SCHOLAR COMMONS
USF UNIVERSITY OF SOUTH FLORIDA

University of South Florida
Scholar Commons

Graduate Theses and Dissertations

Graduate School

2005

Understanding ROI Metrics for Software Test Automation

Naveen Jayachandran
University of South Florida

Follow this and additional works at: https://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation

Jayachandran, Naveen, "Understanding ROI Metrics for Software Test Automation" (2005). *Graduate Theses and Dissertations.*
https://scholarcommons.usf.edu/etd/2938

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.

Understanding ROI Metrics for Software Test Automation

by

Naveen Jayachandran

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Co-Major Professor: Dewey Rundus, Ph.D.
Co-Major Professor: Alan Hevner, Ph.D.
Member: Rafael Perez, Ph.D.

Date of Approval:
June 2, 2005

Keywords: Return on investment metrics, automated testing feasibility, manual vs. automated testing, software quality assurance, regression testing, repetitive testing

© Copyright 2005, Naveen Jayachandran

Dedication

To my Parents, Suja, Rocky and Reemo.

Acknowledgements

I would like to thank my major professors Dr. Alan Hevner and Dr. Dewey Rundus for recognizing and encouraging my interest, giving me this opportunity and providing constant support for my research. I have gained immensely from this experience enabling me to truly appreciate the saying – 'The journey is more important than the destination'. I would like to acknowledge my faculty committee member, Dr. Rafael Perez for his time and valuable comments. This work was supported in part by the National Institute for Systems Test and Productivity (NISTP) at the University of South Florida under the US Space and Naval Warfare Systems Command Contract No. N00039-02-C-3244. I would like to thank Tom Barr, Judy Hyde and the rest of the NISTP team for their useful insights and thoughts. I also acknowledge IBM Rational for providing the Rational Test Suite that has been used extensively in this research study. Finally, I thank my parents and my sister for being my source of motivation and hope.

Table of Contents

List of Tables iv

List of Figures v

Abstract vi

Chapter 1: Introduction 1

1.1 Prelude 1

1.2 The Problem 2

1.3 The Goals 2

1.4 The Questions 3

1.5 The Metrics 3

1.6 The Experiment 4

Chapter 2: Literature Review 5

2.1 Software Testing 5

2.1.1 Manual Testing 5

2.1.2 Automated Testing 6

2.2 Metrics 6

2.2.1 Metrics for Software Testing 6

2.3 Return on Investment (ROI) 7

2.4 Related Work 8

2.4.1 COCOMO II 8

2.4.2 Rational Suite Economic Value Model.. 9

Chapter 3: Experimental Design 10

3.1 The Experiment... 10

3.1.1 Print Dialog Testing (P1) 12

3.1.2 File Dialog Testing (P2)... 12

3.2 The Metrics 13

3.3 Data Collection 14

3.4 Other Details 15

Chapter 4: Results & Discussion 16

4.1 Demographic Data 16

4.2 Schedule.. 18

4.2.1 Regression Testing.. 21

4.2.2 Pure Repetitive Testing... 24

4.3 Cost 31

4.3.1 Fixed Costs.. 31

4.3.2 Variable Costs 32

4.3.3 Regression Testing.. 35

4.3.4 Pure Repetitive Testing... 35

4.3.5 Accounting for Fixed Costs 35

4.4 Effectiveness 37

4.4.1 Lessons Learned.. 41

4.5 Other Metrics 41

4.5.1 Scope of Testing 42
4.5.2 Depth of Testing 43
4.5.3 Test Tool Adoption 45
4.5.4 User Ratings for Effectiveness & Productivity 48
4.5.5 Documentation and Traceability 49
4.5.6 Automation – Individual Vs. Group Effort 50
4.5.7 Test Coverage 50
4.6 Tripod SDLC Model 51
4.6.1 Automation Requirements 52
4.6.2 Automation Feasibility 53
4.6.3 Framework Design 53
4.7 Contributions 53
4.8 Limitations 54
4.8.1 Project Scope 54
4.8.2 Error Seeding 55
4.8.3 Automation Tool 55
Chapter 5: Conclusion 56
5.1 Future Research 57
References 59
Appendices 62

List of Tables

Table 1: Experimental Setup (G – Group, P – Project, M – Manual, A – Automated).... 12

Table 2: Project Timeline... 14

Table 3: 1st Cycle Time Periods – Manual Vs. Automated 21

Table 4: Regression Testing: 2nd Cycle Time Periods – Manual Vs. Automated............. 22

Table 5: Regression Testing: 3rd Cycle Time Periods – Manual Vs. Automated............. 23

Table 6: Regression Testing: Automated Testing Cycle Schedule Overflows................ 24

Table 7: Pure Repetitive Testing: 2nd Cycle Time Periods – Manual Vs. Automated...... 25

Table 8: Pure Repetitive Testing: 3rd Cycle Time Periods – Manual Vs. Automated...... 26

Table 9: Pure Repetitive Testing: Automated Testing Cycle Schedule Overflows.......... 27

Table 10: Regression Testing: Testing Cycle Variable Cost Overflows.......................... 35

Table 11: Pure Repetitive Testing: Testing Cycle Variable Cost Overflows................... 35

Table 12: Actual Defects & False Positives – Manual Vs. Automated............................ 38

Table 13: ROI Recommendations... 56

List of Figures

Figure 1: Experimental Setup 11

Figure 2: Subject's Programming and Testing Experience 17

Figure 3: 1st Cycle Time lines – Manual Vs. Automated 21

Figure 4: Regression Testing: 2nd Cycle Time lines – Manual Vs. Automated 22

Figure 5: Regression Testing: 3rd Cycle Time lines – Manual Vs. Automated 23

Figure 6: Pure Repetitive Testing: 2nd Cycle Time Lines – Manual Vs. Automated 25

Figure 7: Pure Repetitive Testing: 3rd Cycle Time Lines – Manual Vs. Automated 26

Figure 8: Timeline Comparison – Manual Vs. Regression Vs. Pure Repetitive 28

Figure 9: First Cycle Variable Cost – Manual Vs. Automated 33

Figure 10: Actual Defects & False Positives – Manual Vs. Automated 37

Figure 11: Effect of Programming Rating 45

Figure 12: Scripting Experience 46

Figure 13: Training Effectiveness Rating 47

Figure 14: Test Tool Usability Rating 47

Figure 15: Effectiveness & Productivity Rating 48

Figure 16: Documentation Effort 49

Figure 17: Tripod SDLC Model 52

Understanding ROI metrics for Software Test Automation

Naveen Jayachandran

ABSTRACT

Software test automation is widely accepted as an efficient software testing technique. However, automation has failed to deliver the expected productivity more often than not. The goal of this research was to find out the reason for these failures by collecting and understanding the metrics that affect software test automation and provide recommendations on how to successfully adopt automation with a positive return on investment (ROI). The metrics of concern were schedule, cost and effectiveness. The research employed an experimental study where subjects worked on individual manual and automated testing projects. The data collected were cross verified and supplemented with additional data from a feedback survey at the end of the experiment. The results of this study suggest that automation involves a heavy initial investment in terms of schedule and cost, which needs to be amortized over subsequent test cycles or even subsequent test projects. A positive ROI takes time and any decision to automate should take into consideration the profit margin per cycle and the number of cycles required to break even. In this regard, automation has been found to be effective for testing that is highly repetitive in nature like, smoke testing, regression testing, load testing and stress testing.

Chapter 1

Introduction

This chapter introduces the current software testing scenario and prepares the foundation for the rest of the study. The layout of this chapter adopts the goal-question-metric (GQM) paradigm [1] and the sections are included accordingly.

1.1 Prelude

Software defects cost the U.S. economy an estimated $59.5 billion annually or about 0.6 percent of the gross domestic product, according to a study commissioned by the National Institute of Standards and Technology [2]. This cost, unless checked, is bound to grow, since virtually every business today depends on software. The study also states that more than half of this cost is borne by software users and the remaining by software developers and vendors.

The major reason attributed to this cost is an inadequate software testing infrastructure [2]. Recognizing the need, companies worldwide are taking steps to address this issue. This has resulted in a $13 billion software testing market today, according to the Gartner Group [3]. Several software testing techniques are in use and one of the most popular techniques is automated testing, primarily due to its ability to execute test cases faster and without human intervention.

1.2 The Problem

Software testing may be performed either by manual or automated methods. The key is in adopting the right technology at the right time under the right circumstances. Adopting inappropriate testing techniques may prove expensive in the long run. With software systems growing larger and more complex, testing them becomes increasingly time consuming and difficult. To address this problem, more and more software today uses automated testing to check for defects. However, automated testing, or test automation as it is also known, does not always help. Certain tests are better when performed manually. In many cases, manual testing co-exists with test automation and the ratio of contribution depends on the testing requirements.

Since test automation is at least initially more costly than manual testing, it should be thoroughly studied before being adopted. The returns on investing in it have to be calculated. Not many Return On Investment (ROI) models for test automation exist and those that do, are not vendor independent or backed by empirical evidence. The decision to acquire a test automation tool can be made only if reliable ROI metrics for test automation exists.

1.3 The Goals

The goal of this study was to determine what expected returns might be obtained when investing in automated testing, and to develop metrics that highlight the various factors to be considered while making the decision to automate testing.

1.4 The Questions

In order to assess expected returns for a testing project the following questions must be asked:

- What testing technique should be adopted? Manual or Automated?
- What are the testing costs associated with this project?
- How long will the testing project take to complete?
- How effective is the testing project going to be?

1.5 The Metrics

The answers to the above questions may be obtained by identifying, collecting and analyzing the required metrics. Metrics provide an indication of how well a particular project is performing. Metrics can also help identify which processes need to be improved. The key to the success of a project is in the effectiveness of its metrics [4]. Predictive metrics are of particular value since they may be useful in determining the feasibility of a project before its implementation. This research study identifies predictive return on investment (ROI) metrics for software test automation.

During the proposal stages of this study, several metrics were identified:

- Cost
- Schedule
- Effectiveness

These metrics are widely accepted in the software industry as factors affecting the ROI on software related projects. The challenge lies in determining how each of the above metrics actually contributes to software test automation ROI.

1.6 The Experiment

This study analyzes the above metrics and the roles they play. It employs a software test automation experiment to help. Twenty four graduate students worked independently on two testing projects, using manual testing methods for one and automated testing for the other. The experiment was held to a strict timeline and monitored. Sufficient measures were taken to eliminate any bias in the results. These data and the results from a survey at the end of the experiment allowed comparison of both testing methodologies and an analysis of how various factors affect the adoption of software test automation. The details of this experiment will be discussed in chapter 3.

Chapter 2

Literature Review

This chapter discusses the background material required to understand this study. It discusses software testing in general, software metrics and work similar to this research that has been done previously.

2.1 Software Testing

Software testing is a process used to identify the defects in software and to determine whether it conforms to requirements. Although it cannot ensure the absence of defects, testing can at least result in the reduction of defects. Software testing may either be performed manually or by using automated methods.

2.1.1 Manual Testing

Manual testing is a way of testing software where a person creates and executes test cases. The test execution results are compared with the expected output and any defects detected are logged. This whole process is performed manually. Several manual testing methods exist today, ranging from simple black box testing to more complex structural (white box) testing. Some popular methods include Exploratory testing, Stress testing, Load testing, Boundary value testing, Equivalence class testing and Risk based testing. Details on these methods can be found in [5].

2.1.2 Automated Testing

Automated testing uses software to execute the test cases, compare results and log defects automatically without human intervention. Automated testing has gained a lot of attention in recent times as a means to reduce testing costs, find more defects, and save valuable time. In simpler words it is 'writing code to test code'. Most test automation is performed using specialized testing tools and it requires more specialized skills than those needed for manual testing.

2.2 Metrics

A metric is a standard of measurement. It can be based on a single direct measurement or may be obtained by combining several measurements. A software metric is a measure of its characteristic or property. Some popular software metrics are Lines of Code (LOC), function points and McCabe's cyclomatic complexity [5]. A project metric measures characteristics such as schedule, cost and effectiveness. A good metric is defined as one that is valid, reliable, useful and easy to collect.

2.2.1 Metrics for Software Testing

Besides using the common software and project metrics like LOC, schedule, etc., certain other metrics are typical to software testing. Defects per LOC, mean time between failures (MTBF) and mean time to failure (MTTF) are among those used to determine the reliability of the code. MTBF is the average time a repairable system will function before failing [4]. MTTF is similar to MTBF except that it is used with reference to systems that

cannot be repaired and the failure is one time only. The system is more reliable if these numbers are large.

Software test automation is basically a software development project by itself because of the planning, implementation and management of test scripts that is required. It is used to test the output of another software development project, with an intention to detect defects, improve quality and even reduce cost. Since it takes an investment in one software project to improve another, it becomes very important to determine the possible ROI before investing in the former.

2.3 Return on Investment (ROI)

ROI is defined as the measure of the net income an organization's management is able to earn with its total assets. Return on investment is calculated by dividing net profits after taxes by investment. The basic ROI equation [6] [7] is:

$$ROI = \frac{Profit}{Investment}$$

ROI is a financial tool that assists management in evaluating how well it can cover its risks. Based on available data, it is used to predict future performance. This knowledge may be used to maintain the necessary growth for survival. It is also used for planning, decision making, evaluating investment opportunities, evaluating current performance and determining responses to the marketplace.

2.4 Related Work

Several models have been developed for determining the ROI on software projects and some of popular ones are discussed below. Although they are useful for software projects in general, and definitely have been a source of inspiration for this study, they cannot be directly applied to test automation. Moreover, ROI models that were developed for test automation tend to focus purely on monetary issues and lack the empirical evidence to back it up. Hence this research.

2.4.1 COCOMO II

COCOMO II [8] is an objective cost estimation model developed by Dr. Barry Boehm at the University of Southern California (USC) center for software engineering. The model is quite popular and is used for planning and executing software projects. It also supports return on investment decisions. COCOMO II has actually been developed from its precedent COCOMO and its adaptability has been improved so that it can be optimized across various domains and technologies.

COCOMO II has two underlying information models. The first is a framework used to describe a software project. This includes process models, work culture, methods, tools, teams, and the size/complexity of the software product. The second model is an experience base that can be used to estimate the likely resources, for example effort and time, of a project from historical data. This thesis will attempt to study with respect to software test automation, some of the factors COCOMO II considers.

