Comparing Automated Visual GUI Testing Tools: An Industrial Case Study

Vahid Garousi
University of Luxembourg, Luxembourg
Wageningen University, Netherlands
vahid@vgarousi.com

Wasif Afzal
Mälardalen University, Västerås, Sweden
wasif.afzal@mdh.se

Adem Çağlar
HAVELSAN A.Ş., Ankara, Turkey
acaglar@havelsan.com.tr

İhsan Berk Işık
HAVELSAN A.Ş., Ankara, Turkey
ihsanberk@havelsan.com.tr

Berker Baydan
HAVELSAN A.Ş., Ankara, Turkey
bbaydan@havelsan.com.tr

Seçkin Çaylak
HAVELSAN A.Ş., Ankara, Turkey
scaylak@havelsan.com.tr

Ahmet Zeki Boyraz
HAVELSAN A.Ş., Ankara, Turkey
aboyraz@havelsan.com.tr

Burak Yolaçan
HAVELSAN A.Ş., Ankara, Turkey
byolacan@havelsan.com.tr

Kadir Herkiloğlu
HAVELSAN A.Ş., Ankara, Turkey
kherkiloglu@havelsan.com.tr

ABSTRACT

Visual GUI testing (VGT) is a tool-driven technique, which uses image recognition for interaction and assertion of the behaviour of system under test. Motivated by a real industrial need, in the context of a large Turkish software and systems company providing solutions in the areas of defense and IT sector, we systematically planned and applied a VGT project in this industrial context. The goal of the initial phase of the project was to empirically evaluate two well-known VGT tools (Sikuli and JAutomate) to help the company select the best tool for a given testing project. Our results show that both two tools suffer from similar test 'Replay' problems such as the inability to find smaller-sized images. The repeatability of test executions was better for JAutomate in case of one of the two software under test (SUT) while it was comparable for the other. In terms of test development effort, for both tools, there were high correlations with number of steps in test suites, however the effort is reduced if test code is reused. The study has already provided benefits to the test engineers and managers in the company by increasing the know-how in the company w.r.t. VGT, and by identifying the challenges and their workarounds in using the tools. The industrial case study in this paper intends to add to the body of evidence in VGT and help other researchers and practitioners.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**;

KEYWORDS

Graphical User Interface (GUI) Testing, Visual GUI Testing, Automated Testing Tools, Empirical Evaluation, Industrial Case Study

ACM Reference format:
Vahid Garousi, Wasif Afzal, Adem Çağlar, İhsan Berk Işık, Berker Baydan, Seçkin Çaylak, Ahmet Zeki Boyraz, Burak Yolaçan, and Kadir Herkiloğlu. 2017. Comparing Automated Visual GUI Testing Tools: An Industrial Case Study. In *Proceedings of 8th International Workshop on Automated Software Testing, Paderborn, Germany, September 4-5, 2017 (A-TEST'17),* 8 pages.
https://doi.org/10.1145/3121245.3121250

A-TEST'17, September 4-5, 2017, Paderborn, Germany

ACM ISBN 978-1-4503-5155-3/17/09…$15.00
https://doi.org/10.1145/3121245.3121250

1 INTRODUCTION

Visual GUI testing (VGT) is referred to as the third generation of Graphical User Interface (GUI)-based testing approaches [8]. It is a tool-driven technique, which uses image recognition for interaction with and assertions on the behaviour of a given System Under Test (SUT). The benefit of the technique is its flexibility of use with any GUI-based system. However, the technique suffers from false test results due to possible image recognition failure [6].

The company under study is a large Turkish software and systems company providing solutions in the areas of defense and IT sector. The company wanted to adopt automated VGT testing and was interested in comparing popular tools in the area. Motivated by this industrial need, we empirically evaluated two VGT tools (Sikuli [27] and JAutomate [8]) in a case study conducted at the said company. The purpose is to better understand the capabilities of the two tools and to help our case company evaluate the pros and cons associated with them.

Many practicing test engineers have challenges in proper and successful implementation of test automation, especially in the case of VGT [12]. According to multiple sources, e.g., [13], when VGT and GUI test automation are not planned, designed or implemented properly by test engineers, the efforts have led to disappointments and various negative outcomes (e.g., test artifacts becoming not useful or not even re-executable in the case of regression testing). While there are existing studies that compare VGT tools (see Section 2), since the original (raw) data of the related studies were

not available for us to re-analyze them in our new context, we decided to conduct our own evaluation of the two tools in our own industrial context. Thus, this study is a "conceptual extension" [14] of previous related works.

We aimed to answer the following main research question (RQ) in this study: How do the two tools compare when they are used to develop visual GUI test scripts? To assess the RQ systematically and w.r.t. various aspects, we divided it into three sub-research questions:

RQ1: How do the two tools compare in terms of quality of the "Record and Replay" features?
RQ2: How do the two tools compare in terms of robustness and repeatability?
RQ3: How do the two tools compare in terms of test development effort?

The remainder of this paper is structured as follows. Section 2 reviews the related work. We describe the case and company context in Section 3. Section 4 presents the research methodology. Empirical results and their analysis are presented in Section 5. In Section 6, we draw conclusions and discuss our ongoing and future works.

