

Evaluating Rogue User Testing: an Industrial Case Study at Softeam

Sebastian Bauersfeld

Nelly Condori-Fernandez

Tanja E. J. Vos

Etienne Brosse

Alessandra Bagnato

Technical Report UU-CS-2014-010

May 2014

Department of Information and Computing Sciences

Utrecht University, Utrecht, The Netherlands

www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Evaluating Rogue User Testing: an Industrial Case Study at Softeam

Sebastian Bauersfeld¹, Nelly Condori-Fernandez¹, Tanja E. J. Vos¹, Etienne Brosse², and
Alessandra Bagnato²

¹ Universidad Politécnic de Valencia, Valencia, Spain

<http://www.pros.upv.es/>

² SOFTEAM, Paris, France

Abstract. Testing applications with a graphical user interface (GUI) is an important, though challenging and time consuming task. The state of the art in the industry are still capture and replay tools, which may simplify the recording and execution of input sequences, but do not support the tester in finding fault-sensitive test cases. While search-based test case generation strategies, such as evolutionary testing are well researched for various areas of testing, relatively little work has been done on applying these techniques to an entire GUI of an application. In earlier works we presented the Rogue User Technique, which allows fully automatic testing of complex GUI-based applications to find severe faults such as crashes or non-responsiveness. In order to evaluate the utility and acceptance of the Rogue User in an industrial context, we carried out a case study with Softeam, a Paris-based company that participates in the FITTEST project. This document presents the results of this study.

1 Introduction

Software Testing is an important practice to assure the quality of and avoid critical errors in software products. However, modern software is usually very complex and uses advanced Graphical User Interfaces (GUIs) which can be very hard to test. The state of the art of testing such interfaces are Capture and Replay tools which are not as automatic as often suggested by their vendors.

However, tests that aim at critical faults, such as crashes and excessive response times, are completely automatable and can be very effective [4]. These robustness tests often apply random algorithms to select the actions to be executed on the GUI. Since random algorithms might not always exhibit satisfactory test performance, we presented a search-based approach to fully automate robustness testing of complex GUI applications with the goal to enhance the fault finding capabilities [4, 5]. The approach is called Rogue User Testing and uses a well-known machine learning algorithm called Q-Learning in order to combine the advantages of random and coverage-based testing.

In order to evaluate the applicability and acceptance of this technique in an industrial context, we conducted a case study with Softeam, a French software company participating in the European FITTEST project. Softeam develops Modelio SaaS, a cloud-based system to manage virtual machines that run their popular graphical UML editor Modelio. This system provides an environment for users of this editor to collaboratively work on UML documents in the cloud.

Modelio SaaS is a PHP-based solution, which runs in the browser, is attached to a user database and accesses the Amazon EC2 API to create virtual machine instances which run Modelio clients. Softeam utilizes a hand-written test suite to ensure the quality of this system.

We wanted to find out how much acceptance the Rogue User Tool can gain in this environment and among the testers and to what extend it can help to reduce the testing effort induced by the manual test suite.

2 Context – Where has the case study been performed?

Softeam is a private software vendor and engineering company with about 700 employees located in Paris, France.

One of Softeam’s priorities is to maximize user-interaction coverage of their test suites with minimum costs. Specifically, those test suites created for the Modelio SaaS system. To achieve this, Softeam is eager to investigate benefits and drawbacks of using the RU testing tool developed within FITTEST.

This motivation is mainly due to the fact that the current testing process has several limitations such as: i) Test case design and execution is and ii) resources for manual inspection of test cases are limited.

Learning to use and integrate the Rogue User Tool into Softeam’s current testing processes, could allow testers to reduce the time spent on manual testing and could even detect different types of faults, since the tool can generate sequences that are too complex for humans to conceive. The downside of this potential optimization is the extra effort and uncertainty that comes with applying a new test approach. To decide if this extra effort is worth spending, a case study has been planned and carried out. The results will support the decision making about whether to adopt the Rogue User Technique at Softeam.

3 Design of the case study

3.1 Objective - What to achieve?

The goal of the case study is to measure the learnability, the effectiveness, efficiency, maintainability and subjective satisfaction when using the Rogue User in the context of Modelio SaaS.

Analyze	the RU tool
For the purpose of	Evaluation
With respect to	Learnability, Satisfaction, Effectiveness, Efficiency and Maintenance
From the viewpoint	of the testing practitioner
In the context of	Softeam developing and testing for Modelio SaaS departments

RQ1 Learnability How fast are the testing practitioners able to pick up the Rogue User technique?

RQ2 Satisfaction How satisfied are SOFTEAM testers during the installation, configuration and application of the tool when applied in a real testing environment?

RQ3 Effectiveness and Efficiency How does the FITTEST Rogue User contribute to the effectiveness and efficiency of testing when it is used in real industrial environments and compared to the current testing practices at Softeam?

RQ4 Maintainability How much effort does it take to maintain the Rogue User Testing Infrastructure during the development of Modelio SaaS with potential changes to its user interface? These changes can render previous test cases or test setups invalid which might require setup adaptations. We want to find out how much manual labor is involved in this process.

3.2 Objects of the study: the SUT and the injected faults

The System Under Test (SUT) The SUT selected for this study is the Modelio SaaS system (Figure 1), which is a prototype developed at Softeam. Modelio SaaS is a PHP web application, that allows for easy and transparent configuration of distributed environments. It can run in virtual environments on different cloud platforms, offers a large number of configuration options and hence poses various challenges to testing [3]. In the case study we will focus on the web administration console, which allows server administrators to manage projects created with the Modelio modeling tool, and to specify user rights for working on these projects. The source code is composed of 50 PHP files with a total of 2141 lines of executable code.



Fig. 1. Modelio SaaS – the SUT for this case study

TS_{Soft} – Softeam’s existing manual Test Suite The existing test suite is a set of 51 manually crafted system test cases that Softeam uses to perform regression testing of new releases. Each test case describes a sequence of user interactions with the graphical user interface as well as the expected results. Figure 2 shows an example of such a test case.

