Regression Test Generation by Usage Coverage Driven Clustering on User Traces

1st Frédéric Tamagnan
Dept. DISC - FEMTO-ST
Univ. of Franche-Comté
Besançon, France
frederic.tamagnan@gmail.com

2rd Fabrice Bouquet
Dept. DISC - FEMTO-ST
Univ. of Franche-Comté)
Besançon, France
fabrice.bouquet@femto-st.fr

3th Alexandre Vernotte
R&D
Smartesting
Besançon, France
alexandre.vernotte@smartesting.com

4nd Bruno Legeard
Dept. DISC - FEMTO-ST
Univ. of Franche-Comté
Besançon, France
bruno.legeard@femto-st.fr

***Abstract*—Regression testing is used to verify that changes or modifications do not negatively impact existing features or functionality. However, it can be a significant bottleneck due to the challenge of determining the optimal number of tests for adequate coverage and fast execution and the difficulty for Quality Assurance engineers to anticipate end-user behavior. This paper presents a Usage Coverage Driven Clustering (UCDC) approach that regroups user trace logs in clusters of similar behavior, then constitutes the optimal test suite by electing one user trace per cluster as test cases. To find the optimal number of clusters, a novel metric is proposed to evaluate the statistical representativeness of the selected test candidates. Experimentation on three datasets[1] shows that the proposed metric, as compared to generic metrics such as the David-Bouldin index, leads to a smaller number of clusters while ensuring 95% usage coverage by the test suite. Several approaches from the state of the art are combined with UCDC and compared, and two of them outperform the others.**

***Index Terms*—machine learning, clustering algorithms, software testing, test generation**

I. Introduction

Regression Testing is the process of re-running a dedicated set of tests after each code change to ensure that the recent modifications did not affect the existing features of an application. It can be costly in time and resources, as regression tests need to be run after at each iteration of the software development. To minimize the amount of cost related to this task, QA engineers need to 1) automate the Regression Testing process from the creation of the test suite to its execution, 2) find an optimal set of tests: a minimal set of tests that guarantees sufficient coverage of the software under test. Clustering based on unsupervised learning algorithms that groups a set of objects by similarities, has already been used for Test Selection and Prioritization [1]–[6]. Nowadays most IT systems collect activity logs that can be organized in user traces, which represent the behavior of users on those systems. This newly available data can be used for test generation [7], [8]. Indeed, there is a discrepancy between the written tests and the real user paths that the usage data can bridge [9]. For this reason, clustering algorithms have been applied directly to user traces to obtain test cases [10]–[15].

[1]All the experiments and data on Scanner and Spree are available at https://github.com/frederictamagnan/AIST2023UCDC

Clustering segments user traces from the System Under Test (SUT) in groups of similar behavior. The regression test suite is formed by selecting a minimal set of user traces extracted from each cluster and converting the traces to test cases. Nevertheless, the performance of clustering models at disentangling traces is variable depending on their characteristics: size, sequential character, dataset balance, and volume. Thus, the objective is to find, for each use case, the most adapted model to the constraints of the use case's data. But we lack a metric to measure the representativeness of the regression tests generated by these methods with respect to the actual use of the software. Moreover, it is necessary to establish a stopping criterion in the search for the number of clusters and a sampling strategy to choose user traces chosen as candidates to become tests. Although there is confidence in the idea of using clustering to extract test suites from real usage, generic internal metrics [16]–[25] diverge in their evaluation to determine the relevance of the partition and the optimal number of clusters.

A. Research Questions

There are three main research questions addressed in this paper:

- **RQ1: What is the representativeness of a test suite with respect to the user traces?** As it is difficult to obtain a test suite that reflects the real usage of a system, we need to create a metric that represents the disparity between user traces and regression tests to qualify the relevance of a test suite.
- **RQ2: Can clustering techniques help to capture the real usage? And what is the optimal number of clusters to segment the user traces for regression testing purpose?** By selecting test candidates among clusters of user traces, we aspire to get a representative test suite. For this task, it is essential to select the optimal number of clusters. Picking at least one test by cluster, having too many clusters may lead to a test suite with redundant test cases, whereas not having enough clusters may lead to a test suite that insufficiently covers the observed user behavior on the system.
- **RQ3: To what extent the choice of test candidates among clusters is important?** Once clustering is done

979-8-3503-3335-0/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSTW58534.2023.00026

on user traces, we need to elect one or several traces from each cluster as test candidates. Several approaches from the state-of-the-art use random sampling, to what level it is suitable ?

