Abstract—Visual GUI testing (VGT) is an emerging technique that provides software companies with the capability to automate previously time-consuming, tedious, and fault prone manual system and acceptance tests. Previous work on VGT has shown that the technique is industrially applicable, but has not addressed the real-world applicability of the technique when used by practitioners on industrial grade systems. This paper presents a case study performed during an industrial project with the goal to transition from manual to automated system testing using VGT. Results of the study show that the VGT transition was successful and that VGT could be applied in the industrial context when performed by practitioners but that there were several problems that first had to be solved, e.g., testing of a distributed system, tool volatility. These problems and solutions have been presented together with qualitative, and quantitative, data about the benefits of the technique compared to manual testing, e.g., greatly improved execution speed, feasible transition and maintenance costs, improved bug finding ability. The study thereby provides valuable, and previously missing, contributions about VGT to both practitioners and researchers.

Keywords—Visual GUI testing; Test Automation; Test Maintenance; Empirical; Industrial case study.

I. INTRODUCTION

To date, there are no industrial case studies, from the trenches, that visual GUI testing (VGT) works in industry when used by practitioners, nor data to support the long-term viability of the technique. In our previous work, we have shown that VGT is applicable in industry, even for testing of safety-critical software [1]. However, previous work has been essentially driven by researchers, e.g., they applied VGT techniques, compared the resulting test cases to earlier manual efforts, and then collected feedback and refinements from the industrial practitioners. There is a risk that this type of research does not consider all the complexities and problems seen by practitioners when actually applying a technique in practice. Furthermore, researcher driven studies are often smaller in scale and cannot evaluate longer term effects such as maintenance and refactoring of the test scripts or effects on, and of, changes to the system under test (SUT). Hence, there is still a gap in VGT’s body of knowledge regarding if the technique is applicable when performed by industrial practitioners in a real world development context.

In this paper we aim to bridge this gap by presenting an industrial case study from a successful project, driven entirely by industrial practitioners, with the goal to transition into VGT at the company Saab AB, subdivision security and defense solutions (SDS). The company chose VGT because of its ability to automate high system-level test cases, which previous automation techniques, e.g., unit testing [2], [3] and record and replay (R&R) [4]–[6], have had shortcomings in their ability to achieve. High system-level tests developed with automated unit tests have become both costly and complex, thereby spurring a discussion if the technique is applicable for anything but the low system-level testing, for which it was developed [7]. Furthermore, R&R techniques, which were developed for automation of system-level tests, are instead limited by being fragile to GUI layout and API change. Limitations that in the worst case has caused entire automated test suites to become inept [8]. Hence, the previous techniques have shortcomings in terms of flexibility, simplicity and robustness to make them long-term viable.

However, in this case study we show that VGT can overcome these limitations. Hence, showing that VGT has the capability to automate and perform industrial grade test cases that previously had to be performed manually, with equal or even greater fault finding ability, at lower cost. Capability provided by the technique’s use of image recognition that, in combination with scenario based scripts, allow VGT tools to interact with any graphical object shown on the computer monitor, i.e., allowing VGT scripts to emulate a human user. In addition, the study presents the practitioner’s views on using the technique, such as benefits, but also problems and limitations, when performed with the open source tool Sikuli [9]. Consequently, this work shows that VGT works for testing of real-world systems when performed by practitioners facing real-world challenges such as refactoring and maintenance of the SUT. The specific contributions of this work therefore include,

1) An account on how the transition to VGT was successfully conducted by industrial practitioners for a real-world system.
2) The industrial practitioners experiences and perception on the use of VGT.
3) Qualitative and quantitative data on costs, challenges, limitations and solutions that were identified during the
VGT transition project.

Together these contributions can help both industrial practitioners in practice and guide researchers in further advancing the state-of-practice and state-of-the-art in VGT.

The continuation of this paper is structured as follows. In Section II related work is presented, followed by Section III that presents the case study methodology and data collection. Section IV then presents the results and analysis of the study that are then discussed in Section V. Finally the paper is concluded in Section VI.

II. RELATED WORK

The concepts of using image recognition for GUI interaction is quite old and has been evaluated in a considerable body of knowledge. Work on using image recognition for GUI automation can be traced back to the early 90s, e.g. Potter [10] and his computer macro development tool, Triggers. Other early work in this area include the work of Zettlemoyer and Amant that used image recognition in their tool VisMap, which was used to automate the interaction with a visual scripting program as well as the game Solitaire [11]. However, this work focused on using image recognition for automation which we differentiate from testing since not all tools developed for GUI automation are intended for testing and vice versa.

