Project Dissertation Report on

COST-BENEFIT ANALYSIS OF AUTOMATION TESTING

Submitted By:

ARNA CHATTERJEE

2K16/MBA/09

Under the Guidance of:

Ms Deep Shree

PROFESSOR

DELHI SCHOOL OF MANAGEMENT

Delhi Technological University

Bawana Road Delhi 110042

2016-2018

CERTIFICATE FROM THE INSTITUTE

This is to certify that the report titled “Cost-Benefit Analysis of Automation Testing” is an original and bonafide work carried out by Ms. Arna Chatterjee of MBA 2016-18 batch and was submitted to Delhi School of Management, Delhi Technological University, Bawana Road, Delhi-110042 in partial fulfillment of the requirement for the award of the Degree of Masters of Business Administration.

_____
Signature of HOD (DSM)
(Dr. Rajan Yadav)

_____
Signature of Internal Guide
(Ms. Deep Shree)

Place:
Date:

DECLARATION

I hereby declare that the Project Report entitled **"Cost-Benefit Analysis of Automation Testing"** has not been submitted to any other institute or University for the award of any degree/diploma etc. This work embodies the result of my original work conducted under the supervision of Ms Deep Shree (Faculty of Delhi School Of Management, DTU, Delhi). The information submitted is true and original to the best of my knowledge.

NAME – ARNA CHATTERJEE

ROLL NUMNBER - 12/MBA/09

SPECIALIZATION - FINANCE & INFORMATION TECHNOLOGY MANAGEMENT

ACKNOWLEDGEMENT

I have prepared this study paper for the **"Cost-Benefit Analysis of Automation Testing".** Frankly, I have derived the contents and approach of this study paper through discussions with colleagues who are also the students of this course as well as with the help of various Books, Magazines and Newspapers etc.

I would like to give my sincere thanks to **Ms Deep Shree (Faculty of DELSHI SCHOOL OF MANAGEMENT , DTU , Delhi)** and a host of friends and the teachers who, through their guidance, enthusiasm and counseling helped me enormously. As I think there will always be a need of improvement. Apart from this, I hope this study paper would stimulate the need of thinking and discussion on the topics like this one.

SIGNATURE -
NAME – ARNA CHATTERJEE
ROLL NUMBER - 2016/MBA/09
SPECIALIZATION - FINANCE & ITM

ABSTRACT

The objective of this thesis is to put forth a case study on the profitability of Automation Testing for the advantage of the IT industry in India. The cost-benefit analysis has been done by considering the costs and benefits of automation testing, before the final software release. The potential benefits of test automation with regards to software quality after customer release were not assessed.

Test automation is a significant venture which often requires dedicated resources. The investment in test automation can lead to major cost savings by minimizing the need for manual testing labour, especially when the software is developed with an agile methodology. It can diminish the cost of rework of software development, as test automation helps in the detection of defects as early as possible. Automation Testing has many disadvantages as well, such as test maintenance and test feasibility of the software product, and if these errors are not given attention, then the investment in test automation may end up being worthless or it might even deliver negative outcomes. The results of this thesis recommend that automation is extremely profitable and productive at the organization under study.

This thesis is going to focus on the question “When should a test be automated?” as well as the trade-off between automation and manual testing. The paper also studies and analyzes the issues in the simple benefit and cost models that are generally used to make decisions. Here an alternate model is introduced that is based on opportunity cost and factors that influence automation. The aim is to stimulate discussion concerning these factors and their influence on the advantages and costs of automation testing.

Contents

INTRODUCTION

Software testing is a procedure of examining an application or a program with the goal of finding the software defects and bugs.

It can likewise be expressed as the process of validating and verifying that a software program or application or product:

- Meets the business and technical requirements that led to it's design and development
- Works as expected, and not surprisingly
- Can be implemented and actualized with the same characteristic.

Software Testing is necessary since humans commit errors. Ideally, we ought to get another person to check our work because another person is more likely to spot the flaws.

Software Testing has diverse objectives. The significant ones are as per the following:

- Discovering defects made by the developers.
- Earning trust and providing information about the level of quality.
- To avert and check defects.
- To ensure that the end result meets the business and client prerequisites.
- To guarantee that it fulfills the Business Requirement Specification and System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product

“Verification makes sure that the product is designed to deliver all functionality to the customer whereas **Validation** is determining if the system complies with the requirements and performs functions for which it is intended and meets the organization’s goals and user needs.”

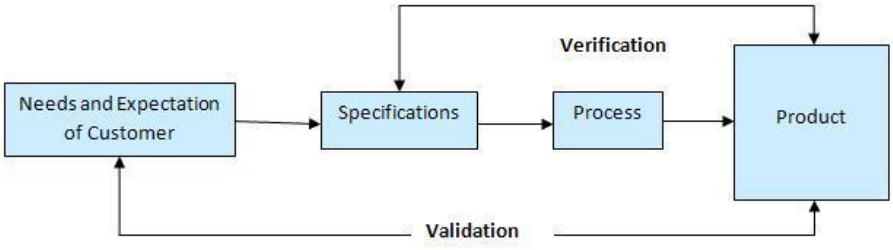

Figure 1: Verification and Validation

Automation testing is the process of using software tools to execute tests scripts on a software application before its release into the production phase. The goal of is to simplify the effort put into testing as much as possible. Unit testing consumes a lot of resources, and hence is a good contender for automation. The tools of automation testing are generally equipped to execute tests scripts, report defects and compare the results with previous test runs. The tests that are carried out with these tools can be run over multiple times, at any time of the day.

LITERATURE REVIEW

Automated software testing is considered to be an effective and reliable method of ensuring software quality and enhancing the reliability of automation when compared to manual testing. It is a critical part of the Software Development Life Cycle (SDLC) through the entire software development process from requirements, system on the grounds that it goes through the entire software improvement process pre-requisites, system design, coding and test execution. There is a great deal of work engaged with test process administration, defect management and training of testers as organizations endeavor to deliver testing in a profitable way. Past studies show that the cost and effort during testing roughly represents 50% of aggregate cost and effort.

Numerous hypotheses have been proposed to clarify best practices in automation. Though the writing covers a wide assortment of such theories, this review will only focus on five subjects throughout the entire study. These topics are: automation testing, automation testing significance, the benefits that could happen to adopting organization, understanding that should be focused upon and success factors for effective implementation of automation testing.

With the growing prevalence of computer application in organizations, the extent of programming is ceaselessly expanding and the logic is increasing in unpredictability. Therefore, automation testing is also becoming more critical. We realize that manual testing is a highly labor-intensive and tedious task because it is more time-consuming, intense, exceedingly inclined to errors and restrictions in test case coverage. Unfortunately, manual testing can’t adapt to the rapid advancements in the realm of innovation. Therefore, it is imperative to consider what procedures can be replaced by automated testing.

1 SOFTWARE DEVELOPMENT AND SOFTWARE TESTING

1.1 Phases and models of software development

Software testing and test automation's productivity and profitability are the subjects under study during this paper work, but testing is only one stage in the development of software. Every software application development goes through the similar phases:

1. Conception
2. Requirements gathering
3. Planning and Design
4. Coding and debugging
5. Testing
6. Release
7. Maintenance/software evolution
8. Retirement

There are a few models that help the management of software development projects, which combine at least two or more phases described in the list above. Generally these models are either conventional, so called **plan-driven models** or the more up-to-date **agile models**. Plan-driven models are more clearly defined in the stages of development, require more documentation of each stage and have more requirements on finishing a stage before moving on to the next stage of development. Waterfall model is the most traditional plan-driven model, which goes through all of the phases above in a sequential order.

Figure 1.1 demonstrates the traditional waterfall method of software development. It begins by gathering the requirements, which can be additionally divided to functional and non-functional requirements. Functional requirements are often depicted as use cases i.e. scenarios of a user interacting with the software. Non-functional requirements can for instance contain performance characteristics or software/hardware environments that it should support. In the design phase the development team creates a detailed design of the system, which is utilized in the next coding stage i.e. translating the design into actual software. In the testing phase diverse components are integrated and tested as a system. When the software testing phase is completed, the software is given to the customers and support phase starts i.e. educating the customers in the use of the software, or how to fix software defects.

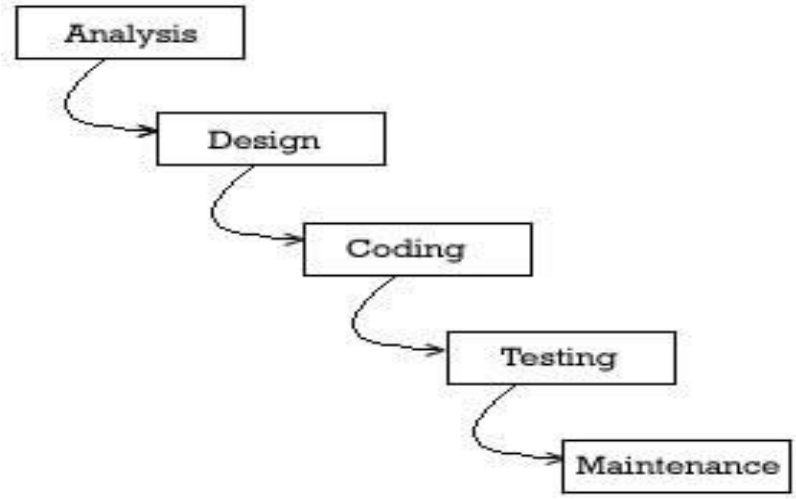

Figure 1.1. SDLC Waterfall Model

The traditional Waterfall model show certain limitations because testing takes place towards the end of the SDLC. To overcome this problem, the V-model was developed. The V-Model is an augmentation of the waterfall model and depends on the relationship of a testing stage for each corresponding development stage. This implies that every phase in the development cycle is directly related to those in the testing phase. This is a highly-disciplined model and the next stage starts only after end of the previous stage.