Test Case SaaS-7: Create a customer account from the administrator account		
<u>Author:</u>	cba	
<u>#:</u>	<u>Step actions:</u>	<u>Expected Results:</u>
1	Sign in as a server administrator	
2	Go to "Compte clients" -> "Créer un compte client"	The form to fill customer's details is displayed.
3	Fill: <ol style="list-style-type: none"> 1. the 'nom' field with a name 2. the "date de soucription" field with a date with format "YYYY-MM-DD" 3. the "date de validité" field with a date with format "YYYY-MM-DD" 4. the 'login' field with a login 5. the 'mot de passe' field with a password 6. the 'e-mail' field with an email address 	
4	Click on 'Créer compte'	The 'Gestion des comptes clients' page must be displayed with a table containing the customer's details.
<u>Execution type:</u>	Manual	
<u>Keywords:</u>	None	

Fig. 2. Manual test case, used by Modelio SaaS testers for functional testing.

Injected Faults The faults to be injected have been selected after interviewing developers involved into Modelio SaaS development. Five types of faults have been identified. All of these faults occurred during the development of Modelio SaaS, which makes them realistic candidates for a test with the Rogue User Tool. Table 1 shows the list of faults, their descriptions and their identifiers,

which we will use for reference throughout the document. Figure 3 and Figure 4 show screenshots of two of the mentioned faults.

ID	Part of the Application	FileLocation	Description
1	Controller	AccountController.php line 102	When clicking on "no" for account deletion confirmation, the system nevertheless deletes the account
2	Controller	LoginController.php line 8	No login fields on login page
3	Controller	ProjectsController.php line 8	Empty page when accessing the project creation page
4	Controller	RamController.php line 31	Description is not added to the database when creating a component
5	Controller	RolesController.php line 48	Page not found error after editing a role
6	Model	DeploymentInstance.php line 10	"An error occurred" message when trying to view properties of a project (see Figure 3)
7	Model	Module.php line 21	"An error occurred" message when trying to add a module to a project (see Figure 3)
8	Model	Module.php line 34	"An error occurred" message when trying to upload a new module (see Figure 3)
9	Model	Project.php line 36	"An error occurred" message when trying to view managed project (need to be a project manager) (see Figure 3)
10	Model	ProjectModule.php line 29	"An error occurred" message when trying to view properties of a project (see Figure 3)
11	View	ComponentSelection line 19	Empty page when trying to add a component to a project
12	View	Modules.php line 18	Allow empty content for module
13	View	ModuleSelection.php line 30	Empty page when trying to add a module to a project
14	View	RoleSelection.php lines 27 to 30	"An error occurred" when trying to edit the role of a user of a project (see Figure 3)
15	View	Server.php line 42	The type of the server is missing
16	View	ServerSelection.php line 13	Empty form (Figure 4) when trying to move a server
17	View	Users.php line 82	Editing is possible when accessing through view link and vice versa

Table 1. Injected Faults

3.3 Cases or Treatments - What is studied?

The FITTEST way of doing Rogue User Testing Modern GUIs are large, complex and difficult to access programmatically which poses great challenges for their testability. The Rogue User is a technique that allows completely unattended testing of large and complex gui-based SUTs. Its basic sequence generation algorithm comprises the following steps:

1. Obtain the GUI's state (i.e. the visible widgets and their properties like position, size, focus ...).
2. Apply an oracle to check whether the state is valid. If it is invalid, stop sequence generation and save the suspicious sequence to a dedicated directory, for later replay.
3. Derive a set of sensible actions (clicks, text input, mouse gestures, ...).
4. Select and execute an action.
5. If the given amount of sequences has been generated, stop sequence generation, else go to step 1.

In its default mode, the RU selects the actions to be executed at random. We believe that this is a straightforward and effective technique of provoking crashes and reported on its success in [5].

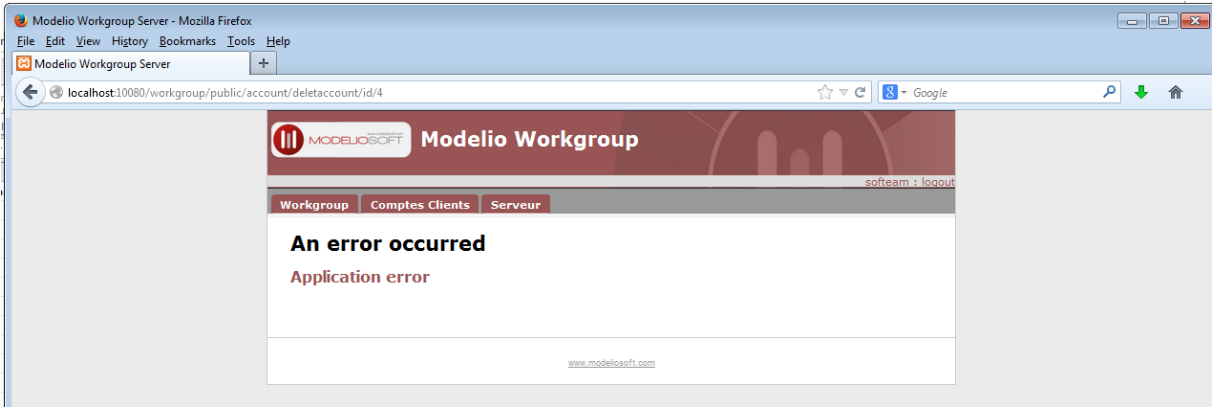


Fig. 3. Easy to detect fault with obvious error message.



Fig. 4. Difficult to detect fault (specific control elements are missing)

We were able to find 14 crash sequences for Microsoft Word while running the Rogue User during 48 hours³.

In large SUTs with many – potentially deeply nested – dialogs and actions, it is unlikely that a random algorithm will sufficiently exercise most parts of the GUI within a reasonable amount of time. Certain actions are easier to access and will therefore be executed more often, while others might not be executed at all.

Therefore, in [4] and [5] we presented an algorithm whose idea it is to slightly change the probability distribution over the sequence space. This means that action selection will still be random, but seldom executed actions will be selected with a higher likelihood, with the intend to favor exploration of the GUI.

The strength of the approach is, that it works with large, native applications which it can drive using complex actions. Moreover, the technique does not modify nor require the SUT's source code, which makes it applicable to a wide range of programs. With a proper setup and a powerful oracle, the Rogue User can operate completely unattended, which saves human effort and consequently testing costs.

The way (GUI) Testing is being done at Softeam Modelio SaaS' testing and development team consists of 1 product director, 2 developers and 3 research engineers who all participate in the testing process. The testing practice at Softeam is to create test cases by relying on specified use cases. Each test case describes a sequence of the user interactions with the graphical user interface as shown by Figure 2.

