



Empowering Students with Modern Skills and Connections Through Open Source GUI Testing with TESTAR

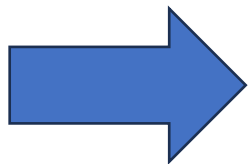
Tanja Vos, **Beatriz Marín**, **Olivia Rodríguez-Valdés**, Lianne Hufkens, Fernando Pastor

21-09-2023



We live in a globalized world

- Speed up movements and Exchange (goods, technology, etc.)
- Increases interactions between different regions



Collaboration
and knowledge
sharing is crucial



Team
Tools



TESTAR team

- **Phd. Tanja Vos**
 - Professor Software Engineering
 - Open Universiteit (OU)
 - Universitat Politècnica de Valencia (UPV)
- **Phd. Beatriz Marín**
 - Senior Researcher at UPV
 - Universitat Politècnica de Valencia
 - Universidad Diego Portales
- **Olivia Rodríguez Valdés**
 - PhD student at the OU
- **Niels Doorn**
 - PhD student at the OU
 - Docent NHL Stenden University of Applied Sciences
- **Fernando Pastor Ricós**
 - PhD student at the UPV
- **Ramón de Vries**
 - Open Source solutions at OU
- **Lianne Hufkens**
 - PhD student at the OU
- **Parsa Karimi**
 - PhD student at the OU

Research on Software Testing



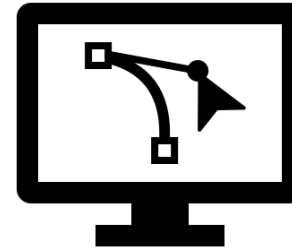
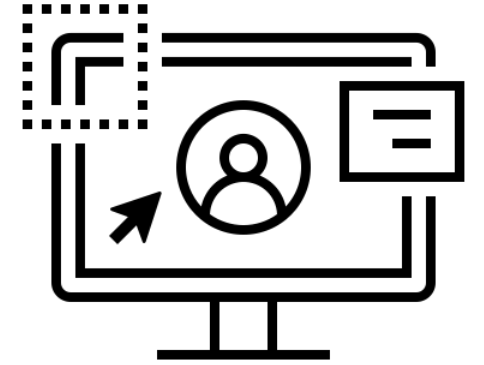
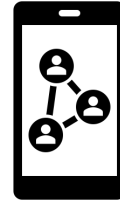
AUTOMATED TESTING AT THE
GUI LEVEL: TESTAR



IMPROVING SOFTWARE TESTING
EDUCATION: ENACTEST

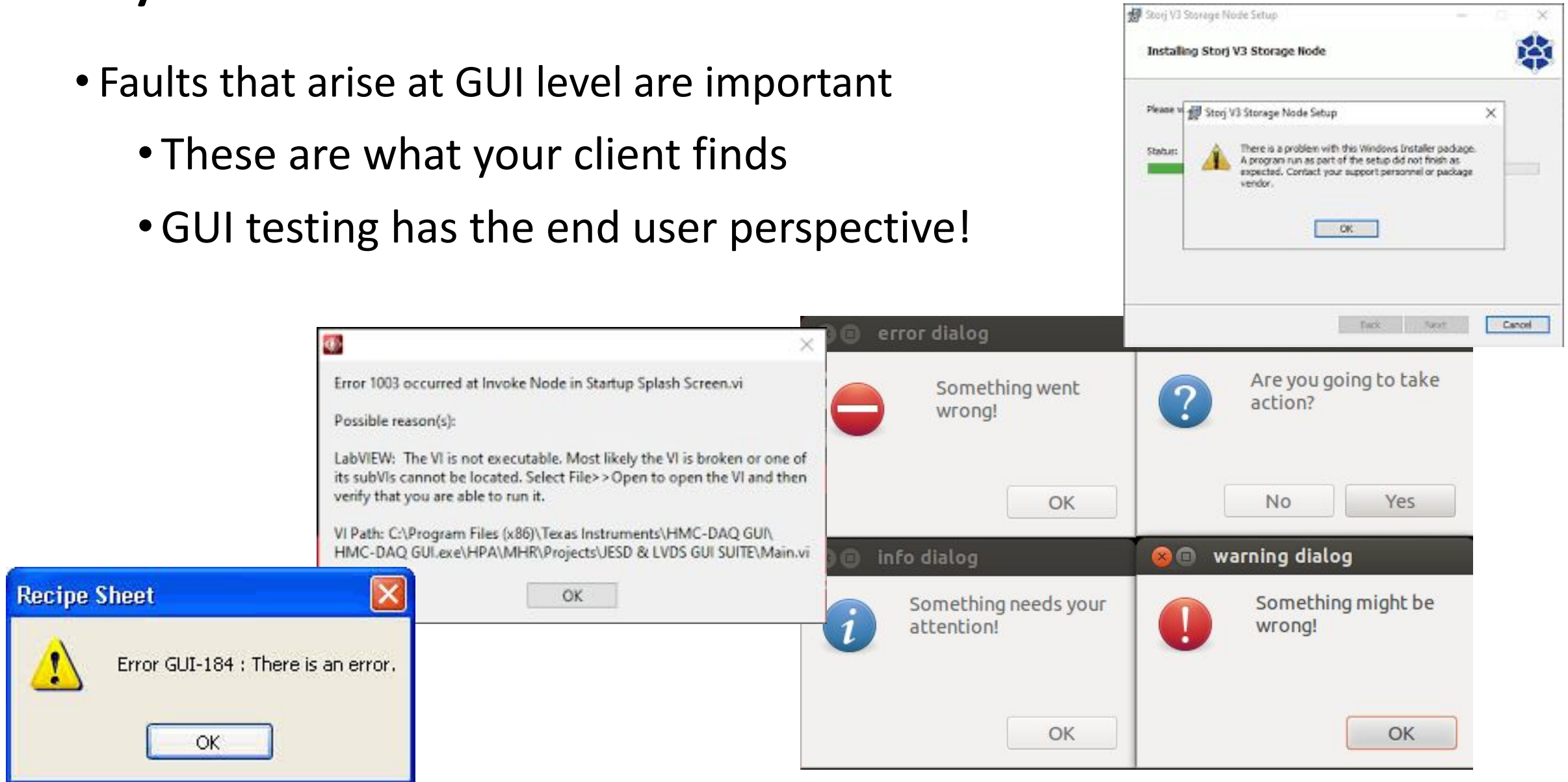
Why testing at the GUI level

- Most applications have graphical user interfaces
 - Computers, tablets, smartphones....
 - Even safety critical applications



Why test at the GUI level

- Faults that arise at GUI level are important
 - These are what your client finds
 - GUI testing has the end user perspective!



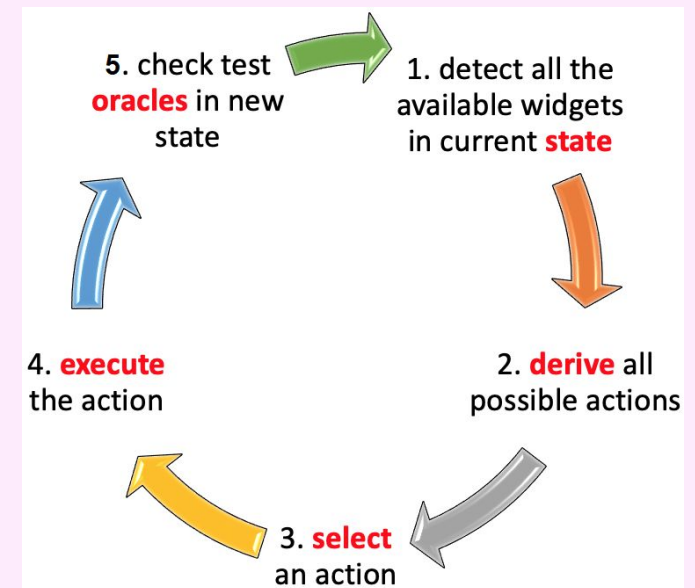
Scripted

Test cases are **manually defined** prior to test execution and the steps are automated in a script

Step 1
↓
Step 2
↓
Step 3
↓
Step 4
↓
Step 5

Scriptless

Test steps are **generated** during test execution based on available actions



Scripted GUI testing – automated test execution

- Pre-defined sequence of test steps
 - Scripts usually manually created (or captured)
 - Test oracles:
 - Assertions with expected output values
 - Each check has to be separately specified

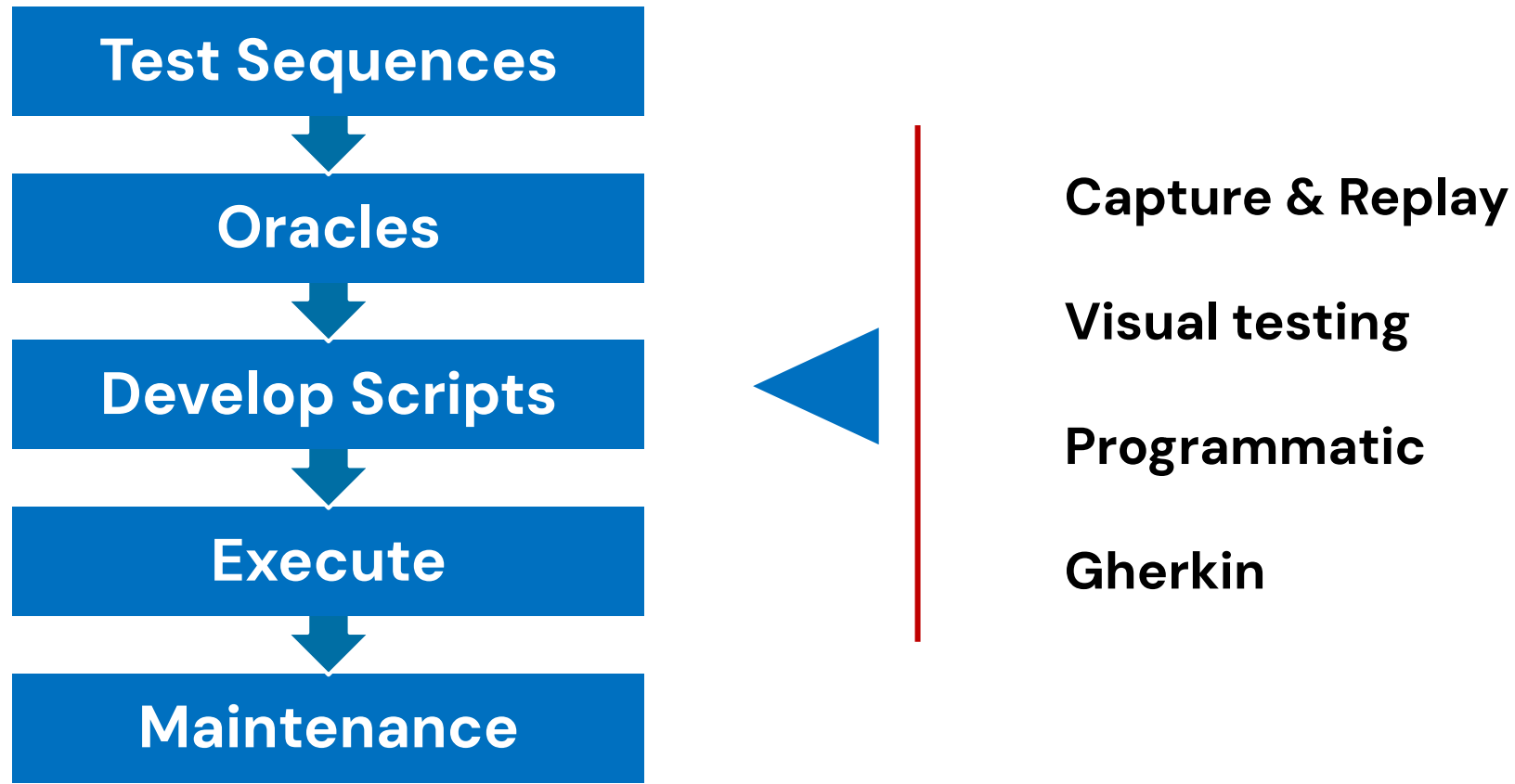
A lot of manual effort required for creating and maintaining the test scripts



Scripted tests for the important test scenarios

- 1 StartWeb "<http://www.wikipedia.org>"
- 2 Check 
- 3 Click 
- 4 Type "tiger[ENTER]"
- 5 Check 

Scripted GUI Testing



Scripted GUI testing

- Selenium example

The screenshot displays the Selenium IDE interface for a project named 'Testen Iris'. The interface is divided into several sections:

- Project:** Testen Iris
- Tests:** A list of test cases including 'login', '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' (highlighted), 'Nieuwe verkoop', and 'Nieuwe verlofaanvraag'.
- Command Table:** A table showing the sequence of commands for the selected test case.
- Command Details:** A section for editing the selected command (59: click).
- Log:** A section showing the execution log of the test suite.