In the V-Model, the testing phase and the development phase are correlated to each other and are arranged in a parallel manner. Hence, there is the Verification phase on one side of the 'V' and the Validation phase on the other side. The Coding Phase combines the two arms of the V-Model.

The diagram below demonstrates the different phases of V-Model.

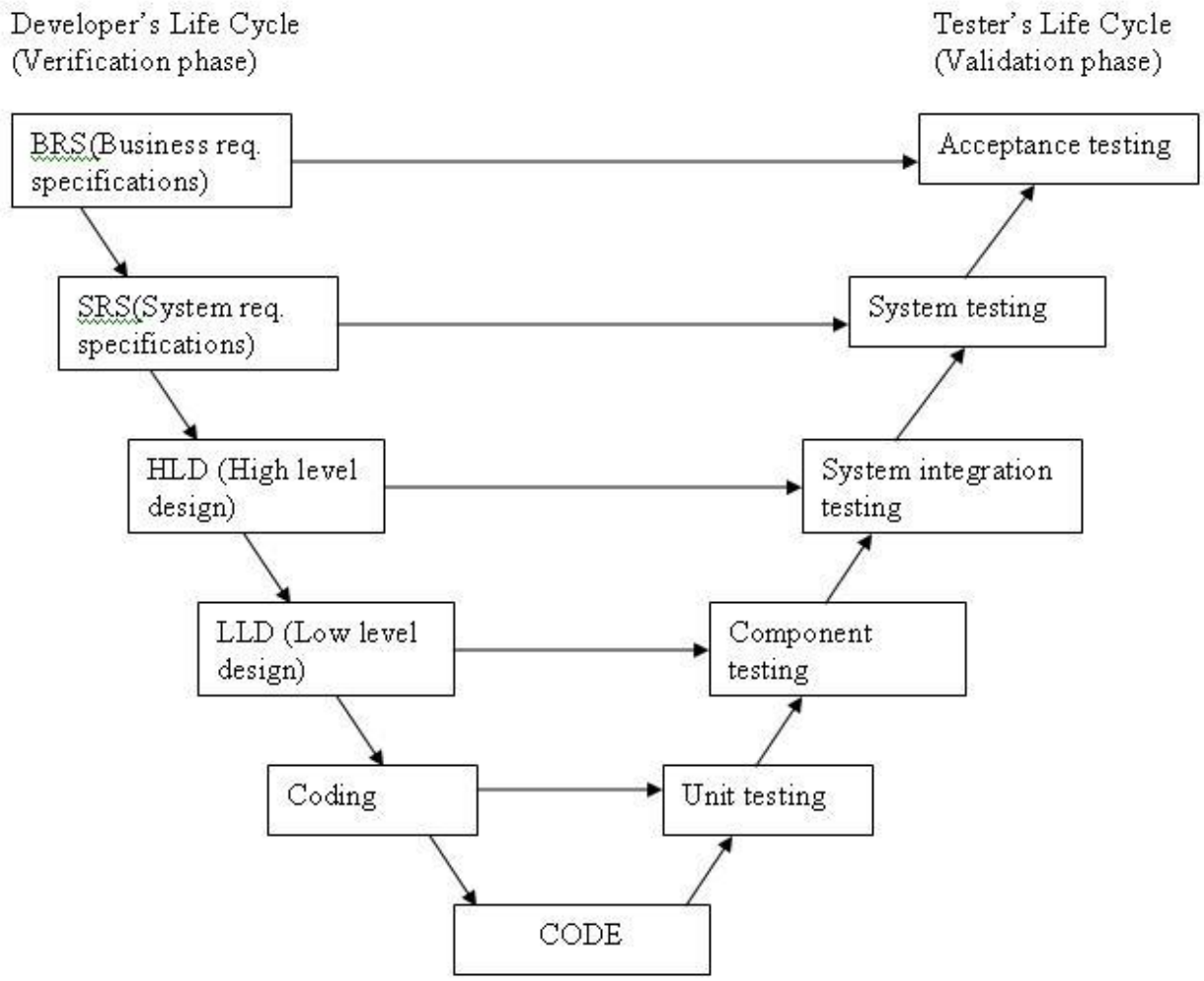

Figure 1.2. V-model

Agile method strategies are basically different as compared to other plan-driven methods because they are *incremental*, with the assumption that small and frequent releases make the product more robust. They have a tendency to likewise have less documentation and the stages tend to blur together more frequently than with plan-driven models. The primary purposes of agile methods against conventional plan-driven models are that they 1) handle changing requirements throughout the development cycle and 2) deliver software under budget limitations and deliver software products quicker.

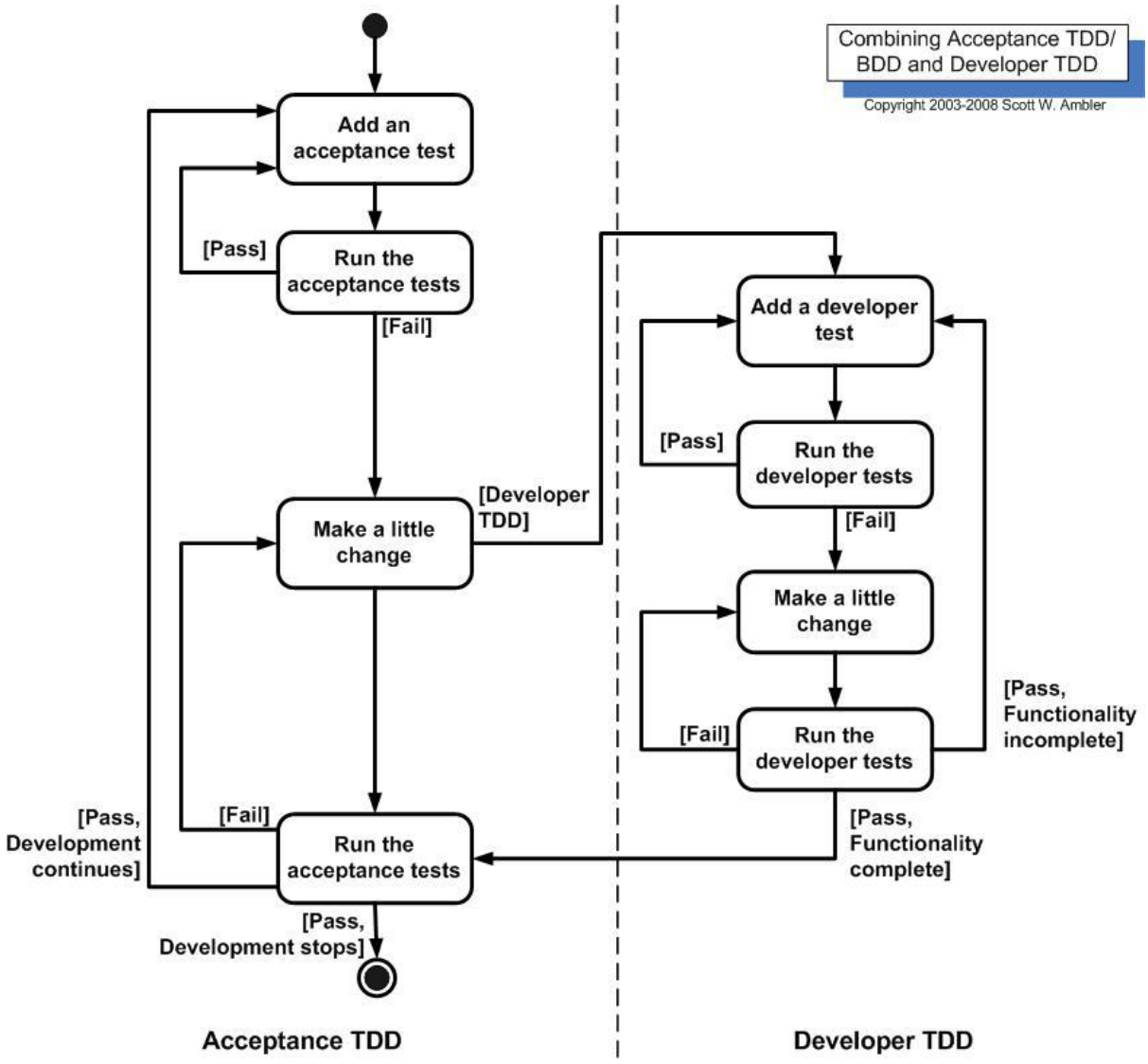

Figure 1.3. Acceptance Testing phases

1.2 Definition and importance of software testing

According to one definition, **software testing** "*is the process of executing a program with the intent of finding errors.*" According to Huizinga & Colawa (2007, p. 249), "*Testing is the process of revealing defects in the code with the ultimate goal of establishing that the software has attained a specified degree of quality*."

