Fostering the Diversity of Exploratory Testing in Web Applications

Julien Leveau, Xavier Blanc, Laurent Réveillère, Jean-Rémy Falleri, Romain Rouvoy

▶ **To cite this version:**

Julien Leveau, Xavier Blanc, Laurent Réveillère, Jean-Rémy Falleri, Romain Rouvoy. Fostering the Diversity of Exploratory Testing in Web Applications. ICST 2020 - IEEE International Conference on Software Testing, Verification and Validation, Mar 2020, Porto, Portugal. 10.1109/ICST46399.2020.00026 . hal-02398969

HAL Id: hal-02398969

https://inria.hal.science/hal-02398969v1

Submitted on 27 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Fostering the Diversity of Exploratory Testing in Web Applications

Julien Leveau*†, Xavier Blanc*§, Laurent Réveillère*, Jean-Rémy Falleri*, Romain Rouvoy‡§
* Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France
† CIS Valley, Bruges, France,
‡ Univ. Lille / Inria, France
§ IUF, France
Email: {julien.leveau, xavier.blanc, laurent.reveillere, falleri}@labri.fr, romain.rouvoy@univ-lille.fr

***Abstract*—Exploratory testing (ET) is a software testing approach that complements automated testing by leveraging business expertise. It has gained momentum over the last decades as it appeals testers to exploit their business knowledge to stress the system under test (SUT). Exploratory tests, unlike automated tests, are defined and executed on-the-fly by testers. Testers who perform exploratory tests may be biased by their past experience and therefore may miss anomalies or unusual interactions proposed by the SUT. This is even more complex in the context of web applications, which typically expose a huge number of interaction paths to their users. As testers of these applications cannot remember all the sequences of interactions they performed, they may fail to deeply explore the application scope.**

This paper therefore introduces a new approach to assist testers in widely exploring any web application. In particular, our approach monitors the online interactions performed by the testers to suggest in real-time the probabilities of performing next interactions. Looking at these probabilities, we claim that the testers who favour interactions that have a low probability (because they were rarely performed), will increase the diversity of their explorations. Our approach defines a prediction model, based on n-grams, that encodes the history of past interactions and that supports the estimation of the probabilities. Integrated within a web browser extension, it automatically and transparently injects feedback within the application itself. We conduct a controlled experiment and a qualitative study to assess our approach. Results show that it prevents testers to be trapped in already tested loops, and succeeds to assist them in performing deeper explorations of the SUT.

***Index Terms*—test, exploratory test, n-gram, web applications**

I. Introduction

Contributions. In this paper, we address this objective and we strive at fostering the diversity of exploratory testing. In particular, we address the challenging case of web applications, as these applications are widely developed and deployed nowadays.

It should be noted that, when the model of the system under test (SUT) exists, a test coverage can likely be computed and fostering diversity therefore consists in advising the testers to explore uncovered parts of the SUT. In some other cases however, when there is no existing model of the SUT and/or when creating one is highly costly, advising the testers to explore uncovered parts cannot be achieved by leveraging on test coverage metrics. Furthermore, building the model of a web application remains a complex problem that is not solved yet [1], and out of the scope of this paper.

To reach this challenging objective, we introduce a prediction model based on n-gram language models. More precisely, we continuously train a prediction model by assimilating the test traces that were previously performed by the testers. Then, when a tester is conducting a new test scenario, we ask the prediction model to propose the next interactions of relevance for the test, which are the ones frequently performed. We then advise the tester not to follow the prediction but, in the opposite, to consider interactions with a low percentage of prediction. If the tester is following our advise, she would then perform new or rarely performed interactions, which will increase the diversity of the test. Once the interaction is performed, we train again the prediction model (with the new test trace), which will then ask for more diversity for the next tests.

The contributions of this paper can therefore be summarized as follows:

1) We present the **test tracker** we use to monitor user interactions with the SUT;
2) We introduce our **prediction model**, based on n-gram language models, that encodes previous performed tests and predict interactions of relevance for the current test;
3) We report on an implementation of our approach as a **server** and a **browser extension** that can automatically inject predictions within any web application as visual highlightings;
4) We describe a **controlled experiment** in which exploratory testers perform a Exploratory testing (ET) session with and without using our approach. In this experiment, we monitor the diversity of the tests performed by each tester;
5) We report on a **case study** in which we gather valuable feedbacks by industrial partners.

Our results assess that our approach fosters testers to collaborate on exploring a wider space of a target SUT. Furthermore, the feedbacks expressed by ET experts indicate that they found the information visually reported by the tool useful. This information helps testers to be more focused on the next interaction to trigger and thus to increase the diversity

of their tests.

The remainder of this paper is organized as follows. Section II introduces the principles of ET. Section IV presents our prediction model, while Section V reports on its implementation in the context of web applications. Section VI assesses the effectiveness of our approach, showing that our solution boost the diversity of explored scenarios. Section VII reports on industrial case studies we performed and the feedbacks we collected in this context. Finally, Section IX concludes and highlights some perspectives for this work.

II. Exploratory Testing of Web Applications

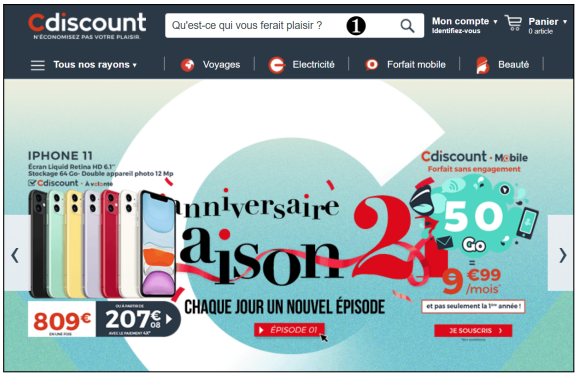
In this section, we introduce and illustrate the core concepts related to our contribution. To that extent, we describe a sample ET session performed on a representative example of modern web applications. We use this example to highlight the problems we address and to overview our approach.

