A search-based approach for cost-effective software test automation decision support and an industrial case study

Amannejad, Y., Garousi, V., Irving, R., & Sahaf, Z. (2014). A search-based approach for cost-effective software test automation decision support and an industrial case study. In *Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014* (pp. 302-311). [6825677] IEEE Computer Society. https://doi.org/10.1109/ICSTW.2014.34

Published in:
Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
Copyright 2014 IEEE. This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Download date:25. Aug. 2022

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

A Search-based Approach for Cost-Effective Software Test Automation Decision Support and an Industrial Case Study

Yasaman Amannejad[1], Vahid Garousi[1, 2], Rob Irving[3], Zahra Sahaf[1]

1: Software Quality Engineering Research Group (SoftQual)
Department of Electrical and Computer Engineering
University of Calgary, Calgary, Canada
{yasaman.amannejad, vgarousi}@ucalgary.ca

2: System and Software Quality Engineering Research Group (SySoQual)
Department of Software Engineering
Atilim University, Incek, Ankara, Turkey
vahid.garousi@atilim.edu.tr

3: Pason Systems Corporation
Calgary, Canada
rob.irving@pason.com

Abstract*—Test automation is a widely-used approach to reduce the cost of manual software testing. However, if it is not planned or conducted properly, automated testing would not necessarily be more cost effective than manual testing. Deciding what parts of a given System Under Test (SUT) should be tested in an automated fashion and what parts should remain manual is a frequently-asked and challenging question for practitioner testers. In this study, we propose a search-based approach for deciding what parts of a given SUT should be tested automatically to gain the highest Return On Investment (ROI). This work is the first systematic approach for this problem, and significance of our approach is that it considers automation in the entire testing process (i.e., from test-case design, to test scripting, to test execution, and test-result evaluation). The proposed approach has been applied in an industrial setting in the context of a software product used in the oil and gas industry in Canada. Among the results of the case study is that, when planned and conducted properly using our decision-support approach, test automation provides the highest ROI. In this study, we show that if automation decision is taken effectively, test-case design, test execution, and test evaluation can result in about 307%, 675%, and 41% ROI in 10 rounds of using automated test suites.

Keywords*—action research, cost-benefit analysis, industrial case study, search-based software engineering, software test automation.

I. INTRODUCTION

Software testing is a major cost factor in the software development life-cycle [1, 2]. Test activities are essential and inevitable to guarantee high quality software products. Test automation is a widely-used approach to reduce the cost of manual testing, while ensuring the quality of software systems.

However, test automation is not always necessarily cost effective. Deciding which parts of a given System Under Test (SUT) should be tested in an automated fashion is a widely-asked and challenging question for practitioner testers [3]. A typical software tester may naively argue that manual testing can be fully replaced with automation, but it could be the case that such a replacement is not cost-effective. On importance of carefully choosing what to automate, Rice et al. [4] state that: "If you do not know which tests are the most important and which tests are the most applicable for automation, the tool will only help perform a bad test faster." Therefore, selecting which parts of the system should be automated and which parts should remain manual is an important decision.

There are a number of studies (e.g., [3-5]) which have tackled the challenges of test automation, but there have been no work focusing on the notion of automation "across" the entire software testing process, from test-case design, to test-case scripting, to test execution, and test-result evaluation. The focus of previous works in the field of software test automation has only been on test execution.

When automation decision should be taken for all features, use cases or parts of a SUT throughout the entire testing process, estimating the Return On Investment (ROI) of automation would not be trivial. Use-cases may impact each other, e.g., if we automate the testing of a use-case, testing others may become easier and/or cheaper. Therefore, ROI of the selected parts of the system for automation should be considered as a whole. In large projects when the number of use cases is large, an effective solution should be applied to find the best solution in a reasonable computational cost. This paper proposes a search-based approach based on Genetic Algorithm (GA) for deciding what parts of the systems should be automated to gain the highest ROI of testing activities.

The main contributions of this study are:

- A consolidated review of test automation across the entire software testing process, not just in test execution (Section 0)
- A search-based approach to address the problem of what (parts of the system) should be automated (Section IV)
- An industrial case study to investigate the applicability and effectiveness of the approach in a real-world industrial setting (Section V)

The remainder of this paper is organized as follows. Section II discusses background and related works. Section 0 revisits the notion of test automation across the entire software testing process. Section IV formulates the cost-benefit of the effective test automation problem as an optimization problem and presents a GA to solve it. Section V describes the industrial case study conducted to evaluate the approach. Finally, Section VI concludes this study and states the future work directions.

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

II. Background and Related Works

This study is in the context of effective software test automation and relates to the following areas: (1) What to automate (for the purpose of testing), (2) Cost, benefit, and ROI of testing, and (3) Action-research and industrial case studies in software testing. The related works in each area are briefly discussed next.

A. What to Automate

When deciding what parts of the system should (not) be automated, several factors need to be considered. Practitioner testers from Microsoft recommended three major factors to be considered [3]: "(1) rate of change of what we are testing: the less stable, the more automation maintenance costs, (2) frequency of test execution: How important is each test result and how expensive is to get it?, and (3) usefulness of automation: Do automated tests have continuing value to either find bugs or to prove important aspects about your software, like scenarios?"

Inside a book on testing [6], generic tips to address the "what to automate" problem were provided. According to this book, certain types of testing such as stress, reliability, and regression testing are amenable types for being automated. Due to the repetitive nature of the regression testing, automation can save significant time and effort and the gained time can be effectively utilized for ad-hoc testing and other more creative avenues [6]. Similar to the challenge in our context, finding the right combination of test cases to be automated, [6] suggested that while starting automation, the effort should focus on areas where good scenarios in terms of ROI exist [6], however no systematic approach was provided.