The words **defect** and **bug** with regards to software development mean basically the same thing: that there is any sort of "*flaw in the specification, design or implementation in a software product*" that should be detected and repaired so that software product works accurately. During this paper work the word "**defect**" is frequently used when discussing an imperfection in the software. The basis of testing is to discover defects and check that the software is working appropriately.

Software quality consists of numerous elements, for example, efficiency and reusability. Customer satisfaction is one of the most important measure of that. Nonetheless, the defect rate is so vital that if it is not in an acceptable level, the other variables lose their significance.

Testing provides feedback for the following stakeholders:

1. *Customers and users* of the software get figures on to what extent software meets its mutually agreed requirements.
2. *Marketing and product managers* can utilize the statistics from testing to plan releases, pricing, promotion and distribution.
3. *Project supervisors* get information to support risk management and project's progress estimation.
4. *Quality managers* receive details to help process improvement and strategies for quality assurance.
5. *Developers* of the software need the information from testing to be sure that their implementation is done accordingly.
6. *Requirements engineers* require the data from testing to validate and verify the software's requirements.

The defects in software may be found during different **testing levels** of software development either inside the software developing department or of the products already in use, found by the clients.

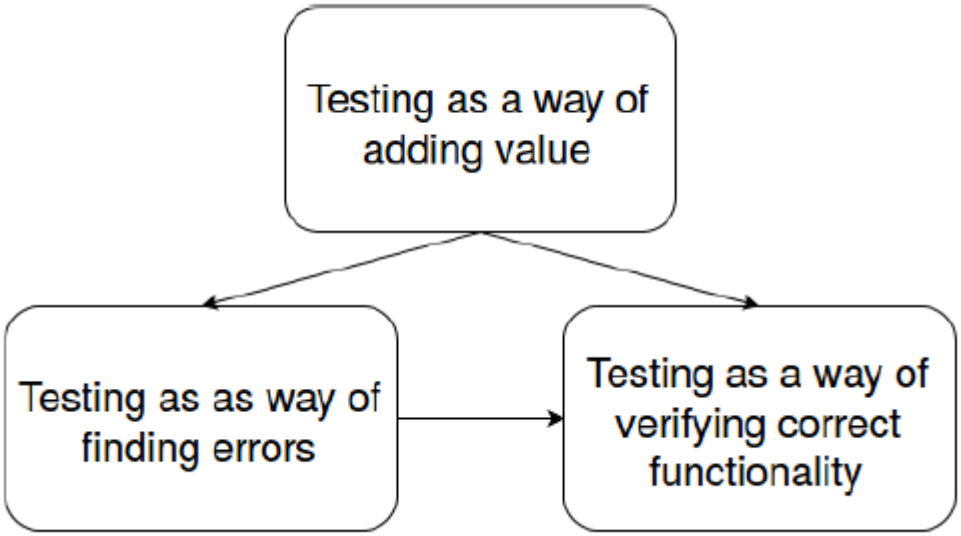

Figure 1.4. The relation between testing philosophies

Here are the different the levels of software testing as follows:

**** Unit testing:** testing of a unit of the software that is further classified as:

- **White box testing**, which is done to software's internal structures in order to uncover development defects and security exposures.
- **Black box testing**, which confirms that certain particular inputs produce desired outputs; internal structure of the software is unknown. Black box testing can also be done to all other test level as mentioned below.

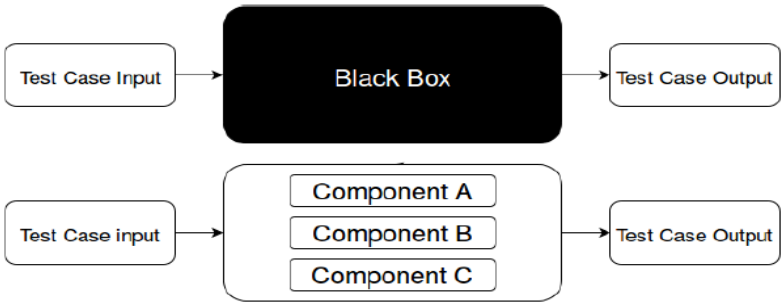

Figure 1.5. The difference between blackbox an whitebox testing

**** Integration testing:** various diverse units, sub-modules and modules of the software are tested together to check that they work accurately together.
**** System testing:** hardware and software are integrated to check that the system meets its pre-requisites and requirements.

There are additionally a couple of other generally used software testing levels and techniques:

- **Release testing:** system is tested as a whole to verify that it is ready for use outside of the development team, mostly to the clientsand users of the product.
    - **Release** is basically a new version of software, which can for example be done because of new fixes to an previous release of the software.
    - The difference between *release testing* and *system testing* is that release testing must be done by a different team that was not involved in the development of the system.
- **Acceptance testing:** software is tested to verify that it meets its acceptance criteria with the customer.
- **Retesting** (from now on called **regression testing**)

is not a testing stage but rather a process, where the tests that were executed in the earlier version of the software are now currently executed on the current version.

Software testing can be done either manually or automatically. **Manual testing** is done by a person, a *tester*, who works as the end user of the software. **Automation testing** implies that the particular software executes as well as creates the tests and delivers their results. Automation testing may sound less difficult than it really is, because it is often a time and resource consuming activity that has many constraints.

Software testing is compared to discovering insects (or “bugs” as they are usually called) in a house: on the off chance that you discover them, there are presumably more of them; if you don’t find any, you still cannot state that no bugs exist. It is a reasonable analogy of software testing as well, since there are numerous inputs, outputs and possible paths to go through the software.

Software testing is hence a risky activity, because it is not possible to test everything and so, some defects might be missed by software testing activities before the release the software to the customers. Figure 1.6 depicts the optimal measure of testing with the expenses of testing and the costs of missed defects, which varies between different software projects.

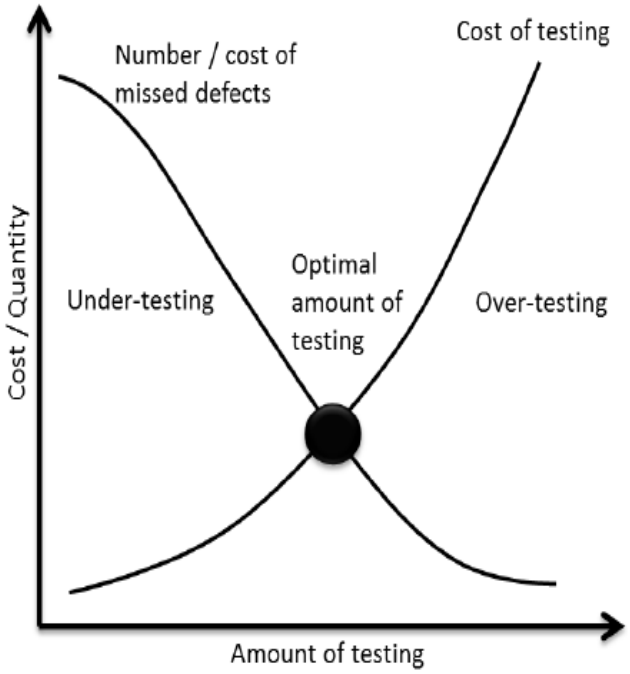

Figure 1.6. Optimal amount of testing required in SDLC

1.3 Manual Testing

Manual testing is software testing where the tester needs to put themselves in the end user's position and test the software in such a way as the client would. This implies that the testing process is human-present and that the tester utilizes real data for input and real environments to test the software so as to create actual usage scenarios. Manual testing is mostly used to find defects in the business logic of the software, i.e. the code that actualizes the client requirements.

Manual testing is necessary when there are just numerous possible scenarios, for instance large ranges of input with too many fields of input, so the tester needs to use their brains to find out the most possible ranges of inputs.Usually, manual testing is guided by a script document that provides instructions and details on how to run

the tests. These archives are very specific, which gives sequential step-wise instructions on input, actions and expected results, or they can be ambiguous, giving a guideline on what to do.

Manual testing recognizes three fundamental levels of testing: ad hoc testing (no content or script required), ambiguous scripts and defined scripts. Ad hoc testing is usually managed without any plan or guide. Rather, the tester comes up with diverse scenarios and inputs to check that the code works as per the requirements. This sort of testing is effective because it makes use of the imagination and creativity of the tester and hence is best suited for testing interfaces which change often, for example modern web testing.

There are relatively less instructions present in vague scripts regarding how to do a step, e.g. "try incorrect inputs", yet the inputs are not provided and need to be made up by the tester. The detailed ones offer more instructions: what parameters to test, the inputs that to be tested and the expected outputs. Detailed scripts are monotonous for testers, since there is no room for creativity. Having said that, it is also true that the detailed scripts are the simplest ones to begin with since all parameters of the tests are provided.