The body of knowledge on using GUI interaction for testing is also considerable, e.g. shown by Adamoli et al. [4] in their paper on automated performance testing that covers 50 papers on automated GUI testing. Automated GUI testing can be performed with different techniques but the most common approach is referred to as record and replay (R&R) [4]–[6]. R&R consists of two steps. First a recording step where user input, e.g. mouse and keyboard interaction, to the system under test (SUT) is recorded in a script. In the second step, the recorded script can automatically be replayed for regression testing purposes. Different R&R tools record SUT interaction on different levels of GUI abstraction where the most common are on GUI bitmap level, i.e. using coordinates, or GUI widget level, i.e. using software references to buttons, textfields, etc. However, both approaches suffer from limitations that affect their robustness. Coordinate based R&R has the limitation that it is sensitive to GUI layout change whilst being robust to SUT code change. Widget based R&R, in contrast, is sensitive to SUT API or code structure change [8], but is instead robust to GUI layout change.

Image recognition based GUI testing with scenario based scripts, which we refer to as visual GUI testing (VGT), does not suffer from these limitations but it is only recently that the technique started to emerge in industry. One plausible explanation to this phenomenon is that the image recognition is performance intensive and it is not until now that the hardware has become powerful enough to cope with the performance requirements. VGT is a tool-supported technique, e.g. by Sikuli [9], EggPlant, etc., which conducts testing through the top GUI bitmap level of a SUT, i.e. the actual bitmap graphics shown to the human user on a computer monitor. Hence, scenario based VGT scripts can emulate a human user and can therefore also test all applications, regardless of implementation or platform, e.g. web, desktop, mobile. In most VGT tools the scenarios have to be developed manually, but there are also tools, e.g. JAutomate, which has record and replay functionality. Typical VGT scripts are executed by first providing the SUT with input, i.e. clicks or keyboard input, after which the new state of the system is observed, using image recognition, and compared to some expected output, followed by a new sequence of inputs, etc. In contrast to previous GUI testing techniques, VGT is impervious to GUI layout change, API or even code changes. However, VGT is instead sensitive to GUI graphics changes, e.g. changes in graphics size, shape or color.

Another approach to GUI testing is to use models, e.g. using finite state machines to generate test cases [12], [13]. These models generally have to be constructed manually, but automatic approaches, e.g. GUI ripping proposed by Memon [14], also exist. The benefit with GUI ripping is that it mitigates the extensive costs related to model creation. Costs that originate in the complexities of developing a suitable model. The limitation of this approach is that it is dependent on the SUT implementation, e.g. development language.

The area of GUI interaction based testing and automation is therefore quite broad but still limited in regards of empirical studies in real-world contexts with industrial grade software systems. R&R tools have been compared [4] and evaluated in industry, for both system- and acceptance-test automation, but, to our best knowledge, it is only our own work that evaluates VGT in an industrial context [1]. Our previous work is however limited since it was conducted only for a small set of real-world test cases and since the VGT automation was performed by researchers rather than practitioners. Hence, the body of knowledge on VGT, to the authors best knowledge, is absent of industrial case studies reporting real-world use of the technique.

Most research on GUI based testing focuses on system testing. However, acceptance testing is an equally important, valid and plausible test aspect to consider, i.e. tests were requirements conformity is validated through end user scenarios performed regularly on the SUT [15]. Scenario based acceptance tests do however distinguish themselves from system tests by including more end user specific interaction information, i.e. how the system will be used in the end users’ domain. Automated acceptance testing has also been a subject of much research, which has resulted in both frameworks and tools, including research into GUI interaction tools [16]. However, to the authors’ best knowledge, only our previous work has nudged on the subject of using VGT for acceptance testing.

III. RESEARCH METHODOLOGY

The following section presents the research company where the VGT transition was performed as well as the research methodology used to collect data during the case study.
The case study presented in this paper was conducted in collaboration with, and at, the Swedish company Saab AB, subdivision SDS, in the continuation of this paper referred to as Saab. The study was conducted at the company because they had taken the initial steps towards transitioning into VGT to automate their current manual testing, which presented an opportunity to collect data to bridge the current gap regarding VGT’s real world applicability. Figure 1 visualizes the stages of the case study, which will be presented in more detail in the following section based on the guidelines for reporting case studies presented by Runeson and Höst, 2009 [17].

Saab develops military control systems for the Swedish military service provider on behalf of the Swedish military forces. The system is, when deployed in the field, distributed between several mobile nodes and provides the ability to map the position of friendly and hostile forces on the battlefield and share this information among the nodes. Hence, the core functionality of the system relies on a map visualization, provided by a map engine, which allows the user to place symbols representing military units onto the map. Due to the systems intended use it is considered both safety and mission critical. In addition, the system is developed for a touchscreen systems intended use it is considered both safety and mission critical. In addition, cost and time metrics were collected using unstructured open interviews with the testers that were driving the VGT transition at the company. Unstructured open interviews was chosen because very little was known about the company at this stage of the study. In addition, several documents were acquired that could provide further information about the manual test suite and the SUT.