2 RELATED WORK

Several empirical studies have been conducted in the context of VGT and also specifically using and often comparing the existing tools in this area, e.g., Sikuli and JAutomate. In our literature search, we were able to identify the following relevant papers: [1–6, 6–9, 18, 19, 23–25]. It seems that more focus on this area has started since 2012. Most papers have done comparative studies of various tools and Sikuli seems to have been considered in most of the studies. In terms of domains of the SUTs, studies have mostly considered defense-related software systems (many in the context of Saab AB, a Swedish defense firm) and web applications. The scale of evidence has ranged from a few test cases to a large number of VGT test cases. Alegroth and Feldt seem to be the most active researchers in the area. A broad spectrum of issues have been addressed in this area, e.g., applicability of VGT in industrial contexts, advantages and disadvantages of VGT for system regression testing, whether VGT is an alternative or only a complement to manual testing, the largest problems with VGT, and the features that must be changed in the VGT tool (Sikuli) to make it more applicable. While many positive experiences have been reported, many of these studies have studied and reported obstacles in this area, e.g., challenges, problems and limitations [6, 8], disadvantages of VGT [9], typical problems [5], robustness of these tools, and limitations [18, 25].

As another line of work, there exist methodological frameworks for evaluating software testing tools and techniques, e.g., [26]. The authors of [26] argued that, for evaluating software testing tools and techniques, using the proposed framework: (1) software testing practitioners can define case studies through an instantiation of the framework, (2) results can be compared since they are all executed according to a similar design, and (3) a body of evidence for evaluating software testing tools and techniques will be initiated.

As another contribution of this work, we report empirical measurements on development and maintenance efforts of VGT and there is related work in this area too, e.g. [17] which compared three selected GUI test tools (Eggplant, QF-Test and TestComplete) in terms of fault detection capabilities, and the effort levels for development and maintenance of test scripts.

Last but not the least, as a related topic, choosing the right test automation tool is an active subject in both the practitioners and the research community. A recent Grey Literature Review (GLR) was reported on this topic in 2017 [22]. The main RQ assessed by that review study was: What are the criteria used (recommended) by practitioners for choosing test right automation tools? The study synthesised and classified the reported criteria from a large pool of sources into the following three categories: (1) task/team/ environment related issues (e.g., matching test requirements, team having necessary skills), (2) test-tool technical issues (e.g., tool stability, usability), and (3) test-tool external issues (e.g., tool cost, level of support).

3 CASE DESCRIPTION

The industrial context described in this paper involves HAVELSAN that is a large Turkish software and systems company providing global solutions in the areas of defense and IT sector. HAVELSAN develops a variety of systems such as naval combat systems, e-government applications, and reconnaissance surveillance systems.

An example system, developed and tested in this company, is called the HELSIM (helicopter simulator) system, which simulates the actual functioning of military helicopters. One type of testing for such a system involves GUI testing. The software running on these simulators are complex and large-scale (each usually having more than several million lines of code). The software has to interact with various external systems, e.g., hardware and pilot controllers onboard, and map systems. It goes without saying that quality in this context is of utmost importance since student pilots learn flying via these systems.

There is an independent testing group in the company's quality, test and process management directorate. The test group itself consists of five test teams:

- Test team for automation applications and image processing technologies
- Test team for ground support systems
- Test team for platform-stationed systems
- Test team for platform integration
- Test team for simulation and training systems

Essentially, each test team is dedicated for quality assurance (QA) of a major business unit of the company. In total, more than 40 test engineers work in the above five test teams. Almost all of the test activities conducted by the group are black-box testing, since the white-box (source-code-based) testing are conducted by the development team (before handing the software to the test team for black-box testing).

The company is keen to assess and improve its testing practices and one of the improvement objectives of the company is to improve testing of GUI-intensive systems. In our on-going collaboration, it was decided to focus on Visual GUI Testing (VGT) to evaluate its usefulness in HAVELSAN's context.

4 METHODOLOGY

For the empirical study, we used an approach generally similar to the related work as discussed above, i.e., [8, 9, 25].

We selected a team of test engineers in the company to act as the subjects of the study who would develop and maintain the test suites using two well-known VGT tools (Sikuli and JAutomate). Two SUTs were chosen as the objects under study (discussed in detail in the next sub-section).

4.1 Subjects and Objects Under Study

Five test engineers from the company's test group were chosen as the subjects. They were experienced manual testers. To ensure meaningful comparison of results and to minimize the selection bias, we ensured selecting test engineers with similar expertise levels (the managers' opinions were used for this purpose). The engineers had similar qualifications. All had at least a bachelor degree in computer or software engineering and have been working in the testing field for at least two years. They were chosen based on "convenience" sampling in the testing department of the company and since they were already assigned to work on testing these SUTs.

An experienced test manager directly managed the team of test engineers during the study. A senior researcher (the first author) also interacted with the team on a regular basis (once almost every two weeks) to ensure proper execution the study as per the plan.

As the objects of the study, we selected two software applications: (1) a prototype tool named PhoneBook, and (2) a real already-deployed electronic train ticket sales system called EYBIS (acronym in Turkish for: *Elektronik Yolcu Bilet Satış-rezervasyon Sistemi*). Lessons and the know-how's to be learnt from this study were planned to be utilized for using the test tools for testing other safety-critical systems of the company (e.g., a large GIS application for combat planning and monitoring). That activity is already underway as of this writing, but due to the classified nature of such applications, there are no plans to write papers from those data.

