NLP-Based Manual Test Case Selection

Marius Bosler

Seminar: Software Quality
Advisor: Roland Würsching
Technical University of Munich
marius.bosler@tum.de

Abstract. Even though there have been many advancements in recent years, a significant portion of testing still needs to be conducted manually. This means that with each software release requiring numerous manual tests, a considerable amount of time is spent executing these tests. Meanwhile, many of these tests tend to overlap regarding bug detection, which could be avoided if a method were found to execute only the necessary number of tests, as multiple tests may detect the same bug. One solution is to cluster tests based on the language used in their descriptions. Then, by carefully selecting only a subset of all tests based on this clustering, a lot of time can be saved, especially if the selected tests cover a significant portion of the detected bugs.

Keywords: Test Selection, Natural Language Processing, Clustering

1 Introduction

In the evolution of software testing, a key goal has always been to make testing faster and more reliable. And even though much can be automated nowadays, it is oftentimes still preferable to use manual instead of automated tests, as they are comparatively easy to create and change. One problem with manual testing, however, is that it usually takes a lot of time and human effort to run the tests. Additionally, tests might often have an overlap with each other, because, among other things, for larger projects many test case maintainers do not know every other test case by heart and therefore might introduce redundancy which is of course especially to be avoided in the case of manual tests that take a long time to execute.

Our approach aims to mitigate this redundancy by examining the relationship between test cases. For instance, if two tests A and B have similar textual descriptions, it might be that they cover similar functionalities and therefore potentially detect the same bugs, so in this case, we might want to only execute Test A and thereby reduce our test execution time significantly, even though we still find both bugs.

A lot of ways to find similar tests like this have been proposed, mostly by analyzing the textual descriptions of these tests, like in the study by Viggiato et al. [3]. In that study, Natural Language Processing (NLP) was used on the test steps to find similar test cases, then clustering said test cases to find similar ones.

Recognizing that many tests redundantly identify the same bugs, this study aims to build on the mentioned approach in that study, to later select only a subset of said tests based on their similarity and thereby hopefully reduce the test cases that need to be executed drastically, while still covering a significant portion of recognized bugs.

In our case, we will use a dataset from XWiki, an open-source wiki platform with 1,600 manual tests. If one of these tests fails during execution, the testers append a Bug Ticket to the test. We will use these Bug tickets to discover and evaluate said redundancies.

The entire approach was implemented in Python because of its ease of use and because it has one of the best library support in the fields we look at, especially in the field of NLP.

2 Related Work

To solve the problem of test selection, a lot of approaches have been proposed. Of particular interest to us is test case selection using NLP. In that field, the work by Sutar et al. [2] stands out prominently. Sutar et al. proposed a method to select regression test cases based on their relevance to defects using NLP. In their approach, they focus on regression testing, analyzing both software requirements and test case descriptions using NLP. By calculating the similarity between these, they prioritize those test cases that are most likely to be impacted by changes to the code. However, in this paper, we want to focus not only on regression test case selection but on general test case selection.

Another influential approach for this paper is the one proposed by Li et al. [1]. The researchers use NLP to group semantically similar test steps across different test cases and then cluster these steps to reveal which ones likely perform the same or similar actions. This allows for the avoidance of duplicate or overlapping API methods when the tests are automated. Although the goal of that paper is not the same as ours, they used a similar approach that this paper will explore.

The fundamental approach to finding test similarity in this paper is mostly based on the paper by Viggiato et al. [3]. The researchers use NLP to identify similar test steps within test cases. Text embedding models then convert the text of test steps into numerical representations. These textual similarities are used to determine how closely related the tests with these embedded steps are, followed by clustering to group similar test cases. This step-level analysis aims to reduce redundancy within the overall test suite.

This paper aims to combine the mentioned NLP approaches and introduce real-life bug data to improve test case selection.

