Information and Software Technology 158 (2023) 107172

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Scripted and scriptless GUI testing for web applications: An industrial case

Axel Bons [a], Beatriz Marín [b], Pekka Aho [a], Tanja E.J. Vos [a,b,*]

[a] *Open Universiteit, The Netherlands*
[b] *Universitat Politècnica de València, Spain*

ARTICLE INFO

Keywords:
Case study
Complementarity
Scriptless testing
Scripted testing

ABSTRACT

Context: Automation is required in the software development to reduce the high costs of producing software and to address the short release cycles of modern development processes. Lot of effort has been performed to automate testing, which is one of the most resource-consuming development phases. Automation of testing through the Graphical User Interface (GUI) has been researched to improve the system testing.
Objective: We aim to evaluate the complementarity of automated GUI testing tools in a real industrial context, which refers to the capability of the tools to work usefully together.
Methods: To address the objective, we conduct an exploratory case study in an IT development company from The Netherlands. We select two representative tools for automated GUI testing, one for scripted and another for scriptless testing. We measure the complementarity by measuring the effectiveness, the efficiency, and subjective satisfaction of the tools.
Results: It can be observed that the scripted tool performs better in detecting process failures, and the scriptless tool performs better in detecting visible failures and also reaching higher coverage. Both tools perform in a similar way in terms of efficiency. Additionally, both tools were perceived to be useful in the survey performed for the subjective satisfaction.
Conclusion: We conclude that scriptless and scripted testing approaches are complementary, and they can improve the effectiveness compared to manual testing processes performed in an industrial context by detecting different failures and reducing the effort and time to find these failures and to reproduce them.

1. Introduction

The use of software systems is continuously growing due to the benefits that they bring both in industrial sectors as well as in daily routine tasks for end users. As a consequence, more software is being produced and slowly almost every company is turning into a software company [1]. However, due to the global competition and short time to market, software companies are struggling. They need to continuously improve their software development processes in order to reduce the time and cost of the development, and at the same time, increase the quality of their software products. More flexible methodologies to produce software and more intelligent automation of the development processes are needed.

Testing is the most commonly used technique for quality assurance in industry [2], but also one of the most resource-consuming phases in the software development process [3]. In an attempt to reduce resources, lots of research has been done to improve test automation. Most of these studies have been mapped and categorized in various literature reviews of testing tools [4–10]. In recent years, an increasing amount of research has focused on the automation of system testing through the Graphical User Interface (GUI), as can be observed in [11–15]. Test case generation through the GUI can be performed by using two different approaches (1) scripted testing, and (2) scriptless testing [16].

For companies with interest in improving their testing processes by incorporating automated GUI testing, it is important to know the benefits and drawbacks of various automated testing approaches. There are many informal sources (like blogs and other grey literature) that discuss this topic by comparing scriptless and scripted testing. But most of them have commercial interests. There are a few empirical studies evaluating only scriptless [16–18] – and separately – scripted testing [19–21] in isolation without comparing them with each other. A recent study compares the effectiveness of a scripted tool as well as a scriptless tool [22], showing that both approaches perform similarly with respect to code coverage. However, the study has been done only with student subjects in a controlled experimental setting; and it does not provide any information about the failures found by these approaches. More research is needed that compares the two approaches in an industrial setting. This paper fills this gap by presenting a case study that aims to explore the benefits of using scriptless testing as

* Corresponding author at: Universitat Politècnica de València, Spain.

https://doi.org/10.1016/j.infsof.2023.107172
Received 16 August 2022; Received in revised form 4 February 2023; Accepted 8 February 2023
Available online 13 February 2023

well as scripted testing in the IT company E-Dynamics that – before this study – used only manual testing. We use Testar () as the scriptless testing tool and Selenium () as the scripted testing tool. We select these tools because they are open source and widely known tools for automated GUI testing. We evaluate the complementarity of both approaches measuring the effectiveness, the efficiency, and the subjective satisfaction on a real industrial software.

The main contribution of this paper is the empirical evidence obtained by means of an industrial case study, showing that scriptless and scripted techniques are complementary: both improve the effectiveness of the testing process in different ways. Scripted testing performs better detecting process failures, and scriptless testing performs better detecting visible failures. Both approaches found different failures than manual testing. Regarding the efficiency, the time needed to apply scripted or scriptless approaches are similar to manual testing. We advocate that this time should decrease in longer periods of observation, but more studies are needed to draw strong conclusions about efficiency. Regarding subjective satisfaction, both approaches are perceived useful but neither of the approaches is perceived as a complete replacement to manual testing.

This contribution is useful for practitioners, as they could take more informed decisions using clear measures when searching for approaches to incorporate in their testing processes. This contribution is also useful for researchers, as they can extend this study with other tools, and, by following the methodological evaluation framework, gather more evidence for complementarity. We also advocate that this contribution can be useful in educational environments, since teachers can provide this result as a knowledge base to students, and they can select a tool to improve the system testing of their projects and learn how to improve system testing by comparing the results of the selected tool with the results obtained manually.

The rest of the paper is structured as follows: Section 2 presents the background, including a brief description of the tools selected for scripted and scriptless testing, and some related work. Section 3 presents the design for our case study. Section 4 presents the results of the study, and the interpretation of findings as well as a discussion of the threats to validity. Finally, Section 5 presents our main conclusions and future research topics.

2. Background and related work

In GUI testing, the system is tested through the elements of the GUI and their properties [10]. In other words, GUI testing means that an application that has a graphical user interface (GUI) is tested solely by doing sequences of events to be executed on the widgets of the GUI [23]. GUI testing consists of performing test sequences of actions corresponding to user interactions (clicks, scrolls, keystrokes, etc.) on available elements (buttons, input, etc.) found on the GUI in various states of the System Under Test (SUT).

It is important to clarify that automated GUI testing comprises tools that allow to automate different steps in the testing process. Thus, automated GUI testing considers approaches that are related to (1) automating the creation of test sequences, (2) automating the execution of test sequences, (3) automating the definition of oracles or automating the evaluation of oracles, and (4) automating the analysis of results obtained by the executed test sequences [10]. To automate the creation of these test sequences, GUI testing tools use mainly two approaches: scripted and scriptless.

Generally speaking, in scripted testing, the tester manually creates scripts for each of the desired test sequences, either by using a capture-replay tool or writing the scripts manually. This means that there is a creation of the script by the tester before the automated execution of test cases. Later, these scripts can be used to automate the execution of tests.

In contrast, in scriptless testing, the creation of test sequences to stimulate the SUT is performed automatically by the tool, i.e. the tester does not indicate the elements nor the set of actions that must be executed in the test sequence. This means that the test sequences are generated on the fly in each test run. There is no creation of a script by the tester before the automated execution of the test cases. In scriptless testing, the tester only needs to configure the tool to make sure the SUT is found and the widgets on the GUI are detected.

The next sections will briefly present the scripted and scriptless GUI testing tools; and some related work that compares GUI testing tools in industrial contexts.

2.1. Scripted GUI testing

There are many tools that implement a scripted testing approach. An extensive list with scripted tools is mentioned in [24]. This list is comprised by the tools Abbot, Jacareto, JFCUnit, Marathon, Pounder, Monkey, UI automator, Espresso, Robotism, Appium; Sikuli(X), JAutomate, eggPlant, and EyeAutomate. Moreover, a simple search of tools for automated scripted testing by Google search returns many more, such as: Ranorex, Telerik, LeanFT, Kantu, iMacros for Chrome, Katalon recorder/studio, Robot framework, Protractor, Screenster, Ghost inspector, UFT, QTP, TestComplete, Sahi, Rational function tester, Froglogic squish, among others.

Nevertheless, Selenium [25] is the most popular tool for scripted testing of web applications [26,27], which has been used in academy and in industry, and many scripted testing tools use the selenium webdriver for targeting elements in a web-application [28]. For this reason, we select Selenium as the scripted testing tool in our case study.

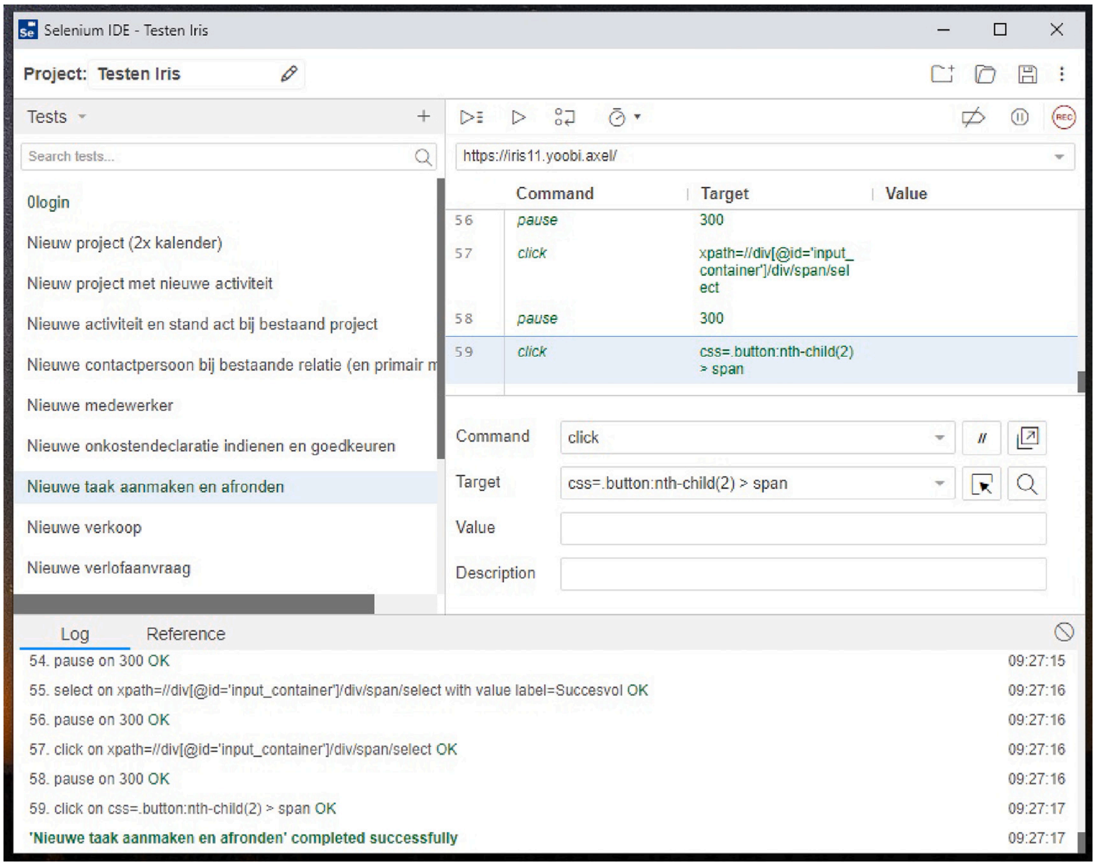
Selenium is an open source tool available as extension in Chrome or Firefox browsers. With this tool it is possible to start a test-session on a webpage or localhost url, and consecutively, it is possible to manually perform user-actions that will be recorded into a test sequence that can be later replayed as a script. GUI elements are detected and saved by using either screen coordinates or element-selectors. After recording the test scripts, it is possible to replay them, and if needed, add or edit commands manually. Element selections can be changed to other given attributes of the target element or added by recording from a specific point. Since GUIs change a lot, the scripts need updating and maintenance to keep them up to date.

An example of an script created with the Selenium IDE tool is shown in Fig. 1. This figure shows a set of test cases in the left part, in green is highlighted the *create and finish new task* test case; in the right part it shows a sequence of commands with the corresponding target values; and at the bottom is presented the log of the execution of the test case.