A. ET running example

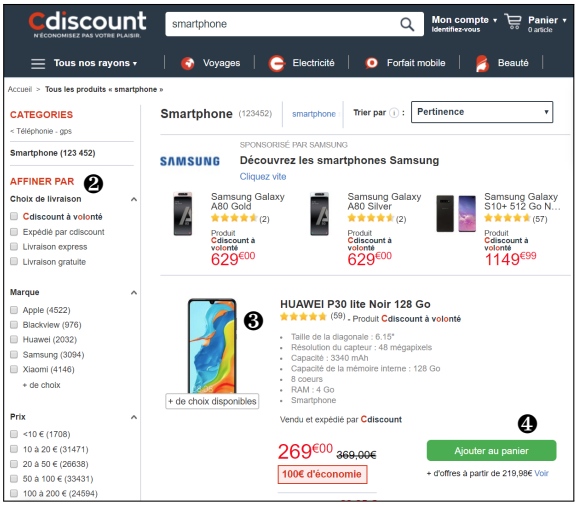
As a running example, we consider an ET session of Cdiscount,[1] which is the online store of one of the largest e-commerce seller in France. The session is performed collaboratively by Alice, Bob and John, whose goal is to identify any anomaly when ordering a smartphone from the website.

The session starts with Alice who uses the search text field to search for "smartphone" (cf. Figure 1a – ❶). Then, after browsing the set of listed products to assess that all of them are smartphones (cf. Figure 1b), she adds the first one to her basket ❹. Finally, in the last step, after having checked that the price is consistent (cf. Figure 1c), she orders the product ❻.

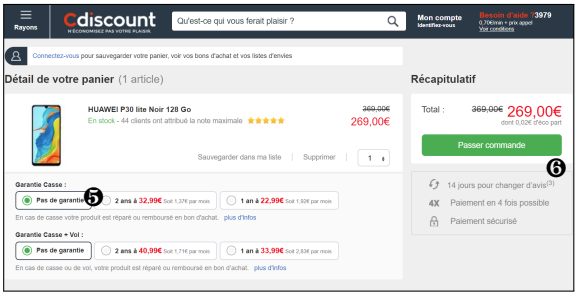
To carry on this session, the group of testers will have to perform more tests, which should increase the diversity and the chances to find defects. In particular, Alice, Bob and John should try several filters ❷, view and select among different listed products ❸ and ❹, and choose different options ❺. Performing such diverse tests can however become problematic because of the three reasons we briefly introduced in Section I. First of all, a tester may be biased by her knowledge of the SUT and therefore may miss or ignore some interactions. In our case study, a search query can be performed by filling the search text field and then clicking on the 'enter' key, which most of the users do. However, another possibility is to fill the search text field and then to click on the magnifying glass button that is located next to the search text field. Such an interaction may be skipped by testers, especially if they are used to press the 'enter' key. Secondly, the complexity of the application may make it difficult or even impossible for the testers to remember which tests they already performed in the session. In our scenario, there are many filters, which moreover depend on the searched product. Knowing which ones have been selected, or not, is almost impossible, thus preventing the tester to efficiently test all possibilities. Finally, when several testers want to collaborate in a session, they have to know what are the scenarios that the other testers have explored



(a) Searching for a product



(b) Filtering, viewing and selecting the product



(c) Ordering options

Figure 1: Product ordering workflow in Cdiscount e-commerce application

or are doing. When such a knowledge cannot be obtained, the SUT may be decomposed into several subparts, and each tester is assigned to a specific subpart. For instance, Alice can be assigned to the filters ❷, Bob to the selection of the product ❸ and John to the options ❺. However, such a decomposition is difficult to achieve in practice and prevent to cover end-to-end tests that span the adopted decomposition.

[1] https://www.cdiscount.com

B. Advising testers to perform interactions with low or null percentage of prediction

To foster the diversity of ET, we argue that testers should be advised to perform the interactions that were not (or rarely) performed earlier in the session.

Such an advise can be easily done if a model of the SUT exists and if test coverage can be computed. If so, the test coverage can tell which parts of the SUT and which interactions have been tested. Advising the testers would then just consist in inviting them to perform not yet covered parts and interactions. Application model and test coverage are however rarely available in the context of ET. Furthermore, it has been shown that automatically building a model of an existing web application is a very complex problem, which has not been closed yet [1]. Web applications, especially the single page applications ones [2], heavily use asynchronous events, whose effects change the graphical user interface in a nondeterministic way, making this problem even harder. At first, our running example appears to be only composed of three main pages, with clearly identified interactions. However, the second page (cf. Figure 1b) depends on the previous query, and keeps changing whenever the user updates the filter. As it cannot be classified as a single unique page, automatically generating a model from it would end in a huge number of similar, but different, pages. We further consider that the model of the SUT is not available, and that the test coverage cannot be computed.

To foster diversity, our proposal leverages on NLP (*Natural Language Processing*) and more precisely n-gram language models, but distorts their usage. More precisely, we consider that the tests that have been previously performed can be integrated into a n-gram language model used to predict future interactions. Such a prediction model however proposes interactions that have been frequently performed in the previous tests. For instance, if we consider that only Alice has trained the prediction model with her first test scenario, the model will then only be able to propose interactions performed by Alice (it only knows these ones). We then propose to move away from this prediction, and invite the tester to consider interactions that are poorly predicted or even not predicted at all. For instance, if Bob starts the second test of the session, the prediction model would advise him to start by searching for 'smartphone', as this is what Alice did. By analyzing all the interactions Bob could perform, we can invite him to rather click on the magnifying glass button, which has a null probability as a prediction score. Indeed, doing so can contribute to increase the diversity of the session.

Later on, with this test, if Bob selects the first listed product, the prediction model would then advise him to order it (as Alice did). Once again, by looking at all the interactions Bob could perform, we can invite him to rather select some options before to order. This should again increase the diversity of the session.

Much more later, when Alice, Bob and John would have performed several tests, the prediction model would certainly be able to predict all the interactions that could be performed by the testers. The prediction model would however propose the testers to perform the interactions frequently performed in the past. Once again, by distorting the prediction, we can invite testers to perform actions with low probability of prediction, which should increase diversity.

