Identifying Similar Test Cases That Are Specified in Natural Language

Markos Viggiato, Dale Paas, Chris Buzon, Cor-Paul Bezemer

Abstract—Software testing is still a manual process in many industries, despite the recent improvements in automated testing techniques. As a result, test cases (which consist of one or more test steps that need to be executed manually by the tester) are often specified in natural language by different employees and many redundant test cases might exist in the test suite. This increases the (already high) cost of test execution. Manually identifying similar test cases is a time-consuming and error-prone task. Therefore, in this paper, we propose an unsupervised approach to identify similar test cases. Our approach uses a combination of text embedding, text similarity and clustering techniques to identify similar test cases. We evaluate five different text embedding techniques, two text similarity metrics, and two clustering techniques to cluster similar test steps and three techniques to identify similar test cases from the test step clusters. Through an evaluation in an industrial setting, we showed that our approach achieves a high performance to cluster test steps (an F-score of 87.39%) and identify similar test cases (an F-score of 83.47%). Furthermore, a validation with developers indicates several different practical usages of our approach (such as identifying redundant test cases), which help to reduce the testing manual effort and time.

Index Terms—Software testing, Test case similarity, Clustering.

1 Introduction

DESPITE the many recent improvements in automated software testing, testing is still a manual process in many industries. For example, in the gaming industry, game developers face several challenges and difficulties with writing automated tests [37, 39, 42]. As a result, test cases are often described in natural language and consist of a sequence of one or more test steps, which have instructions that must be manually performed to test the target game. Furthermore, those test cases are usually defined by employees from different departments, such as Quality Assurance (QA) engineers or developers, which may result in redundant test cases (i.e., test cases that are semantically similar or even duplicates) as the system evolves and the test suite grows [45]. Having redundant test cases is problematic in particular in a manual testing scenario, due to the tediousness and cost of executing such manual tests.

Manually identifying similar or duplicate test cases to reduce test redundancy is an expensive and time-consuming task. In addition, naive approaches (e.g., searching for exactly matching test cases) are not sufficient to capture all similarity, as different test case writers may use different terminology to specify a test case, even for similar test objectives. Approaches proposed by prior work [7, 28, 52] have limitations in terms of scope (e.g., the work by Li et al. [28] can only cluster test steps but not entire test cases), the large manual effort necessary to specify formal descriptions of test cases [52], or the need for the test case source code [7]. Therefore, an automated and unsupervised technique to identify similar test cases (which can be applied directly to the natural language description of entire test cases) is necessary as it can prevent the QA and development teams from wastefully executing test cases that perform the same task. Throughout this paper, for brevity we adopt the term "similar test cases" to refer to semantically similar and duplicate test cases.

In this paper, we propose an approach to identify similar test cases that are specified in natural language. More specifically, (1) we use text embedding, text similarity, and clustering techniques to cluster similar test steps that compose test cases and (2) we compare test cases based on their similarity in terms of steps that belong to the same cluster.

In the first part of the study, we study how text embeddings obtained from different techniques, text similarity metrics, and different clustering algorithms can be leveraged to identify semantically similar test steps. We compare embeddings from five different techniques (Word2Vec, BERT, Sentence-BERT, Universal Sentence Encoder, and TF-IDF), two similarity metrics (Word Mover's Distance and cosine similarity), and evaluate two different clustering techniques (Hierarchical Agglomerative Clustering and K-Means). In particular, we address the following research question for this part of the study:

RQ1: How effectively can we identify similar test steps that are written in natural language?
Understanding if we can effectively identify similar test steps automatically allows to know if we can rely on test step clusters to identify similarity between entire test cases. We found that we can achieve the highest performance (an F-score of 87.39%) using an ensemble approach that consists of different embedding and clustering techniques. In addition, we show that using Sentence-BERT instead of Word2Vec (which was identified as the best-performing model by prior work [28]) yields a slightly lower performance but reduces the execution time from 150 minutes to about 2 minutes.

- *Markos Viggiato and Cor-Paul Bezemer are with the Analytics of Software, Games and Repository Data (ASGAARD) lab at the University of Alberta, Canada. E-mail:{viggiato, bezemer}@ualberta.ca*
- *Dale Paas and Chris Buzon are with Prodigy Education, Toronto, Canada. E-mail: {dale.paas, christopher.buzon}@prodigygame.com*

In the second part of the study, we leverage the previously detected clusters of test steps to identify similar test cases. We compared three different techniques and related variations to compute a similarity score (using the simple overlap, Jaccard, and cosine metrics) to measure the similarity of test cases based on the test step clusters that they have in common. In particular, we address the following research question for this part of the study:

RQ2: How can we leverage clusters of test steps to identify similar test cases?