B. Paper Contributions

- A novel usage coverage metric to evaluate if a regression test suite is relevant with respect to user traces of the SUT. This metric is absolute, bounded and does not require to instrument of the code.
- The Usage Coverage Driven Clustering (UCDC) approach for the regression test generation: the metric is used to fine-tune and benchmark 7 Clustering Pipelines (of which three are from the State-of-the-Art) associated with 2 Sampling Strategies. We find their optimal number of clusters and generate a minimal set of tests. Two of them outperform the others, reaching 95% of Usage Coverage with a few tests.
- Experimentation on three datasets from different web applications.
- One dataset was created and is available as public data. The UCDC code is provided in an open repository to allow the reproducibility of results.

The rest of the paper is organized as follows: the use cases and corresponding datasets are introduced in Section II. Section III presents the state-of-the-art related to regression test selection with clustering, test generation by clustering on user traces, and clustering evaluation. In Section IV, the usage coverage metric is defined along with an illustrative example. The UCDC approach (Fig. 1) is detailed in Section V. In section VI-B, the results of seven clustering pipelines and sampling strategies to reach 95% of usage coverage with few tests are presented. Section VII discusses validity biases and limitations of the UCDC approach, and Section VIII concludes and outlines future work.

II. Use cases and Datasets

A. Definition

From a web application perspective, we define a user trace as a sequence of API calls, also called *Events*, that a user has triggered, browsing a web application. An *Event* consists of an API method, a user identifier, several parameters, and a return code. The experiments presented in this paper were performed on two datasets made available as public data (Scanner and Spree) and one dataset from an industrial partner (namely Orange) left as private data (the application is named Teaming).

B. Scanner

A supermarket scanner allows customers to scan the barcodes of the products they put in their cart and to proceed to self-checkout later. Our SUT is the back-end API of a supermarket scanner device. The API is composed of 10 various *Events*. For example:

- `Unlock`: The customer unlocks an available scanner.

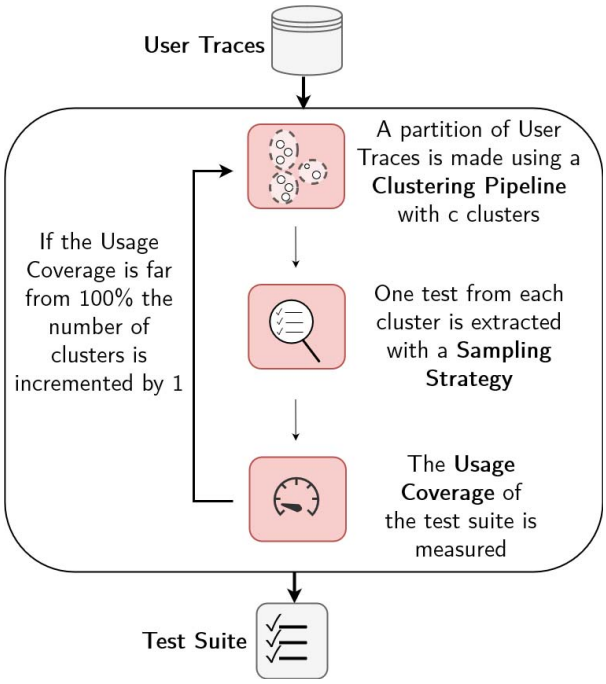

Fig. 1: Overview of the UCDC approach

- `Scan`: The customer scans a product's barcode before putting the product in their physical cart. The barcode is added to the internal list of products maintained by the back-end. When an unknown barcode is scanned, the back-end returns a special return code `-2` and marks the product for later manual processing by a cashier.
- `Delete`: The customer deletes a product from the internal shopping list while putting back the product on the shelf.

A second software, implementing a probabilistic finite state machine, models the behaviors of customers using scanners in a supermarket and allows us to generate user traces artificially. This software is using the API, and the user traces are recorded by the SUT as logs (see Table . I).

This software has been written by a teacher from Univ. of Franche-Comté in Java for a software engineering class. It comes with regression tests written by the teacher.

Timestamp	UserId	API Method	ReturnCode	Param1	Param2
1674724050	client0	Unlock	0	[]	scan0
1674724050	client0	Scan	-2	[35705901109324]	scan0
1674724050	client0	Scan	0	[3017800238592]	scan0
1674724050	client0	Transmission	0	[Cashier3]	scan0
1674724050	client0	Abandon	?	[]	scan0
1674724050	client0	OpenSession	0	[]	Cashier3
1674724050	client0	Add	0	[35705901109324]	Cashier3
1674724050	client0	CloseSession	0	[]	Cashier3
1674724050	client0	Pay	5.5499	[90]	Cashier3

TABLE I: Example of a user trace in the scanner logs

C. Spree: Headless Ecommerce API

Spree is an open-source and modular, headless e-commerce platform for global brands built on a REST API. It allows brands to have a headless ecommerce on which they can plug the customer front-end of their choice. Spree comes with an admin front-end that allows managing stocks, products, orders, and customer accounts.