As a summary, our proposal mainly consists in building a n-gram language model for ET and to distort it with the objective to invite testers to initiate rarely performed interactions. This first requires to dynamically track interactions performed by testers and to convert them in sequence of words for being integrated within a n-gram language model. Secondly, it requires to compute the predictions, to identify not yet performed interactions if any, and to point to the testers the less predicted ones. Section III describes how we monitor a test progress, and how we convert it as a sequence of words. Section IV introduces the n-gram language model we built, and how we distort its predictions.

III. Online Monitoring of Exploratory Tests

During an ET session, a tester navigates through a web application by performing various types of interactions, such as clicking on a button, filling a text field, and even moving the mouse over some elements. Each of these interactions triggers one or more JavaScript events in the Web browser of the tester.[2] In the context of an ET session, however, only some of these JavaScripts events are of interest. For example, with our e-commerce application (cf. Figure 1), the ET session focuses on a business workflow that starts with a product search and ends with a product order. Only the JavaScript events that are in this scope are of interest, such as the $click$ JavaScript event that targets the search bar element. We therefore consider that, at a lower and technical level, a test is represented by a sequence of JavaScript events of interest.

To track the tests that are performed by the tester, following the Micken *et al.* approach [3], we register an event listener on top of the DOM tree of each page (at the $window$ element). Thanks to this listener, we record all the JavaScript events that are triggered by the tester's interactions. Then, we process each event of interest and generate a word that reflects the input event. In our example, the word `SearchClick` abstracts the $click$ event that targets the search bar element. A test scenario, at an abstract and conceptual level, is thus represented as a sequence of words, and can therefore be integrated in a n-gram language model.

Finally, we argue that deciding which JavaScript events should be abstracted as a word is specific to the web application targeted by the ET session. Similarly, the word representation of a JavaScript event is an abstract representation whose semantics is relevant only in the context of the ET session. To this end, we introduce a declarative language, based on a JSON syntax, that supports the expression of mapping rules stating which JavaScript events should be processed, and how to convert them as words.

[2] https://www.w3schools.com/jsref/dom_obj_event.asp

Therefore, before starting an ET session on a new web application, a set of mapping rules should be created (or selected if an existing one fits the objective of the session). Then, whenever an ET session is started, and whenever a tester interacts with some pages, all the JavaScript events are monitored and checked against the mapping rules one by one (in their order of declaration). The first rule that captures the JavaScript event is used to produce its word abstraction, which is appended to the current sequence that captures the ongoing test scenario. Once a JavaScript event has been captured by a rule, the other rules are not considered anymore. As a consequence, each JavaScript event is captured at most once.

A mapping rule contains two parts. The *match* part defines the conditions that need to be fulfilled for a JavaScript event to be captured. The *output* part defines how to convert the matched event into a word.

Listing 1: Search product mapping rule

```
{
  "match": {
    "event": "keyup",
    "code": "Enter",
    "selector": ".hSrcInput > input",
    "multi": "false"
  },
  "output": {
    "prefix": "SearchProduct",
    "suffix": "value"
  }
}
```

Listing 1 illustrates a mapping rule for our running example that handles search queries performed on the search bar. More precisely, the match part of this rule filters in the events performed on the search bar when the tester validates a query by pressing the 'Enter' key. The output part generates a word that starts with `'SearchProduct'` and ends with the content of the value attribute of the event's target. For instance, when Alice performs her 'smartphone' query, the generated word is `'SearchProduct$smartphone'`, which represents an interpretable abstraction of what Alice performed.

The matching part of a mapping rule is defined using the mandatory fields *event* and *selector*. The *event* field defines the type that a JavaScript UI event must have to be captured.[3] If the *event* field defines a type that is among `keypress`, `keyup` or `keydown`, then the optional *code* field defines the code of the key that the event must capture. The *selector* field defines a CSS selector that is used to identify a list of nodes within the DOM tree. The event is captured only if its target is a member of this list. If the optional *multi* field is set to `false`, the event's target must not only belong to the list of identified nodes, but must also be the first element. The value of the *multi* field defaults to `true`.

The output part of a mapping rule is composed of two fields. The value of the *prefix* field is mandatory and defines the prefix of the generated word. The value of the *suffix* field can be `none`, `index`, `value` or `inner`. If its value is `none`, the generated word is equal to the *prefix* field value only. If the *suffix* field is set to `index`, the event's target must be a member of the list that is matched by the selector. In that case, the generated word is concatenated with the index of the event's target in that list. Note that setting the *suffix* field to `index` implies that the `multi` field is set to `true`. If the *suffix* field is set to `value`, the generated word is concatenated with the value of the value attribute of the event's target. If the *suffix* field is set to `inner`, the generated word is concatenated with the inner text of the event's target. Finally, when a *suffix* exists, the concatenation of the *prefix* with the *suffix* is separated by the '$' character.

[3]UI Events are those typically implemented by visual user agents for handling user interaction.https://www.w3.org/TR/uievents/

IV. Real-time Prediction of Interactions

Our prediction model builds on n-gram language models, which are a class of probabilistic models that assume one can predict the probability of some future words by only looking few words into the past [4]. With n-gram (of size n), if s is a sequence of words ($s = w_1, \ldots, w_{m-1}$, with $m > n$), then $P(w_m|s) = P(w_m|w_{m-n}, \ldots, w_{m-1})$ refers to the probability of observing the appending of w_m to the known sequence s. In our context, a word w abstracts a JavaScript event triggered by a tester, and a sequence s is a test scenario that is being performed (see Section III). A n-gram language model can then predict what should be the next abstract interaction just by considering the $n-1$ past ones. Knowing the mapping rules (cf. Section III), it is then possible to identify what are their corresponding JavaScript events and hence, which DOM elements can be targeted to trigger them.

