JAutomate: a Tool for System- and Acceptance-test Automation

Emil Alégroth  
Software Engineering and Technology  
Chalmers University  
Gothenburg, Sweden  
emil.alegroth@chalmers.se

Michel Nass  
Inceptive AB  
Gothenburg, Sweden  
michel.nass@inceptive.se

Helena H. Olsson  
Department of Computer Science  
Malmö University  
Malmö, Sweden  
helena.holmstrom.olsson@mah.se

Abstract—System- and acceptance-testing are primarily performed with manual practices in current software industry. However, these practices have several issues, e.g. they are tedious, error prone and time consuming with costs up towards 40 percent of the total development cost. Automated test techniques have been proposed as a solution to mitigate these issues, but they generally approach testing from a lower level of system abstraction, leaving a gap for a flexible, high system-level test automation technique/tool. In this paper we present JAutomate, a Visual GUI Testing (VGT) tool that fills this gap by combining image recognition with record and replay functionality for high system-level test automation performed through the system under test’s graphical user interface. We present the tool, its general properties, detailed properties of one of the tool’s image recognition algorithms, and a comparison of these properties to two other VGT tools based on information from previous research [7]. In addition, we perform an in depth multi-aspect analysis of the tools impact on a company’s business, architecture, process and organization (BAPO) [8]. The BAPO aspects were chosen to provide breadth and depth to the analysis and show how the introduction of the tool will not just affect the adopting company’s test process but also other aspects of the company. Furthermore, we discuss the potential benefits of using JAutomate compared to manual system- and acceptance-testing and present the results from a survey, performed with 52 industrial practitioners, that shows not only an industrial need for the tool, but also that there are test-related problems in industry that the practitioners perceive can be solved, or mitigated, with the tool.

The specific contributions of this paper are:

1) A presentation of the VGT tool JAutomate, its properties and benefits compared with other VGT tools and manual testing.

2) An analysis of JAutomate’s impact on the BAPO aspects of a company.

3) A presentation of current industrial test-related problems and discussion of how JAutomate, and VGT, can solve, or mitigate, these problems.

The continuation of this paper is structured as follows. In Section II, related work is presented, followed in Section III with an in depth presentation, comparison and analysis of JAutomate. This is followed by Section IV that presents the industrial need of JAutomate and VGT, followed by a discussion of the tool and the technique in Section V. Finally the paper is concluded in Section VI.
II. RELATED WORK

System- and acceptance-testing is generally performed as a manual practice in industry due to the complexity and high level of system abstraction of the tests. The tests aim to verify system conformity to the system requirements, and are performed regularly on the system under test (SUT), i.e., for regression testing [1]. Furthermore, the tests are defined in test specifications, which look different in different companies but with the same basic architecture based on scenarios defined by test steps, i.e., steps that define user input to the SUT for which there is some predefined output. However, this manual test practice is costly, time consuming, tedious and also error prone, requiring resources such as reference systems, testers, etc. Automated testing has been proposed as the solution to these problems, e.g., unit testing [3], [9], model based testing [10], etc.

Unit testing [3], [9], is a common test automation technique in industry which is performed on the SUT’s component level, i.e., to test low-level functionality of the SUT. This technique is therefore limited in terms of usability, due to complexity and maintenance costs, for higher-level tests, i.e., system- and acceptance-tests. These limitations can be mitigated using another commonly used technique in industry, i.e., record and replay (R&R) [4]–[6]. R&R is a tool-driven, two step, technique where user interaction with the SUT is first recorded in a script that can then be replayed automatically to perform regression testing. The recording can be done in several ways, e.g., using references to the SUT’s backend or by using exact coordinates on the SUT’s GUI. However, both of these methods suffer from limitations that, once again, require high maintenance, e.g., reference based R&R is fragile to API or even code change [11], whilst coordinate based R&R is fragile to GUI layout change. Hence, R&R does not fulfill all of industry’s needs for a robust, flexible, high system-level, automated test technique.