In the second stage of the case study, which was four calendar months, a communication process was followed to allow the testers driving the VGT transition and the research team to exchange information on a regular basis, i.e. the row named ‘Case study’ in Figure 1. The communication process was put in place for two reasons. First because the project was to be driven by the testers at the company rather than the research team; the latter deliberately distanced themselves from the project in order for all collected data to genuinely portray VGT’s use in the real world. The second reason was out of necessity due to the physical distance, i.e. 500 kilometers, between the research team’s location and the company. The information exchange took place more often at the start of the project, at least once each week, since the research team had deeper understanding of VGT than the testers, i.e. the research team could provide the testers with expert support. This support included information of how to improve the VGT test suite that was being constructed but also suggestions of how to document the test suite and solutions to specific, low-level, problems that the testers had run into. In cases where the research team did not already have a feasible solution to a problem, the research team instead aided in the information acquisition to help the testers develop a solution. Further into the project, the information exchange became less frequent with telephone or mail communication roughly twice each month. During these discrete instances, challenges, limitations and solutions were discussed as well as the progress of the VGT transition. In addition, cost and time metrics were collected from the testers. Hence, the role of the research team in this stage of the project was two-fold. First to provide support for the VGT transition project, and second to acquire empirical data regarding the VGT transition from the testers.

In the third stage of the study, which aimed to portray the project and its outcome, a second workshop was held on site at the company, during which two structured deep interviews were held with the driving testers, shown in the row named ‘Post-study’ in in Figure 1. Additionally, at this point of the project, an additional tester had joined the transition project who could provide a new perspective and further information about the transition and usage of VGT. The purpose of the interviews was to verify previously collected data, get a deeper understanding of the transition project as well as to collect further data on challenges, limitations and solutions that had been identified. Both of the interviews were recorded and processed into empirical data.
conducted using the same set of questions in order to raise the internal validity of the answers [17]. 71 questions were prepared for the interviews, 67 with the purpose of eliciting and validating previously collected information and 4 attitude questions aimed at capturing the testers views on VGT, post project. More specifically, the four questions were,

1) Does VGT work? Yes/No, why?
2) Is VGT an alternative or only a compliment to manual testing?
3) Which are the largest problems with VGT?
4) What must be changed in the VGT tool, Sikuli, to make it more applicable?

In all of the questions, VGT was in this case equalized with Sikuli [9] since it was the VGT tool that had been used during the project. After the interviews, the recordings were transcribed in order to make the information more accessible. In addition, the answers were analyzed and compared among the respondents, i.e. the driving testers, to ensure that there were no inconsistencies in the factual data. The analysis showed that the respondents had answered the majority of the questions the same, including all attitude questions, but that they had complimenting views on the attitude questions, i.e. what was the largest issue with working with VGT, etc.

IV. RESULTS AND ANALYSIS

The following section presents the results, and analysis of the results, divided according to the three stages of the VGT transition project, i.e. pre-transition (pre-study), during the transition (case-study) and post-transition (post-study) to VGT.

A. Pre-transition

The VGT transition at Saab was initiated out of necessity to shorten the time spent on manual testing. For each release, every six months to one year, the SUT went through extensive regression testing where a selected subset of the SUT’s test cases were manually performed. Each regression test session had a budget of four to six weeks of man-hours. The test cases were documented in 40 test suites, referred to as acceptance test descriptions (ATD). Each ATD consisted of a considerable set of use cases (UC), e.g. roughly 100, which each defined valid SUT input and the expected output. On a meta level these UCs were linked together into test chains that defined the test case scenarios, as exemplified in Figure 2. A test case was defined as a test path through a test chain that could be either linear, or contain branches, where a set of UCs, UC1 and UC2 (Top left of Figure 2), were first executed to set up the SUT in a specific state. The set up was then followed by the execution of one of a set of optional UCs, UC3A-C (Middle of Figure 2) to create a test path. Test paths could also have varying length, as exemplified in the figure where UC3A (Middle left in Figure 2) is followed by UC3AA (Bottom of Figure 2) while the other two branches (UC3B and UC3C) lack following UCs. Hence, each test chain could contained a set of branching test-paths, i.e. test cases, defined by either common or unique UCs. The modular architecture of the manual test cases provided a lot of flexibility but was also considered tedious since some test chains required a lot of setup while only performing a small/short test thereafter.

The manual test period, four to six weeks, for the SUT, was then followed by a factory acceptance test (FAT) with the customer, executed over an additional two to three weeks, to validate the system, i.e. six to ten weeks of testing in total. However, a FAT would only be initiated if the manual tests had been executed successfully. Hence, transitioning to VGT from manual testing would constitute a large gain for the company in terms of development time, cost and potentially in quality, since a larger subset of test cases from the ATDs could be executed faster and at higher frequency [1]. Raising test frequency was also important since manual testing was the only means of testing the system, i.e. no other tests existed for regression testing purposes such as automated unit tests, etc.

Three VGT tools were evaluated for the project, i.e. EggPlant, Squish and Sikuli, to find one suitable for the VGT transition. A brief overview of the results of the evaluation is given in Table I. The primary successfactors during the evaluation, which took six man-weeks, were tool cost and script language ease of use. Each tool was evaluated based on its static properties as well as through ad hoc scripting and automation of actual use cases from the ATDs. In addition, the evaluation took into consideration the results from the research teams previous work [1].