One limitation to the use of n-gram language models, however, is that the value of n must be determined *a priori*. In our context, if n is small, then few past interactions are needed to predict what should be the next one. With our running example, if $n = 2$ for example, then only one previous interaction is needed, which is not large enough to know where the tester is in the product order workflow. If n is too big, for example $n = 10$, then 9 previous interactions are needed and should be the same as the ones performed by the tester to predict the next one. Such a condition would rarely occur, and hence limit the reliability of predictions. To address this limitation, Tonella *et al.* propose to choose an interval rather than a single n, and to compute predictions of each k of the interval [5]. Then, a global prediction P^* is computed as an exponential interpolation of the predictions over each k in the interval. More formally, if $P_k(w|s)$ is the prediction for a given k in the interval, then $P^*(w|s) = \alpha \sum_{\forall k} 2^k . P_k(w|s)$, where α is is a normalization factor. Our model follows this approach.

Figure 2 illustrates three test scenarios performed by Alice, Bob and John at the beginning of their ET session. We consider that all of these three test scenarios have been ingested in a single model shared for the ongoing testing session. Now, we consider that Alice is initiating a new test, and that she starts by triggering the interaction 'SearchProduct$smartphone'. If we ask the model for a prediction, it will look for any k-grams where the $k-1^{nth}$ words match the test being performed by

Alice	SearchProduct$smartphone, AddToBasket$1, ShowBasket, Order
Bob	SearchProduct$smartphone, AddToBasket$1, ShowBasket, RemoveItem$1
John	SearchProduct$smartphone, ShowDescription$1, ShowBasket, Order

Figure 2: Three tests performed by Alice, Bob and John

Alice. As Alice performed only one interaction from now, the model will then look for any 2-grams where the first word matches 'SearchProduct$smartphone'. Three 2-grams will be returned: ⟨SearchProduct$smartphone, AddToBasket$1⟩ two times, and ⟨SearchProduct$smartphone, ShowDescription$1⟩. The prediction for the next interactions will therefore be: 'AddToBasket$1' with 0.66 of probability, and 'ShowDescription$1' with 0.33.

We now consider that Alice is continuing her session, which becomes: SearchProduct$smartphone, AddToBasket$1 and ShowBasket. If we ask again the model for a prediction, it will look for 2-grams, 3-grams and 4-grams, as Alice now did performed 3 interactions. It will return two 4-grams (⟨SearchProduct$smartphone, AddToBasket$1, ShowBasket, Order ⟩), which correspond to the test previously performed by Alice and Bob, and one 2-gram in Bob test: ⟨ShowBasket, Order⟩. The prediction will favor 4-gram by multiplying their probability by 2^4 (the probability of 2-gram is multiplied by 2^2). The prediction will therefore output 'Order' with approximately 0.55, and 'RemoveItem$1' with 0.45.

The designed prediction system, as illustrated by this example, tends to natively invite testers to perform interactions that were frequently performed along the previous tests. However, to foster diversity, we rather invite the tester to perform interactions with low probability, and even interactions that are not predicted yet, but that can still be performed. To that extent, after each interaction, we use the left part of the mapping rules to identify all the elements that may be the target of a JavaScript event of interest in the current page (cf. Section III). Then, thanks to the right part of the rules, we can generate alternative interactions to be performed by the tester.

For example, with Alice, while the prediction model returns 'Order' and 'RemoveItem$1' with 0.55 and 0.45 of probability, she could also perform the 'QuantitySelect' interaction, which is used to change the quantity of selected products. This interaction is identified thanks to the mapping rules defined for that SUT, but was not predicted by the model and has therefore no probability. Our system therefore promotes this interaction to Alice in addition to the other ones.

V. Exploratory Testing Toolkit

As a proof of concept to assess our contribution, we have developed a testing toolkit that fully supports our approach. This toolkit is composed of two components: a server that supports collaborative Exploratory testing (ET) sessions, and a Google Chrome extension that supports the interactions with the testers.[4]

The server is written in JavaScript, for a total of $7,662$ lines of code (LoCs). It supports the management (creation and update) of mapping rules written in our declarative language (cf. Section III). For instance, all the mapping rules we use for our running example (Cdiscount) are available on our open server.[5]

The server further provides three services:

1) Creation of a new collaborative ET session with its associated prediction model (cf. Section III and IV). A session can be configured by setting its associated mapping rules and the value of the interval for the n-gram model. When a session is created, the server provides a unique id for the session that should be used by testers to collaborate;
2) Learning of a prediction model for a session. This is done by sending to the server a sequence of words that abstract the interactions performed so far by a tester (cf. Section IV);
3) Computation of some predictions. This is done by sending to the server a sequence of words that represents past interactions performed by a tester. The server then returns all possible interactions and their associated probabilities.

The browser extension is developed in JavaScript, for a total of 681 LoCs. Each tester that collaborates to a ET session should start up the extension by setting the session id provided by the server. The extension then provides the following services:

1) Monitoring of all the interactions performed by the tester and generation of their corresponding words (cf. Section III);
2) Fetching from the server the predictions for the next interactions given the words corresponding to the sequence of interactions performed by the tester;
3) Visually highlighting the DOM elements to render the predictions returned by the server. For each predicted interaction, the extension selects the mapping rules having the same prefix as the action, executes a querySelectorAll on the selector part of the rule, and applies a CSS border color property around the returned element. If a suffix is defined, the elements returned by the querySelectorAll are filtered according to the type of suffix specified. By default, the extension defines the four following colors for the border color:
   - An element is framed in *blue* when the testers *never* interacted with it (zero probability);
   - An element is framed in *green* when the testers *sometimes* interacted with it (probability is lower than 40%);
   - An element is framed in *orange* when the testers *often* interacted with it (probability is lower than 80%);

[4]Testing toolkit available from https://researchexperimentation.fr/
[5]https://researchexperimentation.fr/config/update