	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 Details (59: click):

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

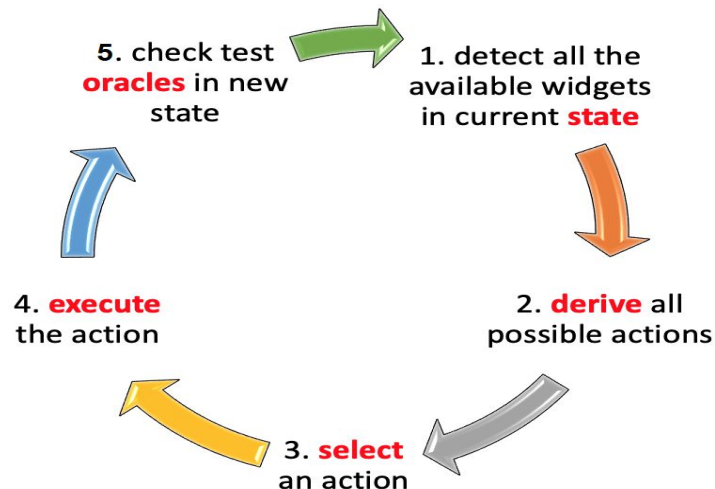
Log:

Log	Reference
54. pause on 300 OK	09:27:15
55. select on xpath=//div[@id='input_container']/div/span/select with value label=Suksesvol 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	

Scriptless GUI testing – automated GUI exploration

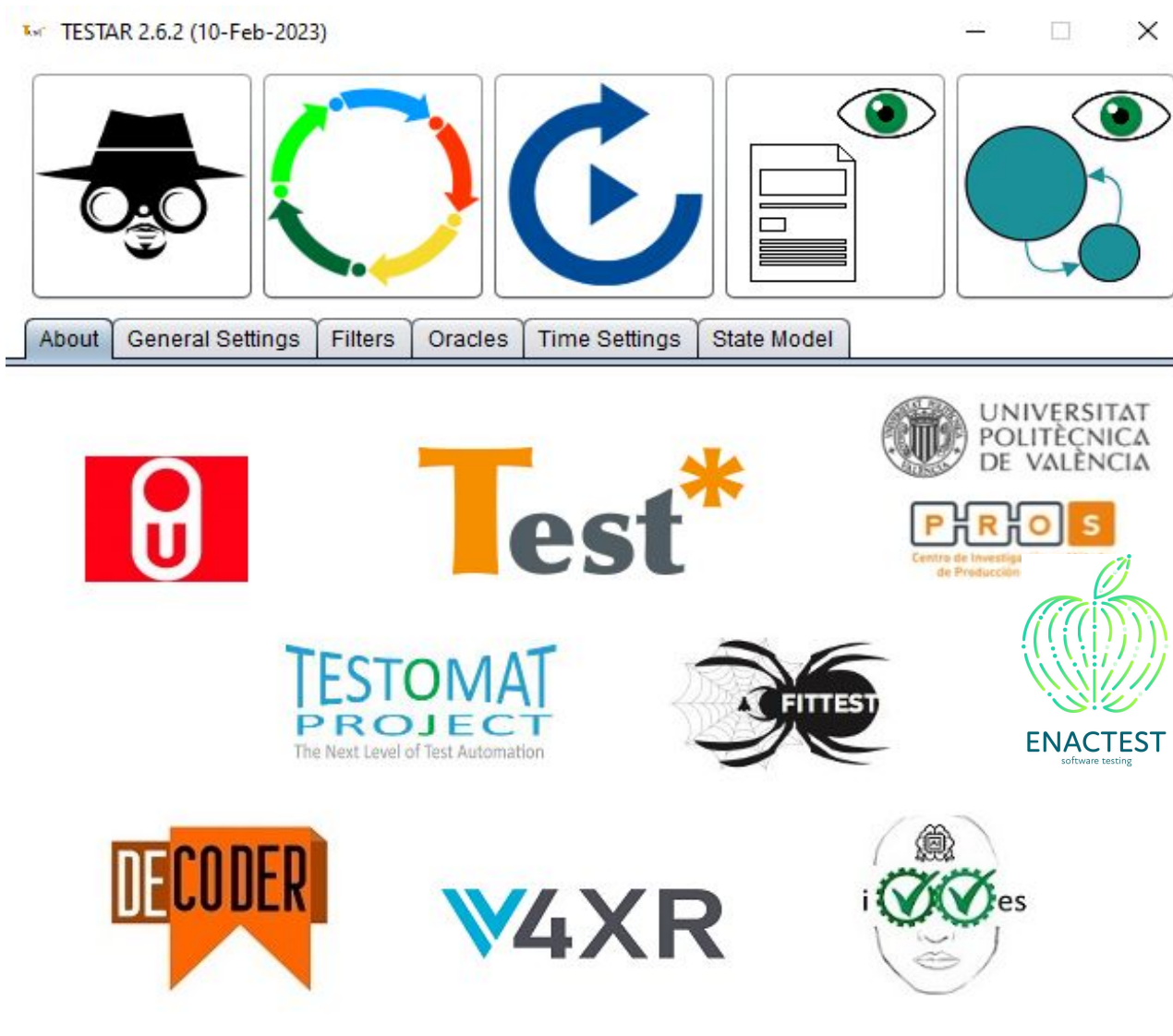
- Online / On-the-fly test generation (no predefined test cases)
 - Usually based on some level of randomness
 - Test oracles:
 - "Free": crashes, unresponsiveness, etc
 - Programmable: "if text Error found..."

Usually some manual effort required for instructing SUT-specific details for the scriptless testing tool



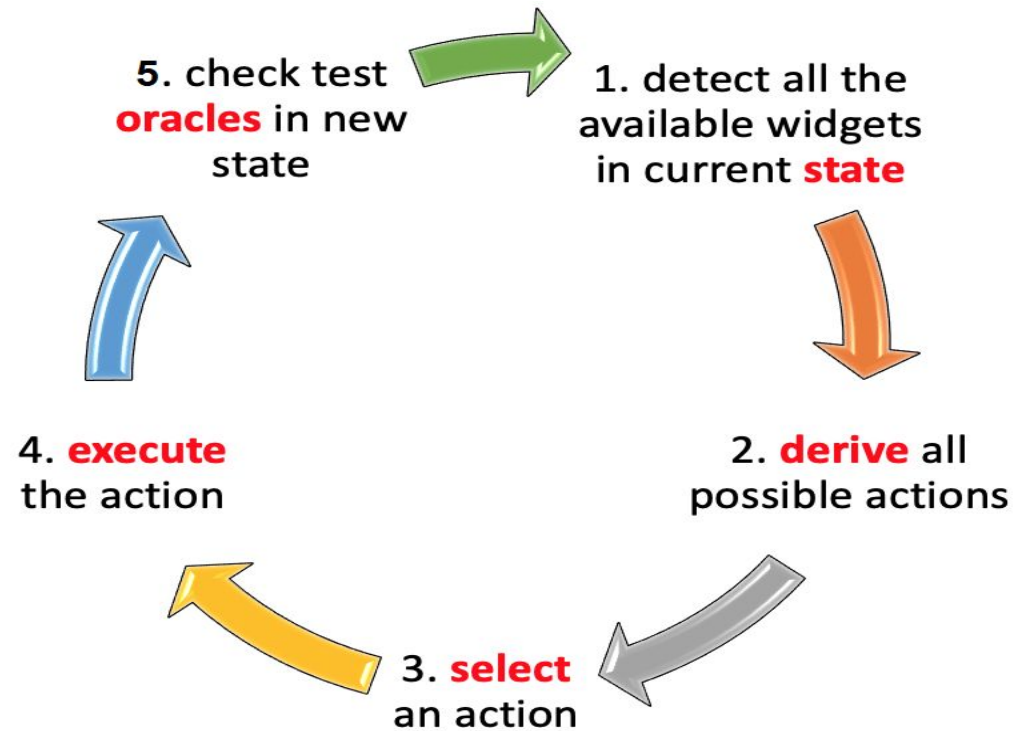
Scriptless nightly testing
for robustness and
coverage

Scriptless GUI testing – TESTAR tool



- Open source tool for scriptless testing through GUI
 - <https://testar.org/>
 - https://github.com/TESTARtool/TESTAR_dev/
- Test steps generated during test execution based on available actions

How does TESTAR work?



How does TESTAR work?

1. Detect all the available widgets in current state



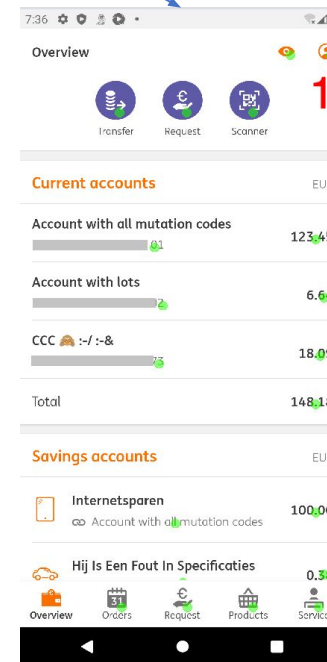
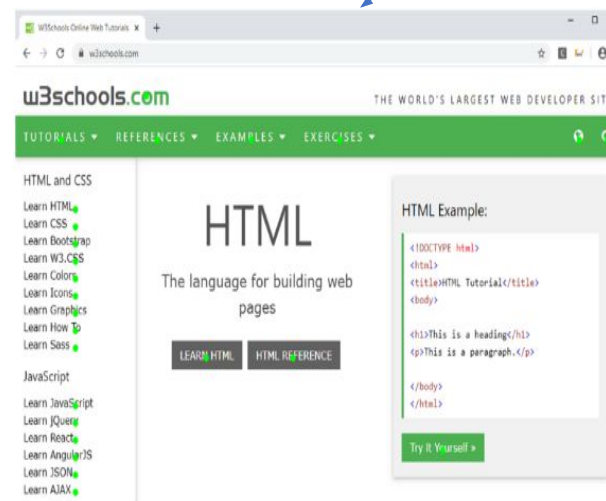
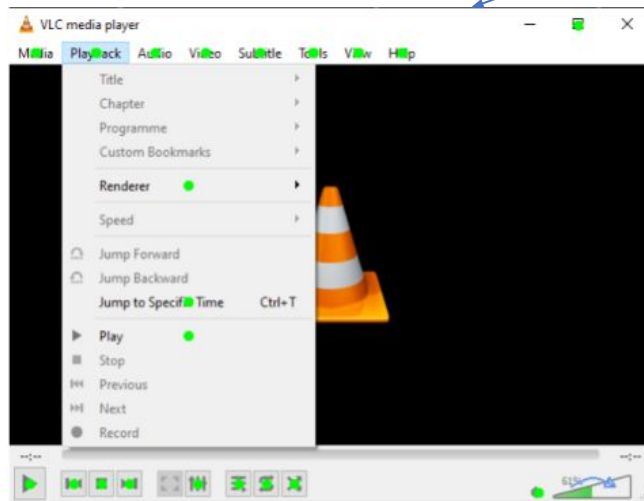
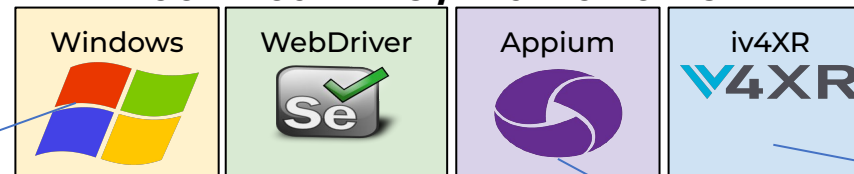
Technical APIs / Frameworks



How does TESTAR work?

1. Detect all the available widgets in current state

Technical APIs / Frameworks



How does TESTAR work?

2. Derive all possible actions (but filter undesired actions)

The screenshot shows the PARA BANK website interface. At the top, the logo and tagline 'Experience the difference' are visible. Below the header, there's a navigation menu on the left with links like 'Solutions', 'About Us', 'Services', 'Products', 'Locations', and 'Admin Page'. The main content area is titled 'Welcome to Account Services' and features a 'Transfer Funds' form. The form includes fields for 'Amount' (with a value of \$sFlvZCtQR1), 'From account #' (12345), and 'to account #' (12345). A 'TRANSFER' button is at the bottom of the form. The footer contains links for 'Home', 'About Us', 'Services', 'Products', 'Locations', 'Forum', 'Site Map', and 'Contact Us'.

- Click "About Us"
- Click "Transfer"
- etc...

- Type text in "amount"
- Type text in "name"
- etc...

- Filter "Log Out"
- Filter "Forum"
- etc...

How does TESTAR work?

2. Derive all possible actions (but filter undesired actions)



Clickable

- <button> buttons
- <a> hyperlinks
- <input type="submit">

Typeable

- <input type="text">
- <textarea>

Filtered

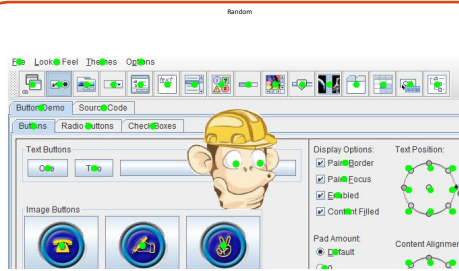
- Regex .*[IL]og out.*
- Outside web domain

How does TESTAR work?

