International Journal of Computer Science and Network (IJCSN)
Volume 1, Issue 1, February 2012 www.ijcsn.org ISSN 2277-5420

Analytical Scenario of Software Testing Using Simplistic Cost Model

[1]**Er. RAJENDER BATHLA**, [2]**Dr. ANIL KAPIL**

[1] Ph.D RESEARCH SCHOLAR DEPTT OF COMPUTER SCIENCE , INDIA

[2] PROF IN DEPTT OF COMPUTER SC & ENGG, HCTM, KAITHAL, 136027 INDIA

Abstract
Software can be tested either manually or automatically. The two approaches are complementary: automated testing can perform a huge number of tests in short time or period, whereas manual testing uses the knowledge of the testing engineer to target testing to the parts of the system that are assumed to be more error-prone. Despite this contemporary, tools for manual and automatic testing are usually different, leading to decreased productivity and reliability of the testing process. Auto Test is a testing tool that provides a "best of both worlds" strategy: it integrates developers' test cases into an automated process of systematic contract-driven testing.

This allows it to combine the benefits of both approaches while keeping a simple interface, and to treat the two types of tests in a unified fashion: evaluation of results is the same, coverage measures are added up, and both types of tests can be saved in the same format. The objective of this paper is to discuss the Importance of Automation tool with associate to software testing techniques in software engineering. In this paper we provide introduction of software testing and describe the CASE tools. The solution of this problem leads to the new approach of software development known as software testing in the IT world. Software Test Automation is the process of automating the steps of manual test cases using an automation tool or utility to shorten the testing life cycle with respect to time.

Keywords
Manual testing, automated testing, testing economics, benefits and costs of automation.

1. Introduction

Software testing is the process of executing a program with the intention of finding errors in the code. It is the process of exercising or evaluating a system or system component by manual automatic means to verify that it satisfies specified requirements or to identify differences between expected and actual results [4]. Software Testing should not be a distinct phase in System development but should be applicable throughout the design development and maintenance phases. 'Software Testing is often used in association with terms verification & validation 'Software testing is the process of executing software in a controlled manner, in order to answer the question: Does the software behave as specified. One way to ensure system's responsibility is to extensively test the system. Since software is a system component it requires a testing process also. The main contribution of this paper lies in the mechanisms that we provide to integrate the manual and automated testing strategies. This integration has the following advantages:

• The overall testing process benefits from the strengths of both manual and automated testing;
• Support for regression testing: any automatically generated tests that uncover bugs can be saved in the same format as manual tests and stored in a regression testing database;[2]
• The measures of coverage (code, dataflow, specification) will be computed for the manual and automated tests as a whole;
• The interface is kept consistent and simple: Auto Test only requires a user to specify the classes that he wants to test.

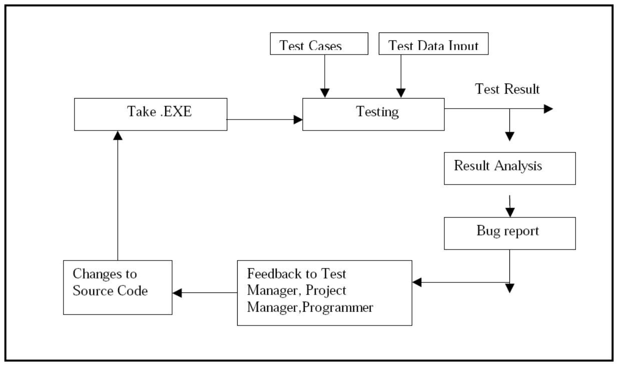


Fig: 1 Model of Manual Testing

Manual unit test cases that are not relevant for any of those classes are automatically filtered out. The paper is organized as follows: the next section contains a general presentation of the manual and automated testing strategies and motivates why they should be combined.

2. Testing strategies

In this section we introduce the two strategies unified by our tool, manual testing and automated testing, then an analysis of the advantages and disadvantages of each, and the rationale for integrating them.

