Experimental Assessment of Manual Versus Tool-Based Maintenance of GUI-Directed Test Scripts

Mark Grechanik, Qing Xie, and Chen Fu
Accenture Technology Labs
Chicago, IL 60601
{mark.grechanik, qing.xie, chen.fu}@accenture.com

Abstract

Since manual black-box testing of GUI-based APplications (GAPs) *is tedious and laborious, test engineers create test scripts to automate the testing process. These test scripts interact with GAPs by performing actions on their GUI objects. As GAPs evolve, testers should fix their corresponding test scripts so that they can reuse them to test successive releases of GAPs.*

Currently, there are two main modes of maintaining test scripts: tool-based and manual. In practice, there is no consensus what approach testers should use to maintain test scripts. Test managers make their decisions ad hoc, based on their personal experience and perceived benefits of the tool-based approach versus the manual.

In this paper we describe a case study with forty five professional programmers and test engineers to experimentally assess the tool-based approach for maintaining GUI-directed test scripts versus the manual approach. Based on the results of our case study and considering the high cost of the programmers' time and the lower cost of the time of test engineers, and considering that programmers often modify GAP objects in the process of developing software we recommend organizations to supply programmers with testing tools that enable them to fix test scripts faster so that these scripts can unit test software. The other side of our recommendation is that experienced test engineers are likely to be as productive with the manual approach as with the tool-based approach, and we consequently recommend that organizations do not need to provide each tester with an expensive tool license to fix test scripts.

1. Introduction

Manual *black-box testing* of *Graphical User Interface (GUI)-based APplications (GAPs)* is tedious and laborious, since nontrivial GAPs contain hundreds of GUI screens and thousands of GUI objects. Test automation plays a key role in reducing high cost of testing GAPs [9][11][5]. In order to automate this process, test engineers write programs using scripting languages (e.g., JavaScript and VBScript), and these programs (*test scripts*) mimic users by performing actions on GUI objects of these GAPs using some underlying testing frameworks. An extra effort put in writing test scripts is paid off when these scripts are run repeatedly to determine if GAPs behave as desired.

1.1. Evolving GAPs And Test Scripts

Unfortunately, releasing new versions of GAPs with modified GUIs breaks their corresponding test scripts thereby obliterating the benefits of test automation [15][4]. Consider a situation when a list box is replaced with a text box in the successive release of some GAP. Test script statements that select different values in this list box will result in exception when executed on the text box. This simple modification may invalidate many statements in test scripts that reference this GUI object. Maintaining test scripts involves changing its code to keep up with changes to their corresponding GAPs.

This and many other similar modifications are typical between successive releases of different GAPs, including such well-known GAPs as Adobe Acrobat Reader and Microsoft Word. As many as 74% of the test cases become unusable during GUI regression testing [21], and internal evaluations of automated testing in Accenture show that even simple modifications to GUIs result in 30% to 70% changes to test scripts. To reuse these scripts, test engineers should fix them. For example, at Accenture over six thousand of test personnel fix test scripts both manually and using different testing tools. The annual cost of manual maintenance and evolution of test scripts is estimated to be between $50 to $120 millions just in Accenture alone.

1.2. Modes of Maintenance

Currently, there are two main modes of maintaining test scripts: tool-based and manual. Existing testing tools detect exceptions in test scripts at runtime, i.e., test engineers must run these scripts (often for many hours because scripts often contain loops) in order to execute statements that reference modified GUI objects. Exceptions interrupt continuous testing and they require human intervention to fix them.

Unlike compilers that check unit tests against the program code, test scripts are based on different type systems than GAPs that they test. As it turns out, multiple disparate type systems make GUI testing very difficult. Existing regression testing approaches work in settings where test harnesses are written in the same language and use the same type system as the programs that these harnesses test (e.g., JUnit test harnesses are applied to Java programs). In contrast, when testing GAPs two type systems are involved: the type system of the language in which the source code of the GAP is written and the type system of the language in which test scripts are written. When the type of the GUI object is modified, the type system of the test script "does not know" that this modification occurred, thereby aggravating the process of maintaining and evolving test scripts.

As a result, tool-based approaches allow testers to find broken statements in test scripts by executing these statements line-by-line against GAPs. This is called the maintenance mode. The presence of loops in test scripts make them run for a long time in order to reach statements that should be checked. Test engineers comment out loops, but their modifications may change testing logic and mask broken statements. Finally, commercial testing tools are expensive (a license for *Quick Test Pro (QTP)*, a flagship industry tool from Hewlett-Packard Corp. costs more than $10,000). Our investigation revealed that this maintenance mode is inherent for most existing open-source and commercial automated testing tools. In this paper, we concentrate on QTP[1].

On the other hand, manual maintenance of test scripts is popular among test professionals. During manual maintenance testers determine differences between successive release of GAPs and they locate and fix statements in test scripts that are affected by these changes. Since the sizes of test scripts are much smaller than GAPs that these scripts act upon (e.g., many scripts are smaller that 1KLOC), it is easier for testers to understand and fix them. In addition, testers are perceived to do a more thorough job of understanding and fixing scripts if they do not rely heavily on tool support. However, some test engineers lack time and necessary skills to understand and fix old scripts, especially if these scripts were created by other engineers.