A comprehensive checklist was provided by another book on test automation [7], which has considered almost all the above mentioned factors. Number of executions, covering critical paths, error-prone areas, data-driven features, number of supported hardware and software, and also having promising ROI are the main factors discussed in this book [7]. Our work is not limited to just providing the cost and benefit factors. Using these factors, we have designed and implemented a decision-support approach for addressing the "what to automate" problem. In addition to the ROI of test-suite development, maintenance, and evaluation which were considered in [7], our approach takes into account the entire testing process including test-case design, test execution, and test-result evaluation.

Surveying the books and literature, we found some textbooks that are discussed. Also, we found that there is lack of relevant papers that mainly focused on the problem of deciding which part of a software should be automated and which parts are better to remain manual. This can show that this area of research needs more attention from practitioners and researchers.

B. Quantifying Cost, Benefit, and ROI of Testing

Thinking of increasing automated test cases without considering costs and benefits of each test case, are the main reasons of failure in test automation projects []. If cost of maintaining a test case is more than the cost saving that will be provided, it should not be considered for being automated. It is worth mentioning that, even if a company spend a lot of set up costs for its automating its test cases, it is still a naive decision to automate all test cases. In order to have a successful test automation solution, precise investigation of cost and benefit, and ROI of each test case is necessary before deciding to automate them.

Many of the works in decision-support for test automation take into account the ROI of testing [1, 7, 8]. In this context, estimating the potential costs and benefits is critical part of articulating the impact of implementing automated software testing [7].

Section 3.3 of the book entitled "Implementing Automated Software Testing" [7] described how to calculate the potential cost savings of implementing automated software testing in comparison to performing manual tests. Tangible factors discussed in [7] for calculating ROI are: test setup, test development, and test evaluation costs.

Reference [8] was an industrial case study which explored state of test automation in software industry. Cost and benefit drivers of test automation were introduced in this paper. According to [8], quality improvement, time saving, and reusing test-ware were benefits of test automation while test-code development and maintenance, and staff training, are among the major cost drivers of test automation. Similar to this study, our industrial case study confirms that time saving and reusing test-ware are benefits of test automation.

A model named "opportunity cost model" was utilized in [1] to present influencing factors on test automation. Fixed and variable costs involved in manual and automated test execution were considered in the opportunity cost model. Risk exposure was defined as benefit of test execution and objective of the project was to maximize the benefit of test process. This model intended to make the model realistic while keeping it simple enough to be used in real-world projects. Although we agree that the model should be simple enough to use but we believe that factors such as test-code maintenance including quality improvement [9] and co-maintenance [10] activities, and reusability of test-ware are important factors that needs to be considered while calculating cost and ROI of test automation.

C. Search-Based Software Enginnering (SBSE)

SBSE approaches have been previously applied various aspects of software testing, such as test data, and test case generation [11, 12]. Different from these works, this paper does not use SBSE to find test suites, or generate test data. Instead, our work uses a search-based technique to facilitate decision making for test automation purposes. Although there are other applications of SBSE in other area of software testing within the literature, the novelty of the work is in its application of SBSE for the particular problem of decision-support for test automation. SBSE is

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

used for identifying parts of the test process to be automated. There were no previous works in this direction.

D. Action Research and Industrial Case Studies in Software Testing

In the context of software engineering research methods, the research approaches we use in this work are "exploratory and improving" case studies [13] and Action Research (AR) [14].

Our approach was initiated, developed and evaluated through an AR project between academia and industry. We have had experience with the AR approach in two recent industrial testing projects [15, 16] and used our experience to initiate, plan and execute this study. The details of the project and case description will be described in Section V. In the "exploratory" phase, we intended to find out and characterize "what is happening" in the existing testing activities and processes of the company under study. In the "improving" phase, we improved the cost effectiveness of software test activities. The entire research project was governed by AR which is an established research methodology for industry-academia collaborations in software engineering. In particular, we followed the AR guidelines proposed by [14].

III. Test Automation Across the Software Testing Process

By consulting several books and online resources [17-19] on software testing and incorporating different views and classifications, we divide the testing tasks into four types:

1. Test-case design: Designating the list of test cases or test requirements to satisfy coverage criteria, or other engineering goals.
2. Test scripting: Documenting test cases in manual test scripts or automated test code.
3. Test execution: Running test cases on the software under test and recording the outputs.
4. Test evaluation: Evaluating results of testing (pass or fail), also known as test oracle or test verdict.

Each of the four activities can be done either manually (by a human), automated (using a software tool) or the mix of the two. The following four sub-sections discuss each of the four activities in detail, and discuss the notion of manual work, partial automation, or full automation in each activity. Note that this entire section is a consolidated overview of the topic and will include references to selected list of existing works in each of the above four testing activities.

A. Automation in Test-case Design

Test-case design is the activity of designating a list of test cases or test requirements. Sub-activities such as identifying test data, including test inputs and expected test outputs, and test paths are part of test design activity. This activity can be done either: (1) based on criteria (e.g., line or requirements coverage), (2) based on human knowledge (e.g., exploratory testing) [19].

To conduct criteria-based test-case design in a fully automated fashion, there is also a large body of knowledge available which is often referred to as software test-data generation. Test data generation, including inputs and expected outputs, is one of the important part of test design activity. The paper by McMinn [17] surveys different techniques and tools for test data generation. Also, to automate combinatorial criteria-based test approaches such as pair-wise testing [19], there are open-source and commercial tools which generate test inputs [20].

Test-case design based on human knowledge is mostly referred to as exploratory testing. As per the nature of this type of testing, it is done fully manual. Criteria-based test-case design can be done manually, automated, or a mix of the two. In fully manual criteria-based test-case design, depending on whether black- or white-box testing is going to be done, tester looks at the requirements or code and derives the test cases without using any tool.