2.1 Manual Testing Scenario

Manual unit testing has established itself as an integral part in modern software development. It only reached a respectable state with the introduction of adequate tool support (the xUnit family of tools, e.g. JUnit for Java, sUnit for Smalltalk, py Unit for Python, and Gobo Eiffel Test for Eiffel). Such frameworks are typically small but they provide significant practical benefits. Manual unit testing frameworks automate test case execution. The test cases themselves (including input data generation and test result verification) need to be created by hand. In manual testing the test team generates various test cases, take the .EXE of the software, and execute the test cases to test each and every functionality. If a defect is found, a bug report is prepared, send it to the project manager, Test manager and to the programmer. The software is modified and the same steps repeated again till the error is removed.[3]

2.2 Automated Testing

Automated tests execute a sequence of actions without human intervention. It is also defined as a testing a system with different data sets again and again without intervention of human. Simply automated testing is automating the manual testing process currently in use. Automation is the use of strategies, tools, and artifacts that augment or reduce the need of manual or human involvement or interaction in repetitive or redundant tasks. Minimally such a process includes: Detailed test cases, including predictable "expected results", which have been developed from Business Functional Specification and Design Documentation. A standalone Test Environment including a Test Database that is restorable to a known constant, such that test cases are able to repeat each time there are modifications made to the application.[1], [4]

3. Problem with Manual Testing

Manual Testing is time consuming.

a) There is nothing new to learn when one tests manually.

b) People tend to neglect running manual tests.

c) None maintains a list of the tests required to be run if they are manual tests.

d) Manual Testing is not reusable.

e) The effort required is the same each time.

f) One cannot reuse a Manual Test.

g) Manual Tests provide limited Visibility and have to be repeated by all Stakeholders.

h) Only the developer testing the code can see the results.

i) Tests have to be repeated by each stakeholder for e.g. Developer, Tech Lead, GM, and Management.

j) Manual Testing ends up being an Integration Test.

k) In a typical manual test it is very difficult to test a single unit.

l) Scripting facilities are not in manual testing.[1]

4. Proposed Automated Tested

Automated testing with Quick Test addresses these problems by dramatically speeding up the testing process. You can create tests that check all aspects of your application or Web site, and then run these tests every time your site or application changes. [3]

Fast: Quick test runs tests significantly faster than human user.

Reliable: Tests perform precisely the same operations each time they are run, thereby eliminating human error.

Programmable: You can program sophisticated tests that bring out hidden information.

Comprehensive: you can build a suite of tests that covers every feature in your web site or application.

Reusable : You can build a suite of tests that covers every feature in your website or application.

5. Auto Test architecture

Auto Test is a framework for fully automated software testing. It allows for arbitrary testing strategies to be plugged in and is not hard coded to a certain testing strategy. The pluggable testing strategy is only concerned with determining exactly how and with what inputs the system under test should be invoked. The actual execution is a task of the framework.[6],[15]