PhoneBook is a prototype phone directory tool, which has been developed in the company a few years ago and is mainly used to train the newly joined test engineers. PhoneBook has been developed in Java, and to simulate real-life training contexts, it has been evolved through multiple versions. In each version, several new features have been added and/or a subset of faults was fixed. EYBIS is a web-based application which is already installed and is in use by the Turkish state railways company and provides online train ticket sales and reservation to the public in Turkey. This web-based application has been developed using Java (Spring Framework) and JSF (Java Server Faces) technologies.

After selecting the subjects and objects, we assigned the tasks of developing and maintaining automated VGT test scripts among the subjects and objects as shown in Table 1. Five test engineers (TE1...5), who are actually among the co-authors of this paper, were allocated as subjects of the study to develop automated test suites for each of the two objects under study (SUTs) using each of the two test tools.

As discussed earlier in this sub-section, to ensure meaningful comparison of results and to minimize the selection bias, we ensured selecting test engineers with similar expertise. For example, as Table 1 shows, TE2 was assigned to develop automated test suites using Sikuli for EYBIS, while TE1 was assigned to develop automated test suites using JAutomate for the same SUT. As per the performance profiles of test engineers TE1 and TE2, as assessed by the managers in the previous projects, we ensured that their background levels (skill profiles) in this task were as similar as possible. For developing automated test scripts, we considered that one needs to have a good grasp of testing concepts, OO programming, and reasonable familiarity with the SUT and its requirements.

Table 1: Assignments of the Subjects and Objects in the Study (TE: Test Engineer)

Subjects	Objects (systems under test)			
	PhoneBook		EYBIS	
	Sikuli	JAutomate	Sikuli	JAutomate
TE1		×		×
TE2	×		×	
TE3	×			
TE4	×			
TE5		×		

For PhoneBook, the task of developing automated test scripts using Sikuli and JAutomate was divided among three and two testers, respectively, to share the workload, and to allow multiple data points for measures. Also, only TE1 and TE2 were familiar with EYBIS and had done manual testing of that SUT in the past. We thus only assigned them for automated testing of EYBIS. We assigned five TEs to PhoneBook to enable multiple data points for measurements. We would have liked to have equal number of TEs for testing PhoneBook using Sikuli and JAutomate (both three persons,), but as it is the case in real-world industrial empirical studies, the company context only had 5 TEs assigned for this project, and thus 3 and 2 TEs were assigned for the two tools. Also, we should mention that the latest versions of the VGT tools, at the time of this project, were used in this study: Sikuli version 1.1 and JAutomate version 10.1.

In this context, we were aware of the issue of " learning curve" in using a test tool by test engineers and that such factor would impact the measurements, e.g., of test development effort, in the case study. Other researchers have also talked about and considered the learning curve in empirical studies, e.g., [15]. To minimize this unwanted factor, we conducted a "warm-up" (self-training) period in which each subject (test engineer) developed a non-trivial number of test scripts in both tools in several simple Windows applications (e.g., Microsoft Paint and Notepad) before engaging in the experiment and starting the formal measurements for this study.

4.2 Test-Script Development Approach

For development of VGT test scripts, a set of already-developed manual system test suites were "transitioned" to automated tests. We should note that "transitioning" manual tests to automated tests is a popular practice in industry, e.g., [5]. For both of the two SUTs, there were written formal requirements. PhoneBook had 21 requirement items while EYBIS had 668 requirement items, out of which 260 were software related and the remaining (408) were hardware related. All the test team (five test engineers as shown in Table 1) developed test scripts to cover all the requirements of PhoneBook (details shown in Table 2). Conventional black-box testing techniques (e.g., equivalent classes and boundary value testing) were used for test-case design and 100% requirements was achieved by the test suites. The workload of developing 59 test

scripts for PhoneBook was evenly distributed among the team members, i.e., roughly 12 test scripts per person, for each tool. On average, each script tested about 4 requirements. Note that in Table 2 the Script Development Effort (SDE) values for PhoneBook are for its version 0.5, and the Script Maintenance Effort (SME) measures are for its version 0.7.

For EYBIS, out of 260 software requirement items, to keep our efforts manageable, a test manager from the company (a co-author of this paper) prioritized the requirements as per his expert opinion and experience, and selected a subset of 85 most critical requirements (core functions) for test-suite development, e.g., requirements corresponding to the following use-cases: ticket sales, ticket reservation, change reservation, and cancel reservation. All those 85 most important requirements were previously covered in five manually-written test suites, as listed in Table 3: *TS-TicketSales*, *TS-TicketReservation*, *TS-ChangeReservation*, *TS-ChangeReservationToSales*, and *TS-CancelReservation*. Similar to PhoneBook, for the development of manually-written test suites, conventional black-box testing techniques (e.g., equivalent classes and boundary value testing) were used for test-case design. Also similar to the case of PhoneBook, we "transitioned" [5] all of the manual test suites into five automated test suites, without changing the test-case designs, nor the number of test cases. List of the test suites for EYBIS and their relevant metrics are shown in Table 3.

5 EMPIRICAL RESULTS

We discuss next the empirical results for each of the RQs 1 to 3.