2.4.2 Rational Suite Economic Value Model

The Rational Suite Economic Value model [9] is used to determine the value obtained from adopting the Rational Suite of tools. It has been developed by Rational and is based on COCOMO II. It focuses on metrics like managing complexity, tool experience, use of tools, architecture/risk resolution, team cohesion and process maturity. One of the assumptions made by this model is that that the entire suite of tools is adopted and not just the Rational testing tool. While this thesis employs the Rational Robot test tool for the ROI case study, it focuses more broadly on test automation tools in general. The implemented version of the Rational Suite Economic Value model can be obtained as an MS Excel sheet by emailing the author (contact details are provided in the section 'About the Author').

Chapter 3

Experimental Design

As stated in chapter 1, the aim of this experiment was to compare manual and automated testing in order to understand the ROI metrics for test automation. The metrics studied were test automation cost, schedule and effectiveness. This chapter describes the experimental design used to conduct this study.

3.1 The Experiment

To study the effect of the above mentioned metrics the experiment was designed as follows. A group of subjects was selected for the study and every member of the group performed manual and automated testing on a given software project. Since performing manual testing followed by automated testing might introduce a bias from previously testing the same software and vice versa, two similar projects were considered (P1, P2), instead of just one. One project was tested using manual testing methodologies (M) and the other used automated testing (A). The group was also separated into two sub groups (G1, G2). There were no separation criteria and the members of each sub group were selected randomly. G1 performed manual testing on P1 and automated testing on P2. Whereas, G2 performed manual testing on P2 and automated testing on P1 (Figure 1).

This way, two different groups worked on the same project (say P1), but one used automated testing and the other used manual testing methods. The results of this experiment have been used to compare the two testing methodologies (manual and automated). A similar comparison was performed with the results from the other project (i.e. P2). By cross verifying these two comparisons, unwanted variations were removed (Table 1).

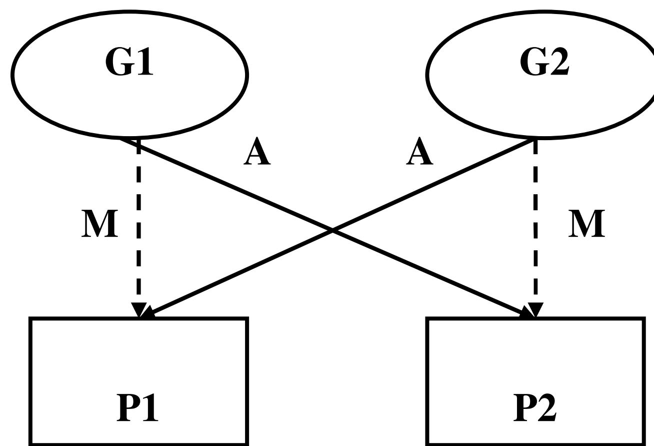


Figure 1: Experimental Setup

The test projects were chosen such that complete test coverage was possible within the project timeline. They were large enough to generate interesting test cases and at the same time, small enough to remain within the scope of this experiment. Two standard, frequently used utilities in the windows environment were selected for functional black box testing namely, the print dialog and the file dialog.

3.1.1 Print Dialog Testing (P1)

Project 1 required the subjects to test the standard windows print dialog. This dialog can be invoked from the menu option: "File > Print…" or by using the shortcut "Ctrl+P" from the application "Notepad" or several other windows based applications. The goal was to come up with a package containing 10 or more test cases that could test the important operations and settings possible using the print utility.

3.1.2 File Dialog Testing (P2)

In this project, the standard windows file dialog was tested. It can be invoked from the menu option: "File > Open…" or by using the shortcut "Ctrl+O" from the application "Notepad" or several other windows based applications. The goal here too, was to come up with a package containing 10 or more test cases that will test the important operations possible using the File access utility.

Table 1: Experimental Setup (G – Group, P – Project, M – Manual, A – Automated)

Group	Project
G1	P1 – M P2 – A
G2	P1 – A P2 – M

The experiment was introduced to the subjects as a class assignment. This assignment was a part of the graduate level software testing course curriculum at the University of South Florida (USF), in which the students were enrolled. The testing assignment document is provided in appendix A.

3.2 The Metrics

Choosing the metrics was influenced by the needs of the study as well as the experimental constraints. The study was made possible by the availability of the Rational Robot testing tool via the IBM Rational SEED program. No monetary investment was made to acquire the tool. All cost considerations made for this study are based on licensing costs quoted by IBM Rational [10] [11] and the estimates provided by the Rational Suite Economic Value model [9]. In addition, the effort in terms of person hours was also considered for the cost metric.

The project schedule was an important area of focus for this study. The study itself was conducted as a part of the course curriculum and thus had a well defined timeline. It was conducted over a period of five weeks, where the first two weeks were allocated for tool training and the remaining three weeks for implementation.

The effectiveness metric was subjective in nature. Although the measurement of this metric may be debated, the urgent need to study its effect, and the absence of a similar study, encouraged it to be a part of this study. Test automation effectiveness was measured by studying the number of actual defects and false positives reported. Another measure of effectiveness was the response given by the subjects to a survey. The survey itself allowed for ratings using five point Likert scales as well as textual feedback. The responses were compared to factual data such as the number of person hours put in by the subjects and the actual output generated. This combination provided the productivity details as well as the human factors that influenced it.

Finally, it is well known that people are the most important resource of any operation. Measuring their contribution, capability and the factors that affect them is vital

to a project's success. Data such as previous testing experience, productivity and effectiveness user ratings, test coverage, testing scope, testing depth, documentation and traceability etc. were collected. They have been discussed in chapter 3.

3.3 Data Collection

The data was collected over a period of five weeks. A project timeline (Table 2) was followed and the subjects' progress during this period was monitored.

Table 2: Project Timeline

Phase	Duration
Training	Two weeks (March 7 – March 20, 2005)
Implementation	Three weeks (March 21 – April 10, 2005)

During the first two weeks the subjects were trained in automated testing techniques. They were first introduced to test automation via a presentation (appendix D) and this included an automated test script demonstration. The sample test script used for the demonstration is included in appendix D. This was followed by two hands-on lab training sessions that were spaced a week apart. The training documents used for these sessions have been provided in appendix E and appendix F. In addition, the Rational Robot testing tool user guide and the SQABasic scripting language reference guide from IBM Rational [12] were made available for training. The training phase was followed by the three week implementation phase of the experiment, where the test scripts were created and executed. The test case template form has been provided in appendix B. At

the end of the experiment, the twenty test cases (10 automated, 10 manual) and defects (if any) submitted by each of the subjects were collected. The defect reporting template form has been provided in appendix C. These test cases and defects were checked for validity before being accepted.

Finally, a user experience survey was used to collect subjective information. The survey contained a total of 37 questions, ranging from simple Yes/No and 5 point Likert scale questions, to questions requiring detailed answers. The survey was divided into four sections. The first three concentrated on one of the three main metrics considered for this study. The last section focused on other contributing subjective factors. Each of these responses was cross verified and used to justify the actual experimental results. The survey document has been provided in appendix G.

3.4 Other Details

This experiment was conducted under strict compliance with the Institutional Review Board (IRB) rules and guidelines at USF (IRB approval # 103422F). The subjects faced no known risks by participating in this study.

Chapter 4

Results & Discussion

Demographic data for the experiment has been provided. This is followed by the results. They have been organized based on the three parameters of interest, for which the metrics were collected.

- Schedule
- Cost
- Effectiveness

The results under each category are followed by a discussion which includes the inferences made during the study. The test scripts, test case forms, defect reports and survey data submitted by the subjects for this experiment is available upon request [17].

4.1 Demographic Data

The group selected for the study had 24 subjects. 12 were male. The average age of the subjects was 25 years.

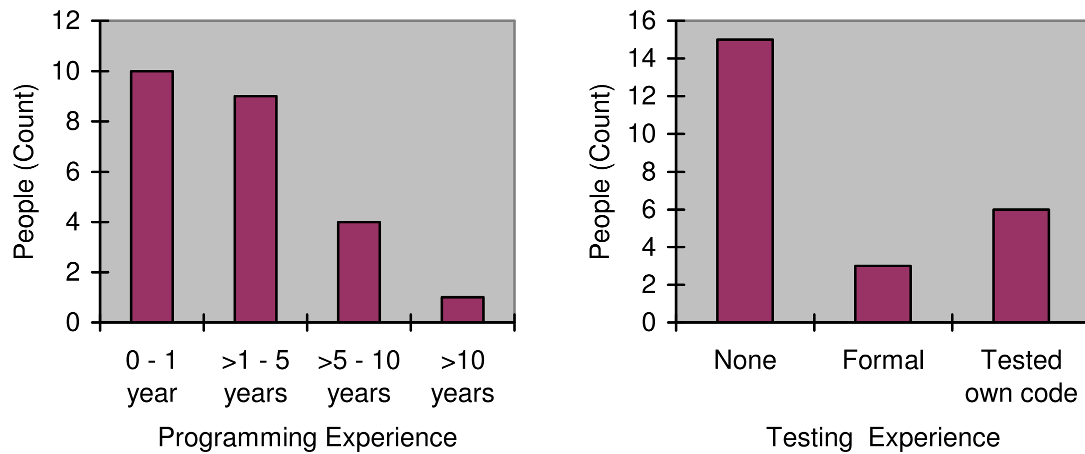


Figure 2: Subject's Programming and Testing Experience

Nearly 80% of the subjects had less than 5 years of programming experience. A lot of the experience was around 2-3 years. None of them had zero programming experience (Figure 2). Due to the lack of prior software testing experience, the 24 subjects could not be categorized by the number of years. Rather the subjects were categorized based on whether they had any testing experience involving formal testing techniques, tested their own code during software development or had no testing experience at all. From figure 2, it is observed that more than 60% of the subjects had no formal software testing experience. About 15% of the test population had a formal experience with software testing. The remaining 25% of them who had experience testing their own code had performed unit and structural tests. Debugging code during software development was not considered as code testing experience. None of them had taken a software testing course before enrolling in the current graduate level software testing course of which this experiment is a part. Except for one individual, no one else had prior

experience with an automation tool. Having limited past training in automation makes it better for the experimental setup since it eliminates any prejudice and ensures that all feedback from the subject is based on recent experience from the experiment.

4.2 Schedule

This section presents the metrics collected for the schedule of the software testing project. Based on the number of builds, a project usually consists of several testing cycles where each testing cycle performs one complete test of the software. Based on the different tasks performed, a testing cycle may be classified into different phases as follows:

- Phase 1: Analysis
- Phase 2: Documentation
- Phase 3: Tool Training (Applicable to automation only)
- Phase 4: Implementation (Applicable to automation only)
- Phase 5: Execution

Each phase takes a certain amount of time to be completed. This duration depends on the nature of the tasks performed during the phase. A testing cycle can be represented in the form of a timeline, which is actually a graphical representation of all the phases of a testing cycle over time (Figure 3). The length of the timeline is the sum of the durations it takes to complete each of the fives phases.

Timeline length = Analysis time + Documentation time + Tool training time + Implementation time + Execution time

To study the effect of automated versus manual testing on the project schedule; their respective testing cycle timelines can be compared. Data from the experiment is used to construct the automated and manual timelines for the first testing cycle. It is later projected for the second and third testing cycles in order to observe a trend (if any). The timelines are cumulative in nature and each subsequent timeline is an extension of the previous testing cycle's timeline.

After the first testing cycle, based on the nature of testing to be performed, the remaining testing cycles may be further categorized. The automated and manual testing timelines of the second and third testing cycles are discussed separately under each of the following categories:

- Regression testing
- Pure repetitive testing

For test automation the phases 1, 2 and 5 are the same as in manual testing. Phases 3 and 4 apply to test automation. This gives us an advance indication that extra effort, cost and time might be incurred. In a sense, test automation is more of an extension to manual testing.

Keeping the current experimental setup in context, the time it would take to make an analysis and come up with 10 test cases is estimated as 1 hour. Similarly the time to document the 10 test cases would take another 1 hour. These estimates need not be very accurate for comparison purposes, as phase 1 and 2 are the same for manual and automated testing. According to the data collected from the experiment, the average time to learn the testing tool (phase 3) was 3.85 hours, and the average time to implement the

test cases (phase 4) using the test tool scripting language was 4.01 hours. The average LLOC (Logical Lines Of Code) per user was 140. This means that the time taken to write a single LLOC is 1.7 minutes, which is reasonable considering it includes the time to compile and debug the test script errors.

On average, it takes 30 minutes to execute the ten test cases manually. This includes the time to read and interpret the test cases and other human latency. Automated test case execution takes around a minute per test case bringing the time for 10 test cases to 10 minutes. This is one third of the time it takes to execute the test cases manually. Therefore the time saved by automated test case execution is 20 minutes.

Now, the time saved is actually more than 20 minutes if we understand what we are concerned with:

Actual automated execution time

Vs.

Time used up on the automated timeline

For comparison purposes, we are concerned with the time used up on the automated timeline. The actual automated execution time may be 10 minutes but as long as execution can occur in parallel with other tasks, without human intervention, it does not affect the tester's timeline. The tester is free to work on the analysis (phase 1) and documentation (phase 2) phases of the next testing cycle if more software changes are being introduced in the next build. All the tester needs to do is to start the automated execution process and continue with other tasks. So, the time duration that reflects on the timeline is the time it takes to start the automated execution process and it has been estimated as 3 minutes (10% of the manual execution time) for this experiment.

Therefore, the time saved by automated test case execution is now 27 minutes. Hence forth, the automated execution time will refer to the time used up on the timeline.

Figure 3 shows the automated and manual testing timelines for a single tester. Both of them correspond to the first testing cycle. Comparing manual and automated testing in figure 3, the automated testing timeline is longer than the manual testing timeline by 7.41 hours. This schedule overflow is due to the large training and implementation times for the automated testing cycle (Table 3).