In the early 90s, Potter presented his tool Triggers [12], for system automation using image recognition. Other early work on automation using image recognition was performed by Zettlemoyer and Amant [13]. However, in recent years, the use of image recognition has also been transferred to testing, in tools such as JAutomate, Sikuli [14], etc. These tools use image recognition and scenario based scripts to perform tests through the SUT’s GUI, a technique that Börjesson and Feldt refer to as Visual GUI Testing (VGT) [7]. In their work, they provide support for VGT’s industrial applicability by comparing two VGT tools based on their static properties and the tools’ ability to automate manual, industrial grade, test cases for a safety-critical air traffic management system. VGT is perceived to resolve many of the limitations of previous techniques, i.e., R&R, because the technique is black-box [15], i.e., does not require any knowledge about the SUT. Additionally, it is robust to GUI layout change due to the image recognition. However, most VGT tools require the test scripts to be written manually, which is associated with an up-front investment for the VGT transition, i.e., the test case automation. Consequently, VGT is perceived to be more flexible and robust than previous automation techniques, but the script development requires an upfront cost.

In this paper, we present JAutomate, a VGT tool that combines image recognition with record and replay capabilities. These capabilities perceivably lower the automation costs, since scripts can be recorded during regular manual test case execution, but retain all the benefits of the image recognition based playback, e.g., imperviousness to GUI layout change. Changes to the bitmap graphics of the GUI are instead what imposes the most amount of script maintenance. However, this maintenance can be done at low cost in JAutomate which supports simple swapping of images within the scripts. Thus, JAutomate perceivably fills the gap in industry for a high-level, cost-effective, flexible and robust test automation tool.

However, the introduction of JAutomate, like any new technique, method or tool, in a company, will affect several aspects of said company. There are several frameworks that capture these aspects, e.g., BAPO [8] and PESTEL [16]. BAPO, which stands for business, architecture, process and organization, was chosen as the framework for the analysis in this paper because it provides a comprehensive high-level view of what will be affected during software development change in a company. Hence, the introduction of JAutomate in a company will potentially not just affect the testing process, but also the company’s business, e.g., raised quality can be used as a business advantage, architecture, e.g., architectural changes may be required to apply the tool, and organization, e.g., new roles may be required. Thus, BAPO was used to give both a broader, and deeper, analysis of the impact JAutomate can/will have at a company that chooses to adopt it for VGT.

III. JAUTOMATE

JAutomate is a commercial Visual GUI Testing (VGT) tool developed by Innovative Tool Solutions in collaboration with the Swedish test consultant company Inceptive AB. The tool was innovated by Michel Nass, co-author of this paper, in 2006, after recognizing the potential of the VGT technique and that there were no tools adopting this technique available on the market. JAutomate was in 2006 presented as a concept to the leading test tool vendor at that time, Mercury (Today Hewlett Packard (HP), but no interest was shown by the company. In 2011 the first version of JAutomate was released to the market and has since then been utilized in several industrial projects, e.g., at Volvo, Siemens, CompuGroup Medical. Figure 1 shows a screenshot of JAutomate’s Integrated Development Environment (IDE). The IDE is used to develop, i.e., record, execute, i.e., replay, and maintain test scripts.

JAutomate, as mentioned, is however not the only VGT tool available to the market, both open source and commercial alternatives exist. So what makes JAutomate preferable over the other available tools? In the following sections we aim to answer this question by first presenting a comparison of the static properties of JAutomate and two other VGT tools, i.e., Sikuli and CommercialTool (that for legal purposes will be kept anonymous in this report), results shown in Table
I. The information for this comparison was acquired from our previous work [7] and complimented with information provided by JAutomate’s developer, Michel Nass. Secondly, we present a description of JAutomate’s perceived impact on the BAPO aspects of a company [8], based on previous academic empirical work with VGT, and Michel Nass’s expert experiences with JAutomate in industry.

A. Tool comparison

The following section presents a comparison between JAutomate, CommercialTool and Sikuli based on the tools’ static properties, results summarized in Table I.

Developed in. JAutomate, similar to Sikuli is developed in Java. However, in comparison to Sikuli, JAutomate does not include any native methods which makes it platform independent, i.e. it can be executed on any operating system that supports Java. In contrast, CommercialTool is developed in C# and is therefore only, by the tool’s vendor, supplied for Windows and MacOs.

Image recognition algorithm. This property was not evaluated in detail during our previous work and is therefore unknown on a detailed level for CommercialTool and Sikuli. However, JAutomate uses two algorithms, one based on color and the other on contrast, combined into the so called Vizion Engine. The benefit of having several algorithms, also supported in CommercialTool, is that it adds script robustness. Hence, if one algorithm fails, another can be used instead.