5.1 RQ 1-Quality of the 'Record and Replay' Features

'Record and replay' (also called playback) has been a well-known feature of GUI testing tools and has recently started to also be adopted in the VGT tools. To assess the quality of the 'Record and Replay' features, after the test-suite development, we asked the team members to provide their opinions for the following four questions for each of the two tools:

(1) How powerful is the tool's Record feature (in terms of number of features offered)?
(2) How powerful is the tool's Replay feature?
(3) How usable and efficient it is to use the tool's Record feature?
(4) How usable and efficient it is to use the tool's Replay feature?

To ensure rigor in the voting process, we used the Delphi method, which is a structured communication technique, and a systematic, interactive method to get experts opinions. A 5-point Likert scale (1-5) was used. Table 4 shows the voting results. We believe that the "votes" of TEs are representative of what one might expect from a typical test engineer, since the five team members (TEs) and, as per our experience in working with many other test engineers in the software industry, they are good representatives of industrial practitioners, given their test experience, qualifications (all had the ISTQB certifications), and experience in tool usage. Only JAutomate has the 'Record' feature and Sikuli does not support it. That is why the first question has received 0 (N/A) for Sikuli. There were some challenges in some of the factors (e.g., question #1 for JAutomate) and that's why it was assessed as score of 4 (one less than the perfect score of 5). Note that, by reviewing Table 4, the reader may think, since Sikuli lacks the recording functionality, thus this question is biased in JAutomate's favor. However, note that the RQs were raised (developed) prior to any use of the tools and knowing their features. Note that both "record and replay" features go hand in hand usually for GUI testing tools. Thus it is natural to expect a tool to provide both. Sikuli provided the replay feature only. Thus it was still proper to compare the tools w.r.t. both features. Furthermore, during all of our assessments, we ensured that no bias was placed towards any of the two tools. Our goal was simply to objectively assess the two tools.

5.2 RQ 2-Robustness and Repeatability of Test Executions Across Many Runs

According to the foundations of software testing, e.g., [21], the execution of a test case should be both robust and repeatable. This denotes that the test case's outcome should be the same across multiple runs, given that the system state and the environment do not change. However, we realize that this is sometimes hard to ensure in reactive, real-time and distributed systems [10].

We assessed robustness and repeatability across many runs as follows: (1) stability (robustness) of image recognition across many runs, and (2) repeatability of test outcomes, i.e., do test scripts always give the same test outcome (pass or fail) or are there intermittent issues? The empirical results for robustness and repeatability of test executions for the two SUTs are shown in Tables 5 and 6.

PhoneBook test suites developed in both of the tools were executed 250 times. Note that we were aware that, after each execution of each test case, we had to reset the test environment, which is well known as test "teardown" phase in the community. Also, when starting test execution, we put in place proper test setups to prepare the test environment for each test case. In fact, we followed the well-known four-phase test pattern [20] (setup, exercise, verify and teardown) in development of all automated test scripts.

As we can see in Tables 5 and 6, the execution times are quite comparable: 280 seconds for each iteration of the Sikuli test-suite, and 223 seconds for the JAutomate one. There were issues with stability (robustness) of image recognition across many runs for the case of Sikuli as the test-suite gave a failure in the midst of the 250 iterations, i.e., execution #156, #136, and #23 in three consecutive trials. As we conducted root-cause analysis of the failure, the reason was RAM fill-up and thus, we believe that the Sikuli tool has some defects in terms of memory management in high number of test executions. The PC we executed the tests with had the following specifications: Windows 7 Enterprise 64 bit, CPU: Intel(R) Core(TM) Quad Q9550 2.83GHz, RAM: 3 GB. The exact error message from Sikuli was as follows (note that the screen capture file 1441040638074-2.png was in the images folders, but the tool could not find it):

```
[error] script [run250times] stopped with error in
line 16
[error] FindFailed (cannot find 1441040638074-2.png in
R[0,0 1152x864]@S(0))
[error] --- Traceback --- error source first line:
module (function) statement 22: AdresEklemeSayisi (
<module> ) paste("1441040638074-2.png", "cep10
[error] --- Traceback --- end --------------
```

Table 2: List of Automated Test Suites for PhoneBook (SDE: Script Development Effort in Minutes, SME: Script Maintenance Effort)

No.	Name of test suite (requirement groups under test)	No. of test cases	Test scripts for version 0.5						Test scripts for version 0.7					
			Sikuli			JAutomate			Sikuli			JAutomate		
			LOC	SDE	norm SDE	LOC	SDE	norm SDE	LOC	SME	norm SME	LOC	SME	norm SME
1	TS-LookupContact	5	29	20	0.69	51	31	0.61	29	10	0.34	51	1	0.02
2	TS-AddContact	7	62	28	0.45	82	21	0.26	62	19	0.31	82	3	0.04
3	TS-OtherPhoneTypes	8	51	18	0.35	58	32	0.55	49	10	0.20	58	5	0.09
4	TS-AddAddress	5	54	25	0.46	45	30	0.67	54	15	0.28	46	1	0.02
5	TS-OtherAddressTypes	7	51	15	0.29	47	15	0.32	49	8	0.16	47	5	0.11
6	TS-AddContactInvalidCases	21	163	120	0.74	240	82	0.34	160	40	0.25	239	10	0.04
7	TS-ReadOnlyAndEdittableFields	6	29	10	0.34	32	18	0.56	29	5	0.17	32	1	0.03
	Total	59	439	236	0.54	555	229	0.41	432	107	0.25	555	26	0.05