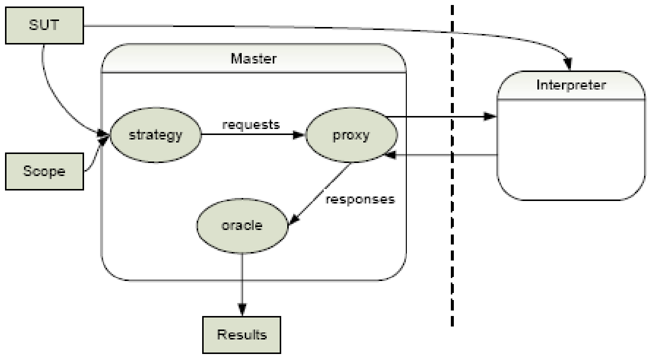


Fig: 2 Auto Test Architecture

6. Analytical On Overly Simplistic Cost Models for Automated Software Testing

Accurate estimates of the return on investment of test automation require the analysis of costs and benefits involved. However, since the benefits of test automation are particularly hard to quantify, many estimates conducted in industrial projects are limited to considerations of cost only. In many cases the investigated costs include: the costs for the testing tool or framework, the labor costs associated with automating the tests, and the labor costs associated with maintaining the automated tests. These costs can be divided into fixed and variable costs. Fixed costs are the upfront costs involved in test automation. Variable costs increase with the number of automated test executions. In [7], a case study originally published by Linz and Daigl [14] is presented, which details the costs for test automation as follows: *V := Expenditure for test specification and implementation D := Expenditure for single test execution* Accordingly, the costs for a single automated test (*Aa*) can be calculated as: $Aa := Va + n * Da$ whereby *Va* is the expenditure for specifying and automating the test case, *Da* is the expenditure for executing the test case one time, and *n* is the number of automated test executions. Following this model, in order to calculate the break-even point for test automation, the cost for manual test execution of a single test case (*Am*) is calculated similarly as $Am := Vm + n * Dm$ whereby *Vm* is the expenditure for specifying the test case, *Dm* is the expenditure for executing the test case and *n* is the number of manual test executions. The break-even point for test automation can then be calculated by comparing the cost for automated testing (*Aa*) to the cost of manual testing (*Am*) as: $E(n) := Aa / Am = (Va + n * Da)/ (Vm + n * Dm)$ According to this model, the benefit of test automation seems clear: "From an economic standpoint, it makes sense to automate a given test only when the cost of automation is less than the cost of manually executing the test the same number of times that the automated test script would be executed over its lifetime." Figure 1 depicts this interrelation. The x-axis shows the number of test runs, while the y-axis shows the cost incurred in testing. The two curves illustrate how the costs increase with every test run. While the curve for manual testing costs is steeply rising, automated test execution costs increase only moderately. However, automated testing requires a much higher initial investment than manual test execution does. According to this model, the break-even point for test automation is reached at the intersection point of the two curves. This "universal formula" for test automation costs has been frequently cited in software testing literature (e.g. [7], [8], [15]) and studies to argue in favor for test automation.

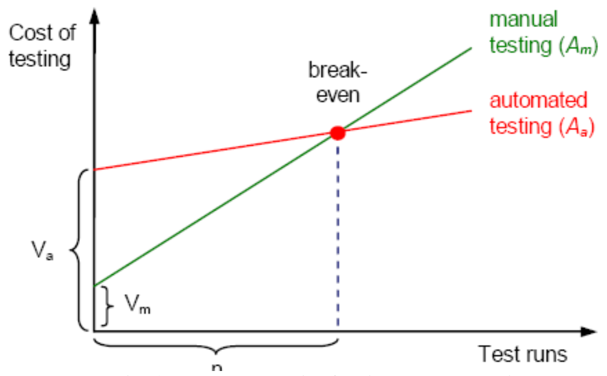


Fig: 3 Break-even point for Automated Testing

Depending on the author, the quoted number of test runs required to reach the break-even point varies from 2 to 20. The logic of this formula is appealing and – in a narrow context – correct. "As a simple approximation of costs, these formulas are fair enough. They capture the common observation that automated testing typically has higher upfront costs while providing reduced execution costs." [12] For estimating the investment in test automation, however, it is flawed for the following reasons:

• *Only costs are analyzed* – The underlying model compares the costs incurred in testing but excludes the benefits. Costs and benefits are both required for an accurate analysis, especially when the analyzed alternatives have different outcomes. This is true for test automation, since manual testing and automated testing follow different approaches and pursue different objectives (e.g., exploring new functionality versus regression testing of existing functionality).

• *Manual testing and automated testing are incomparable* – Bach [2] argues that "hand testing and automated testing are really two different processes, rather than two different ways to execute the same process. Their dynamics are different, and the bugs they tend to reveal are different. Therefore, direct comparison of them in terms of dollar cost or number of bugs found is meaningless."

• *All test cases and test executions are considered equally important* – Boehm criticizes in his agenda on value-based software engineering [4]: "Much of current software engineering practice and research is done in a value-neutral setting, in which every requirement, use case, object, test case, and defect is equally important." In a real-world project, however, different test cases and different test executions have different priorities based on their probability to detect a defect and on the impact which a potential defect has on the system under test.

• *Project context is not considered* – The decision about whether or not to automate testing is restricted to a single, independent test case. Nevertheless, the investment decision has to be made in context of the particular project situation, which involves the total set of test cases planned and the budget and resources allocated for testing.

• *Additional cost factors are missing* – A vast number of additional factors influencing costs and benefits are not considered in the overly simplistic model [11]. Examples are costs of abandoning automated tests after changes in the functionality, costs incurred by the increased risk of false positives, or total cost of ownership of testing tools including training and adapting workflows.