For partial automation, tester would use any of the many-available code coverage tools [18], but has to conduct manual efforts to derive test cases to increase coverage.

The output of test-case design activity is a test suite which is a set of test cases (input and expected output values) or test requirements (e.g., control flow paths to be covered). The next test activity (test scripting) will make use of this output.

B. Automation in Test Scripting

In manual testing, test scripting is documenting test cases, generated in the previous activity, either in simple technologies such as Word file or Wiki's or using test-support tools, e.g., IBM Rational Manual Tester. In automated testing, test scripting is developing test code (scripts) from test cases generated in the previous activity. There are many test tools and frameworks available, e.g., JUnit [21] and Selenium [22].

Test scripting can also be done manually, partially automated, or fully automated. Manual development of test scripts is obvious, as developing manual test cases is the usual and the basic step that is done by most manual testers. For partial automation in this context, manual test-support tools such as IBM Rational Manual Tester are used. Also, there are many tools for testing via Graphical User Interfaces (GUI) which are mostly called "record-and-playback" tools, e.g., HP QuickTest Professional (QTP). Using these tools, testers records a test scenario (test case) by interacting with the SUT's GUI, while the tool automatically generates the test script in the background. When recording (test scripting) is done, the automated test code can be executed as needed.

Finally, for fully automated test scripting, there has been a good progress in the last decade and many tools for automated generation of automated test code are now available [5, 16, 23].

The outputs of test scripting activity are either: (1) manual test scripts (in variety of formats), or (2) automated

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

test suites (e.g., using tools such as JUnit, HP QTP). The next test activity (test execution) will make use of test scripts.

C. Automation in Test Execution

Test execution is defined as running the test cases on the SUT and recording the results or observing the SUT's output/behavior [19]. We have observed in our interactions with our industrial partners that when most practitioners discuss about test automation, they often limit their focus on automation of test execution only.

Similar to the above activities, test execution can also be done manually, partially automated, or fully automated. We can observe at this point that there are inter-dependencies among the choices (decisions) made in following activities. For example, if a tester chooses to develop all the tests as automated test suites, test execution will obviously be done fully automated. In contrast, if the test team decides to develop all their tests as manual test scripts, test execution has to be done fully manual. Partial automation of test execution will be in the case that a ratio of test suites is automated scripts and the others are manual test scripts.

D. Automation in Test Evaluation (Test Oracles)

After the SUT is exercised by a test case, evaluating the test outcome (pass or fail) is an important phase. There are usually three approaches for this: (1) a human tester may make such a judgment, (2) we may decide to incorporate ("hard code") test evaluations as a verification points (or known as assertions) in test-code, and (3) there have also been more advanced techniques to build "intelligent" (learning) test oracles using machine learning and artificial intelligence, e.g., [24, 25]. The approaches (2) and (3) are considered to have automated test evaluation, but with different levels of intelligence.

E. Assumptions

Each use case or object-oriented unit under test can have several test cases. Although it is possible to make fine grain decisions in test case level, in this paper that, we make decision in use case level. In fact, we have assumed that when using our approach, all test cases of a use-case will either be decided to be automated or done manually. This assumption helps us reduce the complexity of our decision support approach and make it applicable to be used in real world scenarios.

IV. Decision Support for Effective Test Automation as an Optimization Problem

In this section, we formulate the effective test automation as an optimization problem and develop a solution approach using GA.

A. Multiple Decisions for Test Automation and a Motivating Example

We start with an example hypothetical test-automation scenario. Let us assume that we intend to conduct black-box testing of a SUT with a GUI. Assume that the SUT has *n* use-cases: $UC_1, UC_2 \ldots, UC_n$. For each of these use cases, the above mentioned test activities (discussed in Section 0) could be done manually or automated. One possible combination of test automation approach for this SUT is shown in Table I. We refer to the matrix notion shown in Table I as a Test-Automation Decision Matrix (TADM), where each matrix cell is a binary value (0 or 1). Note that as we discussed in Section 0, some of the four test activities could also be done in a partially automated manner. However, as the initial version of our approach, we only include either complete manual or complete automated testing combinations in the rest of this article.

By reviewing the TADM in Table I, we can easily see that there could be a large number of combinations of possible choices in this context. In cases that there is a fixed budget for all test activities, which is the usual situation in practice, decision should be taken considering all activities together. Also, selection of use cases for automation can impact each other. Therefore, the number of possible choices is dependent on the number of use cases and the number of test activities. There are $2^{|\#of\ use\ casse|\times|\#\ of\ test\ activities|}$ different combinations that should be considered to find the most optimal (cost effective) combination.

For example, let us assume that we have three use cases and the four test activities. In this case, there will be 2^{3x4}=4,096 different combinations of manual/automated choices in this matrix. Each of those combinations will be one possible decision for test automation in this context, and will lead to a different ROI at the end for the entire testing process. If any of the "micro" decisions, e.g., whether to automate test execution for UC_1 is made improperly, large amounts of rework and test-code "repair" (co-maintenance) costs [10, 26] will be potentially incurred to the project and will negatively impact the cost-benefit of testing.

TABLE I. A Test Automation Decision Matrix (TADM)

Use-case	Test design	Test scripting	Test execution	Test evaluation
UC_1	1[a]	0	0	0
UC_2	1	1	1	0
UC_3	1	0	0	1
….	…	…	…	…
UC_n	0	1	1	1

[a] 1 = Automated, 0 = Manual

B. Formulation of the Optimization Problem

The objective of the optimization problem is to find a single combination among all possible combinations of the search space (all possible values in a TADM) that would yield the highest ROI.