[1]While the worldwide market for automated test tools is over $1.1Bil, QTP is used by over 90% of Fortune 500 companies [3].

1.3. Cost-Benefit Analysis

Cost-benefit analysis of tool-based approach versus the manual one is complicated. Licenses for tools are expensive. At the same time the time of testers is also expensive. If using testing tools does not result in significant time savings, the cost of tool licenses will increase the cost of testing. It is imperative to understand when using test tools is effective, so that managers can plan test automation for software projects.

Besides test engineers, GUI developers test their code by creating and eventually maintaining test scripts. Since these developers spend major part of their time writing source code of GAPs rather than testing them, managers often think that the purchasing licenses of expensive testing tools for developers will not result in significantly increased savings of their time. On the other hands, developer's time is more expensive, and it is desirable that they spend their time writing code than fixing test scripts. Testing tools may make developer's work more productive. Thus, it is important for cost-benefit analysis to understand what approach is more effective for programmers as well as testers.

Leasing cars versus buying them can serve as a useful analogy to understand the process for leasing testing tools versus buying them. Leasing tools for a short period of time results in a higher cost per hour of usage of these tools, however, the overall cost of ownership is low. If GAPs did not evolve, their test scripts would likely to stay unchanged once created, and leasing testing tools for a short period of time to create these scripts would be economic. However, since test scripts should be maintained on a regular basis, tool licences should be purchased to reduce the overall cost of ownership. Purchasing more licenses than it is economically required is detrimental for the cost of software projects.

To run test scripts, only one tool license is required; however, if testers maintain test scripts on a regular basis, many tool licenses are needed, one per tester. When testers maintain scripts manually, tool license can be leased for a short period of time to run scripts, and the cost of the ownership of the tool is minimal. On the other hand, it requires significant investment for each tester to maintain scripts with the help of testing tool, since it would mean a purchasing many licenses of testing tools.. A trade-off between the cost of the tool licenses and the increase in testing productivity justifies using manual versus tool-based approaches.

In practice, there is no consensus which approach testers and GUI developers should use to maintain test scripts. Test managers make their decisions ad hoc, based on their personal experience and perceived benefits of the tool-based approach versus the manual. We do not know of any comprehensive study that investigated these approaches and offered statistically significant results. Currently, testers use tool-based approaches on an ad-hoc basis, while programmers rarely use GUI-directed test tools for unit testing [1]. A survey in 250 organizations found that only 35% of testers used automated testing tools one year after the tool installation [19].

1.4. Our Contributions

This paper makes the following contributions to the field of GUI testing:

- a large scale case study with forty five professional programmers and test engineers to empirically assess the cost-benefit ratio for maintaining GUI-directed test scripts using a tool-based versus a manual approach,
- statistical evidence that demonstrates users find more broken test script statements due to changes in GUI objects between successive releases of GAPs and report fewer false positives in test scripts with QTP than with a manual approach,
- statistical evidence that suggests significantly higher productivity with programmers but similar results with experienced test engineers when using the testing tool compared to those who maintained test scripts using the manual approach,
- recommendation that organizations supply programmers with testing tools that enable them to fix test scripts faster so that these scripts can unit test software but do not need to provide each test engineer with an expensive tool license to fix test scripts.

2. Case Study Design

We gathered open source GUI applications (two versions each) as benchmarks with test scripts for the old version of each GAP. Participants of the case study are asked to identify failures when the scripts are run against the new version of corresponding GAP. We define test script statements that access and manipulate GUI objects as *failures* if these statements are broken because of modifications made to the referenced GUI objects in the successive releases of QTP. Our goal is to evaluate how well these participants can find failures in test scripts (when running against the new version of the GAP) using two different approaches: manual and QTP as a guiding tool. Specifically, we want to determine using which approach users can report more *correctly identified failures (CIF)* in test scripts that result from changed GUI objects between successive releases of the subject GAPs, and with which approach users report fewer false positives (FPs), i.e., correct statements in test scripts that participants report as failures by mistake. We also report how many failures participant miss, i.e., *missed failures (MFs)*. We are especially interested how two different groups of users (**testers** who have testing experience and **non-testers** as programmers who do not have any testing experience) perform using the tool-based and the manual approaches.

2.1. Hypotheses

We introduce the following null and alternative hypotheses to evaluate how close the means are for the CIFs, FPs, and MFs for control and treatment groups. Unless we specify otherwise, participants of the treatment group use QTP, and participants of the control group use the manual approach. We seek to evaluate the following hypotheses at a 0.05 level of significance.

H_0 The primary null hypothesis is that there is no difference in the numbers of CIFs, FPs, and MFs between participants who attempt to locate failures in test scripts manually or using QTP.