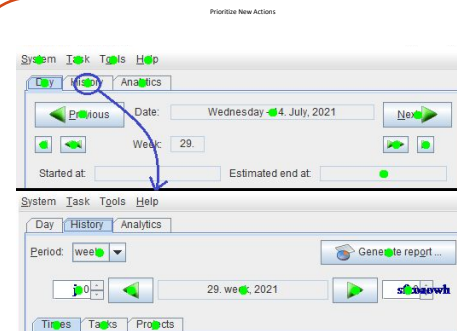
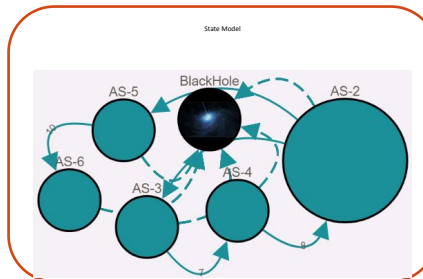
3. Select Action + 4. Execute selected Action

Test*

Integrate Action Selection Mechanism (ASM)



$$Q(s, a^*) \leftarrow R(s, a^*) + \gamma \cdot \max_{a \in A_s} Q(s', a)$$
$$R(s, a, s') := \begin{cases} R_{max} & \text{if } x_a = 0 \\ \frac{1}{x_a} & \text{otherwise} \end{cases}$$



Random

Prioritization

Q-Learning

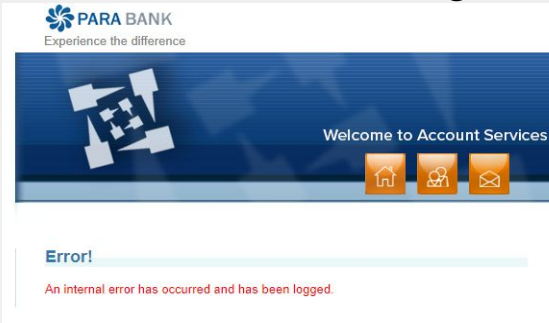
SARSA

State model

How does TESTAR work?

5. Check test **oracles** in the new state

Integrate Test Oracles




```
<!-- <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-t
<html xmlns="http://www.w3.org/1999/xhtml">
<!-->
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Parabank | Error!</title>
<link href="/parabank/template.css" rel="stylesheet"
type="text/css" />
<link href="/parabank/style.css" rel="stylesheet"
type="text/css" />
<script src="/parabank/webjars/angularjs/1.6.9/angular.min.js"></script>
</head>
<body>
<div id="mainPanel">
```

```
#####
Fecha: 24/02/2020 20:04
CODIGO ERROR: 1
Clase: CN_UsuarioAplicacionService_Impl
M???todo: getRegistro
Excepci???n:
java.lang.NullPointerException
#####

24/02/2020 20:11:19 ERROR es.arista.mvc.control.TramTipocampoAction -
#####
Fecha: 24/02/2020 20:11
CODIGO ERROR: 8
Clase: TramTipocampoAction
M???todo: guardafiltro
Excepci???n:
null
#####
```

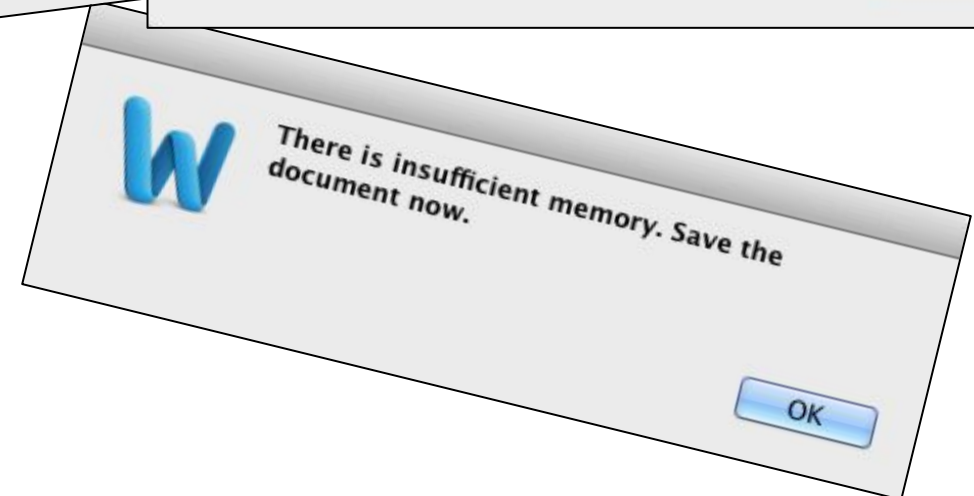
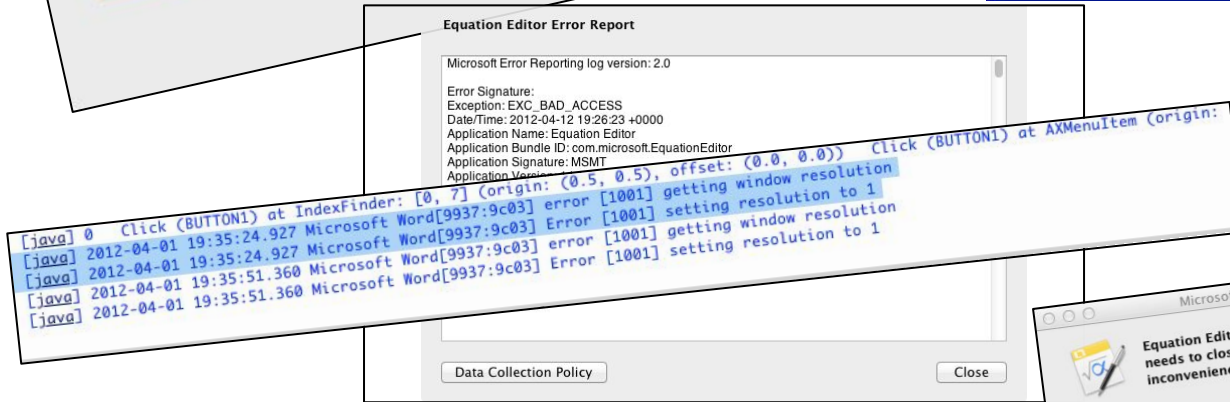
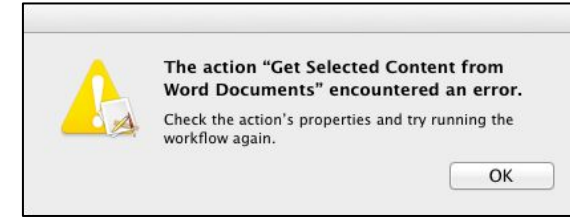
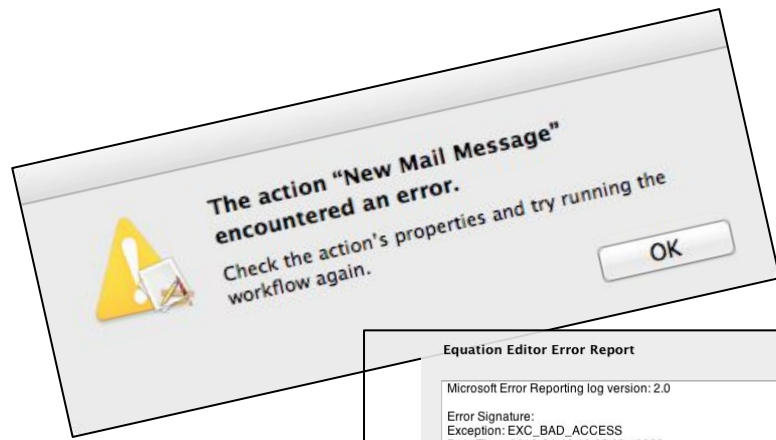
RelVerzoek

- RelatieHierarchie
- RelatieHierarchie
- RelatieOfPersoon
- RelatieOfPersoon
- Relatieomvang
- Relatieomvang
- RelatieSoort
- RelatieSoort
- RelatieStatus
- RelatieStatus
- RelatieStatusAanleiding
- RelatieStatusAanleiding





- Oracle = Mechanism to determine whether we found a failure
- In TESTAR: Oracles give verdicts about each state of the GUI
- Per default: general system failures:
 - Crashes
 - Hangs
 - suspicious values on the GUI
 - suspicious output on the standard output

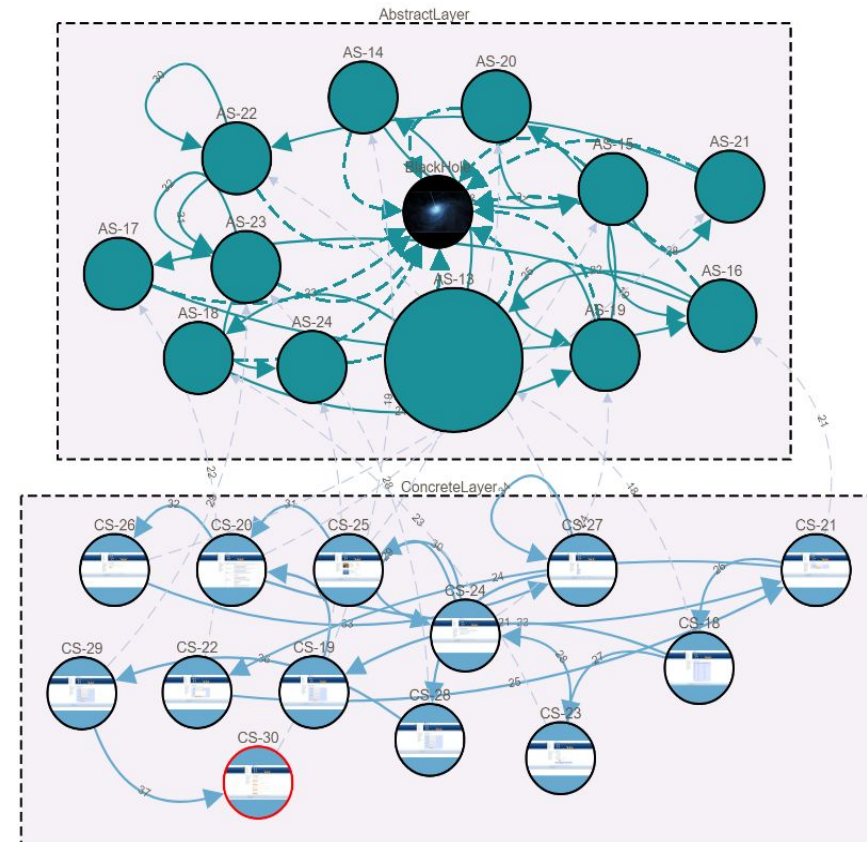


- Oracles for suspicious values and outputs
- Specify them with a regular expression
`.*[eE]rror.*|.*[eE]xcepti[o?]n.*`