The result of the evaluation was that EggPlant was a mature and suitable tool but that it was very expensive and that the tool’s scripting language was a limitation, i.e. it had a high learning curve and did not suit the modular design of the tests that the testers were aiming for. Squish, used by other departments at Saab, was not suitable either since it performed GUI based testing through manipulation of execution threads in the application. However, the SUT was running roughly 40 threads at a time, spread over different system components, which limited Squish ability to interact with the SUT. Additionally, the tool was unable to identify objects placed on the map, due to its limited image recognition capabilities, which
was a key feature of the SUT that the VGT tool had to be able cope with in order to be applicable. Lastly, Sikuli was evaluated and found to be a feasible option, partly because the tool is open source, and thereby carries no up front cost, but mostly because of the tool’s scripting language which is based on Python. Python was considered valuable since it has a familiar syntax, i.e. common to most iterative and object-oriented programming languages, and because Python provides the capabilities of an object-oriented programming language. The main limitation with Sikuli, that was identified at this stage of the project, was that the tool did not have built in support for neither development or management of test suites. However, thanks to the power of the tool’s scripting language this was considered a minor obstacle since a custom solution could easily be developed by importing and extending existing testing and test suite libraries for Python. Another problem that was identified was that Sikuli did not have any built in virtual network connection (VNC) support, required to test the SUT’s distributed functionality. However, by pairing Sikuli with a third part VNC client-server application, this issue was also easily solved.

B. During transition

The VGT transition took place during roughly four calendar months, during which three representative ATDs were fully implemented into a VGT test suite. Representativeness was measured by the ATD’s complexity, where two of the chosen ATDs were considered more complex than the average 40 ATDs, whilst the third was equal in complexity to the remaining ATDs. The VGT test suite architecture, visualized in Figure 3, consisted of two main parts. First, a main script for each ATD that imported all the automated ATD test cases, i.e. the test chains built from use cases. The second part was the test cases themselves which were executed by the main script according to the numerous test paths in each test chain. This architecture was required since Sikuli does not, as mentioned, provide any support for either development or management of test suites. The VGT test suite was also developed using external libraries, ’lib’ in Figure 3. One of these libraries was a Python library for formatting and producing output. Output that could be viewed graphically through any web browser, i.e. the result of each test case was visualized as passed or failed in a table. Additionally, a Java library for taking screenshots was incorporated in the VGT test suite. The screenshots provided additional value to the result output by capturing the state of the system when a bug was identified, i.e. the faulty state of the GUI was captured for further analysis and for manual recreation of the bug. According to the testers, this functionality made it easier to explain, and present, the bugs they encountered to the developers, thereby quickening SUT maintenance time. In addition, all global variables used in the scripts were placed in its own library called ‘glob’, whilst the ‘cfg’ library included all external paths, i.e. paths to where to save log files, find the external libraries, etc.

After automation of the three ATDs, the testers compared the VGT test suites’ execution times against the manual execution times. The results showed an estimated speed up of a factor 16, from two working days (16 hours) to 1 hour for the two complex ATDs and from 1 day to 30 minutes for the third. Hence, the automation constituted a huge gain in test case execution time with no reported detrimental effects on bug finding ability, i.e. all bugs in the system that were identified using manual test practices could also be identified using the VGT test suite. In addition, due to the quicker execution speed, the automated ATDs could be run several times in sequence. The iterative test suite execution placed the SUT in states that the automated ATDs could be run several times in sequence. The iterative test suite execution placed the SUT in states that the automated ATDs could be run several times in sequence. The iterative test suite execution placed the SUT in states that the automated ATDs could be run several times in sequence. The iterative test suite execution placed the SUT in states that the automated ATDs could be run several times in sequence.

The screenshots provided additional value to the result output by capturing the state of the system when a bug was identified, i.e. the faulty state of the GUI was captured for further analysis and for manual recreation of the bug. According to the testers, this functionality made it easier to explain, and present, the faults they encountered to the developers, thereby quickening SUT maintenance time. In addition, all global variables used in the scripts were placed in its own library called ‘glob’, whilst the ‘cfg’ library included all external paths, i.e. paths to where to save log files, find the external libraries, etc.

After automation of the three ATDs, the testers compared the VGT test suites’ execution times against the manual execution times. The results showed an estimated speed up of a factor 16, from two working days (16 hours) to 1 hour for the two complex ATDs and from 1 day to 30 minutes for the third. Hence, the automation constituted a huge gain in test case execution time with no reported detrimental effects on bug finding ability, i.e. all bugs in the system that were identified using manual test practices could also be identified using the VGT test suite. In addition, due to the quicker execution speed, the automated ATDs could be run several times in sequence. The iterative test suite execution placed the SUT in states that the automated ATDs could be run several times in sequence.