7. Opportunity Cost in Test Automation

In this section we present a fictitious example to illustrate the problems listed in the previous section. Please note that this example simplifies a complex model to highlight and clarify some basic ideas. We discuss and expand the model in sections 4 and 5 where we add further details and propose influencing factors typically found in real-world projects. The example describes a small system under test. The effort for running a test manually is assumed to be 0.25 hours on average. For the sake of simplicity, we assume no initial costs for specification and preparation. Automating a test should cost 1 hour on average, including the expenditures for adapting and maintaining the automated tests upon changes. Therefore, in our example, running a test automatically can be done without any further effort once it has been automated. According to the model presented in the previous section, the break-even point for a single test is reached when the test case has been run four times.

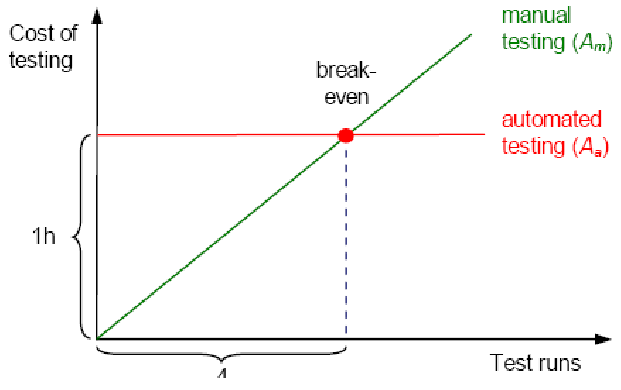


Fig 4: Break-even point for a single test case

Furthermore, for our example let us assume that 100 test cases are required to accomplish 100 percent requirements coverage. Thus it takes 25 hours of manual testing or 100 hours of automated testing to achieve full coverage. Comparing these figures, the time necessary to automate all test cases is sufficient to execute all test cases four times manually. If we further assume that the project follows an iterative (e.g. [14]) or agile (e.g. [10]) development approach, we may have to test several consecutive releases. To keep the example simple, we assume that there are 8 releases to be tested and each release requires the same test cases to be run. Consequently, completely testing all 8 releases requires 200 hours of manual testing (8 complete test runs of 25 hours each) or 100 hours to automate the tests (and running these tests "infinitely" often without additional costs).
Taken from the authors' experience, the average time budget available for testing in many industrial projects is typically far less – about 75 percent at most – than the initially estimated test effort. In our example we therefore assume a budget of 75 hours for testing. Of course, one could argue that a complete test is not possible under these limitations. Yet many real-world projects have to cope with similar restrictions; fierce time-to-market constraints, strict deadlines, and a limited budget are some of the typical reasons. These projects can only survive the challenge of producing tested quality products by combining and balancing automated and manual testing. Testing in the example project with a budget of 75 hours would neither allow to completely test all releases manually nor to completely automate all test cases. A trade-off between completely testing only some of the releases and automating only a part of the test cases is required. In economics, this trade-off is known as the "production possibilities frontier".

Figure 3 shows the combinations of automated and manual test cases that testing can possibly accomplish, given the available budget and the choice between automated and manual testing. Any combination on or inside the frontier is possible. Points outside the frontier are not feasible because of the restricted budget. Efficient testing will choose only points on rather than inside the production possibilities frontier to make best use of the scarce budget available.

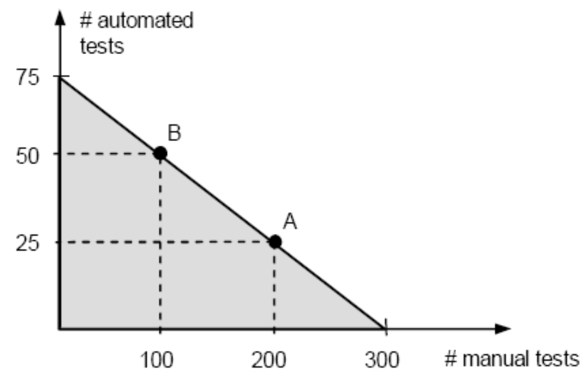


Fig 5: Production possibilities frontier for an exemplary test budget of 75 hours