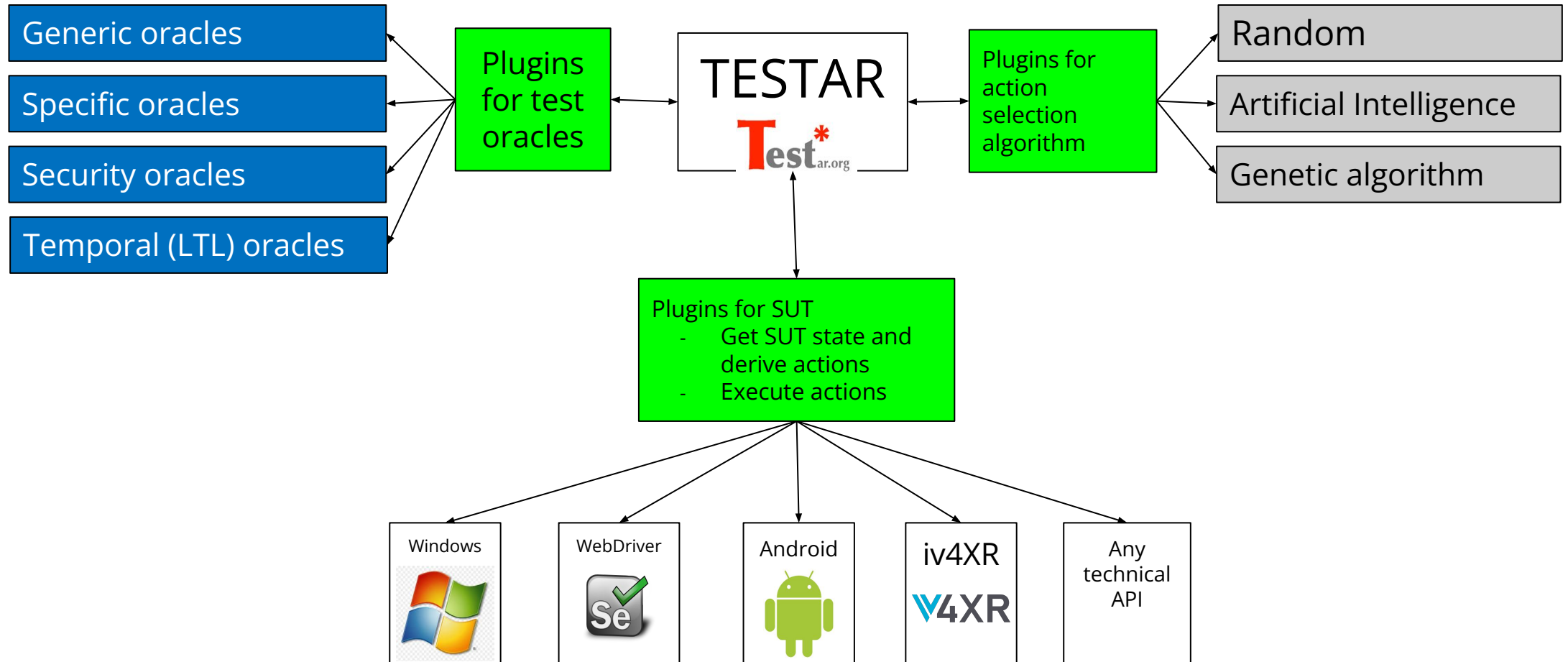
Data persistence: State Model



OrientDB: Graph-oriented database management system

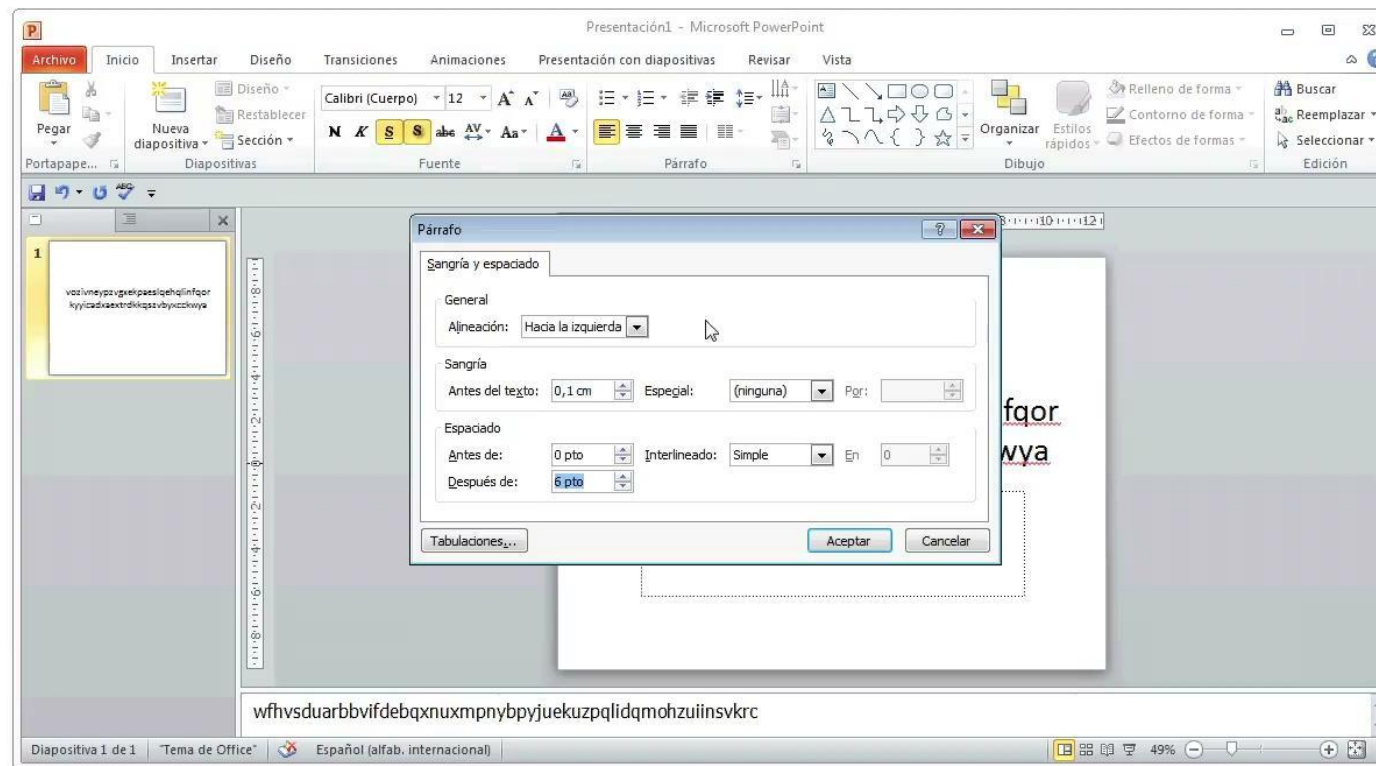


TESTAR plugin architecture



Desktop

https://drive.google.com/file/d/1Ce3zlab66f1NYCMKZnV7_-mhb_dgJkCdt/view?usp=sharing



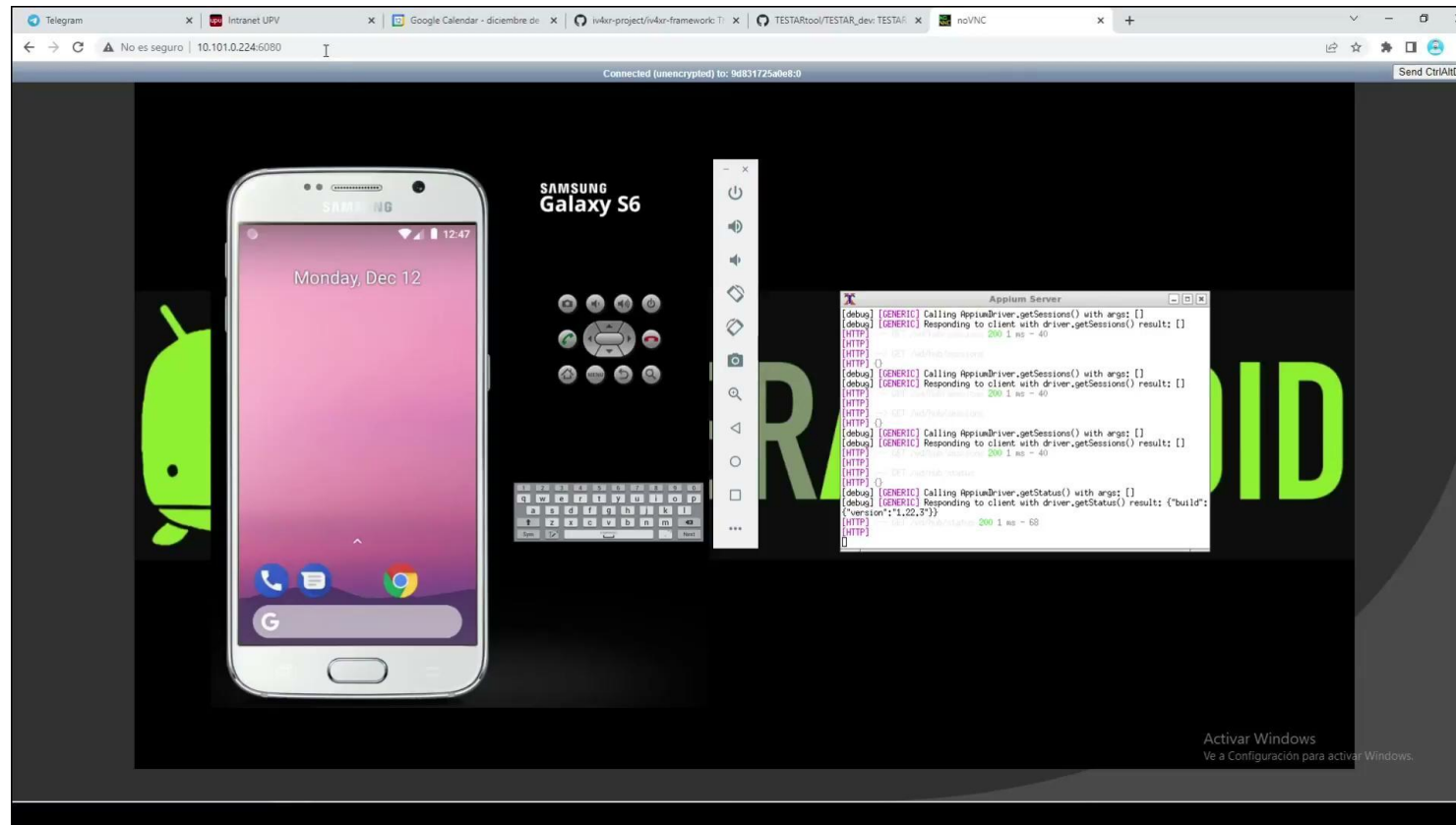
Web application

<http://www.youtube.com/watch?v=ELmjUjbN9GA>



Android app

<https://drive.google.com/file/d/1hJer6myChzSx3e4FlBv2-NFTRqvRpwN4/view?usp=sharing>



eXtended Reality game example

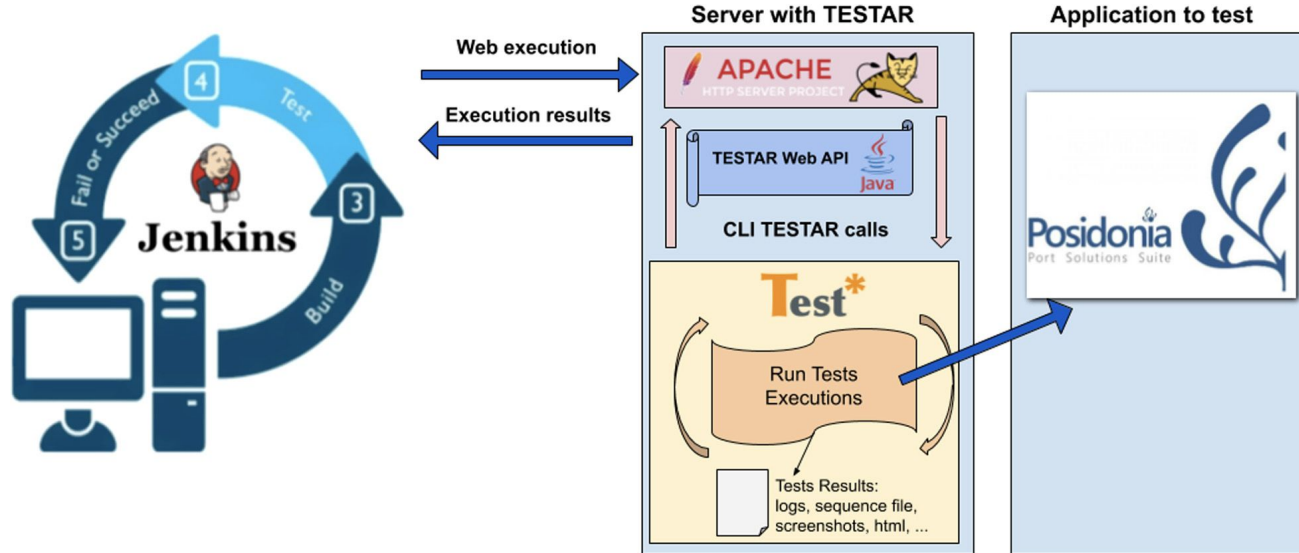
<https://drive.google.com/file/d/1EBYjHKe3SbHqrc2UguFIyYfAXkY823hk/view?usp=sharing>





Research performed with master students

Scriptless testing at CI Pipeline



Deploying TESTAR to Enable Remote Testing in an Industrial CI Pipeline: A Case-Based Evaluation

Fernando Pastor Ricós^{1(✉)}, Pekka Aho^{2(✉)}, Tanja Vos^{1,2(✉)},
Ismael Torres Boigues^{1,2,3(✉)}, Ernesto Calás Blasco^{1,2,3},
and Héctor Martínez Martínez³

¹ Universitat Politècnica de València, 46002 Valencia, Spain
ferpasri@inf.upv.es

² Open Universiteit, Heerlen, The Netherlands

{pekka.aho,tanja.vos}@ou.nl

³ Prodevelop, Valencia, Spain
{itorres,info}@prodevelop.es

Abstract. Companies are facing constant pressure towards shorter release cycles while still maintaining a high level of quality. Agile development, continuous integration and testing are commonly used quality assurance techniques applied in industry. Increasing the level of test automation is a key ingredient to address the short release cycles. Testing at the graphical user interface (GUI) level is challenging to automate, and therefore many companies still do this manually. To help find solutions for better GUI test automation, academics are researching scriptless GUI testing to complement the script-based approach. In order to better match industrial problems with academic results, more academia-industry collaborations for case-based evaluations are needed. This paper describes such an initiative to improve, transfer and integrate an academic scriptless GUI testing tool TESTAR into the CI pipeline of a Spanish company Prodevelop. The paper describes the steps taken, the outcome, the challenges, and some lessons learned for successful industry-academia collaboration.

Keywords: Automated testing · GUI level · TESTAR · CI · Technology transfer

1 Introduction

The development of cost-effective and high-quality software systems is getting more and more challenging for SMEs. Modern systems are distributed and become larger and more complex, as they connect multitude of components that interact in many different ways and have constantly changing and different types of requirements. Adequately testing these systems cannot be faced alone with traditional testing approaches.

© Springer Nature Switzerland AG 2020
T. Margaria and B. Steffen (Eds.): ISoLA 2020, LNCS 12476, pp. 543–557, 2020.
https://doi.org/10.1007/978-3-030-61362-4_31



ISOLA 2020

Scriptless testing at Cap Gemini/ProRail

- Web Application: Java (JEE6) with 12,263 LOC
- System to manage the assignment of train platforms
- Existing testing strategy: manual testing with 100 test cases.
- After a change in the system, the 100 test cases are running again as regression testing.