The API is composed of 56 various methods, grouped by categories, some of them are:

- `Authentication`: Manages the authentication of a user
- `Cart, Line Items`: Allows a guest customer or a logged user to initialize a cart and start to add products to their cart
- `Checkout`: Manages the various check-out steps such as updating the billing, shipments and payment information of the customer
- `Shipments`: Allows retrieving the various shipments proposed and compute the shipping rates.

UserId	Abstract Name	Method	URL	ReturnCode
312458	createACart	POST	api /v2/storefront/cart	201
312458	listAllTaxons	GET	api/v2/storefront/taxons/?per_page=100	200
312458	SortByPrice	GET	api/v2/storefront/products	200
312458	retrieveAProduct	GET	api/v2/storefront/products/long-sleeve-jumper	200

TABLE II: Example of a user trace in the Spree logs

For this use case as well, we develop a second software, implementing a probabilistic finite state machine, modeling the behaviors of customers doing shopping through the API and allowing us to generate customer execution traces artificially (see Fig. II).

D. Teaming

Teaming is a modular solution for IP telephony, conferences, and office automation tools. Teaming has a set of REST API web services whose API is composed of 97 various requests. Orange gave us real customers and test execution traces of this use case.

E. Dataset Summary

- `Scanner`: 7000 generated user traces and 27 regression tests.
- `Spree`: 5000 generated user traces.
- `Teaming`: Orange provided a dataset of 9662 real user traces and 236 regression tests.

III. Related Work

A. Regression Test Selection and Prioritization with Clustering

Clustering has regularly been investigated in the literature to achieve regression test selection and prioritization. Chen et al. introduced the semi-supervised k-means method to perform regression test selection [1]. SSDR is applied to perform preprocessing on the test data in order to obtain vectors that will be passed to a K-means model [26]. The Euclidean distance between two test vectors tends to be large, if the two tests have different function calls profiles and they detect different faults in the version history of the system under test. In Kandil et al. [2], the goal is to select and prioritize the best suitable tests according to the software development level: release or test. They use K-means in one of the stages of their methodology to cluster tests belonging to user stories, according to the module coverage of the user stories. For Almaghairbe et al. [3], clustering models such as Agglomerative Hierarchical Clustering, DBSCAN, and Expectation-maximization (EM) clustering, separate failing from passing test executions based on their textual input and output and their execution traces. Carlson et al. [4] employ Agglomerative Hierarchical Clustering on tests to perform test prioritization with the following assumption: tests belonging to the same clusters have the same fault detection ability. They feed models with code coverage, fault history, and dependency information of each test. Clustering can also be utilized to classify tests according to the business requirement priority in several groups (high, medium and low) [5]. Wang et al. [6] adopted k-means in their prioritization pipeline to group similar tests according to their function calls profile.

B. Test Generation With Clustering Applied on User Traces

Clustering can also be employed to extract knowledge from user traces. By clustering user traces, the main behaviors of users can be extracted, and the most representative set of test cases that cover the most system can be found [10]–[14] or identify missing tests [15]. For this purpose, Luo et al. [10] partition user traces by associating them with the several services of a web application thanks to a distance measure. Then, they choose test candidates among these clusters by browsing a dependency graph of the web pages. Liu et al. [11], use hierarchical clustering to cluster user traces together, delete redundancy and use them as test candidates. The same approach was used by Li et al. [12] with the k-medoids clustering algorithm. Dorcis et al. [13] employ a Sequences String Comparison Clustering inspired by gene sequencing comparison by extracting string patterns from execution traces to select user traces as tests. Afshinpour et al. [14] apply Word2Vec session averaging encoding, K-means and t-SNE, to find the best regression tests among user traces. Finally, Utting et al., use MeanShift algorithm to identify missing regression tests by clustering jointly user traces and test traces and showing which clusters contain no test.

C. Evaluation of Clustering

In order to evaluate clustering models, there are two types of evaluation methods: internal evaluation and external evaluation.

In the recent clustering state-of-the-art, scientists work with external evaluation to evaluate clustering models thanks to its ease to be used to benchmark several models [16], [17]. External evaluation benefits from pre-labeled data to evaluate model performance. For this purpose, Rosenberg et al. [18] introduced a unified measure called v-measure, which is the

	Silhouette Coefficient	Davies-Bouldin Score
Scanner	20	117
Spree	10	123
Teaming	22	100

TABLE III: Optimal number of clusters given by state-of-the-art scores

Session id	Api Call Sequence
d_1	login add
d_2	login add add
d_3	login add delete
d_4	login add add

TABLE IV: List of user traces of the illustrative example

harmonic mean of completeness and homogeneity. Perfect labeling is both homogeneous and complete, hence having a score of 1.0.