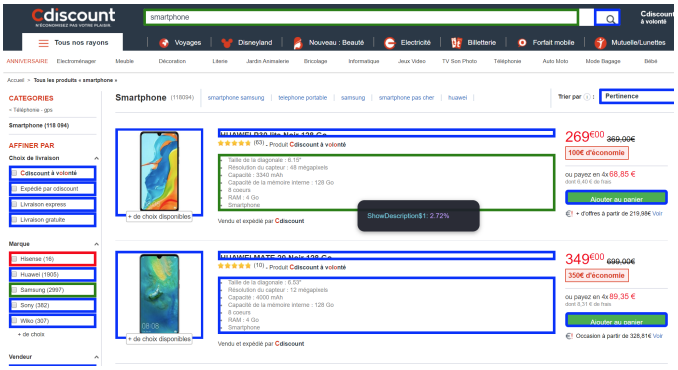


Figure 3: Product ordering page on Cdiscount

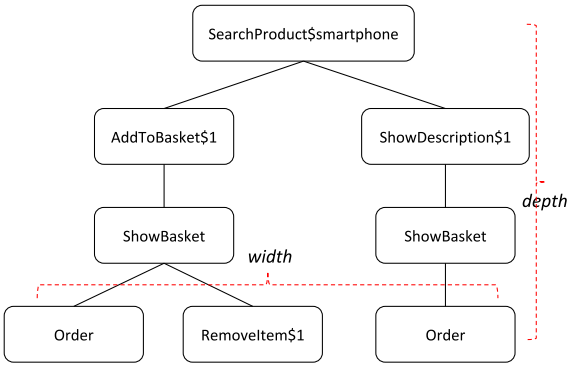


Figure 4: Navigation tree of the session of Figure 2.

- An element is framed in *red* when the testers *very frequently* interacted with it (probability is greater than 80%).

In addition, a popup is shown when the tester move the mouse over the elements of interest. The popup displays the real percentage of the prediction and, if any, the index, value or inner text that have been used in the previous interactions.

Figure 3 depicts a screenshot of our testing toolkit on our running example. Several elements of the page have been highlighted. Some of them have no probability and are then highlighted in blue. Others have some probabilities and are highlighted in red and green. In the middle of the screen, the mouse is over the description of the first phone. A popup then displays the corresponding probability (2.71%).

VI. Controlled Study

In this section, we describe the controlled study we performed to investigate the hypothesis that our approach assists a group of testers to explore web applications in a more "diverse" way.

A. Experimental setup

To that extent, we recruited 39 students out of 42 enrolled in the last year of a software engineering curriculum. We randomly split these students in 13 groups of three students. We want to study the impact that our approach has on the diversity of the test scenarios explored by a group of testers during a session. To do so, we setup a repeated measure design [6]. Therefore, each group is performing one session with our assistant, and one session without.

To setup some realistic exploratory test sessions, we used a production-scale web application: Cdiscount, our running example. As a test objective, we asked the groups to investigate the search and ordering pipeline of Cdiscount, as previously depicted in Figure 1. We presented to each group the elements of the GUI that were part of the test session, and then requested to find the usability issues in this part of the application. Each group completed two sessions of 10 minutes: one with our assistant, one without. Our assistant used a prediction model configured with $N = 8$, and we wrote the mapping rules for the application. The order in which the groups performed the assisted and not-assisted sessions was decided by doing the following. In the course schedule, there were two classes at two days of interval. Therefore, we arbitrarily chose one class to perform the assisted session first, and the other class to do the reverse. Finally, among the 13 groups of students, 5 groups started using the assistant and 8 started without. For each group, we record the test sequences produced in the two sessions.

Our goal is to measure the impact on the diversity of the tests performed in a given test session. To define a quantifiable metrics for this, we introduce the concept of *navigation tree*, which is computed from the test sequences performed along a session. This tree is the prefix tree of all the test sequences performed during the session, as shown in Figure 4. From the navigation tree, we derive two metrics. First, the *width of a session* (S_w), which is defined as the maximum width of the navigation tree. Second, the *depth of a session* (S_d), which id defined as the maximum depth of the navigation tree. Both S_w and S_d are positive integers. Their values in our sample navigation tree are illustrated in Figure 4. The larger the width and depth, the more diverse the tests performed in the session. In addition to these two metrics, we also introduce a third one that aims at quantifying the amount of repetition performed by a group of testers within a session. This measure, called *diversity ratio* or S_u, analyzes all the n-grams computed during a session and is defined as the number of distinct n-grams divided by the number of all n-grams. S_u is a real number between 0 and 1, 1 meaning that all explorations are unique (as all n-grams are unique) and a value close to zero indicates that many n-grams have been extracted many times, thus indicating a high repetition. Therefore, the greatest the diversity ratio, the greatest the diversity of the tests performed during a session.

Then, we define three null and alternatives hypotheses:

- H_0^1 : using our assistant has no effect on the width of a session and H_a^1 using our assistant increases the width of sessions,
- H_0^2 : using our assistant has no effect on the depth of a session and H_a^2 using our assistant increases the width of sessions,
- H_0^3 : using our assistant has no effect on the diversity

ratio of a session and H_a^3 using our assistant increases the diversity ratio of sessions.

To assess our hypotheses, we first measure for each group the values S_d^a, S_w^a and S_u^a (resp. S_d^0, S_w^0 and S_u^0) resulting from the explorations they performed in the session with the assistant (resp. without). To assess the significance of the results, we use a one-tailed paired Wilcoxon signed rank test. This test is non-parametric, as we do not have any assumption about the distributions of our measures. For each test where we accept the alternative hypothesis, we also report on the effect size, using Rosenthal's r [7] that we interpret using the common thresholds [8]: small for $0.1 <= r < 0.3$, moderate for $0.3 <= r < 0.5$ and large for $r >= 0.5$.

B. Experimental results

Figure 5 depicts the results we obtained with regards to the width, depth and diversity ratio. As a general trend, we notice that the values for the assisted sessions of groups are very often higher than the values for the not assisted sessions. To assess the significance of this observation, we run three Wilcoxon signed rank tests, as previously explained.

Concerning the depth, the p-value of the test is 0.01371, therefore we reject the null hypothesis and accept the alternative hypothesis. The effect size is $r = 0.621$ that we can interpret as large. Concerning the width, the p-value of the test is 0.0001221, therefore we reject the null hypothesis and accept the alternative hypothesis. The effect size is $r = 0.882$ that we can interpret as large. Concerning the diversity ratio, the p-value of the test is 0.04016, therefore we reject the null hypothesis and accept the alternative hypothesis. The effect size is $r = 0.494$ that we can interpret as moderate.