Scriptless Testing at the GUI Level in an Industrial Setting

Hatim Chahim¹, Mehmet Duran², Tanja E. J. Vos^{1,4(BS)}, Pekka Aho³, and Nelly Condori Fernandez²

¹ ProRail, Utrecht, The Netherlands

² Capgemini, Utrecht, The Netherlands

³ Open Universiteit, Heerlen, The Netherlands

⁴ Universidad Politécnica de Valencia, Valencia, Spain
t.vos@dsic.upv.es

⁵ University of A Coruña/Vrije Universiteit, Amsterdam, The Netherlands

Abstract. TESTAR is a traversal-based and scriptless tool for test automation at the Graphical User Interface (GUI) level. It is different from existing test approaches because no test cases need to be defined before testing. Instead, the tests are generated during the execution, on-the-fly. This paper presents an empirical case study in a realistic industrial context where we compare TESTAR to a manual test approach of a web-based application in the rail sector. Both qualitative and quantitative research methods are used to investigate learnability, effectiveness, efficiency, and satisfaction. The results show that TESTAR was able to detect more faults and higher functional test coverage than the used manual test approach. As far as efficiency is concerned, the preparation time of both test approaches is identical, but TESTAR can realize test execution without the use of human resources. Finally, TESTAR turns out to be a learnable test approach. As a result of the study described in this paper, TESTAR technology was successfully transferred and the company will use both test approaches in a complementary way in the future.

Keywords: GUI test automation tools · TESTAR · Compare test approaches · Industrial case study · Technology transfer · Railway sector

1 Introduction

Testing software at the Graphical User Interface (GUI) level is an important part of realistic testing [1]. The GUI represents a central juncture to the Software Under Test (SUT) from where all the functionality is accessed. Consequently, testing at the GUI level means operating the application as a whole, i.e., the system's components are tested in conjunction. However, automated testing at the GUI level is difficult because GUIs are designed to be operated by humans. Moreover, they are inherently non-static interfaces, subject to constant change caused by functionality updates, usability enhancements, changing requirements or altered contexts. This makes it very hard to develop and maintain automated test scripts. And today, most companies still resort to time-consuming

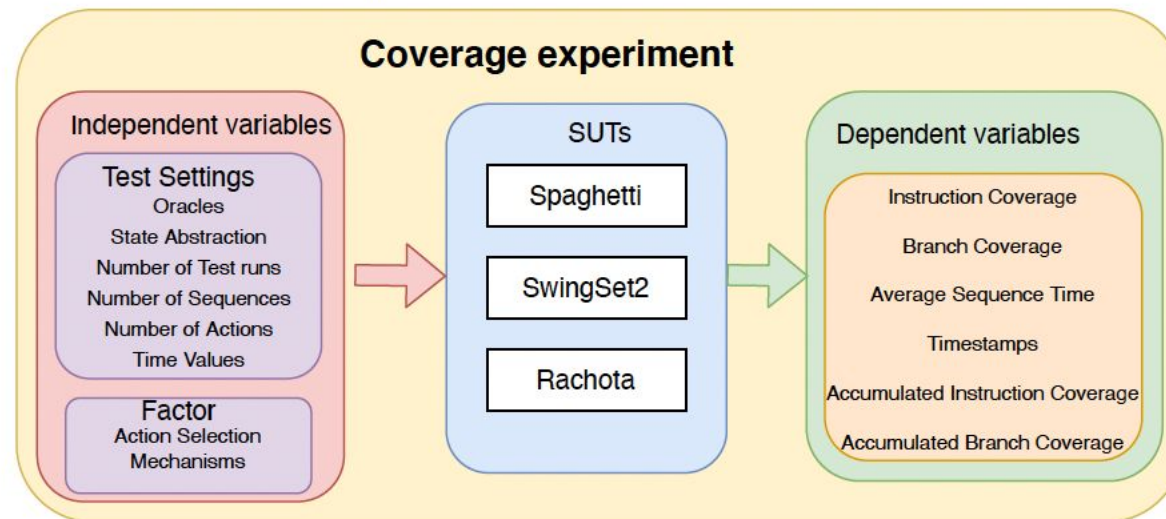


RCIS 2020

	TESTAR	Manual
Preparation	44 hours	43 hours
Testing	(51 hours)	6 hours
Post testing	5 hours	2 hours
Critical failures	4	0
Functional coverage	80%	73%

Evaluation of TESTAR's effectiveness

- Surrogate measure: coverage



Evaluating TESTAR's effectiveness through code coverage

Aaron van der Brugge¹, Fernando Pastor Ricós², Pekka Aho¹, Beatriz Marín², and Tanja E.J. Vos^{1,2}

¹ Open Universiteit, The Netherlands

² Universitat Politècnica de València, Spain

Abstract. Testing is of paramount importance in assuring the quality of software products. Nevertheless, it is not easy to judge which techniques or tools are the most effective. A commonly used surrogate metric to evaluate the effectiveness of testing tools is code coverage, which has been widely used for unit and integration testing. However, for GUI testing approaches, this metric has not been sufficiently investigated. To fill this gap, we run experiments with the TESTAR tool, a scriptless testing tool that automatically generates test cases at the Graphical User Interface (GUI) level. In the experiment, we analyze and compare the obtained code coverage when using four different action selection mechanisms (ASMs) in TESTAR that are used to test three SUTs.

Keywords: Effectiveness · Code Coverage · Experiment · GUI Testing.



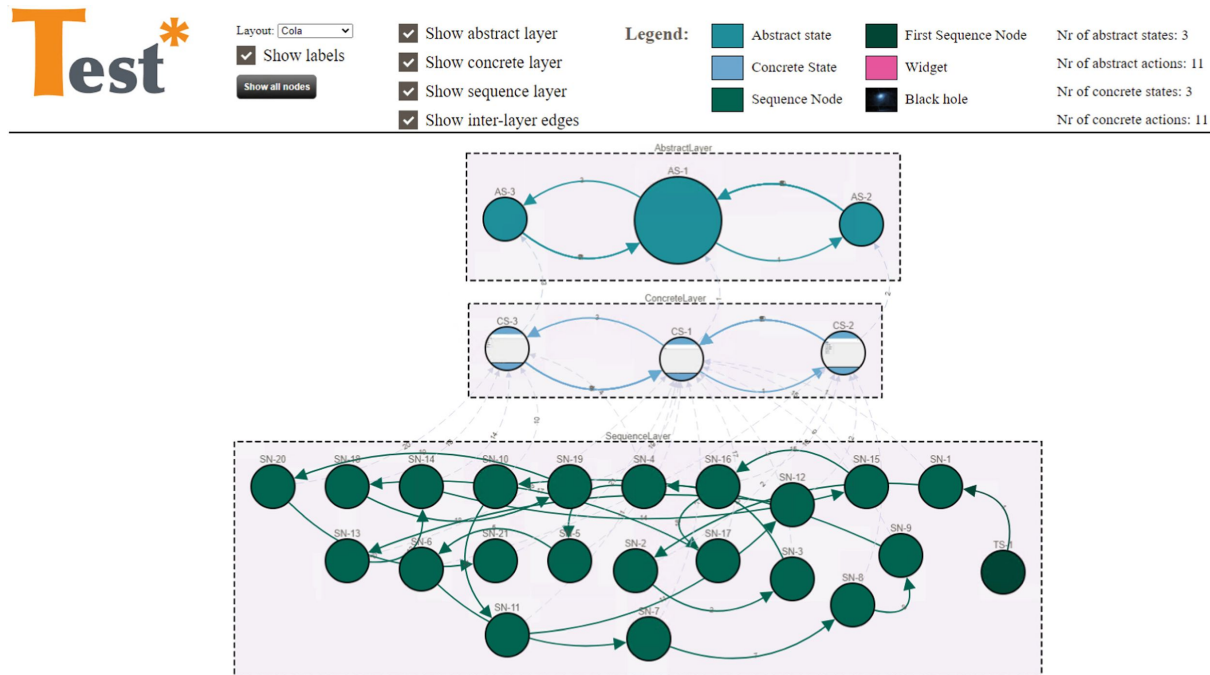
1 Introduction

Nowadays there are a lot of methods with different activities, roles and artifacts to develop software. In all of them, the testing phase is of paramount importance for quality assurance. The best way to evaluate the quality of testing would be to use the percentage of failures that were found executing the tests cases. However, it is not possible to know in advance how many bugs exist in the software, and if we don't find any, this doesn't mean that they don't exist. Consequently, in order to analyse the effectiveness of testing, surrogate measures [8] are used. A commonly used surrogate measure is code coverage [2]. Code coverage has been widely used for measuring the quality of unit and integration testing, however for system testing approaches that test at the GUI level, this metric have been less investigated.

We want to fill this gap by evaluating the coverage of a GUI testing tool in an experiment. We select the TESTAR tool [25] since it is an open-source tool that allows to automatically generate the test cases from the GUI. To do that, TESTAR implements a scriptless approach, meaning that the test cases do not have to be defined prior to test execution. Instead, each test step is generated during the test execution, based on (1) the actions that are available in that specific time and state of the GUI, and (2) the action selection mechanism (ASM). Selecting and executing the most suitable actions can both improve the likelihood and

JISBD 2021

State model inference



State Model Inference Through the GUI Using Run-Time Test Generation

Ad Mulders¹, Olivia Rodriguez Valdes¹, Fernando Pastor Ricós², Pekka Aho¹ (✉), Beatriz Marín², and Tanja E. J. Vos^{1,2}

¹ Open Universiteit, Heerlen, The Netherlands

pekk.aho@ou.nl

² Universitat Politècnica de València, Valencia, Spain

Abstract. Software testing is an important part of engineering trustworthy information systems. End-to-end testing through Graphical User Interface (GUI) can be done manually, but it is a very time consuming and costly process. There are tools to capture or manually define scripts for automating regression testing through a GUI, but the main challenge is the high maintenance cost of the scripts when the GUI changes. In addition, GUIs tend to have a large state space, so creating scripts to cover all the possible paths and defining test oracles to check all the elements of all the states would be an enormous effort. This paper presents an approach to automatically explore a GUI while inferring state models that are used for action selection in run-time GUI test generation, implemented as an extension to the open source TESTAR tool. As an initial validation, we experiment on the impact of using various state abstraction mechanisms on the model inference and the performance of the implemented action selection algorithm based on the inferred model. Later, we analyse the challenges and provide future research directions on model inference and scriptless GUI testing.

Keywords: Model inference · Automated GUI testing · TESTAR tool



RCIS 2022

1 Introduction

The world around us is strongly connected through software and information systems. Graphical User Interface (GUI) represents the main connection point between software components and the end users, and can be found in most modern applications. Testing through GUI is an important way to prevent end users from experiencing the effects of software bugs. Although GUI testing can be done manually, it is a very time consuming and costly process [14]. There are various tools to capture or manually define scripts for regression testing of GUIs, but the main challenge is the high maintenance cost of the scripts when the GUI changes [25]. In addition, GUIs tend to have a large state space, so creating scripts to cover all the possible paths and defining test oracles to check all the elements of all the states would be an enormous effort.

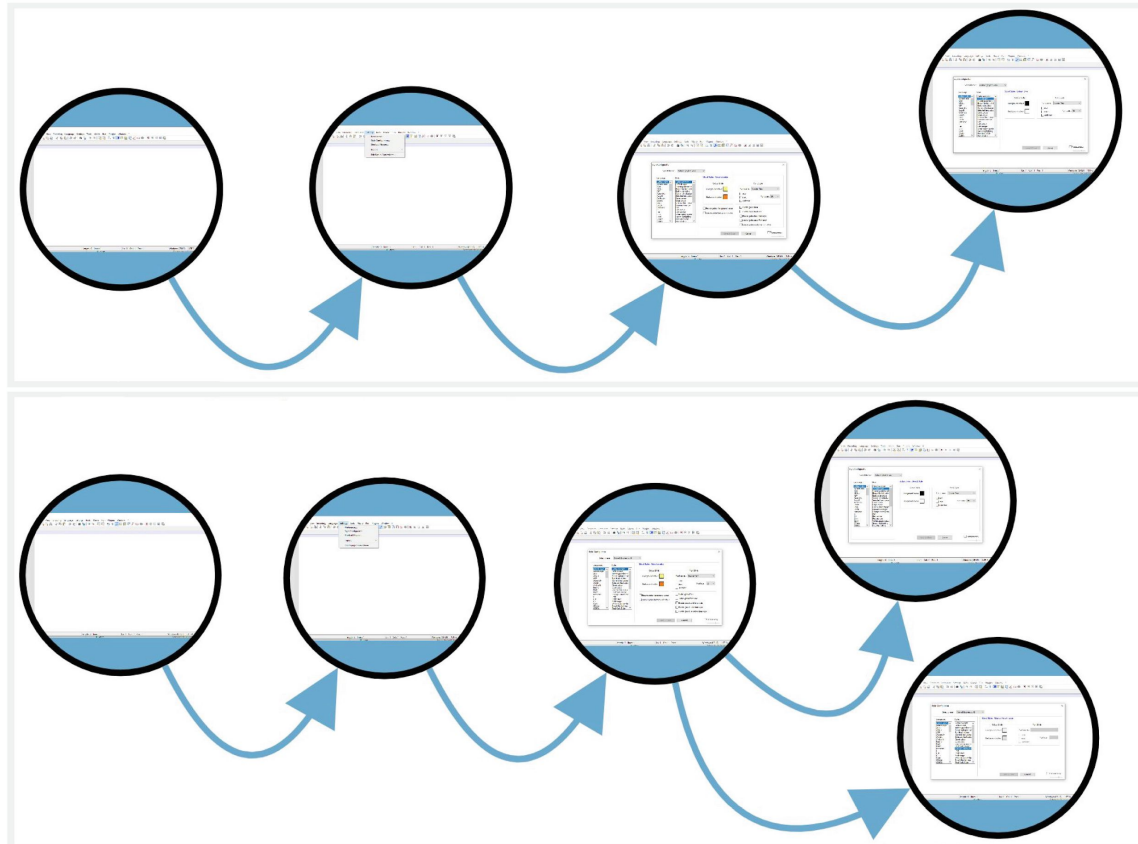
© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
R. Guizzardi et al. (Eds.): RCIS 2022, LNBP 446, pp. 546–563, 2022.
https://doi.org/10.1007/978-3-031-05760-1_32

The figure illustrates the process of identifying a UI element for accessibility testing. On the left, the 'Money Transfer' app's 'Overview' screen is shown, displaying a balance of 148.18 EUR, current accounts, savings accounts, and a bottom navigation bar. A red box highlights the 'Overview' screen. On the right, the Android Studio layout view is shown, displaying the XML hierarchy for the 'button1' element, which is an 'android.widget.Button' with text 'OK'. A red box highlights the layout view. The 'button1' element is identified as the 'OK' button in the 'Current accounts' section of the app.