The test cases are managed with the *TestLink*⁴ software and grouped as test suites according to the part of the system that they enable to test. All them are executed manually by a test engineer. If a failure occurs, the test engineer reports it to the *Mantis*⁵ bug tracking system and assigns it to the developer in charge of the part affected by the failure. He also provides the Apache log file for the web UI as well as the Axis log file for the web services. Then, Mantis mails the developer in charge of examining/fixing the reported failure.

It took approximately 7 work days to design and build the test suite and together, the testers need an hour to execute all of the 51 test cases if no errors occur.

Softeam's testing process in projects other than Modelio SaaS is similar. A tester has access to the project specifications (most of the time a textual description). From this specification, he manually creates a first test suite under *TestLink* and then executes it. According to its results he will modify the test suite, enter failure(s) into *Mantis*, discuss the issues with the development team and re-execute the modified test cases, etc.

3.4 Subjects - Who applies the techniques?

We gave each of the testers a demographic questionnaire to fill out. The questions of and answers to this questionnaire are listed in Table 2. The subjects are two computer scientists that are novice testers from Softeam. Trainee one is a senior analyst (5 years), and trainee two is a software developer with 10 years of experience. Both have less than one year of experience in software testing: Both had previously modeled test cases using the OMG UML Testing Profile (UTP) [2] and the Modelio implementation of the UML Testing Profile [1]. In addition, both testers also claim to be proficient in Java, the language used to develop and extend the Rogue User Tool.

3.5 Variables - What is measured?

Independent variables:

- The Rogue User GUI Testing Tool (RU),

³ Videos of these crashes are available at <https://staq.dsic.upv.es/sbauersfeld/index.html>

⁴ <http://sourceforge.net/projects/testlink/>

⁵ <http://www.mantisbt.org/>

Question	Subject	
	Alessandra Bagnato	Etienne Brosse
How many years of programming experience do you have?	5	10
How long have you been working as a software tester?	1	1
In what types of testing activities did you engage? Briefly describe the projects you were working on and what your tasks have been.	"Test Case Modeling with UTP"	"Test Case Modeling with UTP"
What programming languages have you used in the past and which one are you most proficient in?	"Java"	"Java"

Table 2. Results of the Demographic Questionnaire

- Complexity of the SOFT case study system (Modelio SaaS),
- Level of experience of the SOFT testers who will perform the testing,
- The quality of the setup for the Rogue User as designed by the testers.

Dependent variables:

1. Measuring Learnability:

- (a) Perceived difficulties during the learning process
 - i. Learnability Interview with the Softeam testers. In this interview the testers will be asked which aspects of the tool / technique are intuitive and which ones are harder to understand.
 - ii. Learnability Interview with the trainer. The trainer will portray his view on the learning process of the testers. We will ask him about which parts of the technique the testers struggled with the most.
 - iii. Evolution of artifact quality as rated by the trainer. The trainer will assess the quality of the different artifacts (oracle, action definition, stopping criteria) that the testers will produce. The idea is that since the testers will produce different versions on their way to the final setup, we will see how much they do improve during the course of the study.
 - iv. Exam about the Rogue User Technique: The testers will have to show whether they understood the Rogue User Technique and know the necessary theory behind it.
- (b) Time needed to set up the testing infrastructure for the Rogue User. This includes:
 - i. Oracle Design + Implementation
 - ii. Action Definition + Implementation (the RU's behavior)
 - iii. Stopping Criteria (criteria which determine when a sufficient amount of testing has been performed)

2. Measuring Satisfaction:

- (a) Reactions through reaction word cards: After the case study, we will ask the testers to describe their experiences with the tool by choosing words from a word list as used by Benedek et al. [6]. They will only have a short period of time (5 minutes) to select the words that they think fit best.
- (b) We will carry out a satisfaction interview in which we will ask the testers about their opinion about the tool.
- (c) Face questionnaire to obtain information about satisfaction through facial expressions: We will ask the testers whether they would recommend the tool to their peers or promote its use to the management of their company. Meanwhile, we will observe their facial expression, in order to draw conclusions about their opinion on the tool. The purpose of the face questionnaire is to complement the satisfaction interview in order to determine whether their gestures harmonize with their given answers.

3. **Measuring Effectiveness.** For TS_{Soft} and TS_{RU} :
- Number of failures observed by both test suites. The failures relate to the ones in Table 1 that were injected into the current version of Modelio SaaS.
 - Achieved code coverage (We measured the line coverage of the PHP code executed by both test suites. We took this as an indicator of how “thorough” the SUT has been executed during the testing process)
4. **Measuring Efficiency** (Time will be measured in minutes).
- Time needed to design and develop test suites TS_{Soft} and TS_{RU} . In the case of the Rogue User we took the time that was necessary to develop the oracle, action definitions and stopping criteria.
 - Time needed to run TS_{Soft} and TS_{RU} .
 - Reproducibility of the faults detected by TS_{Soft} and TS_{RU} . Sometimes a test suite triggers a fault which is hard to reproduce through a subsequent run of the faulty sequence. Sometimes the environment is not in the same state as it was during the time the fault has been revealed, or the fault is inherently indeterministic. The timing of the tool used for replay can also have a major impact. We will measure what percentage of failures has been reproducible.