3 Methodology

3.1 Data Collection and Description

We used data from XWiki, an open-source wiki platform. From their website, we scraped a dataset of 1,600 manual test cases and 52,860 test executions. Each test case is structured with a description, title, steps to reproduce, and expected results. For instance, a typical test case might look like this:

```
"https://test.xwiki.org/xwiki/bin/view/File%20Manager%20Tests/Delete%20a%20file": {
    "title": "Delete a file",
    "steps_to_reproduce": [
        "Click on File Manager from Applications Panel",
        "Click on All Files",
        "Select a file",
        "Click on Delete",
        "Click on OK"
    ],
    "expected_result": [
        "The file is deleted."
    ]
}
```

The dataset chosen for this study is especially fitting for us because it contains detailed textual descriptions in the form of test steps. These descriptions can help determine the similarity between tests, which we can use to cluster them together.

3.2 NLP Techniques and Test Selection Process

To find similar test cases, we use a mix of text embedding and clustering. The approach to finding similar test cases is based on the approach by Viggiato et al. [3].

First, we converted all words to lowercase and used word stemming, with the python library *nltk*. For example, this converts the sentence *Click on File Manager from Applications Panel* to important word stems: *click file manag applic panel*.
This is done to make text analysis more efficient, especially if we only want to extract the meaning and similarity of test steps.

We then converted the resulting strings to their embeddings with the Python library *SentenceTransformer* and the model *all-mpnet-base-v2*. Embeddings can be thought of as representations of textual information in a numerical format. This is important because they enable us to efficiently compare similarities between different elements mathematically. Every sentence is now represented as a vector of numbers with a length of 768:

```
[-8.30066670e-03, -4.04462852e-02, -2.05343179e-02,  3.46409194e-02,
⋮
1.46980640e-02,  3.09091341e-02,  3.18837278e-02,  3.14172916e-02]
```

To then find similarities between test cases, we needed to find a way to find out which tests have similar test steps. This is done in the following way: First, we are clustering the embeddings of all test steps, so every test case has several clusters whose test steps are in. As a simplified example, we might sort them into 10 clusters, then our example test case steps might be in the clusters with ids *4, 2, 2, 6, 7* (the results are better if we use much more than 10 clusters). For the actual clustering of test cases, we've employed three distinct algorithms: KMeans, DBScan, and Optics. After some evaluation, it became clear that KMeans was the most fitting choice for our specific needs, as we later want to choose the n dissimilar test cases and KMeans lets us define how many test clusters we want. Then we need to find similarities between actual test cases: For every test case, we create an empty null vector with the length of the number of different clusters. Then we fill it at the positions of said cluster IDs with how many each cluster appears:

```
[0, 0, 2, 0, 1, 0, 1, 1, 0, 0]
```

These Vectors now represent our tests in a mathematical way, which, as mentioned earlier, we want to compare the tests with other tests. To achieve this, we cluster again after these vectors and as we want to select n dissimilar test cases, we cluster with KMeans to cluster the tests into n clusters.

After that, we have n clusters of test cases from which we then randomly select one test each.

4 Evaluation

4.1 Research Questions

To test our approach, we want to use our dataset of XWiki to answer the following Research Questions (RQs):

- RQ1: *How many bug tickets are found when selecting only a subset of tests using NLP-Clustering by Test Specification* The goal of this paper is to discover whether a substantial amount of time and test executions can be saved when only selecting a portion of all tests when choosing the NLP-based approach

– RQ2: *How does the performance of NLP-based test selection compare with simpler, more naive methods of test selection in terms of bug detection?* This is done to show whether it is even worth it to implement something comparatively complex over something simple.

4.2 Evaluation Metrics

To evaluate how well our algorithm works, we want to find out how many bugs are covered by a certain percentage of run tests. Our source also includes bug tickets and we can link each test to its bug ticket. That way, we can go through all n selected tests, create a set of bug tickets covered, divide its size by the number of all bug tickets and thereby know the percentage of bug tickets covered. As we do not yet know how many tests should be selected, we will go through all the percentage numbers from 10 to 70 in steps of ten and see how many bug tickets have been covered by each number of tests and could then select the one appropriate for a given project.