QRS 2022

Using GUI change detection for Delta testing



Using GUI Change Detection for Delta Testing

Fernando Pastor Ricós¹✉, Rick Neef², Beatriz Marín¹,
Tanja E. J. Vos^{1,2}, and Pekka Aho²

¹ Universitat Politècnica de València, Valencia, Spain
fpastor@pros.upv.es

² Open Universiteit, Heerlen, The Netherlands

Abstract. Current software development processes in the industry are designed to respond to rapid modification or changes in software features. Delta testing is a technique used to check that the identified changes are deliberate and neither compromise existing functionality nor result in introducing new defects. This paper proposes a technique for delta testing at the Graphical User Interface (GUI) level. We employ scriptless testing and state-model inference to automatically detect and visualize GUI changes between different versions of the same application. Our proposed offline change detection algorithm compares two existing GUI state models to detect changes. We present a proof of concept experiment with the open-source application Notepad++, which allows automatic inference and highlights GUI changes. The results show that our technique is a valuable amplification of scriptless testing tools for delta testing.

Keywords: Delta testing · Scriptless testing · State-model inference · GUI change detection

1 Introduction

Agile methods [1, 17] have evolved into an efficient software development process that emphasizes collaboration between Development and Operations (DevOps) teams and leverages Continuous Integration (CI) tools to automate the build and test stages [6]. This shift towards rapid-release deployment leads to many software versions and needs a comprehensive delta testing approach to maintain the high quality of the software product.

Software updates and new versions often involve multiple changes to enhance features or fix issues. Delta testing checks that the changes are intentional and do not compromise existing functionality or result in introducing new defects. In this paper, we concentrate on delta testing at the software's Graphical User Interface (GUI), the primary end-users entry point to interact with applications.

Existing tools perform change detection at the code level [15], compare GUI changes to repair test scripts [4], or are restricted to comparing data differences between individual documents [11]. However, none of these tools intend to highlight how GUI transitions evolve. Therefore, we aim to develop a novel tool that

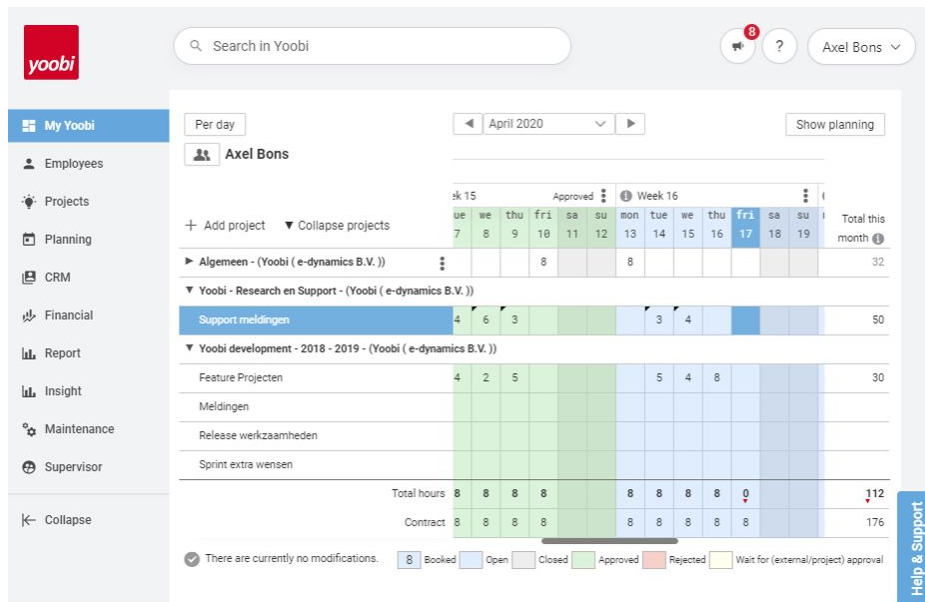
© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
S. Nurcan et al. (Eds.): RCIS 2023, LNBP 476, pp. 509–517, 2023.
https://doi.org/10.1007/978-3-031-33080-3_32



RCIS 2023

Adoption of Scriptless and Scripted testing at Yoobi

Unique bugs found in the first 22 days: Selenium = 4 Testar = 9 Manual =



	Injected	Selenium	Testar	Known	Prio
1	x		Console	x	-
2	x	Task	Class	x	-
3	x		Text	x	-
4		Project		x	high
5			Console		high
6			Text		low
7			Title		low
8		Expense			high
9		Task		x	high
10			Text		low
11			Console		high
12			Text		low

Scripted testing is better to be used for important scenarios.
Scriptless testing is better to be used to fully test the SUT.
They detect different types of failures -> they are complementary techniques

Information and Software Technology 156 (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 viable 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

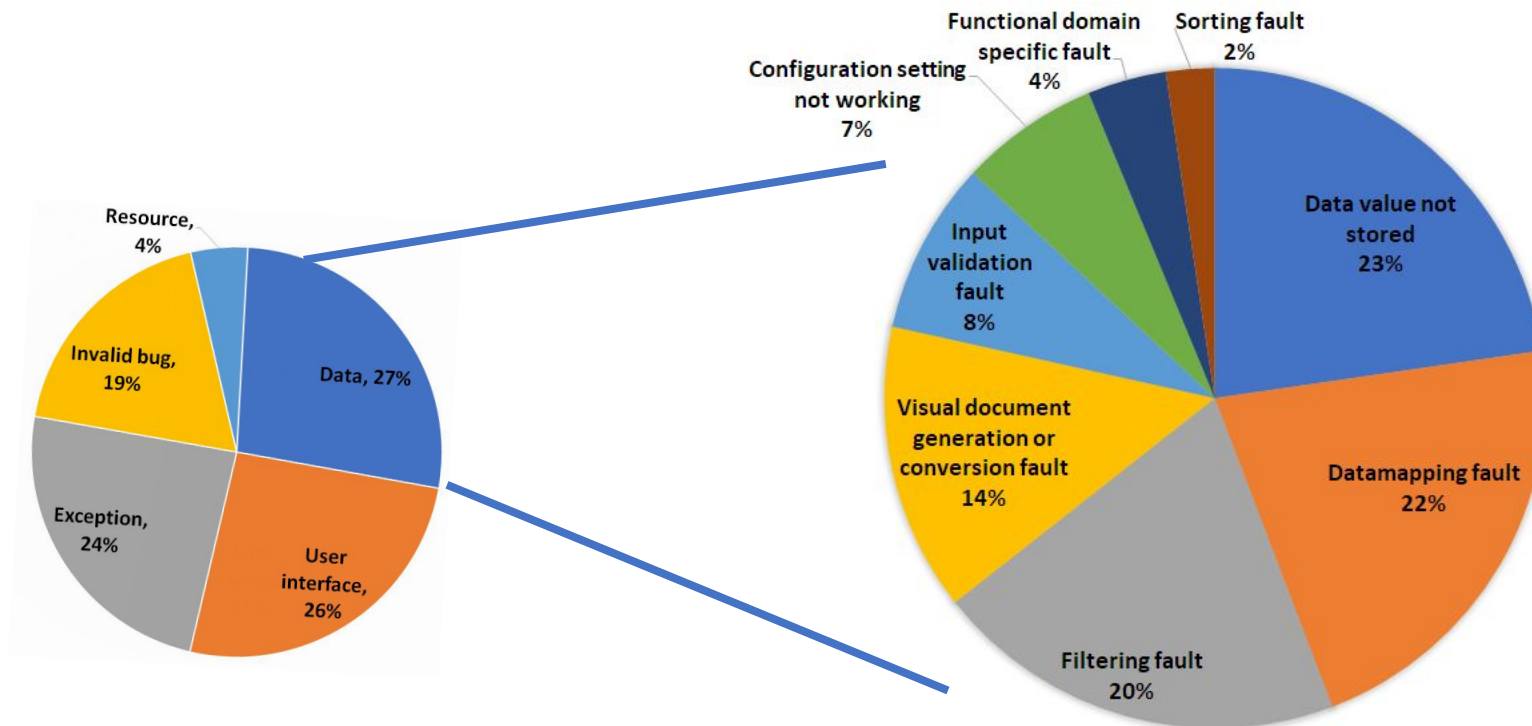
* Corresponding author at: Universitat Politècnica de València, Spain.
<https://doi.org/10.1016/j.infsof.2023.107172>
Received 16 August 2022; Received in revised form 4 February 2023; Accepted 8 February 2023
Available online 13 February 2023
0950-5849/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY 4.0 license.



IST 2023

Empirical Bug Taxonomy

- Database with 12100 historical bugs at DigiOffice
- Analysis of 2500 bugs



RESEARCH CASE STUDY IMPROVE BUG
FINDING ABILITY OF TESTAR BY USING
HISTORICAL BUG DATABASE OF DIGIOFFICE]

by

Robin R. Bouwmeester BSc

Student number: 852110566
Course code: IM9703
Date: September 2nd, 2023
Thesis committee: Prof. Dr. Tanja E.J. Vos, Open University
Dr. Beatriz Marín, Universitat Politècnica de València
Fernando P. Ricós, Universitat Politècnica de València





HandsON



SCAN ME



<https://testar.org/rcis2023/>

HandsON



Open Universiteit
www.uu.nl



VRAIN
Valencian Research Institute
for Artificial Intelligence



Automated Testing at the GUI level
Hands-on do it yourself manual



2023

The TESTAR team

Last updated: 23/05/2023 at 19:09.

HandsON



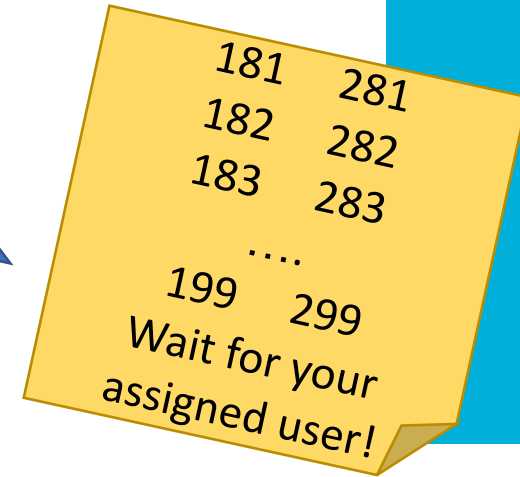
APACHE GUACAMOLE

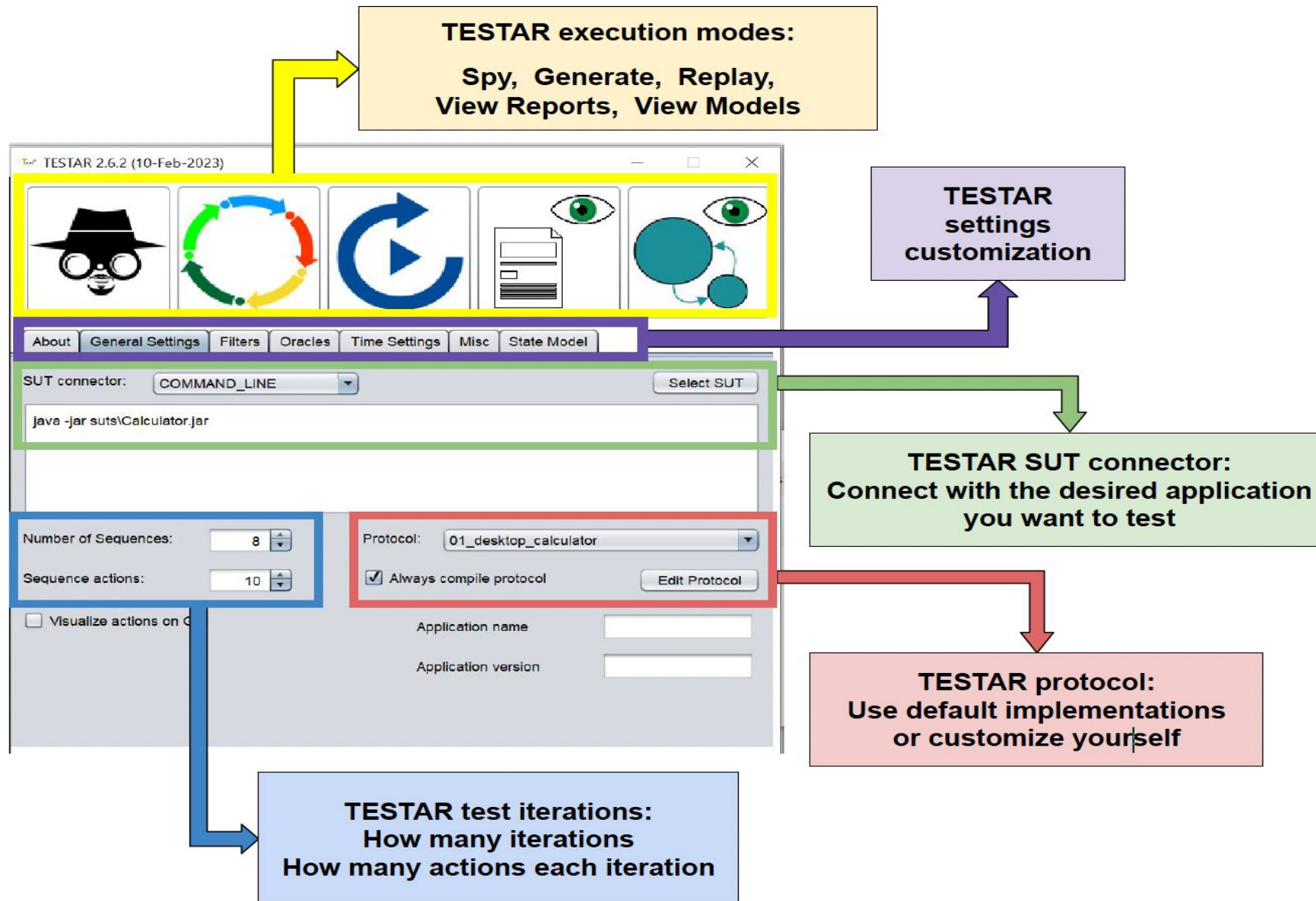
handson

.....

