

White paper on Unit Testing

R.Venkat Rajendran, Director, Deccanet Designs Ltd.

Introduction

The world has started to rely increasingly on software that is becoming more and more complex. Today software is everywhere - part of every system or machine that we use in daily life. Software Quality and Reliability have become an important concern of the software industry. While improved and formal software development processes help avoid more and more defects, a large number of defects are still left in software. In one of history's well-known bugs, the entire long distance of AT&T was down for nine hours. The melt down was finally traced to a single line of code that could have been best detected by some effective unit testing. Modern wireless handsets are recent examples of the problem faced by software designers. Beizer argues that the number of bugs left behind after a software engineer "delivers" his code to others is still too high for comfort. While improved design techniques try to *avoid* defects, *effective testing* continues to play an important role in *removing* defects left behind and is the cornerstone of the Software Quality Assurance activity.

Increasing complexity and Reducing time to market

The complexity of software is increasing sharply while the time to market is growing equally fast. For wireless mobile handsets used to have a few tens of thousands of code a few years back and they now carry a few millions line of code. But due to competitive pressure the development time for such complexity has comedown by 50%. Consequently, the 3G handsets that were released in Japanese market had to be recalled due to software defects. Better early phase testing is a solution that can address this problem well.

Testing takes significant effort

Testing accounts for significant part of the software development lifecycle. According to Boris Beizer "Testing and debugging costs range from 50% to 80% of the cost of producing the first working version of a software package. If the lifecycle of software is considered from inception to retirement, then test and quality assurance related costs are an even larger part of the total cost." [1]

Unit Testing plays a major role in the total testing efforts. Studies by Barry Boehm reveal that coding and unit test takes 36 to 42% of the total *resources* consumed in software projects of various sizes [2]. In projects that demand high quality and reliability, unit testing is a key phase of testing.

Why Unit Testing ?

Units are the smallest building blocks of software. In a language like C, individual functions make up the units. Unit testing is the process of validating such small building blocks of a complex system much before testing an integrated large module or the system as a whole. Some of the major benefits are:

- Be able to test parts of a project with out waiting for the other parts to be available,
- Achieve parallelism in testing by being able to test and fix problems simultaneously by many engineers,
- Be able to detect and remove defects at a much less cost compared to other later stages of testing,
- Be able to take advantage of a number of formal testing techniques available for unit testing,
- Simplify debugging by limiting to a small unit the possible code areas in which to search for bugs,
- Be able to test internal conditions that are not easily reached by external inputs in the larger integrated systems (for example, exception conditions not easily reached in normal operation)
- Be able to achieve a high level of structural coverage of the code,
- Avoid lengthy compile-build-debug cycles when debugging difficult problems.

Studies have shown that Unit testing is more cost effective compared to the other stages of testing. The following chart by Capers Jones shows the relative costs of defects found in different testing stages.

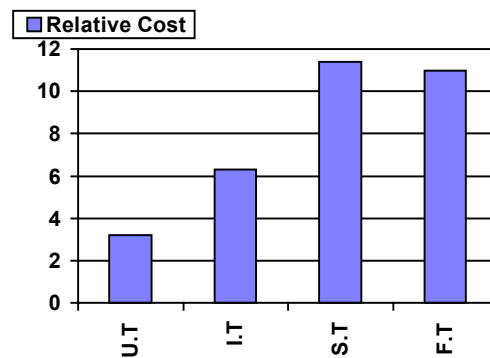


Fig 1: Relative Cost per defect for different Testing stages

The cost includes the effort taken for preparing the test cases, executing the test cases, analysing the results, and fixing the defects [3] This data proves the value of Early phase testing - Unit and Integration testing. This leads to the conclusion that better testing at the early phases is a smarter way to detect and fix defects.

Unit testing can detect and remove a significant portion of the defects. A study by Thayer and Lipow shows that comprehensive path and parameter testing can remove 72.9% of the defects.

Unit testing - some typical problems

Testing is monotonous, boring and repetitive: Unlike “debugging” which is like solving a puzzle, testing is an activity that does not have a predictable end. A Test case may or may not reveal a defect. Testing gets repeated many times. There is no other activity in the software engineering life cycle, which is as repetitive as testing. Each time a change is made on any software, as an enhancement or for fixing a problem it is desirable that all the test cases are repeated. In the absence of any reasonable automation, this is an activity no one enjoys and hence never is done sufficiently well. Testing, unlike other software development activities, suffers from a very poor degree of automation. Surprisingly, while software engineers have gone about automating practically the whole world around them, they have ignored all along one of the most manual and disliked activity in their own backyard. *Automation of as many of the routine activities as possible will help a long way in reducing the monotony of testing. Defining concrete completeness criteria for the testing activity also brings some predictability to testing in the sense that now there is a concrete goal to be achieved.* Simply exhorting the engineers to do better and more complete testing without giving them good tools to reduce the monotony will continue to fail to produce results.