Deciding what to automate in the context of cost-effective test automation is thus a combinatorial optimization problem. Given the influencing factors in each phase of test automation, TADM matrix representing a candidate solution containing binary choices for automating each use case in each phase of software testing process. In selection of use cases for automation, possible

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

number of usage of the test-ware during the life of the SUT is important. Based on the estimated number of usage of test artifacts during the life-span of the software product, use cases for automation are selected by the proposed solution in a way that the highest ROI is achieved before retirement of the application and its test suites.

The ROI is usually calculated by net profit divided by the investment. We calculate the ROI of test automation in comparison to manual testing, as in Equation (1).

$$(1) ROI = \frac{\Delta\, Benefit\ (automation\ over\ manual)}{\Delta\, Cost\ (automation\ over\ manual)}$$

The goal is to maximize the ROI value. Each decision in TADM has its costs and benefits. Total cost and benefit of all automation decisions can be calculated using Equations (2), (3), respectively.

$$(2) \Delta Cost = \sum_{i=1}^{n}\sum_{j=1}^{m} TADM_{i,j}.COST_{i,j}$$

$$\Delta Benefit = \sum_{i=1}^{n}\sum_{j=1}^{m} TADM_{i,j}.BNFT_{i,j} \qquad (3)$$

where *COST* and *BNFT* are two matrices representing the cost and benefits of automating use cases in testing activities during the life-span of the SUT. TADM, COST, and BNFT are $n \times m$ matrices, where n is the number of use cases, and m is the number of test activities which equals to 4.

C. Solution Approach and Design of the GA

The goal of the optimization is to maximize the ROI value. Due to the nature of the *COST* and *BNFT* matrices, the problem is non-linear. Looking back to the TADM example provided in Table I, we can easily see that there could be a large number of combinations of possible choices. Brute force, as a simple search technique systematically enumerates all possible candidates for the solution. In most real contexts, the number of use cases is large enough to make brute force an inefficient technique for this problem.

To deal with the large search space, meta-heuristic search techniques are usually used in other software engineering problem domains, referred to as SBSE. We thus select GA as our solution approach.

We designed a GA in which the initial population is a set of randomly-generated individuals (chromosomes). Each individual represents a TADM matrix. Each cell of a TADM matrix (a binary decision for automation) corresponds to a gene (genome) in our GA. In each iteration of the GA, the algorithm selects the parents using the fitness function to produce the next generation. The ROI of test automation serves as the fitness function. Crossover and mutation are two operators of the GA that occur based on defined probabilities. Crossover operator aims at passing desirable genes to the next generation and creates new children by mixing the selected fragments of the parents. Our crossover operator selects a random position in the list of use cases, from which the next rows of TADM matrix in two parents are replaced (Fig. 1). To avoid the GA from staying in local optima, mutation operator is used. The mutation operator alters defined number of genes in each individual, depending on the probability of mutation (Fig. 1). The termination condition is specified as a maximal number of iterations. The selected chromosome shows the best combination of automation decisions for all use cases through the entire testing activities.

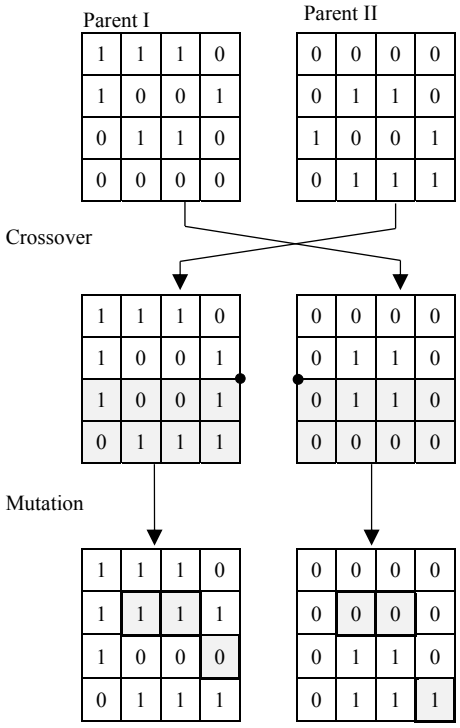

Fig. 1. Crossover and mutation of chromosomes.

D. Implementation (Tool Support)

To select the best combination of use cases to be automated in each test activity, we implemented a tool called the *Automated Testing Decision-Support System (ATDSS),* which automated our approach. ATDSS is implemented in Java and supports both brute force and the proposed GA-based technique. The tool is provided as open source and can be downloaded online from [27].

The inputs of the tool include: cost and benefit measures (*COST* and *BNFT* matrices). Output of the system is the optimal TADM matrix. Also, for the selected combination, the ROI estimated from automating the selected use cases in each activity is computed as output of the tool.

V. Industrial Case Study

A. Case Description

The case study was conducted in the context of an Action Research (AR) project as an industry-academia collaboration between Pason Systems Corporation and a team of researchers from the University of Calgary. The goal was to select the right combination of use cases for automation to maximize the ROI of the testing process. The SUT is called Electronic Tour Sheet (ETS), which is an

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

important sub-system of Pason's Electronic Drilling Recorder (EDR) system [28]. EDR is a legacy system and now is in maintenance phase. ETS has about 30 use cases. Company has just started to add automated test cases for their future regression tests of this system, and they needed to select the test cases that will be more cost-effective.

B. Case Study Design

Based on the guidelines provided by [13], the design of the case study is outlined below.

1) Goal and Research Questions

The overall business goal of the study is to decrease cost of software testing while increasing effectiveness of regression tests and ensuring the quality of the SUT. To implement this strategy and reach the business goals, we followed the GQM+ methodology [29] and defined the technical goal of this study as follows: design and implementation of a methodology for improving the cost effectiveness of software testing activities from the point of view of researchers and practitioner software testers. Based on the above goal, we raised the following research questions (RQ):

- RQ 1 – What is the best combination of use cases to be automated in each phase of testing?
- RQ 2 – How does the approach scale to larger number of use cases?