4.3 Comparison of Approaches

To then find out how our algorithm compares to naive approaches we first have to choose a naive approach. The first proposition that comes to mind is just choosing n random tests from our test cases. Our alternative approach is only possible with data that has some sort of categorization. As an example, the test at

`https://test.xwiki.org/xwiki/bin/view/File%20Manager%20Tests/Delete%20a%20file`

we mentioned earlier, is in the category *P1.Extensions - File Manager Tests*. In total, there are 68 categories like this in our data set. As we can assume similar bugs belong to similar test categories, this implementation might lead to a well enough grouping of test cases.

4.4 Results

In the results, we chose to only select up to 70% of tests, as clustering with KMeans did not provide enough separate clusters for 80% or more of the test cases for our method, as our approach might lead to fewer distinct test-case vectors than actual test-cases. To avoid departing from the relatively simple and, as the results show, comparatively effective method, we decided to only select up to 70% of tests. This limitation isn't problematic in our case, as the focus is to especially show the effectiveness of the approach for a more significant test case reduction than just 20%.

We compared the three methods of test selection to evaluate their effectiveness in covering bug tickets. These methods are represented in Table 1 as:

– Method 1, selecting test cases randomly from the entire dataset
– Method 2, selecting test cases randomly from each test category
– Method 3, being our presented approach with selecting tests by clustering test steps.

Table 1. Coverage of Bug Tickets by Percentage of Tests Run for Different Methods

Percentage of Tests Run	Method 1	Method 2	Method 3
10%	14.22%	15.14%	18.81%
20%	26.15%	27.75%	36.24%
30%	37.84%	38.53%	51.61%
40%	47.25%	47.94%	59.86%
50%	52.98%	54.82%	72.48%
60%	69.95%	68.35%	80.96%
70%	73.62%	70.87%	83.03%

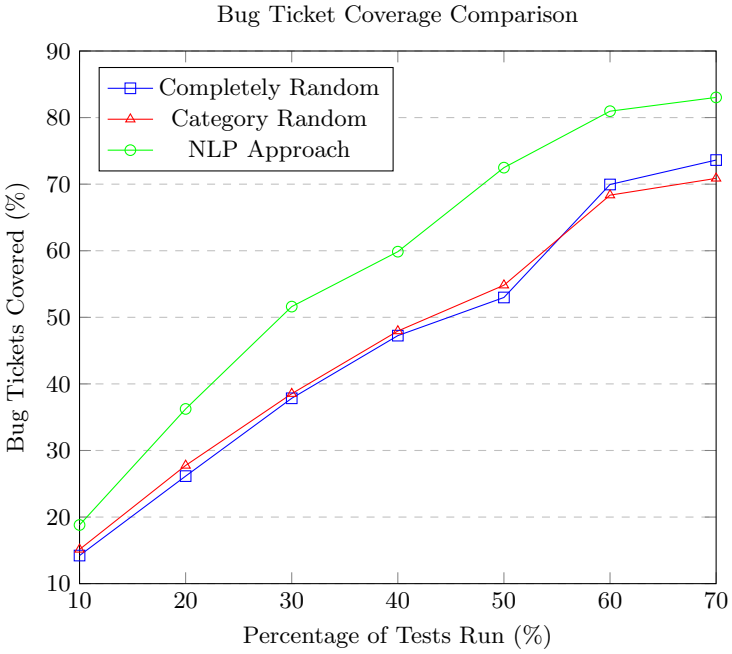

Fig. 1. Comparison of different test selection approaches in terms of bug tickets covered.

Fig. 1 shows a significant difference between more primitive approaches and the more advanced NLP-based approach.

5 Discussion

This section shall discuss the results and the research questions to analyze the effectiveness of our approach.

RQ1: *How many bug tickets are found when selecting only a subset of tests using NLP-Clustering by Test Specification*

Table 1 shows that a NLP-based selection of 30% of the tests achieves a bug detection quota of 51.61%, while 50% results in a quota of 72.48%. This means we can still detect nearly $\frac{3}{4}$ of all bugs when only executing $\frac{1}{2}$ of all tests. This is a much bigger portion of bugs covered than tests executed and therefore a relatively satisfactory result and at the same time, we are sadly missing out on a lot of bugs we used to discover. However, a large project such as the XWiki project or other even larger ones might often have problems finding enough people to execute the tests on every new version, especially when the number of manual tests keeps rising. On smaller versions, it might therefore be preferable to only choose as many test cases as can be executed in a realistic amount of time until release. So, if a project has a problem with manual testers, this approach achieves a reasonable tradeoff between time spent and bugs found.