2.2. Scriptless GUI testing

Scriptless GUI testing tools automatically create and execute test sequences during the testing process. These tools first identify all the available widgets in the GUI. After that, the tool derives all the possible actions that can be performed with those widgets. This information represents the observed actual state of the SUT. Then, the tool selects some of these actions to build the test sequences by using an Action Selection Mechanism (ASM). The most straightforward ASM is random; this procedure is also known as monkey testing, or random testing [29]. Later, the tool executes the test sequences, checks the oracles and again observes the state of the SUT by identifying the widgets and deriving the possible actions to create a new test sequence in an iterative process. If an error is found, the tester can replay the sequence to investigate what happened.

There are less scriptless tools than scripted tools in literature, as can be observed in [24]. We found the following tools that are scriptless: monkeytest.it [30], webui-test, and Testar [31]. In this paper, we select Testar since it is an open source tool that has an active community that is maintaining the tool, and, moreover, it has been successfully used in academy and industry [16,17].



Tests:
- 0login
- Nieuw project (2x kalender)
- Nieuw project met nieuwe activiteit
- Nieuwe activiteit en stand act bij bestaand project
- Nieuwe contactpersoon bij bestaande relatie (en primair m
- Nieuwe medewerker
- Nieuwe onkostendeclaratie indienen en goedkeuren
- Nieuwe taak aanmaken en afronden
- Nieuwe verkoop
- Nieuwe verlofaanvraag

https://iris11.yoobi.axel/

	Command	Target	Value
56	pause	300	
57	click	xpath=//div[@id='input_container']/div/span/select	
58	pause	300	
59	click	css=.button:nth-child(2) > span	

Command: click
Target: css=.button:nth-child(2) > span
Value:
Description:

```
54. pause on 300 OK                                                               09:27:15
55. select on xpath=//div[@id='input_container']/div/span/select with value label=Succesvol OK   09:27:16
56. pause on 300 OK                                                               09:27:16
57. click on xpath=//div[@id='input_container']/div/span/select OK                09:27:16
58. pause on 300 OK                                                               09:27:16
59. click on css=.button:nth-child(2) > span OK                                   09:27:17
'Nieuwe taak aanmaken en afronden' completed successfully                         09:27:17
```

Fig. 1. Excerpt of script created with Selenium tool.

In Testar, the default ASM is random, but more advanced ASMs are available that use Q-learning [32] to learn how to select actions that have not been executed yet, or unvisited actions that prioritizes to execute new actions [33]. Faults are found using oracles: either pre-defined default oracles for (crashes, hangs or suspicious texts) or user-defined failure detection conditions that are SUT dependent.

Testar [13] must be manually configured with the credentials to use (settings), elements not to use (filters) and the conditions that are considered failures (oracles). After that, the test cases are automatically generated at run-time. These test cases are stored, so that they can be replayed later with Testar. Failures found by Testar should be evaluated by the testers on the SUT in order to determinate whether it is a fault and whether it is reproducible.

Testar has been evaluated in several case studies, some performed with large companies, such as Softeam and Cap Gemini, and some with SMEs, such as Prodevelop and Indenova. SUTs used in the evaluations include both desktop and web applications. An overview is given in [13], together with aggregated evidence that Testar is an effective complement to manual testing and it can be used to find undiscovered failures in the SUT. In this work, we select Testar to evaluate the complementarity with the scripted tool Selenium.

2.3. Related work

In [34], a comparison of Sikuli and JAutomate is performed. Both tools use scripted GUI testing. The tools were applied in a Turkish company to compare the quality of record and replay features, robustness and repeatability of test execution, and test development effort. Results shows that JAutomate is slightly better for the Turkish company. This work is not focused on the complementarity of the tools, as is the case of our study. Moreover, this paper does not follow any methodological framework to compare the tools, which later could allow to aggregate the results or make comparisons with other results that can be found in literature.

Eggplant, QF-test, and TestComplete are compared in [35]. This work analyzes the development time and maintenance effort of test suites with these tools. Results indicate that QF-Test is fast for the creation of test suites and EggPlant has a small advantage in the maintenance effort. Even though this paper intends to compare a visual GUI testing tool with two capture and replay tools, all of them are scripted tools, and the work does not follow well-known methodological guidelines to perform the case study. Thus, it is more difficult to aggregate or compare the results in order to facilitate the selection of tools by the companies.

Shtakova presents a comparison of UISpec4J, Fest, and Jubula in [36]. UISpect4J and Fest are scripted tools and Jubula uses scriptless and record/replay testing. These tools were applied to Scila Surveillance application. The comparison was based on a list of criteria defined by the company, such as the feasibility to reproduce the test cases or the ease of configuration of the test cases. However, this study is not focused on the complementarity of tools.

Kumar performs a comparative study of two open source tools: Selenium and SoapUI, and with two licensed tools: HP Unified functional testing and Test complete in [37]. The study aims to analyze the functionality (for instance the ability of the tool to record scripts, or to record and playback), and the active support of the tools, cost of licences, cost for training, among others. But, the study does not present a well-defined design of the case study or clear measures that allows to understand the complementarity of the tools.

A comparison of Selenium and UFT is presented in [38]. This study is focused on the identification of the type of SUT that can be tested with these tools and the execution efficiency. Not enough details are given, so it is difficult to replicate.

An experiment that compares visual GUI testing (with EyeAutomate) and capture and replay GUI testing (with Espresso) for Android applications is presented in [39]. The experiment measure the productivity and the quality of the test suites for both tools with students. Even though the experiment is correctly designed and executed, authors recognize that productivity and quality are not the most suitable metrics to measure ease of use, which becomes a threat to the construction validity of the experiment. For this reason, proper measures to evaluate the ease of use should be followed, such it is stated at [40].

In [41], a comparison of strengths and drawbacks of EyeSel and Selenium for testing web applications is presented. To do that, an experiment is executed in order to measure the effectiveness by counting the number of components tested, and efficiency by the execution time and the development time of one script. Even though the experiment is properly defined and executed, we advocate that there are too few measures considered to describe the effectiveness and efficiency of these tools, which makes it difficult to analyze their complementarity.

A recent study compares the test effectiveness of a capture and replay tool with three automated input generation tools for Android applications [22]. This study presents two experiments performed with master students (as novice engineers) to test four small-size open-source Android applications by using Robotium Recorder (capture and replay), and three other tools: AndroidRipper, Sapienz, and Robotool. This study measures the effectiveness by using the coverage reached by the different approaches. Results indicate that manual testing generally outperforms the automated testing tools, and that the input generation tools have similar coverage than the capture and replay tools. Nevertheless, authors recognize the use of students as a limitation of the work, and that the coverage is not the best metric to compare the effectiveness. They argue that it is necessary to identify the failures found by the different approaches to evaluate the effectiveness as a future work, and also to replicate the experiment in an industrial context. In our work, we tackle the limitations of this study by performing an industrial case study and measuring the effectiveness by focusing on the failures found by the different approaches.

In summary, we found case studies that focus on the comparison of two or more GUI testing tools, but they lack a well-defined process or definition of the corresponding measures, which makes it difficult to replicate or compare them. Moreover, a few studies address the complementarity of scriptless and scripted tools, but they are using small or toy applications in controlled experimental settings. To fill this gap, we perform a case study following a methodological framework to measure the complementarity of two testing tools corresponding to scripted and scriptless approaches, measured by effectiveness, efficiency and subjective satisfaction. The novelty of our work is the case study performed with a real case (not toy project) in a company to evaluate the complementarity of using these approaches without replacing manual testing, which can be useful to smooth the process in the companies that want to improve their software development processes by using automated GUI testing tools with scripted or scripted approaches. We advocate that the results of our case study is one step towards helping companies to take informed decisions in the selection of tools to automate their testing processes taking into account the complementarity of the tools.

3. Case study design

The design of this study follows the guidelines presented in [40], which is a methodological framework specifically designed to evaluate testing tools, and that allows to generate evidence that later can be easily compared and aggregated in secondary studies. This framework has been used already several times to evaluate the scriptless testing tool in industrial settings [13]. In this case study, we use this framework to report a scriptless tool and a scripted tool in an industrial setting, which allows us to further compare the results of this case study with existing studies.

The guidelines in [40] have been defined following well-known evidence-based software engineering guidelines such as [42–45], with a specific focus in the empirical evaluation of testing tools.

3.1. Overall goal of the study and its treatments

This case study is an exploratory study about the complementarity of different approaches (treatments) to perform automated system testing at the GUI level in an industrial context. To this end, this study aims to measure effectiveness, efficiency, and subjective satisfaction of two different automated GUI testing approaches: scriptless testing with the Testar tool and scripted testing using the Selenium tool. With these measures we advocate that it is feasible to evaluate the capability of the tools to work usefully together, i.e. the complementarity of both approaches. It is important to mention that even the tools use different approaches to create the test sequences, they also have commonalities, for instance: both tools perform automated GUI testing, both can be used in web applications, both are applied to system testing and both have failures as output. The tools are described in Table 1 following [40].

3.2. Context: company E-dynamics

The study was performed in an IT development company called E-Dynamics (The Netherlands) [51], a small company with 18 employees. They develop software following agile practices of Scrum [52] and Kanban [53].

To develop software, the company is using the roles defined in Scrum, sprints are planned in two weeks, Kanban board is used to visualize the product backlog items and to assign high priority tickets to a sprint. The development team performs weekly updates, including the finished and tested changes (Kanban). They also perform daily stand-up meetings, and at the end of each sprint, the retrospective meeting (Scrum).

The company has some quality assurance activities in place before the current study started. *Static code checking and inspection* is done weekly, before release. The code is scanned with a linter-tool that is configured for checking an in-company predefined set of rules. Subsequently, new code is inspected manually by two developers following the 4-eyes principle. This process usually takes 2 h per week per developer.

Regression tests are performed by the developers manually. During development of new code, some cases need to be retested after updates to the server, extensions to the code or integration of new packages. At the time of writing this section, there were 12 regression cases to retests known issues in previous versions. No code coverage metrics are measured in the company.

So-called *debug functions* are written by developers to easily check the results of the regression tests. For example, debug functions are created to inspect query results, API results, and user-event-logs in order to find problems faster when doing the regression tests.

System testing is done manually and in an exploratory way by the testers. Testing is performed with different configurations, permissions and settings, observing results, and evaluating the product and its behavior.

Acceptance testing is performed in a similar way, but includes the designer and the product owners. The company does not maintain the acceptance test cases, as commonly occur in agile developments. Therefore, test cases written in the past are mostly outdated due to many reasons, i.e. abundance of documentation of test cases, change of developers, reluctance and no strict company policy to maintain the test cases, etc.

Monitoring through event logs is done by storing all requests by users, APIs or scheduled tasks in a log database. If a problem occurs these logs are used to pinpoint the problem or reproduce steps taken. Kibana [54] is used to show graphs of the event log data that can be useful for statistics.

The testers of the company spend on average 10 h a week on exploratory system and acceptance testing for the parts of the system recently changed. And each developer spends on average 1 h per day in retesting latest changes performed by a colleague on the same or next day. The actual time for testing in each sprint varies, but is around 30% of the time spent on each sprint.

Table 1
Description of the treatments following [40].

Prerequisites	Testar [31] (see Section 2.2)	Selenium [25] (see Section 2.1)
SUT type	Desktop, Web, Android	Web
Lifecycle phase	GUI testing	GUI testing
Environment	Web (Chromedriver) Desktop (Windows) Android (Appium)	Web (Chromedriver)
Input	Protocol and settings for configuring how actions are selected to create test sequences, abstraction and oracles	Scripts containing predefined steps for test sequence including test oracles
Knowledge	Structure of GUI elements, Java programming	Structure of GUI elements
Experience	Advanced tester	Beginner tester
Results		
Output	Test sequences, state model, failures	Failures
Completeness	Investigated by [33] [17] [16]	Depends on the defined scripts
Effectiveness	Investigated by [16–18,33,46–49]	Depends on the defined scripts
Defect types	Failures, crashes, suspicious titles	Depends on the hand crafted oracles
Test suite size	Configured number and length of sequences	Number of predefined test scripts
Operation		
Information/help	Webpage of the tool, contact the developers	Webpage of the tool and stackoverflow
Task applicability	System testing	System testing
Comprehensibility	Investigated by [17].	Investigated by [20].
Satisfaction	Investigated by [16–18,46–49]	Investigated by [50].
Maturity	Academic research tool under development	Widespread use outside of own organization
Obtaining the tool	Open source BSD3, support from researchers.	