Script language syntax. The scripting language in CommercialTool is custom, based on natural language to make it intuitive for novice users. In Sikuli, the scripting language is based on Python, allowing the user to make use of all the aspects of Python, including iterative statements, conditional branching, etc. JAutomate, in turn, provides the user with a multi-level scripting interface which on the top layer is designed for novice users without programming experience, whilst on the lower levels, it allows the user to change all aspects of the script through Java code.

Image representation in tool IDE. Sikuli and JAutomate both visualize the sought images in the scripts, whilst CommercialTool represents the images as text, i.e. string variables. The benefit of using the actual images is that it makes the scripts more intuitive, but with the drawback, in Sikuli, that the same image might be used in several places and therefore has to be changed in several places during maintenance. However, in JAutomate, this problem has been solved by the use of reusable images, a feature that allows the user to update all images, in all impacted scripts, in a test suite by simply replacing the reusable image (Supported in JAutomate version 11.1 and forward).

Image recognition sweeps per second. This metric is dependent on the computer the tool is executed on, but as reported by the official documentation of Sikuli and CommercialTool the tools can make approximately 5 and 7 sweeps per second

<table>
<thead>
<tr>
<th>Property</th>
<th>JAutomate</th>
<th>CommercialTool</th>
<th>Sikuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developed in</td>
<td>Java</td>
<td>C#</td>
<td>Python</td>
</tr>
<tr>
<td>Image recognition algorithm</td>
<td>Vizion Engine</td>
<td>Several algorithms</td>
<td>-</td>
</tr>
<tr>
<td>Script language syntax</td>
<td>Custom</td>
<td>Custom</td>
<td>Python</td>
</tr>
<tr>
<td>Image representation in tool IDE</td>
<td>Text strings / images</td>
<td>Text-Strings</td>
<td>Images</td>
</tr>
<tr>
<td>Image recognition sweeps per second</td>
<td>Depends on CPU(s) and image size</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Image recognition failure mitigation</td>
<td>Can automatically select from two different algorithms (color and contrast) or perform semi-automated test steps</td>
<td>Multiple algorithms to choose from</td>
<td>Image similarity configuration</td>
</tr>
<tr>
<td>Test suite support</td>
<td>Yes</td>
<td>Yes</td>
<td>Unit tests only</td>
</tr>
<tr>
<td>Remote SUT connection support</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Remote SUT connection requirement</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cost</td>
<td>$849 per user per year</td>
<td>10,000 Euros per license per computer</td>
<td>Free</td>
</tr>
<tr>
<td>Record and Replay functionality</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Manual test step redundancy</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Semi-automated test steps</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Backwards compatibility</td>
<td>Yes (based on standard text files and png images only)</td>
<td>Guaranteed</td>
<td>Uncertain</td>
</tr>
</tbody>
</table>

TABLE I
Summary of the comparison of JAutomate, CommercialTool and Sikuli’s static properties.
respectively. No such metric is available for JAutomate since the image recognition algorithms are not only dependent on the performance, i.e. clock rate, of the CPU, but also the number of cores in the CPU, i.e. the algorithms’ performance scales linearly with the number of cores.

**Image recognition failure mitigation.** The image recognition algorithms of all three tools are quite robust, but in some instances they have been known to fail. To mitigate such failure, both JAutomate and CommercialTool deploy a solution with several redundant image recognition algorithms. Hence, if one algorithm fails, the other(s) are used instead. In Sikuli all redundancy has to be built into the scripts manually, e.g. by using exception handling.

**Test suite support.** Both JAutomate and CommercialTool have built in support to build test suites, linking scripts together into more advanced test structures, etc. However, in Sikuli no test suite support exists, only support for individual unit tests. Hence, in order to develop a test suite of Sikuli scripts a custom test suite solution has to be developed, e.g. using Python’s ability to import scripts into other scripts.

**Remote connection support and requirement.** Neither JAutomate or Sikuli has support to control remote computers, e.g. over VNC or remote desktop. CommercialTool in contrast does not only support such a feature, but requires it. Hence, in order to execute scripts locally, with CommercialTool, a virtual machine has to be set up. The benefit of remote test script execution is that it separates the performance intensive execution of the image recognition algorithm from the SUT execution, i.e. mitigating the risk of incorrect SUT behavior due to lack of performance resources.

**Cost.** Sikuli is an open source product, meaning that it is free of charge. In contrast, CommercialTool and JAutomate Studio, i.e. JAutomate’s integrated development environment, are both commercial products that cost 10,000 euros per license per year and $849 per computer per year respectively. JAutomate’s runtime environment is however free of charge and can be used with a regular text editor together with a screen capture tool such as Snagit, etc.