H_1 An alternative hypothesis to H_0 is that there is statistically significant difference in the numbers of CIFs, FPs, and MFs between participants who attempt to locate failures in test scripts manually or using QTP.

Once we test the null hypothesis H_0, we are interested in the directionality of means, μ, of the results of control and treatment groups. We are interested to compare the effectiveness of the tool QTP versus the baseline manual approach with respect to CIFs, MFs, and FPs.

H1 (CIFs for all participants) The effective null hypothesis is that $\mu_{cif}^{QTP} = \mu_{cif}^{Manual}$, while the true null hypothesis is that $\mu_{cif}^{QTP} \geq \mu_{cif}^{Manual}$.

H2(FPs for all participants) The effective null hypothesis is that $\mu_{fp}^{QTP} = \mu_{fp}^{Manual}$, while the true null hypothesis is that $\mu_{fp}^{QTP} \leq \mu_{fp}^{Manual}$.

H3(MFs for all participants) The effective null hypothesis is that $\mu_{mf}^{QTP} = \mu_{mf}^{Manual}$, while the true null hypothesis is that $\mu_{mf}^{QTP} \leq \mu_{mf}^{Manual}$.

H4 (CIFs for testers) The effective null hypothesis is that $\mu_{cif}^{QTP} = \mu_{cif}^{Manual}$, while the true null hypothesis is that $\mu_{cif}^{QTP} \geq \mu_{cif}^{Manual}$.

H5(FPs for testers) The effective null hypothesis is that $\mu_{fp}^{QTP} = \mu_{fp}^{Manual}$, while the true null hypothesis is that $\mu_{fp}^{QTP} \leq \mu_{fp}^{Manual}$.

H6(MFs for testers) The effective null hypothesis is that $\mu_{mf}^{QTP} = \mu_{mf}^{Manual}$, while the true null hypothesis is that $\mu_{mf}^{QTP} \leq \mu_{mf}^{Manual}$.

H7 (CIFs for non-testers) The effective null hypothesis is that $\mu_{cif}^{QTP} = \mu_{cif}^{Manual}$, while the true null hypothesis is that $\mu_{cif}^{QTP} \geq \mu_{cif}^{Manual}$.

H8(FPs for non-testers) The effective null hypothesis is that $\mu_{fp}^{QTP} = \mu_{fp}^{Manual}$, while the true null hypothesis is that $\mu_{fp}^{QTP} \leq \mu_{fp}^{Manual}$.

H9(MFs for non-testers) The effective null hypothesis is that $\mu_{mf}^{QTP} = \mu_{mf}^{Manual}$, while the true null hypothesis is that $\mu_{mf}^{QTP} \leq \mu_{mf}^{Manual}$.

In addition, we want to know if the performance of the participants who have testing experience differs from those who do not have any testing experience. The categorical variables are testing experience and reported CIFs, FPs, and MFs. Hypothesis H10 considers correlation between the hypotheses H4 and H7, hypothesis H11 considers correlation between the hypotheses H5 and H8, and the hypothesis H12 considers correlation between the hypotheses H6 and H9.

H10 (Independence of testing experience from CIFs) the testing categorical variable is independent from the variable CIF; the alternative is that they are associated.

H11 (Independence of testing experience from FPs) the testing categorical variable is independent from the variable FP; the alternative is that they are associated.

H12 (Independence of testing experience from MFs) the testing categorical variable is independent from the variable MFs; the alternative is that they are associated.

2.2. Subject GAPs and Test Scripts

We selected four open source subject GAPs based on the following criteria: easy-to-understand domain, limited size of GUI (less than 200 GUI objects), and two successive releases of GAPs with modified GUI objects. `Twister` (versions 2.0 and 3.0.5) is a real-time stock quote downloading programming environment that allows users to write programs that download stock quotes[2]. `mRemote` (versions 1.0 and 1.35) enables users to manage remote connections in a single place by supporting various protocols (e.g., SSH, Telnet, and HTTP/S)[3]. `University Directory` (versions 1.0 and 1.1) allows users to obtain data on different universities[4]. Finally, `Budget Tracker` (versions 1.06 and 2.1) is a program for tracking budget categories, budget planning for each month and keeping track of expenses[5].

[2]http://sourceforge.net/projects/itwister/

[3]http://sourceforge.net/projects/mremote/

[4]http://sourceforge.net/projects/universitydir/

[5]http://sourceforge.net/projects/budgettracker/

Subject Program	Size				Total Failures
	Script LOC	Refd GUI, objs	Add, GUI objs	Del GUI objs	
Twister	492	54	81	12	16
mRemote	538	17	42	20	17
Univ Dir	920	36	35	9	13
Budget Tr	343	8	18	5	14

Table 1. Subject GAPs and test scripts. Column `Size` contains five subcolumns reporting the numbers of LOC in test scripts, the number of GUI objects that are referenced in the script, and the numbers of added GUI objects to the successive version of the GAP and the number of the deleted GUI objects from the previous version of the GAP. The column `Total Failures` shows the number of failures in test scripts.