The screenshots provided additional value to the result output by capturing the state of the system when a bug was identified, i.e. the faulty state of the GUI was captured for further analysis and for manual recreation of the bug. According to the testers, this functionality made it easier to explain, and present, the faults they encountered to the developers, thereby quickening SUT maintenance time. In addition, all global variables used in the scripts were placed in its own library called ‘glob’, whilst the ‘cfg’ library included all external paths, i.e. paths to where to save log files, find the external libraries, etc.

After automation of the three ATDs, the testers compared the VGT test suites’ execution times against the manual execution times. The results showed an estimated speed up of a factor 16, from two working days (16 hours) to 1 hour for the two complex ATDs and from 1 day to 30 minutes for the third. Hence, the automation constituted a huge gain in test case execution time with no reported detrimental effects on bug finding ability, i.e. all bugs in the system that were identified using manual test practices could also be identified using the VGT test suite. In addition, due to the quicker execution speed, the automated ATDs could be run several times in sequence. The iterative test suite execution placed the SUT in states that the automated ATDs could be run several times in sequence.

The screenshots provided additional value to the result output by capturing the state of the system when a bug was identified, i.e. the faulty state of the GUI was captured for further analysis and for manual recreation of the bug. According to the testers, this functionality made it easier to explain, and present, the faults they encountered to the developers, thereby quickening SUT maintenance time. In addition, all global variables used in the scripts were placed in its own library called ‘glob’, whilst the ‘cfg’ library included all external paths, i.e. paths to where to save log files, find the external libraries, etc.

After automation of the three ATDs, the testers compared the VGT test suites’ execution times against the manual execution times. The results showed an estimated speed up of a factor 16, from two working days (16 hours) to 1 hour for the two complex ATDs and from 1 day to 30 minutes for the third. Hence, the automation constituted a huge gain in test case execution time with no reported detrimental effects on bug finding ability, i.e. all bugs in the system that were identified using manual test practices could also be identified using the VGT test suite. In addition, due to the quicker execution speed, the automated ATDs could be run several times in sequence. The iterative test suite execution placed the SUT in states that the automated ATDs could be run several times in sequence.
for new input before the script continued its execution.

During development, attempts were made to integrate the VGT test suite into the SUT's build system, i.e. to allow completely automatic system regression testing after each new build. However, since the VGT test suite required manual setup and some configuration before execution, such a scheme was never implemented due to time constraints. Instead, the VGT test suite was run on an ad hoc basis, i.e. not periodically, but with much higher frequency than the previous manual testing. The higher frequency regression testing was reported as most beneficial for the development of the SUT since it provided the developers with quicker feedback.

1) VGT test suite maintenance for improvement: To ensure validity of the automated test scripts, they were developed as a 1-to-1 mapping of the manual test cases, i.e. the manual tests were used as a specification for the automated scripts. However, later during the project, the VGT test suite was subject to maintenance. The maintenance done to the test case scenarios included, but was not restricted to, modification of the order of script operations, in order to provide smoother and quicker test case execution, and further modularization to facilitate strategic reuse. Hence, breaking the 1-to-1 mapping in some of the test cases. However, the purpose of each automated test case, i.e. the functionality the test case aimed to verify in the SUT, was kept the same. Consequently, a conclusion can be drawn that strict automation, i.e. 1-to-1 mapping, of the manual test specification may not necessarily be the best automation approach. Rather, the specification should only be used to specify what to test in the SUT, not necessarily how. The reason is because with automatic testing you can, and often want, to improve the test execution speed as much as possible, which can be done by grouping certain actions together. In contrast, manual test scenarios need to be unambiguous and test actions defined logically to have high quality [18], which isn’t necessarily the fastest. Hence, the quality of a VGT script is greatly affected by how it is designed and implemented, i.e. narrowing the gap between testing and traditional software development.

The performed refactoring of the VGT test suite was required since this project was conducted under continuous time pressure, with project managers expecting quick results. This pressure resulted in, as presented by the testers, development of the first possible solution for certain problems which necessarily wasn’t always the best solution in terms of script quality, performance, reusability, etc. Additional refactoring was also required due to the testers inexperience of using Sikuli at the start of the project. Among the refactoring that was made, in order to improve maintenance of the scripts, all global variables were moved to a common namespace, i.e. ‘glob’, as shown in Figure 3. Hence, all variables were clustered in one library and then, together with the libraries, ‘lib’ and ‘cfg’, imported to all scripts that required them.

During the VGT test suite maintenance, the testers observed that it was easier to maintain the scripts that they had written last since they had a clearer memory of what the scripts did. Additionally, they reported that whilst maintenance of their own code was almost as quick as writing code from scratch, maintenance of scripts written by the other tester took considerably longer. One solution to mitigate these problems would have been a common coding standard of how to name variables, write loops and branches, etc. This problem, once again, illustrates how VGT, using Sikuli, in many respects has more in common with traditional software development than testing. However, as reported by the testers and in contrast to traditional development, the maintenance work was made easier by the scenario based structure of the scripts and the intuitiveness provided by inclusion of images in the scripts, a feature provided by Sikuli’s IDE. It was perceived by the testers that pure Python code would have been more difficult to maintain; the in-script images simplified understanding and remembrance.