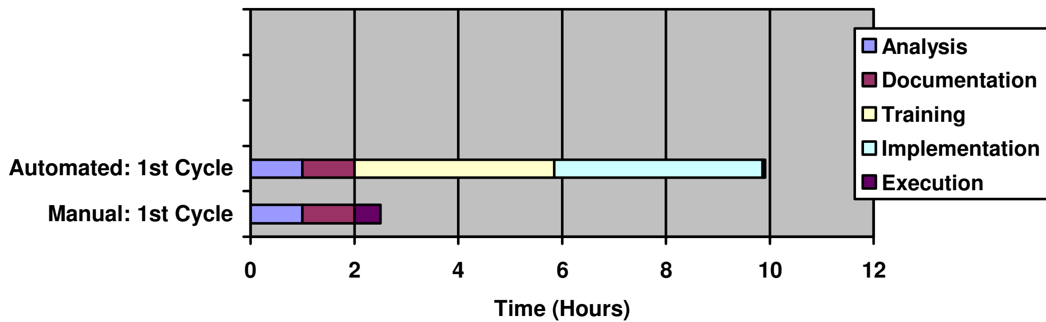


Figure 3: 1st Cycle Time lines – Manual Vs. Automated

Table 3: 1st Cycle Time Periods – Manual Vs. Automated

Phase	Manual (Hours)	Automated (Hours)
Analysis	1	1
Documentation	1	1
Tool Training	0	3.85
Implementation	0	4.01
Execution	0.5	0.05
Total	2.5	9.91

4.2.1 Regression Testing

Regression testing is the testing of existing code when new code is introduced. Projecting the above results for regression testing in the second test cycle (Figure 4,

Table 4), we remove the training time and cut down the implementation time to 10% of the original value. According to industry averages [13], 10% of the original script implementation time is required to maintain and modify the scripts for the next cycle. For this experiment it is 0.40 hours. It can go up to 15% based on project complexity. The automated and manual timelines for the second testing cycle involving regression testing are cumulative in nature and are shown in figure 4.

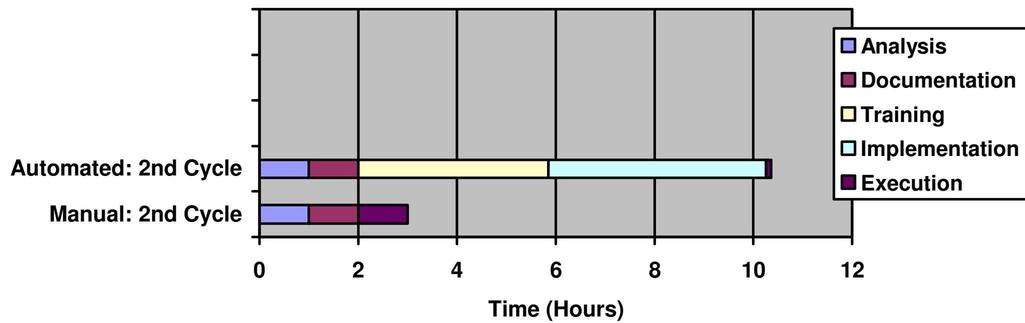


Figure 4: Regression Testing: 2nd Cycle Time lines – Manual Vs. Automated

Table 4: Regression Testing: 2nd Cycle Time Periods – Manual Vs. Automated

Phase	Manual (Hours)	Automated (Hours)
Analysis	1	1
Documentation	1	1
Tool Training	0	3.85
Implementation	0	*4.41*
Execution	*1*	*0.1*
Total	3	10.36

The numbers in table 4 contributing to the growth of the cumulative timeline in this testing cycle are in italic. We observe a schedule over flow of 7.36 hours for automated testing when compared to that of manual testing. This is lower than the overflow recorded at the end of the first testing cycle which was 7.41 hours. This

indicates that the automated testing timeline is growing at a slower rate than the manual testing timeline.

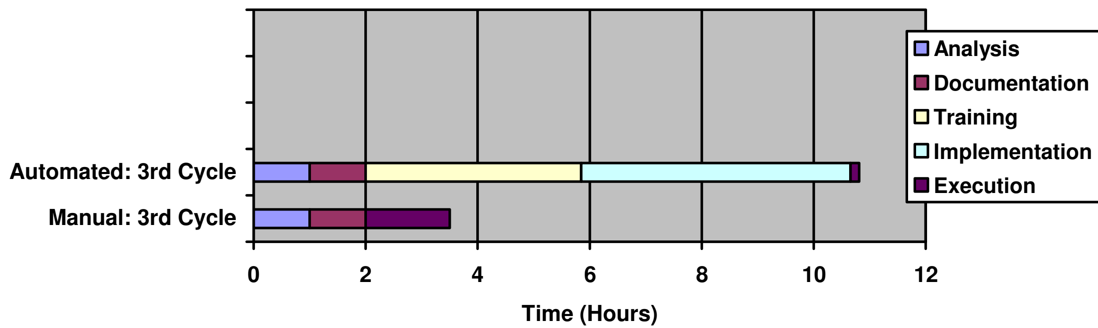


Figure 5: Regression Testing: 3rd Cycle Time lines – Manual Vs. Automated

Table 5: Regression Testing: 3rd Cycle Time Periods – Manual Vs. Automated

Phase	Manual (Hours)	Automated (Hours)
Analysis	1	1
Documentation	1	1
Tool Training	0	3.85
Implementation	0	*4.81*
Execution	*1.5*	*0.15*
Total	3.5	10.81

And by the third testing cycle (Figure 5, Table 5) the automated testing schedule overflow comes down to 7.31 hours. The decreasing overflow trend as seen in table 6, illustrates the fact that after the first cycle, manual testing is more time consuming than automated testing, but the automation timeline is still longer than the manual timeline due to the large initial investment of time made during the training and implementation phases of the first testing cycle.

Table 6: Regression Testing: Automated Testing Cycle Schedule Overflows

Testing Cycle	Overflow (Hours)	Differential (Hours)
Cycle 1	7.41	
Cycle 2	7.36	0.05
Cycle 3	7.31	0.05

The overflow gap is steadily decreasing every test cycle and at the current rate of 0.05 hours per testing cycle it would take 147 test cycles before the breakeven point is reached. This is when automation becomes more productive on schedule and ultimately helps ship the product sooner.

4.2.2 Pure Repetitive Testing

Sometimes the breakeven point can be attained much earlier than the 147 test cycles required for regression testing. This would be in the case of pure repetitive testing like, stress, load and performance testing [29]. Here the unmodified test scripts are run several times on the same software system. These scripts do not require implementation time as the scripts remain unchanged between test cycles.

If you observe the manual and automated timelines for the second testing cycle involving regression testing (Tables 4, 5 and 6), you will notice that the manual timeline grows at a rate of 0.5 hours per cycle, whereas the automated timeline grows at a rate of 0.45 hours (0.40 for implementation and 0.05 for execution). This time difference of 0.05 hours per testing cycle (Table 6) is what causes the eventual breakeven for regression testing.

When considering pure repetitive testing, script implementation time is brought down to zero as the software has not changed and therefore no script modifications are

needed. This would eliminate 0.40 of an hour and the automated timeline would only grow at a rate of 0.05 hours per testing cycle. The manual timeline continues to grow at the rate of 0.5 hours per testing cycle. The time difference between the two timelines is now 0.45 hours per testing cycle.

Let us see the timelines for the second testing cycle involving pure repetitive testing (Figure 6, Table 7). Notice that the schedule overflow is now only 6.96 hours instead of the 7.36 hours overflow we observed during the second testing cycle of regression testing.

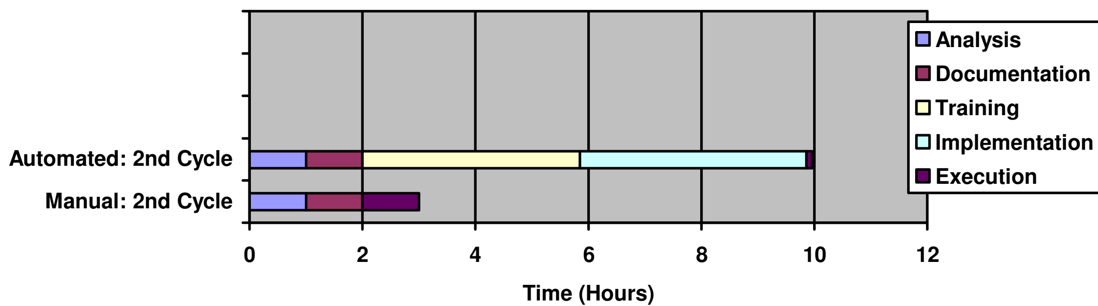


Figure 6: Pure Repetitive Testing: 2nd Cycle Time Lines – Manual Vs. Automated

Table 7: Pure Repetitive Testing: 2nd Cycle Time Periods – Manual Vs. Automated

Phase	Manual (Hours)	Automated (Hours)
Analysis	1	1
Documentation	1	1
Tool Training	0	3.85
Implementation	0	4.01
Execution	*1.0*	*0.10*
Total	3.0	9.96

Finally, let us compare the manual and automated timelines for the third testing cycle (Figure 7, Table 8). Again, the schedule overflow is only 6.51 hours instead of the 7.31 hours overflow observed during the third testing cycle of regression testing.

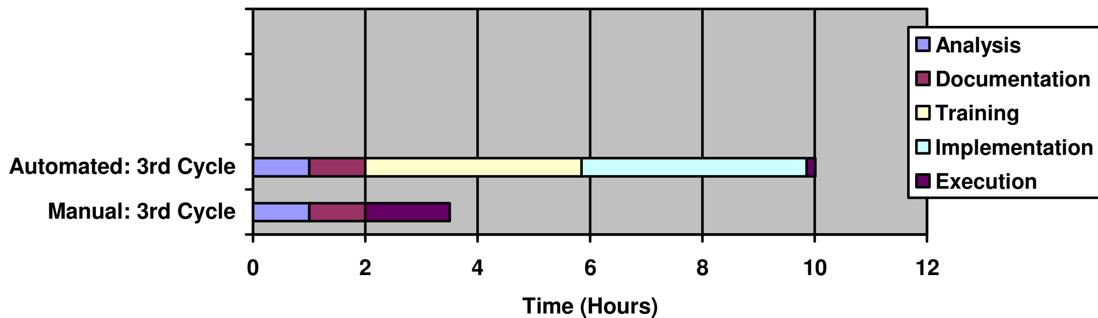


Figure 7: Pure Repetitive Testing: 3rd Cycle Time Lines – Manual Vs. Automated

Table 8: Pure Repetitive Testing: 3rd Cycle Time Periods – Manual Vs. Automated

Phase	Manual (Hours)	Automated (Hours)
Analysis	1	1
Documentation	1	1
Tool Training	0	3.85
Implementation	0	4.01
Execution	*1.5*	*0.15*
Total	3.5	10.01

From table 9, we see that the convergence to breakeven point is faster (0.45 per testing cycle) and requires just 15 more test cycles.

Table 9: Pure Repetitive Testing: Automated Testing Cycle Schedule Overflows

Testing Cycle	Overflow (Hours)	Differential (Hours)
Cycle 1	7.41	
Cycle 2	6.96	0.45
Cycle 3	6.51	0.45

Drawing an estimate based on the Rational Unified Process [14] for a project of this scale, there will be at least one testing cycle each for the implementation and testing phase of software development. Sometimes there are two testing cycles for each of these phases followed by a user acceptance test during the software development deployment phase. So, most organizations on average have between 2 and 5 test cycles before a product release. According to these calculations it becomes clear that automation would not benefit this project, be it regression (147 cycles to breakeven) or pure repetitive testing (15 cycles to breakeven). This outcome was expected as it is quite obvious from the size and scope of this project that we could have easily completed 4 manual test cycles in the time it took to automate the test cases.

Figure 8 compares the projected timelines for manual, automated regression and automated pure repetitive testing. It is observed that the pure repetitive testing timeline reaches the breakeven point much earlier than automated regression testing timeline. For the sake of clarity, a change of scale has been introduced between the 15th and 25th testing cycles on figure 8. This has been done to clearly show the crossover when the breakeven occurs. It is also observed that manual testing has a shorter testing schedule when compared to automated regression testing, until the breakeven point is reached at around the 150th testing cycle. It is therefore vital to choose the testing technique based on the size and scope of the testing project.

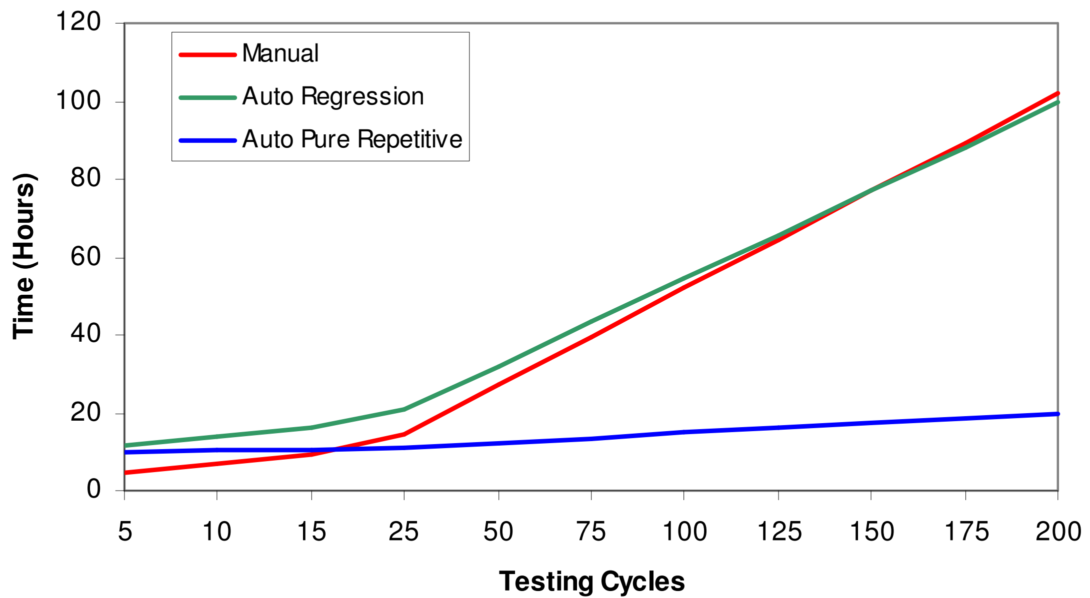


Figure 8: Timeline Comparison – Manual Vs. Regression (auto) Vs. Pure Repetitive (auto)

The important thing to understand are the factors to be considered while deciding whether to automate or to test manually. These would be,

Implementation time + Automated execution time (for automation timeline)

Vs.