Most of these applications are nontrivial, they are highly ranked in Sourceforge with the activity over 95%.

Next step was to obtain test scripts for subject GAPs. We obtained existing test scripts from sample script libraries that come with QTP. These scripts contained both GUI and non-GUI related code (e.g., setting values of environment variables and reading and manipulating directories contents). To make these scripts thorough, we generated statements that referenced GUI objects in the subject GAPs using QTP. Then we interspersed and replicated the generated statements throughout the test scripts. Information on subject GAPs and test scripts can be found in Table 1.

2.3. Context

We developed a novel approach for maintaining and evolving test scripts so that they can test new versions of their respective GAPs. We built a tool to implement our approach, and we conducted a case study with forty five professional programmers and test engineers to evaluate this tool [13]. During the course of this study we evaluated our tool with respect to QTP and the manual approach. In this paper, we report only results of evaluation of the manual versus QTP without mentioning the results of the evaluation of our tool.

2.4. Methodology

We used a cross validation study design in a cohort of 45 participants who were randomly divided into two blocks labeled using different color labels. The study was sectioned in two experiments in which each block was given a different approach (manual and QTP) to apply to the subject

GAPs. Thus each participants used each approach on different GAPs in the process of the case study. We randomly distributed participants so that each block has approximately the same number of participants with and without testing experience. Before the study we gave two one-hour tutorials on using each of these approaches on a GAP (mRemote) that was not used during the experiments thereby eliminating the knowledge of the GAP as a possible confounding factor.

All participants are Accenture employees who work on consulting engagements as programmers and managers for different client companies. These participants have different backgrounds, experience, and belong to different groups of the total Accenture workforce of approximately 180,000 employees. Out of 45 participants (14 of whom are women), 23 had prior testing experience ranging from three weeks to ten years, and 18 participants reported prior experience with writing programs in scripting languages, including test scripts. Seven participants reported prior experience with QTP (which is used in this case study), six participants reported prior experience with other GUI testing tools. Twenty nine participants have bachelor degrees and ten have master degrees in different technical disciplines.

2.5. Tests and The Normality Assumption

We use one-way ANOVA, t-tests for paired two sample for means, and χ^2 [28] to evaluate the hypotheses that we stated in Section 2.1. One-way ANOVA tests differences among two or more independent groups, and the two-group case can be covered by t-tests. χ^2 test analyzes contingency tables that consist of rows and columns to determine if the observed cell frequencies differ significantly from the expected frequencies. These tests are based on an assumption that the population is normally distributed. The law of large numbers states that if the population sample is sufficiently large (between 30 to 50 participants), then the central limit theorem applies even if the population is not normally distributed [28, page 244-245]. Since we have 45 participants, the central limit theorem applies, and the above-mentioned tests have statistical significance.

2.6. Normalizing Sources of Variations

We design this experiment to drastically reduce the effects of covariates (i.e., nuisance factors) in order to normalize sources of variations. Using the cross-validation design we normalize variations to a certain degree since each participant uses all three approaches on different subject GAPs. We selected participants who had no prior knowledge of the subject GAPs. At the same time, subject GAPs belong to domains that are easy to understand, and these GAPs have similar complexity, so variations between them are negligent. However, different computing environments and prior experience of users with testing scripts and subject GAPs are major covariates.

We eliminated the effect of the computing environments by providing all participants with Dell Latitude D630 laptops with Intel Core 2 Duo Processor 2.4GHz with 4MB L2 Cache, 1Gb RAM, and 14.1" WXGA+ displays. Technical support at Accenture burnt the same standard Windows XP-based image on these laptops. We installed GAPs, scripts, and tools in a virtual machine that runs on top of the Microsoft Virtual PC thereby allowing participants to obtain a common environment of the entire experimental setup.

2.7. Threats to Validity

A threat to validity of this case study is that our subject GAPs have GUI screens that are of small to moderate size (a couple of hundreds of GUI objects). Increasing the size of GUIs of GAPs to thousands of GUI objects may lead to a nonlinear increase in the analysis time and space demand for both tool-based and manual approaches.

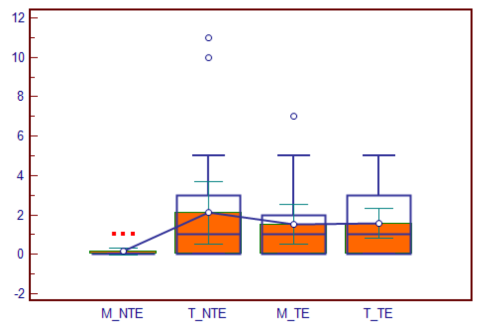
Since seven participants reported prior experience with QTP, this case study can be viewed as biased towards QTP versus the manual approach and Manual. To reduce this bias, we provided a comprehensive tutorial on QTP for all participants of the study, and given the large number of participants we expect this bias to be negligent. However, the results of this study show that participants who had prior experience with QTP performed better with the manual approach. In addition, prior testing experience of the participants remains a source of variation.