2) VGT test suite maintenance required due to SUT change: Three calendar months into the VGT transition project a huge change was made to the SUT which included replacement of the map engine. Since the map engine was part of the core functionality of the SUT this change also affected the VGT test suite, i.e. causing 85-90 percent of the scripts to fail and thereby require some kind of maintenance, which included changing 5-30 percent of the images in every maintained script. The maintenance effort required to get the VGT test suite working completely again took roughly three man-weeks (240 man-hours) of work, which is to be related to the VGT test suite development time of three man-months (1032 man-hours). Hence, the estimated maintenance time of the entire VGT test suite, all 100 percent of the test scripts, would be 25.8 percent of the development time, i.e. 266 man-hours, which can be compared to the manual test budget of 480 man-hours per SUT development iteration. Note, the 4-6 week manual execution time, 120 hours, is with two testers. Consequently, the estimated development time of all 40 ATDs would be 13760 man-hours (7.6 man-years) and assuming all of the tests broke, the maintenance time would be 3550 man-hours, equal to roughly 21 man-months of continuous work or equivalent to the budge of 7 iterations of manual testing, i.e. roughly 3.5 years. However, the time required to execute all of the 40 ATDs manually is estimated to 2400 hours. Hence, assuming that none of the tests required maintenance and the complete VGT test suite (40 ATDs) was executed continuously, i.e. 24 hours a day, the ROI for the entire development would be positive after roughly 8 days (199 hours), i.e. after executing all the 40 automated ATD’s 6 times. Additionally, for the three ATDs that were automated in the project, a positive ROI would be reached after 13 executions, i.e. after 32.5 hours of continuous execution, which is less than the time of the manual ATD execution, i.e. 80 man-hours.

These numbers, summarized in Table II, do however not reflect the manual testing that is performed during the VGT test script development, required to validate test script conformance to the manual test specifications. Furthermore, the numbers do not take into account aspects such as the number of faults found during the test execution, i.e. quality gained from quicker feedback to the SUT developers and other benefits
The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability.

The single largest problem, as described by the developers, was however the failure rate of Sikuli’s image recognition algorithm, which was not improved by running the scripts from the command line. Estimates done by the testers indicate that the VGT test suite only had a success rate of 70 percent. This low success rate has been established by the testers to be due to the use of VNC. The VNC server-viewer application is prone to failure in certain instances, e.g. if the execution thread of a script is manually terminated, or if the tool is terminated with an unsaved script, etc. The solution to solve these problems has been to only use Sikuli’s IDE for script development and instead run the developed VGT test suite from the command line, which was found to greatly improve stability. 
TABLE III SUMMARY OF PROBLEMS AND SOLUTIONS IDENTIFIED DURING THE VGT TRANSITION PROJECT.

<table>
<thead>
<tr>
<th>Title</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNC</td>
<td>VNC has negative effects on the image recognitions ability to identify GUI graphics</td>
<td>Minimize use of VNC if possible, use high-quality VNC application, use EggPlant</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Understanding other developers scripts can be problematic even with the scenario based structure of the scripts</td>
<td>Enforce coding standards to raise understandability and readability of the scripts</td>
</tr>
<tr>
<td>1-to-1 mapping</td>
<td>1-to-1 mapping between manual and automated tests is not always possible or favorable</td>
<td>Modularization of test scripts can increase test execution speed and reusability. Hence, a 1-to-1 mapping should be strived for only if it does not have detrimental effects on test quality.</td>
</tr>
<tr>
<td>Sikuli IDE</td>
<td>Sikuli is not a finished product and therefore cause the Sikuli IDE to fail unexpectedly</td>
<td>Use IDE only for script development but execute scripts from command-line</td>
</tr>
<tr>
<td>Volatility</td>
<td>Sikuli’s API is poorly documented</td>
<td>Ensure internet connectivity to make it possible to look up solutions and other information online.</td>
</tr>
<tr>
<td>Lack of document</td>
<td>Many problems were identified with Sikuli’s image recognition, e.g. spontaneous inability to find images, click operations performed next to intended location, etc.</td>
<td>No one solution was identified, but potential solutions include fine-tuning the scripts, better selection of images, running scripts locally without VNC, etc.</td>
</tr>
</tbody>
</table>

because it is improbable that test scenarios can be devised that cover all states of a large systems, which is equally unlikely for manual scenario based test cases. Instead, manual exploratory testing should be used to uncover new faults.