In conclusion, our assistant has a significant and large effect on the depth and the width of the navigation tree extracted from a session. Assisted sessions lead to navigation trees with a bigger width and depth. Our assistant has also a significant and medium effect on the diversity ratio computed from a session. Assisted sessions therefore lead to a bigger diversity ratios. All these results corroborate our hypothesis that our assistant helps groups of testers to perform more diverse tests.

C. Threats to validity

Regarding the internal validity, we had 5 groups that started the experiment by using our assistant, and 8 without. If we assume that our experimental design may suffer from an exhaustion or learning effect, it might have affected the results. However, the difference between our situation and the most fair situation (6 vs 7 groups) is small, therefore the effect of these biases is low in our opinion. Regarding the external validity, the participants of the experiment are students who are familiar with the application, but not experts. Therefore, there is no evidence that the results would generalize to experienced testers or domain experts. Also, our application was from the domain of e-commerce, but there is no evidence that our results would generalize to other types of web applications. Finally, the duration of the sessions in our experiment was 10 minutes. Our results might not generalize to longer or shorter sessions.

VII. Case study

The case study we report in this paper was conducted in collaboration with CIS Valley, which is a company with a software development activity. CIS Valley seeks to improve their testing processes in order to facilitate the evolution of the software products they develop. The goal of our case study was to check that our approach provides benefits in a industrial context.

A. Context

CIS Valley employs 160 people in 5 agencies. The company, among other activities, develops Sonate, an ERP (*Enterprise Resource Planning*) mainly used by public organizations. The development of Sonate started in 2017 and was estimated for a cost of 3000 man-days. Sonate is a web application composed of a frontend, based on the Angular framework, and a backend written in C#.

The project involves three teams (feature, development and migration teams) supervised by two managers and one project director. The feature team is composed of three senior business consultants who are in charge of the functional aspects of Sonate. The feature team manually writes validation test scenarios, which are executed manually during the internal recipe phase. The development team includes four developers: a frontend developer and three backend developers. Three developers have more than five years of experience, the other one is a junior developer. Development teams cultivate a culture of testing, whether by code coverage using unit tests or manual writing of end-to-end automated tests. The migration team is composed of four senior members who are in charge of transferring and synchronizing the previous version of Sonate with the newer version.

The nature of Sonate involves many business rules that still change frequently, involving significant changes in the code base. Because test scenarios are written manually, these changes make it difficult to maintain a baseline of test scenarios. For this reason, CIS Valley is interested in applying exploratory tests.

B. Method

The methodology we followed is composed of three steps:

1) Defining the mapping rules for Sonate,
2) Running two ET sessions (one with our approach in a hidden mode, one with our approach in a show mode),
3) Performing a post analysis of the session.

1) mapping rules: To scope the ET sessions, and to define the mapping rules for Sonate, we asked to the managers on which part of the application they wanted to focus on. They selected the part used for create and modify records corresponding to individuals registered in the ERP. From a technical point of view, this part is composed on only one page (Single Page Application), composed of multiple components. Some components are mandatory and are used to fulfill basic

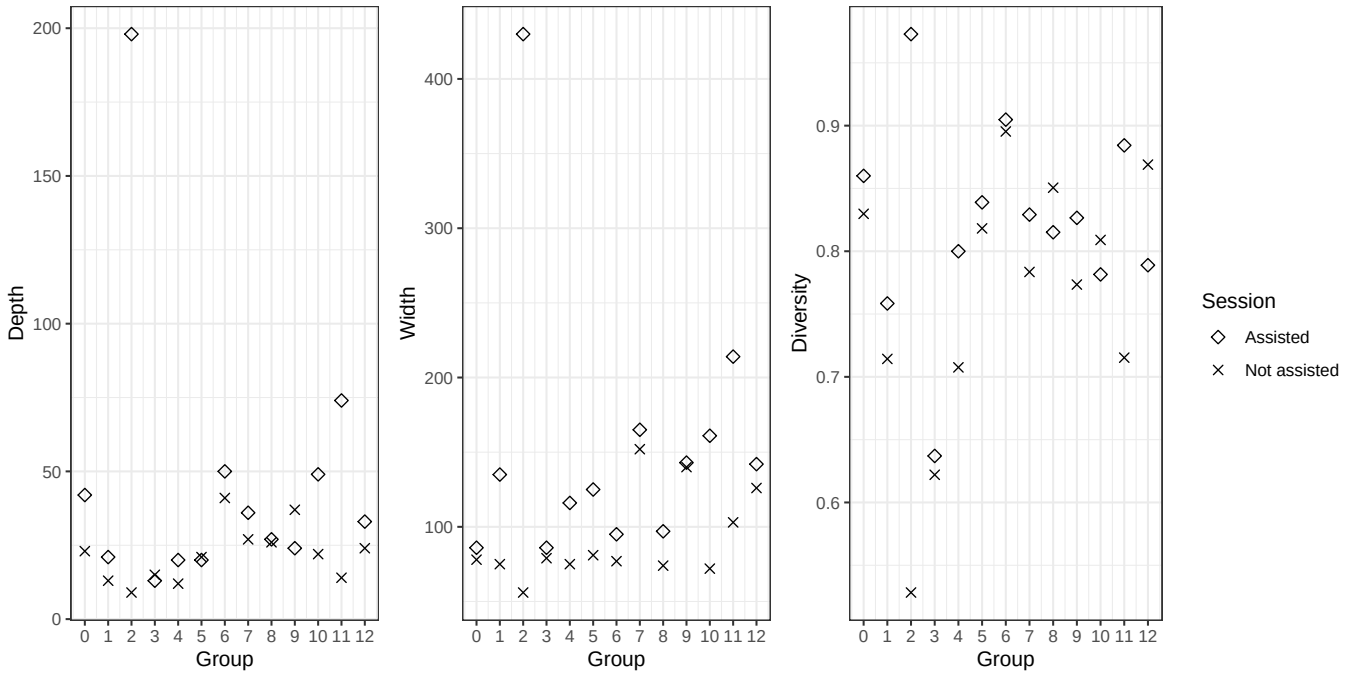


Figure 5: The depth, width and diversity of assisted and not assisted sessions of the groups.

identification information: firstname, lastname, birthdate, etc. Other components are optional, and can even be enabled or disabled dynamically.

