An Introduction to Software Testing

Executive Summary

This paper provides an introduction to software testing. It serves as a tutorial for developers who are new to formal testing of software, and as a reminder of some finer points for experienced software testers.

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.

IPL is an independent software house founded in 1979 and based in Bath. IPL was accredited to ISO9001 in 1988, and gained TickIT accreditation in 1991. IPL has developed and supplies the AdaTEST and Cantata software verification products. AdaTEST and Cantata have been produced to these standards.

IPL
Eveleigh House
Grove Street
Bath
BA1 5LR
UK
Phone: +44 (0) 1225 444888
Fax: +44 (0) 1225 444400
Email: tools@ipl.com

1. What is Software Testing?

There are many published definitions of software testing, however, all of these definitions boil down to essentially the same thing: 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?

The term **bug** is often used to refer to a problem or fault in a computer. There are *software bugs* and *hardware bugs*. The term originated in the United States, at the time when pioneering computers were built out of valves, when a series of previously inexplicable faults were eventually traced to moths flying about inside the computer.

Software testing should not be confused with debugging. Debugging is the process of analyzing and locating bugs when software does not behave as expected. Although the identification of some bugs will be obvious from playing with the software, a methodical approach to software testing is a much more thorough means of identifying bugs. Debugging is therefore an activity which supports testing, but cannot replace testing. However, no amount of testing can be guaranteed to discover all bugs.

Other activities which are often associated with software testing are **static analysis** and **dynamic analysis**. Static analysis investigates the source code of software, looking for problems and gathering metrics without actually executing the code. Dynamic analysis looks at the behaviour of software while it is executing, to provide information such as execution traces, timing profiles, and test coverage information.

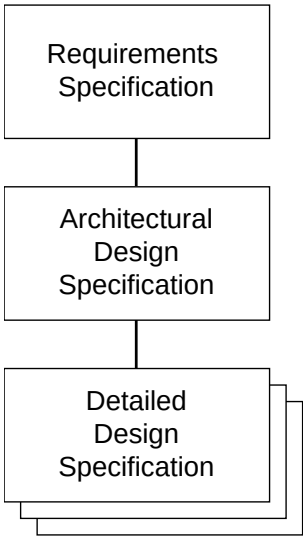
2. Software Specifications and Testing

The key component of the above definitions is the word **specified**. Validation and verification activities, such as software testing, cannot be meaningful unless there is a specification for the software. Software could be a single module or unit of code, or an entire system. 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.



With such a hierarchy of specifications, it is possible to test software at various stages of the development, for conformance with each specification. The levels of testing which correspond to the hierarchy of software specifications listed above are:

- **Unit Testing**, in which each unit (basic component) of the software is tested to verify that the detailed design for the unit has been correctly implemented.
- **Software Integration Testing**, in which progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a whole.
- **System Testing**, in which the software is integrated to the overall product and tested to show that all requirements are met.

A further level of testing is also concerned with requirements:

- **Acceptance Testing**, upon which acceptance of the completed software is based. This will often use a subset of the system tests, witnessed by the customers for the software or system.

Once each level of software specification has been written, the next step is to design the tests. An important point here is that the tests should be designed before the software is implemented, because if the software was implemented first it would be too tempting to test the software against what it is observed to do (which is not really testing at all), rather than against what it is specified to do.

Within each level of testing, once the tests have been applied, test results are evaluated. If a problem is encountered, then either the tests are revised and applied again, or the

software is fixed and the tests applied again. This is repeated until no problems are encountered, at which point development can proceed to the next level of testing.

Testing does not end following the conclusion of acceptance testing. Software has to be maintained to fix problems which show up during use and to accommodate new requirements. Software tests have to be repeated, modified and extended. The effort to revise and repeat tests consequently forms a major part of the overall cost 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.

3. Test Design Documentation

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.

The design of tests has to be driven by the specification of the software. 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. As with any design process, each stage of the test design process should be subject to informal and formal review.

The ease with which tests can be designed is highly dependant on the design of the software. It is important to consider testability as a key (but usually undocumented) requirement for any software development.

3.1. 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 an organisations software developments.

Developing a test strategy which efficiently meets the needs of an organisation is critical to the success of software development within the organisation. The application of a test strategy to a software development project should be detailed in the projects **software quality plan**.

3.2. 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. This would usually be published as a separate document, but might be published with the system test plan as a single document.
- 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.

3.3. 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.

It is important to design test cases for both **positive testing** and **negative testing**. Positive testing checks that the software does what it should. Negative testing checks that the software doesn't do what it shouldn't.

The process of designing test cases, including executing them as *thought experiments*, will often identify bugs before the software has even been built. It is not uncommon to find more bugs when designing tests than when executing tests.

3.4. 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 straight forward 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.

Test procedures should always be separate items, because they contain a great deal of detail which is irrelevant to software specifications. If AdaTEST or Cantata are used, test procedures may be coded directly as AdaTEST or Cantata test scripts.

4. 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.

5. Further Results and Conclusion

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. Developers should plan for between 30% and 70% of a projects effort to be expended on verification and validation activities, including software testing.

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. Such consequences can range from a minor inconvenience of having to find a work-round for a bug to multiple deaths. Often overlooked by software developers (but not by customers), is the long term damage to the credibility of an organisation which delivers software to users with bugs in it, and the resulting negative impact on

future business. Conversely, a reputation for reliable software will help an organisation to obtain future business.

Efficiency and quality are best served by testing software as early in the life cycle as practical, with full regression testing whenever changes are made. The later a bug is found, the higher the cost of fixing it, so it is sound economics to identify and fix bugs as early as possible. Designing tests will help to identify bugs, even before the tests are executed, so designing tests as early as practical in a software development is a useful means of reducing the cost of identifying and correcting bugs.

In practice the design of each level of software testing will be developed through a number of layers, each adding more detail to the tests. Each level of tests should be designed before the implementation reaches a point which could influence the design of tests in such a way as to be detrimental to the objectivity of the tests. Remember: software should be tested against what it is specified to do, not against what it actually observed to do.

The effectiveness of testing effort can be maximised by selection of an appropriate testing strategy, good management of the testing process, and appropriate use of tools such as AdaTEST or Cantata to support the testing process. The net result will be an increase in quality and a decrease in costs, both of which can only be beneficial to a software developers business.

The following list provides some rules to follow as an aid to effective and beneficial software testing.

- Always test against a specification. If tests are not developed from a specification, then it is not testing. Hence, testing is totally reliant upon adequate specification of software.
- Document the testing process: specify tests and record test results.
- Test hierarchically against each level of specification. Finding more errors earlier will ultimately reduce costs.
- Plan verification and validation activities, particularly testing.
- Complement testing with techniques such as static analysis and dynamic analysis.
- Always test positively: that the software does what it should, but also negatively: that it doesn't do what it shouldn't.
- Have the right attitude to testing - it should be a challenge, not the chore it so often becomes.