Manual execution time (for manual timeline)

They are responsible for the growth of their respective timelines after the first testing cycle. The implementation phase after the first testing cycle mostly performs test script maintenance. The implementation time during this phase varies according to

- Frequency of software changes

- Script complexity
- Level of script reusability
- Effort required to process test data before script run
- Effort required to restore system state before script rerun
- Effort required to restore system state after abnormal termination of the test execution

The key is to keep this implementation time low enough so that the automated timeline grows at a rate slower than the manual timeline. In other words, opt for test case automation only if the sum of implementation time and automated execution time is lower than the manual execution time.

$$implementTime_{auto} + execTime_{auto} < execTime_{man}$$

Automation would therefore be a good idea in testing projects where the test scripts are short, simple and reusable i.e. where little effort is spent on implementation for test script maintenance. Another point to remember is that the breakeven point needs to be reached within the planned number of test cycles for the project. The difference between the automated timeline and the manual timeline for any testing cycle gives the current automated testing schedule overflow that needs to be covered in order to break even. Based on the remaining 'x' number of testing cycles, 1/x of this overflow should be lesser than the additional growth rate automated testing has over manual testing in each testing cycle. This difference in growth rate should recover at least 1/x of the overflow during each of the 'x' testing cycles. This difference may be calculated by subtracting the

sum of the implementation time and automated execution time from the manual execution time. This observation is shown in the form of an equation below.

$$\frac{1}{x}\left(timeline_{auto} - timeline_{man}\right) < execTime_{man} - \left(implementTime_{auto} + execTime_{auto}\right)$$

Where,

x = number of testing cycles remaining for the project

For example, in our experiment if we had performed pure repetitive testing, at the third testing cycle, we needed 15 more testing cycles to break even. Substituting this and the values from table 8 in the equation above,

=> (1/15) * (6.51) < (0.50 – (0.0 + 0.05))

=> 0.434 < 0.45

4.3 Cost

There are several kinds of costs incurred when introducing test case automation. They may be classified as either fixed or variable costs [15].

4.3.1 Fixed Costs

The following fixed costs are applicable to test automation:

- Licensing cost
- Training cost
- Infrastructure cost

There are several options that may be considered when licensing a tool:

Buy vs. Lease – A testing tool may be purchased outright or leased on a monthly or yearly basis if the duration of the project is not certain. The monthly lease may be slightly more expensive when compared to the monthly cost if the tool were bought, but it is a good option for pilot projects and customer specific short term testing requirements. For this study, the purchase option was considered, as the goal was to understand how automation would affect a long term software testing project.

Authorized vs. Floating user – By buying an authorized user license, the software is registered to a single user on a single computer and the license may not be transferred. Effectively only one dedicated user may use the software and it cannot be shared. It costs $4120 for one authorized Rational Robot testing tool license [10]. A floating user on the other hand can share the license with other users but only one user may have the license at any one time. This gives more flexibility and would need a licensing server to be setup.

This flexibility comes at a cost. The price for a Rational Robot floating user license with software maintenance is $7930 [10].

The training cost for this project was calculated as follows. The training phase was spread over two weeks and much of it was covered with two 3 hour hands-on lab sessions. A similar 2 day classroom and hands-on lab training provided by Rational costs $1,195 per person [11].

The infrastructure required besides what will be used normally for manual testing would be a powerful server that can host a licensing server and databases for defect tracking and test case version control. A server machine costs about $1000.

Total fixed costs for this experiment = Licensing cost + Training cost + Infrastructure cost

= $7,930 + $1,195 + $1,000

= $10,125

4.3.2 Variable Costs

The variable costs applicable to test automation are:

- Project cost
- Maintenance cost

The project cost is the cost incurred during the various testing cycles of the project. It is dependant upon the project schedule and is calculated as follows:

*Project cost = number of person hours * cost of tester per hour*

Where,

Number of person hours = number of testers * hours worked

The maintenance costs could involve hiring consultants for tool expertise, infrastructure maintenance, tool upgrades, tech support etc. or all these can be covered as a part of the tool vendor licensing contract, at a fixed additional cost. The latter applies to this study. Therefore the variable cost for this experiment is simply the project cost alone. Let us now consider the variable cost of the first testing cycle for this experiment as shown in figure 9.

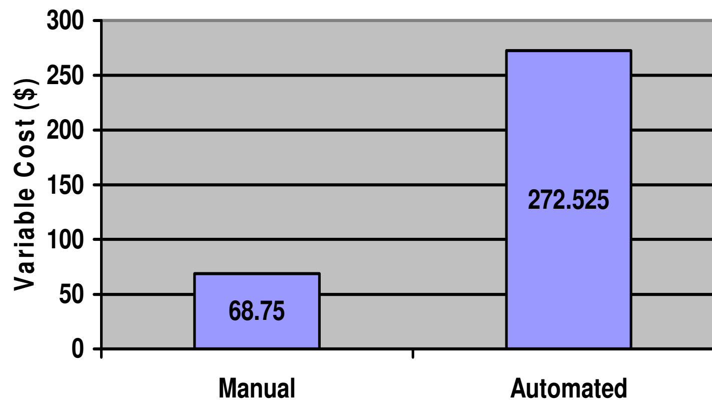


Figure 9: First Cycle Variable Cost – Manual Vs. Automated

*Variable cost for the first cycle (automated) = 9.91 * 27.5 * 1 = $272.525*

*Variable cost for the first cycle (manual) = 2.5 * 27.5 * 1 = $68.75*

Where,

Hours worked = 9.91 hours for automated testing, 2.50 hours for manual testing (From table 3).

The average wage per hour for a software tester = $27.5 (This data is obtained from the Monster Salary Center [16] and is calculated based on 40*50 work hours per year, excluding holidays).

Number of testers = 1

Note: the total cost can be calculated by adding the fixed cost and the variable cost. However, for the sake of clarity, the fixed cost will be discussed separately later. From figure 9, it is clear that the variable cost for automated testing is greater than the variable cost for manual testing. This variable cost over flow for automated testing can be calculated as follows:

Automated variable cost – Manual variable cost = $272.525 – $68.75 = $203.775

From the above discussion it is observed that the variable cost overflow is directly proportional to the schedule overflow. We may calculate the variable cost overflow for the second and third testing cycles by using the schedule overflows calculated in the previous sections.

4.3.3 Regression Testing

Since we already have the regression testing schedule overflow values from table 6 in section 4.2.1, we can directly calculate the overflow variable costs for the second and third regression testing cycles. This is shown in table 10.

Table 10: Regression Testing: Testing Cycle Variable Cost Overflows

Testing Cycle	Overflow (Hours)	Cost ($)
Cycle 1	7.41	203.775
Cycle 2	7.36	202.4
Cycle 3	7.31	201.025

4.3.4 Pure Repetitive Testing

Using table 9 from section 4.2.2, the variable cost overflow for second and third pure repetitive testing cycles can be calculated. This is shown in table 11.

Table 11: Pure Repetitive Testing: Testing Cycle Variable Cost Overflows

Testing Cycle	Overflow (Hours)	Cost ($)
Cycle 1	7.41	203.775
Cycle 2	6.96	191.40
Cycle 3	6.51	179.025

4.3.5 Accounting for Fixed Costs

As seen in section 4.2, when the automated testing schedule and the manual testing schedule reach a breakeven point after several testing cycles, the automated variable cost overflow will also be accounted for. This hypothesis is supported by the steadily decreasing variable cost overflow in table 10 and table 11. However, what is not

accounted for is the fixed cost for automated testing. Considering the best case scenario which is pure repetitive testing, the difference between any two consequent variable cost overflows in table 11 gives the profit margin. This is the cost savings automated testing has over manual testing, per testing cycle. It is $12.375 for this experiment. With a low profit margin and a high fixed cost ($10,125), it doesn't make sense to hope for a positive return on investment from the testing cycles of the current project alone. Rather, the best option would be to amortize the fixed cost over several testing projects. The average profit margin of several testing cycles (spread across several testing projects) can be calculated by summing up their individual profit margins and dividing it by the total number of testing cycles minus one. This is shown as an equation below:

Average Profit Margin =

$$\sum_{i=1}^{T-1} \frac{\textit{Automated Variable Cost Overflow}_i - \textit{Automated variable cost overflow}_{i+1}}{\textit{Number of testing cycles (T)} - 1}$$

The fixed cost when divided by the average profit margin, gives the minimum number of testing cycles needed to recover the cost. This should be lower than the number of testing cycles an organization plans to complete within a prescribed number of years (n), by which it hopes to obtain a positive ROI. The average time by which an ROI is expected for a software technology is usually three years [9], which is the time it takes for the technology to become obsolete. This is expressed as an equation below:

$$\frac{\textit{Fixed Cost}}{\textit{Avg Profit Margin}} < n * 12 * \textit{Projects per Month} * \textit{Test Cycles Per Project}$$

The conclusion drawn is that it is important to consider the fixed cost investment before adopting automated testing. Since the fixed costs are large, it is more practical to try and amortize the cost over several testing projects instead of just one. Besides the average profit margin of the testing cycles belonging to different testing projects should be large enough to amortize the fixed costs within the planned number of years. This should drive the decision to automate.

4.4 Effectiveness

The test effectiveness for this experiment is determined by studying the nature and number of defects found. Several defects were reported but not all were actual defects. Some of them were false positives. Following is a discussion of the interesting defects detected by manual and automated testing. An attempt is made to recognize if a defect pattern exists.

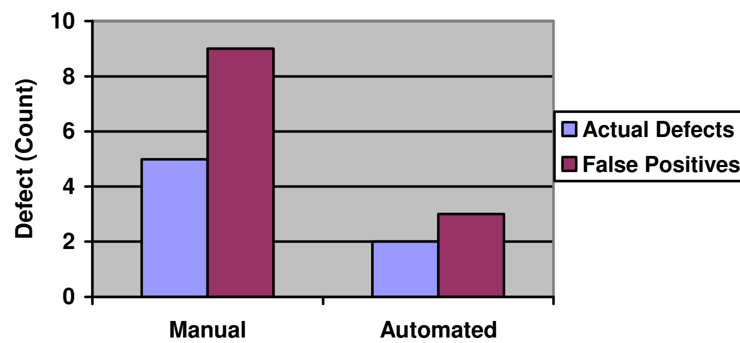


Figure 10: Actual Defects & False Positives – Manual Vs. Automated

Table 12: Actual Defects & False Positives – Manual Vs. Automated

Classification	Manual	Automated
Actual Defects	5	1
False Positives	9	3

A total of 14 defects (Figure 10, Table 12) were reported via manual testing out of which 5 were valid and 4 defects were reported via test automation, out of which 1 was valid.

Here is the defect that was detected during automated testing:

- The PDF995 print driver that was used to simulate printing while testing the file print dialog did not support multiple page printing and collation. Even if these features were not part of the intended functionality, they should at least be disabled on the file print dialog.

Here are some of the defects that were detected during manual testing:

- In the Microsoft Word file open dialog, typing a filename that does not exist and clicking on the 'open' button, does not invoke any system response. The user should be informed that the requested file could not be found.
- In the Microsoft Word file open dialog, the 'create new folder' option is enabled. If a new folder is created, it certainly has no files within it that can be opened, which makes this option inappropriate for a file open dialog.
- A recently accessed document is usually listed in the 'My Recent Documents' section of a Windows operating system and it can be invoked from this list. Now, delete

this document from its original location and replace it with a different document giving it the same name as the original. The original document will still be listed in 'My Recent Documents' section and trying to invoke it will bring up the new document instead of the original.

- When large files are attempted to be opened using the file open dialog in the applications 'Notepad' or 'Adobe Acrobat Reader', the systems hangs if enough memory is not available to load the file. The applications must handle the situation gracefully. Memory checks can be enforced and users can be given an option to terminate the action if required.

By analyzing the manual defects we get an idea of how devious the manual test cases were [17]. This is why manual testing is highly suited for exploratory testing. The automated test cases in comparison didn't have much variation. The test cases (scripts) were modifications of a basic test script [17]. As a result the scripts were performing localized testing. This is probably because scripting takes up valuable test case creation time and testers tend to reuse test scripts since it saves a lot of effort. In addition, writing a test script can hinder the creative thinking process. A tester needs to think like a programmer and a tester at the same time, which is not easy.

On the other hand, it was interesting to see the nature of the false positives detected via test automation.

- In one case, the tester's lack of experience with the tool caused him to use the wrong functions in the testing script. The tester was trying to open a text file in Microsoft

Word. Failure due to the use of incorrect functions was misinterpreted as failed functionality.

- In another case a tester actually embeds the error within the test script unknowingly (faulty test data). When specifying a file name in the file save dialog, it accepts all special characters except '*'. For this particular test case, every time the test script was run, the test case failed when it encountered the fourth special character. It was wrongly assumed that a maximum of only 3 special characters are supported by the file save dialog. The actual reason was that the '*' character was the fourth character in the test data. The tester failed to recognize this fact because he/she did not handle the test data during the automated test case execution. The test case itself was correct but the defective data led to test failure.

There were several false positives detected by manual testing [17]. They are not discussed in detail here since they all had a common cause. The tester simply misunderstood the intended functionality of the software.

From the above discussion it is clear that manual testing is suited for exploratory testing and in addition to the kind of false positives detected by manual testing, test automation is capable of generating false positives that are testing tool and automation specific. Finally, due to the lack of variation in the automation scripts compared to the manual test cases, and their very low execution times (discussed in section 4.2), automation will probably work well for high volume repetitive testing like stress, load and performance testing, rather than exploratory testing. This hypothesis can be explored during future work.

Note: Neither the DDP (Defect Detection Percentage) or the DRE (Defect Removal Efficiency) metric could be used for this experiment, since only one major testing cycle was performed and an estimate of the final number of defects could not be made.

4.4.1 Lessons Learned

- Be aware of false positives that are testing tool and test automation specific.

- The simple fact that the test case fails and displays an authoritative 'FAIL' log message in red should not psychologically influence the tester to log a defect report. Instead, analyze the defect first.

- Use data pools for testing, wherever possible. It can provide random test data each time the test case is run. Even if it fails once, it has a good chance of succeeding the next time it is run to verify the error. This can reduce the occurrence of data specific errors as observed earlier with an automation false positive.

4.5 Other Metrics

In addition to schedule, cost and effectiveness, here are some other observations that were made during the study. These could affect test automation either directly or indirectly. A few of them are subjective in nature. But according to Dreyfus' model of skill acquisition [18], as a person learns by experience, he/she no longer relies on rules, guidelines or maxims. Rather he/she has the ability to see situations holistically and

intuitively grasps them based on deep understanding. I hope the following discussion encourages such understanding.

4.5.1 Scope of Testing

For this project the subjects found it easy to execute the test cases once the scripts were ready. Also the application under test did not undergo any changes enabling them to use the test scripts repeatedly. As discussed in the previous section, automated test cases had less variation when compared to the manual ones [17]. They not only were fewer in number, but also exposed fewer defects.

Automated testing could be used for creating smoke tests. A smoke test is a non-exhaustive software test to ascertain that the most crucial functions of a system work [20]. The finer details of the system are not tested during a smoke test. A smoke test can be used to verify the stability of a system before introducing any changes.