Table 3: List of Automated Test Suites for EYBIS and Their Statistics

No.	Name of test suite (requirement groups under test)	No. of test cases	Test scripts					
			Sikuli			JAutomate		
			LOC	SDE	norm SDE	LOC	SDE	norm SDE
1	TS-TicketSales	16	112	75	0.67	196	187	0.95
2	TS-TicketReservation	21	80	72	0.90	113	15	0.13
3	TS-ChangeReservation	42	98	48	0.49	117	24	0.21
4	TS-ChangeReservationToSales	14	88	65	0.74	115	17	0.15
5	TS-CancelReservation	9	33	55	1.67	36	17	0.47

Table 4: Comparison of the Two Tools w.r.t. Quality of the 'Record and Replay' Features

Aspects	Questions	Sikuli	JAutomate
Features offered	How powerful is the tool's Record feature?	0 – (N/A) No record feature	3 – In some cases, maintenance (correction) of test code can be required after recording, e.g., during recording, the tool generates annoying test code pieces such as unnecessary mouse clicks, drag and drop actions or unnecessary waits.
	How powerful is the tool's Replay feature?	4 – Sometimes, the tool cannot find the images if the image size is too small. For example, in the EYBIS system, the ticket number could not be detected by Sikuli.	4 – Sometimes, the tool cannot find the images if the image size is too small.
Usability & Efficiency	How usable and efficient it is to use the tool's Record feature?	0 – (N/A) No record feature	4 – During recording, the mouse freezes sometimes on the screen. We could not find the root cause of this issue.
	How usable and efficient it is to use the tool's Replay feature?	4 – When the script is executed in another computer (different than the PC where it was developed in), the tool can click wrong coordinates due of change of the screen resolution. To prevent such issues, we have made it a common practice to specify the test resolution on top of test scripts.	4 – The same problem as Sikuli

In terms of repeatability of test outcomes across many runs, PhoneBook JAutomate test suites were all repeatable as all test cases passed in all the test iterations.

For EYBIS, our goal was to execute the test suites developed in each of the tools for 100 times. However, mainly due to complexity of the SUT, there were issues, which led to abnormal termination of test executions as discussed next. For example, we observed that, during Sikuli execution of EYBIS test scripts, test execution would stop after 9 times. While Sikuli was running the "*TS-CancelReservation*" test suite, it was unable to find the image corresponding to "*Cancel reservation*". Therefore, Sikuli stopped running the rest of test cases. In another try to execute EYBIS test scripts for 100 times in Sikuli, Sikuli stopped in the 5th execution in the midst of the "*TS-Reservation*" test suite, since there was no journey left for the particular time of departure mentioned in the test suite (6:00 PM), and thus, the test tool could not pass this step. This denotes the importance and sensitivity of test suites to business logic details, e.g., time of train departure in this case.

Also, for repeatability analysis of EYBIS test suites writing JAutomate, we aimed at executing the test suites for 10 times (less than 250 times as it was the case for PhoneBook, due to larger size of tests). 9 of those test iterations passed, while one iteration failed. In analyzing the root-cause for the test failure, we found that during execution of test suite *TS-ChangeReservation*, a message box (with caption "Please wait..." ("Lütfen bekleyiniz...") as shown in Figure 1) appeared but did not disappear in a timely manner, thus causing the time-out status and then leading to failure in the test execution. In summary, out of 10 test iterations that we aimed for, coincidentally in the case of both tools, 9 test iterations passed, while only 1 test iteration failed. Thus, there was no statistically-significant differences between the two tools w.r.t. robustness and repeatability of test executions.

Table 5: Robustness and Repeatability of Test Executions (SUT: PhoneBook)

Metric	Metric	JAutomate
Test iterations aimed for	250	250
Time duration of test runs	280 seconds (4.6 minutes) for each iteration 12, 13 hours for 156 iterations	223 seconds for each iteration 15.4 hours for 250 iterations
# image recognition failures	The execution halted and image recognition failed in execution #156, #136, and #23 in three consecutive trials due to RAM fill-up incident (detail in the text).	None
# tests passed	All tests had passed until the halting execution.	All
# tests failed	No failed tests until the halting execution	None
Remarks (notes)	The entire batch of 250 iterations could not be finished due to full RAM fill-up and PC halt.	The test log (consisting mostly of screenshots) generated by the tool consumed about 50 GB of disk space.

Table 6: Robustness and Repeatability of Test Executions (SUT: EYBIS)

Metric	Metric	JAutomate
Test iterations aimed for	10	10
Time duration of test runs	About 260 seconds for each iteration. 40 minutes in total	About 231 seconds for each iteration. 29 minutes in total
# image recognition failures	None	None
# tests passed	9 test cases	9 test cases
# tests failed	1 test case	1 test case
Remarks (notes)	Due to database connection problems in the SUT, robustness testing cannot be completed. Robustness tests were executed for only 3 times.	Due to database connection problems in the SUT, robustness testing cannot be completed. Robustness tests were executed for only 3 times.