3.3. The research questions

In order to study the complementarity of scripted and scriptless testing tools to do the system testing at E-Dynamics, we have formulated the following research questions (RQs):

- RQ1. How effective is it to extend the testing process with Testar and Selenium?
- RQ2. How efficient is it to extend the testing process with Testar and Selenium?
- RQ3. How satisfied are subjects in learning, configuring and applying Testar and Selenium?

The main idea behind this case study is to compare two approaches to perform automated system testing – scriptless and scripted – in order to obtain knowledge about the complementarity and the benefits of these two approaches, and later, to extend the current manual testing process of E-Dynamics with them. RQ1 is precisely formulated to obtain knowledge about the effectiveness of scriptless and scripted testing through the use of Testar and Selenium tools and the amount of faults found by these tools. It is important to note that we are not using coverage to evaluate the effectiveness of the automated testing tools since in the state of the art was demonstrated that it is not the best metric to understand effectiveness. Indeed, related works suggest the use of failures to evaluate effectiveness. RQ2 is related to evaluating the efficiency of using automated GUI testing, which take into account the learning curve of Testar and Selenium; and the execution time to generate/develop and run the test cases. We put special focus in the learning and configuration time of he tools to evaluate the efficiency since the execution time depends on the infrastructure of the company; and since the execution is automated with the tools, it outperforms the manual execution of test cases. Finally, RQ3 was formulated to know the perceptions of subjects related to the ease of use of both automated GUI testing tools. We advocate that if the subjects are satisfied learning, configuring and applying the tools, then it facilitates the extension of manual testing in the company with one or both tools.

3.4. Object

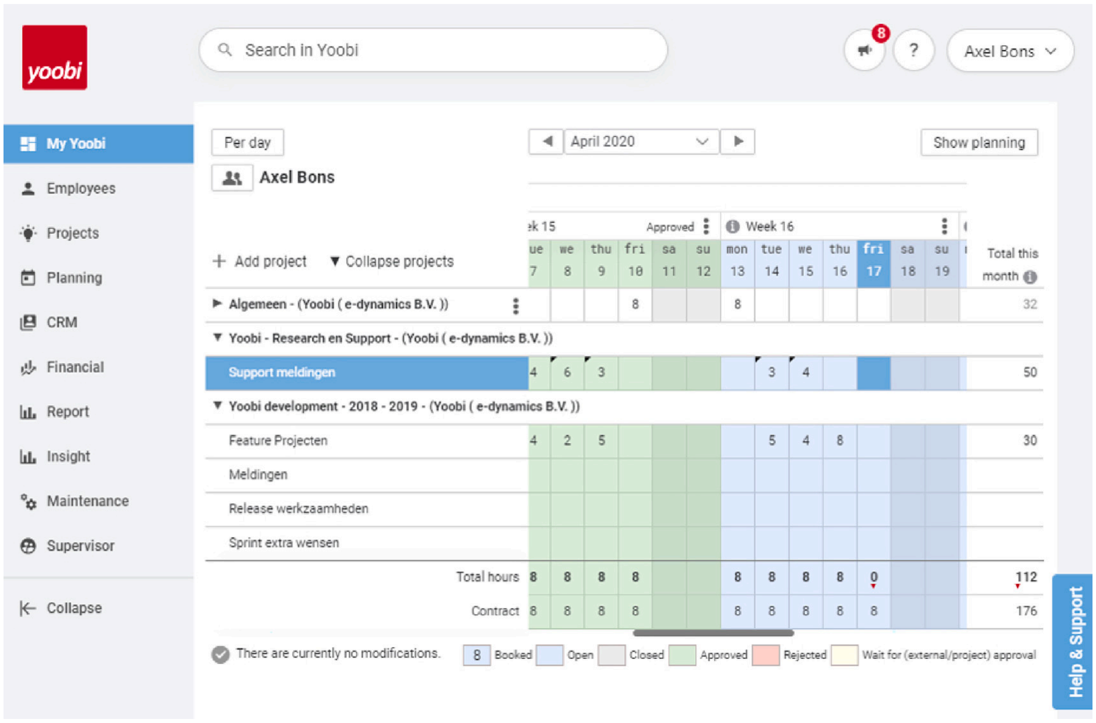
The object of the study is a software product called Yoobi. Yoobi is a time tracking web and mobile application with over 32k users, with supported languages in Dutch, English, German and French. The first version of Yoobi started in September 2011, and it has been evolving since that. Yoobi is used for tracking work hours, costs, managing projects, budgets and customers, leave hours, overtime, tasks, planning, sales and billing. Fig. 2 shows an excerpt of Yoobi application.

There are four different types of users of Yoobi:

- Employees: they communicate worked hours, apply for leaves and complete tasks.
- Project managers: they manage budgets, rates, schedule and approve employee's hours.
- Sale managers: they manage sales, send quotes and create orders of customer approved.
- Financial managers: they manage orders and billing of approved hours, subscriptions or products quotes.

Yoobi users are part of an organization or a company who purchased a subscription of Yoobi. Each organization has its unique domain where their users can login. The subscription(s) of an organization contains a set of modules giving access to different parts of Yoobi or adding functionality to existing parts. The amount of permissions and settings for an organization depends on the modules in use. These permissions contain the possibility to create, read, update and delete access which can be coupled in a user-, project- or department-role and assigned to users within the organization. The settings contain object default values, API connection settings, email, language and system alterations for organizations, and some settings can be overruled for departments and users.

Development of Yoobi is performed following an agile approach, where small feature requests and correction of faults are deployed weekly, and bigger features are planned in 2 week sprints. The current version is 2.3.4, the last digit will change weekly with every new hotfix release, containing small changes. The middle digit usually changes



	ek 15					Approved	Week 16							
	ue 7	we 8	thu 9	fri 10	sa 11	su 12	mon 13	tue 14	we 15	thu 16	fri 17	sa 18	su 19	Total this month
▶ Algemeen - (Yoobi (e-dynamics B.V.))				8			8							32
▼ Yoobi - Research en Support - (Yoobi (e-dynamics B.V.))														
Support meldingen	4	6	3					3	4					50
▼ Yoobi development - 2018 - 2019 - (Yoobi (e-dynamics B.V.))														
Feature Projecten	4	2	5					5	4	8				30
Meldingen														
Release werkzaamheden														
Sprint extra wensen														
Total hours	8	8	8	8			8	8	8	8	0			112
Contract	8	8	8	8			8	8	8	8	8			176

There are currently no modifications. 8 Booked ▭ Open ▨ Closed 🟩 Approved 🟥 Rejected 🟨 Wait for (external/project) approval

Fig. 2. Excerpt of Yoobi application that shows the timesheet.

Table 2
Prior experience of subjects (S1–S6) participating in this study.

Experience	S1	S2	S3	S4	S5	S6
Product knowledge	x	x				
Customer Support		x				
TESTAR	x					
Selenium	x		x			
Senior Developer	x					
Developer				x	x	
Novice Developer			x			
UX designer						x

after a sprint period, containing bigger changes. The first digit does not change that often.

Nowadays, Yoobi is supporting 1828 organizations and 32 959 users. In order to give an insight of the size of Yoobi, the total number of files of Yoobi is 4344; the total lines of code is 616 336; the number of tables in the database is 433 and the number of records is 172 211 387. This information demonstrate that Yoobi is not a small application, as was observed in the related works.

In summary, Yoobi is a big application that supports critical management actions for 32k users. It is developed following agile practices and it is tested with manual exploratory testing. Yoobi is an object representative for real industrial applications, which nowadays are commonly developed by several developers using agile practices. Yoobi contains real faults (not injected by the researchers) and test cases are not maintained to lighten the testing process (i.e.; there is not a repository with the test cases and the corresponding results for each release), provoking that more effort is needed for testing the next versions of the application. Therefore, the faults that the versions of Yoobi may have, and the time needed to find them are not biased by the researchers.

3.5. Selection of subjects

The subjects of this case study corresponds to people with knowledge of Yoobi as well as people with knowledge about the testing process.

Six subjects were involved in this study. Two testers whose daily work includes manual testing of Yoobi, one UX designer, one support person and two fullstack developers. These last four subjects do not have expertise on scriptless nor scripted testing.

There is one subject that has some beginner experience in using the scriptless testing (Testar) and 2 subjects with experience in scripted testing (Selenium). Since scriptless and scripted are automated GUI testing approaches, we advocate that the selection of these subjects are enough to manage the tools and extract the results.

Table 2 displays a summary of the subjects per initials participating in this study with their prior experience.

3.6. The design of the study

We execute the study in three different phases: (1) the set-up and learning phase, (2) the testing phase, and (3) the subjective evaluation phase.

The *set-up and learning phase* follows a process to download, install and learn how to use both testing tools. The learning is hands-on and iterative to configure the tools with the creation of specific scripts for Selenium (see Fig. 3) and specific oracles for Testar (see Fig. 4). It is important to mention that to use Selenium, the subject must configure the url of the SUT, and then the subject performs user actions on the elements of the GUI, which are recorded in a script that can be executed later. In addition, the subject can select a previously recorded script and modify it by adding more actions or changing the sequence of

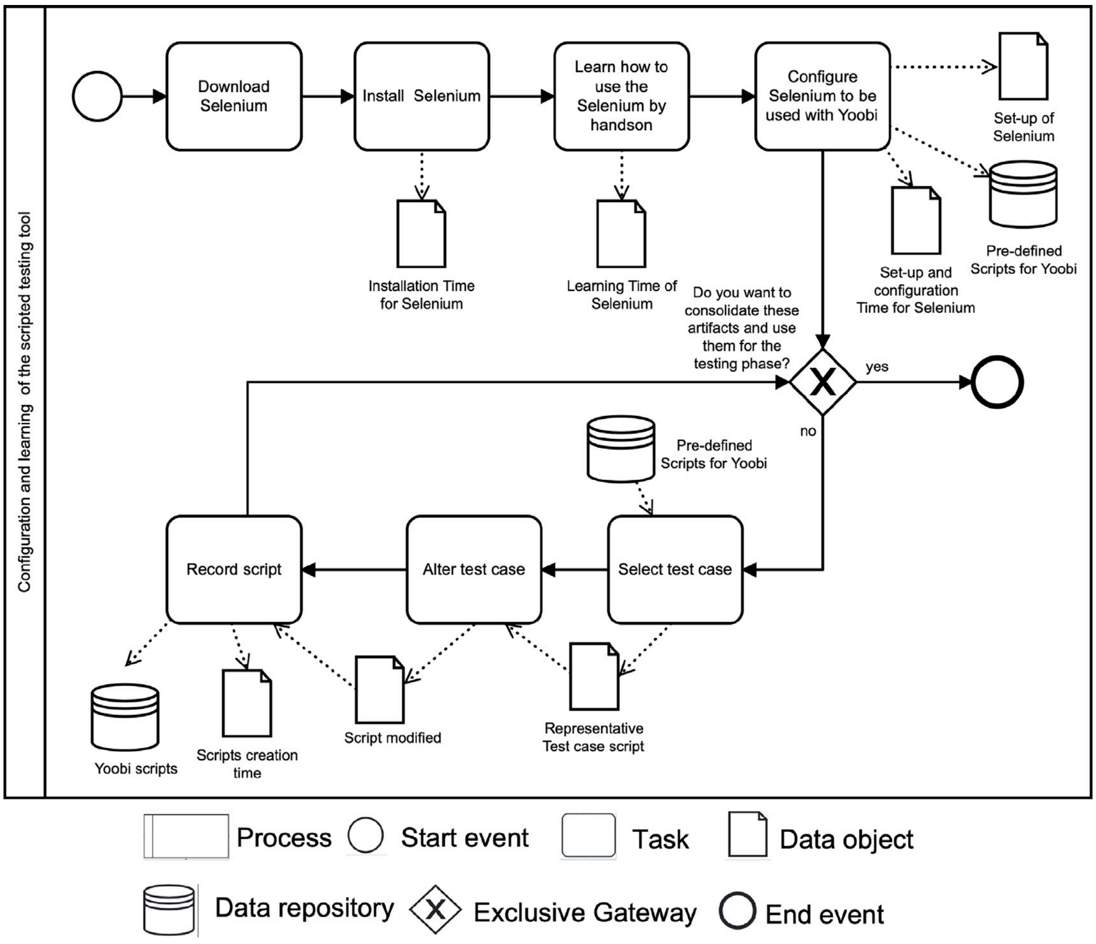

Configuration and learning of the scripted testing tool
Download Selenium
Install Selenium
Learn how to use the Selenium by handson
Configure Selenium to be used with Yoobi
Set-up of Selenium
Pre-defined Scripts for Yoobi
Installation Time for Selenium
Learning Time of Selenium
Set-up and configuration Time for Selenium
Do you want to consolidate these artifacts and use them for the testing phase?
yes
no
Pre-defined Scripts for Yoobi
Record script
Alter test case
Select test case
Yoobi scripts
Scripts creation time
Script modified
Representative Test case script
Process
Start event
Task
Data object
Data repository
Exclusive Gateway
End event

Fig. 3. Configuration and learning process for the scripted testing tool Selenium.