Internal evaluation mostly describes if a clustering model produces clusters with high similarity within a cluster and low similarity between elements from different clusters. Internal evaluation is useful to tune the hyperparameters of a model, particularly to adjust the right number of clusters to be chosen. Some of the best known internal evaluation metrics are David-Bouldin Index [20], [21], and the Silhouette coefficient [22]–[25] and help the validation of cluster consistency.

Although internal evaluation metrics appear to us to be the best way to set up the number of clusters of one model, as it is shown in Rendon et al. [19], applying those two scores to the three use cases resulted in various numbers of clusters (Table III) when clustering with K-means and a Bag-of-Words preprocessing. Those variations are not unusual [19] and constitute a problem in doing a clustering approach for regression test selection.

The related works gave us confidence to use clustering as a relevant method to identify regression tests among user traces. Nevertheless, no one in the section III-B solves the problem of the optimal number of clusters to identify with an absolute and bounded metric.

IV. Usage Coverage Metric

A. Goal of the Metric

The main goal is to build a metric that:

- reflects if a regression test suite covers the most representative user flows from a statistical perspective;
- measures the quality of regression testing suites with an absolute (compared to the relativity of generic internal evaluation metrics) and bounded (ranging from 0 to 100) metric
- does not need to instrument the code, to avoid difficulties in implementing the metric because the code of the system under test is not always easily accessible to the QA engineers.

B. A Usage Coverage Metric based on N-grams

1) Definition: The usage coverage metric is based on N-grams. N-grams come from the field of NLP and are contiguous sequences of N items from a given sample of text. For instance, from the sentence "Mathematics is a language" the following unigrams can be extracted: $\{(mathematics), (is), (a), (language)\}$, the following bigrams: $\{(mathematics, is), (is, a), (a, language)\}$ and the following trigrams: $\{(mathematics, is, a), (is, a, language)\}$.

We define the usage coverage metric as the ratio between the number of actual N-grams in the test execution traces with respect to the number of n-grams existing in the user traces. This ratio is weighted by the count of each N-gram present in the user traces.

The N-grams extracted from the user traces represent the sequences of actions on the software performed by the users. In the same way, the N-grams extracted from the test execution traces represent the action sequences tested during the execution of the regression test suite. N-grams allows us to capture the sequential nature of the user traces.

2) Formalizing the Metric: Let T be a test suite, D a set of user traces, and $n \in \mathbb{N}$. Let D_g the list of $1..n$-grams in D, T_g the list of $1..n$-grams in T, and $Count_D(g)$ the number of occurrences of a specific $1..n$-gram g in D_g. Then we define the usage coverage of T with respect to D as the following:

$$UsageCoverage(T, D) = \frac{\sum_{g \in T_g} Count_D(g)}{\sum_{g \in D_g \cap T_g} Count_D(g)} \quad (1)$$

3) An illustrative Example:

- Let a dataset D of 4 user traces d_1, d_2, d_3 and d_4 recorded from a SUT (Table IV).
- The exhaustive list of possible API calls is $login$, add and $delete$
- Let a test suite T composed of 2 tests t_1 and t_2 (Table V)
- user traces or test traces are sequences of API calls and share the same vocabulary.

Computation of the usage coverage of T with respect to D of Equation (1) with $n = 2$:

1) Extract the unigrams and the 2-grams of the user traces data (column 1,2-grams of Table VI).
2) Count the occurrence of each 1,2-gram in the user traces data (column A of Table VI)
3) Identify the presence of each 1,2-gram in the test suite (column B of Table VI)
4) Compute the weighted presence of each 1,2-gram in the test suite with respect to the user traces data (A times B, column A·B of Table VI)
5) Sum the A column and the A·B column separately. Dividing the sum of A·B by the sum of A gives us the usage coverage of the test suite. In this example, the usage coverage of the test suite is 88,2%

Test id	Api Call Sequence
t_1	login add delete
t_2	login login

TABLE V: List of tests of the illustrative example

1,2-grams	A	B	A·B
	# in user traces	In Test Suite ?	
(login)	4	True	4
(add)	5	True	5
(delete)	1	True	1
(login,add)	4	True	4
(add,add)	2	False	0
(add,delete)	1	True	1
sum	17	-	15

TABLE VI: List of 1,2-grams of the illustrative example and their counts in the user traces (US), their presence in the test suite (TS), and their weighted presence in the test suite (WP)

V. Usage coverage-driven Clustering

The goal of our approach UCDC is to find a set of tests among the user traces that allow us to approach 100% of usage coverage with a ϵ tolerance threshold. UCDC uses a clustering pipeline and a sampling strategy to choose test candidates among clusters.

A. Clustering Pipeline

The clustering pipeline is composed of a preprocessing stage and a clustering algorithm. The preprocessing stage encodes user traces into a format that allows them to be fed to a clustering algorithm. The clustering algorithm used in UCDC uses a hyperparameter that controls the number of clusters. It takes as input the user traces and a number of clusters and returns a label for each user trace.