For the third question, what is the biggest problem with VGT, one of the testers stated, “I don’t see any problems with it, but we need to get around the fact that it does not always work and that we always don’t know why.”, referring to Sikuli’s volatility. Another tester answered, “The image recognition comes with an inherent uncertainty”, i.e. fragility to unexpected SUT behavior, etc. However, the testers had a pragmatic approach to these issues and stated, “Sikuli is a program, it’s also a system and systems have faults”. Hence, they had accepted the tools limitations but also identified that most of these limitations could be mitigated through structured script development, redundancies in the scripts and other failure mitigation practices.

Finally, when asked what can be improved with the VGT tool, the testers answered that the reliability of the tool should be increased or at least a study should be conducted that can explain why the image recognition works in some cases, for some images, and not for others. Additionally, the tool documentation needs to be improved and since one of the largest issues during the VGT transition was found to be how the tool interacted with VNC, Sikuli should be fitted with VNC capability, similar to EggPlant. As stated by the developers, “EggPlant was much more stable with VNC. We have not managed to make Sikuli as stable.”.

Due to the success of the transition project, i.e. identification of previously unknown faults in the SUT and the perceived cost-effectiveness of the technique, the use of VGT has also been accepted by the customer as a compliment to the manual testing. Additionally, because of the success, the company plans to continue the automation of more ATDs and also develop a new VGT test suite to test all basic functionality of the SUT. This new VGT test suite will not be based on the manual ATDs but rather on domain knowledge about the intended low-level functionality of the SUT. The testers at Saab have also started looking at the possibility of creating an automatic thread-based exploratory test (TBET) based VGT application. TBET, a refinement of exploratory testing [19], is executed by following one or several execution threads, scenarios, through the SUT to find faults, and also their causes. However, no actual implementation had been conducted on such a solution at the time of the project.

Hence, it can be concluded that even though VGT has its limitations, challenges and problems, it is still a viable and applicable technique for industrial use when performed by practitioners. This conclusion is strengthened by the impact that the transition project has had within the Saab corporation where more Saab companies have started working with the technique. Even though, as reported by the testers, there are naysayers claiming that “Automation did not work 25 years ago and therefore it won’t work now.”. However, in this paper we have presented information that contradicts the naysayers claims, e.g. feasible development and maintenance costs, raised fault finding ability.

V. DISCUSSION

The data collected during the industrial case study shows that the transition to VGT was both successful and of benefit to Saab, benefits summarized in Table V. Firstly, the execution speed of the company’s previously manual tests was greatly improved that allows for greater test frequency and thereby faster feedback to the developers, i.e. from months to hours.
Past 3, previously Linear cost Benefit (versus man-
and replay (R&R). Firstly, results show that the maintenance 
were identified that are of benefit for future use of VGT and 
quickly, as shown in Table IV.
executions required to reach a positive ROI can be performed 
due to the execution speed of a VGT test suite, the number of 
after which the cost of executing the scripts is constant. Hence, 
only has an initial cost for developing the automated test suites 
increases linearly with each development iteration, but VGT 
identification due to raised test frequency. Manual testing cost 
from the normal manual testing, and the benefit of faster fault 
transition to ensure script validity, i.e. not taking time away 
that the manual tests are continuously performed during VGT 
transition. One benefit being the techniques, e.g. R&R, where entire test suites were rendered useless due to SUT change. Consequently, the black box nature of VGT, due to the image recognition, makes changes to the SUT maintainable. However, the collected data is not enough to draw a definitive conclusion that the maintenance costs of VGT scripts are feasible for industrial use; more research is needed on this in the future.

Secondly, Table V presents data regarding the stability of VGT when used together with a virtual network connection (VNC). VNC was used during the project because the SUT was distributed over several computers. However, this pairing was recognized as a large problem since it lowered the success rate of the automated test suite, when it should have succeeded, to roughly 70 percent, i.e. due to image recognition failures. The VNC problem was identified by running a subset of test scripts, which could be run locally, against the SUT that resulted in a success rate of 100 percent, even when the tests were rerun continuously for 24 hours. Hence, Sikuli’s image recognition was not the source of the problem, but rather it was the third party VNC application, mitigated by local VGT test script execution. However, since the system was distributed over several computers, i.e. nodes, this solution instead limited which test cases could be executed. Hence, this was not identified as a benefit but rather an improvement of how to use Sikuli to raise test suite stability. Consequently, either a better VNC application to be obtained or VNC should be integrated into Sikuli as already available in the VGT tool EggPlant, which was perceived by the testers be much more stable in this regard. However, EggPlant, as reported by the testers, had other limitations, e.g. a high cost and, what they considered, an unintuitive and more restricting scripting language. Consequently, existing VGT tools suffer from important, but different limitations, that makes it likely that manual test execution will still have to complement automated testing. However, the testers’ common view is that VGT both works and provides substantial value to the company, even given the tools’ limitations.