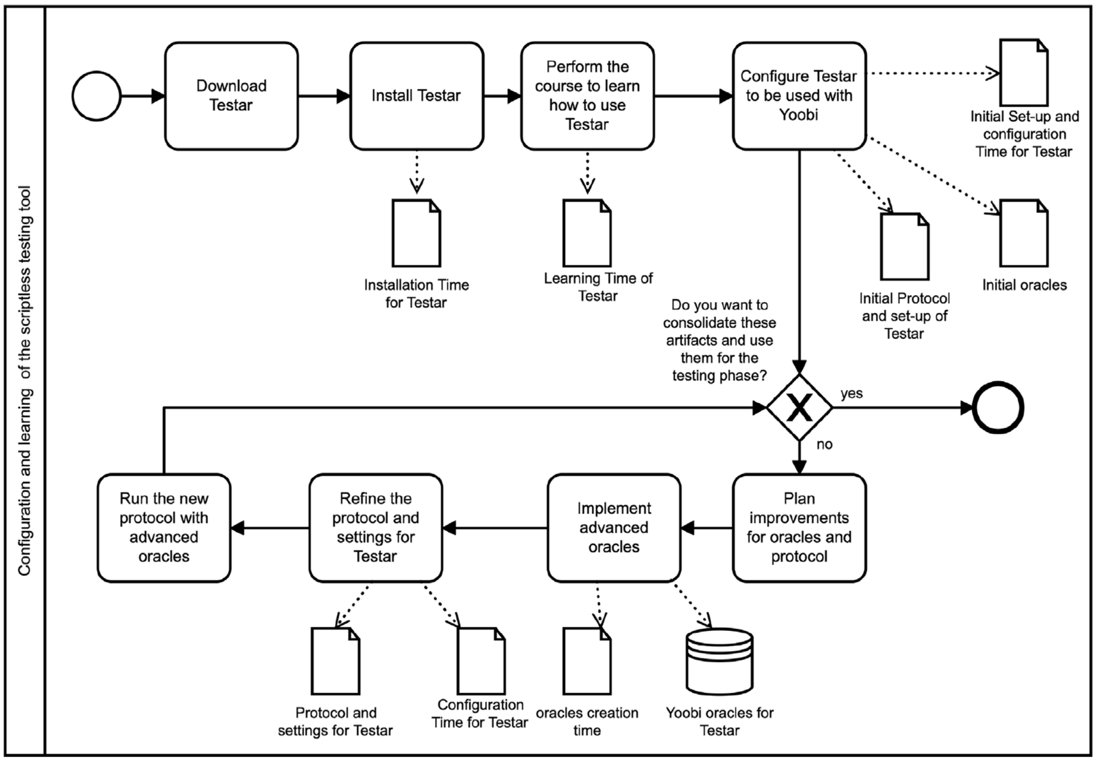

Configuration and learning of the scriptless testing tool
Download Testar
Install Testar
Perform the course to learn how to use Testar
Configure Testar to be used with Yoobi
Initial Set-up and configuration Time for Testar
Installation Time for Testar
Learning Time of Testar
Initial Protocol and set-up of Testar
Initial oracles
Do you want to consolidate these artifacts and use them for the testing phase?
yes
no
Run the new protocol with advanced oracles
Refine the protocol and settings for Testar
Implement advanced oracles
Plan improvements for oracles and protocol
Protocol and settings for Testar
Configuration Time for Testar
oracles creation time
Yoobi oracles for Testar

Fig. 4. Configuration and learning process for the scriptless testing tool Testar.

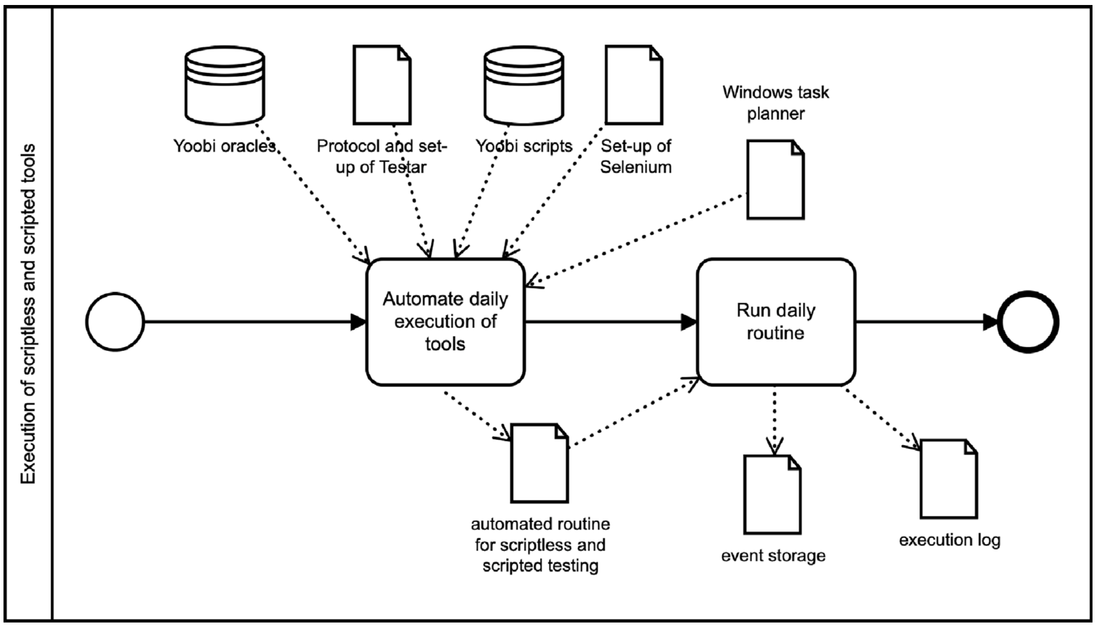

Execution of scriptless and scripted tools
Yoobi oracles
Protocol and set-up of Testar
Yoobi scripts
Set-up of Selenium
Windows task planner
Automate daily execution of tools
Run daily routine
automated routine for scriptless and scripted testing
event storage
execution log

Fig. 5. Daily execution process of the scriptless and scripted testing tools.

actions. Regarding learning how to use Testar, the subject configures the url of the SUT, and then the subject must select an action selection mechanism (ASM) that the tool will use to create the test cases and subsequently execute the test cases. Moreover, the subject can add new oracles, for instance, domain specific oracles, which is performed using the facilities of the tool. The learning process ends when the subjects consider they have learned enough and they want to consolidate the configurations, oracles, and scripts they have created. Although this is a subjective criterion, this is how most new tools are deployed in companies.

The *testing phase* consists of a process that runs the consolidated configurations of the first phase and analyze the results obtained by the tools (see Figs. 5 and 6). Both tools are used during the automation of a daily execution routine.

The last phase, i.e. the *subjective evaluation phase*, consists of a process to measure the subjective satisfaction of the subjects.

As Figs. 3 and 4 show, the configuration process start with downloading and installing the scripted and the scriptless automated GUI testing tools, Selenium and Testar, respectively. After that, hands-on learning starts to learn how to use these tools. Testar provides a hands-on course available at its website.[1] In this study we used the version of February 2020. In contrast, Selenium does not provide a getting starting guide or a course, it just provides some guidelines and bad practices. Both automated GUI testing tools must be configured to be used with Yoobi. Both tools need to be configured such that they are able to login to the SUT with specific user credentials. Moreover, for Testar it is important to configure *Settings* (i.e. which protocol to use, chromedriver location and url of the website); and to define a *Protocol* (allowed domains, clickable/ignored elements, login, cookie consent, etc.). For the configuration of Selenium, it is important to assign the url of Yoobi; and create a set of scripts that can be used as basis for the generation of other scripts for Yoobi.

After that, subjects should access the ready to use tools. Subjects start testing with the scripted tool, and after that, they use the scriptless tool. For scripted testing (Selenium), subjects select one of the representative manual test cases (test cases that validate the reachability and responsiveness of important parts of the SUT), and record the script to automate the execution of the test case. These activities are performed iteratively while the subject wants to create another test case. For scriptless testing (Testar), the protocol is refined and specific oracles for advanced fault detection are defined.

For the testing phase, the execution of the tools is automated in the Windows task planner and the test routines are executed, resulting on an execution log and an event storage, as it is shown in Fig. 5.

After both tools have been executed, an analysis of the failures is performed by the subjects. The analysis consist of: understanding the failure, determining if it has been analyzed previously in order to leave it as a duplicate. If the failure is not a duplicate, it is important to look for the root cause of the failure, assign a priority, and create a Jira ticket for the fault (see Fig. 6). Later, developers will modify the code of Yoobi to correct the issues reported in Jira.

Regarding the subjective evaluation, the process consists of the preparation of the material to perform the interviews with the possible questions and the reaction cards, and then, iteratively, the selection of a subject to interview and the video recording of the interview.

The configuration and learning process should be performed at the beginning of the study. The subjective evaluation process should be performed at the end of the study. These two processes are performed outside the sprints. Nevertheless, the execution process and the analysis of failures should be performed during each sprint of the development of Yoobi.

3.7. Data collected to measure effectiveness

The following dependent variables are collected during the test execution and used to answer the RQs.

– Num_F: The number of failures found during the daily test cycle.
– Type_F: Type of failures, for instance, the following failures are commonly found at E-Dynamics during system and acceptance testing process of Yoobi:

  • missing labels, which mainly occur since developers tend to write the Dutch texts first and forget to add the label for English, French or German.
  • lost changes, which occur by merge conflicts (overlapping code in version management).
  • wrongly used permissions or settings.