**Record and Replay functionality.** Whilst both CommercialTool and Sikuli require the user to manually input the test scripts, JAutomate supports automatic recording of the test scripts. This functionality lowers the implementation time of the scripts, especially for scripts that are developed as 1-to-1 mappings of manual test cases, since these scripts can be recorded during scheduled execution of the manual test cases.

**Manual test step redundancy and semi-automated test steps.** Script failure mitigation due to an identified bug, faults in the script, etc., have already been discussed. However, JAutomate also has built in support during failure to, as a last resort, ask the user to perform a test step manually. Hence, if the image recognition cannot resolve a failure during execution it prompts the user, through a pop-up, to manually resolve the failure. In addition, JAutomate includes the possibility of adding manual test steps in the automated execution, i.e. making it possible to include test steps that require physical interaction with hardware, e.g. turning on a printer.

**Backwards compatibility.** Both JAutomate and CommercialTool are guaranteed to be backwards compatible. However, Sikuli, being open-source, may change its instruction set. In previous releases of Sikuli, i.e. version rc2 to rc3, several methods were changed and new ones introduced. The old methods still worked, but the recommendation from the developers was to migrate to the new methods.

Consequently, there are clear differences between the three tools that also affect their suitability and impact in different contexts. The context impact of JAutomate on the BAPO aspects is discussed further in the following sections.

**B. Business**

From the business aspect of a company, JAutomate, and VGT, will have several potential effects, even though they could be considered secondary effects. The purpose of JAutomate is to provide the company with automated system- and acceptance-regression tests, which allows the company to raise regression test frequency. Hence, provide the developers with quicker feedback, which will lower lead-times and thereby potentially lower development time and raise quality. Lower development time means lower cost for the customer, and raised quality that the customer receives a better product, which are both potential market advantages.

Furthermore, due to the record functionality of JAutomate, the tool could be used during manual acceptance testing with the customer to capture how the end-user will actually work with the system. Hence, capturing how the system will be used in the real-world and thereby improving the tests and raise the quality of the system.

Consequently, JAutomate has properties that affects a company’s business through lowering development cost and raising quality, which will provide the customer with a cheaper, yet higher quality, end product, i.e. qualities that can give the development company a market edge.

**C. Architecture**

The architectural aspect can be viewed from two perspectives. First, the perspective of JAutomate, i.e. the tool itself, and second, from the perspective of the system under test (SUT). In the following sections, these two perspectives are discussed together with the benefits and drawbacks of different architectures.

1. **The tool:** The core of JAutomate, like most VGT tools, is its image recognition capabilities. In order to make the tool as fast and fault tolerant as possible, JAutomate has two image recognition algorithms that are combined into an artifact called the Vizion Engine. The first algorithm identifies images on the screen by comparing pixel color, i.e. comparison between the colors of the image in the script and the image shown on the SUT’s GUI. This approach makes the algorithm fast but less reliable since it makes the algorithm sensitive to the actual color of a widget, which might change dependent on what state the widget is in. For example, the algorithm can not detect an icon, on the Windows operating system (OS), that was unselected during recording if it is selected during playback.
which consists of Java code. Additionally, for advanced users, from the top custom script language down to a core level to manipulate the scripts on several levels of abstraction, advanced users, the architecture of the tool allows the user to users without programming experience. However, for more scripting language which has been designed to be intuitive. Furthermore, scripts in JAutomate Studio consist of a custom on new information acquired during execution of the script. JAutomate contains automated script adjustment support, i.e. script with the SUT, etc. In addition, to improve script quality, perfect after the initial recording and it is therefore possible to manually adjust and modify the scripts post-recording to improve captured images, add delays to better synchronize the script with the SUT, etc. In addition, to improve script quality, JAutomate contains automated script adjustment support, i.e. the tool can automatically improve captured images based on new information acquired during execution of the script. Furthermore, scripts in JAutomate Studio consist of a custom scripting language which has been designed to be intuitive to users without programming experience. However, for more advanced users, the architecture of the tool allows the user to manipulate the scripts on several levels of abstraction, from the top custom script language down to a core level which consists of Java code. Additionally, for advanced users, JAutomate provides features to tailor the high level scripts, e.g. manipulation of test script parameters, test suite support, conditional test execution and iterations. For instance, JAutomate supports data driven testing where the input parameters are provided by a comma separated text file. This feature lowers development cost since it allows the tool to perform the same test scenario over and over, but with different input- and output-parameters, instead of requiring one script per input and output. In addition, the tool supports creation of suites of tests that can either be executed in sequence or build on one another to form more advanced tree structured test hierarchies, which promotes modularization and reusability. Conditional test case execution is also supported, which is useful when there are dependencies between test scripts, i.e. the outcome of test case X prohibits the execution of test case Y, requiring test case Z to be executed instead. This feature mitigates the risk of test suite failure due to identification of a bug, image recognition failure or other failure caused by JAutomate or the SUT itself. Consequently, JAutomate is a multi-layered tool that is perceivably appealing to both novice and expert users by having high learnability for simple functionality, but with high degrees of flexibility due to the tool’s advanced functionality.