Manual testing is often criticised for being slow, ad hoc and monotonous. Hence, it is used to along with automation testing so those, automation can do most of the repeatable works. However, manual tests are better to discover new defects because bugs are mostly found when the test runs for the first time.

1.4 Test Automation

In literature there exists countless definitions to test automation, but perhaps the most fitting one is by Koirala and Sheikh (Koirala and Sheikh, 2009):

"Automation is the integration of testing tools into the test environment in such a manner that the test execution, logging, and comparison of results are done with little human intervention."

This definition shows that automation testing isn't only about automating test scripts, but also automating the support activities as much as possible. Apart from execution, different things that can be automated are test data generation i.e. creating combinations of inputs, populating databases with test data, system configuration (preserve or recreate environments), simulation (mocking system features that are not yet available, mocking network, database access etc.), results analysis, recording activities and coverage and communicating test results

1.5 Implementing Automated Testing

Dustin et. al. (Dustin et al., 2009) present six keys for successful automation:

- Know your requirements
- Develop an automation strategy
- Verify the automation framework
- Track progress and address accordingly
- Implement automation processes
- Appropriate usage of skills

The most important thing is to be aware of the client requirements of the system which is to be tested. The pre-requisites of the framework under tests fill in as the baseline for the entire process of automation, and so it is one of the vital variables that will affect the success of the project. Therefore, all the requirements must be recorded in a concise manner so that they are precise.

The following step is to build an automation strategy. This strategy can be viewed as a blueprint, which would define the scope, goals, approach, structure, tools,

conditions, and schedule and tester requirements for the automation. It also includes information on the maintenance of the tests, and estimates of expenses and advantages of the project. Actually, the strategy should begin with smaller steps, instead of automating everything together. In case the strategy is not definite before execution, the outcome can be disastrous.

Third point is the verification of the automation structure. The framework that is selected for automation must be verified so that is works correctly and takes into account modifications. Tool selection should be done carefully and trial versions used before making the decision to buy lists guidelines in tool selection: Does the tool have all features needed? Is it reliable? Does it work beyond examples? Is it easy to learn and operate? Is it powerful enough? Does it simulate actual users well?

If the testing process is observed efficiently, modifications, such as fixing defects, and lessening goals, can be made to avoid problems later on. Lessons learned must be gathered, for instance by using root cause analysis and eliminate it.

The fifth point is to administer an automation procedure. The process itself should be lightweight, well defined and structured but contain as little overhead as possible. The whole automation process should be treated as any software development effort, which means it should include defining what to automate, design and execution. The process should also be very flexible to allow for easy feedback loops and convenient to review. Also, a clear distinction between the automation process and the process it automates should be made

Ultimately, proper utilization of skills implies that individuals are at the heart of successful automation execution, and that they must be appointed as per their capabilities. It is additionally proposed that the project should have a senior expert, who has skills in project management and undertaking, and would be responsible for managing the project and to facilitate communication between the testers and the developers.

1.6 Benefits of test automation

1.6.1 Time savings and quality of testing

The benefits of automation testing are related to the speed of executing the tests. It takes much less time to execute a test automatically as compared to manually. Thus, more test cycles can be executed that too during any time of the day. Automation executing of tests do not require any human interference. Tests can also be executed in parallel manner with various computers, so as to enable the execution of many tests at the same time.

Manual testing is not sufficient unless an organization continually builds resources in cycle timeframe. It is so because when applications change and becomes more complex, the number of tests to keep up the satisfactory test coverage additionally develops constantly. It is additionally important that even a little increment in code, suppose 5%, still requires that all of the features must be tested i.e. *regression*. This risk is shown in figure 1.7.

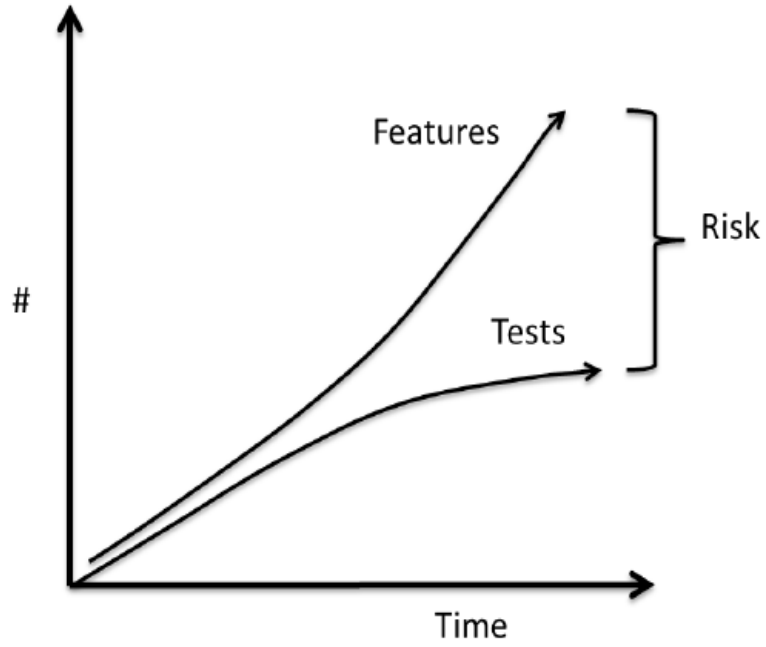

Figure 1.7. Defect risks caused due to inappropriate software testing

Regression testing is significant because when a bug is detected and fixed, the fix might not result to the wanted scenario.

A specific fix may, indeed,

a) Fix the defect reported
b) Fail to fix the defect reported
c) Fix the defect but break a functionality that was working in the previous release
d) Fail to fix the and break a previously working functionality

Regression testing is to check that the new software doesn't affect the old functionality. It is thus more of a "quality control" than a "front-line testing method". Automation must be applied to tests cycles that have less changes per cycle, since they are tedious to prepare. Most of the defects detected during automated tests occur during the development phase. All the automated tests are executed whenever a new change is made in the software, to ensure quality. It is so because the functionality of automated tests sometimes breaks during software development.

1.6.2 Effects on rework and software quality

Figure 1.8 shows the relative size and division of aggregate expenses of an improvement of a software product. General accessibility is the point where software is released to the clients and construction time costs of a product. Fixed & overhead expenses are in each project and they incorporate things like power and office space costs. Construction costs are related to the actual development of the program software such as requirements analysis, design and coding.

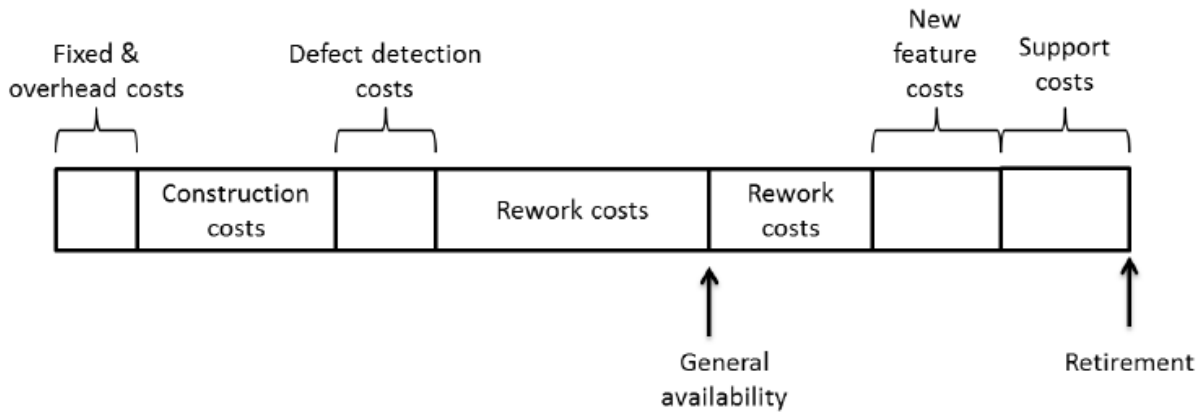

Figure 1.8. Division of total costs under different segments

Defect detection is for finding defects i.e. root cause analysis, testing and inspections / peer reviews. Rework costs, i.e. costs related to defect fixing, are divided to pre- and post-release time. Pre-release rework costs are defects fixes of the defects found in construction phase, whereas post-release rework costs are from fixing defects found by the customers. Post-release costs also include new feature costs and support function costs.

As the figure shows, rework costs are typically large in a software development project and thus a potential aspect in cost savings and acceleration of the development process. Avoidable rework is typically a large part of a development project, i.e. 20-80%. Different studies on automating of unit tests and test-driven development (TDD) at Microsoft and found that the relative amount of defects found by developers, manual testers and customers decreased significantly.

1.7 Costs, pitfalls and other issues regarding test automation

Test automation is incorporated with other changes and techniques in the software development process. One of the common design techniques associated with test automation is “test-driven development (TDD)”, where developers write the test scripts before production code. Another common development process, mostly associated with TDD, is the integration of agile development methods.