[1] https://testar.org/download/

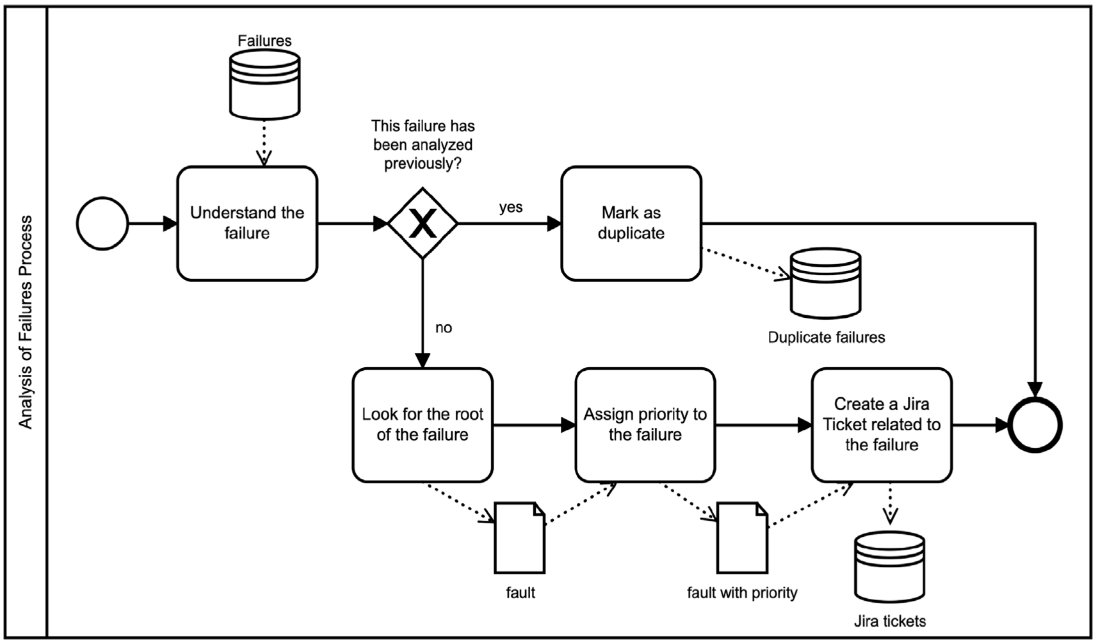

Analysis of Failures Process
Failures
Understand the failure
This failure has been analyzed previously?
yes
no
Mark as duplicate
Duplicate failures
Look for the root of the failure
Assign priority to the failure
Create a Jira Ticket related to the failure
fault
fault with priority
Jira tickets

Fig. 6. Analysis of failures process for scriptless and scripted testing.

– Sev_F: Severity of failures
– Ev_Cov: Event coverage, calculated with the use of the existing event logging. The coverage is measured as the percentage of all executed events from all existing events (this includes deprecated or non-reachable events that exists because of the software's age) for new users which are only used by Selenium or Testar (or to replay a fault).

3.8. Data collected to measure efficiency

To measure efficiency of the studied methods, we measure the manual effort required to learn, configure and apply both automated testing method. For this aspect, we consider the following variables:

– MEf_Learn_T: Manual effort to learn how to use Testar, measured in hours.
– MEf_Conf_T: Manual effort to set-up and configure Testar (webdriver, url, login) measured in hours.
– MEf_Ora_T: Manual effort to add custom oracles on Testar based on website elements and text (SUT specific) measured on hours.
– MEf_Learn_S: Manual effort to learn how to use Selenium, measured in hours.
– MEf_Conf_S: Manual effort to set-up and configure Selenium (browser plugin, url, login) measured in hours.
– MEf_Rec_S: Manual effort to record (or write) and replay test scripts in Selenium measured in hours.
– MEf_Ana_F: Manual effort for analyzing the failure
– MEf_Conf_Sche: Manual effort to set-up a fully automated daily test cycle using windows scheduler for Testar and Selenium measured in hours.

3.9. Data collected to measure subjective satisfaction

To measure the subjective satisfaction we use interviews related to the perception of subjects about the learnability, maintainability, usefulness and intention of use of the automated testing tools. We use semi-structured interviews, where a set of questions is previously defined, and one interviewer ask the questions to at least one respondents and has the possibility to clarify the meaning of each question [55]. The questions were answer with a 5-point Likert scale. We use the following questions to guide the interviews:

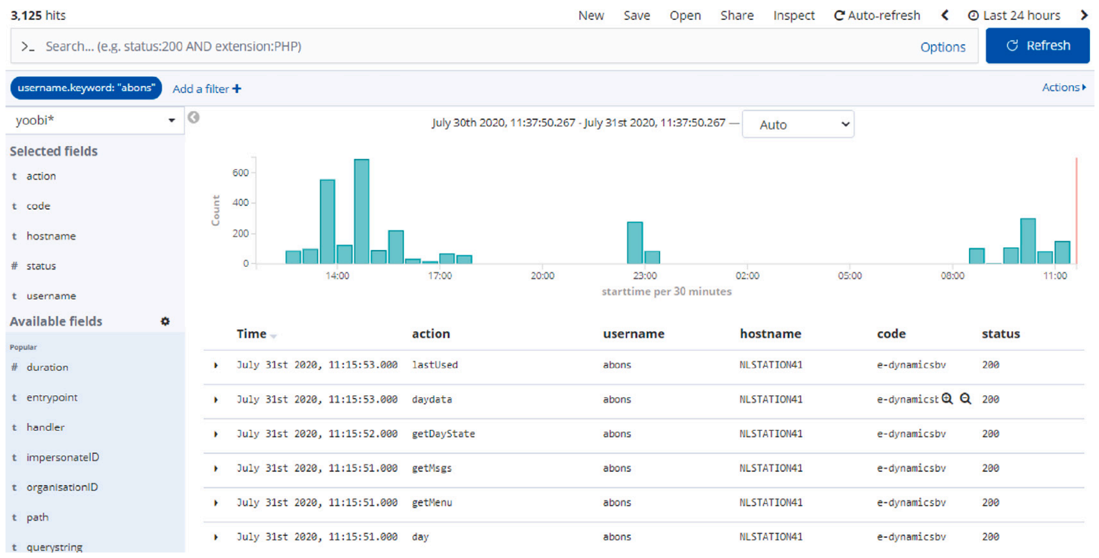
1. Would you recommend Selenium to new colleagues. Explain.
2. Would you recommend Testar to new colleagues. Explain.
3. Could you persuade management to invest in Selenium? Explain.
4. Could you persuade management to invest in Testar? Explain.
5. Do you consider easy to configure Selenium at Yoobi?
6. Do you consider easy to configure Testar at Yoobi?
7. Do you consider easy to create and manage testcases/oracles in Selenium?
8. Do you consider easy to create and manage testcases/oracles in Testar?
9. Do you consider easy to understand test reports in Selenium?
10. Do you consider easy to understand test reports in Testar?
11. Do you consider the bug quality of Selenium is better than manual testing?
12. Do you consider the bug quality of Testar is better than manual testing?

Moreover, we use reaction cards to gain knowledge of subjective satisfaction as shown in [56]. The subjects needs to select a maximum of 3 reaction cards from the listed 55 cards proposed in [56] to describe their perceptions about Testar and Selenium.

3.10. Data collection procedures

When an empirical study is performed it is important to obtain reliable information about the phenomenon studied. To do that, it is often best to use multiple data collection procedures to learn about different aspects of the phenomenon [57]. After that, to analyze the data, it is important to triangulate the data to increase the precision of empirical research [58,59]. Data triangulation, which refers to use of more than one data source to collect the same data in different occasions, is used since we have different versions of Yoobi that are measured. Moreover, we use two different techniques for automated GUI testing: scriptless testing performed by using Testar, and scripted testing performed by using Selenium.

The time measurements for efficiency during the learning and the testing phase (the independent variables in Section 3.8), were collected manually in a time tracking excel sheet describing when and how long each activity took. The use of Yoobi time tracking is mandatory within the company, so time is manually added in Yoobi on activities. Testar



Time	action	username	hostname	code	status
July 31st 2020, 11:15:53.000	lastUsed	abons	NLSTATION41	e-dynamicsbv	200
July 31st 2020, 11:15:53.000	daydata	abons	NLSTATION41	e-dynamicst	200
July 31st 2020, 11:15:52.000	getDayState	abons	NLSTATION41	e-dynamicsbv	200
July 31st 2020, 11:15:51.000	getMsgs	abons	NLSTATION41	e-dynamicsbv	200
July 31st 2020, 11:15:51.000	getMenu	abons	NLSTATION41	e-dynamicsbv	200
July 31st 2020, 11:15:51.000	day	abons	NLSTATION41	e-dynamicsbv	200

Fig. 7. Monitorization of events with Kibana.

and Selenium now both have corresponding activities added to the time tracking system. Manual testing already had an activity called "testing". The time tracked is also copied to an excel sheet with a description and link to the action.

The measurements for effectiveness during the testing phase (the independent variables in Section 3.7) are maintained in Jira like is common practice at the company. To extract failure information from Testar, the Testar logs were used together with an automated script to obtain unique failures and not too many duplicates.

Event coverage (Ev_Cov) is measured by logging all executed events (i.e. events in the code reached from the interactions in Yoobi) in an AWS database such that they can be monitored in Kibana as shown on Fig. 7. We collect the id of the automated user by using the event usage logging. Both Selenium and Testar tools have their own unique login setting in the AWS database, so that all the events logged could be filtered using the login credentials (id of the user). The filtered logs also contain the name of the event, which will be matched with each available event of the system to find the coverage for each day and the total coverage after a month.

To measure the subjective satisfaction we performed informal taped interviews with the subjects. Each subject was interviewed separately by one of the authors of this paper. After that, all the interviews were reviewed to obtain the answers to the questions and to gather the information of the reaction cards. We used a table to register the answers to the questions and the selected reaction cards. For each question, the explanations of the subjects regarding the grade assigned were analyzed to obtain deeper knowledge of the perceptions of the subjects.

4. Results

This section presents the details of the execution of the study, the answers to the research questions, the analysis and interpretation of the findings, and an analysis of the threats to the validity of our study.

4.1. Execution of the set-up and learning phase of the study

Selenium

The first 4 activities of Fig. 3 took 14 h and included a demo to the whole team of 6 subjects showing initial scripts for Yoobi. After download and install, the subjects studied the starting guide[2] that is rather basic and not sufficient to create reusable scripts for the test cases, e.g., the API documents describe the available commands but without examples. Therefore, the only way left is to start using Selenium and learn by trial and error. Most questions that arise while using Selenium can be answered by FAQs, for example, Stackoverflow has over 80k selenium and 2k selenium-ide questions, but looking for a similar question and the corresponding answer is time consuming.

After the demo, 3 iterations were done before it was decided that the artefacts (i.e. the test scripts) could be consolidated for the next phase. The *first iteration* resulted in an initial set of test cases with Selenium, one of the subjects created 16 scripts in 2 h by recording them with Selenium IDE. In this phase, some failures found with Selenium were not SUT related and required changes in the scripts of the test cases. The *second iteration* fixed some problems that were encountered when trying to replay these scripts. This iteration took 4 h by another subject to fix the element selection and reset actions. The *third iteration* that took 6,5 h, included a demo session with the team of 6 subjects. Here it was decided that five of the test scripts were abandoned because they were too data-dependent and difficult to make replayable for daily usage at E-Dynamics.

The consolidated 11 test scripts were related to the following features of Yoobi: login, relation, contact, employee, expense, leave, project, activity, sale and task. In total, 6 h of manual effort was spent to properly create the initial set of 11 replayable Selenium scripts. It took less than 2 min to execute the scripts, while executing manually the same tests would take about 30 min.

Testar

The first 4 activities of Fig. 4 took 13,5 h that included a demo to the whole team showing initial workings of Testar on the Yoobi SUT. After download and install, the getting started manual was studied. This manual is very detailed with examples and exercises. In comparison to the FAQs of Selenium, there is no Stackoverflow discussion for Testar, but there are over 200 questions available in Github and questions can be asked to the open source team.

After completion of the first 4 activities from Fig. 4, 4 iterations were done before it was decided to consolidate the artefacts and continue to the next phase.

The *first iteration*, that took 5 h, started with configuring Testar. WebDriver generic Testar protocol was selected as a template to be adapted for the SUT. First, the ChromeDriver location and the URL of the SUT had to be set. Other SUT specific configuration included dealing with the SUT's allowed domains, the cookie consent, the login

[2] https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started

and the elements that can be ignored. Finally, Testar's configuration was set to generate 160 sequences per day with maximum of 20 actions per sequence. During the first runs many failures were found. It required quite some analysis to figure out if the failure was SUT related or not. The logical approach is to replay the faulty sequence, but the replay sometimes failed (actions executed outside of the browser) in the Testar version used for this study.[3] Because the replay was not always working, it was decided to use the generated HTML report file to see the last action(s) resulting in the error state in an attempt to replay them manually. However, during the first runs many duplicated failures were found, and manual failure evaluation using the HTMLs was very time-consuming. It was decided that for the next iteration the number of sequences per day should be reduced to 80 and automate part of the failure evaluation.