2) Metrics and Measurements

To answer the RQs, Table II shows the mapping between the RQs and the selected metrics. We briefly describe next the selected metrics and discuss the measurement method we used in this case study.

TABLE II. Metrics used in this study

RQ	Metrics
1	Fixed costs of each activity, Test design productivity, Test scripting productivity, Test execution productivity, and test-result evaluation productivity, Maintenance costs (Co-maintenance, and Quality improvement), Number of input parameters of each use case, Number of manual test steps, Number of assertions, Number of test cases for each use case
2	Execution time of the GA and Brute force algorithm, Number of use cases

Productivity measures the efficiency of conducting an activity. We measured productivity metric for each test activity and used it to find out the average time needed for fulfillment of the activity. For example, Equation (4) shows how we calculated the test-case design time.

$$DesignTime = \frac{1}{Test\ Design\ Productivity} \times \#TCs \quad (4)$$

Productivity of the activities is measured for manual and automated tests separately. For example, for calculating the test-case design productivity of manual testing, we manually applied the equivalence class partitioning, boundary value analysis, and combinatorial strategies to generate test data for each use case. For automated test-case design, we used the productivity value achieved by using Hexawise [30], an automatic data generator tool based on equivalence class partitioning. We should mention that all these measurements were conducted systematically and directly in the context of the project and inside the company when conducted by practitioners. We benefitted from systematic measurement guidelines such as [31] in our measurement activities.

We used different methods for data measurements, as discussed next. Whenever data were available in the software repositories in the firm (e.g., test management, issue tracking, code management tools), we used the existing recorded data. For the cases when the data were not available or had not been recorded precisely, we used interview or estimation models. For example, for test cases whose manual test execution time had not been recorded precisely, we customized and used a test execution effort estimation model presented in [32]. Due to confidentiality, we are not able to disclose further details of our measurement activities.

For test scripts, in addition to development cost, we considered the maintenance cost. There are two main sources for maintenance of test-ware which we considered in our approach: (1) maintenance in the production code which leads to co-maintenance cost, and (2) preventive and perfective refactoring activities in the test code which lead to quality improvement cost. We used the software maintenance estimation principle of COCOMO [14] to calculate the test maintenance cost according to test development cost, as shown in Equation (5).

$$Manitenance\ Cost = \alpha \times Test\ Development\ Cost \quad (5)$$

Where α shows the proportion of development cost that is invested for maintenance purposes. In our industrial case study, we calculated α according to the average maintenance effort invested in similar projects within the company.

Extracting cost and benefit matrixes needs some efforts. We collected data in an interactive way with the practitioner testers, using available data for current project and also data from similar projects of the company. All our collected and estimated data have been verified by company's manual testers, automation experts before using in the experiment.

3) Procedure

In the "exploratory" phase of the case study [13], researchers were involved in characterizing "what is happening" in Pason's current testing activities and processes. In the "improving" phase, we improved the cost effectiveness of testing activities.

In terms of our research approach, we followed both the improving case study [13] and the AR approaches [33]. To plan, execute and manage our AR project, we used the recommendation and guidelines provided in [33].

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

C. Results

In this section, the results of the study are presented for each of the RQs 1-2.

1) RQ 1-Use case selection for optimized automation

For selection of the best combination of use cases to be automated in each test activity, we applied the designed GA. Before applying GA, we had to tune its parameters. Other researchers, e.g., [34], and we [35, 36] have empirically found in previous studies that the execution time, the quality of outputs, and convergence of a GA are dependent on how well its parameters have been tuned. Using the guidelines reported in [34-36], we empirically tuned the control parameters of the GA. The parameter values that led to the most optimal GA performance in our case study were as follows: number of GA iterations = 160, population size=60, crossover ratio=0.85, mutation ratio=0.008. GA is repeated 1000 number of times for each experiment. Then, we executed our tool using GA and the most optimal TADM was generated for the estimated number of usage of artifacts during the life-span of the SUT. In fact, based on the estimated usage of test artifacts during the life-span of the software product, use cases for automation were selected in a way that benefits achieved before retirement of the application. Table III shows a sample result of the tool which relates to five rounds of usage/execution of the automated test artifacts. Due to space limitation, only five out of 30 use cases are shown in Table III and the names of use cases have been masked due to confidentiality. After executing our tool, the binary decision for all use cases and activities, and also the ROI of the selected combination were reported by our tool. The total ROI is the sum of ROI values achieved by different test activities. For this scenario, total ROI (367%) is the sum of ROI of test activities for five rounds of execution in Fig. 2, which is 101% and 266% for test design, and test execution, respectively.

TABLE III. The most optimal TADM

UC	Test Design	Test Scripting	Test Execution	Test Evaluation
…	…	…	…	…
UC 6	1	0	0	0
UC 7	1	1	1	0
UC 8	1	0	0	0
UC 9	1	1	1	0
UC 10	1	1	1	0
…	…	…	…	…
Total ROI = 378%				

To provide insights into the most optimal TADM in our case study, we discuss the case of one example use-case. For UC 8, the GA recommended that test design to be done automated while other test activities were selected to be done manually. The rationale behind these decisions were that the amount of effort needed to be invested in development and maintenance of this use case was greater than the amount of time saving that could be returned in five rounds of test execution. Due to interdependency between test scripting and execution activities, these activities were selected to be done manual. Fixed costs related to implementing facilities for automated test evaluation were high to be returned in five rounds of usage of automated test suites. Thus, test evaluation is also selected to be done manually. Based on the estimated number of usage of test suites during the life-span of the SUT, as an input for our tool, the decision can be changed. For example, if the life of SUT was estimated to be longer and it was possible to get returns before retirement of the SUT, the use case would be selected by the tool for automation.