In this study we distinguish between participants as testers and non-testers (programmers) based on the data that these participants reported in their resumes and our questionnaires. This division does not account for large variations in different skills that may affect the results of this case study. Specifically, it is desired to conduct an independent evaluation of the skills of the participants using programming tasks, which was not done due to time constraints.

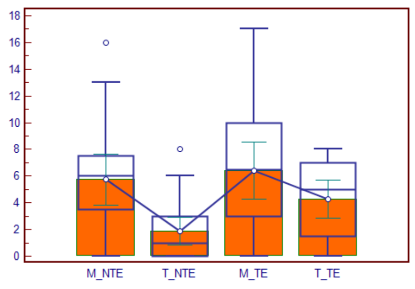
3. Results

3.1. Benchmark Evaluation

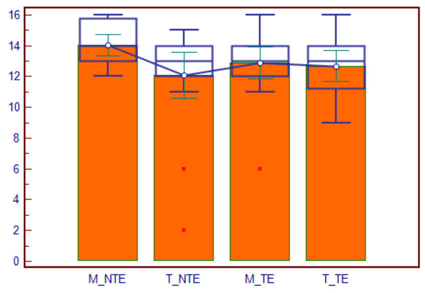
We measured characteristics of GAPs and test scripts by running these scripts using Windows XP Pro that ran on a computer with Intel Pentium IV 3.2GHz CPU and 2GB of RAM. Experimental results of applying QTP and scripts to the subject programs and scripts are shown in Table 1.



(a) Correctly Identified Failures (CIF).

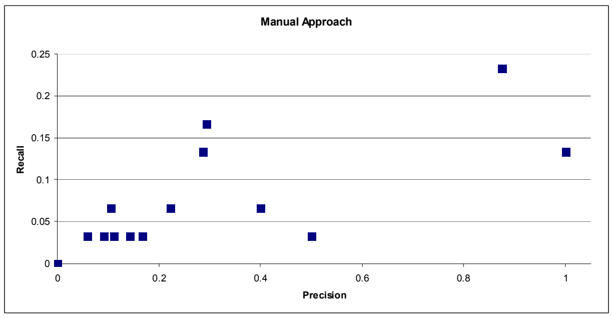


(b) False Positive (FPs) by approach.

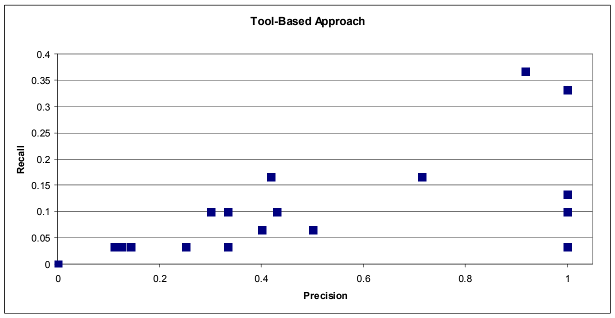


(c) Missed Failures (MFs) by approach.

Figure 1. Statistical summary of the results of the case study for CIFs and FPs. The central box represents the values from the lower to upper quartile (25 to 75 percentile). The middle line represents the median. The thicker vertical line extends from the minimum to the maximum value, excluding outside and far out values, which are displayed as separate circles and small squares. The filled-out box represents the values from the minimum to the mean, and the thinner vertical line extends from the quarter below the mean to the quarter above the mean. An outside value is defined as a value that is smaller than the lower quartile minus 1.5 times the interquartile range, or larger than the upper quartile plus 1.5 times the interquartile range (inner fences). A far out value is defined as a value that is smaller than the lower quartile minus three times the interquartile range, or larger than the upper quartile plus three times the interquartile range (outer fences). Each bar is marked with the description of the experiment, where M stands for manual, T stands for tool-based approach, TE stands for the group with testing experience, and NTE stands for the group with no testing experience.



(a) Manual approach.



(b) Tool-based approach.

Figure 2. Precision and recall graphs.

3.2. Case Study Results

In this section, we report the results of the case study and evaluate null hypotheses. We use one-way ANOVA, t-tests for paired two samples for means, and χ^2 to evaluate the hypotheses that we stated in Section 2.1.

3.2.1 Variables

A main independent variable is the approach (manual, QTP) that participants use to find failures in test scripts. The other independent variable is participants' testing experience. Dependent variables are the numbers of correctly identified failures (CIFs), false positives (FPs), and missed failures (MFs). We report these variables in this section. The effect of other variables (GAPs, test scripts, prior knowledge) is minimized by the design of this case study.

3.2.2 Testing the Null Hypothesis

We used ANOVA to evaluate the null hypothesis H_0 that the variation in an experiment is no greater than that due to normal variation of individuals' characteristics and error in their measurement. The results of ANOVA with respect to CIFs confirm that there are large differences between the groups for CIF with $F = 4.12 > F_{crit} = 3.97$ with $p < 0.05$ which is statistically significant. The mean CIF for the manual approach is 0.84 with the variance 2.6, which is smaller than the mean CIF for QTP, 1.84 with the variance 6.6.