It can also be used to test the unmodified parts of the software after additional features have been introduced. This will have a better chance at detecting defect creep as the power of automation is in its ability to execute the test case exactly as it was intended to.

It may be used to stress the system as it is easy to execute the test case several times once it has been created. Every automated test case is derived from a manual test case. It doesn't justify the effort in creating an automated test case without manually trying it out first. Unless, it is a performance based test, which usually cannot be done manually.

Automation does not suit exploratory testing. A test case is automated with an intention of repeating it. It loses its novelty when repeated and is not exploratory anymore. It's only as devious as its manual counterpart.

Furthermore, test automation tools today are environment and platform dependant. The Rational tool used for this experiment can be used for automating the testing of Microsoft Windows client/server and Internet applications running under Windows NT 4.0, Windows XP, Windows 2000, Windows 98, and Windows Me. It supports testing of applications developed with IDE's such as Visual Studio.NET, Java, HTML, Visual Basic, Oracle Forms, Delphi, and PowerBuilder. It does support testing of custom components but the capabilities are limited. These factors must be considered while choosing a testing tool.

4.5.2 Depth of Testing

This is one area where the project suffered most. Much of the effort was spent in scripting the test cases. Since the project schedule could not be increased, the project scope was reduced. This affected the depth of testing. Also, not all test cases that needed automation could be automated due to technical difficulties. In some cases Rational Robot did not recognize certain custom controls in Microsoft Word and they had to be tested manually. Verification of graphical output data, such as .PDF files was done manually too. Another challenge was the added overhead of writing scripts to reset the system state every time a test script was executed. This was required so that the test script could be run again in its original environment. It also ensures that the test script executions are independent of one another, wherever required.

Of the four popular test automation frameworks [21], the test script modularity framework was selected due its simplicity and macro nature. It focuses on the creation of small independent test scripts that represent functions and sections of the application under test (AUT). Keeping it small and function specific enables test case reuse. This was important for this project which was small scale and the subjects were relatively inexperienced with the tool.

The test library architecture framework creates a library of procedures that can be called to test functions and sections of the AUT. It requires creation of an extensive library and is justified only if script reuse (not just test case reuse) is anticipated. It also helps when testers inexperienced with the tool are involved in modifying test scripts and a level of abstraction is required.

The data driven test framework is quite effective in bringing test variation to automated test scripts. It involves using variables in the test script which are replaced with random values from a database each time the test case is run. Interestingly, a false positive defect was reported which would not have been if this framework had been adopted. Details discussed in section 4.4.

Finally, the keyword driven architecture involves the creation of a keyword table with the columns like 'Window', 'Control', 'Action' and 'Arguments'. The script picks up details of the action to be performed on a control in a particular window from this table. This would require an extensive inbuilt script library to exist.

Besides the testing framework, the recognition capability of the tool also plays an important role. Screen controls are recognized in two ways [21]:

- Recognition by screen coordinates

- Object recognition

Object recognition is better of the two and gives greater control over the depth of testing. Recognition is by a unique property of the control and is irrespective of its position on the screen. Recognition by screen coordinates does not give access to the properties and state of the control.

4.5.3 Test Tool Adoption

Adopting a test tool is a major challenge when shifting from manual to automated testing. Test automation involves test case scripting and this requires a certain level of programming experience, without which the testers cannot understand IF, FOR and other programming constructs used in the scripts. Manual testers by default are not expected to be experienced in programming. According to the experiment's survey (Figure 11), this is how the subjects in this experiment thought prior programming experience affected their performance on a scale of 1 (negligible) to 5 (very high). Those with no prior programming experience, answered 'N/A'.

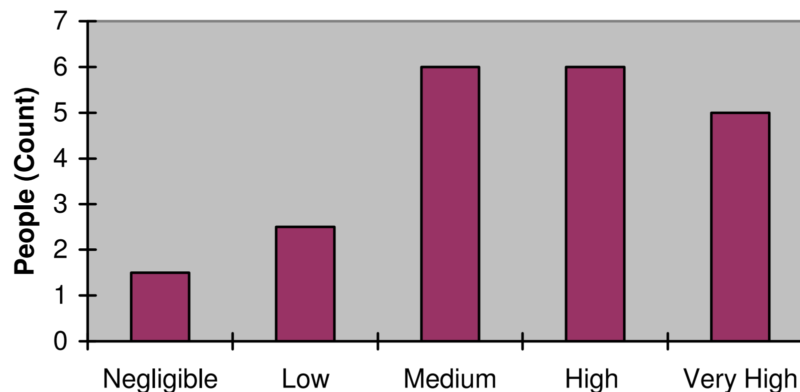


Figure 11: Effect of Programming Rating

Since most of the subjects had 2-3 years of programming experience (Figure 2), they were comfortable with the concept of test case scripting. In addition, the GUI recording option of the testing tool made learning easier. The subjects could auto-generate scripts and modify them according to requirements. However, the script creation process was time consuming. Subject reports of script writing experience are shown in figure 12.

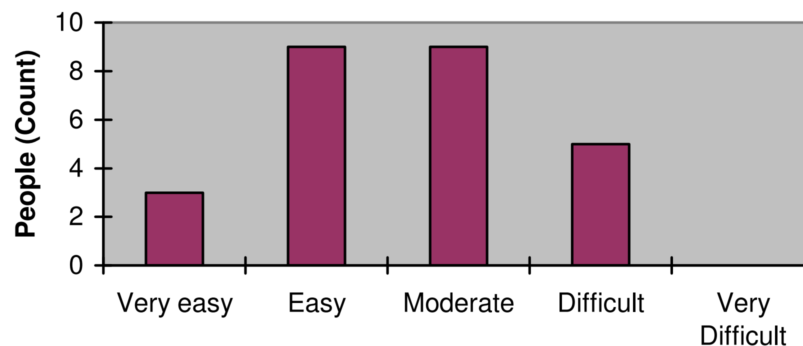


Figure 12: Scripting Experience

If the available testers lack programming experience, a dedicated test automation programmer can be hired. Although it helps, this person need not be a tester. His/her responsibilities would be to automate the test cases rather than thinking them up. This person may be shared across several projects.

Figure 13: Training Effectiveness Rating

Besides prior programming experience, adequate tool training also plays a vital role in a successful test tool adoption. For this experiment, training was provided over a period of two weeks with several hands on lab sessions (details in chapter 3). Subject feedback on the effectiveness of training is shown in figure 13.

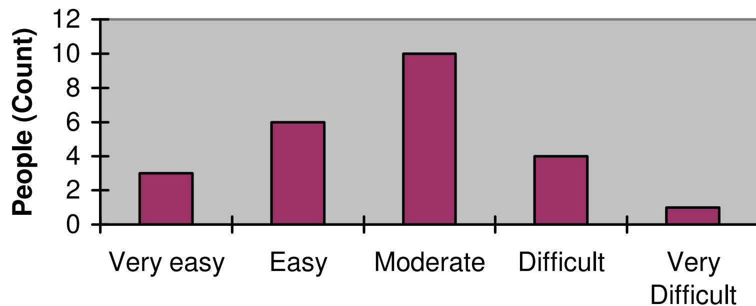


Figure 14: Test Tool Usability Rating

Finally, the usability factor of the tool also contributes to test tool adoption. If a tool is user friendly, it has a low learning curve. The subjects found the Rational Robot testing tool used in this experiment user friendly and intuitive (Figure 14). They were able to successfully adopt and use the tool within the planned two week training period.

4.5.4 User Ratings for Effectiveness & Productivity

Figure 15: Effectiveness & Productivity Rating

The user rating on the effectiveness and productivity of test automation is shown in Figure 15. Subjects referred to test automation as an important skill in the market. One subject said he did not find much use for automation. Actually, the subjects didn't see the benefit for the small project they were working on and believed it would do better for regression testing (point highly stressed). They also felt that it improved test coverage and eliminated human error. Finally although, automation cannot replace human creativity, it is faster and the amount of productivity depends on what is being automated.

It was interesting to observe a good rating on test automation effectiveness and productivity even though the number of defects found using automation was far less than with manual testing. Much of the optimism was based on projecting its capability on a project of larger proportions that would involve regression testing.

Furthermore, it reflects the high confidence in automation among users and customers. Surprisingly this is a primary reason why many organizations adopt automation. It gives them a unique selling point and buy-in from the customer.

4.5.5 Documentation and Traceability

The effort subjects put in for documentation is as follows.

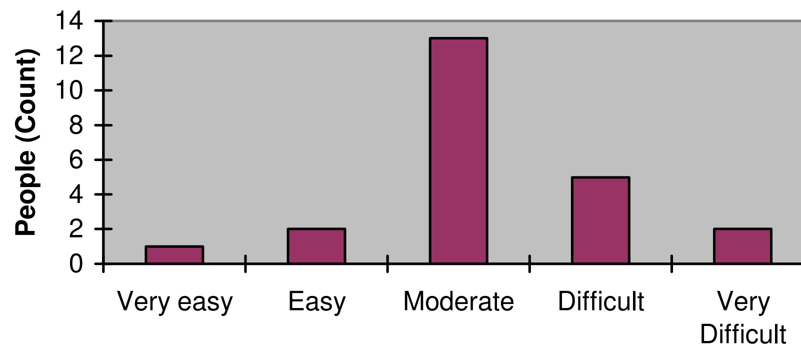


Figure 16: Documentation Effort

Clearly, effort has been put in to document the testing process (Figure 16) and rightly so. Test documentation is vital for measuring test coverage, tracking changes and managing defects. Test automation aids this process by maintaining logs and generating reports. This information may be used to automate documentation and reduce manual effort. Automated test case execution may be tracked and test failures may auto generate defect reports. This can improve traceability.

In this regard, not just test execution but the whole testing process may be automated. Integrated tools [23][24] exist for documentation, defect tracking, requirements management, test coverage, configuration management etc.

4.5.6 Automation – Individual Vs. Group Effort

This experiment involved 24 individual test automation and 24 individual manual testing projects. More than 65 % of the subjects recommended Individual work. The subjects appreciated the hands on experience and felt that the dedicated individual effort ensured better learning. They also noted that groups would enable sharing of ideas. The test cases and testing approach may be discussed in groups whereas the script implementation may be performed individually. This may be considered analogous to a software development project, where coding is an individual effort. Even the pair programming paradigm [33] recommends one person think up the approach/solution and the other implements it. Thinking and execution don't seem to be possible in parallel and hence the pair programming concept.

4.5.7 Test Coverage

Based on a survey on the thoroughness of testing achieved, 44% of the subjects observed that they could test more thoroughly using automation. Automation ensured that they didn't miss any steps during test case execution. Some quoted that the initial tool learning curve, took up time and prevented them from testing thoroughly.

Two inferences can be derived from the above observation. Firstly, automation ensures the test case is executed exactly the same way each time. Sometimes testing gets mundane and humans tend to skip test case steps. This doesn't happen with automation. There is little scope for human error. Automated tools can also be used to track execution of test paths. Some tools can generate all possible program paths based on which the test

cases may be created. Executing the test cases and tracking them using the automated tool helps monitor test coverage.

The results, inferences and lessons learned in this chapter set the stage for further discussion. This information may be used to estimate the behavior of an automation project and how automation in general compares to manual testing. Of course all this may be done keeping in mind the limitations as well as contributions of this experiment towards this research study.

4.6 Tripod SDLC Model

Manual testing and automated testing are quite different from one another and comparing them is like comparing apples with oranges. Although both are based on a common testing paradigm, one is based on a creative process, whereas the other is strongly influenced by the software development life cycle. This made it difficult to come up with a statistical comparison. Choosing one and eliminating the other was not an option. Therefore, for an effective testing setup, it is important to analyze the roles they play and adopt them for the right testing requirements accordingly.

Based on this research study and the experience of others in automation [19], it is clear that the effort required to adopt test automation has been widely underestimated. This is probably why there are so many failed attempts at test automation even though it is quite a useful technique. In this light, I would like to propose a modified V-model software development life cycle, called the tripod model (Figure 17).

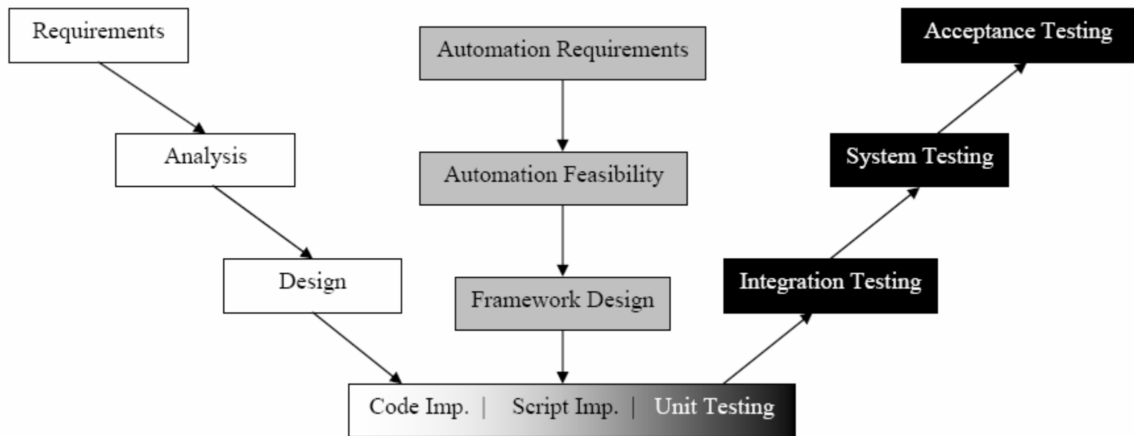


Figure 17: Tripod SDLC Model

The tripod SDLC model is basically a modified V- model that has been customized for software test automation. Here test automation has been treated as a dedicated development path with its own corresponding requirements, analysis and design phases. This is required as the automation effort today can be compared to software development with its own scripts that are implemented, tested, managed, version controlled and maintained with frequent updates and upgrades. The various phases of test automation are discussed below:

4.6.1 Automation Requirements

In this phase, the customer requirements for automation, like smoke tests, user acceptance tests (UAT), etc. are gathered and documented. The training, licensing, and other infrastructure details of automation are taken care of in this phase. The customer business rules are studied to understand how automation can be applied effectively to test the software. This phase lays the foundation for automation in the following phases.

4.6.2 Automation Feasibility

Here, the test cases created as a part of the 'analysis' phase of the software SDLC are inspected to see if they can be automated, given the current system environment, testing methodologies and available testing tool. Schedule constraints and test case priority are also considered. The feasible test cases are short listed for automation.