Poor Documentation of Test cases: According to G.J.Myers, “throw away” the test cases only if it is a “throw away” program [4]. Test cases are as valuable an asset as the program itself. Test cases are modified and rerun many times in the future. So, documenting the test cases is very essential for effective future usage. In practice, test cases are not documented adequately. When the code gets changed, the affected test cases may have to be modified and the documentation must be updated. Testing often includes on-the-fly test cases that are invented while executing a pre defined set of test cases. Such test cases are very rarely documented. *It is very useful to ensure that test case documentation gets automated as part of the testing process. This way while testing gets done the documentation continues to get generated.*

Coding Drivers and Stubs: Unit Testing involves writing code for drivers and stubs which together often add up much more code than the unit under test. Testers feel reluctant to write such code that do not go into the final system. The drivers and stubs may have bugs themselves that result in a lot of additional debugging effort. *Automation of code generation for drivers and stubs can result in an useful saving of effort for the tester. It also will ensure that there are no defects in the stubs or drivers that results in avoidable loss of time.*

Informal testing process: While the early part of the software development life-cycle like Analysis and are well defined and widely practiced, the testing process is defined much less formally. Well known and effective testing techniques are often not practiced. While testing accounts for 50% of the total software development efforts, the software engineers get very little formal training on testing. In computer science courses, testing is hardly dealt as an independent discipline. As a result, testing continues to be an informal and ill-understood process. *Combining Functional (Black box testing based on the Specifications), Structural (White box testing based on the structure of the code) and Heuristic (based on human intuition) testing techniques provide much better results*

than simply using an intuitive approach to testing. Many software engineers run a few intuitive test cases - just enough to unearth some defects, then they go into “debugging” these defects. *Testing must be mostly systematic and partly intuitive instead of the general practice of mostly intuitive and partly systematic approach to testing.*

Poor Regression Testing: Testing is a very repetitive activity that needs to be repeated whenever an enhancement or change is made to an existing code. For example, if the Test case No. 15 in a large test suite fails, the problem gets analysed, fixed and the failed test case alone is rerun to validate the fix. Then the tester moves on to run Test case no. 16. During the maintenance and upgradation phase, it is extremely desirable to rerun all the test cases that were successfully run earlier on the program being modified. However, for every fix it is safer to rerun all the affected test cases (better still, all test cases) which have already been successfully run. Testing today is an essentially manual process and regression testing is not normally practiced to the desired degree. *It is very useful to build a capability of retaining automated test cases as a useful resource along with the code. Automation is the only solution to regression testing.*

Lack of Complete Testing tools: While a large number of CASE tools have emerged over the last decade that address the early phases of software development lifecycle, **Testing** has not been so fortunate. Given that Unit testing takes significant portion of the total effort, very few tools are available to help in unit testing. Only coverage analysis tools are available but they address a part of the whole unit testing activity. *Computer Aided Software Testing (CAST) Tools are a fast growing discipline. Good unit test automation tools are beginning to become available. Evolution of such tools achieving a more comprehensive automation of Unit testing activities are likely to help in a big way in solving many of the problems currently faced in Unit testing.*

Unit testing techniques

A number of effective testing techniques are usable in unit testing stage. The testing techniques may be broadly divided into three types:

- Functional Testing
- Structural Testing
- Heuristic or Intuitive Testing

The defects in software can in general be classified as Omissions, Surprises and Wrong Implementations. Omissions are requirements that are missed in the implementation, Surprises are implementation that are not found in the requirements and wrong implementation is incorrect implementation of a requirement.

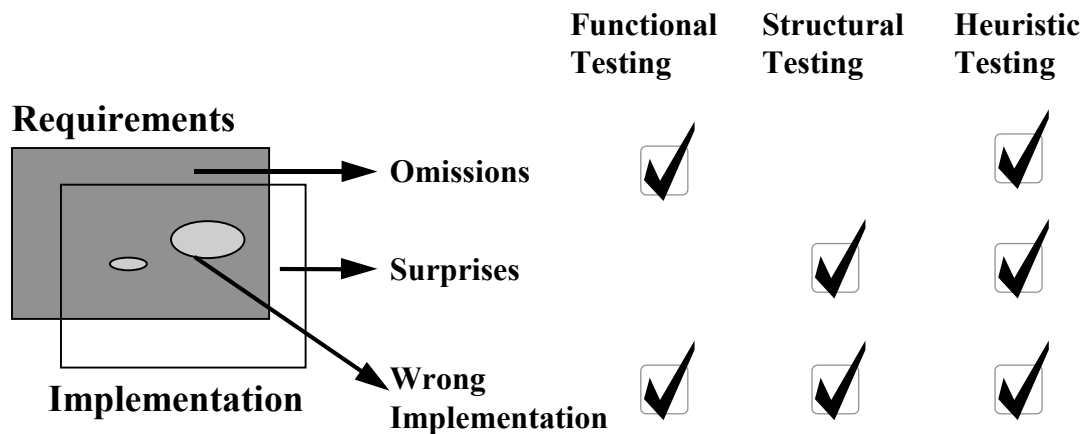


Figure 2: Testing Techniques and Types of Defects

Figure 2 shows the major categories of testing techniques and what types of defects they are effective against. While Functional Testing techniques help catch omissions and wrong implementations, Structural Testing techniques help catch surprises and wrong implementations. Heuristic or intuitive testing techniques help catch all types of defects. But Intuitive testing is effective only when complementing the systematic types of Functional and Structural testing techniques.