Similarly, the results of ANOVA with respect to FPs confirm that there are large differences between the groups for CIF with $F = 12.1 > F_{crit} = 3.97$ with $p < 0.0009$ which is strongly statistically significant. The mean FP for the manual approach is 6.1 with the variance 19.4, which is bigger than the mean FP for QTP, 3.1 with the variance 8.7.

Finally, the results of ANOVA with respect to MFs confirm that there are large differences between the groups for MF with $F = 4.43 > F_{crit} = 3.97$ with $p < 0.04$ which is strongly statistically significant. The mean MF for the manual approach is 13.5 with the variance 3.6, which is bigger than the mean MF for QTP, 12.4 with the variance 7.1. Based on these results we reject the null hypothesis and we accept the alternative hypothesis H_1.

A statistical summary of the results of the case study for CIFs and FPs (median, quartiles, range and extreme values) are shown as box-and-whisker plots in Figure 1(a), Figure 1(b), and Figure 1(c) correspondingly with 95% confidence interval for the mean.

3.2.3 Comparing QTP with Manual

To test the null hypotheses H1 to H9, we applied two t-tests for paired two sample for means, for CIFs, FPs, and MFs for participants who used QTP and the manual approach. Based on results of the experiments we reject the null hypotheses H2, H7, H8, and H9, and we accept the alternative hypotheses for H7, H8, and H9 that say that **participants without testing experience (programmers) who use QTP report fewer false positives, correctly identify more failures, and miss fewer failures in test scripts than those who use the manual approach**.

Alternatively, we accept null hypotheses H1, H3, H4, H5, and H6 and reject the corresponding alternative hypotheses for H4, H5, and H6 that say that **participants with testing experience (testers) who use QTP report approximately the same numbers of false positives and correctly identify the same number of failures in test scripts than those who use the manual approach**.

We explain this result using the key differentiator between testers and programmers - testing experience. Programmers who do not have testing experience and whose goal is to write code rather than test it, rely on testing tools more than testers who understand test scripts and know how to fix them. Testers know the limitations of testing tools, and they can work without them as effectively as when using these tools.

Based on the results of our case study and considering a high cost of the programmers' time and a lower cost of the time of test engineers [1], and considering that programmers often modify GAP objects in the process of developing software [19] we recommend to supply programmers with testing tools that enable them to fix test scripts faster so that these scripts can unit test software. The other side of our recommendation is that experienced test engineers are likely to be as productive with the manual approach as with the tool-based approach when fixing test scripts, and we consequently recommend that organizations do not need to provide each tester with an expensive tool license to fix test scripts.

3.2.4 Precision and Recall

The qualities of both approaches are measured using two ratios: precision and recall. The precision ratio is computed as $\frac{CIF}{CIF+FP}$, and the recall ration is computed as $\frac{CIF}{TF}$, where TF is the total number of failures in test scripts. The precision ratio shows how mistaken participants are when analyzing failures in test scripts, and the recall is the ratio of correctly recovered failures.

The idea behind computing the precision and recall is to evaluate the difference between CIFs and FPs. If all identified failures are in fact real failures and not FPs, i.e., FP=0, then precision=1. If all identified failures are FP, i.e., CIF=0, then precision=0 and recall=0. The ratio recall is analogous to the recall parameter in information retrieval, which is the ratio of the number of relevant documents retrieved by a search divided by the total number of existing relevant documents, and the ration precision is analogous to the definition of precision in information retrieval, which is the number of relevant documents retrieved to the total number of documents retrieved by the search.

The graphs for precision and recall for the manual approach and QTP are shown in Figure 2(a) and Figure 2(b) correspondingly. Histogram of precision and recall values for these approaches are shown Figure 3 and Figure 4 correspondingly. From these graphs one can see that precision and recall is somewhat better with tool-based approach than with the manual approach.

3.2.5 Testing Relationships

We construct a contingency table to establish a relationship between CIFs and FPs for participants with and without testing experience as shown in Table 2. To test the null hypotheses H5 and H6 that the categorical variables CIFs and FPs are independent from the categorical variable testing experience, we apply two χ^2-tests, χ^2_{cif} and χ^2_{fp} for CIFs and FPs respectively. We obtain $\chi^2_{cif} = 21.3$ for $p < 0.0001$ and $\chi^2_{fp} = 11.5$ for $p < 0.0031$. The high values of χ^2 allow us to reject H10, H11, and H12 in favor of the alternative hypotheses suggesting that **there is statistically strong relationship between testing experiences of participants and the numbers of reported CIFs and FPs**. χ^2-tests reveal that QTP made a positive difference for inexperienced participants, while those with testing experience still performed better with the manual approach than with QTP.