Fortunately, the backend provides an introspection API that lists the components contained in a given web page of the application. When applied on the page targeted by our case study, this API reports all the components contained in that page. We then used this API to automatically generate our mapping rules (cf. Section III). In this way, we managed to generate 153 mapping rules.

2) ET sessions: Eight employees of the company participated to the two ET sessions. Seven of them were stakeholders: two managers, three business consultants and two members of the migration team. One was no more a stakeholder, but used to be a stakeholder of the previous version. Their level of knowledge of Sonate, from a user point view, was therefore heterogenous. One tester declared having a very good knowledge, one declared to have never used it. The other testers have some knowledge. Five of them declare having a good knowledge about software testing, and two about exploratory testing.

Before running the first session, we briefly presented the principles of exploratory testing. Then, the testers performed a first session of 20 minutes. During this first session, the testers used our approach in a hidden mode. They did not receive any prediction, and cannot see highlights.

We then introduced our assistant before running the second session by performing a 5 minutes demonstration on the Cdiscount website. The testers then performed the second sessions with our test assistant in a show mode. Testers received predictions, and saw the highlights. Unfortunately, because of a technical issue, we had to stop the second session after 13 minutes, which impacts our observation, as discussed in the following.

3) Survey: Three weeks after the two sessions, we submitted a survey to the testers. We asked for their opinion by a rating of 1 to 5. We asked few questions about exploratory testing in general and then several questions about our approach and our testing toolkit. The questions and answers are available online[6].

C. Practitioner Feedbacks

We received 6 answers from the 8 testers.

From a general point of view, it has emerged that the use of exploratory testing is a practice of great interest to them, especially as it allows teams with a heterogeneous knowledge of the system under test (SUT) to gain knowledge about the target application.

Regarding our approach, respondents noted 4.5 on average, when asked if they found it useful to have real-time visual feedback of probabilities. It turns out all of them found useful to display the probabilities with a color code, and that the probabilities were considered as relevant. They were not disturbed by the fact that the actions must be configured *a priori* via the declarative language we propose. In addition to the rendering of probabilities, the respondents expressed the wish to be able to recover the sequences of interactions performed when a bug is found, or even to be able to generate a test script to replay a faulty test scenario.

Respondents also appreciated the collaborative nature of our plugin. They particularly appreciated the fact that the

[6]https://researchexperimentation.fr/survey

interactions of the other testers were taken into account in the predictions. Finally the respondents noted 4 the choice of a Chrome plugin, installed directly in the browser, mostly because they would like to be able to perform additional sessions with alternative browsers.

VIII. Related Work

In this section, we discuss related work in the fields of exploratory testing, the use of n-grams in software engineering, and the testing of web applications.

Exploratory testing. The ET of web applications is related to the GUI testing of web applications, as both domains consider that web applications are black boxes and that testers (automated or not) interact with them [9]. In particular, our concept of *action* is closely related to the concept of *event* that is used in the event-flow model [10], where each interaction is modeled by the event it raises. Finally, our goal of increasing the diversity of the tests is similarly reached by Yuan *et al.* who propose to increase the test coverage by automatically generating long sequences of events [11]. Furthermore, they show in their study that long sequences of action have higher chances to detect defects. The main difference is that ET does not require a design model, while GUI testing aims to build a model (event-flow model) prior to specifying the tests and executing them.

N-gram models and software engineering. The use of n-gram models has shown to be very useful in software engineering, for example to measure source code quality and to suggest modifications [12]–[14]. Recently, Wand *et al.* mention that n-gram models may be useful to model execution traces and then be used to compare different coverage criteria [15]. To the best of our knowledge, there is no existing approach that uses n-gram models to guide testers in the context of exploratory testing. However, Tonella *et al.* [5] used such models to guide the generation of tests when using model based testing. They introduced and successfully used interpolated n-grams to reduce the chance of generating an infeasible path. In this paper, we use their model but with a different purpose: increasing the diversity of test scenarios in an exploratory test session. Additionally, we do not assume the existence of a navigation model for the web application. Sant *et al.* [16] also successfully used n-gram models (unigrams, bigrams and trigrams) to automatically generate test cases by looking at the requests sent along user sessions. Finally, Sprenkle *et al.* [17] also use n-gram models to build navigation models of web applications, by using the requests triggered by the users. They build on the approach of Sant *et al.* [16] and experiment different ways of representing the requests and choosing the right value of n for the n-gram model. They find out that their technique is effective for n ranging from 1 to 10. In contrast to the two previous approaches, our approach uses the model of Tonella *et al.* [5] and not a model with a fixed value of n. Additionally, our models are based upon abstract actions, computed from JavaScript events.

Web applications testing and visualization. The testing of web applications have been widely investigated in the past [18]. According to this survey, a large range of test approaches have been developed for web applications: graph and model based techniques, mutation testing, search-based techniques, crawling techniques, random and fuzz techniques, concolic testing and finally user-session based testing. In this paper, we introduce an approach to foster the diversity of exploratory testing for web applications. Exploratory testing is complementary to these techniques.

Our approach relies on CSS selectors to track the interactions between the testers and the GUI elements. Such selectors can break which may introduce bias when tracking testers interactions [19]. As a perspective, we plan to use more robust approaches as ROBULA for example [20].

The visual indicators used in our approach are similar to the so-called heatmaps [21]. A heatmap highlights the parts of a GUI that are frequently used by a user by looking at the position of the mouse's cursor. Therefore, the visual indicators used by heatmap do not take into account the preceding actions performed by the user. In contrast, our indicators consider these actions by using the interpolated n-grams, as previously described.