During the *second iteration*, that took 3 h, the manual effort for investigating the many (duplicate) failures problem was reduced by writing a script that could create a custom error-log for Yoobi by automatically filtering the Testar result logs.

During the *third iteration*, that took 6,25 h, new Testar test oracles were defined to find failures from the SUT states specific for Yoobi. The default oracles in Testar include checking predefined attributes of all the GUI elements for suspicious text defined as a regular expression, e.g., HTTP error codes to detect when the links are not working or detecting other error messages. The default test oracles were not enough to detect some of the failures that the developers were aware of and should detectable when they happen in Yoobi. The following oracles were added to Testar for Yoobi before consolidation:

– Oracle 1. Console oracle: Stop on error log from the browser console.
– Oracle 2. Class oracle: Stop on a specific class selector, e.g., a debug element which should not be visible.
– Oracle 3. Text oracle: Stop on a specific text, e.g., a fallback text for a missing translation.

During the *fourth iteration*, that took 1,75 h, in order to test the new Testar oracles, we injected 3 failures into Yoobi (one for each oracle), which were detected by both oracle 1 and 3. These oracles were coded into the getVerdict() function of the webdriver protocol, which is automatically executed at the end of each action. By using the webdriver library it is possible to find browser elements, search in the html-text and read console messages. If this data contains unwanted values a new verdict can be raised by using a different oracle, resulting in new failure conditions for the test execution.

Consolidated artefacts

At the end of the learning phase, all subjects were shown a demonstration of both tools where the consolidated set-up and configurations of the tools were presented as follows:

– Testar was configured as follows:
  • one test run consists of 80 sequences per day with maximum of 20 actions per sequence.
  • oracles: default oracles + the 3 new oracles explained above
  • action selection was random
– Selenium was set on daily execution to test with the above-mentioned 11 test cases reaching the most important components of the SUT.

[3] Note: this is fixed in the latest Testar version

Including in the daily routine (MEf_Conf_Sche)

To allow automatically executing the testing approaches and include them in the daily test routine as depicted in Fig. 5 for Selenium this costs 3,25 h and for Testar 4,25 h.

To run the scripts with Selenium IDE, we used selenium-side-runner which provides a command line approach that can be used in Windows taskplanner. Most of the time spent was dedicated to solving an issue with the execution speed of the selenium-side-runner. The main problem was that the execution speed to run the test cases from the IDE is faster than the time required for the execution of actions, and the wait-for-element actions did not work. During this case study, we had to do a workaround to make it usable, which corresponds to adding pause actions whenever an element is not instantly available because of an animation or a long loading time.

Testar was executed from the command line scheduled with Windows taskplanner. In addition to the settings files, Testar allows defining any number of the configuration settings also as command line parameters during the start up.

Regarding the execution of the tools (cf. Fig. 5), Selenium was used for specific components of Yoobi targeted by the created scripts. In contrast, Testar generated test scripts for the entire SUT, although the sequence length was at maximum 20 actions. The subjects run the Selenium test scripts every night and check the results the next morning. Subjects also run multiple Testar sequences every night, and check the results at the end of the sprint.

The testing phase consisted of running the routine overnight from 11-03-2021 till 15-04-2021. Regarding the failures found by Selenium, their analysis was performed at the beginning of every day in the daily meeting. For the failures found by Testar, the analysis was performed at the end of every sprint. Every week the results were evaluated as depicted in Fig. 6. This resulted in 5 evaluation sessions that took approximately 1 h for Testar and 0,2 h for Selenium.

4.2. Effectiveness of Testar and Selenium

As outlined in Section 3.7 effectiveness (RQ1) was determined during the testing phase and by the amount (Num_F), type (Type_F) and severity (Sev_F) of failures found by each automated testing tool, plus the event coverage (Ev_Cov).

The findings regarding the failures found are shown in Table 3. If the failure was found manually before the tool execution, the "Known" column is marked with x. Selenium and Testar can find GUI failures caused by the latest changes or already existing failures.

As we can see from Table 3, only failure **2** is found by both tools and that failure was one of the injected ones. The remaining 11 unique failures are only found by either Selenium or Testar. This demonstrates that the approaches cannot replace each other, instead they are complementary to each other. Using both can improve a testing process since they are finding different types of failures:

– Selenium testing scripts are needed to test whether the common use scenarios are implemented well
– Testar tests are needed to find faults in the less probable paths of the application.

We also analyzed the severity of the failures found. Severity was assigned based on usability and visibility. High severity was assigned when the system was not responding. Low severity was assigned if end-users would not see or are not able to reach the failure.

Selenium found **4** failures of high severity, **2** unknown to manual testers. The failures found with Selenium are all process failures that had to be fixed.

Testar found **9** failures, **2** of high severity, and **6** were unknown to manual testers. The failures found with Testar are state failures, some of them did not have to be fixed immediately, as the processes from that state often still worked as intended, so that they were classified as low severity.

Table 3
Failures detected at E-Dynamics (Yoobi product) by using scriptless and scripted testing.

id	Selenium	Testar	Known	Severity	Description
1		Console	x	none	Injected javascripterror to test custom verdict.
2	Task	Class	x	none	Injected cfdump output in code to test custom verdict class detection.
3		Text	x	none	Injected new label to test custom verdict, missing translation text detection.
4	Project		x	high	Project edit/create does not work after API changes.
5		Console		high	Javascript error when switching list detail view to only detail
6		Text		low	Missing label in supervisor menu.
7		Title		low	Two suspicious titles for flexmonter (js report plugin).
8	Expense			high	Expense add error because of debug data in data-call.
9	Task			high	Failure when editing and creating a task due to last release change.
10		Text		low	Missing translation in optional filter for activity in bulkmutation.
11		Console		high	Javascript error when opening the tour.
12		Text		low	Never used missing translation in source code.

Event coverage was measured with existing event logs. Events are the functions accessed from the interface, also called the controller in the model-view-controller pattern. The Selenium test cases combined had an event coverage of **3.06%**. Testar reached **8.37%** coverage in the same period.

4.3. Efficiency of Testar and Selenium

Efficiency was measured by the manual effort required for applying scriptless and scripted testing method. In addition to the manual effort spent in learning and set-up phase (MEf_Learn_T, MEf_Conf_T, MEf_Ora_T, MEf_Learn_S, MEf_Conf_S, MEf_Rec_S), we measured the manual effort required for analyzing each failure to reproduce it (if possible) and to find the root cause of the failure (MEf_Ana_F), and the maintenance effort for the test scripts (although, the time period was too short and test set too small for analyzing maintenance effort properly).

In total, including set-up and learning phase, Selenium took **27,5** hours. We have to point out that the test set included 11 test cases, and creating more test cases will be required in the future to increase the coverage.

In total, including set-up and learning phase, Testar took **34,5** hours, a big part spent in learning and configuring and creating new test oracles for the SUT.

At the same time period, the manual testing took **32,5** hours written on the activity *testing* in the Yoobi time tracking system during the same period.

Efficiency was determined by the work log, rounded to the half hour per session. Automated time per test session was tracked manually and completed with the log details for single test sequences from the tool. The total hours of manual effort spent by the subjects can be seen in Table 4.

Results indicate that the time spent on manual testing is similar (just a few hours of difference) with both tools. Even though the time measured does not allow us to draw strong conclusions regarding efficiency, we advocate that the time spent with the scriptless and the scripted tools can be considered as learning time, which we predict will be reduced in the future, improving the efficiency of the testing process of the company over time. In any case, with the invested effort, both tools found unique faults that were not discovered by the manual testing.

From this case study, we gain some lessons learned about how to measure the efficiency of testing when we are using automated tools. First of all, we should focus on the measurement of the learning, set-up and analysis of results time instead of the time needed for the execution of the test sequences. From this study we have preliminary evidence that these measures can be more meaningful than the execution time. Moreover, a broader period to perform the study could be beneficial to measure the knowledge gain of using the tools over time, and to have that broader period it is necessary that the company has enough resources to facilitate the inclusion of new technologies without harming its business. Another lesson related to the efficiency measurement is that the adoption of new tools and their inclusion on the company's routines should be performed with a research team and later propagate the results known to the entire company.

4.4. Subjective satisfaction while learning, configuring and applying Testar and Selenium?

We interviewed the subjects participating in the study in order to gain knowledge about their perceptions at using a scripted tool and a scriptless tool in the testing process at E-Dynamics regarding the Yoobi product. It is important to mention that six subjects participated in the study, but one of the interviews could not be recorded, so that we analyze the interviews of five subjects. The answers of subjects are presented in Fig. 8.

It is interesting to note that none of the subjects disagree or strongly disagree to recommend the tools to their colleagues. In contrast, they answer neutral, agree or strongly agree to recommend the tools to their colleagues. For instance, subject S_4 states "*I would recommend the tools because it provides extra certainty before releasing new versions*".

Moreover, the majority of subjects could persuade managers to invest in the tools. However, one subject stated that he disagrees or strongly disagrees to persuade to invest in the tools Testar and Selenium, respectively. Subject S_3 states: "*it is still uncertain if I could persuade management to invest*". From this answer we can observe that the subject focused the answer in the capability to persuade instead of interest in investing in the tools. Moreover, the subject strongly disagrees to persuade to invest in Selenium and just disagrees to persuade

Table 4
Total hours spent by the subjects.

			Subjects					
		hours	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*
Selenium	learn	14	4	2	2	2	2	2
	iteration 1	2		2				
	iteration 2	4	4					
	iteration 3	6,5	2	1	1	1	1	1
	testing phase	1	1					
	Total	***27,5***						

			Subjects					
		hours	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*
TESTAR	learn	13,5	8,5	1	1	1	1	1
	iteration 1	5	5					
	iteration 2	3	3					
	iteration 3	6,25	6,3					
	iteration 4	1,75	1,8					
	testing phase	5	3,5			1		1
	Total	***35***						

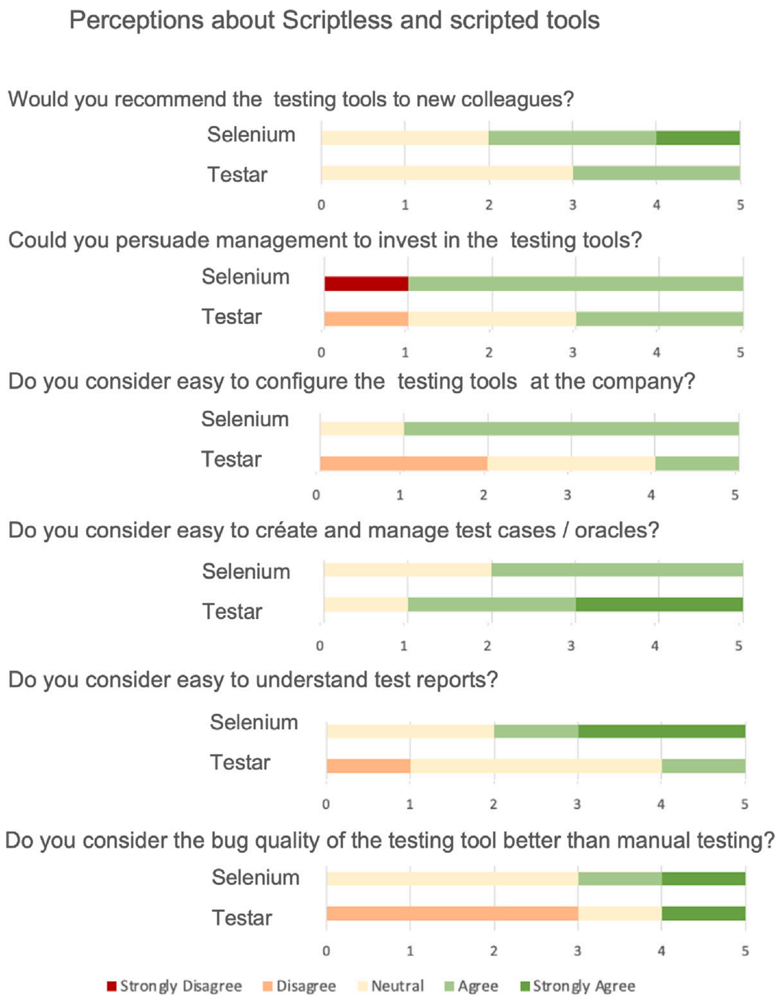


Fig. 8. Perceptions of using scriptless and scripted testing tools at E-Dynamics.

to invest in Testar, so we can observe a positive tendency to have more interest in Testar than Selenium.