4. Related Work

GAPs present special challenges to regression testing because the input-output mapping does not remain constant across successive versions of the software [22][20]. Numerous techniques have been proposed to automate regression testing. These techniques usually rely on information obtained from the modifications made to the source code. Some of the popular regression testing techniques include analyzing the program's control-flow structure [2], analyzing changes in functions, types, variables, and macro definitions [6][16], using def-use chains [14], constructing procedure dependence graphs [7][27], and analyzing code and class hierarchy for object-oriented programs [18][23]. These techniques are not directly applicable to black-box GUI regression testing, since regression information is derived from changes made to the source code.

To the best of our knowledge, there is no existing empirical study analyzing the cost-benefit of manual and tool-based approaches on maintaining test scripts of GAPs. Many similar studies investigating the cost-effectiveness of different techniques, (e.g., techniques used in test prioritization [26] [10] and test selection [25][12]) and factors (e.g., test frequency [17], test suite granularity [24], and context and life cycle [8]) applied on regression testing have been conducted. Like most empirical studies, we follow a well-known framework described in [29].

5. Conclusion

In practice, there is no consensus what approach testers should use to maintain test scripts. Test managers make their decisions ad hoc, based on their personal experience and perceived benefits of the tool-based approach versus the manual. We do not know of any comprehensive study that investigated these approaches and offered a statistically significant results.

In this paper we describe a case study with forty five professional programmers and test engineers to experimentally assess the tool-based approach for maintaining GUI-directed test scripts versus the manual approach. Based on the results of our case study and considering the high cost of the programmers' time and the lower cost of the time of test engineers, and considering that programmers often modify GAP objects in the process of developing software we recommend organizations to supply programmers with testing tools that enable them to fix test scripts faster so that these scripts can unit test software. The other side of our recommendation is that experienced test engineers are likely to be as productive with the manual approach as with the tool-based approach when fixing test scripts, and we consequently recommend that organizations do not need to provide each tester with an expensive tool license to fix test scripts.

References

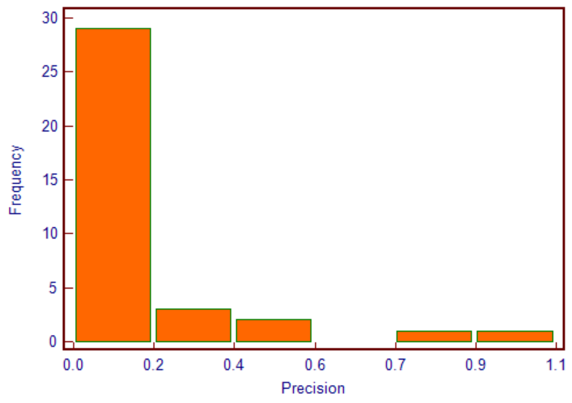
[1] *The Economic Impacts of Inadequate Infrastructure for Software Testing, Planning Report 02-3*. NIST, May 2002.

[2] T. Ball. On the limit of control flow analysis for regression test selection. In *Proceedings of ISSTA-98*, volume 23,2 of *ACM Software Engineering Notes*, pages 134–142, New York, Mar.2–5 1998.

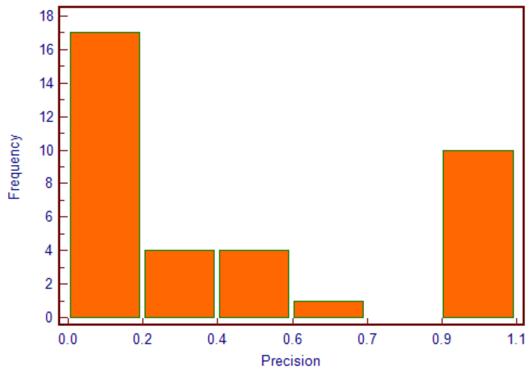
[3] M.-C. Ballou. Worldwide distributed automated software quality tools: 2007-2011 forecast and 2006 vendor shares: Dominating quality. *IDC Report 210132*, 1, Dec. 2007.

[4] S. Berner, R. Weber, and R. K. Keller. Observations and lessons learned from automated testing. In *ICSE '05*, pages 571–579, New York, NY, USA, 2005.

[5] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *FOSE '07: 2007 Future*

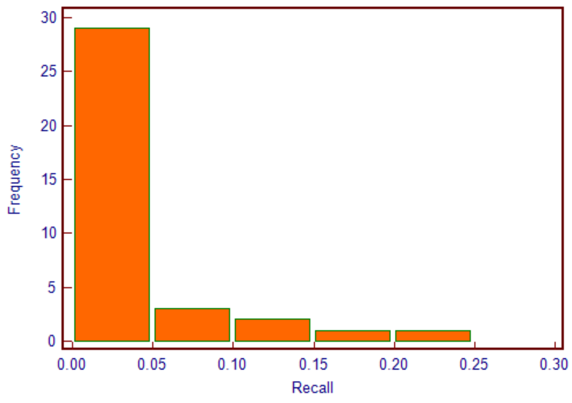


(a) Manual approach.