Additionally, the architecture of the Vizion Engine is designed to scale based on the number cores in the central processing unit (CPU) of the computer executing the tool. Thus, the speed of the algorithms increase linearly with the number of cores in the CPU, which can be observed both during playback and recording of the scripts. Additionally, the tool supports several connected computer monitors, i.e. desktops with higher resolution. However, higher resolution has detrimental effects on the speed of the image recognition algorithm since more area has to be sought through to find a matching image. However, the minimum requirements to execute JAutomate are still quite modest, despite the performance intensive image recognition algorithm, requiring only 128 MB of free Random Access Memory (RAM). For longer scripts 256 MB of RAM is however recommended. JAutomate is developed in pure Java code, i.e. the tool does not contain any native code, third party modules or libraries, which makes it platform independent. Hence, the tool can run on any computer as long as a Java runtime environment, Java 6 or a later version, supplied by Oracle, is installed. In addition, as reported by the tool’s developer, the use of pure Java code makes the tool easy to maintain and extend with new functionality. All maintenance of the tool is performed by Innovative Tool Solutions in collaboration with Inceptive which is the tool provider and also the provider of a tool implementation service. Tool support is provided through e-
mail by Innovative Tool Solutions or by Inceptive through telephone or personal support (Restricted to customers of JAutomate Studio).

2) The system under test (SUT): Similar to all VGT tools, JAutomate considers the system under test (SUT) as a black-box [15], meaning that the tool does not require any knowledge about the SUT’s internal architecture, components, development language, etc. The black-box approach is achieved through the high-level interaction of the tool, i.e., through the image recognition that allows the tool to provide input and observe output through the SUT’s GUI. Consequently, all automated tests developed using JAutomate are executed in the same way as a human user would perform them, i.e., through the operating system’s internal methods which makes the tool’s interactions indistinguishable from a human user from the SUT’s perspective. In contrast, previous GUI-based test techniques, e.g., widget based record and replay, interact with the SUT by calling operations/methods within the SUT, i.e., white-box testing [15]. Hence, these interactions are performed below the GUI layer and thereby differentiate from end user interactions. In addition, some white-box tools require hooks, or interfaces, to be added in the SUT, thereby changing the internal architecture of the SUT. Hence, JAutomate, in contrast to white-box tools, is non-intrusive and does not require any changes to the SUT in order to be applicable. Furthermore, due to the GUI-based interaction, the tool can interact with any type of system that can be accessed manually from a computer client, i.e., it can test desktop, web and even mobile applications.

Because of these properties, JAutomate can be used to convert manual test cases into equivalent automated tests, i.e., produce 1-to-1 mapped automated tests from the manual tests. Automated 1-to-1 mapped tests require less resources than their manual equivalents and allow a company to perform system regression tests with higher frequency, providing developers with quicker and more comprehensive feedback. However, depending on the SUT architecture, which can contain hardware components, e.g., devices such as printers, there are manual test steps that are impossible to automate using software test tools, e.g., check if a printer is turned on or physically press an emergency stop button. Therefore, JAutomate supports the insertion of manual test steps in the automated scripts. When a manual test step is triggered, the automated test execution is paused and a pop-up dialog is displayed on the screen with the manual test step instructions. Once the manual test step has been performed by the user, the user can tell the script to continue its execution. In addition, JAutomate supports semi-automated test steps. Hence, if the automatic execution fails, e.g., due to image recognition failure, the user is prompted with a manual instruction to resolve the problem before the automated test execution can be continued. Semi-automated test steps are useful when testing volatile, or concurrent, systems where the SUT’s behavior is difficult to predict. This feature also helps mitigate failure due to unexpected system behavior such as pop-up windows prompted by the operating system, other applications, etc.