3.6 Protocol

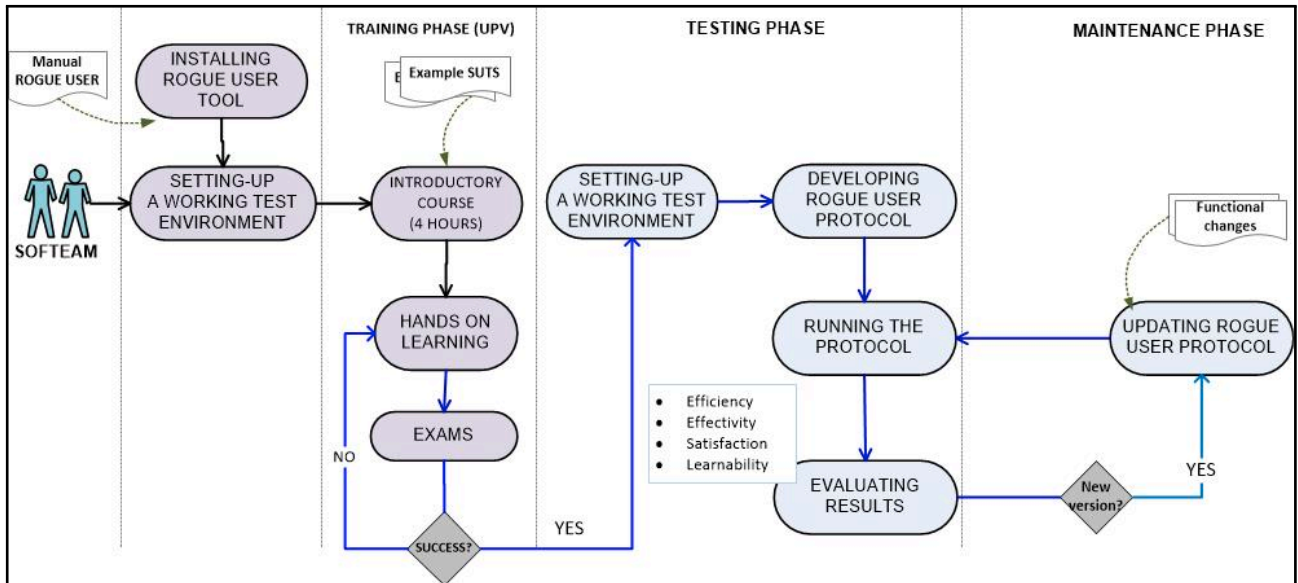


Fig. 5. Case Study Protocol

The case study has been divided into 3 phases:

1. **Training Phase:** A trainer will introduce the subjects to the Rogue User Testing Technique, provide working examples, point out challenges and difficulties and eventually gather first impressions of how well the subjects understood the concepts of the technique and whether they are prepared to proceed to phase 2.
2. **Testing Phase:** The subjects will apply the learned knowledge to the System Under Test. They will develop a test environment on their own. In case of problems they will consult the documentation or (in very problematic cases) the trainer (through Skype or similar platforms).

3. Maintenance Phase: The subjects will continue to use the developed test environment and maintain it as the System Under Test changes. This phase is important to evaluate long-term costs and potential problems with the Rogue User Technique and it will continue to run until after the end of the FITTEST project.

Training Phase The subjects start to develop a working test environment for Softeam's case study system.

1. The testers fill in a demographic questionnaire with information about their experience in the industry and the tools, programming languages and testing techniques they are familiar with.
2. Presentational learning through training in Rogue User testing: The trainer gives an introductory course to the Rogue User technique in which he presents working examples of Rogue User Tests. He uses example SUTs which are unrelated to the later case study system so that the subjects get an insight into:
 - How to setup the Rogue User for a given SUT?
 - How to tell the tool which actions to execute?
 - How to program an effective test oracle for different types of faults?
 - How to define and program stopping criteria?
3. Autonomous hands-on learning (i.e. learning by doing) with online help from the Rogue User course instructor through Skype. The subjects will apply the learned techniques on two example SUTs (unrelated to the later subject SUT). Each tester documents the progress in working diaries which contain information about:
 - (a) The activity that has been performed (for example: read the Rogue User manual, ask questions to the trainer, make an example Rogue User, write an email to the trainer, etc.
 - (b) Minutes spent on each activity
 - (c) The questions and doubts that the tester had at the time he was doing this activity (so one can see if those were solved in later learning activities)
 - (d) Artifacts generated, including:
 - i. Versions and evolutions of Rogue User Protocols that are produced during these hands-on learning activities
 - ii. Any other artifact that they think might be of interest.
4. In order to find out whether the testers are ready to go over to phase 2
 - (a) The trainer evaluates the generated artifacts and the maturity of the questions he receives.
 - (b) The trainer evaluates their knowledge with an exam that covers the provided training material.
 - (c) The trainer interviews the subjects.
 - (d) The trainer himself is interviewed to find out how he thinks about the progress of the testers.
5. The results are summarized to obtain values for variables 1.a).

Testing Phase The subjects start to develop a working test environment for Softeam's case study system.

1. Each tester develops a Rogue User Protocol (working diaries will be maintained) and measures the variables 1.b).
2. Each tester runs his / her Rogue User protocol and evaluates the results.
3. We will carry out a reaction cards session as has been done in [6]
4. Satisfaction interviews (answer 2 questions while recording the face of the interviewee).
5. The trainer will evaluate the quality of the protocols and the results of the test runs (subjective assessment as lecturer to evaluate how smart the protocols were made and how likely they are to work in later tests)

Maintenance Phase

1. The subjects will continue to use the Rogue User Technique in particular for new versions of the SUT. This will be a long-term phase in which the testers can experiment with the technique and figure out potential problems.
2. They will continue to measure variables 2.x and 3.x for each new SUT version, i.e. they will also measure the time that they need to adjust the current Rogue User protocol to the new SUT version (e.g.: the protocol might not work correctly after the GUI of the SUT changed, etc.).

4 Results and Discussion

To answer all posed research questions we followed the protocol described in Section 3.6. The very first thing that the participating testers had to do was to fill out a demographic questionnaire whose results are shown in Table 2. The table shows that both testers have sufficient programming experience yet are beginners in the field of testing. They both are familiar with the Java language which is used to extend the Rogue User and to write its oracle definition.

The case study took place during a period of 2 months. Table 3 lists the overall time of different activities that the testers and the trainer carried out throughout the course of the study.

Activities	Variable	Time reported in minutes		
		S1	S2	In Pairs
Oracle design + implementation	1b) i	1200	30	30
Action definition + implementation	1b) ii	820	30	20
Stopping Criteria	1b) iii	30	0	10
Evaluating run results	/	240	20	30
Skype meeting with trainer	/	60	10	15
Total time	/	2350	90	105

Table 3. Self-reported activities during the hands-on learning process. The results correspond to the variables 1b) i, ii and iii of Section 3.5.

4.1 RQ1: Learnability

To answer the question as to how learnable the tool is, and to get values for the variables 1a), we relied on the following information and resources gathered during the training and testing phases of the case study:

1. A learnability interview with both testers, where the testers describe positive and negative aspects of the tool and list the problems they had during the training and testing phases. Table 4 shows the results of this interview.
2. A learnability interview with the trainer. In this interview the trainer answered questions about the impressions he obtained during the introductory course, the hands-on training and the actual application of the tool in the testing phase. Table 5 shows the results of this interview.
3. A histogram of the quality of the artifacts generated during training and testing phases. The trainer had to assess the quality of oracles, action sets and stopping criteria on a scale between 0 and 5. During the course of the case study, the testers generated 4 increasingly powerful versions of the Rogue User setup. Figure 6 shows how the trainer rated those versions.
4. The introductory course exam that the testers had to fill in after the Rogue User presentation in Paris. Appendix .1 shows the test. The trainer used its results to rate learnability in Table 5.