IX. Conclusion

Exploratory Testing is gaining more and more attention as it allows testers to leverage on their knowledge of the system they test to find defects. One of the difficulties of ET, however, is the ability to efficiently test the SUT, especially when there is no model of the system and no model can ever be built.

This paper addresses this issue by introducing an approach that helps testers who participate to an ET session to be more focused on the next interaction to take and thus to increase the diversity of their tests. Our approach is based on n-gram language models that make the assumption that a prediction of a future interaction can be done just by looking at the previous ones. We propose to monitor the tests performed by the testers, and to dynamically train a n-gram model. Thanks to this model, we encourage testers to perform interactions rarely performed during the session. Our approach comes with a declarative language that is used to define which parts of a SUT should be monitored and how the events triggered by the performed interactions should be converted into words to be integrated in a n-gram model.

We further developed a tooling that fully supports our approach. It is composed of a server and a Chrome extension. The server handles ET sessions that are performed collaboratively by groups of testers. The chrome extension assists the testers by highlighting them which elements of a web application should be the targets of future interactions.

We validated our approach by performing a controlled study and a case study. The controlled study was performed on a large e-commerce web site. It has shown that our approach supports testers to perform deeper, wider and more divers tests. Our case study was performed on a SME that is currently developing a large ERP. It has shown that our declarative language can be used to express mapping rules for an industrial

web application. Further, it has also shown that our tooling can be used by industrials who want to run ET sessions.

Finally, our approach provides predictions, which can be seen as new metrics for tests performed in ET sessions. These metrics invite testers to dynamically change their behavior to foster diversity. As a perspective, we are working on using these metrics to assess test diversity, with the idea to be able to stop an ET session when the diversity does not progress anymore.

REFERENCES

[1] A. Mesbah, A. v. Deursen, and S. Lenselink, "Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes," *TWEB*, vol. 6, no. 1, p. 3, 2012. [Online]. Available: http://doi.acm.org/10.1145/2109205.2109208

[2] A. Mesbah and A. v. Deursen, "Migrating Multi-page Web Applications to Single-page AJAX Interfaces," in *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, Mar. 2007, pp. 181–190.

[3] J. Mickens, J. Elson, and J. Howell, "Mugshot: Deterministic Capture and Replay for Javascript Applications," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 11–11, event-place: San Jose, California. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855722

[4] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based N-gram Models of Natural Language," *Comput. Linguist.*, vol. 18, no. 4, pp. 467–479, Dec. 1992. [Online]. Available: http://dl.acm.org/citation.cfm?id=176313.176316

[5] P. Tonella, R. Tiella, and C. D. Nguyen, "Interpolated N-Grams for Model Based Testing," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 562–572, event-place: Hyderabad, India. [Online]. Available: https://doi.org/10.1145/2568225.2568242

[6] N. Salkind, "Encyclopedia of Research Design," Thousand Oaks, California, Feb. 2020. [Online]. Available: http://sk.sagepub.com/reference/researchdesign

[7] R. Rosenthal, "Parametric measures of effect size." in *The handbook of research synthesis.* New York, NY, US: Russell Sage Foundation, 1994, vol. 621, pp. 231–244.

[8] P. Ellis, "Thresholds for interpreting effect sizes," *Retrieved January*, vol. 13, p. 2014, 2009.

[9] A. A. Andrews, J. Offutt, and R. T. Alexander, "Testing Web applications by modeling with FSMs," *Software & Systems Modeling*, vol. 4, no. 3, pp. 326–345, Jul. 2005. [Online]. Available: http://link.springer.com/10.1007/s10270-004-0077-7

[10] A. M. Memon, "An event-flow model of GUI-based applications for testing," *Softw. Test., Verif. Reliab.*, vol. 17, no. 3, pp. 137–157, 2007. [Online]. Available: https://doi.org/10.1002/stvr.364

[11] X. Yuan, M. B. Cohen, and A. M. Memon, "GUI Interaction Testing: Incorporating Event Context," *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 559–574, Jul. 2011.

[12] S. Han, D. R. Wallace, and R. C. Miller, "Code Completion from Abbreviated Input," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. Auckland, New Zealand: IEEE, Nov. 2009, pp. 332–343. [Online]. Available: http://ieeexplore.ieee.org/document/5431761/

[13] ——, "Code completion of multiple keywords from abbreviated input," *Automated Software Engineering*, vol. 18, no. 3, pp. 363–398, Dec. 2011. [Online]. Available: https://doi.org/10.1007/s10515-011-0083-2

[14] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the Naturalness of Software," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 837–847, event-place: Zurich, Switzerland. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337322

[15] Q. Wang, Y. Brun, and A. Orso, "Behavioral Execution Comparison: Are Tests Representative of Field Behavior?" in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Mar. 2017, pp. 321–332.

[16] J. Sant, A. Souter, and L. Greenwald, "An exploration of statistical models of automated test case generation," *ACM SIGSOFT Software Engineering Notes*, vol. 30, pp. 1–7, 2005.

[17] S. Sprenkle, L. Pollock, and L. Simko, "A Study of Usage-Based Navigation Models and Generated Abstract Test Cases for Web Applications," in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, Mar. 2011, pp. 230–239.

[18] Y.-F. Li, P. K. Das, and D. L. Dowe, "Two decades of Web application testingA survey of recent advances," *Information Systems*, vol. 43, pp. 20–54, Jul. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437914000271

[19] M. Hammoudi, G. Rothermel, and P. Tonella, "Why do Record/Replay Tests of Web Applications Break?" in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2016, pp. 180–190.

[20] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "Robula+: an algorithm for generating robust XPath locators for web testing," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 177–204, 2016. [Online]. Available: https://doi.org/10.1002/smr.1771

[21] R. Atterer and P. Lorenzi, "A Heatmap-based Visualization for Navigation Within Large Web Pages," in *Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges*, ser. NordiCHI '08. New York, NY, USA: ACM, 2008, pp. 407–410, event-place: Lund, Sweden. [Online]. Available: http://doi.acm.org/10.1145/1463160.1463206