4.6.3 Framework Design

Based on the test cases short listed in the previous phase and the software design created in the 'design' phase of the software SDLC, an appropriate test framework is chosen for test automation. Depending on the need, it could either be the test script modularity framework, the test library architecture framework, the data driven test framework or the keyword driven architecture framework.

4.7 Contributions

Extended schedules, budget overruns and poor test results are often encountered when dealing with test automation. This continues to happen even when current ROI calculators (Chapter 2) project very high ROI percentages for the test project. This thesis explores the real reasons behind this unfavorable pattern. Understanding the strengths, weaknesses and amount of effort required for automation can go a long way in preventing these failures. The main contribution of this research is the detailed exploration of the three main metrics that would be considered for an ROI study of software test automation. These are schedule, cost and effectiveness. Recommendations based on empirical evidence have been made in the form of equations for these metrics. Using

these equations as guidelines while adopting automation should give a good return on investment.

Other factors that could affect a test automation project have been discussed, even though some of them were subjective in nature. It was more of an attempt to understand them and attain a holistic view.

Finally, a modified V-model SDLC called the Tripod-model has been proposed. It should hopefully cater to current and future software development projects where test automation is becoming an integral part of the development and testing cycle.

4.8 Limitations

Just as it has interesting contributions, this study also has its limitations. Unfortunately none of them were avoidable under the circumstances and scope of this study. They are discussed below with the hope that others doing similar research can avoid them or plan accordingly. They also serve as a check list for things to consider in my future work.

4.8.1 Project Scope

The scope of the project was limited. It had to be reduced when the subjects found it too difficult, especially the subjects who were being introduced to automation for the first time. An alternative solution of extending the schedule was not possible as the study was a part of a course curriculum.

4.8.2 Error Seeding

Again due to schedule considerations, we had to test readily available software instead of developing it ourselves first. This eliminated any chance of error seeding to that would have further helped in determining the effectiveness of automated testing. By choosing relatively simple software for testing, based on considerations like user experience, tool environment support and time, the number of defects turned out be very low. Without a good sample, analysis becomes difficult.

4.8.3 Automation Tool

The research study was limited to the Rational Robot testing tool. Thanks to the IBM Rational SEED program [15], this tool was made available free of cost in support of educational research. Although Robot is a well recognized representative of all testing tools on the market today, without actual comparison with a similar testing tool, some of the biases that are tool specific may not be removed. Testing was also restricted to environments and platforms supported by the tool.

Chapter 5

Conclusion

In conclusion, the goal of studying the major factors affecting test automation, namely schedule, cost, effectiveness and developing metrics for ROI calculations has been met. The detailed discussions on the above factors have been covered in chapter 4. A summary of the study results is provided in the form of recommendations (Table 13). They have been designed to maximize ROI. One may choose either automation or manual testing for a project based on these recommendations. For details please refer to chapter 4.

Table 13: ROI Recommendations

	Automated Testing	Manual Testing
Schedule	▪ Implementation time + Automated execution time < Manual execution time. ▪ {(1/x)* (Automated timeline – Manual timeline)} < {Manual execution time – (Implementation time + Automated execution time)}. ▪ Rapid User Acceptance Testing (UAT).	▪ Already late projects.

Cost	▪ Multiple testing projects to amortize fixed costs. ▪ Average profit margin = $\sum^{T-1}$ (Automated variable cost overflow$_{(i)}$ – Automated variable cost overflow$_{(i+1)}$) / Number of testing cycles (T) -1 ▪ (Fixed costs / Average profit margin) < n*12*Number of projects per month*Number of testing cycles per project.	▪ Low Training budget. ▪ Low Infrastructure budget.
Effectiveness	▪ For Smoke testing, Regression testing, Stress testing and Load testing. ▪ For multi versioned software products. ▪ For Repetitive testing. ▪ Frequently upgraded software. ▪ High testing redundancy.	▪ For Exploratory, Functional testing. ▪ For software services. ▪ For testing with fewer test cycles. ▪ Frequently modified software.

5.1 Future Research

In this study, the effect of the second and consecutive test cycles have been projected based on the behavior of the first. Industrial averages have been used in the calculations wherever data was unavailable. All this is attributed to the reason that the experiment was completed under a tight schedule with no scope for more than one test cycle. The test population was relatively small too. For the sake of reproducibility and verification it would be preferable to perform the study again without the above limitations and the ones mentioned in section 4.8.

Two factors that strongly influenced the cost and schedule during this study were the initial investments made for training and test script implementation. Any positive returns test automation brought was spent in recovering these initial investments. These

investments could have been specific to the tool we used. In the future, it would be a good idea to perform a case study of available test automation tools. Going further, the future of automation tools can be explored based on what the current tools lack. The basic need is to cut down on training and implementation investments and this can be achieved by making tools easier to use and auto-generate test scripts. Certain tools do auto generate test scripts today, but they tend to recognize controls on the screen using screen coordinates and the tools that recognize controls as objects are not very efficient. The underlying problem is that whether you auto-generate your test scripts or write them manually, when the software code changes, the test scripts have to change. So, by deriving a metric that measures software code change between test cycles, ROI calculations can be made more accurate. Alternatively this problem may be handled by exploring testing frameworks that totally eliminate scripting. A step in that direction is the keyword driven testing framework [21].

Finally, this study is based on functional testing. Future work can try and relate these results with testing based on security, stress, load and performance.

References

[1] V.R. Basili, "Software Modeling and Measurement: The Goal Question Metric Paradigm," Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, September 1992.

[2] RTI, "The Economic Impacts of Inadequate Infrastructure for Software Testing", NIST, Planning Report 02-03, May 2002.

[3] The Garner Group, URL: http://www.gartner.com.

[4] N. Fenton, R. Whitty, Y. Iizuka, "Software Quality Assurance and Measurement: A Worldwide Perspective", International Thomson Computer Press, 1995.

[5] P. C. Jorgensen, "Software Testing: A Craftsman's Approach", Second Edition, CRC Press, 2002.

[6] Six Sigma SPC, "Quality Control Dictionary", URL: http://www.sixsigmaspc.com/dictionary/glossary.html

[7] R. Rachlin, "Return On Investment Manual: Tools and Applications for Managing Financial Results", Sharpe Professional, 1997.

[8] B. W. Boehm et al, "Software Cost Estimation with COCOMO II", Prentice Hall PTR, 1st edition, 2000.

[9] J. Barnes, "Creating an ROI assessment for implementing IBM Rational solutions", Rational DeveloperWorks, 15 Oct. 2004, URL: http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/oct04/barnes/

[10] IBM Rational, "Rational Robot Licensing", 2005, URL: https://www-112.ibm.com/software/howtobuy/buyingtools/paexpress/Express?P0=E1&part_number=D53NDLL,D52ZFLL&catalogLocale=en_US&Locale=en_US&country=USA&S_TACT=none&S_CMP=none.

[11] IBM Rational, "Rational Robot Training", 2005, URL: http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=RT511#2.

[12] IBM Rational, "Rational Robot User Guide", Version 2003.06.00, 2003, URL: http://www.rational.com.

[13] Mentora Group, "ROI of an Oracle Apps Repeatable Automated Test Suite", URL: www.mentora.com/library.

[14] IBM Rational, "The Rational Unified Process (RUP)", URL: http://www.rational.com/rup/

[15] D. Hoffman, "Cost Benefits Analysis of Test Automation", STAR West 1999, Oct 1999.

[16] Monster Salary Center, "Median Salary for a Software Quality Analyst", 2005, URL: http://salary.monster.com/.

[17] N. Jayachandran, "Compiled Test Scripts, Test Cases and Defect Reports From The Experiment Conducted For This Thesis", 2005, Email: naveenjc@yahoo.com (Available upon request).

[18] H. L. Dreyfus, "A Phenomenology of Skill Acquisition as the basis for a Merleau-Pontian Non-representationalist", University of California Berkeley.

[19] M. Fewster, D. Graham, "Software Test Automation: Effective Use of Test Execution Tools", Addison-Wesely, 1999.

[20] search390.com, "Definitions – Smoke Test", 2003, URL: http://search390.techtarget.com/sDefinition/0,,sid10_gci930076,00.html

[21] M. Kelly, "Choosing a test automation framework", IBM DeveloperWorks, 18 Jul 2003, URL: http://www-106.ibm.com/developerworks/rational/library/591.html

[22] J. LaRogue, "IBM Rational SEED Program", The Rational Edge, 2004, URL: http://www-128.ibm.com/developerworks/rational/library/2273.html

[23] IBM Rational, "Rational Suite", 2005, URL: http://www-306.ibm.com/software/awdtools/suite/

[24] Mercury, "Mercury TestDirector", 2005, URL: http://www.mercury.com/us/products/quality-center/testdirector/works.html

[25] J. W. Spechler, “Managing Quality in America’s Most Admired Companies”, Berrett-Koehler Publishers, 1993.

[26] Task Committee on Software Evaluation, “Guide for Evaluating Software Engineering”, ASCE, 1989.

[27] B. C. Meyers, P. Oberndorf, “Managing Software Acquisition: Open Systems and COTS Products”, Addison-Wesley, 2001.

[28] R. S. Pressman, “Software Engineering: A Practitioner’s Approach”, Second Edition, Mc-Graw Hill, 1987.

[29] C. Kaner, J. Falk, H. Q. Nguyen, “Testing Computer Software”, Second Edition, Thomson Computer Press, 2001.

[30] S. H. Kan, “Metrics and Models in Software Quality Engineering”, Second Edition, Addison-Wesely, 2004.

[31] G. G. Schulmeyer, J. I. McManus, “Handbook of software Quality Assurance”, Macmillan, 1987.

[32] V. Sikka, “Maximizing ROI on Software Development”, Auerbach Publications, 2005.

[33] L. Williams, R.R.Kessler, W. Cunningham, R. Jeffries, “Strengthening the Case for Pair-Programming”, IEEE Software, 2000.

[34] J. Varghese, “Test Automation – An ROI based Approach”, 3rd Annual International Software Testing Conference, Bangalore, India, 2001.

[35] J. Bach, “Test Automation Snake Oil”, 14th International Conference and Exposition on Testing Computer Software, 1997.

Appendices

Appendix A: Experiment

ISM 6930 – SOFTWARE TESTING SEMINAR
SPRING SEMESTER 2005
TESTING ASSIGNMENT 5 – Due: April 11, 2005
Automated Testing vs. Manual Testing

1. Assignment

Now that you are familiar with manual and automated testing methodologies, this testing assignment will train you to judge when to choose automated testing over manual methods and vice versa. For this assignment, the class will be split into two equal groups (G1, G2), where the group members will be selected according to the alphabetical order of their last names. The groups are:

Group 1 (G1): Bassi, Cao, Chandra, Connor, Douglas, Gibson, Gowda, Guduru, Hadadare, Jordan, King, Magnusson, Marudur Srinivasan, McCart

Group 2 (G2): Morrill, Nemakal, Nidamarthi, Ostapenko, Pasupathy, Rakasi, Senarathne, Shahasane, Srivastava, Tatiparthi, Tello, Warrilow, Yassin, Yussouff

Each group will work on two sub-assignments over a period of two weeks as follows (Fig. 1):

Group	**Sub-Assignments**	
G1	M-1, A-2	(M – Manual, A- Automated)
G2	M-2, A-1	

Therefore, both groups will work on one manual testing sub-assignment and one automated testing sub-assignment each. The above arrangement is used to avoid any testing bias.

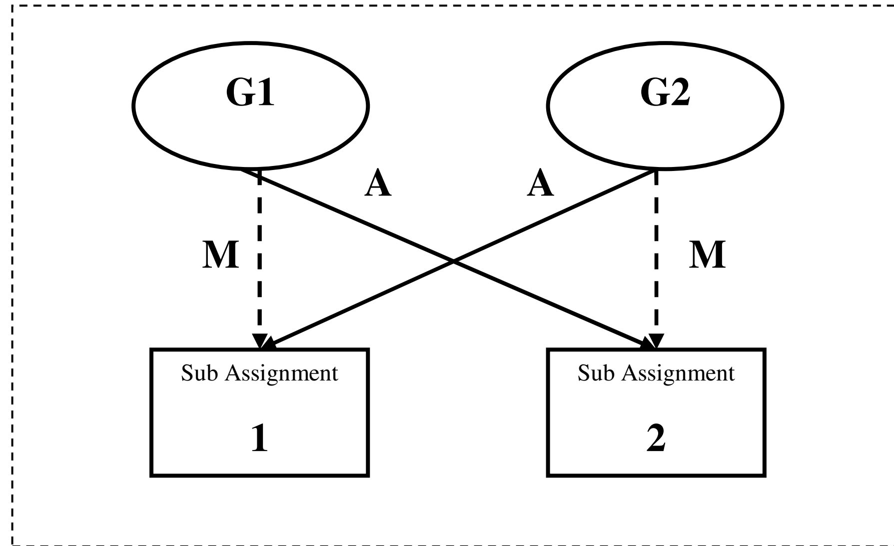


Fig 1: Experimental setup

1.1 Testing Sub-assignment 1 (Print Dialog)

You will be testing the standard windows print dialog. This can be invoked from the menu option: "*File > Print…*" or by using the shortcut "*Ctrl+P*" from the application "*Notepad*" or several other windows based applications. The goal is to come up with a package containing 10 or more test cases that will test the important operations and settings possible using the print utility.

Testing Sub-assignment 2 (File Dialog)

In sub-assignment 2, you will be testing the standard windows file dialog. This can be invoked from the menu option: "*File > Open…*" or by using the shortcut "*Ctrl+O*" from the application "*Notepad*" or several other windows based applications. The goal here too, is to come up with a package containing 10 or more test cases that will test the important operations possible using the File access utility.

Note:

The advantage of the automation exercise is that you may import and reuse these scripts when testing the print or file access functionality of any application in the future.

2. Automation guidelines

2.1 Read through the *Rational Robot: User's Guide* available in the *downloads* section of blackboard.

2.2 Write down the tests you would like to perform on the module as simple one line sentences. You may use any functional testing paradigm to develop your test cases. Remember to be devious in your selection of test cases. (You may later use them to create your test scripts and even add more detail. It is advisable to keep your scripts well documented and maintainable.)

2.3 Use the object recording feature to give you an idea of the functions you can use to perform an operation. Then select the desired function and press 'F1' to learn how to use the selected function in your own script.

3. Deliverables

3.1 You are expected to deliver at least 10 test cases each, for sub-assignments 1 and 2 (A total of 20 or more). Report each test case using Functional Test Case Reporting Forms attached to this assignment.