In Tables 4 and 5 the interviewees had to answer the questions by indicating how much they agree on a scale between 0 and 5, with 0 being total rejection and 5 full consent. The questions in Table 4 have been taken from [7] where the authors are analyzing the learnability of CASE tools. They have been divided into 7 categories to separate different aspects of the tool. Averaged over all questions and both testers, the Rogue User tool scored 4.15 out of 5 possible points and a median of 4 which proves the testers' positive view on complexity and learnability of the tool.

Learnability Categories	Question	Tester's Answer	
		Alessandra Bagnato	Etienne Brosse
Ease of learning	1.a) It was easy for me to get started and to learn how to use the tool.	5	5
	1.b) I was able to use the tool right from the beginning, without having to ask my tutors or peers for help.	4	4
	1.c) The tool encouraged me to try out new system functions by trial and error.	5	5
	1.d) It was easy for me to remember commands from one session to another.	4	5
	1.e) The explanations provided during the introductory class, the Skype conversations and the documentation helped me to become more and more skilled at using the tool.	4	5
Familiarity	2.a) Was your prior knowledge of other computer-based systems useful in the learning of the RU tool?	5	4
	2.b) Was your prior knowledge of other testing tools useful in the learning of the RU tool?	5	4
Consistency	3.a) The tool is consistently designed, thus making it easier for me to do my work.	5	5
	3.b) I find that the same function keys are used throughout the program for the same functions.	3	5
Predictability	4.a) The tool behaves similarly and predictably in similar situations.	3	4
	4.b) When executing functions, I get results that are predictable.	3	5
Informative Feedback	5.a) Performing an operation leads to a predictable result.	5	5
Error Messages	6.a) If I make a mistake while performing a task, I can easily undo the last operation	3	5
	6.b) Error messages clarify the problem.	3	3
	6.c) I perceive the error messages as helpful	3	3
Specific Functions	7.a) I found it easy to setup powerful oracles that detect errors in the SUT.	3	4
	7.b) I found it easy to setup powerful action sets that drive the SUT and allow to find problematic input sequences.	3	5
	7.c) I found it easy to define when to stop tests / when the SUT was sufficiently tested.	3	5

Table 4. Learnability Interview (Variable 1a) i)

The results in Table 5 harmonize with this view. The trainer had a generally positive impression of how well the testers performed during the case study and gave on average 3.71 out of 5 points. However, he emphasizes that the testers had a few problems with the definition of the Rogue User's action set. This set defines the RU's behavior and is crucial to its ability to explore the SUT and trigger crashes. Action definitions comprise the description of trivial behavior such as clicks and text input, as well as more complicated drag and drop and mouse gestures.

Question	Answer
Did you get the impression that the testers internalized the learned material quickly?	4
Would you say that the testers made significant progress throughout the training phase?	4
What aspect of the Rogue User Testing technique caused the most difficulties for the testers?	Action Definition
How would you rate the quality of the resulting RU oracles as designed by the testers?	4
How would you rate the quality of the resulting RU action sets as designed by the testers?	3
How would you rate the quality of the resulting RU stopping criteria?	3
How well did the testers answer the questions asked in the written Rogue User Test?	4
Would you say that the testers only had few problems in setting up the Rogue User for tests?	4

Table 5. Learnability Interview – The trainer’s point of view (Variable 1a) ii).

Throughout the course of the case study, the testers developed 4 different versions of the Rogue User’s setup, with increasing complexity. The first setup offered a rather trivial oracle, which scraped the screen for critical strings such as "Error" and "Exception". The testers supplied these strings in the form of regular expressions. Obvious faults such as number 6 (see Table 1 for the list of injected faults), which is depicted in Figure 3, are detectable with this strategy. However, this heavily relies on visible and previously known error messages. More subtle faults, such as number 16, depicted in Figure 4, are not detectable this way. The second oracle version made use of the web server’s logging file which allowed to detect additional types of faults (e.g. errors caused by missing resource files, etc.). Versions 3 and 4 also incorporated a consistency check of the database used by Modelio SaaS. This makes sense, since certain actions such as the creation of new users, access the database and could potentially result in erroneous entries. Each new oracle version was accompanied by a corresponding action set and test stopping criteria. For example: The more powerful database oracle in version 3, requires appropriate actions, that heavily stress the database. Thus, the Rogue Users should prefer to create / delete / update many records. Finally, one needs to supply a reasonable test stop criterion, that defines when the SUT had been thoroughly tested.

Figure 6 shows the quality of the different Rogue Users setups, as rated by the trainer. The trainer rated each artifact of a version separately, i.e. oracle, action set and stopping criterion. The perceived quality increases with each version and eventually reaches its maximum in the last one. Although, as mentioned before, the trainer is not entirely satisfied with the quality of the testers’ action definitions, the graphic shows a clear increase in sophistication, indicating the ability of the testers to learn how to operate the tool.

4.2 RQ2: Satisfaction

To obtain information on whether the testers were satisfied with the tool, we used the following sources (Variables 2a), 2b) and 2c)):

1. A product reaction cards interview (Figure 7, where we asked the testers to mark words that they associate with the tool.
2. A satisfaction interview (Table 6), where we explicitly asked the testers what they think about the tool and how happy they have been with it.
3. A face questionnaire (Table 7 and Figure 4.2), where we asked two direct questions and observed the testers’ first facial reaction.

Table 6 shows the results of a direct satisfaction interview with the testers. Again, we used questions from [7] and in addition asked the subjects to describe in their own words the most positive and negative parts of the tool. The testers answered the questions about the tool predominantly positive and used pleasant words to describe its characteristics. However, both testers also agree on the necessity to improve the tool’s documentation.