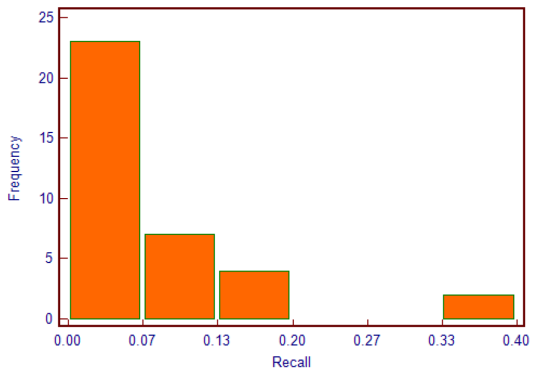


(b) Tool-based approach.

Figure 3. Histograms of precision distribution values.



(a) Manual approach.



(b) Tool-based approach.

Figure 4. Histograms of recall distribution values.

Test Exp	CIFs			FPs			MFs			Ratio	Ratio
	Man	QTP	Total	Man	QTP	Total	Man	QTP	Total	$\frac{CIFs}{FPs}$	$\frac{CIFs}{MFs}$
Yes	29	30	59	128	85	213	245	241	486	0.28	0.12
No	3	40	43	104	33	137	254	216	470	0.31	0.09
Total	32	70	102	232	118	350	499	457	956	0.29	0.11

Table 2. Contingency table shows relationship between CIFs, FPs, and MFs for participants with and without testing experience. The last two colums show the ratios of CIFs to FPs and CIFs to MFs that describe how precise and thorough participants are in findings CIFs.

of Software Engineering, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[6] J. Bible, G. Rothermel, and D. S. Rosenblum. A comparative study of coarse- and fine-grained safe regression test-selection techniques. *ACM Trans. Softw. Eng. Methodol.*, 10(2):149–183, 2001.

[7] D. Binkley. Reducing the cost of regression testing by semantics guided test case selection. In G. Caldiera and K. Bennett, editors, *ICSM*, pages 251–263, Washington, Oct. 1995.

[8] H. Do and G. Rothermel. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In *FSE'06*, pages 141–151, 2006.

[9] E. Dustin, J. Rashka, and J. Paul. *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley, September 2004.

[10] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *ISSTA'00*, pages 102–112, 2000.

[11] M. Fewster and D. Graham. *Software Test Automation: Effective Use of Test Execution Tools*. Addison-Wesley, September 1999.

[12] T. L. Graves, M. J. Harrold, J.-M. Kim, A. A. Porter, and G. Rothermel. An empirical study of regression test selection techiques. *ACM Trans. Softw. Eng. Methodol.*, 10(2):184–208, 2001.

[13] M. Grechanik, Q. Xie, and C. Fu. Maintaining and evolving gui-directed test scripts. In *ICSE '09*, New York, NY, USA, 2009.

[14] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Transactions of Software Engineering and Methodology*, 2(3):270–285, July 1993.

[15] C. Kaner. Improving the maintainability of automated test suites. *Software QA*, 4(4), 1997.

[16] J.-M. Kim and A. A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *ICSE*, pages 119–129, 2002.

[17] J.-M. Kim, A. A. Porter, and G. Rothermel. An empirical study of regression test application frequency. In *ICSE'00*, pages 126–135, 2000.

[18] D. C. Kung, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen. On regression testing of object-oriented programs. *The Journal of Systems and Software*, 32(1):21–31, Jan. 1996.

[19] K. Li and M. Wu. *Effective GUI Testing Automation: Developing an Automated GUI Testing Tool*. Sybex Publications, November 2004.

[20] A. M. Memon. *A Comprehensive Framework for Testing Graphical User Interfaces*. Ph.D. thesis, Department of Computer Science, University of Pittsburgh, July 2001.

[21] A. M. Memon and M. L. Soffa. Regression testing of GUIs. In *Proceedings of the ESEC and FSE-11*, pages 118–127, Sept. 2003.

[22] B. A. Myers. Why are human-computer interfaces difficult to design and implement? Technical report, Pittsburgh, PA, USA, 1993.

[23] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley. Chianti: a tool for change impact analysis of java programs. In *OOPSLA*, pages 432–448, 2004.

[24] G. Rothermel, S. G. Elbaum, A. G. Malishevsky, P. Kallakuri, and B. Davia. The impact of test suite granularity on the cost-effectiveness of regression testing. In *ICSE'02*, pages 130–140, 2002.

[25] G. Rothermel and M. J. Harrold. Empirical studies of a safe regression test selection technique. *IEEE Trans. Software Eng.*, 24(6):401–419, 1998.

[26] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: An empirical study. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, pages 179–188, Washington, DC, USA, 1999. IEEE Computer Society.

[27] R. A. Santelices, P. K. Chittimalli, T. Apiwattanapong, A. Orso, and M. J. Harrold. Test-suite augmentation for evolving software. In *ASE*, pages 218–227, 2008.

[28] R. M. Sirkin. *Statistics for the Social Sciences*. Sage Publications, third edition, August 2005.

[29] C. Wohlin, P. Runeson, and M. Host. *Experimentation in Software Engineering: An Introduction*. Springer, 2000.