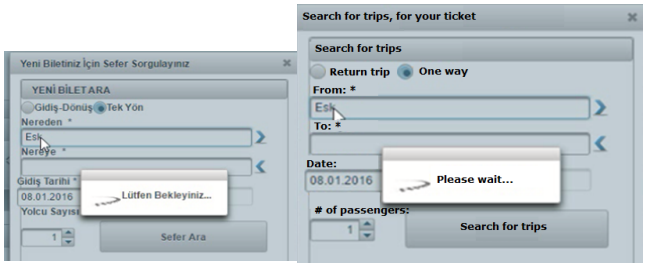

Figure 1: The message "Please wait..." ("Lütfen bekleyiniz..." in Turkish) appearing in the GUI of EYBIS and not going away, causing the time-out status of the test execution (the right UI is the English translation of the left UI)

5.3 RQ 3-Test Development Effort

Recall that the list of automated test suites for each SUT and their relevant metrics (e.g., test development effort in minutes) were

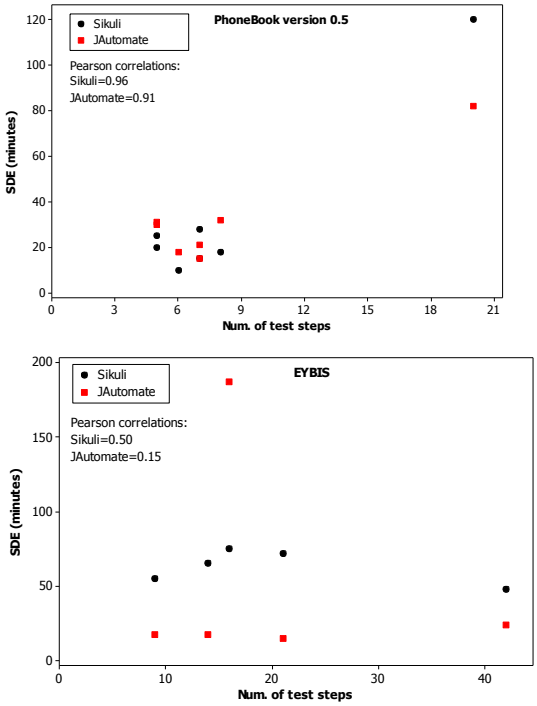

Figure 2: Script development effort (SDE) versus # of test cases

shown in Tables 2 and 3. We visualize those data based on two perspectives, i.e., script development effort (SDE) versus # of test cases and test LOC versus # of test cases, in Figure 2 and Figure 3 (along with Pearson correlation values), and discuss the observations next. The former perspective (Figure 2) denotes the actual effort in terms of cost of "transitioning" manual to automated tests and the latter (Figure 3) denotes the test suite size (as measured by LOC) when conducting that transition.

As Figure 2 shows, the SDE has high correlations with the number of test cases in both tools for PhoneBook version 0.5 (trends are similar for version 0.7 as well). This means that, as one would expect, a test engineer could expect to invest more effort into test development as the number of test cases increase in a test suite. However, for the case of EYBIS, the correlations are low to almost non-existent (for JAutomate, the value is 0.15). The reason is that when writing tests for one of the EYBIS test suites (*"TS-TicketSales"*) using JAutomate, we intentionally decided to follow test-code reuse which is a type of test patterns (best practice), recommended by many researchers and practitioners [11, 16] to be used in test automation. The idea was to compare the two treatments: having test code-reuse and the other treatment as not having test code-reuse. Thus, as a benchmark, for writing EYBIS tests using Sikuli, we did not (actually could not) apply test-code reuse, since Sikuli did not support calls to user-defined functions in test code-base.

To apply test code-reuse when writing EYBIS tests using JAutomate, some effort (187 minutes) was spent to develop several utility test functions, e.g., selection of passenger seats, which would be called (reused) later on by other test suites. As we will discuss below,

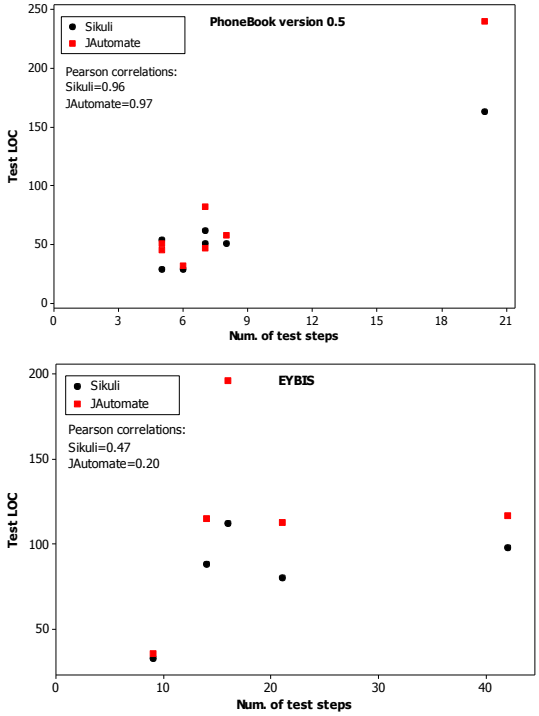

Figure 3: Test LOC vs. # of test cases

this best practice provided benefits (cost savings) in writing test suites. To show the impact of modularity and code reuse in our test suites, Figure 4 shows a call-graph depicting the call relationships among different test-script functions of JAutomate test suite for EYBIS. We can see in the call graph that five different test suites (starting with `TS-*`) are sharing several utility-like test functions, e.g., `dateAndTime`, `singlePassenger`, and `cancelReservation`.

When we added all the values for the case of EYBIS in Figure 3, the sum of SDE values for Sikuli and JAutomate test suites were 315 and 260 minutes, respectively. Although the lower total SDE value of JAutomate test suites may be partially due to test-code reuse (as discussed above), we cannot for sure say that this was the only factor as other factors could have been involved as well, e.g., the ease of use of writing test code in either of the tools. This aspect needs further future investigations.