Functional testing Techniques (Some examples)

Boundary Value Analysis: Testing the edge conditions of boundaries

Equivalence Partitioning: Grouping test cases into classes in which executing one test cases is equivalent to executing any other test cases in the same group

Cause Effect Graphing: When the behaviour of the unit under test is specified as cause and effect. Design test cases that validate this relationship.

Structural test Cases Techniques (Some examples)

Statement Coverage: Identify Test cases such that every line of code is executed in one test case or other.

Branch Coverage: Identify Test cases such that every branch of code is executed in one test case or other. 100% Branch Coverage automatically assures 100% Statement Coverage.

Condition Coverage: Identify Test cases such that condition in each predicate expression is evaluated in all possible ways.

Modified Condition-Decision Coverage: Identify Test cases such that each Boolean operand can independently affect the outcome of a decision.

Tools for Unit Testing

In the recent times there is an increasing awareness of Test automation. While many test automation tools are available for system testing at the GUI level, few complete tools are available at the Unit and Integration testing stages. The tools available are mainly coverage analysis tools or script based test case specification tools. FasTest [5] is a complete test tool that supports all the internal stages of Unit testing. It finds effective solutions for most of the major problems of unit testing. FasTest provides complete automation for most stages of unit testing.

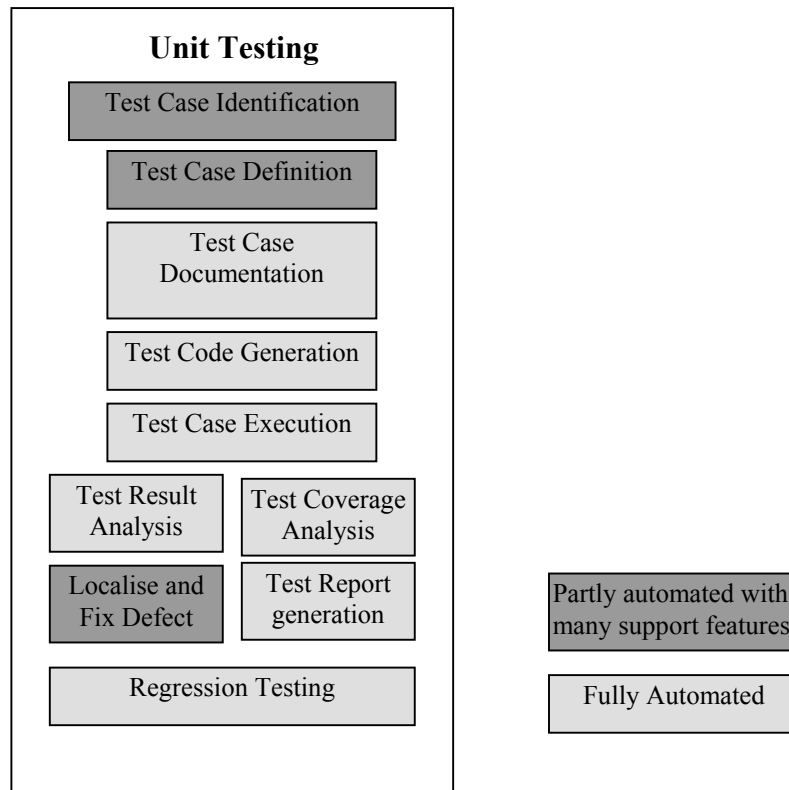


Fig 3: Stages in Unit Testing and level of automation by FasTest

The figure above shows the level of automation provided by FasTest. The product achieves a high level of automation in most phases of testing. With a powerful GUI based interactive test case specification capability, it avoids the need for writing long winding scripts which tend to get more complex than writing the code for the unit under test.

Conclusion

Unit Testing is the earliest stage of testing and is most cost effective testing stage in removing defects. In later stages of testing, detecting and fixing defects is more difficult involving increased effort and time. Unit testing takes significant time and effort. Unit

testing must employ Functional, Structural and Heuristic testing techniques to be effective against the different types of defects. Though effective tools have not supported testing so far, promising test automation tools are beginning to be available. FasTest is one such complete Unit Test automation tool. With the use of good techniques with support from tools like FasTest, Unit testing can be very effective and affordable. It will result in reduction of total efforts while simultaneously increasing the quality of the product significantly also reducing in the long-term maintenance cost and the total life cycle cost. CAST Tools are becoming the key to solve the major problems faced by Unit testers.

References:

- [1] Boris Beizer, System Testing and Software Quality Assurance, Boris Beizer, International Thomson Computer Press, 1996
- [2] Barry W. Boehm, Software Engineering Economics, Prentice Hall, Inc, 1991
- [3] Capers Jones, Applied Software Measurements, McGraw Hill, 1991
- [4] G.J.Myers, The Art of Software Testing, John Wiley & Sons, 1979
- [5] FasTest is a complete Test automation tool from Deccanet Designs Ltd, Bangalore, India.