Login

User: handson181
Password: norway2109





`.*Backspace.* | .*CE.* | .*View.*`

contains Backspace OR contains CE OR contains View

Exercise (hands-on 15):

Add filter to exclude the action derivation of the “Close button” (X on top right)

Exercise (hands-on 16):

Try yourself to filter actions with the clickfilter

Exercise (hands on 18):

1. Add oracles using a regular expression to detect errors containing:
Error OR Exception
2. Run Testar on the calculator to check whether your new oracles detect the bugs.

Exercise (hands on 20):

1. Add oracles using a regular expression to detect errors from the process output:

Error OR Exception

2. Run Testar on the calculator to check whether your new oracles detect the bugs.

Congrats! You made it to the web section!

Exercise (hands on 29-30):

1. Try out the SPY mode to see what actions are detected
2. Run GENERATE mode to see how it works.

Exercise (hands-on 31):

You can copy/paste the snippet of code from the hands-on exercise ;)

1. Edit the protocol to enable login.
 - 1.1 Find the *beginSequence* method
 - 1.2 Click and type pre-defined actions for the username and the password and click on the submit button

```
waitLeftClickAndTypeIntoWidgetWithMatchingTag(  
    Tags.Title, // Tag Name  
    "username", // Tag Value to find  
    "testar", // Text to type  
    state, // State to search  
    system, // System to interact  
    5, // maxNumberOfRetries  
    1.0); // waitBetweenRetries
```

```
waitAndLeftClickWidgetWithMatchingTag(  
    WdTags.WebValue, // Tag Name  
    "Log In", // Tag Value to find  
);
```

Or... even better, you can uncomment the code in the protocol

Compound Actions

- A compound action triggers the execution of multiple click and type actions.

```
CompoundAction.Builder multiAction = new CompoundAction.Builder();  
multiAction.add(ac.clickTypeInto( nameWidget, "Triggered Name", true), 1.0);  
multiAction.add(ac.leftClickAt(anotherWidget), 1.0);  
multiAction.build();
```

Triggered Actions

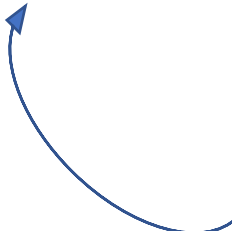
- We can trigger actions every time certain criteria is met

Update Profile

First Name:	<input type="text" value="John"/>
Last Name:	<input type="text" value="Smith"/>
Address:	<input type="text" value="Custom Street"/>
City:	<input type="text" value="Custom City"/>
State:	<input type="text" value="CA"/>
Zip Code:	<input type="text" value="90210"/>
Phone #:	<input type="text" value="310-447-4121"/>
<input type="button" value="UPDATE PROFILE"/>	

Exercise (hands-on 32):

1. Edit the method *deriveActions* in the protocol to detect the Update Profile page and trigger a compound action to fill in the form.
2. Run GENERATE mode until you see that Testar clicks on Update Profile and your triggered compound action is executed.



You can copy/paste the snippet of code from the hands-on exercise ;)

Or... even better, you can uncomment the code in the protocol

Advanced TESTAR oracles

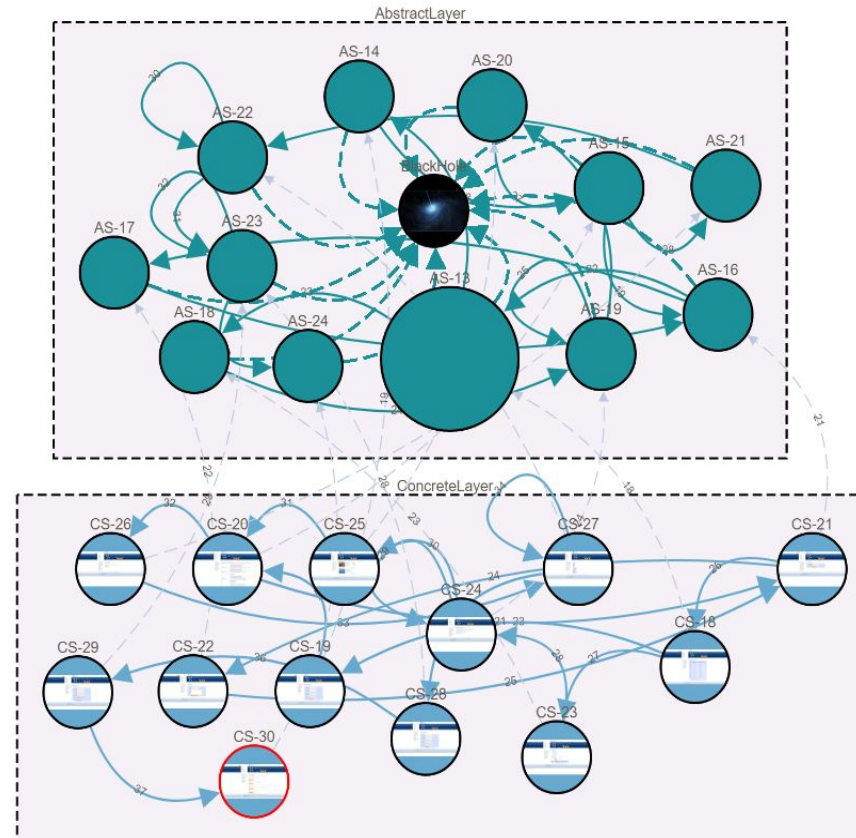
So far, TESTAR can detect if the SUT:

- Closes unexpectedly by a **crash**
- Stops responding due to a **freeze**
- Messages containing **Exception** or other keywords related to a failure
 - Widgets
 - Windows apps: output buffer of process
 - Web apps: browser console

Exercise (hands-on 31):

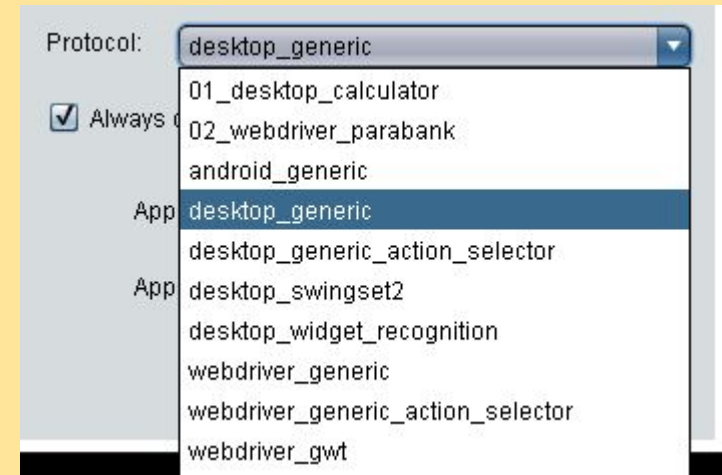
1. Edit the protocol to enable advanced oracles.
 - 1.1 Find the *getVerdict* method
 - 1.2 Uncomment the custom verdicts
 - 1.3 Save and compile the protocol
 - 1.4 Run GENERATE and check in the HTMLreports if TESTAR detects some WARNING sequences

Data persistence: State Model

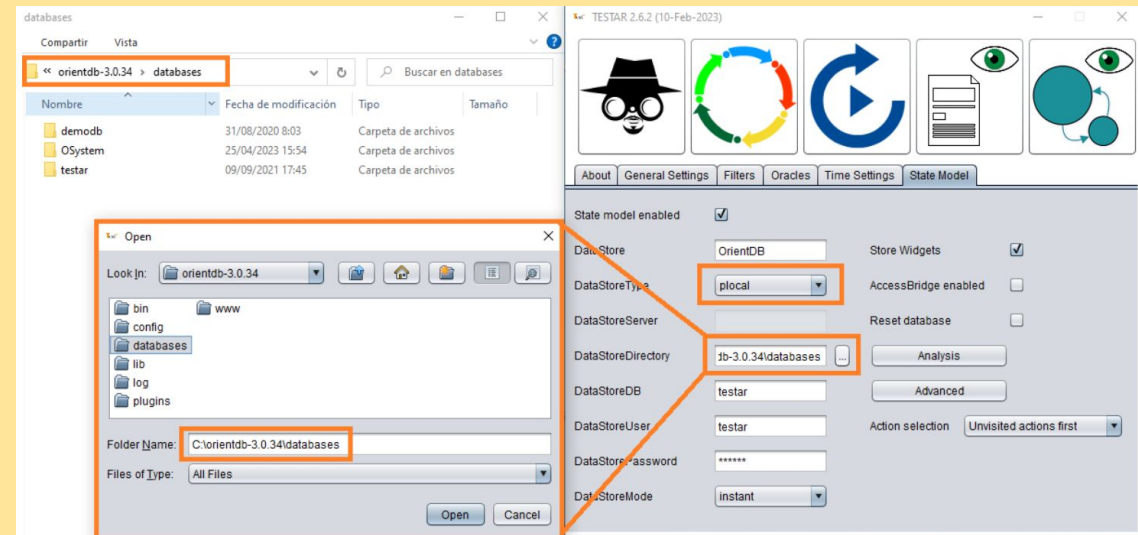


Exercise (hands-on 45):