RQ2: *How does the performance of NLP-based test selection compare with simpler, more naive methods of test selection in terms of bug detection?*

Table 1 shows that a random selection of 30% of tests results in a bug detection quota of 37.84%. This overshoot occurs because most bug tickets usually are associated with multiple test cases. As a result, a test case might cover a higher percentage of bug tickets than its proportion within the test suite would suggest.

The next naive approach was to select based on categories. Selecting 30% of test cases from separate categories first showed a bug detection quota of 38.53%. This result was somewhat surprising because one might expect a significantly better result than just choosing tests randomly. After all, the categories show an intrinsic test similarity based on human judgment. That this did not significantly improve bug coverage showed us how difficult it might be to improve upon random selection.

All in all, the proposed approach to selection gave us a significantly better selection compared to more naive approaches. Given the fact that executing the algorithms on a system with an intel i7-12850HX with 24 cores only takes about 3.5 minutes, we can also recognize a drastic reduction in time spent executing manual tests, which can also lead to a faster release of new versions.

6 Threats to Validity

External:

The dataset of XWiki might not represent all manual test suites equally. It is possible that the suite was a lucky choice for our approach and other datasets might not have the same level of improvements over random selection. If the XWiki dataset has bug distributions significantly different from what is commonly seen in software projects (e.g., very few bugs concentrated in a small portion of the system), it might limit the generalizability of the findings. Also, if the quality of test case descriptions varies significantly (level of detail, clarity etc.), it could weaken the quality of the NLP analysis and clustering.

Internal:

There might be some bugs that are more important or take more time than others. If the NLP-based approach only selects dissimilar tests, we might have a bias on not executing a set of more important test cases if they are too similar to each other and thereby might not find a set of more important bugs. While textual similarity of test case steps leads to an improvement of bug coverage in our case, it is not guaranteed that it actually effectively identifies test similarity all the time. Not every relation between tests might be covered by textual similarity and there might even be datasets where the textual similarity does not at all correlate with similar code that is tested in the background. This could lead to drastically worse results on other datasets.

Mitigation

To avoid these threats one might want to introduce bigger datasets and also look at the severity of bugs. On top of that, manually identifying similar tests might give an idea of how good NLP-based similarity works to find similar tests compared to other approaches and might display inadequacies of this approach. Also, including other parts of the test case descriptions and a different way to cluster the embeddings could improve bug coverage even more and make the NLP-based approach work on more datasets that might not work with our exact approach.

7 Conclusion

The results of this paper show that using NLP techniques to cluster test cases based on textual similarity offers notable improvements in bug detection efficiency compared to simpler test selection methods. Selecting even 30% of tests using our approach revealed a substantially improved bug detection rate compared to just selecting random tests.

As human resources might become a problem for rapidly growing projects that also have a growing number of manual tests to be performed, especially in the open-source world, our method can achieve a favorable tradeoff between time spent testing and bugs covered.

Furthermore, our work supports the findings of Viggiato et al. [3], indicating that textual similarity of test steps provides a useful indicator of overall test case similarity, at least within our dataset.

References

1. Li, L., Li, Z., Zhang, W., Zhou, J., Wang, P., Wu, J., He, G., Zeng, X., Deng, Y., Xie, T.: Clustering test steps in natural language toward automating test automation. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 1285–1295. ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3368089.3417067, https://doi.org/10.1145/3368089.3417067
2. Sutar, S., Kumar, R., Pai, S., BR, S.: Regression test cases selection using natural language processing. In: 2020 International Conference on Intelligent Engineering and Management (ICIEM). pp. 301–305 (2020). https://doi.org/10.1109/ICIEM48762.2020.9160225
3. Viggiato, M., Paas, D., Buzon, C., Bezemer, C.P.: Identifying similar test cases that are specified in natural language. IEEE Transactions on Software Engineering **49**(3), 1027–1043 (2023). https://doi.org/10.1109/TSE.2022.3170272