The production possibilities frontier shows the trade-off that testing faces. Once the efficient points on the frontier have been reached, the only way of getting more automated test cases is to reduce manual testing. Consequently Marick [10] raises the following question: "If I automate this test, what manual tests will I lose?" When moving from point A to point B, for instance, more test cases are automated but at the expense of fewer manual test executions. In this sense, the production possibilities frontier shows the opportunity cost of test automation as measured in terms of manual test executions. In order to move from point A to point B, 100 manual test executions have to be abandoned. In other words, automating one test case incurs opportunity costs of 4 manual test executions. [9]
Test runs breakeven 4 1h manual testing (*Am*) automated testing (*Aa*) Cost of testing 75 100 300 # manual tests B 50 25 200 A# automated tests 87

8. A Cost Model Based on Opportunity Cost

Building on the example from the previous section, we propose an alternative cost model drawing from linear optimization. The model uses the concept of opportunity cost to balance automated and manual testing. The opportunity cost incurred in automating a test case is estimated on basis of the lost benefit of not being able to run alternative manual test cases. Hence, in contrast to the simplified model presented in Section 2, which focuses on a single test case, our model takes all potential test cases of a project into consideration. Henceforth, it optimizes the investment in automated testing in a given project context by maximizing the benefit of testing rather than by minimizing the costs of testing.[7]

8.1 Fixed Budget

First of all, the restriction of a fixed budget has to be introduced to our model. This restriction corresponds to the production possibilities frontier described in the previous section. R1: *na * Va + nm * Dm* $\leq$ *B na := number of automated test cases nm := number of manual test executions Va := expenditure for test automation Dm := expenditure for a manual test execution B := fixed budget* Note that this restriction does not include any fixed expenditures (e.g., test case design and preparation) manual testing. Furthermore, with the intention of keeping the model simple, we assume that the effort for running an automated test case is zero or negligibly low for the present. This and other influence factors (e.g., the effort for maintaining and adapting automated tests) will be discussed in the next section. This simplification, however, reveals an important difference between automated and manual testing. While in automated testing the costs are mainly influenced by the number of test cases (*na*), manual testing costs are determined by the number of test executions (*nm*). Thus, in manual testing, it does not make a difference whether we execute the same test twice or whether we run two different tests. This is consistent with manual testing in practice – each manual test execution usually runs a variation of the same test case [6]

8.2 Benefits and Objectives of Automated and Manual Testing

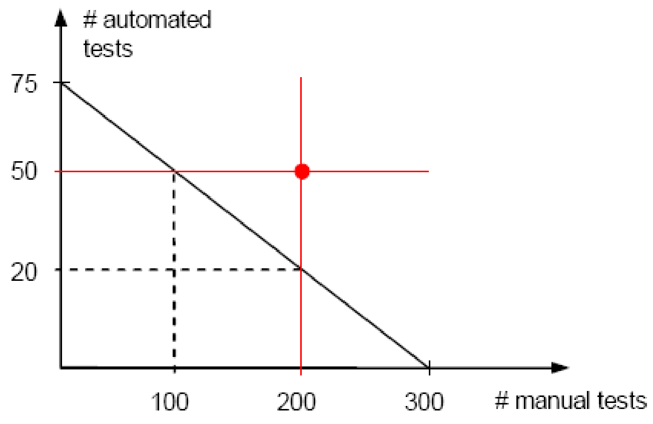
Second, in order to compare two alternatives based on opportunity costs, we have to valuate the benefit of each alternative, i.e., automated test case or manual test execution. The benefit of executing a test case is usually determined by the information this test case provides. The typical information is the indication of a defect. Still, there are additional information objectives for a test case (e.g., to assess the conformance to the specification). All information objectives are relevant to support informed decision making and risk mitigation. A comprehensive discussion about what factors constitute a good test case is given in [13].

8.3 Maximizing the Benefit

Third, to maximize the overall benefit yielded by testing, the following target function has to be added to the model. T: *Ra(na) + Rm(nm)* □ *max* Maximizing the target function ensures that the combination of automated and manual testing will result in an optimal point on the production possibilities frontier defined by restriction *R1*. Thus, it makes sure the available budget is entirely and optimally utilized.