D. Process

Similar to the introduction of any tool, practice, method, etc., JAutomate will impact the process of the target company, i.e., change how development and testing is performed. However, no empirical study has been conducted with the purpose of identifying what these changes might be or what impact they will have. The design of JAutomate does however allow it to run manual tests, as explained in Section III-C2, similar to the manual test runner in HP Quality Center. Thus, the tool can be used for creating test suites that are a mix of automated, manual and even semi-automated tests. Additionally, this functionality allows a test team to use JAutomate to define all their system- and acceptance-tests, i.e., not just automated tests. However, we do not propose that JAutomate should replace a company’s other testing tools, methods or practices, rather, we propose that a company should add the tool to their existing toolbox. Hence, JAutomate is not a replacement, but rather a compliment, to other tools and/or manual practices, which provides support for automated system- and acceptance-tests.

In order for JAutomate, and VGT as a technique, to be effective, the tool needs to be incorporated into the company’s testing process and be used in a continuous integration fashion, i.e., the VGT scripts should be executed every time the system is rebuilt. Additionally, a maintenance process needs to be put in place in order to ensure that the automated test suite is continuously up to date with the SUT, similar to the practice of maintaining manual test suites. Consequently, JAutomate will introduce overhead in terms of additional maintenance since both the scripts and the manual tests need to follow the system specification, especially for 1-to-1 mapped tests. However, due to JAutomate’s ability to define manual test steps within the scripts, the manual tests can be migrated into JAutomate test cases, i.e., have the manual test specification defined in JAutomate scripts. A practice that, perceivably, would mitigate the need for maintenance of several equivalent test artifacts.

E. Organisation

JAutomate is not considered to have a large impact on the organization, e.g., in terms of requiring new roles or redistribution of resources. The reason is because unlike other test techniques for complex testing, that sometimes require test automation specialists or experts to develop tests of high quality, e.g., in machine learning [17], JAutomate has been designed to be simple to use for any user. Thus, JAutomate has been developed to have high usability and learnability to allow both developers and testers with, or without, previous SUT or programming experience to use the tool. Hence, JAutomate does not require any change of a company’s current roles, introduction of new roles or infer replacement of human testers in their current roles. Instead, JAutomate is primarily a compliment to previous practices to make the testing more efficient in terms of cost and quality. In addition, it is perceived that VGT tools, e.g., JAutomate, help alleviate the tediousness of performing manual tests over and over. Tediumness that lead to developers and testers taking shortcuts, making mistakes, etc., which lowers the quality of the testing, and in extension,
software quality. Consequently, the impact on a company’s organization is low but high on the company as a whole.

IV. THE INDUSTRIAL NEED

In order to investigate the industrial need for JAutomate, and VGT in general, a survey was performed during a seminar about the technique, attended by approximately 100 industrial practitioners. The survey had two purposes. First to identify the test related problems that Swedish software development companies are currently facing. Second to evaluate the industrial practitioners’ knowledge about VGT, and JAutomate, and if VGT perceivably could solve, or mitigate, some of the test-related problems experienced by the practitioners. The first purpose was investigated through a question where the industrial practitioners got to distribute 100 points to rank 17 predefined test-related problems. 52 questionnaires were collected, hence a response rate of roughly 50 percent. Figure 4 visualizes the results from the first question, shown as the percentage of distributed points over the test-related problems.

Both the highest ranked and the third highest ranked problems, i.e. test-related cost and testing to late, can perceivably be mitigated by using VGT since VGT scripts can be executed without additional cost which makes it possible to raise the test frequency. Additionally, the problem with insufficient customer feedback can be mitigated by recording customer acceptance tests, mitigating the need for having customers on site for acceptance regression testing. Problems regarding test coverage can perceivably also be mitigated since VGT releases resources required for manual testing, i.e. developers and testers. These resources can instead be used to perform exploratory testing of the system to uncover previously unknown faults. Hence, VGT, performed with JAutomate, has the potential to solve or mitigate several of the largest test related problems encountered in industry.