Continuous integration (CI) is associated with automation testing. It is “a software development practice where members of a team integrate their work frequently”. It is quite mainstream with agile development methods, and integration may happen multiple times, with the expectation to decrease time spent on defect identification by discovering issues early.

It is recommended that automated test environment is important in agile software development, since developers “need to be able to get real-time feedback for each single code change immediately”. It is also claimed that automation of test cases “is a key factor to success with improved productivity in a software development project”, because the execution of manual test cases multiple times is too expensive as it is a labor-intensive activity.

The usage of new processes and tools in an association causes up-front costs such as device acquisitions and training, which are typically one-time ventures and don’t really pay off directly during the first project. Changed method for work, might change the expenses in the long run. Those expenses can be difficult to measure if the new procedures. Maintenance costs of automation testing are additionally something that turns out to be easily uncontrollable.

Automation testing requires high product testability which is done by making the inner state of the software discernible and the software under test controllable, which puts forth the test case development and localization of defects less demanding.

2 DECISION MAKING BETWEEN AUTOMATED AND MANUAL TESTING

It is not feasible to automate all the tests, so decisions must be made regarding automating specific tests. This segment discusses this topic both in a general level and with two models to help decision making between automation or manual execution of tests.

2.1 General

Automation testing has the potential to decrease time and expenses particularly if the software is developed with highly iterative procedures. It pays off if the expenses of execution manual tests are more than automation, and as automated tests frequently require high initial effort, they must be executed a multiple times to “break-even”.

The choice between automation and manual testing depends upon the nature of the tests and how regularly they are run. Automation testing is better to address regression risk, i.e. that a previously working functionality doesn’t work after new commitments to the code. Manual testing, in contrast, is suitable to explore new ways to break functionality.

Manual testing is better at adjusting to changes, as testing should be ought to be an interactive procedure rather than a sequence of actions, and with automation each assessment must be particularly arranged.

2.2 Models to support decision making

There are some models as well to help in decision making between automation and manual testing that are based on time required by both activities and the different aspects of testing. A simplistic, “universal formula” for test automation costs, is widely cited in software testing literature. It is originally published by Linz and Daigl (1998) and it defines the following variables:

S = Expenditure for test specification and implementation
T= Expenditure for single test execution

Therefore, the cost for a single automated test (X (a)) is calculated in equation 2 as follows:

$$X(a) = S(a) + n * T(a) \quad (2)$$

Where S(a) is the cost of automating the test case, T(a) is the cost of executing the test case one time, and n is the number of automated test executions. The cost of a single manual test case would therefore be:

$$X(m) = S(m) + n * T(m) \quad (3)$$

Where S(m) is the cost of specifying the test case, T(m) is the of executing the test case and n is the number of manual test executions. In order to find the break-even point from automation testing by this model, we have to compare the cost of automation testing to that of manual testing as shown in equation 4. Equation 5 demonstrates the number of test executions (n) to make automation beneficial, as calculated from equation 4.

$$E(n) = \frac{X(a)}{X(m)} = \frac{S(a)+n*T(a)}{S(m)+n*T(m)} \quad (4)$$

$$n = \frac{S(a)-S(m)}{T(m)-T(a)} \quad (5)$$

Suppose a single test specification takes 15 minutes with manual testing (S(m)) and 60 minutes with automation (S(a)), and test execution takes 5 minutes with automation (T(a)) and 20 minutes with manual testing (T(m)), the break-even point would be 3 test executions. Automation would be worthwhile if there were 3 or more test executions. Figure 2.1 illustrates the break-even point in automation testing.

$$n = \frac{60-15}{20-5} = 3 \quad (6)$$

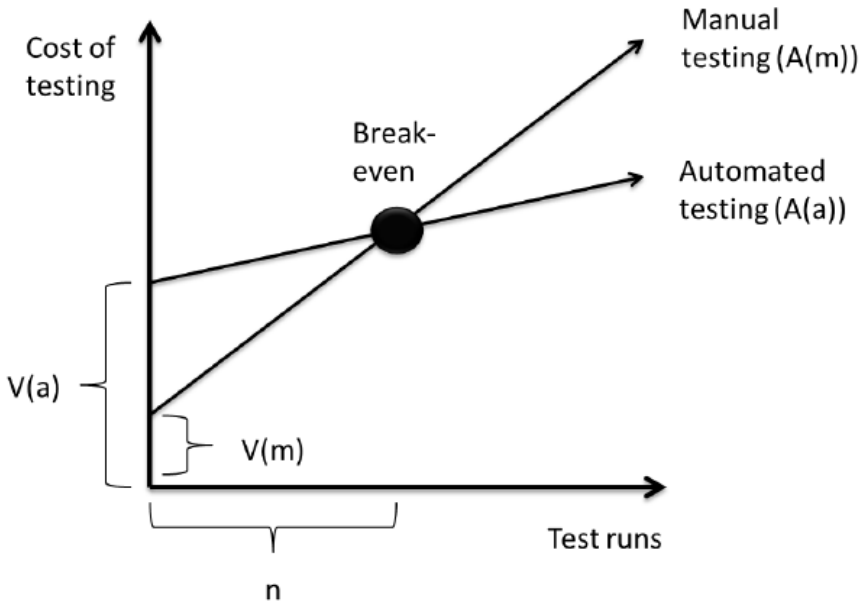

Figure 2.1. Automation testing break-even point

There is an alternate model to help decision making in automating tests based on the model described above. Firstly, a budget needs to be established, which is usually much less than what is typically budgeted to test the software. Under this financial plan, all the tests must be executed. The time used for testing activities is simplified so that manual test takes same equal time. On the other hand, for automation tests, the testing time is equal to the time it takes to make the actual test, while the execution time is not being taken into account because it takes little time to execute it.

All of the possible combinations of testing fall thus under the following equation:

$$n(a) * S(a) + n(m) * T(m) \leq B \quad (7)$$

Where n(a) = number of automated test cases

n(m) = number of manual case executions

S(a) = expenditure for test automation

T(m) = expenditure for a manual test execution

B = fixed budget

2.3 Discussion in regards to cost-benefit analysis

The characteristics of manual and automation testing are different and the time consumed differs as well. The making of a single automation test takes a lot of time and the execution takes less time whereas manual testing's time consumption lies often in the execution portion itself. Based on literature review above it seems that the tests that are repeated often should be automated and the ones that aren't should be done with manual testing.

Automation testing is seen as a necessity in agile development methods, to provide fast feedback to the developers. This can only be achieved through automation.

Advantages	**Measure**
Speed of testing	Saving time by replacing manual with automation
Productivity of software development	Reduction of rework
Quality of the software products	Reduction of costs related to fixing post-release defects

Table 1. Benefits of test automation and possible measures of them.

3 THE BASIS FOR A COST-BENEFIT ANALYSIS

3.1 Definition of investment

An investment can be seen as "*an asset or item that is purchased with the hope that it will generate income or appreciate in the future*" (Investopedia, 2013).

Cost-benefit approach is used to make decisions regarding resource allocations. For example decisions regarding hiring of new employees or purchasing new software. The approach is that the expected benefits of the resource acquired must exceed the costs.

Discounted cash flow methods are generally used to decide, whether an investment is worth it. It is the basis for calculating different profitability measures. It measures estimated cash inflows and outflows caused by an investment. Discounted cash flow methods consider time value of money, which means that money is worth more today than any point in the future. That is because money can be invested at an interest rate so it grows by the end of the year.

3.2 Methods of investment appraisal

3.2.1 Net present value (NPV)

Net present value (NPV) ascertains the majority of the expected future cash flows by discounting them to the present value with a particular rate of return. As per NPV, only the ventures that have a positive value are acceptable because the return of the investments exceed the cost of capital. There are three steps of using the NPV method: 1) drawing a sketch of cash inflows and outflows, 2) computing the correct discount factors and 3) totaling the present value figures to calculate the NPV of the investment.

This approach is set out on a yearly basis. The initial investment is typically set to take place at the year 0, which is generally the start of the year 1. The initial year is when majority of the capital consumption happens. However, for complex projects where the initial cost of investment is spread amongst many years, there are two options:

- The end of the year from when the main capital outflow occurred
- The last when the project was completed

The below figures demonstrate the typical patterns of cash flow of a capital expenditure. Figure 3.1 is about a one-time initial investment at year 0 that generates cash inflows in the following years. Figure 3.2 is about the initial investment that is divided between multiple years. It also illustrates the possibility of an investment to have negative cash flows in the future too, perhaps in the form of additional investments in the project. Investment on automation testing can easily be viewed as a complex venture as illustrated in figure 3.1. Its “initial investment” is in reality difficult to determine and the yearly expenses are much more significant.