B. Sampling Strategy

The sampling strategy takes as input the user traces and their labels. It returns as an output a test suite by picking one test per cluster of user traces.

C. Details

Once a clustering pipeline and a sampling strategy are chosen:

1) A clustering pipeline object with c clusters is instantiated (**ClusteringPip(c)**)
2) A clustering model is trained on the set of user traces with a number of clusters equal to c and we predict labels for each trace (**cp.fit_predict(D)**)
3) A test is extracted for each cluster thanks to the sampling strategy (**SamplingStrat.choose(labels,D**) to constitute a test set of c tests.

Algorithm 1 Usage Coverage-Driven Clustering

Input: ϵ : precision threshold, D : user traces Data, ClusteringPip, SamplingStrat, r : nb of repetitions
Ouput: Optimal number of cluster, labels of clusters
$\hat{u} \leftarrow 0$
$c \leftarrow 1$
while $1 - \hat{u} > \epsilon$ **do**
  $c \leftarrow c + 1$
  for $k = 1$ to r **do**
    cp=ClusteringPip(c)
    labels=cp.fit_predict(D)
    T=SamplingStrat.choose(labels,D)
    u_k=compute_usage_coverage(T,D)
  end for
  compute the mean $\hat{u}$ of $u_1, u_2, ..u_r$
end while
return c, labels

4) The usage coverage of the test suite is computed with respect to the reference usage traces (**compute_usage_coverage(T,D)**)
5) These 4 operations are repeated r times to have an average coverage usage of a test suite extracted thanks to this clustering model with c clusters. The cost functions of the clustering models can converge to local minima, which is why a robust mean estimate of the coverage usage is needed.
6) If the average usage coverage is close enough to 100% (at a threshold of ϵ), then the algorithm is stopped. Otherwise, the number of clusters is increased by 1 and the loop starts again.

VI. Experiments

A. Clustering Pipelines and Sampling Strategies

We explored various clustering pipelines and sampling strategies.

1) BoW + K-means / Random Sampling: The first step of the clustering pipeline is a preprocessing stage consisting of a bag-of-words encoding of the user traces (sequences of *Events*). Each *Event* is the concatenation of the API method and the return code i.e. for the scanner dataset, an *Event* can be $scan\_2$ (see Fig. I). We use a Standard Scaler preprocessor over the vectors obtained. Then K-means is used as the clustering algorithm. *Random Sampling* (RS) is chosen for the sampling strategy: **one** user trace is randomly sampled from each cluster.

2) BoW + K-means / Best Usage Choice: The same preprocessing and clustering stage is used as previously (Section VI-A1). For the sampling strategy, for each cluster, the user trace that represents the best usage coverage with respect to the N-grams of the cluster is selected. All the N-grams found in the cluster are sorted by descending frequency, the algorithm iterates on it and keeps the traces that contain all the N-grams that have been browsed during iteration until there

are no more traces corresponding to the next iteration. This is detailed in Algorithm 2: *Best Usage Choice* (BUC).

Algorithm 2 BestUsageChoice(BUC)

```
1: Input: D : user traces composing a specific cluster
2: Ouput: The test with the best usage
3: Sort the ngrams in D by descending frequency
4: ngramIndex ← 0
5: while len(D) > 1 and len(ngrams) > ngramIndex do
6:     for trace in D do
7:         if ngram not in trace and len(D) > 1 then
8:             D.remove(trace)
9:         end if
10:    end for
11:    ngramIndex ← ngramIndex+1
12: end while
13: return D[0]
```

3) Baseline: No Clustering/Random Sampling: The baseline does not use clustering. It follows algorithm 1 but without the clustering stage (lines 8-9). So in each iteration of the `for` loop, it samples tests randomly among all user traces, to obtain c tests.

4) W2V+ K-means/Random Sampling: Reproducing [14], word2vec averaging is used to encode the user traces and K-means to cluster. *Random Sampling* is taken to extract tests from user trace clusters.

5) W2V+ K-means/BestUsageChoice: It is the same as the previous clustering pipeline but *Best Usage Choice* (section VI-A2) is taken as a sampling strategy to extract tests from user traces clusters.

6) K-medoids/Random Sampling: Reproducing [12], we use K-Medoids as a clustering algorithm based on their similarity measure. *Random Sampling* is taken to extract tests.

7) Agglutinate Hierarchy Clustering /Random Sampling: Reproducing [11], Agglutinate Hierarchy Clustering (AHC) is used as a clustering algorithm based on their similarity measure. A *Random Sampling* strategy extracts the tests .