1. Change to Desktop Generic protocol



2. Configure the state model



3. Run GENERATE with 1 sequence of 10 actions

4. Click on analysis on the State Model tab to check the generated model

Abstraction of the State Model

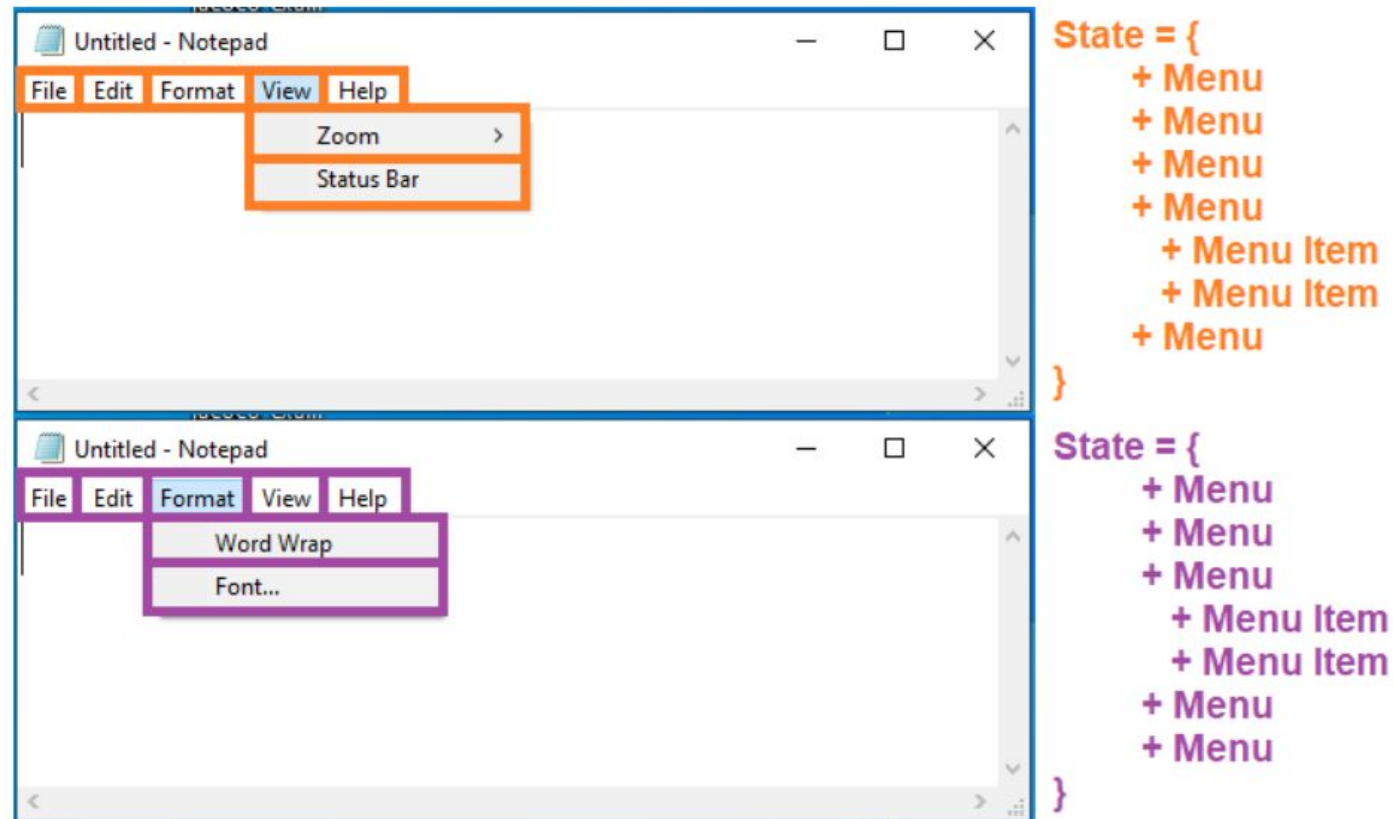


Figure 21: State Model Widget Control Type Abstraction

Abstraction of the State Model

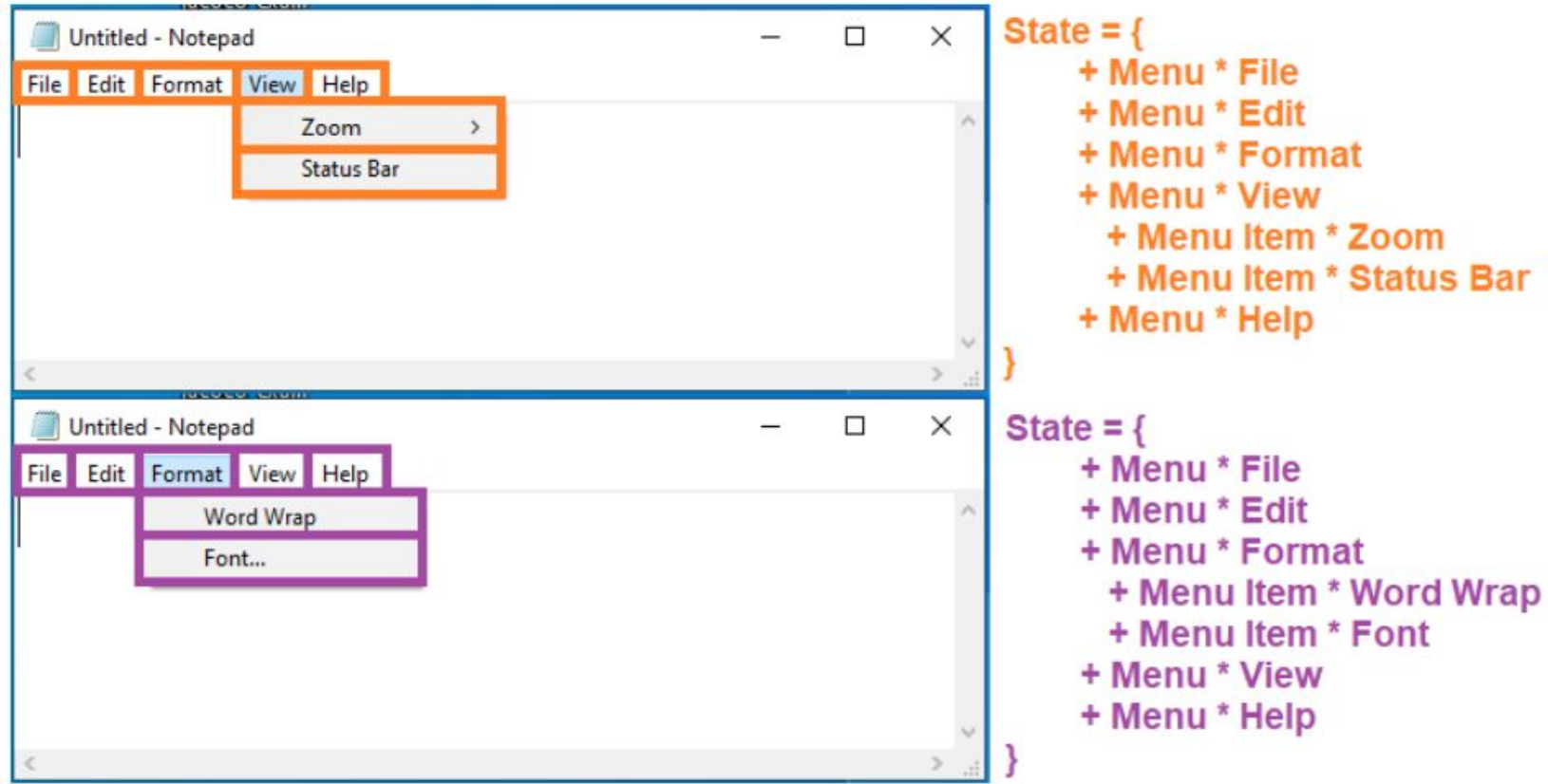


Figure 22: State Model Widget Control Type and Title Abstraction

New ideas for research

- Type of SUT: **From** desktop, web **to** mobile, XR
- Obtaining and defining the state:
 - Image based state detection
 - State abstraction and the consequences
- Action selection: **From** random **to** intelligent AI-based agents
 - Reinforcement Learning
 - Evolutionary algorithms
 - Learning a model, exploring new actions
- Experiments:
 - How do we know what we tested? Metrics?
 - How do we know we tested well?
- Oracles
 - Through model-learning, calculating the difference
- Apply to more real industrial systems
- Dockerization and distributed testing

Examples of Possible Graduation Projects



Using AI and image recognition to detect available actions and GUI state

- Challenge:
 - Various operating systems and platforms have different APIs to access GUI components (e.g. Windows UI Automation API and Selenium Web Driver)
- Possible solution:
 - Create a platform independent solution by using image recognition on the screenshots of GUI and use AI to train the widget detection, TESTAR can automatically label widget pics
 - Using feature recognition and OCR (optical character recognition) to recognize widgets, for example OpenCV library
 - Detecting widget locations by changing focus with "Tabulator" key
 - Using ML algorithms for image recognition

Examples of Possible Graduation Projects



Finding better metrics to evaluate GUI testing

- Challenge:
 - How to evaluate if GUI testing was done well?
- Possible solutions:
 - GUI mutation testing (generate bugs and see if your tests find them)
 - Test systems with known bugs (old versions)
 - Surrogate metrics, like code coverage, unique logs, GUI coverage, ...

Examples of Possible Graduation Projects



Evaluating automated change detection in DevOps environment

- Challenge:
 - How to automatically detect changes between versions of software in CI?
- Possible solutions:
 - There is a proof-of-concept implementation for automatically comparing the models of 2 consequent SUT versions to detect and report changes between versions
 - The approach for change detection should be evaluated during software development, and compared with traditional regression testing

Examples of Possible Graduation Projects



Improved test reporting

- Challenge:
 - How to show visually what TESTAR tested and a summary of the test results in a dynamic dashboard?
- Possible solutions:
 - Collecting all the results into a database (allowing the use of multiple TESTAR instances and many SUTs)
 - Generating web-based test reports that are easy to use and visualize what the user wants to see (configurable)
 - Finding the shortest path to reproduce a failure found by TESTAR to help debugging

Examples of Possible Graduation Projects



Listening mode for TESTAR – learning from manual/scripted testing

- Challenge:
 - TESTAR often requires system specific instructions to reach all parts of the GUI
 - Companies want to test happy paths, or specific use cases/flows
- Possible solution:
 - Record manual or scripted test cases into a GUI state model (in graph database)
 - Show the GUI coverage of manual/scripted tests
 - Generate new test cases to cover the missing parts
- Other benefits:
 - Automated maintenance of GUI testing scripts
 - When the GUI changes, automatically extract a new GUI state model and repair the changed parts of the scripts

Examples of Possible Graduation Projects



AI, ML, or swarm intelligence in action selection

- Challenge:
 - How to constantly improve GUI testing?
 - Hybrid action selection strategies and metrics to automatically change from one algorithm to another
- Possible solutions:
 - Use artificial intelligence, machine learning or swarm intelligence (or something else), so that TESTAR learns to optimize its testing
 - Combine different strategies and learn when to use a specific and when to change to another
 - Combine into change detection by models or code coverage vs. code commits
 - N-switch metric could be used after all states have explored

Examples of Possible Graduation Projects



State and action **abstraction** for state model inference

- Challenge:
 - TESTAR is able to automatically infer state models based on user defined set of widget properties and the rest of the properties are ignored .How to improve the state abstraction and select (with AI or ML) a suitable abstraction for a specific SUT automatically?
- Possible solutions:
 - Using image recognition and similarity metrics on the screenshots
 - Using code coverage measurement to for state abstraction (if different code is executed, the state should be different)
 - Using available actions, or ignoring specific widgets, etc
 - Dynamically changing level of abstraction
 - Learning suitable level of abstraction, based on user input on screenshots of “same state”

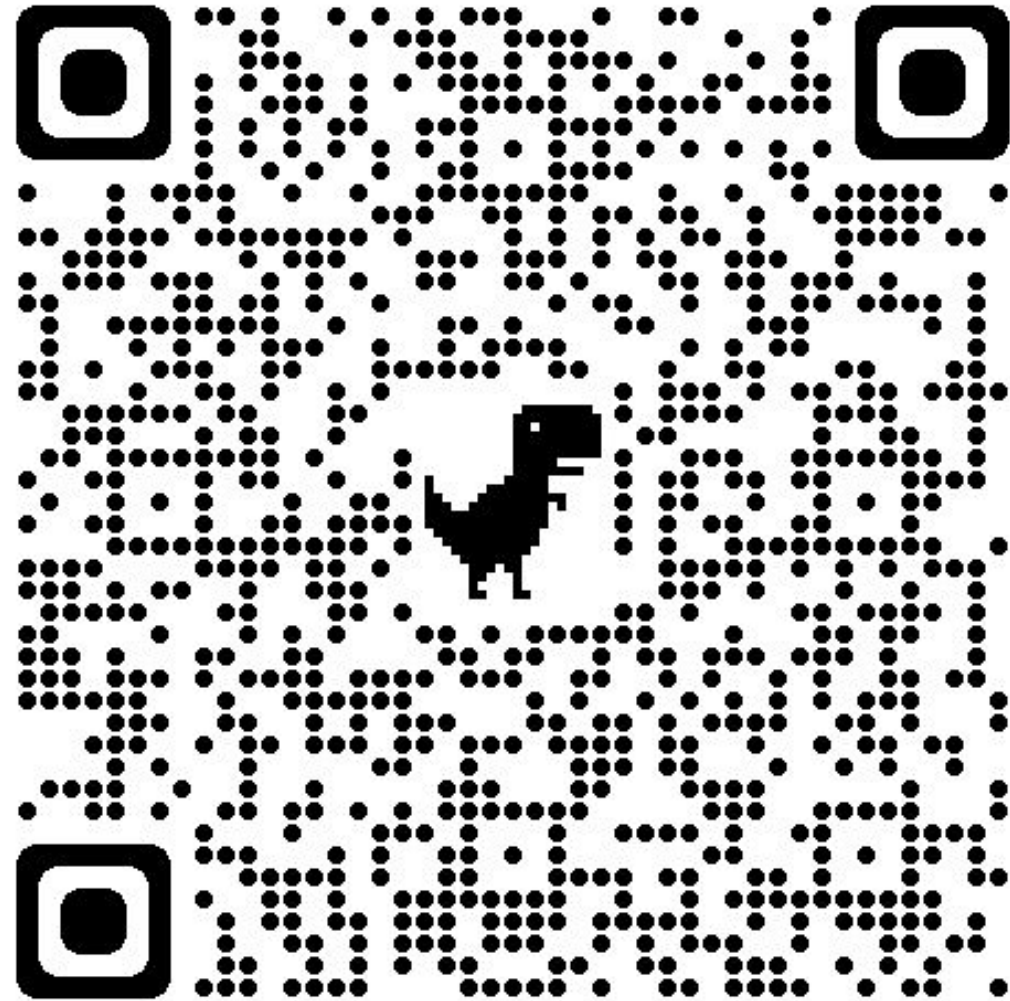
Examples of Possible Graduation Projects



GUI analysis for better input generation

- Challenge:
 - Some input fields require specific kind of input (for example a date or a number)
- Possible solutions:
 - Use behavior analysis during TESTAR execution to deduct the rules for a specific input
 - Analyze the GUI to deduct what kind of input is expected, for example the neighbouring widgets of an input widget, or hints of expected input shown for the user
 - Use of existing "faker" libraries for suitable data generation

Tutorial
Post-Question
naire





Open Universiteit

Empowering Students with Modern Skills and Connections Through Open Source GUI Testing with TESTAR

bmarin@dsic.upv.es

orv@ou.nl

Contact the TESTAR team:

<https://testar.org/>

<https://testar.org/tad2023/>

https://github.com/TESTARtool/TESTAR_dev