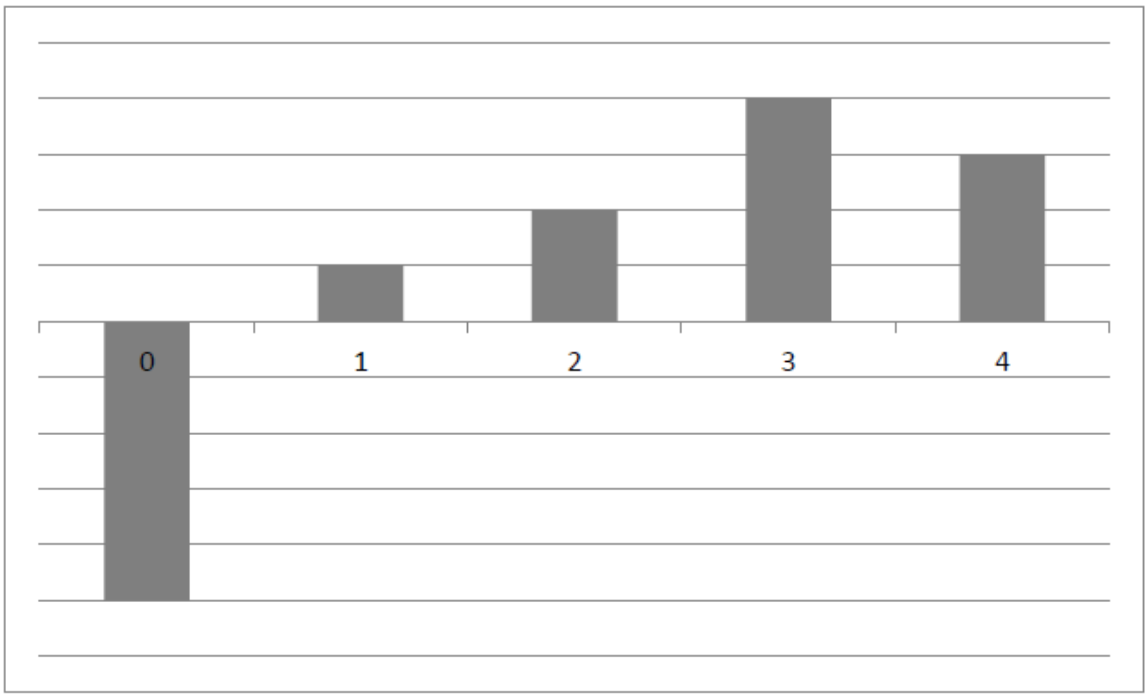

Figure 3.1. A typical pattern of cash flow

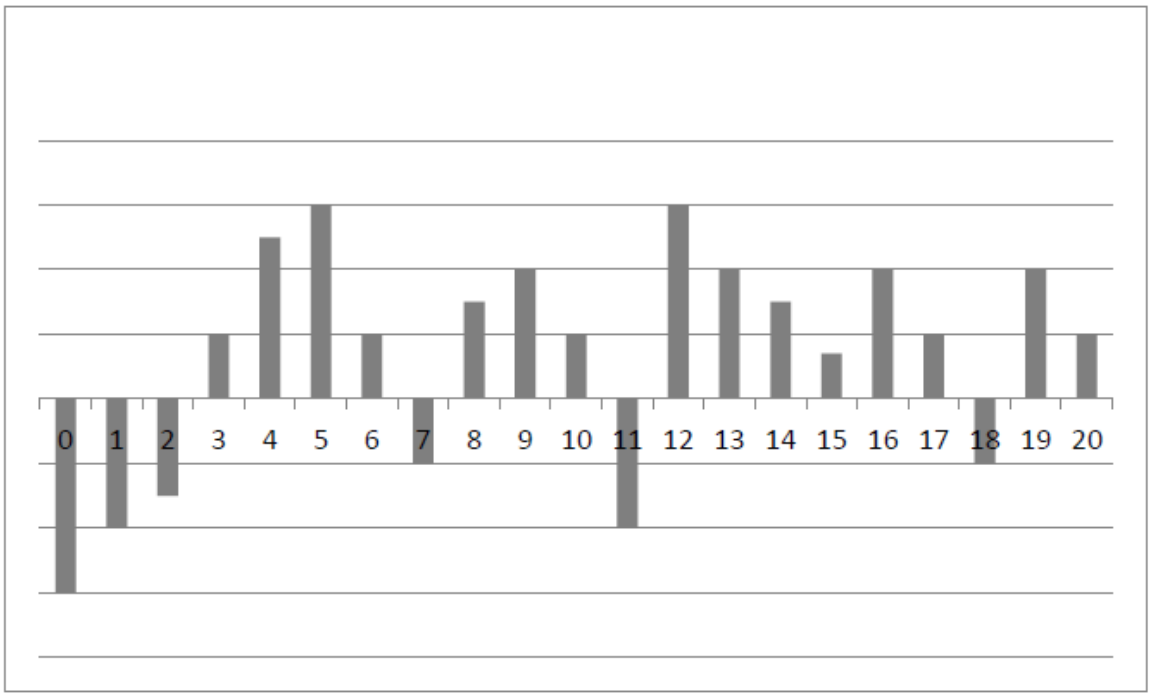

Figure 3.2. Cash flow pattern of a complex project

As mentioned earlier, the future value of 1 € is at a given interest rate bigger in the future than today and 1 € received at any point in the future is of less value than 1 € today. In fact, the interest factor of 1 € is

$$(1 + i)\text{^}n$$

Where i = the rate of interest as a decimal value and
n = the number of years.

By the same logic, the present value and discount factor of 1 € from any point in the future is

$$(1 + i)\text{^} - n$$

Where i = the rate of interest as a decimal value and
n = the number of years.

Figure 3.3 demonstrates the approach to compute a simple project's NPV, where there is a single initial investment and a yearly cash inflow of the same amount of money. The time value of money is represented as the present value of 100 000 €

drops yearly, and the NPV is 57 700 € instead of 100 000 € (-200 000 € + 3 * 100 000 €) that would be a simple aggregate of the yearly cash flows.

		Net initial investment	200 000 €			
		Useful life	3 years			
		Annual cash inflow	100 000 €			
		Required rate of return	8 %			
	Present value of cash flow	Present value of 1 € discounted at 8 %	Relevant cash flow at end of each year			
			0	1	2	3
Net initial investment	-200 000 €	1,000	-200 000 €			
	92 600 €	0,926		100 000 €		
Annual cash inflow	85 700 €	0,857			100 000 €	
	79 400 €	0,794				100 000 €
NPV if new machine purchased	**57 700 €**					

Figure 3.3. NPV calculations

3.2.2 Benefit-cost ratio (BCR)

The impacts of an investment project can either be positive or negative. Cost-benefit analysis' role is to compute the difference between the two. A vital aspect of cost-benefit analysis is opportunity cost, which represents the cost of not executing a competing project and decision criteria of investment in cost-benefit analysis is shown in figure 3.4.

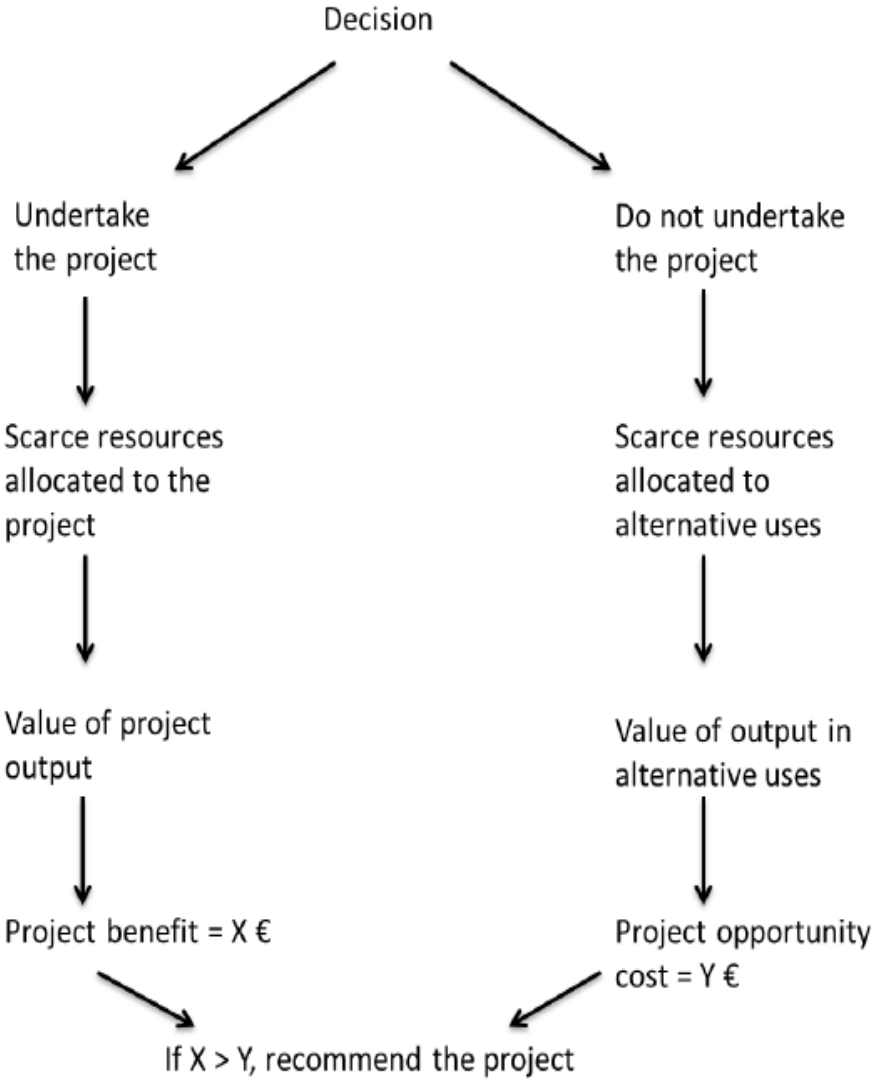

Figure 3.4 Measure of resources allocated for cost-benefit analysis

The common measure of cost-benefit analysis is benefit-cost ratio (BCR). It is basically a variation of NPV, but instead of subtracting the present value of project's benefits from the present value of costs, the benefits are used as numerator and the costs are used as denominator. Equation (8) indicates how BCR is calculated

$$BCR = \frac{PV(Benefits)}{PV(Costs)} \quad (8)$$

3.2.3 Payback method and internal rate of return (IRR)

Payback of an investment “is achieved when cumulative net cash flow turns from negative to a positive figure.” Payback is thus estimated in years and months when the first investment is recovered by the annual estimated cash flows. Payback is mostly computed from nominal cash flows without an interest rate / time value of money, but it can also be computed from discounted cash flows.