8) Experimental Setup: We ran the experiments on a cloud instance with 6 vCPU cores and 16GB of RAM. It took us a total of 85 hours. We used scikit-learn [27] for the K-means and Agglutinate Hierarchy Clustering implementation, Gensim [28] for Word2vec and PyClustering [29] for K-medoids. We set ϵ to 5% and r to 50. We made several adaptations for [11], [12], [14] to apply them to our use cases. For [14], we skipped their t-SNE stage for time constraints. We adapted the similarity measure of [11], [12] by taking into account only API calls names and return codes instead of all parameters as it was more relevant in our case. For [11], we use a *Random Sampling* strategy instead of theirs, as their sampling strategy was extracting more than one test by cluster. We chose 1-4 N-grams, as it was a sufficient length of N-grams to show differences in performance between the various clustering pipelines and sampling strategies (only unigram would allow all the clustering pipelines and sampling strategies to converge to 100%).

Clustering Pip.	Sampling.	Scanner	Spree	Teaming	Time (hours)
No Clustering	RS	11	12	-	0.03
K-medoids	RS	8	9	-	64.30
AHC	RS	11	7	-	1.92
BoW+K-means	RS	7	6	92	6.05
W2V+K-means	RS	8	7	77	6.09
BoW+K-means	BUC	**6**	4	87	6.85
W2V+K-means	BUC	8	**3**	**72**	6.89

TABLE VII: Optimal number of clusters/tests by Clustering Pipeline/Sampling Strategy given by UCDC

B. Results

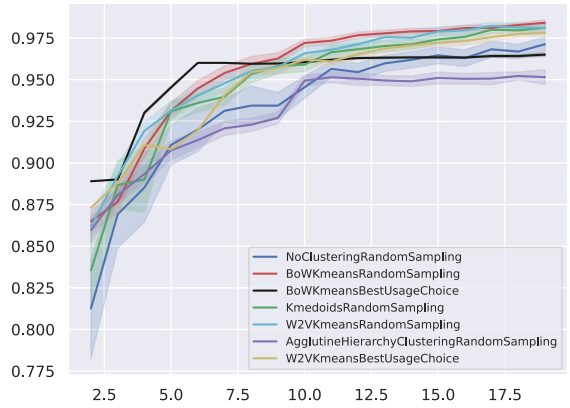

Fig. 2: Scanner: usage coverage w.r.t. the length of the test suite (with 95% confidence interval)

Iterations of UCDC are plotted for each approach in the figures 2, 3 and 4 and summarized the results in Table VII. On Scanner and Spree, which are relatively simple use cases, all the approaches allowed extracting a test suite approaching 100% of usage coverage. The approach *BoW+K-means/BUC* gives the best results with 6 tests for Scanner. For Spree, *W2V+K-means/BUC* gives the best results with 4 tests. On Teaming, which is more complex and is a real use case, three of the approaches do not converge and their test suite stays below 80% of usage coverage. *W2V+K-means/BUC* has the best result again on this dataset, approaching 100% of usage coverage with 72 tests.

Moreover, we show that the *Best Usage Choice* sampling strategy is better than *Random Sampling* as *Bow+K-means/BUC* and *W2V+K-means/BUC* perform better than the corresponding clustering pipelines with *Random Sampling*, on the three use cases (equally for *Bow+K-means* on Scanner). The convergence of *Bow+K-means/BUC* is also better than *Bow+K-means/RS* on the teaming use case, *Bow+K-means/BUC* reaching 80% of usage coverage with 60 tests while *Bow+K-means/RS* is below 50%. While in NLP, where it is easy

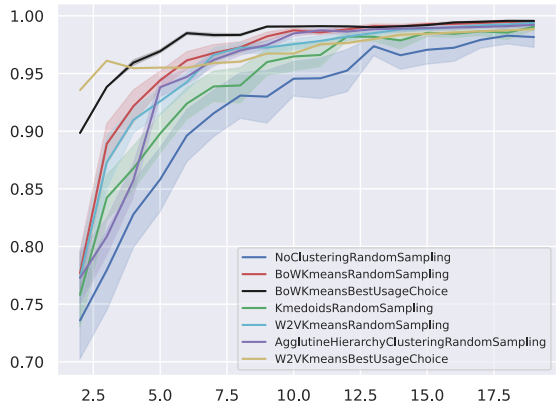

Fig. 3: Spree: usage coverage w.r.t. the length of the test suite (with 95% confidence interval)

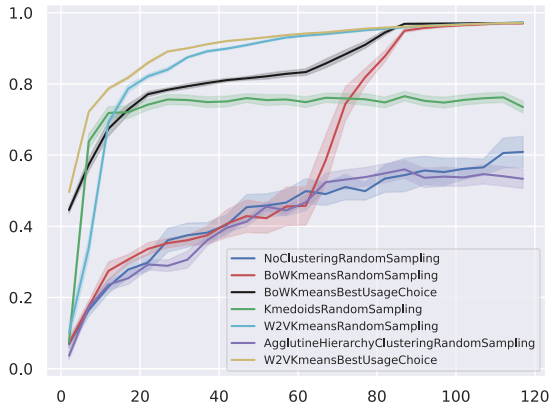

Fig. 4: Teaming: usage coverage w.r.t. the length of the test suite (with 95% confidence interval)

to demonstrate whether a clustering model is the best, since the input data is always the same, it is difficult to determine whether one model will beat all the others with user uraces, since the data may be very different from one system to another. Nevertheless, we have empirically shown asymptotic behaviors of usage coverage as a function of the number of clusters with the UCDC approach, which facilitates benchmarking of models for the purpose of generating regression tests.