In addition to find the most optimal TADM, we were interested to find out the ROI achieved from different test activities for the selected TADM by our tool. Fig. 2 shows the ROI received from each test activity for different number of usage/execution of test suites. As it can be seen from the chart, ROI of all activities increase by the rise in the usage of test artifacts during the life-span of SUT. For test design, test execution, and evaluation, receiving benefits from automation starts after 2, 3, and 8 executions, respectively. Therefore, the tool has not selected any use case for these activities before these points. This means that if the life-span of the SUT is short in a way that it would not be possible to utilize the automated test evaluation artifacts for at least 8 times, SUT will be retired before providing benefits for the test process. Automation before these points will result in loss of investment.

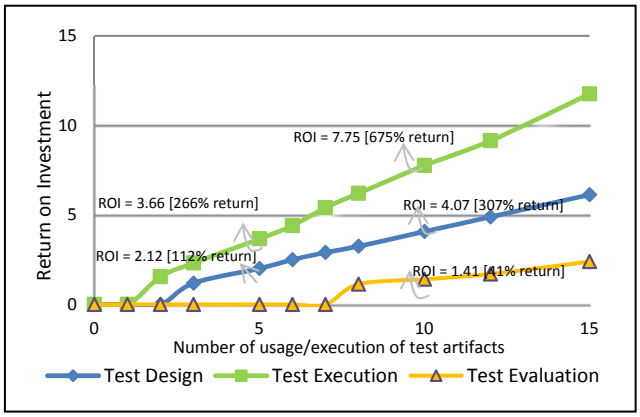

Fig. 2. ROI of test activities for different number of test artifact usage

Fig. 2 also shows that the major amount of ROI is acquired from automated test execution, followed by automated test design, and test-result evaluation. The results confirm that, although in comparison to other activities, automation in scripting activity leads to the highest ROI in test execution and evaluation, automation of other activities is also important. To receive higher ROI from test automation, analysis of all activities is essential. For this case study, when test artifacts are used for 10 times, in total 307%, 675%, and 41% return will be achieved in test-design, test execution, and test evaluation activities, respectively. In term of time saving, the achieved ROI are equal to 85, 275, and 21 days (8-hour work day) saving for the test process. According to the results, investment in scripting and design activities is highly

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

beneficial for this software. Importance of these activities can be due to the fact that SUT involved in this study was a data-intensive application. When we applied data-driven testing approach for this SUT, in which one test script can be used for execution of several designed inputs and outputs, we received high amount of ROI.

Since GA is a randomized algorithm, we discuss the repeatability of the algorithm next. We investigated the repeatability of GA results by analyzing the variation in maximum values of objective function of the best GA individual (chromosome) after the execution was finished. Then, we assessed the extent to which those values were repeatable. For this purpose, we run the algorithm for 1000 times and Fig. 3 shows distribution of the values for the objective function.

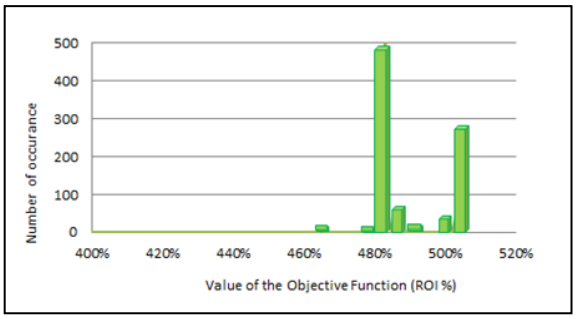

Fig. 3. Experimental results for repeatibality of the GA

As it can be seen in Fig. 3, there is a variation in GA results. Such a variation in fitness values across multiple runs is expected when using randomized algorithm such as GAs on a complex optimization problem such as decision making for test automation.

Most of the results are in range of [480 - 500], which shows the ability of GA algorithm to find a near optimal solution. As it is discussed in section V.B.2, each GA run lasts a few milliseconds, and it is possible to rely on multiple runs for making decision using GA. TABLE IV. shows the statistical result of our experiment with GA. These values are extracted from 1000 rounds of experiments for test artifacts with life-span of five times.

TABLE IV. Statistical values for objective function of GA

Average	4.78 (378%)
Minimum	3.99 (299%)
Maximum	5.05 (4.05%)

2) RQ 2-Scalability of the GA

To evaluate scalability of the designed approach with the increasing number of use cases, we compared the execution time of the GA for different number of use cases with the execution time of the brute force approach in Fig. 4. We used a similar approach followed in our previous GA-based test technique which was in the context of performance testing of real-time systems [35, 36]. Brute force is a search technique which tries all combinations (searches the entire search space exhaustively) and guarantees to find a solution if it exists. Therefore, cost of brute force search is proportional to the size of the search space.

Fig. 4 shows the execution time of both algorithms for different number of use cases. It should be noted that in order to show both curves clearly, we had to use different scales for the algorithms. Execution time of the brute force algorithm is shown in Fig. 4–a in "minute" scale, and Fig. 4–b relates to GA, in "second" scale. Both approaches were implemented in Java as part of our ATDSS tool [27] and were executed in the same machine and environment. The execution time was recorded for both genetic and brute force algorithms. There are $2^{|\#\text{USE CASE}|\times|\#\text{ACTIVITY}|}$ different combinations that should be considered in brute force approach and as it can be seen from the graph, with the increase in the number of use cases, the execution time of brute force approach grows sharper and this limits the scalability of the brute force technique for larger systems. However, the GA shows acceptable execution time for large number of use cases.

D. Discussions and Lessons Learned

From a technical point of view, the ultimate goal of practitioners was selecting the right (the most cost effective) combination of use cases to be tested in an automated manner. Selection of use cases before starting to automate them can prevent large amounts of rework in terms of test maintenance activities in the project.