3.2 You also need to submit the test scripts you create for the above test cases, using the tool, for A-1 or A-2, according to the group you belong to (G2 or G1, respectively). Include comments in your scripts to make it easier to understand and separate one test case from the other.

3.3 Complete Defect Reporting Forms if running a test case results in a failure. Report the defect using the report form attached.

3.4 Finally, you need to submit a completed copy of a survey, which will be provided during the lab session on March 28.

4. Assignment timeline

The following timeline is recommended so that you may complete your assignment comfortably and efficiently.

4.1. Reading phase: (March 7, 2005 to March 20, 2005)

Two weeks to go through the Robot documentation, get used to the Robot editor and create simple scripts. Last date for queries and clarifications on the above is March 20, 2005.

4.2. Working phase: (March 21, 2005 to April 11, 2005)

Three weeks to create test scripts and complete the assignment. Last date for queries and clarifications on the above is April 1, 2005.

4.3. The assignment is due by 6pm on Monday, April 11, 2005. Please submit your deliverables as a single compressed folder, via blackboard.

I will be available to answer your queries regarding this assignment at naveenjc@yahoo.com

5. Lab: Getting started

5.1. Rational license server: ibmlab5

5.2. Common space on the network has been provided so that you create, modify and store your project online. This space has been provided as a hidden network drive. To access this space:

You need to map a network drive. Right click on 'My Computer' and select 'Map Network Drive…' In the window (dialog) that pops up, enter the following information.

Drive: H
Folder: \\bsnlab\hevner$

Note:

1. Remember to uncheck the checkbox 'Reconnect at logon'. If you fail to do this, other students who are not a part of this course can access this drive when they logon using the same computer.

2. Create a new directory within the H: and give it your name. Your working and storage space must be within this directory only. This is to ensure that users do not accidentally overwrite other user's files.

3. When you create a Robot project (will be discussed in class), password protect it, so that other students who can access the common network drive, cannot copy your test scripts.

4. You are personally responsible for your files. Please take a backup of your test script files on a floppy disk, after each Robot session. If you have a flash drive, you may take a backup of the entire project, each time (will not exceed 15 MB).

5. You are given full rights (read, write and execute) on this network drive to enable hassle free working on the assignment. All activities on this network drive are logged.

Appendix B: Functional Test Case Reporting Form

Test Case ID			
Component			
Purpose of Test Case			
Functional Test Type			
Pre-Conditions			
Inputs			
Expected Outputs			
Post-Conditions			
Execution History	**Date**	**Result**	**Tester**

Test Case ID			
Component			
Purpose of Test Case			
Functional Test Type			
Pre-Conditions			
Inputs			
Expected Outputs			
Post-Conditions			
Execution History	**Date**	**Result**	**Tester**

Test Case ID			
Component			
Purpose of Test Case			
Functional Test Type			
Pre-Conditions			
Inputs			
Expected Outputs			
Post-Conditions			
Execution History	**Date**	**Result**	**Tester**

Appendix C: Defect Reporting Form

DEFECT ID:

TITLE:	**COMPONENT:**
REPORTED BY:	**SUBCOMPONENT:**
REPORTED ON:	**VERSION:**
PLATFORM:	**RESOLUTION PRIORITY:**
OPERATING SYSTEM:	**RESOLUTION SEVERITY:**
RELATED TEST CASE IDS:	**CONSISTENCY:**

DEFECT SUMMARY:

STEPS TO REPRODUCE:

COMMENTS:

ATTACHMENTS:

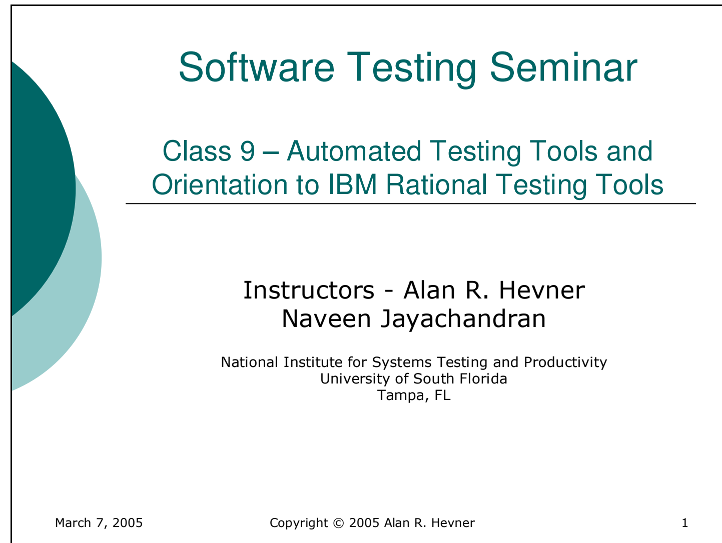
1.
2.

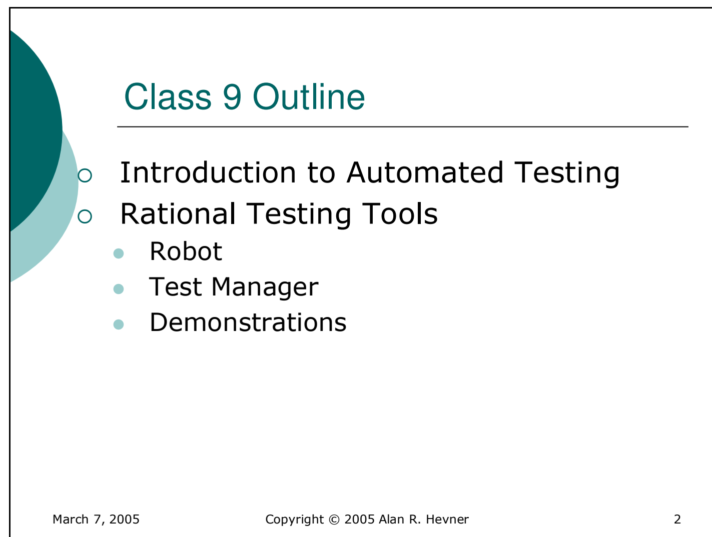
For evaluators use only

DEFECT STATUS: __________APPROVED __________ REJECTED

REASON: ____

Appendix D: Tool Training Presentation Slides





Automated Testing Key Points

- Understand the purpose of Automated Testing
- Use Automated Testing for Unit, Integration, and System Testing
- Test Automation Strategy
  - What will be automated?
  - What resources are needed?
  - How will scripts be developed and maintained?
  - What are the costs and benefits?
- Look on Automated Testing as an investment for future projects.

More Key Points

- Testing tools are very sophisticated and use coding languages
- Do not let the 'tail wag the dog', continue software development best practices
  - Fit tools into software development life cycle
  - Automated testing is not a replacement for Reviews, Inspections, and Walkthroughs
- Developers and Testers must have the correct skill sets for Automated Testing
  - Testers must write test scripts (i.e. code)

Appendix D: (Continued)

Advantages of Automated Testing

- Facilitates regression testing
- Less tedious
- Effective use of time and resources - Can run tests overnight
- Can simulate conditions that are difficult to create in reality

March 7, 2005 Copyright © 2005 Alan R. Hevner 5

Are You Ready for Test Automation?

- Tools are expensive
- Requires at least one trained, *technical* person – in other words, a *programmer*.
- Time consuming - Takes between 3 to 10 times as long (or longer) to develop, verify, and document an automated test case than to create and execute a manual test case.

March 7, 2005 Copyright © 2005 Alan R. Hevner 6

Automated Testing Risks

- Test Automation is Software Development
- Test Automation is a Long-Term Investment
- Assess Your Resources: People and Skills
- No One-Size-Fits-All Approach
  - Starting Large
  - Starting Small
- Gauge Your Maturity Levels

March 7, 2005 Copyright © 2005 Alan R. Hevner 7

Capture/PlayBack

- Cannot rely solely on this feature of the tool
- The scripts resulting from this method contain *hard-coded values* which must change if anything at all changes in the application.
- Trying to maintain dozens or hundreds of capture/playback scripts over time can be a nightmare.
- If the application changes, the test must be re-recorded.
- If the tester makes an error entering data, etc., the test must be re-recorded.

March 7, 2005 Copyright © 2005 Alan R. Hevner 8

Appendix D: (Continued)

Class 9 Discussion Questions

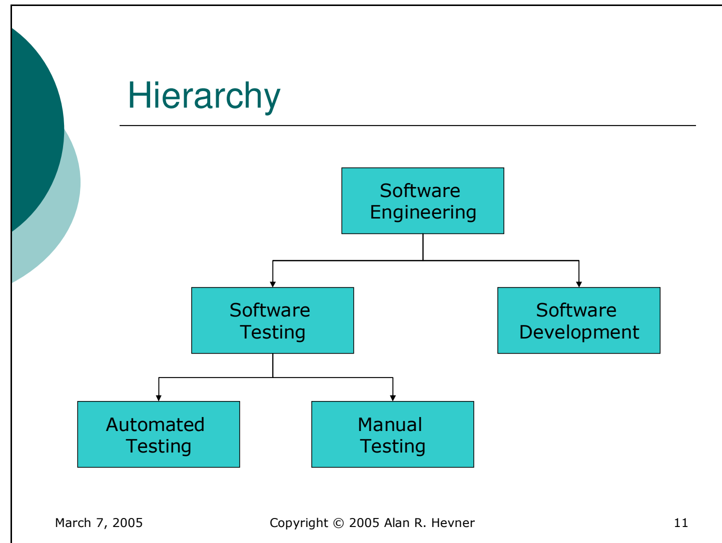
1. How would you as a project manager best use automated testing tools on your project? Give reasons to support your answer.
2. What new features are needed in automated testing tools?
3. How big should a software project be (software size and number of staff) before test automation is justified?

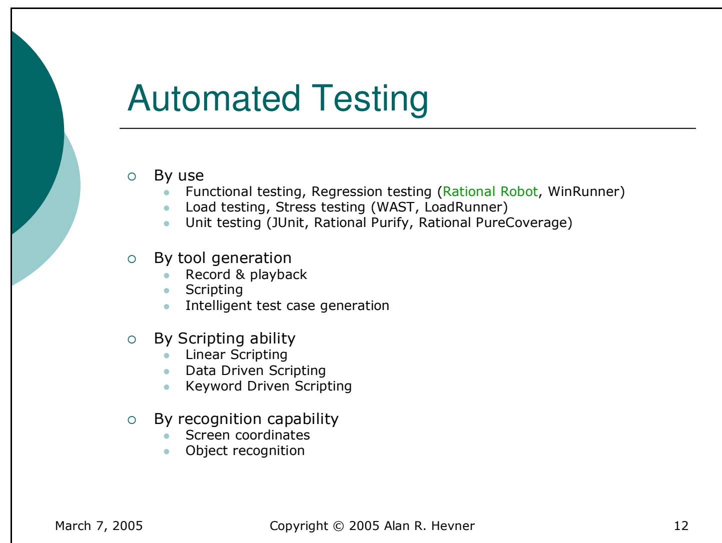
March 7, 2005 Copyright © 2005 Alan R. Hevner 9

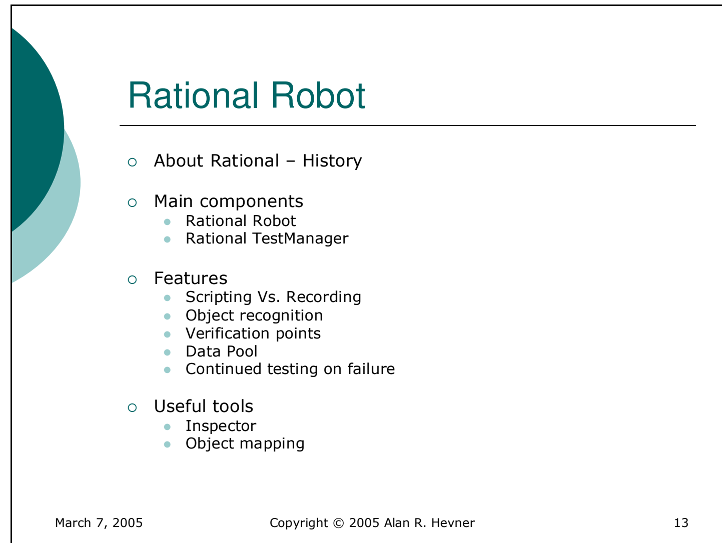
IBM Rational Testing Tools Orientation Outline

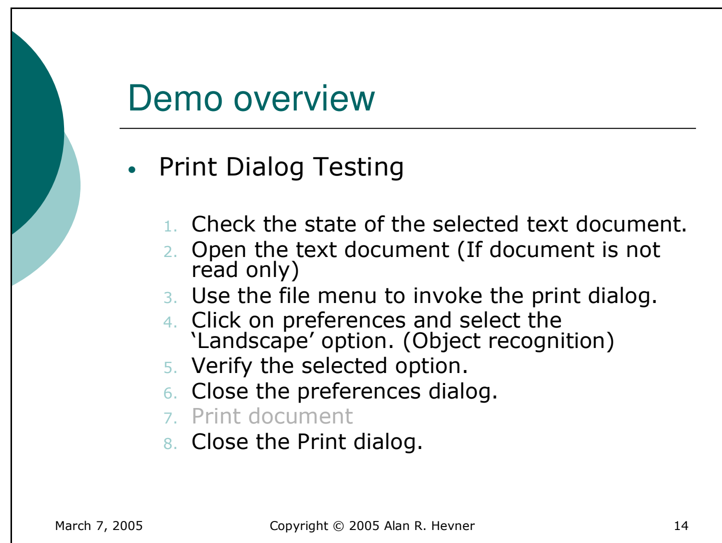
- Hierarchy
- Automated Testing
- Rational Robot
- Demo overview
- Demo
- Testing Assignment 5
- Recommended Reading
- Conclusion
- Lab: Getting started

March 7, 2005 Copyright © 2005 Alan R. Hevner 10


Hierarchy
Software Engineering
Software Testing
Software Development
Automated Testing
Manual Testing
March 7, 2005
Copyright © 2005 Alan R. Hevner
11


Automated Testing
By use
Functional testing, Regression testing (Rational Robot, WinRunner)
Load testing, Stress testing (WAST, LoadRunner)
Unit testing (JUnit, Rational Purify, Rational PureCoverage)
By tool generation
Record & playback
Scripting
Intelligent test case generation
By Scripting ability
Linear Scripting
Data Driven Scripting
Keyword Driven Scripting
By recognition capability
Screen coordinates
Object recognition
March 7, 2005
Copyright © 2005 Alan R. Hevner
12