V. DISCUSSION

Visual GUI Testing (VGT), with tools such as JAutomate, have been designed for development of high system-level tests for automated regression testing. Automated regression testing is generally proposed to be a good practice and has been incorporated in agile development processes such as eXtreme Programming to facilitate continuous integration (CI) [18]. However, CI testing is mostly associated with XUnit testing [3], but, as proposed by Fowler [18], other tools for end-to-end testing should be incorporated into the test process as well, e.g. FitNesse [19]. JAutomate has been designed with CI in mind and the test scripts can even be exported and run in FitNesse, or stand alone as a compliment to other automated testing. Hence, JAutomate is, and was designed, to be a compliment to other automated testing in a CI context, i.e. providing support for automated system- and acceptance-testing that previously had to be performed manually.

Furthermore, this report has presented JAutomate as a testing tool to perform VGT. However, given the capabilities of the tool, e.g. its ability to interact with any GUI based application, the tool can also be used for automation, e.g. to automate the build process performed during CI to minimize the risk of erroneous builds due to complexity [18]. In addition, JAutomate opens up new possibilities for monitoring of systems where it is unfeasible to use humans, e.g. to monitor memory usage during long-time tests or tests where input is given continuously to the SUT for longer periods of time, e.g. for 24 hours straight. Furthermore, JAutomate, and VGT, is perceived to be able to test non-functional properties of a SUT, e.g. usability, performance, etc, but no study has been performed to validate this claim, which is a subject of future research.

This report also presents a comparison between JAutomate and two other VGT tools based on their static properties,
results shown in Table I. An analysis of these results show that JAutomate has several benefits compared with the other tools, such as being platform independent, has record functionality for fast script development as well as manual and semi-automated test step execution. The tool also has beneficial features in common with one or both of the other compared tools, such as multiple image recognition algorithms, images within the scripts, comprehensive failure mitigation, test suite support and backwards compatibility. However, the tool lacks built in support for remote SUT connection, supported by CommercialTool. Previous research with Sikuli, corroborated with information collected by Michel Nass at CompuGroup Medical, does however show that VGT tools, including JAutomate, can be executed on top of third party remote SUT connection applications, e.g. virtual network connection (VNC) applications. Remote SUT connection allows VGT tools to test SUTs that are distributed over several computers and removes the performance intensive image recognition execution from the SUT. Thus, mitigating the risk of faulty SUT behavior due to lack of performance resources.

However, the maintenance costs of VGT scripts, e.g. with JAutomate, are still unknown and is therefore the main focus of future research. Empirical support that validates the feasibility of these costs is essential for the long-term applicability of the technique, especially since maintenance costs have been identified as one of the main problems with previous, similar techniques such as widget based record and replay.

VI. CONCLUSION

In this paper we have presented JAutomate, a tool, for Visual GUI Testing (VGT) with record and replay support for cost effective script development, for graphical user interface (GUI) based, automated, system- and acceptance-testing. The tool has several benefits compared to manual regression testing, and other VGT tools, which will impact a company in several aspects. Furthermore, a survey with industrial practitioners, showed that there is both industrial need and interest for the tool and VGT as a technique.

The industrial needs to lower lead times and raise software quality are ever growing, paving the way for new research into automated techniques and tools for all aspects of software engineering, from requirements engineering to testing. Test related automation techniques approach testing on different levels of system abstraction, from system- to code-level. However, due to different limitations of these techniques and tools there is currently a gap for a high-level, cost-effective, flexible and robust tool for system- and acceptance-test automation.

JAutomate fills this gap, with simple record and replay functionality, combined with image recognition, which allows the user to automate all types of user interaction performed through the system under test’s (SUT) GUI. A comparison between JAutomate and two other VGT tools shows that JAutomate has several benefits over the other tools, but also that the tools have different properties that make them suitable in different contexts. In addition, we have presented how JAutomate will have beneficial impact on other aspects than just the test process. For instance, from a business aspect, JAutomate will help raise software quality, which can be a business advantage. Additionally, since the tool is black-box it does not require any knowledge or modification to the SUT’s architecture in order to be applicable. Furthermore, the tool is designed with both novice and advanced users in mind and does therefore not affect the company’s organization, e.g. by requiring new, or changed, roles.

In addition, a survey performed with industrial practitioners showed that JAutomate, and VGT as a technique, can perceptively help solve many of the software market’s current test-related problems and that there is an industrial interest for the tool and the technique. The tool also opens up new possibilities for testing and automation, e.g. emulation of user monitoring during long-time tests.

Consequently, JAutomate is a promising tool that fills a current need in industry for a flexible, robust, easy, cost-effective, automation tool for GUI-based system- and acceptance-testing, to compliment a company’s toolbox of test techniques.

REFERENCES