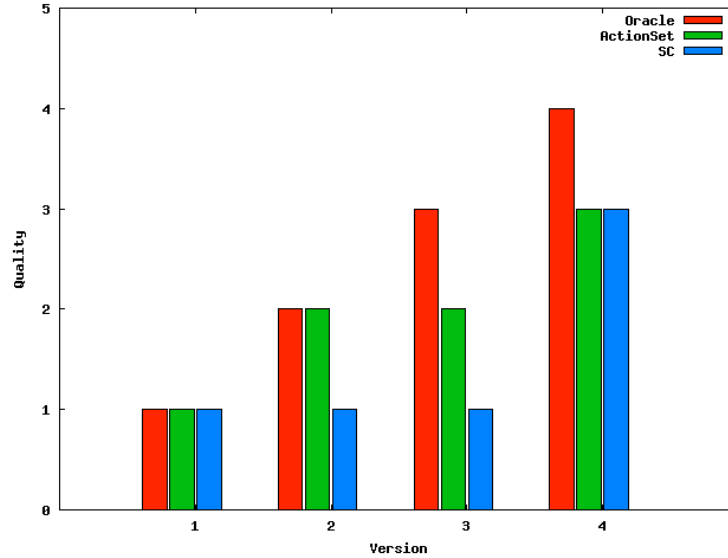


Fig. 6. Evolution of artifact quality as rated by the trainer (Variables 1a) iii)

Question	Tester's Answers	
	Alessandra Bagnato	Etienne Brosse
The use of the RU tool for this case study was a good idea.	4	5
The tool made my work more interesting.	4	5
The use of the RU tool enabled me to complete my tasks more quickly.	4	4
Name the three most positive features of the Rogue User	"Innovative, intuitive, time-saving"	"Highly extensible / configurable + deals with boring tasks + easy to use"
Name the three most negative features of the Rogue User	"The user friendliness of the tool could be improved adding notes from the quick manual in there. Might be nice to save configurations of the rogue user for certain tools and be able to load them again. Filters writing might be helped with some notes on the syntax."	"Needs more integrated help + Installation Issues (dll dependencies) + Startup actions need to be scripted"

Table 6. Satisfaction Interview (Variable 2b)

A second source that we used to gain insight into the testers' mind were reaction cards as defined in [6]. We gave the testers a list of words and asked them to mark the ones, that they associate the most with the Rogue User tool. We then kept the intersecting set of words chosen by both testers. Figure 7 shows the result where, again, we can see that the large majority of words have a positive connotation, such as "Fun", "Desirable", "Time-Saving" and "Motivating".

The complete set of 118 Product Reaction Cards				
Accessible	Creative	Fast	Meaningful	Slow
Advanced	Customizable	Flexible	Motivating	Sophisticated
Annoying	Cutting edge	Fragile	Not Secure	Stable
Appealing	Dated	Fresh	Not Valuable	Sterile
Approachable	Desirable	Friendly	Novel	Stimulating
Attractive	Difficult	Frustrating	Old	Straight Forward
Boring	Disconnected	Fun	Optimistic	Stressful
Business-like	Disruptive	Gets in the way	Ordinary	Time-consuming
Busy	Distracting	Hard to Use	Organized	Time-Saving
Calm	Dull	Helpful	Overbearing	Too Technical
Clean	Easy to use	High quality	Overwhelming	Trustworthy
Clear	Effective	Impersonal	Patronizing	Unapproachable
Collaborative	Efficient	Impressive	Personal	Unattractive
Comfortable	Effortless	Incomprehensible	Poor quality	Uncontrollable
Compatible	Empowering	Inconsistent	Powerful	Unconventional
Compelling	Energetic	Ineffective	Predictable	Understandable
Complex	Engaging	Innovative	Professional	Undesirable
Comprehensive	Entertaining	Inspiring	Relevant	Unpredictable
Confident	Enthusiastic	Integrated	Reliable	Unrefined
Confusing	Essential	Intimidating	Responsive	Usable
Connected	Exceptional	Intuitive	Rigid	Useful
Consistent	Exciting	Inviting	Satisfying	Valuable
Controllable	Expected	Irrelevant	Secure	
Convenient	Familiar	Low Maintenance	Simplistic	

Fig. 7. Product Reaction Cards (Variable 2a) – Developed by and © 2002 Microsoft Corporation. All rights reserved. The marks show the intersecting set of words chosen by both testers.

Finally, to cross-validate the testers claims, we conducted a face questionnaire as described in [6]. The idea is to elicit feedback about the product, particularly through emotions that arose in the participants while talking about the product (e.g. frustration, happiness). We video taped the testers while they responded to the questions in Table 7. We asked whether they 1) would recommend the tool to other colleagues and 2) whether they think they could convince their management to invest in it. Figure 4.2 shows their first facial expressions, right after the questions had been asked. The left and right hand sides relate to questions 1 and 2, respectively. The authors of [6] use a scale and a canonical facial expression to rate the expressions of the photographed subjects. Table 7 shows this scale. The more the subjects' faces resembled the given face, the less motivated and satisfied we assumed they were. For question 1, we assigned 3 / 8 points to the first tester, since he is smiling confidently and his face does not resemble the negative canonical face. This corresponds with the answers he gave. He argued that the Rogue User tool is quite suitable for many types of applications. He described it as time-saving, especially in the context of simple and repetitive tests. This allows him to concentrate on the difficult tests which are very hard to automate. His facial expression after question 2, is not so confident anymore. We assigned 7 / 8 points, since he does not look optimistic that he will be able to convince his management to invest

in the tool. He argues that this is difficult, since one would have to convince many people and prepare a strong business case.

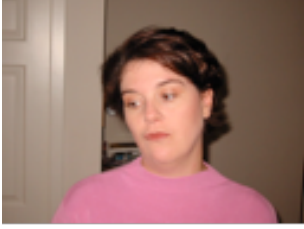
	Question 1	Would you recommend this tool to other colleagues? If not why? If yes what arguments would you use?						
	Question 2	Do you think you can persuade your management to invest in a tool like this? If not why? If yes what arguments would you use?						
		How much does the interviewee's face resemble the one depicted on the left?						
	1 Not at all	2	3	4	5	6	7	8 Very much

Table 7. Facial Expressions Chart (Variable 2c)



Fig. 8. Face Questionnaire: Images on the left show reactions to question 1, on the right to question 2 of Table 7.

Similar to tester 1, the face of tester 2 appears positive and affirming. Additionally, she argues that the tool is very easy to use. We assigned 2 / 8 on the image scale, since her smile is more intense than that of the first tester. Contrary to tester 1, however, she is also more confident to be able to promote investment of the tool within the company. We assigned 3 / 8 points for the second question. She argues that, although she would need strong arguments, it would be realistic to convince the people in charge, due to the tool's quality.