In terms of test LOC versus # of test cases (Figure 3) we can see that, again, for the case of PhoneBook, there are high correlations in both test tools, meaning that a test engineer should expect to develop more test code (as measured by its LOC) as the number of test cases increase in a test suite. Again, for the case of EYBIS, the situation is different since test-code reuse has been utilized.

5.4 Limitations and Threats to Validity

In this section, we discuss the limitations and potential threats to the validity of our study and the steps we have taken to minimize or mitigate them. A threat to internal validity in this study lies in the selection bias (i.e., related to the test engineers who participated in the study). To ensure meaningful comparison of results and to minimize the selection bias, we ensured selecting test engineers with similar expertise levels (the managers' opinions were used for this purpose). Also, we were aware of the issue of 'learning curve' in using a test tool by test engineers and that such factor would impact the measurements, e.g., of test-development effort, in the case study. Again, as discussed earlier in the paper, to minimize this unwanted factor, we conducted a 'warm-up' (self-training) period in which each subject (test engineer) developed a non-trivial number of test scripts in both of the tools in several simple Windows applications before engaging in the experiment and starting the formal measurements. Another internal validity threat is the low number of subject involved in this study (only five test engineers). Conducting further studies with more number of test engineers would be recommended.

We would note at the outset of this study that only two SUTs and two VGT tools were studied in this paper and thus the results cannot and should not be generalized to other SUTs and test tools. However, our results contribute to the already-expanding body of empirical evidence in the area of VGT and, thus, we believe that other researchers and practitioners will benefit from our results.

6 CONCLUSIONS

Motivated by a real industrial need, in the context of a large Turkish software and systems company, this study empirically evaluated two VGT tools. The goal was to compare the two tools (Sikuli and JAutomate), for the purpose of determining a suitable tool for VGT in the company from the point of view of software test engineers and managers in the company. We conducted a comparison between the two tools in terms of their record and replay feature, robustness, repeatability and test development effort. Our results show that, while for some comparison criteria (e.g., quality of the 'record and replay' features), one tool ranked higher than the other, it ranked lower for some other criteria. The study has provided a lot of benefits to the test engineers and managers in the company by increasing the know-how in the company w.r.t. VGT and by identifying the challenges and their workarounds in using the tools and we hope that it will also benefit other researchers and practitioners. The results of this study has also already been useful for our other industry partners by adopting and reusing our comparison criteria and approach in comparing other test tools in other contexts. As for the issue of comparing the two tools, our results recommended to us in the company that JAutomate is slightly better for our needs, however both tools are kept for potential usage in current and future testing projects, given their strengths and weaknesses as identified in this empirical study.

In future, we plan to conduct the following investigations: (1) assessing the advantages and disadvantages of VGT (there has been some existing work in this area, e.g., [9]), (2) comparing the VGT tools in terms of fault detection effectiveness.

ACKNOWLEDGMENTS

Vahid Garousi was partially supported by the National Research Fund, Luxembourg FNR/P10/03. Most of the work reported in this paper was conducted while the first author was with Hacettepe University. Wasif Afzal was partly supported by the Knowledge Foundation through grant 20160139 (TestMine) and 20130085 (TOCSYC).

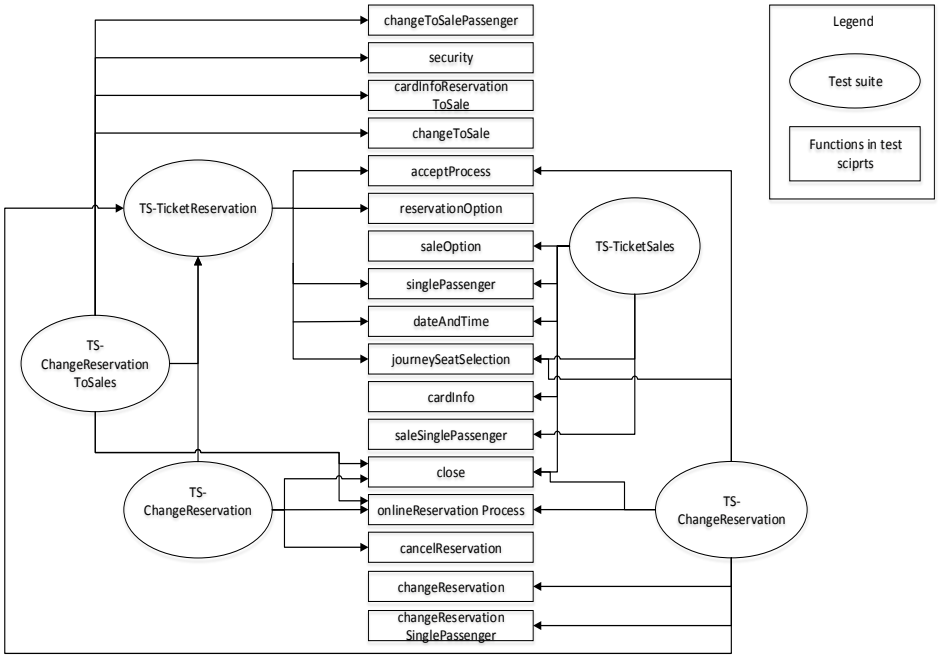

Figure 4: Call (dependency) graph showing the call relationships among test-script functions of JAutomate test suite for EYBIS

REFERENCES

[1] Emil Alégroth. 2013. *On the Industrial Applicability of Visual GUI Testing.* Master's thesis. Chalmers University of Technology, Sweden.