C. Comparison with Human-Written Tests

We analyzed the two sets of tests provided with Scanner and Teaming written by humans (respectively a developer, and a Q&A engineers team). There is no test for Spree.
For Scanner, the usage coverage of the global test suite of 27 tests written by the developer is 87%. We beat this coverage (95%) with UCDC coupled with *Bow+K-means/BUC* with only 6 tests. For Teaming, the usage coverage of the 236 tests is 99%, we reach a similar coverage with *W2V+K-means/BUC* with 72 tests.

VII. Validation and Biases

A. Answers to Research Questions

We conducted an experimental validation of our approach at three levels: on a fully controlled application (Scanner example), on a large open-source eCommerce software with artificial users (Spree), and an industry case study with real users (Teaming). We have translated the coverage of the main user paths to a statistical metric that varies from 0 to 100%, indicating that a test suite is good with respect to the user traces when reaching 100%. That allows us to answer RQ1. We used that metric to fine-tune clustering pipelines. The asymptotic character of the usage coverage during the iteration of UCDC allows capturing the optimal point where a sufficient number of clusters is found. Coupled with UCDC, Clustering ensures 95% of usage coverage, mimicking the main (weighted by counts) paths of real users. That answers RQ2. We have shown that the sampling strategy can also be crucial in choosing representatives. Out of 6 experiments (2 clustering pipelines: *Bow+K-means* and *W2V+K-means*, and three datasets) *Best Usage Choice* sampling strategy has given better results in 5 cases than Random Sampling. The results are equal in one case. In those 6 experiments, *Best Usage Choice* has allowed obtaining in average 19,3 % less number of tests in the test suites than *Random Sampling*, answering to RQ3.

B. Discussion of Biases

There are a few biases to our study that must be acknowledged:

- The experiments were conducted within a limited scope since just three use cases were addressed, two of which were composed of artificial user traces. Nonetheless, the industrial use case is highly representative of the testing environment one might find in most software development companies. The artificial use cases were crafted with realism in mind, and their synthetic nature allowed for controlled experimentation.
- UCDC has been operated with a small selection of clustering pipelines and sampling strategies. We carefully selected each combo: our choices were informed by prior literature and expert knowledge in the field, ensuring that they were appropriate for the scope of our analysis. Based on the literature and to the best of our knowledge, we believe we have included the most relevant opposing approaches.
- The experimentation can be extended with a comparison of the N-grams coverage approaches and real detection of faults. The absence of test execution metrics such as the proportion of detected faults or mutation score indicator to cross-check the obtained results. However, the results are supported by qualitative analysis and these metrics

could be used in future studies to further validate our findings.

VIII. Conclusions and Future Work

This paper presents an approach to generate regression tests by clustering user traces using an usage representativeness metric. The complete approach is named Usage Coverage Driven Clustering (UCDC). We used it to benchmark, evaluate and fine-tune several clustering pipelines and sampling strategies for a regression test generation purpose. We showed that clustering algorithms could have various performance results on different sets of traces to extract an optimal test suite that reaches 95% of usage. Experiments are fully reproducible through two public datasets and open-source code. In the future, we want to investigate further clustering algorithms, such as those from the recent NLP advance using transformers. In addition, we plan to apply the UCDC approach to more real use cases and datasets. Moreover, we would like to look over the discrepancy between the UCDC-generated test suites and the human-written test suites. Additionally, we intent to build new and more robust sampling strategies. We will explore as well the influence of the length of N-grams on the computation of usage coverage. Finally, we would also like to extend test suites diversity by a generation approach to generate non-existing, but plausible new user flows.

IX. Acknowledgement

This work was supported in part by the French National Research Agency: PHILAE project (N° ANR-18-CE25-0013).

References

[1] S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng, "Using semi-supervised clustering to improve regression test selection techniques," in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, 2011, pp. 1–10.

[2] P. Kandil, S. Moussa, and N. Badr, "Cluster-based test cases prioritization and selection technique for agile regression testing," *Journal of Software: Evolution and Process*, vol. 29, 07 2016.