In summary, despite, small criticism regarding the documentation and installation process of the tool, the testers' reactions and statements encountered during the interviews and the face

questionnaire, indicate that they were satisfied with the testing experience. We came to a similar conclusion regarding the tool’s learnability. Although, the trainer reported certain difficulties with the action set definition, the constant progress and increase of artifact quality during the case study, points to an ease of learnability.

4.3 RQ3: Effectiveness and Efficiency

To answer the third research question regarding the efficiency and effectiveness of the Rogue User, we compared the existing manual test suite (TS_{Soft}) with the test suite generated by the Rogue User Tool (TS_{RU}). To make a judgement on this matter we compared the variables 3.x and 4.x of Section 3.5 of both test suites against each other. Table 8 shows the values of these variables. To obtain data for TS_{RU} we used the last of the 4 versions of the setup for the Rogue User Tool created by the testers during the case study. However, we also included the time necessary to develop the earlier versions in the development time, since these intermediate steps were necessary to build the final setup. To measure the variable values for TS_{Soft} we employed Softeam’s current manual test suite for which the company has information about man hours dedicated to its development.

Description	Variable	Test Suite	
		TS_{Soft}	TS_{RU}
Faults discovered	3a)	14 + 1	10 + 1
Code coverage	3b)	86.63%	70.02%
Time spent on development	4a)	40h	36h
Run time	4b)	3h 10m	77h 26m
Faults reproducible	4c)	100%	91.76%
Number of test cases	/	51	dynamic
Number of executed sequences	/	294	90

Table 8. Comparison between manual and Rogue User generated tests

The way both test suites were executed was as follows:

1. In the case of the manual test suite TS_{Soft} , a tester started manually carrying out the specified test cases until an unexpected result, i.e. a fault was detected. In the case of the Rogue User we started the tool and let it run until it triggered and detected a fault.
2. The responsible tester then analyzed the fault, tried to reproduce it by replaying the corresponding sequence and eventually fixed the fault, before proceeding with the testing process.
3. This process has been repeated until none of the test cases triggered any (detectable) fault.

As we can see in the table, the manual test suite (TS_{Soft}) and the suite generated by the Rogue User (TS_{RU}) obtained in some cases substantially different results. TS_{Soft} consists of a fixed set of 51 hand-crafted test cases, whereas TS_{RU} does not comprise specific test cases, but rather generates them as needed. Softeam states to have spent approximately 40 hours of development time on crafting the manual test cases, which roughly equals the 36 hours that their testers needed to setup the Rogue User for the final test (including earlier setup versions). The first major difference is the number of actually generated sequences. Due to the many faults contained in the tested SUT version, many test cases had to be repeated after fixing a fault, in order to make sure that they then executed flawless. Despite this, the Rogue User tool generated considerably less sequences and needed significantly more time to do so. This is due to the fact, that those sequences are often very long (the testers set an action limit of 500 for the tests), whereas the manual sequences only consist of a few actions necessary to carry out the specified test cases.

As mentioned before, the running times of the two suites largely differ. The testers took about 3 hours to execute all manual test cases and fix the faults, whereas the setup that they conceived

for the Rogue User, ran about 78 hours, before no faults were triggered anymore and the tool stopped its execution. Of course they could have decided to perform a shorter run, but since the tool works completely automatic and ran over night, it did not cause any manual labor. The only thing that the testers had to do, was to, in the mornings, consult the logs for potential errors, fix these and continue to run the tool. This is a very positive outcome compared to the manual test suite, which caused more than 3 hours of human labour.

In terms of code coverage, the manual suite outperformed the automatically generated tests. However, the difference of approximately 16% is modest. Manual testing allows the tester to explore forms that might be locked by passwords or execute commands that require specific text input. A way to enable the Rogue User to explore the GUI more thoroughly, would be to specify more complex action sets. We consider this as a plausible cause, as the trainer pointed out, that he was not entirely satisfied with the action definitions that the testers designed (see Figure 6).

Finally, considering the amount of seeded faults that have been detected by both suites, the manual tests, unsurprisingly, outperformed those generated by the Rogue User tool. TS_{Soft} detected 14 of the seeded faults and the testers even found a previously unknown error. All of the erratic behaviors were reproducible without any problems. TS_{RU} , on the other hand, detected 11 faults, including the previously unknown one. However, as expected, the tool had problems detecting certain kinds of faults, since it can be hard to define a strong oracle for those. Examples include errors similar to number 16 (Figure 4). Nevertheless, obvious faulty behavior, which often occurs after introducing new features or even fixing previous bugs, can be detected fully automatic. Given the low amount of manual labor involved in finding those, the Rogue User tool can be a useful addition to a manual suite and could significantly reduce manual testing time. One definite advantage, that the RU has over the manual suite is, that the setup can be replayed arbitrary amount of times, at virtually no cost, e.g. over night, after each new release. The longer the tool runs, the more likely it is to detect new errors. We think that the development of a powerful oracle setup pays off in the long term, since it can be reused and replayed without demanding valuable human time.

4.4 RQ4: Maintenance

To be able to answer the last research question about how much effort is necessary to maintain a Rogue User Setup for consecutive versions of the SUT and how this effort compares to the one necessary to maintain the current suite TS_{Soft} , we will continue to work with Softeam beyond the time span of the FITTEST project. We are particularly interested in how stable Rogue User setups are against changes in the SUT's user interface and how automatic the approach really is. Therefore, we will observe difficulties with both TS_{Soft} and TS_{RU} during each new release of Modelio SaaS. In particular, what problems are encountered, what is necessary to fix them, and finally how much time is necessary to adjust the suite to the new conditions.

The results of this study will be delivered at a later stage as an addendum to this document.