The testers also identified other less quantifiable benefits with VGT during the project. One benefit being the techniques flexibility and ability to work with any application regardless of implementation language or even platform, i.e. web, desktop, mobile, etc. This flexibility allows VGT to interact with the SUT whilst also interacting with SUT related simulators, written in other programming languages, or even the operating system if required. This is a specific benefit of VGT that might or might not be present with other similar testing tools, such as R&R or GUI testing techniques that are specific to the GUI library in use. In addition, the VGT tool that was used, i.e. Sikuli, uses Python as a scripting language that provides the user with all of the properties of a lightweight, object-oriented programming language. These properties presents new interesting opportunities for automated testing but also new problems. Since the scripts follow the rules of traditional

<table>
<thead>
<tr>
<th>Description</th>
<th>Past</th>
<th>Current</th>
<th>Benefit (versus manual testing) or improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATD execution time</td>
<td>1-2 days per ATD (60 man-weeks for all ATDs), manually</td>
<td>0.5-1 hour per auto. ATD (Estimated 33 hours for all automated ATDs)</td>
<td>Test execution 16 times faster, higher test frequency, quicker feedback to developers</td>
</tr>
<tr>
<td>Fault finding</td>
<td>-</td>
<td>3, previously unknown, faults found</td>
<td>VGT provides greater fault identification ability, higher system quality</td>
</tr>
<tr>
<td>Test ROI</td>
<td>Linear cost (Manually)</td>
<td>Constant cost after 1 iteration (Automatic)</td>
<td>Positive ROI after one iteration, feasible development cost</td>
</tr>
<tr>
<td>Script maintenance cost</td>
<td>Unfeasible in the worst case for previous GUI test techniques (record and replay) [8]</td>
<td>~25% of the development cost of the VGT test suite (Saab project, with Sikuli)</td>
<td>Maintenance cost perceived feasible</td>
</tr>
<tr>
<td>Sikuli executed over VNC</td>
<td>~70% success rate with VNC</td>
<td>100% success rate without VNC</td>
<td>Sikuli stable when executed locally</td>
</tr>
</tbody>
</table>

TABLE V
SUMMARY OF QUANTITATIVE BENEFITS AND IMPROVEMENTS IDENTIFIED DURING THE VGT TRANSITION PROJECT.
software development they are also subject to the same types of faults, i.e. if implemented incorrectly they can contain bugs. Consequently, an inherent risk with complex scripts is that they report type 2 errors, i.e. false negatives, due to the scripts themselves being faulty. Hence, the question becomes, how do you verify the tests? Verification of scenario based scripts that strictly follow a manual test description can perceivably be done through comparison with the outcome of the manual tests. However, for more advanced VGT-based test applications a more complex verification technique might be required, e.g. based on oracles or properties, or other state-of-art techniques, to ensure that all faults in the SUT are identified.

A. Threats to validity

The main threat to the validity of this study is that it only presents results from one VGT transition project at one company. Hence, the results may have low external validity for other companies and domains [17]. In addition, since no structured data collection process could be performed by the driving testers during the project, due to resource constraints on the company’s end, quantitative metrics were only sparsely collected. The risk that very little quantitative data would be available from the project was identified already before the case study started and originates in the fact that this project was performed in a real-world context with real-world time constraints. Consequently, the results presented in this work are primarily based on data collected through interviews and are therefore mostly qualitative in nature. Further work is therefore required in more companies to provide additional support regarding the real-world applicability of VGT. Another threat is that the driving testers at the company might have been biased, i.e. wanting the transition to be successful. However, based on their thorough descriptions of faults, limitations and problems, this threat is considered minor.

VI. CONCLUSION

In this paper we present an industrial case study from a successful visual GUI testing (VGT) transition project, performed by practitioners, at the company Saab AB, subdivision SDS. Additionally, problems, limitations and solutions that were identified during the project are presented. Furthermore, support is given that the maintenance costs of a VGT test suite, developed in Sikuli, are not excessive, i.e. in this project 25.8 percent of the VGT test suite development cost.

In previous work we have shown the industrial applicability of VGT, but in a smaller transitioning project driven by researchers with expert knowledge of the technique. The more extensive transitioning project presented in this paper was instead initiated from industry, and originated in the business need to shorten the execution time of manual regression testing. The main limitation of the VGT tool, Sikuli, used during the project, was its unpredictability, e.g. uncertain image recognition outcome and tool IDE instability, which was partly mitigated through local test suite execution via the command line. The benefits of VGT were reported to be the technique’s flexibility to work with any application, greatly improved test execution speed (16 times faster than manual tests) and ability to identify all faults found by the previous manual tests. Furthermore, the VGT test suite could identify previously unknown faults, due to increased test execution speed that allowed the tests to be run several times in sequence. Results also showed that the VGT transition cost, of three automated acceptance test descriptions (ATD), was feasible, but that VGT transition of all of the company’s 40 ATDs would take 7.5 man-years of work, i.e. prioritization of the ATD transition will be required. However, the practitioners perception was still that the developed VGT test suite was beneficial and will provide the company with positive return on investment for all future use. Hence, even though there were problems and limitations, the practitioners’ perceptions, and collected data, show that VGT is a beneficial and feasible technique for industrial system test automation.

REFERENCES