[3] R. Almaghairbe and M. Roper, "Separating passing and failing test executions by clustering anomalies," *Software Quality Journal*, 2017.

[4] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," in *27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 382–391.

[5] Z. Khalid and U. Qamar, "Weight and cluster based test case prioritization technique," in *IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2019, pp. 1013–1022.

[6] Y. Wang, Z. Chen, Y. Feng, B. Luo, and Y. Yang, "Using weighted attributes to improve cluster test selection," in *IEEE Sixth International Conference on Software Security and Reliability*, 2012, pp. 138–146.

[7] D. Tiwari, L. Zhang, M. Monperrus, and B. Baudry, "Production monitoring to improve test suites," *IEEE Transactions on Reliability*, 2021.

[8] S. Thummalapenta, J. De Halleux, N. Tillmann, and S. Wadsworth, "Dygen: Automatic generation of high-coverage tests via mining gigabytes of dynamic traces," in *International Conference on Tests and Proofs*. Springer, 2010, pp. 77–93.

[9] X. Blanc, T. Degueule, and J.-R. Falleri, "Diffing e2e tests against user traces for continuous improvement," 2022.

[10] X. Luo, F. Ping, and M.-H. Chen, "Clustering and tailoring user session data for testing web applications," in *2009 International Conference on Software Testing Verification and Validation*, 2009, pp. 336–345.

[11] Y. Liu, K. Wang, W. Wei, B. Zhang, and H. Zhong, "User-session-based test cases optimization method based on agglutinate hierarchy clustering," in *ITHINGSCPSCOM'11, International Conference on Internet of Things*, 10 2011, pp. 413–418.

[12] J.-h. Li and D.-d. Xing, "User session data based web applications test with cluster analysis," in *CSIE 2011, International Conference on Computer Science and Information Engineering*, vol. 152, 01 2011, pp. 415–421.

[13] V. Dorcis, F. Bouquet, and F. Dadeau, "Clustering of usage traces for regression test cases selection," in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2022, pp. 138–145.

[14] B. Afshinpour, R. Groz, M.-R. Amini, Y. Ledru, and C. Oriat, "Reducing regression test suites using the word2vec natural language processing tool." in *SEED/NLPaSE@ APSEC*, 2020, pp. 43–53.

[15] M. Utting, B. Legeard, F. Dadeau, F. Tamagnan, and F. Bouquet, "Identifying and generating missing tests using machine learning on execution traces," in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 2020, pp. 83–90.

[16] J. Xu, P. Wang, G. Tian, B. Xu, J. Zhao, F. Wang, and H. Hao, "Short text clustering via convolutional neural networks," in *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*. Denver, Colorado: Association for Computational Linguistics, Jun. 2015, pp. 62–69. [Online]. Available: https://aclanthology.org/W15-1509

[17] J. Xu, B. Xu, P. Wang, S. Zheng, G. Tian, J. Zhao, and B. Xu, "Self-taught convolutional neural networks for short text clustering," *Neural Networks*, vol. 88, pp. 22–31, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608016301976

[18] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 410–420.

[19] E. Rendón, I. Abundez, A. Arizmendi, and E. M. Quiroz, "Internal versus external cluster validation indexes," *International Journal of computers and communications*, vol. 5, no. 1, pp. 27–34, 2011.

[20] A. K. Singh, S. Mittal, P. Malhotra, and Y. V. Srivastava, "Clustering evaluation by davies-bouldin index (dbi) in cereal data using k-means," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2020, pp. 306–310.

[21] S. M. S. Hosseini, A. Maleki, and M. R. Gholamian, "Cluster analysis using data mining approach to develop crm methodology to assess the customer loyalty," *Expert Systems with Applications*, vol. 37, no. 7, pp. 5259–5264, 2010.

[22] D.-T. Dinh, T. Fujinami, and V.-N. Huynh, "Estimating the optimal number of clusters in categorical data clustering by silhouette coefficient," in *Knowledge and Systems Sciences*, J. Chen, V. N. Huynh, G.-N. Nguyen, and X. Tang, Eds. Singapore: Springer Singapore, 2019, pp. 1–17.

[23] H. B. Zhou and J. T. Gao, "Automatic method for determining cluster number based on silhouette coefficient," in *Advanced materials research*, vol. 951. Trans Tech Publ, 2014, pp. 227–230.

[24] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2020, pp. 747–748.

[25] A. M. Bagirov, R. M. Aliguliyev, and N. Sultanova, "Finding compact and well-separated clusters: Clustering using silhouette coefficients," *Pattern Recognition*, vol. 135, p. 109144, 2023.

[26] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[29] A. Novikov, "Pyclustering: Data mining library," *Journal of Open Source Software*, vol. 4, no. 36, p. 1230, apr 2019. [Online]. Available: https://doi.org/10.21105/joss.01230