Once we used our GA tool to determine the most optimal TADM, we started the "improving" phase of the case study to improve the cost effectiveness of testing activities in the project under study. The improvements have started already in the project under study and we have already been observing noticeable cost savings in the test processes.

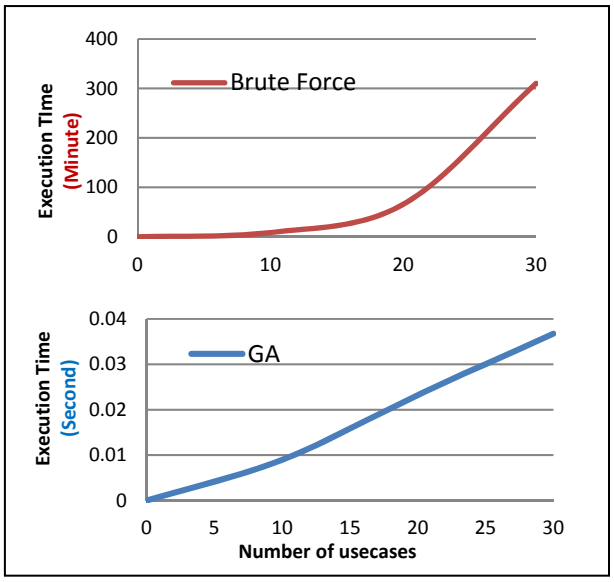

Fig. 4. Execution time of genetic and brute force algorithms

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

Based on our study, the decision for test automation should be studied in a wider context than only writing test scripts. Automation should be seen as a strategy to be applied on the entire testing process rather than just in test execution. Automating other phases of testing can be helpful for getting higher ROI, as we discussed in Section V.C. Also, the scalability of such an approach to be applicable in industrial case studies is very important and solutions designed in theory should be applicable in real contexts to be able to utilize them.

In terms of lessons learned from the research approach standpoint, we learned throughout the project that AR studies do not only help solving a real industrial problem, but it also helps both researchers and practitioners to collaborate more closely and exchange solutions and points of view. Researchers' on-site presence helped them to better understand real industry-relevant issues and challenges. Collaboration and exchange of information between researchers and practitioners can help to design more applicable approaches that are easily transferable to the industry. Moreover, we learned that although AR studies are initiated to solve an immediate problem, the researchers can use the opportunity to design general approaches applicable in other contexts, while making sure that the approach benefit the practitioners in current context. The proposed search-based approach designed in this study is independent of the properties of the SUT and will replace subjective automation decisions with systematic approach for making automation decisions.

E. Threats to Validity

Based on the guidelines provided in [13], this section discusses the construct and external threats to the validity of the reported study. We discuss next the steps we have taken to mitigate the potential threats to validity.

Construct validity aspects reflect to what extent the operational measures that are studied really represent the aim of the study. To limit this type of threat and to follow the goal of the practitioners in the operational measurement level, we used the GQM$^+$ approach [29] which explicitly links measurement goal to business level goals. Measurement goal, question and metrics were reviewed and finalized among researches and practitioners before conducting the measurements. One other potential construct threat to validity of this study is concerned with measurement bias, i.e., the extent with which the data and the analysis as conducted by the specific researchers. To limit the validity, two of the researchers involved in this study measured the data individually and compared and discussed the measured values before using them in the model. Also, we reviewed the measured data with practitioner testers in our regular weekly meetings. However, there is still possibility that the collected data can be different from the real context and it would be better for industrial settings to have systematic provisions in place to collect and update the influencing factors such as execution time, scripting, and maintenance rate during their test process.

External validity is concerned with to what extent the results of a study can be generalized to other contexts. Although this study is done as an AR-based study to solve a specific problem in a single company, we do not see any reason that our proposed solution would be limited for that specific context. The proposed search-based technique is intended to be applicable in other industrial contexts. Although at the moment, we have no empirical data on whether this approach will work equally well on other contexts, but can think of no reason why it would not.

VI. Conclusions and Future Works

In this study, we investigated effective test automation problem across the entire testing life-cycle. In practice, the number of possible combinations to be considered can be large and needs effective search techniques to be applied. A specific GA was developed in this study to find the best solution at a reasonable computational cost. In our context, experiments show that the GA worked faster than the brute force algorithm for higher number of use cases.

Similar to many other practitioners and researchers [3-5, 37], the focus of test automation in the company under study was only on test execution. The results of this study show that automation in other test activities can have benefits for test process and practitioner test automation experts need to pay attention to automation of the entire test activities.

The proposed model was designed to support two extreme decisions for each test activity: (1) fully manual, and (2) fully automated. As a future step, we plan to extend the approach to support the in-between case as well, i.e., partially automated. Also, we plan to conduct further empirical validations on this approach by applying it in other industrial settings and relaxing the assumptions. Furthermore, similar to our previous work in other area of SBSE [36, 38], we plan to conduct empirical studies on the performance and repeatability aspects of our GA approach.

Acknowledgment

This work was primarily supported by an NSERC ENGAGE grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). Vahid Garousi was additionally supported by the Visiting Scientist Fellowship Program (#2221) of the Scientific and Technological Research Council of Turkey (TÜBİTAK) and Atilim University. We would like to sincerely thank all the software engineers in Pason who devoted their time to this action-research project.

References

[1] R. Ramler and K. Wolfmaier, "Economic perspectives in test automation: balancing automated and manual testing with opportunity cost," in *Proceedings of the 2006 international workshop on Automation of software test*, 2006, pp. 85-91.

[2] I. Burnstein, T. Suwanassart, and R. Carlson, "Developing a testing maturity model for software test process

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

evaluation and improvement," in *Test Conference, 1996. Proceedings., International*, 1996, pp. 581-589.