Rational Robot
About Rational – History
Main components
Rational Robot
Rational TestManager
Features
Scripting Vs. Recording
Object recognition
Verification points
Data Pool
Continued testing on failure
Useful tools
Inspector
Object mapping
March 7, 2005
Copyright © 2005 Alan R. Hevner
13

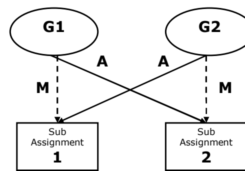

Demo overview
Print Dialog Testing
1. Check the state of the selected text document.
2. Open the text document (If document is not read only)
3. Use the file menu to invoke the print dialog.
4. Click on preferences and select the 'Landscape' option. (Object recognition)
5. Verify the selected option.
6. Close the preferences dialog.
7. Print document
8. Close the Print dialog.
March 7, 2005
Copyright © 2005 Alan R. Hevner
14

Appendix D: (Continued)

Testing Assignment 5

Group	Sub-Assignments
G1	M-1, A-2
G2	M-2, A-1

Where, (M – Manual, A- Automated)



March 7, 2005 Copyright © 2005 Alan R. Hevner 15

Recommended Reading

- H. Robinson, "Intelligent Test Automation," Software Testing and Quality Engineering, September/October 2000. (BlackBoard)
- K. Zallar, "Are You Ready for the Test Automation Game?" Software Testing and Quality Engineering, November/December 2001. (BlackBoard)
- Mark Fewster and Dorothy Graham, "Software Test Automation – Effective use of test execution tools"
- Rational Robot User Guide. (BlackBoard)
- SQABasic Reference. (BlackBoard)

March 7, 2005 Copyright © 2005 Alan R. Hevner 16

Appendix D: (Continued)

The Rational Robot test script for the 'Demo Overview' presentation slide in Appendix D is included below. The slide describes the actions of the test script. This test script was used to train the subjects and gives an idea about the automated test scripts written by the subjects for the experiment.

```
Sub Main
  Dim Result as Integer
  Dim returnValue As Integer
  Dim attrValue as Integer
  Dim msgBoxText as String
  Dim value as String

  On Error Goto ExceptionHandler

  attrValue = Getattr("C:\Documents and
Settings\Nav\Desktop\RobotDemo.txt")
  SQAConsoleWrite "File Attribute is: " & attrValue

  If attrValue = 0 then
      SQAShellExecute "C:\Documents and
Settings\Nav\Desktop\RobotDemo.txt","",""
      Window SetContext, "Caption={*Notepad}", ""
      MenuSelect "File->Print..."
      Window SetContext, "Caption={*Print}", ""
      PushButton Click, "Text=Preferences"
      Window SetContext, "Caption={*Preferences}", ""
      RadioButton Click, "ObjectIndex=2"
      Result =
SQAGetPropertyAsString("Type=RadioButton;ObjectIndex=2",
"State", value)
      SQAConsoleWrite "State is: " & value
      Window CloseWin, "Caption={*Preferences}", ""
      Window CloseWin, "Caption={*Print}", ""
      Window CloseWin, "Caption={*Notepad}", ""
      Msgbox "Test Run Complete"
      Exit Sub
  Else
      Exit Sub
  End if

ExceptionHandler:
    msgBoxText="Error code: " & Err & " Error line: " & Erl
    MsgBox msgBoxText
```

Appendix E: Testing Tool Lab Training 1

ISM 6930 – SOFTWARE TESTING SEMINAR
SPRING SEMESTER 2005
TESTING ASSIGNMENT 5
Rational Robot Exercise 1 – March 21, 2005

This exercise has been designed to get you started with the basic Robot commands. The functionality of most of these commands is evident from the command name itself. Your task is to create a test script and explore the commands listed below.

Getting Started

1. Make sure the H: drive is mapped to \\bsnlab\hevner$
2. Open Rational Robot. (Start > All Programs > Rational Software > Rational Robot
3. If you can't find your project in the drop down list, then you need to register it using Rational Administrator (Open Rational Administrator and use the option 'Register Project'. If you are unsure, ask me how).
4. Once you are within Rational Robot, with your project open, you need to create a new test script (File > New > Script…).
5. In the script editor that opens up, you can type in the commands given below. Place your cursor on the command you are interested and press F1. A comprehensive help dialog opens up. The help dialog explains the command syntax and usage. It usually also has example scripts that you can refer to.

Explore and enjoy!

Command List

1. Dim (Also read about the various data types)
2. Window *SetContext*
3. PushButton *Click*
4. InputKeys
5. ComboEditBox
6. ComboBox
7. ComboEditBox
8. ListBox
9. CheckBox
10. MsgBox
11. DelayFor
12. SQAGetProperty

Appendix E: (Continued)

Note

1. To explore controls that you do not already know the details of, use the Inspector tool (Tools > Inspector…). After invoking the tool, you might have to wait for a few seconds for it to evaluate all screen objects and initialize. After it comes up, you can click on the hand icon at the lower right hand corner of the tool, drag and drop it on the control you wish to use. The tool will provide you with all the details you need about the control.
2. You may also use the Record and Playback tool (discussed in the previous class – March 7) to get an idea of how the commands are used.

Tip

1. When looking up help on a command, click on the link 'See Also' on the top left hand corner to explore related commands.

Appendix F: Testing Tool Lab Training 2

ISM 6930 – SOFTWARE TESTING SEMINAR
SPRING SEMESTER 2005
TESTING ASSIGNMENT 5
Rational Robot Exercise 2 – March 28, 2005

The goal of exercise 2 is to use some of the commands you learned in exercise 1 to convert your TA5 test cases to automated scripts. Below you are provided with two sample scripts one for each group (G1, G2), to get you started. So, today you will,

- Create, execute and understand the two sample scripts.
- Think of similar test cases for TA5 and try to automate them based on what you have learned.

1. G1: File Dialog Testing (A2)

1.1. Sample Test Case

Open a text file by specifying a path in the file dialog.

1.2. Sample Test Script

```
Sub Main
SQAShellExecute “C:\windows\notepad.exe”,””,””
InputKeys “^{o}”
Window SetContext, “Caption=Open”, “”
ComboEditBox Click, “ObjectIndex=3”, “”
InputKeys “{bksp 5}”
InputKeys “C:\<enter filepath here>”
PushButton Click, “Text=Open”
End Sub
```

1.3. Note

1.3.1. Make sure you create a text file and use the correct path in the script.
1.3.2. Try and modify the above script so that it automatically closes the text file to avoid having to manually do it each time.

2. G2: Print Dialog Testing (A1)

2.1. Sample Test Case

Print a text file using the print dialog.

Appendix F: (Continued)

2.2. Sample Test Script

```
Sub Main
SQAShellExecute “C:\<enter filepath here>”,””,””
Window SetContext, “Caption={*Notepad}”, “”
MenuSelect “File->Print...”
Window SetContext, “Caption={*Print}”, “”
PushButton Click, “Text=Cancel”
Window SetContext, “Caption={*Notepad}”, “”
MenuSelect “File->Exit”
End Sub
```

2.3. Note

2.3.1. Make sure you create a text file and use the correct path in the script.

2.3.2. For trial purposes, the script clicks on the ‘Cancel’ button, instead of the ‘Print’ button.

3. Executing the scripts

To execute the script, click on the ‘Go’ button on the toolbar (14th from the left; has a play icon and horizontal lines on it). You may even use the shortcut ‘F5’.

Executing the script also compiles it if it hasn’t been done before. However, if you want to just compile your script, you may do so by clicking on the ‘Compile’ button (13th from the left; has a green tick mark on it) or by using the shortcut ‘Ctrl+F7’. The compile results (warnings and errors, if any) are displayed in the build tab at the bottom of the screen.

Once the script completes execution, the test manager window opens up to display the log i.e. the test results (pass/fail). Close the log file before executing the script again.

To stop script execution at any point, use the shortcut ‘F11’.

Reminder: The assignment is due on April 11, 2005. Check TA5 document for submission details. The coding phase begins today. Make sure you are clear with the basics before the end of class.

Appendix G: Survey

Informed Consent

Social and Behavioral Sciences
University of South Florida

Information for People Who Take Part in Research Studies

The following information is being presented to help you decide whether or not you want to take part in a minimal risk research study. Please read this carefully. If you do not understand anything, ask the person in charge of the study.

Title of Study: Test Automation ROI Metric

Principal Investigator: Naveen Jayachandran

Study Location(s): You are being asked to participate because you are enrolled in the course CIS/ISM 6930. The research study is based on one of the assignments you will be completing as part of regular coursework.

1.1 General Information about the Research Study

The purpose of this research study is to develop a test automation ROI (Return On Investment) metric. For this, a test automation project (Assignment 5) completed by you as part of your routine coursework will be monitored and studied. In addition, the data provided by you via a survey after completing the assignment will be used for research purposes.

1.2 Plan of Study

You will be doing your coursework (Assignment 5) as usual. At the end of your assignment you will be asked to complete a survey form based on your experiences. This data will be used for the research study.

1.3 Payment for Participation

You will not be paid for your participation in this study.

1.4 Benefits of Being a Part of this Research Study

By taking part in this research study, you should learn the effective and appropriate use of test automation tools and manual testing methods. It will also provide you with a sound basis to judge the testing methodology you would like to adopt for future projects.

1.5 Risks of Being a Part of this Research Study

There are no known risks incurred by being a part of this study.

1.6 Confidentiality of Your Records

Your privacy and research records will be kept confidential to the extent of the law. Authorized research personnel, employees of the Department of Health and Human Services, the USF Institutional Review Board, its staff, and other individuals acting on behalf of USF, may inspect the records from this research project.

The results of this study may be published. However, the data obtained from you will be combined with data from others in the publication. The published results will not include your name or any other information that would personally identify you in any way. Only the principal investigator and the members of this research study will have access to the data. The data will be stored at a secure location.

1.7 Volunteering to Be Part of this Research Study

Your decision to participate in this research study is completely voluntary. You are free to participate in this research study or to withdraw at any time. There will be no grade penalty or loss of benefits you are entitled to receive, if you stop taking part in the study.

1.8 Questions and Contacts

- If you have any questions about this research study, you may contact Naveen Jayachandran at naveenjc@yahoo.com ; (813) 298 9504 or Dr. Dewey Rundus at rundus@csee.usf.edu or Dr. Alan Hevner at ahevner@coba.usf.edu.
- If you have questions about your rights as a person who is taking part in a research study, you may contact the Division of Research Compliance of the University of South Florida at (813) 974-5638.

Appendix G: (Continued)

ISM 6930 – SOFTWARE TESTING SEMINAR
SPRING SEMESTER 2005
TESTING ASSIGNMENT 5
Learning Experience Survey

Age:		Sex (M/F):	
Group (G1/G2):		Sub Assignments (M1,A2/M2,A1)	

Section A

1. Do you have prior programming experience? If yes please specify (In years and months).

Ans.

2. Do you have prior software testing experience? If yes please specify (In years and months).

Ans.

3. Have you taken a software testing course before the current course? If yes, please specify the course and when and where you took it.

Ans.

4. Do you have prior experience in the field of software test automation? If yes please specify (In years and months)

Ans.

5. Have you used Rational Robot before? (Yes/No)
    a. If yes, describe your experience.

Ans.

Appendix G: (Continued)

6. Have you used any other test automation tool (E.g. SilkTest, WinRunner, etc.)? If yes, please specify the tool and describe your experience with it

Ans.

7. How would you rate the effectiveness of test automation on a scale from 1 (poor) to 5 (excellent)?

Ans.

8. How would you rate your learning experience with Rational Robot on a scale from 1 (difficult) to 5 (easy)?

Ans.

9. How would you rate Rational Robot on usability on a scale from 1 (very easy to use) to 5 (very difficult to use?

Ans.

10. Do you think test automation is more productive than manual testing? (Please explain why or why not?)

Ans.

11. How difficult did you find writing test scripts on a scale from 1 (very easy) to 5 (very difficult)?

Ans.

12. How many defects did you find during automated testing?

Ans.

13. How many defects did you find during manual testing?

Ans.

14. Did automation help you conduct consistent repeatable testing? (Yes/No)

Ans.

15. Did automation help you find regression bugs? (Yes/No)

Ans.

16. Did automation help you run more tests? (Yes/No)

Ans.

17. Did automation help you test more thoroughly? (Yes/No)

Ans.

18. Did automation help you increase your confidence in the software? (Yes/No)

Ans.

19. Did automation help you complete your testing faster? (Yes/No)

Ans.

Section B

1. How many person hours did it take you to learn the test automation tool sufficiently to do the assignment?

Ans.

2. How many person hours did it take you to write the required test scripts for the assignment?

Ans.

3. For a smaller project, would you recommend manual or automated testing? Why?

Ans.

4. For a larger project, would you recommend manual or automated testing? Why?

Ans.

5. How big would a project need to be to use automated testing in a cost effective way?

Ans.

Section C

1. How many LLOC (Logical Lines Of Code) did you write in your test scripts? (Consider multi-line IF statements to be a single LLOC, etc.)

Ans.

2. How many individual test cases did you create in automated testing?

Ans.

3. How many individual test cases did you create in manual testing?

Ans.

4. How many times did you run the automation scripts?

Ans.

5. Did automation help you reduce your testing effort? (Yes/No)

Ans.

6. How much effort did you spend to document your test cases on a scale of 1 (low) to 5 (high)?

Ans.

Section D

1. What could have made the learning experience better?

Ans.

Appendix G: (Continued)

2. How much do you think prior software testing experience affected your performance on a scale of 1 (little) to 5 (greatly)? If you had no prior testing experience, answer 'N/A'.

Ans.

3. How much do you think prior programming experience affected your performance on a scale of 1 (little) to 5 (greatly)? If you had no prior programming experience, answer 'N/A'.

Ans.

4. How much do you think prior automated testing tool experience affected your performance on a scale of 1 (little) to 5 (greatly)? If you had no prior programming experience, answer 'N/A'.

Ans.

5. How much did you learn about automated testing tools in this assignment on a scale from 1 (very little) to 5 (a great deal)?

Ans.

6. Would you recommend that this assignment be performed by individuals or small groups? Why?

Ans.

7. What recommendations do you have to improve this assignment?

Ans.

About the Author

The author is a Certified Software Quality Analyst (CSQA) and a Master's student at the University of South Florida with research interests in the field of software engineering, software testing, test automation and quality assurance (QA). The Author has prior industry experience in software testing and QA. Currently, as a Graduate Assistant, he tutors graduate students in software testing and test automation tools. Updated information on his current research activities and contact details can be obtained at www.smartsoftwaretesting.com.