8.4 Example

To illustrate our approach we extend the example used in Section 3. For this example the restriction *R1* is defined as follows. R1: *na * 1 + nm * 0.25* $\leq$ *75* To estimate benefit of automated testing based on the risk exposure of the tested object, we refer to the findings published by Boehm and Basili [5]: "Studies from different environments over many years have shown, with amazing consistency, that between 60 and 90 percent of the defects arise from 20 percent of the modules, with a median of about 80 percent. With equal consis- tency, nearly all defects cluster in about half the modules produced." Accordingly we categorize and prioritize the test cases into 20 percent highly beneficial, 30



percent medium beneficial, and 50 percent low beneficial and model following alternative restrictions to be used in alternative scenarios. R2.1: *na* $\geq$ *20* R2.2: *na* $\geq$ *50* To estimate the benefit of manual testing we propose, for this example, to maximize the test coverage. Thus, we assume an evenly distributed risk exposure over all test cases, but we calculate the benefit of manual testing based on the number of completely tested releases. Accordingly we categorize and prioritize the test executions into one and two or more completely tested releases. We model following alternative restrictions for alternative scenarios. R3.1: *nm* $\geq$ *100* R3.2: *nm* $\geq$ *200* Based on this example we illustrate three possible scenarios in balancing automated and manual testing. Figures 4a, 4b and 4c depict the example scenarios graphically.

• **Scenario A** – The testing objectives in this scenario are, on the one hand, to test at least one release completely and, on the other hand, to test the most critical 50 percent of the system for all releases. These objectives correspond to the restrictions *R3.1* and R2.2 in our example model. As shown in Figure 4a the optimal solution is point *S1* (*na* = 50, *nm* = 100) on the production possibilities frontier defined by *R1*. Thus, the 50 test cases referring to the most critical 50 percent of the system should be automated and all test cases should be run manually once.

• **Scenario B** – The testing objectives in this scenario are, on the one hand, to test at least one release completely and, on the other hand, to test the most critical 20 percent of the system for all releases. These objectives correspond to the restrictions *R3.1* and *R2.1* in our example model. As shown in Figure 4b any point within the shaded area fulfills these restrictions. The target function, however, will make sure that the optimal solution will be a point between *S1* (*na* = 50, *nm* = 100) and *S2* (*na* = 20, *nm* = 220) on the production possibilities frontier defined by *R1*. Note: While all points on *R1* between the *S1* and *S2* satisfy the objectives of this scenario, the point representing the optimal solution depends on the definition of the contribution to risk mitigation of automated and manual testing, *Ra(na)* and *Rm(nm)*.

• **Scenario C** – The testing objectives in this scenario are, on the one hand, to test at least two releases completely and, on the other hand, to test the most critical 50 percent of the system for all releases. These objectives correspond to the restrictions *R3.2* and *R2.2* in our example model. As shown in Figure 4c a solution that satisfies both restrictions cannot be found.