[3] K. Stobie, "Too much automation or not enough? When to automate testing," 2009.

[4] R. W. Rice, C. CSQA, and L. Rice Consulting Solutions, "Surviving the top ten challenges of software test automation," *CrossTalk: The Journal of Defense Software Engineering*, pp. 26-29, 2003.

[5] D. J. Mosley and B. A. Posey, *Just Enough Software Test Automation*: Prentice Hall Professional, 2002.

[6] S. Desikan and G. Ramesh, *Software Testing: Principles and Practices*: Pearson Education India, 2006.

[7] E. Dustin, T. Garrett, and B. Gauf, *Implementing automated software testing: How to save time and lower costs while raising quality*: Addison-Wesley Professional, 2009.

[8] S. Berner, R. Weber, and R. K. Keller, "Observations and lessons learned from automated testing," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 571-579.

[9] G. Meszaros, *xUnit test patterns: Refactoring test code*: Pearson Education, 2007.

[10] B. Daniel, T. Gvero, and D. Marinov, "On test repair using symbolic execution," in *Proceedings of the 19th international symposium on Software testing and analysis*, 2010, pp. 207- 218.

[11] F. Ensan, E. Bagheri, and D. Gašević, "Evolutionary search-based test generation for software product line feature models," in *Advanced Information Systems Engineering*, 2012, pp. 613-628.

[12] K. Lakhotia, M. Harman, and P. McMinn, "A multi-objective approach to search-based test data generation," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, pp. 1098-1105.

[13] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131-164, 2009.

[14] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson, "A model for technology transfer in practice," *Software, IEEE*, vol. 23, pp. 88-95, 2006.

[15] C. Pinheiro, V. Garousi, F. Maurer, and J. Sillito, "Introducing Automated Environment Configuration Testing in an Industrial Setting," in *Software Engineering and Knowledge Engineering (SEKE)*, 2010, pp. 186-191.

[16] S. A. Jolly, V. Garousi, and M. M. Eskandar, "Automated Unit Testing of a SCADA Control Software: An Industrial Case Study Based on Action Research," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, 2012, pp. 400-409.

[17] P. McMinn, "Search-based software test data generation: a survey," *Software Testing, Verification and Reliability*, vol. 14, pp. 105–156, 2004.

[18] A. P. Mathur, *Foundations of Software Testing*: Addison-Wesley Professional, 2008.

[19] P. Ammann and J. Offutt, *Introduction to Software Testing*: Cambridge University Press, 2008.

[20] J. Czerwonka, "Pairwise Testing," *http://www.pairwise.org/*, Online, Last accessed: Aug. 4, 2013.

[21] P. Tahchiev, F. Leme, V. Massol, and G. Gregory, *JUnit In Action*: Manning Publications Company, 2010.

[22] C. T. Brown, G. Gheorghiu, and J. Huggins, *An Introduction to Testing Web Applications with twill and Selenium*: O'Reilly Media, 2007.

[23] S. R. Shahamiri, W. M. N. W. Kadir, and S. Z. Mohd-Hashim, "A Comparative Study on Automated Software Test Oracle Methods," in *International Conference on Software Engineering Advances*, 2009.

[24] S. R. Shahamiri, W. M. N. W. Kadir, and S. Z. Mohd-Hashim, "A comparative study on automated software test oracle methods," in *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on*, 2009, pp. 140-145.

[25] M. S. Feather and B. Smith, "Test oracle automation for V&V of an autonomous Spacecraft's planner," in *Proc. of the 2001 AAAI Symposium*, 2001.

[26] S. R. Choudhary, D. Zhao, H. Versee, and A. Orso, "WATER: Web Application TEst Repair," in *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, 2011, pp. 24 - 29.

[27] Y. Amannejad and V. Garousi, "ATDSS source code," *http://www.softqual.ucalgary.ca/sw_tools.html*, Online, Last access: Sep. 22, 2013.

[28] P. S. Corporations, "Products & Services," *http://www.pason.com/*, Online, Last accessed: Aug. 4, 2013.

[29] V. Basili, J. Heidrich, M. Lindvall, J. Munch, M. Regardie, and A. Trendowicz, "GQM^+ Strategies--Aligning Business Strategies with Software Measurement," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 488-490.

[30] Hexawise Team, "Hexawise – Pairwise Testing Made Easy," *http://www.hexawise.com*, Online, Last accessed: Dec. 3, 2012.

[31] R. E. Park, W. B. Goethert, and W. A. Florac, "Goal-Driven Software Measurement. A Guidebook," *Technical report, CMU/SEI-96-HB-002*, 1996.

[32] E. Aranha and P. Borba, "An estimation model for test execution effort," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 107-116.

[33] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen, "Action research," *Communications of the ACM*, vol. 42, pp. 94-97, 1999.

[34] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Search Based Software Engineering*, ed: Springer, 2011, pp. 33-47.

[35] V. Garousi, "A genetic algorithm-based stress test requirements generator tool and its empirical evaluation," *Software Engineering, IEEE Transactions on*, vol. 36, pp. 778-797, 2010.

[36] V. Garousi, "Empirical analysis of a genetic algorithm-based stress test technique," in *Proceedings of the 10th*

This is the post-print of a paper that has been published with the following DOI: http://dx.doi.org/10.1109/ICSTW.2014.34

annual conference on Genetic and evolutionary computation, 2008, pp. 1743-1750.

[37] E. Dustin, J. Rashka, and J. Paul, *Automated software testing: introduction, management, and performance*: Addison-Wesley Professional, 1999.

[38] V. Garousi, L. C. Briand, and Y. Labiche, "Traffic-aware stress testing of distributed real-time systems based on UML models using genetic algorithms," *Journal of Systems and Software*, vol. 81, pp. 161-185, 2008.