Regarding the question whether they considered it easy to configure the tools at Yoobi, the majority of subjects agreed that Selenium is easy to configure. Subjects S_5 states: "*You can get a lot of testing done with minimal configuration and it is very easy to use, it might need some extra time to configure it for specific cases, though*".

We observe that one subject (S_3) answered neutral to this question regarding Selenium and states that: "*I like how Selenium is accessible as browser plugin, but I don't like the required alterations to make it work from CLI*".

Moreover, subject S_6 states that: "*I find the configuration time-consuming, because it does not pick the best selector in our situation.*" although the subject agreed it was easy to configure Selenium.

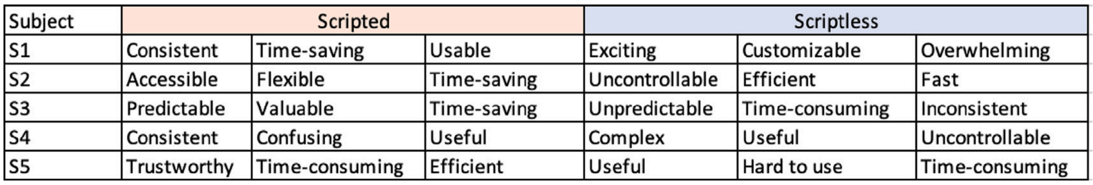
Regarding the ease to configure Testar at Yoobi, 2 subjects strongly disagreed. One of the subjects (S_6) stated that "*I'm convinced of the usefulness of Testar, but it requires a lot of time to configure it*".

The other subject (S_5) that strongly disagrees says: "*Testar is nice to use as a testing tool, but I wouldn't use it as my only testing tool. It takes a bit more time to configure correctly, but once you do it you probably won't have to touch it anymore.* "

This comment brought us to understand that the initial configuration could require more time, but later it is not necessary to configure it again.

Regarding the managing of test cases and oracles, the majority of subjects agreed or strongly agreed that it is easy to do it with both tools. For instance, S_4 states "*it is easy/intuitive to add new test cases in Selenium*", and "*Testar oracles are useful and preferable with random testing,* "*because of the extra maintenance it would be for scripted test cases*". Another subject (S_3) states *The Testar oracle to detect JavaScript" errors is useful. The oracles are preferable scriptless, because not all is covered with scripts*".

The majority of subjects agreed or strongly agreed that the reports of Selenium are easy to understand. Regarding Testar, one of the subjects



Subject	Scripted			Scriptless		
S1	Consistent	Time-saving	Usable	Exciting	Customizable	Overwhelming
S2	Accessible	Flexible	Time-saving	Uncontrollable	Efficient	Fast
S3	Predictable	Valuable	Time-saving	Unpredictable	Time-consuming	Inconsistent
S4	Consistent	Confusing	Useful	Complex	Useful	Uncontrollable
S5	Trustworthy	Time-consuming	Efficient	Useful	Hard to use	Time-consuming

Fig. 9. Reaction cards selected by subjects for the testing tools.

(S_5) stated that "*it works fast and gets results, but the reporting could be a bit better*". Other subject (S_4) stated "*Testar results were time-consuming to handle due to many duplicates*".

Regarding the quality of bugs found by the tools, the majority of subjects disagreed or strongly disagreed that Selenium or Testar are better than manual testing. For instance, one subject (S_4) stated "*Selenium can be high maintenance after visual changes, which are no problem for Testar*".

At the end of each interview, the interviewer tells the subjects that they must select 3 reaction cards based on their observation and experience to define their perception of each tool from a total of 55 physical cards presented to the respondents.

The reaction cards that subjects selected are presented in Fig. 9 in the order that they select them. We can observe positive cards like useful selected for both tools, and negative cards such as time-consuming also selected for both tools. Nevertheless, we cannot make a strong conclusion about the perceptions for scripted and scriptless testing tools by using the cards, so that we plan to replicate this study to obtain more knowledge by assigning a weight to the cards selected and diminishing the possible bias in this part of the study.

4.5. Threats to validity

Even though we followed well-known guidelines to conduct this case study, there are some threats to the validity of our results that are worth to mention.

Regarding construct validity, we decided to perform semi-structured interviews to the subjects in order to have the possibility to clarify the questions. Nevertheless, during the interviews we did not perceive that one of the questions was not fully understood by one of the subjects, which is a threat to our results regarding subjective satisfaction.

Regarding internal validity, we identify that low priority faults found by Testar tool were not immediately fixed, which provokes many duplicates. We observe that one of the subjects point this in his perception of the report. With enough resources these problems could have been fixed earlier resulting in less duplicates. Nevertheless, E-Dynamics is a small company, and it was not possible to assign more resources during the evaluation period. In contrast, these duplicates did not appear in Selenium because the scripts were created with only high priority assertions, so that the found bugs were fixed immediately. For this reason, we advocate that the perception of easy to understand the reports could be threatened for Testar tool.

During the testing period, only small changes were made to the components for which the scripted test cases were made. Thus, the time for maintenance of Selenium scripts could not be measured, even though subjects recognize that it could be complicated later.

Regarding external validity, we are conscious that only a few subjects participated in the study. However, considering that the company is small (only 18 employees) and the impact in the daily work of the company for using the subjects, we could not use more subjects in the study.

The configuration and usage of Selenium and Testar for the case study took place within the same weeks, so the resulting failures from both tools are based on the same time frames and on the same version of the SUT.

Moreover, the subjects had zero experience with automated GUI testing tools but are experienced programmers with a bachelor's degree in Information Technology. The most useful experience they had for using these test tools was CSS and jQuery, because of the similarity in element selectors used. This demonstrated the feasibility of using these tools by inexperienced people.

Regarding the SUT, we selected a SUT that has been developed for 12 years with continuous improvements and extensions with modern technologies. This provokes that some parts of the system are server-side-rendered of pure HTML pages with little JavaScript and some parts are client-side-rendered using JavaScript framework VueJS. Considering that the SUT is a web application, only web-based testing is used for the comparison of the automated tools in order to extends the manual testing. Nevertheless, the testing tools can also be used in other domains such as mobile or desktop applications.

Moreover, we are aware that the period of using the tools is too short for long-term benefits. Thus, the configuration and learning time required after years of using the tools could be very different. We think the manual time needed for Testar would decrease as it is not dependent on current state of the SUT, while Selenium needs to be extended and altered after every change of the GUI.

4.6. Interpretation of findings, answering the RQs

The goal of this paper was to explore the complementary benefits of using scriptless testing with the Testar tool as well as scripted testing with the Selenium tool in the IT company e-Dynamics, which performed only manual testing in their software products before this study.

Results show that after **62** hours in learning, preparation and evaluation, **9** unique failures were found, not counting the injected failures.

Moreover, regarding the *Effectiveness* of tools (RQ1), Selenium demonstrated to be good in detecting process failures. Testar demonstrated to be good in detecting visible failures per state and also it can reach higher event coverage. Selenium requires more scripts to reach higher event coverage and detecting visible failures per state. It is difficult to detect process failures with Testar, as defining state based test oracles for detecting them is difficult. Thus, the tools work well together, complementing each other.

With respect to the *Efficiency* (RQ2), Selenium requires a good understanding of web-element selectors for script creation. The time needed for script creation depends on the script-size and previous experience. Evaluating failures is easy and takes less than a minute, as the browser will wait in the failed state. Testar benefits from SUT specific oracles that were added. To add oracles for a web application, knowledge of Java is required, as the default oracles were not enough to detect all detectable failures of Yoobi. Besides adding oracles, Testar does not require much time in configuration, as there is no need for script creation. In contrast, failure evaluation took more time for Testar. The main challenge with the efficiency of scriptless test generation that became apparent in this case study are the duplicate reports of the same failure. Analyzing the results at the end of sprint, compared to daily analysis of Selenium script results, made it worse since there were so many failures to analyze. Therefore, one of the main topics to deal with in the near future will be automated detection and handling of duplicate failures.

Even though the human effort required to incorporate the scripted tool or the scriptless tool in the company is similar, this effort is referring to different activities that concerns to the specific workflows of the tools, e.g. Selenium needs more time for creating test cases and Testar needs more time in evaluating test reports. The reported results show the complementarity of the tools because both tools improve the effectiveness of the testing process in different ways due to the type of failures found are different: Selenium is better for detecting if a given process is still working as intended, and Testar is better for detecting unwanted elements, because it can reach many states automatically without first creating test cases manually and adding assertions in each state.

Finally, with respect to *Subjective satisfaction*, both tools were perceived to be useful in the interviews to the subjects (see Fig. 8). Testar is useful in finding failures easy to miss, because of the focus in manual testing on the certain parts of the system that were added or changed. Selenium provides extra certainty of important processes, captured in a test script, to remain working at all time.

5. Conclusions

Automated GUI testing has been the focus of many researchers during the last years. Companies interested in including automated testing to their processes need to know the benefits and drawbacks of existing proposals. There are studies that evaluate scriptless tools in real contexts, scripted tools in real contexts, and scarce gray literature exists about the evaluation of scriptless and scripted testing in real contexts. In this paper, we fill this gap by presenting evidence about the complementarity of scriptless testing with scripted testing in an IT company that before the study only performed manual testing in their products.

In the case study, we evaluated the effectiveness, efficiency and subjective satisfaction using two tools, Selenium for scripted testing and Testar for scriptless testing, respectively. Results show that the Selenium is better in detecting process failures and the Testar is better in detecting visible failures and also reached higher event coverage. Both tools performed similarly in terms of efficiency, both requiring similar amount of effort for finding 2 unique high severity faults that were not discovered by manual testing. Moreover, both Testar and Selenium were perceived to be useful for the E-Dynamics company. Therefore, we can conclude that the scriptless and scripted approaches are complementary, and they can improve the manual testing processes performed in industrial contexts.

In order to continue using the scriptless tool in Yoobi, an immediate future work is related to the improvement of duplicate failures recognized by the scriptless tool in order to reduce the manual time required for analyzing them. This can be done by adding an automated filter in the Testar results. We also want to develop a scheduled task that reads the filtered log results and sends an email to developers if new failure records are found. We advocate that these tiny improvements can provoke a big enhancement in the testing process at E-Dynamics.

In order to continue using the scripted tool in Yoobi, a future work is related to including an activity to write test scripts as part of the development process. We plan to assign all the subjects of the study to write test cases for the new mobile web app of Yoobi that is currently being developed. We advocate that this could benefit the testability of the SUT, because the developers will have to include testable attributes, and also it could increase the event-coverage at the same rate with the SUT's growth.

In this case study, we used a web application as the object. In order to generalize our results, we want to replicate this case with different widely used types of SUTs, such as desktop, mobile and XR applications. Moreover, we want to compare the usability of automated GUI testing tools in agile development processes (by executing the tools in CI/CD pipeline after code changes instead of once a day) as well as traditional development processes, and also taking into account distributed environments for the execution of the GUI testing tools.