A hindrance of payback method is that it doesn’t calculate the gain over the entire life of the investment; it just shows when the investment has paid itself back. While comparing two different investments, this might deliver a lesser payback time for an investment that has a shorter lifecycle; an investment with an extended lifecycle might have a longer payback time but greater profits over the entire lifecycle. This inconvenience can be overcome by utilizing payback method in along with internal rate of return (IRR).

IRR is “often described as the rate of interest that results in a net present value of zero.” IRR method only accepts the projects where IRR equals or exceeds the required rate of return (RRR), which is the minimum acceptable rate of return of an investment. RRR is determined by the company itself, and it is the return that the company could expect to receive elsewhere if it invested the money elsewhere. If IRR equals or exceeds the RRR, it means that the estimated future cash flows are both adequate to recover the initial investment and earn a return of exactly the value of IRR during the whole lifecycle of the investment.

3.2.4 Return on investment (ROI)

Return on Investment (ROI) is a well-known method for estimating the execution of an investment. It is mainstream since it blends everything of profitability into a single rate and it can be compared with the rate of return of different investments, both inside and outside of an organization. Return on Investment additionally shows how much profit each monetary unit spent on an investment produces.

ROI is computed by dividing the net profit of a regular year by the average investment.

$$ROI = \frac{Net\ Profit}{Average\ Assets} \quad (9)$$

However, the equation to calculate the return differs. The basic method of computing ROI in these instances is a variation of NPV and BCR that is a simple correlation of net benefits and costs.

ROI is defined as:

$$ROI = \frac{Benefits - Costs}{Costs} \quad (10)$$

If ROI is computed as in equation 10, Net Present Value can be utilized as a basis for ROI computation. Then the NPV value calculated is used as numerator and PV of costs is used as denominator of the equation above. Therefore, if the example calculation of chapter 3.2.1 is used as an example, the "ROI" of that investment would be:

$$ROI = \frac{(92600€ + 85700€ + 79400€) - 200000€}{200000€} = 28.85\% \quad (11)$$

4 RETURN ON INVESTMENT OF AUTOMATION TESTING

ROI is a common method of measuring cost-benefit analysis. When measuring the efficiency of test automation, there exists the problem of actually assessing intangible benefits. In a management perspective, when the intangible are difficult to assess, the tangible must then take priority. Here RoI calculations become important, since they offer an easy and effective way of identifying and explaining the effectiveness of test automation. Additionally, it is useful for upper management to have financial information about test automation effectiveness in order to make better decisions whether or not to introduce test automation in the first place. Data that can be used to measure the Return on Investment of automation testing are tangible costs and intangible costs.

4.1 Tangible Factors

Financial costs are generally separated into fixed and variable expenses. Fixed costs incorporate things such as consumptions for hardware, devices etc., to be specific, items that are not influenced by the amount of test runs or tests created. Variable expenses are factors that are impacted by the amount of test cycles or by the number of tests created.

Factor	Definition
Cost of hardware	Additional hardware must sometimes be purchased for test automation, since it might need more power for handling data and large amounts of test runs.
Licences for tools and testware	Automated testing usually requires additional software and frameworks for running tests. These might not always be free.
Tool training	The tools used by automation are not often used in manual testing, and thus may require training.
Test automation environment implementation and maintenance	The manual testing environment usually cannot be used for automated testing, or requires extensions and maintenance.

Table 2 Fixed Costs for Test Automation

Hardware costs are one of the most important fixed costs for automation. Automation testing is mostly suitable for testing that requires reiteration, or dealing

with big amounts of data. Moreover, projects typically have equipments readily present, for instance servers for deployment.

Another, potentially real cost of automation testing are the tools of testing, frameworks. Tools like Selenium are available free of cost. Some test wares are proprietary and hence approved licences need to obtained. Training of testers is another factor of increasing costs. Ordinarily this training can take up to 6 months.

Finally, the execution and upkeep of the test environment can end up being difficult. Manual testing depends on working with the development environment, but automated tests might require separate special environments to operate. Modern automation is fairly editable and can be easily integrated with the development and testing environments.

The biggest variable expense in automation testing is the development of the test cases. In manual testing the test cases are created once, and then repeated without many changes. In automation the test case is first designed and then executed. The development of the tests cases in automation is much more arduous. Furthermore, the automated test cases are mostly derived from manual test cases, which means that the expenses of developing manual tests cases are included in automated test scenarios.

Related to test creation, the maintenance of the automated tests is usually a big expense. With manual tests, the tests are rarely changed, and changing them requires only minor redesign work. Automated tests are carefully scripted and if the implementation of the program changes, the tests need to be adjusted accordingly, unless made robust or to test high level components (i.e. unit tests automation vs. component level automation).

Although it seems that automated tests mostly raise investment costs for the project. The real benefit is during test execution, because executing automated tests is far quicker than manual tests. The benefit is because more tests can be run more often.

Factor	Definition
Test case implementation	Creating automated test cases can be compared to software development in means of effort, which is a lot more than in manual testing.
Test case maintenance	Automated test cases require more maintenance, since often when code is changed the tests need to be modified as well.
Test Execution	Test execution means executing the automated test scripts.
Test results analysis	Automated tests can generate significant amounts of test logs, which explain the reasons the tests succeeded or failed.
Personnel considerations	Using automation might require different skill sets, or a different amount of test engineers.

Table 3 Variable Costs for Test Automation

Tests results analysis can take a long time in testing when done manually, i.e. comparing test output to predefined result criteria. Automated tests can also create a large amount of output, but it can also be used to parse and compile test results into a more readable form. Expected test results can be coded into the test script, compared against automatically and the output of the test can indicate, for example by colour coding, the status of the test.

4.2 Intangible Factors

Intangible factors are the ones that cannot be directly estimated in monetary terms. Even then, the variables may add to a major part of the ROI. There are many intangible advantages of automation testing. Firstly, with the help of automation those tests that cannot not be run manually are executed. Automation empowers testing monotonous scripts that might affect worker's motivation and result in decreasing work performance.

Automated scripts run consistently thereby reducing the error of human testers. The tests are run in the exact same way in different environments. "Automation is considered to be software development in its own right". This implies that it is more challenging and rewarding than manual testing.

5 ROI CASE DESCRIPTIONS

The case study is based on an automation venture of a large pharmaceutical sector company known as Cerner. For over 30 years, Cerner has been working on its mission to make health care safer and more effective. Headquartered in Dublin, Ireland, the organization provides systems for consumers, specialist doctors, hospitals, and countries. Cerner solutions support clinical decisions, prevent medical blunders, and empower patients in their care. They are used in roughly 10,000 facilities around the globe including more than 2,700 hospitals.

Cerner's health care team needed to streamline and automate the coordination of information from different applications into one basic electronic medical record application for its clients.

A project of automating the tests was suggested to implement an automation infrastructure, which would include a continuous integration server and a reporting view, so as to reduce the amount of manual testing labour. The target was to decrease around a week's worth of manual testing per cycle by the end of the year 2018.

As discussed earlier, return on investment comprises of investigating the expenses the expenses and benefits of the investment. The case presented in this paper has been under way for 17 months (at the time of composing this thesis), which is 2 full cycles and one partial iteration. Each iteration lasts for 6 months, containing a 9 week testing period. Since the third iteration was almost complete (one out of six months left) at the time of writing this thesis, the estimated numbers for third iteration testing period are used, which at this point are fairly accurate since the iteration is nearly complete.

5.1 Project Data Collection and Analysis

The data gathered from the project was depended on the documents of the implementing organization and the notes of the developers who executed the project. All financial data has been changed in order to protect the privacy and business of the company. Most of the other relevant data's have been collected from the company website, balance sheet, annual report, etc

The case study data analysis was carried out by posting all variables like the test implementing costs, test execution costs, design including the project expenses and advantages. Then analysis was done with the theory provided in the literature review. After the variables, had been identified and measured, calculations based on the literature review were made.