5 Conclusions

In this document we presented the results of a case study that we carried out together with Softeam, a software vendor located in Paris, France. We settled out to determine whether the automated Rogue User tool will be accepted and appreciated by the testers in an industrial context. Therefore, we trained two novice testers on how to use the tool and how to design an effective setup. We then accompanied and assisted them in setting up a test environment for Softeam's Modelio SaaS solution, a web-based administration system to manage collaborative environments for the popular *Modelio* client. We observed their progress and evaluated the artifacts they developed. The satisfaction interviews that we conducted yielded predominantly positive feedback, both from the testers, which claimed enjoying the work with the tool, as well as from the trainer, who observed a constant learning progress. The constant improvement of the developed artifacts and the positive

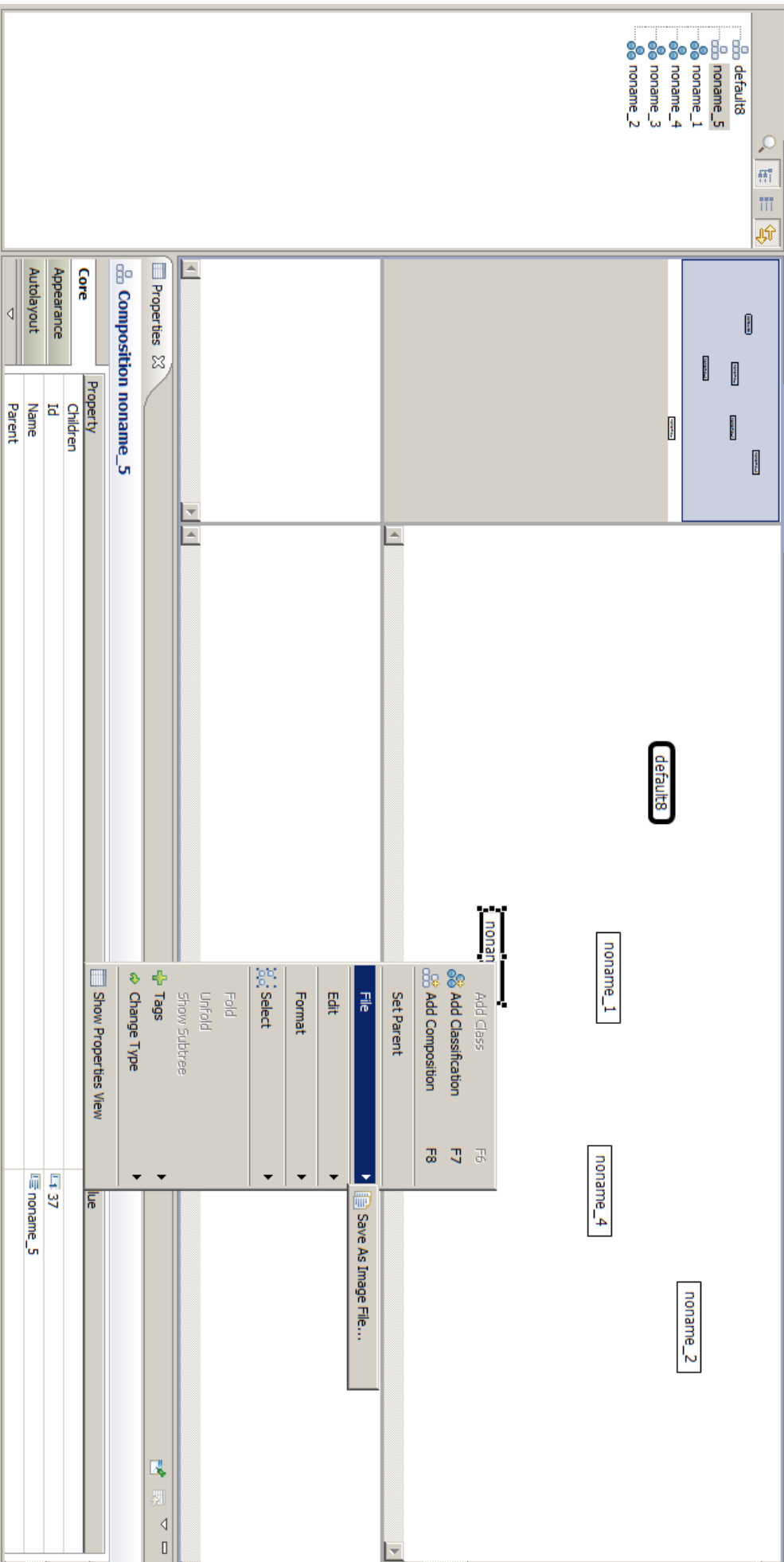
results of a test with an erroneous version of Modelio SaaS, support the view that the Rogue User is a practical and learnable technique, accepted by the testers.

We furthermore wanted to see how effective and efficient the solutions developed by the testers were compared to Softeam's current manual testing practice. Therefore, Softeam provided an erroneous version of their SUT, with realistic faults observed during its development. We ran the Rogue User Tool and the current manual test suite on this version and measured the time and the amount of faults revealed by each technique. The results were surprisingly positive, as the Rogue User managed to detect several severe faults while involving less manual labor than the current approach. The different types of faults detected by both test suites lead us to conclude that the Rogue User technique is a valuable supplement for a manual test suite, which can reduce the manual testing burden and increase testing effectivity.

Acknowledgment

This work was financed by the FITTEST project, ICT-2009.1.2 no 257574.

.1 Introductory Course Test



Properties

Composition noname_5

Core

Property

Children

Id

Name

Parent

- Add Class F6
- Add Classification F7
- Add Composition F8
- Set Parent
- File
- Edit
- Format
- Select
- Fold
- Unfold
- Show Subtree
- Tags
- Change Type
- Show Properties View

Save As Image File...

References

1. Modelio implementation of the uml testing profile: <http://www.modeliosoft.com/en/modelio-store/modules/modeling-extensions/utp.html>, October 2013.
2. Uml testing profile (utp) web site, online: <http://utp.omg.org/>, October 2013.
3. A. Bagnato, A. Sadovykh, E. Brosse, and T.E.J. Vos. The omg uml testing profile in use—an industrial case study for the future internet testing. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 457–460, 2013.
4. Sebastian Bauersfeld and Tanja Vos. A reinforcement learning approach to automated gui robustness testing. In *In Fast Abstracts of the 4th Symposium on Search-Based Software Engineering (SSBSE 2012)*, pages 7–12. IEEE, 2012.
5. Sebastian Bauersfeld and Tanja E. J. Vos. Guitest: a java library for fully automated gui robustness testing. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012*, pages 330–333, New York, NY, USA, 2012. ACM.
6. J. Benedek and T. Miner. Measuring desirability: New methods for evaluating desirability in a usability lab setting. *Proceedings of Usability Professionals Association, Orlando, USA*, 2002.
7. Mali Senapathi. A framework for the evaluation of case tool learnability in educational environments. *Journal of Information Technology Education: Research*, 4(1):61–84, January 2005.