CRediT authorship contribution statement

Axel Bons: Software, Investigation, Validation, Writing – original draft. **Beatriz Marín:** Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing. **Pekka Aho:** Software, Resources, Writing – original draft. **Tanja E.J. Vos:** Conceptualization, Methodology, Investigation, Writing – original draft, Writing – review & editing, Supervision.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.infsof.2023.107172. Pekka Aho reports financial support was provided by Information Technology for European Advancement. Beatriz Marin reports financial support was provided by European Commission.

Data availability

The data that has been used is confidential.

Acknowledgments

This research has been funded by iv4XR, H2020, grant number 856716 (https://iv4xr-project.eu), and IVVES, ITEA, contract number 18022 (https://ivves.eu/) projects.

References

[1] M. Andreessen, Why software is eating the world, Wall Street J. 20 (2011) (2011) C2.
[2] M. Tuteja, G. Dubey, et al., A research study on importance of testing and quality assurance in software development life cycle (SDLC) models, Int. J. Soft Comput. Eng. (IJSCE) 2 (3) (2012) 251–257.
[3] S.A. Slaughter, D.E. Harter, M.S. Krishnan, Evaluating the cost of software quality, Commun. ACM 41 (8) (1998) 67–73.
[4] H.V. Gamido, M.V. Gamido, Comparative review of the features of automated software testing tools, Int. J. Electr. Comput. Eng. 9 (5) (2019) 4473.
[5] I. Rana, P. Goswami, H. Maheshwari, A review of tools and techniques used in software testing, Int. J. Emerg. Technol. Innov. Res. 6 (4) (2019) 262–266.
[6] F. Ferreira, J.P. Diniz, C. Silva, E. Figueiredo, Testing tools for configurable software systems: A review-based empirical study, in: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, 2019, pp. 1–10.
[7] A. Arora, M. Sinha, Web application testing: A review on techniques, tools and state of art, Int. J. Sci. Eng. Res. 3 (2) (2012) 1.
[8] P.K. Singh, O. Sangwan, A. Sharma, A systematic review on fault based mutation testing techniques and tools for Aspect-J programs, in: 2013 3rd IEEE International Advance Computing Conference, IACC, IEEE, 2013, pp. 1455–1461.
[9] M. Shafique, Y. Labiche, A systematic review of state-based test tools, Int. J. Softw. Tools Technol. Transfer 17 (1) (2015) 59–76.
[10] O. Rodríguez-Valdés, T.E.J. Vos, P. Aho, B. Marín, 30 Years of automated GUI testing: a bibliometric analysis, in: International Conference on the Quality of Information and Communications Technology, Springer, 2021, pp. 473–488.
[11] R.M. Moreira, A.C. Paiva, M. Nabuco, A. Memon, Pattern-based GUI testing: Bridging the gap between design and quality assurance, Softw. Test. Verif. Reliab. 27 (3) (2017) e1629.
[12] D. Amalfitano, A.R. Fasolino, P. Tramontana, B.D. Ta, A.M. Memon, MobiGUITAR: Automated model-based testing of mobile apps, IEEE Softw. 32 (5) (2014) 53–59.
[13] T.E.J. Vos, P. Aho, F. Pastor, O. Rodriguez-Valdes, A. Mulders, Testar – Scriptless testing through graphical user interface, J. Softw. Testing Verif. Reliab. 31 (3) (2021).
[14] P. Aho, T. Kanstrén, T. Räty, J. Röning, Automated extraction of GUI models for testing, in: Advances in Computers, vol. 95, Elsevier, 2014, pp. 49–112.
[15] E. Alégroth, R. Feldt, P. Kolström, Maintenance of automated test suites in industry: An empirical study on visual GUI testing, Inf. Softw. Technol. 73 (2016) 66–80.

[16] H. Chahim, M. Duran, T.E.J. Vos, P. Aho, N.C. Fernandez, Scriptless testing at the GUI level in an industrial setting, in: International Conference on Research Challenges in Information Science, Springer, 2020, pp. 267–284.
[17] S. Bauersfeld, T.E.J. Vos, N. Condori-Fernández, A. Bagnato, E. Brosse, Evaluating the TESTAR tool in an industrial case study, in: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2014, pp. 1–9.
[18] M. Martinez, A.I. Esparcia, U. Rueda, T.E.J. Vos, C. Ortega, Automated localisation testing in industry with test*, in: IFIP International Conference on Testing Software and Systems, Springer, 2016, pp. 241–248.
[19] M. Leotta, D. Clerissi, F. Ricca, C. Spadaro, Repairing selenium test cases: An industrial case study about web page element localization, in: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, IEEE, 2013, pp. 487–488.
[20] I. Singh, B. Tarika, Comparative analysis of open source automated software testing tools: Selenium, sikuli and watir, Int. J. Inform. Comput. Technol. 4 (15) (2014) 1507–1518.
[21] V. Jain, K. Rajnish, Comparative study of software automation testing tools: OpenScript and selenium, Int. J. Eng. Res. Appl. 8 (2) (2018) 29–33.
[22] S. Di Martino, A.R. Fasolino, L.L.L. Starace, P. Tramontana, Comparing the effectiveness of capture and replay against automatic input generation for android graphical user interface testing, Softw. Test. Verif. Reliab. 31 (3) (2021) e1754.
[23] I. Banerjee, B. Nguyen, V. Garousi, A. Memon, Graphical user interface (GUI) testing: Systematic mapping and repository, Inf. Softw. Technol. 55 (10) (2013) 1679–1694.
[24] Y. Min, S. Cai, Comparing different approaches of GUI testing for mobile applications on android platform, 2018.
[25] Selenium homepage, 2021, https://www.selenium.dev, (Accessed 21 April 2021).
[26] B. García, M. Gallego, F. Gortázar, M. Munoz-Organero, A survey of the selenium ecosystem, Electronics 9 (7) (2020) 1067.
[27] M. Cerioli, M. Leotta, F. Ricca, What 5 million job advertisements tell us about testing: a preliminary empirical investigation, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, 2020, pp. 1586–1594.
[28] M. Niranjanamurthy, R. Arun Kumar, M.R. Sahana Srinivas, Research study on web application testing using selenium testing framework, Int. J. Comput. Sci. Mob. Comput. 3 (10) (2014) 121–126.
[29] P. Aho, Automated State Model Extraction, Testing and Change Detection Through Graphical User Interface (Ph.D. thesis), UNIVERSITATIS OULUENSIS, 2019.
[30] Monkey test it homepage, 2022, https://monkeytest.it/readme, (Accessed 31 May 2022).
[31] Testar homepage, 2022, https://testar.org. (Accessed 31 May 2022).
[32] A. Esparcia-Alcazar, F. Almenar, M. Martınez, U. Rueda, T.E.J. Vos, Q-learning strategies for action selection in the TESTAR automated testing tool, in: Proceedings of the 6TH International Conference on Metaheuristics and Nature Inspired Computing, ISBN: 978-3-319-47443-4, 2016, pp. 174–180, URL https://meta2016.sciencesconf.org/ 6th International Conference on Metaheuristic and Nature inspired Computing, META 2016 ; Conference date: 27-10-2016 Through 31-10-2016.
[33] A. van der Brugge, F. Pastor-Ricós, P. Aho, B. Marín, T.E.J. Vos, Evaluating testar's effectiveness through code coverage, Actas Las XXV J. IngenieríA Del Softw. Y Bases de Datos (JISBD 2021) (2021) 1–14.
[34] V. Garousi, W. Afzal, A. Çağlar, İ.B. Işık, B. Baydan, S. Çaylak, A.Z. Boyraz, B. Yolaçan, K. Herkiloğlu, Comparing automated visual GUI testing tools: an industrial case study, in: Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing, 2017, pp. 21–28.
[35] A. Kresse, P.M. Kruse, Development and maintenance efforts testing graphical user interfaces: a comparison, in: Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation, 2016, pp. 52–58.
[36] M. Shtakova, Evaluation of methods for automated testing in large-scale financial systems, Uppsala Universitet, 2012.
[37] Y. Kumar, Comparative study of automated testing tools: selenium, soapui, hp unified functional testing and test complete, J. Emerging Tech-Nologies Innov. Res. 2 (9) (2015) 42–48.
[38] P. Kunte, D. Mane, Automation testing of web based application with selenium and HP UFT (QTP), Int. Res. J. Eng. Technol. (IRJET) 6 (2017) 2579–2583.
[39] L. Ardito, R. Coppola, M. Morisio, M. Torchiano, Espresso vs. eyeautomate: An experiment for the comparison of two generations of android gui testing, in: Proceedings of the Evaluation and Assessment on Software Engineering, 2019, pp. 13–22.
[40] T.E.J. Vos, B. Marín, M.J. Escalona, A. Marchetto, A methodological framework for evaluating software testing techniques and tools, in: 2012 12th International Conference on Quality Software, IEEE, 2012, pp. 230–239, http://dx.doi.org/10.1109/QSIC.2012.16.
[41] H. Jiang, Y. Chen, Comparison of different techniques of web GUI-based testing with the representative tools selenium and EyeSel, 2017.
[42] H. Do, S. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact, Empir. Softw. Eng. 10 (4) (2005) 405–435.
[43] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.
[44] B. Kitchenham, S. Linkman, D. Law, DESMET: a methodology for evaluating software engineering methods and tools, Comput. Control Eng. J. 8 (3) (1997) 120–126.
[45] B. Kitchenham, L. Pickard, S.L. Pfleeger, Case studies for method and tool evaluation, IEEE Softw. 12 (4) (1995) 52–62.
[46] F.P. Ricós, P. Aho, T.E.J. Vos, I.T. Boigues, E.C. Blasco, H.M. Martínez, Deploying TESTAR to enable remote testing in an industrial CI pipeline: a case-based evaluation, in: International Symposium on Leveraging Applications of Formal Methods, Springer, 2020, pp. 543–557.
[47] S. Bauersfeld, A. de Rojas, T.E. Vos, Evaluating rogue user testing in industry: an experience report, in: 2014 IEEE Eighth International Conference on Research Challenges in Information Science, RCIS, IEEE, 2014, pp. 1–10.
[48] P. Aho, T.E. Vos, S. Ahonen, T. Piirainen, P. Moilanen, F.P. Ricos, Continuous piloting of an open source test automation tool in an industrial environment, J. IngenieríA Softw. Y Bases Datos (JISBD) (2019) 1–4.
[49] T.E.J. Vos, P.M. Kruse, N. Condori-Fernández, S. Bauersfeld, J. Wegener, Testar: Tool support for test automation at the user interface level, Int. J. Inform. Syst. Model. Des. (IJISMD) 6 (3) (2015) 46–83.
[50] S.A. Sualim, N.M. Yassin, R. Mohamad, Comparative evaluation of automated user acceptance testing tool for web based application, Int. J. Softw. Eng. Technol. 2 (2) (2017).
[51] E-dynamics homepage, 2021, https://www.yoobi.eu/, (Accessed 21 April 2021).
[52] K. Schwaber, Agile Project Management with Scrum, Microsoft Press, 2004.
[53] D. Anderson, et al., The Principles of the Kanban Method, David J. Anderson & Associates, 2010.
[54] Kibana homepage, 2021, https://www.elastic.co/es/kibana/, (Accessed 13 sept 2021).
[55] F. Shull, J. Singer, D.I. Sjø berg, Guide to Advanced Empirical Software Engineering, Springer, 2007.
[56] J. Benedek, T. Miner, Measuring desirability: New methods for evaluating desirability in a usability lab setting, Proc. Usability Professionals Assoc. 2003 (8–12) (2002) 57.
[57] T.C. Lethbridge, S.E. Sim, J. Singer, Studying software engineers: Data collection techniques for software field studies, Empir. Softw. Eng. 10 (3) (2005) 311–341.
[58] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2) (2009) 131–164.
[59] L. Bratthall, M. Jørgensen, Can you trust a single data source exploratory software engineering case study?, Empirical Softw. Eng. 7 (1) (2002) 9–26.