[2] Emil Alégroth. 2013. Random Visual GUI Testing: Proof of Concept. In *Proceedings of the 2013 International Conference on Software Engineering and Knowledge Engineering.*

[3] Emil Alégroth. 2015. *Visual GUI Testing: Automating High-level Software Testing in Industrial Practice.* Ph.D. Dissertation. Chalmers University of Technology, Sweden.

[4] Emil Alégroth and Robert Feldt. 2014. *Industrial Application of Visual GUI Testing: Lessons Learned.* Springer International Publishing, Cham, 127–140.

[5] Emil Alégroth, Robert Feldt, and Helena H. Olsson. 2013. Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation.*

[6] Emil Alégroth, Robert Feldt, and Lisa Ryrholm. 2015. Visual GUI Testing in Practice: Challenges, Problems and Limitations. *Empirical Software Engineering* 20, 3 (2015), 694–744.

[7] Emil Alégroth, Zebao Gao, Rafael A.P. Oliveira, and Atif Memon. 2015. Conceptualization and Evaluation of Component-Based Testing Unified with Visual GUI Testing: An Empirical Study. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST).*

[8] Emil Alegroth, Michel Nass, and Helena H. Olsson. 2013. JAutomate: A Tool for System- and Acceptance-test Automation. In *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation.* IEEE Computer Society.

[9] Emil Borjesson and Robert Feldt. 2012. Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation.* IEEE Computer Society.

[10] Vahid Garousi, Lionel C. Briand, and Yvan Labiche. 2008. Traffic-aware Stress Testing of Distributed Real-time Systems Based on UML Models Using Genetic Algorithms. *Journal of Systems and Software* 81, 2 (2008), 161–185.

[11] Vahid Garousi and Michael Felderer. 2016. Developing, Verifying, and Maintaining High-Quality Automated Test Scripts. *IEEE Software* 33, 3 (2016), 68–75.

[12] Vahid Garousi, Michael Felderer, Marco Kuhrmann, and Kadir Herkiloğlu. 2017. What Industry Wants from Academia in Software Testing?: Hearing Practitioners' Opinions. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering.* ACM.

[13] Vahid Garousi and Mika V. Mäntylä. 2016. When and What to Automate in Software Testing? A Multi-vocal Literature Review. *Information and Software Technology* 76, C (2016), 92–117.

[14] Natalia Juristo and Omar S. Gómez. 2012. Empirical Software Engineering and Verification. Springer-Verlag, Chapter Replication of Software Engineering Experiments, 60–88. http://dl.acm.org/citation.cfm?id=2184075.2184077

[15] C. F. Kemerer. 1992. Now the Learning Curve Affects CASE Tool Adoption. *IEEE Software* 9, 3 (1992), 23–28.

[16] M. Khalili. [n. d.]. Maintainable Automated UI Tests. https://code.tutsplus.com/articles/maintainable-automated-ui-tests–net-35089. ([n. d.]). Last accessed: July 2017.

[17] Antonia Kresse and Peter M. Kruse. 2016. Development and Maintenance Efforts Testing Graphical User Interfaces: A Comparison. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation.* ACM.

[18] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. 2014. *Visual vs. DOM-Based Web Locators: An Empirical Study.* Springer International Publishing, Cham, 322–340.

[19] Grischa Liebel, Emil Alégroth, and Robert Feldt. 2013. State-of-Practice in GUI-based System and Acceptance Testing: An Industrial Multiple-Case Study. In *Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications.*

[20] Gerard Meszaros. 2006. *XUnit Test Patterns: Refactoring Test Code.* Prentice Hall PTR, Upper Saddle River, NJ, USA.

[21] Glenford J. Myers and Corey Sandler. 2004. *The Art of Software Testing.* John Wiley & Sons.

[22] Päivi Raulamo-Jurvanen, Mika Mäntylä, and Vahid Garousi. 2017. Choosing the Right Test Automation Tool: A Grey Literature Review of Practitioner Sources. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering.* ACM.

[23] P. J. D. Sanmartin. 2014. *Case Study to Evaluate the Feasibility of using of "Sikuli" in the Company Sulake.* Master's thesis. Universitat Politecnica de Valencia, Spain.

[24] I. Singh and B. Tarika. 2014. Comparative Analysis of Open Source Automated Software Testing Tools: Selenium, Sikuli and Watir. *International Journal of Information & Computation Technology* 4 (2014), 1507–1518.

[25] J. Sjöblom and C. Strandberg. 2014. *Automatic Regression Testing using Visual GUI Tools.* Master's thesis. Department of Computer Science and Engineering, Chalmers University of Technology & University of Gothenburg, Gothenburg, Sweden.

[26] T. E. J. Vos, B. Marin, M. J. Escalona, and A. Marchetto. 2012. A Methodological Framework for Evaluating Software Testing Techniques and Tools. In *2012 12th International Conference on Quality Software.*

[27] Tom Yeh, Tsung hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI Screenshots for Search and Automation. In *Proceedings of the ACM Symposium on User Interface Software and Technology.*