5.2 Automation costs

Starting with the expenses of the automation venture, the aggregate costs are displayed in the same form as in tables 3 and 4. All the cost factors are clarified with detailed computations below, and then multiplied by the amount of cycles (3) to be displayed in table 4.1. For the benefit of this case, test implementation and design and execution and analysis were consolidated to one cost factor. The combinations were made since for manual testing, the total cost of design and implementation and of execution and analysis were known, but not separately. Thus, they were combined in both manual and automated costs to give a better comparison of the two.

Factor	Amount	% Of Total
Cost of hardware	8000€	3.2
Licences for tools and testware	Negligible	-
Tool training	Included in other activities	-
Test automation environment set-up and maintenance	3000	1.2
Test case implementation & design	160380€	66
Test case maintenance	18000€	7.4
Test Execution & analysis	53820€	22
Personnel considerations	0€	0
Total	**243200€**	

Table 3 Total Costs of automation case

Hardware costs consist of a single test server. The cost of the hardware was estimated at 8000€ by the IT specialist of the company. Licences for tools and test ware were considered irrelevant. All other tools utilized were open source systems with no licence fees.

Tool training was not considered a different expense item because most automation developers were already used to the technologies, and those who were not were given organized training.

Test environment set-up was computed to be a two-day exercise, costing 1200€. This comprised of installing software, tools and configuring the system as per the needs of the project. Test environment maintenance was a 2 day job in each cycle, which costed around 1800€.

Test case calculation implementation was separated into two fields: during test cycle, and after test cycle. The costs for each cycle of implementation were computed as follows:

Bit = Implementation during test cycle * test cycle weeks *
weekly cost of automation consultants
=0.5 * 9 * (5d * 7.5h/d * 80€/h)
=13500€

Bid = Implementation during development cycle * test cycle weeks *
weekly cost of automation consultants
=0.9 * 14.8 * (5d * 7.5h/d * 80€/h)
=39960€

Bi = Automation implementation total per iteration
= B*it* + *Bid*
=53460€

Using the same logic as in the automation implementation formula, the costs for automated tests execution, maintenance and environment maintenance were computed for each cycle.

Bet = Execution during test cycle * test cycle weeks *
weekly cost of automation consultants
= 0.5 * 9 * (5d * 7.5h/d * 80€/h)
= 13500€

Bed = Execution during development cycle * test cycle weeks *
weekly cost of automation consultants
=0.1 * 14.8 * (5d * 7.5h/d * 80€/h)
=4440€

Be = Automation execution total per iteration
= B*et*+*Bed*
= 17940€

Bmt = Test maintenance/iteration * cost of automation consultants
= 2w * (5d * 7.5h/d * 80€/h)
= 6000€

Bme = Env maintenance/iteration * weekly cost of automation consultants

= 1d * (7.5h/d * 80€/h)

= 600€

In spite of the fact that the iterations may differ in workloads, the same estimations of work allocation, as reported by the consultants who implemented the automation project, were mostly accurate.

5.3 Manual Testing Costs

To make an even comparison with automated test costs, the same variables for manual testing will be calculated and explained. Starting with a similar table as table 3, the costs of manual testing during the automation project are separated in table 4. The cost of hardware licences, training, and environments in the manual testing project are all either zero or negligible compared to the size of the testing process. The numbers are negligible since most of the hardware, for example, was already there for the project.

Factor	Amount	% Of Total
Cost of hardware	0€	0
Licences for tools and testware	0€	0
Tool training	0€	0
Test automation environment implementation and maintenance	0€	0
Test case implementation & design	191450€	22
Test case maintenance	Included in implementation and execution	-
Test Execution & analysis	681140€	78
Personnel considerations	0€	0
Total	**872590€**	-

Table 4 Total Costs of manual testing during automation case

Test case implementation and design was led by 12 designers, who worked on making tests for this project on a 20% work designation. The pay checks of test designers were not uncovered, but an industry standard of 4500€ was used for the estimations:

Md = Amount of designers * 26 weeks/iteration * allocation * Salary
= 12 * 26w * 0.2 * (5 / 22 * 4500€)
= 63800€

5.4 Scaling effort and project RoI

Having computed the expenses of both manual and automated testing, there was one more thought to make before estimating the RoI of the venture: amid the project the similar amount of automated test cases were not executed and made as manual tests. Distinctive amounts of manual tests were executed in the cycles, and both manual and automated tests were created amid iterations. The summary of the difference is displayed in table 5:

	Manual	Auto	Ratio
Tests executed in iteration 1	255	26	0.102
Tests executed in iteration 2	385	43	0.112
Tests executed in iteration 3	392	97	0.247
Tests implemented in iteration 1	33	26	0.788
Tests implemented in iteration 2	33	17	0.515
Tests implemented in iteration 3	33	54	1.636

Table 5 Automation and manual tests execution and implementation differences

The purpose of posting these values and proportions is to scale the effort of automation development and execution to the same effort as with manual testing. Moreover, the tests that were executed were fairly similar. The scaling of expenses per cycles are displayed in table 5. The column manual is estimated by taking the

ratio column from table 6 and multiplying the corresponding manual cost by it.

	Manual	Auto	Manual scaled	Automation benefit
Execution cost in iteration 1	227000	17940	23100	5200
Execution cost in iteration 2	227000	17940	25300	7400
Execution cost in iteration 3	227000	17940	56100	38200
Total	**681100**	**53820**	**104600**	**50870**
Implementation cost in iteration 1	63800	53460	50280	−3180
Implementation cost in iteration 2	63800	53460	32800	−20580
Implementation cost in iteration 3	63800	53460	104400	50979
Total	**191450**	**160380**	**187580**	**27200**

Table 6 Automation and manual tests costs and scaling

Having calculated and scaled all the costs of both manual and automated testing, we can apply the formula for RoI by Kelly (2004):

$$RoI = \frac{Benefit}{Investment} = \frac{Cost\ of\ manual\ testing - Cost\ of\ automated\ testing}{Investment}$$

	Manual	Auto
Total cost for iteration 1	73430	87200
Total cost for iteration 2	58230	78000
Total cost for iteration 3	160600	78000
Total cost for project	**292270**	**243200**

Table 7 Automation and manual tests costs and scaling

Table 8 displays the summary of the RoI for all cycles of the project until iteration 3.

Manual	Auto
RoI for iteration 1	−0.158
Roi for iteration 2	−0.253
Roi for iteration 3	1.059
Total Roi for project	0.201

Table 8 Automation and manual tests costs and scaling

6 DISCUSSION

6.1 How can test automation effectiveness be measured?

The literature review conducted in this paper proposes that the most well-known method of estimating automation testing efficiency is to calculate its return on investment. In other words, the benefit is computed from "How much money do we use to do this manually" minus "How much money do we use to do this automatically".

Some portion of the costs comprise of the equipment costs, permit costs, tool training, test case implementation and upkeep and test outcomes analysis. Extended from the main formula of Return on Investment, a variant of the common RoI formula was a part of this thesis, which clarifies the logic of finding the advantage and cost of automation:

$$RoI = \frac{Benefit}{Investment} = \frac{\sum(Cost\ of\ manual\ testing - Cost\ of\ automated\ testing)}{Investment}$$

The value in the numerator means estimating the difference between manual and automated testing for all expense factors. The cost of automation testing comprises of the design, execution, defect reporting, server costs, software permits and training.

6.2 What is the impact of test automation to software project profitability?

In this paper, the effect of automation testing on software business was studied by conducting a case study on the business information gathered from the company website. 10% of the tests were automated, so the expenses of manual testing were downsized to the point where only 10% of testing was manual.

The outcomes from the calculations in the outcomes chapter demonstrates that automation gave a benefit in the testing effort in the case project, and after 18 months of automation were around 0.20. This paper only focused on one large case, and thus the precision of the result is not very high. In any case, the project has been running for 18 months, with dozens of testers involved, which increases the size of the study.

6.3 What are the benefits and pitfalls of automated testing?

Finally, this study attempt combined the benefits and pitfalls of automation testing. Speed, consistency, regression tests and better utilization of resources are the benefits automation testing. Initial effort, upkeep needs, discovering new defects and failure to replace people were all often mentioned as automation pitfalls.

CONCLUSIONS

The research question of this paper was: “how does the implementation of test automation affect software projects?” Judging from the outcomes of the case study, the effect of automation testing appears to have a positive effect on business by giving savings against manual testing. The outcomes appear to demonstrate that the benefits of automation are acknowledged after a long time.

Notwithstanding fiscal advantage, the study uncovers intangible benefits that comprise of tests that could not be run manually, consistency of test cycles and tester job satisfaction through motivation and reward.

This paper only covered a single case organization. While the case was an extensive project, the information gained from it is only from the company website. This leaves space for more exhaustive case research, to incorporate more information. Next, the likelihood of analyzing impacts on business could be examined from another point of view, such as code coverage or number of new defects during development.

REFERENCES

- www.automationanywhere.com
- www.cernertech.com
- www.softwaretestingclass.com
- www.istqb.com
- www.irisa.fr
- www.searchitoperations.com
- www.techtarget.com
- www.doria.fi