as a Verification Cross Reference Index (VCRI) and is attached to the test specification. Cross reference indexes may also be used with unit testing and software integration testing.

4.4. Test Procedures

The final stage of test design is to implement a set of test cases as a test procedure, specifying the exact process to be followed to conduct each of the test cases. This is a fairly straightforward process, which can be likened to designing units of code from higher level functional descriptions. For each item to be tested, at each level of testing, a test procedure will specify the process to be followed in conducting the appropriate test cases. A test procedure cannot leave out steps or make assumptions. The level of detail must be such that the test procedure is deterministic and repeatable.

5. TEST RESULTS DOCUMENTATION

When tests are executed, the outputs of each test execution should be recorded in a test results file. These results are then assessed against criteria in the test specification to determine the overall outcome of a test. If AdaTEST or Cantata are used, this file will be created and the results assessed automatically according to criteria specified in the test script. Each test execution should also be noted in a test log. The test log will contain records of when each test has been executed, the outcome of each test execution, and may also include key observations made during test execution. Often a test log is not maintained for lower levels of testing (unit test and software integration test). Test reports may be produced at various points during the testing process. A test report will summarise the results of testing and document any analysis. An acceptance test report often forms a contractual document within which acceptance of software is agreed.

6. SOFTWARE DEVELOPMENT LIFECYCLES

There are a number of different models for software development lifecycles. One thing which all models have in common is that at some point in the lifecycle, software has to be tested.

6.1. Sequential Lifecycle Models

The software development lifecycle begins with the identification of a requirement for software and ends with the formal verification of the developed software against that requirement. Traditionally, the models used for the software development lifecycle have been sequential, with the development progressing through a number of well defined phases. The sequential phases are usually represented by a V or waterfall diagram. These models are respectively called a V lifecycle model and a waterfall lifecycle model.

![Figure 2 V Lifecycle Model](image-url)
There are in fact many variations of V and waterfall lifecycle models, introducing phases to the lifecycle and creating different boundaries between phases. The following set of lifecycle phases fits in with the practices of most professional software developers. The Requirements phase, in which the requirements for the software are gathered and analyzed, to produce a complete and unambiguous specification of what the software is required to do. The Architectural Design phase, where a software architecture for the implementation of the requirements is designed and specified, identifying the components within the software and the relationships between the components.

The Detailed Design phase, where the detailed implementation of each components specified. The Code and Unit Test phase, in which each component of the software is coded and tested to verify that it faithfully implements the detailed design. The Software Integration phase, in which progressively larger groups of tested software components are integrated and tested until the software works as a whole. The System Integration phase, in which the software is integrated to the overall product and tested. The Acceptance Testing phase, where tests are applied and witnessed to validate that the software faithfully implements the specified requirements.

6.2. Progressive Development Lifecycle Models

The sequential V and waterfall lifecycle models represent an idealised model of software development. Other lifecycle models may be used for a number of reasons, such as volatility of requirements, or a need for an interim system with reduced functionality when long timescales are involved. As an example of other lifecycle models, let us look at progressive development and iterative lifecycle models. A common problem with software development is that software is needed quickly, but it will take a long time to fully develop. The solution is to form a compromise between timescales and functionality, providing "interim" deliveries of software, with reduced functionality, but serving as a stepping stones towards the fully functional software. It is also possible to use such a stepping stone approach as a means of reducing risk. The usual names given to this approach to software development are progressive development or phased implementation. The corresponding lifecycle model is referred to as a progressive development lifecycle. Within a progressive development lifecycle, each individual phase of development will follow its own software development lifecycle, typically using a V or waterfall model. The actual number of phases will depend upon the development.
Each delivery of software will have to pass acceptance testing to verify the software fulfils the relevant parts of the overall requirements. The testing and integration of each phase will require time and effort, so there is a point at which an increase in the number of development phases will actually become counter productive, giving an increased cost and timescale, which will have to be weighed carefully against the need for an early solution. The software produced by an early phase of the model may never actually be used, it may just serve as a prototype. A prototype will take short cuts in order to provide a quick means of validating key requirements and verifying critical areas of design. These short cuts may be in areas such as reduced documentation and testing. When such short cuts are taken, it is essential to plan to discard the prototype and implement the next phase from scratch, because the reduced quality of the prototype will not provide a good foundation for continued development.

6.3. Iterative Lifecycle Models

An iterative lifecycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model. Consider an iterative lifecycle model which consists of repeating the four phases in sequence, as illustrated by figure 5. A Requirements phase, in which the requirements for the software are gathered and analyzed. Iteration should eventually result in a requirements phase which produces a complete and final specification of requirements. Design phase, in which a software solution to meet the requirements is designed. This may be a new design, or an extension of an earlier design. An Implementation and Test phase, when the software is coded, integrated and tested. A Review phase, in which the software is evaluated, the current requirements are reviewed, and changes and additions to requirements proposed. For each cycle of the model, a decision has to be made as to whether the software produced by the cycle will be discarded, or kept as a starting point for the next cycle (sometimes referred to as incremental prototyping). Eventually a point will be reached where the requirements are complete and the software can be delivered, or it becomes impossible to enhance the software as required, and a fresh start has to be made.

![Diagram of Iterative Lifecycle Model](image)

**Figure 5 Iterative Lifecycle Model**

Eventually a point will be reached where the requirements are complete and the software can be delivered, or it becomes impossible to enhance the software as required, and a fresh start has to be made. The iterative lifecycle model can be likened to producing software by successive approximation. Drawing an analogy with mathematical methods which use successive approximation to arrive at a final solution, the benefit of such methods depends on how rapidly they converge on a solution. The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification (including testing) of each version of the software against those requirements within each cycle of the model. The first three phases of the example iterative model are in fact an abbreviated form a sequential V or waterfall lifecycle model. Each cycle of the model produces software which requires testing at the unit level, for
software integration, for system integration and for acceptance. As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

7. MAINTENANCE

Successfully developed software will eventually become part of a product and enter a maintenance phase, during which the software will undergo modification to correct errors and to comply with changes to requirements. Like the initial development, modifications will follow a software development lifecycle, but not necessarily using the same lifecycle model as the initial development. Throughout the maintenance phase, software tests have to be repeated, modified and extended. The effort to revise and repeat tests consequently forms a major part of the overall costs of developing and maintaining software. The term regression testing is used to refer to the repetition of earlier successful tests in order to make sure that changes to the software have not introduced side effects.

8. CONCLUSIONS

Software can be tested at various stages of the development and with various degrees of rigour. Like any development activity, testing consumes effort and effort costs money. From an economics point of view, the level of testing appropriate to a particular organisation and software application will depend on the potential consequences of undetected bugs. Irrespective of the lifecycle model used for software development, software has to be tested. Efficiency and quality are best served by testing software as early in the lifecycle as practical, with full regression testing whenever changes are made.

REFERENCES