Figure 6: Example scenario A

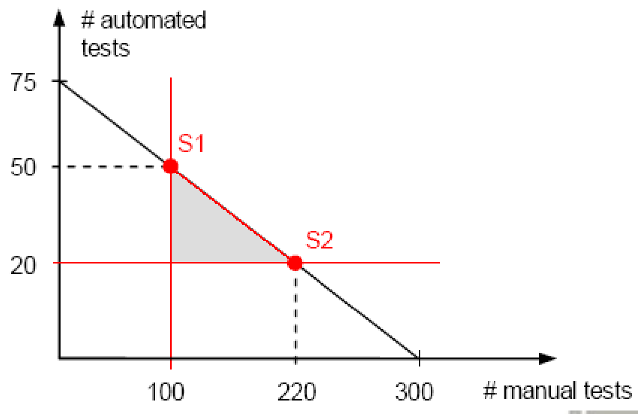


Figure 6: Example scenario B

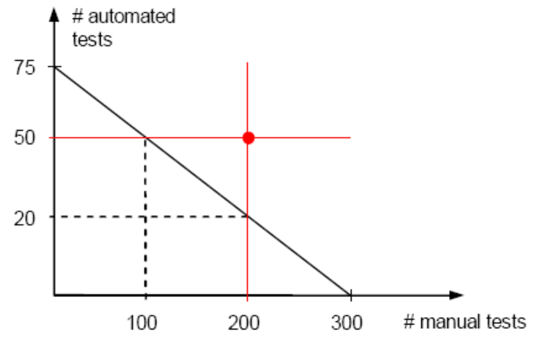


Figure 6: Example scenario C

9. CONCLUSION

In this paper we discussed cost models to support decision making in the trade-off between automated and manual testing. We summarized typical problems and shortcomings of overly simplistic cost models for automated testing frequently found in literature and commonly applied in practice:

• Only costs are evaluated and benefits are ignored
• Incomparable aspects of manual testing and automated testing are compared
• All test cases and test executions are considered equally important
• The project context, especially the available budget for testing, is not taken into account.

10. References

[1] Innovative approaches of automated tools in software testing and current technology as compared to manual testing, Global journal of enterprise of information system, jan 2009-june 2009.

[2] Leckraj Nagowah and Purmanand Roopnah, "AsT -A Simp le Automated System Testing Tool", IEEE, 978-1-4244- 5540-9/10, 2010.

[3] Alex Cerv antes, "Exploring the Use of a Test Automation Framework", IEEEAC p ap er #1477, version 2, up dated January 9, 2009.

[4] A. Ieshin, M. Gerenko, and V. Dmitriev, *"Test Automation- Flexible Way",* IEEE, 978-1-4244-5665-9, 2009.

[5] Boehm, B., *Value-Based Software Engineering: Overview and Agenda*. In: Biffl S. et al.: Value-Based Software Engineering. Springer, 2005.

[6] Schwaber, C., Gilpin, M., *Evaluating Automated Functional Testing Tools*, Forrester Research, February 2005.

[7] Ramler R., Biffl S., Grünbacher P., *Value-based Management of Software Testing*. In: Biffl S. et al.: Value-Based Software Engineering. Springer, 2005.

[8] M.Grechanik, q. Xie, and Chen Fu, "*Maintaining and Evolving GUI- Directed Test Scripts*", IC SE'09, I EEE, Vancouver, Canada, 978-1-4244-3452-7, May 16-24, 2009.

[9] Khaled M.Mustafa, Rafa E. Al-Qutaish, Mohammad I. Muhairat, "*Cassification of Software testing Tools Based on the Software Testing Methods*", 2009 second International Conference on Computer and Electrical Engineering, 978-0-7695-3925-6, 2009.

[10] R.S.Pressman, " Software Engineering A Practitioner's Approach", Mcgraw-Hill International Edition, ISBN 007-124083-7.

[11] D. Marinov and S. Khurshid, "Test Era: A Novel Framework for Automated Testing of Java Programs," in *Proc.~16th IEEE International Conference on Automated Software Engineering (ASE)*, 2001, pp. 22-34

[12] P. Tonella, "Evolutionary testing of classes," in *International symposium on Software testing and analysis (ISSTA'04)*. Boston, Massachusetts, USA: ACM Press, 2004, pp. 119-128.

[13] N. K. Patrice Godefroid, Koushik Sen, "DART: directed automated random testing," presented at PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, 2005.

[14] Dustin, E. et. al., *Automated Software Testing*, Addison- Wesley, 1999.

[15] Fewster, M., Graham, D., *Software Test Automation: Effective Use of Text Execution Tool*, Addison-Wesley, 1999.

First Author
Er. Rajender Bathla Ph.D Research Scholar in the deptt of Computer Science & Engineering India .His area of specialization in Software Testing. He did completed his master degree M.Tech from M.M.University Mullana in 2009. Presently working as Asstt Prof in Computer Sc & Engg Deptt at HIET Kaithal. He has more than 35 research papers in reputed conferences and journals.

Second Author
Dr. Anil Kapil is working as Professor in the Deptt of CSE at HCTM Kaithal. He did complete his doctorate degree in 2006 in the area of distributed data bases. He has more than 40 research papers in a reputed journals. He got completed more than 6 candidates of Ph.d and more than 20 candidates of M.Phil and M.Tech degree under his supervision.