Given the difficulty of identifying similar test cases, which are usually composed of several steps, we use clusters of similar test steps to identify similar test cases. We found that test step clusters can be used to identify test case similarity with a high performance (an F-score of 83.47%).

Our work presents an approach to identify similar test cases based only on their natural language descriptions. We highlight that our approach is unsupervised as it does not require labelled data nor requires human supervision. In addition, no test source code or system model is necessary. QA engineers and developers can use our approach to obtain groups of similar test cases, which can be used, for example, to identify and remove redundant test cases from the test suite. Furthermore, existing groups of similar test cases can be leveraged to support the design of new test cases and help to maintain a more consistent and homogeneous terminology across the test suite. Finally, we provide access to the source code of our approach and the experiments that we performed.[1]

The remainder of the paper is organized as follows. In Section 2, we present background information about text embedding, clustering techniques and game testing. We discuss related work in Section 3 and our proposed approach in Section 4. Section 5 presents the dataset that we used to evaluate our approach. Sections 6 and 7 discuss the experiments that we performed to evaluate the two main stages of our approach. In Section 8, we discuss our results and the approach validation. Finally, Sections 9 and 10 present the threats to validity and conclude our work, respectively.

2 Background

In this section, we present an overview of the terminology and concepts that we use throughout the paper. In this work, we use “test cases” to refer to manual test cases that are described in natural language as a sequence of steps, i.e., test cases for which there is no source code associated.

2.1 Text representation

In order to use text data as input for a machine learning algorithm, we first need to convert the text into a numeric vector through a process called *text embedding* [56, 57]. Different methods can be used to obtain a text embedding, and the embedding can be done at different granularity levels, such as at word and sentence-level. Below, we explain the different techniques that we use in this work to obtain the numeric representation of words and sentences.

2.1.1 Word Embedding

A word embedding is the representation of a single word through a real-valued (and usually high-dimensional) numeric vector. In this study, we use two natural language processing techniques to obtain word-level embeddings: Word2Vec [34] and BERT [9]. Figure 1a presents two examples of pre-processed test steps along with part of their word embeddings obtained by the Word2Vec and BERT models. Next, we explain how each word embedding technique works and how the example embeddings presented in Figure 1a are computed.

Word2Vec transforms words into high-dimensional numeric vectors that are able to maintain the syntactic and semantic relationships between words in the vector space [34, 35]. This means that embeddings of similar words will (most of the time) be close in the vector space (i.e., the distance between the embedding vectors is small). Furthermore, with Word2Vec, each word is assigned a single numeric vector regardless of the context in which it is used, as we can see for the words “verify” and “item” in the two steps in Figure 1a. In this work, we used the continuous bag-of-words (CBOW) model architecture of Word2Vec, which is faster than the other possible architecture, called skip-gram [35].

Differently from Word2Vec, **BERT (Bidirectional Encoder Representations from Transformers)** is a transformer-based model that can be used to extract contextual word embeddings, i.e., embeddings that change depending on the context in which a word is present [9]. The context of a target word refers to the words that surround it, i.e., the words that appear before and after the target word. This means that the same word may have different embedding vectors, as we can see in Figure 1a, where the BERT embeddings for the words “verify” and “item” are different in the two test steps because those words are in different contexts.

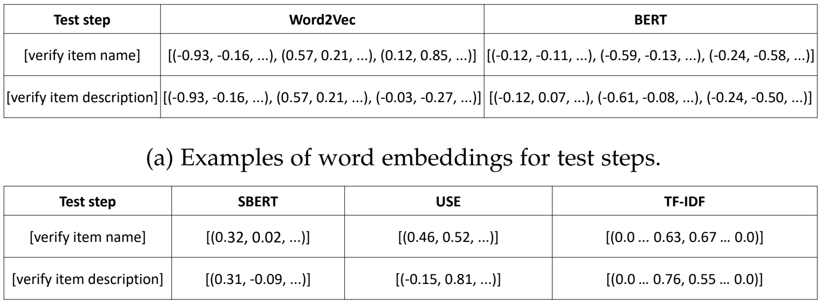
BERT is available as a model that was pre-trained on lower-cased English text (uncased BERT). This pre-trained model can further be trained with a domain-specific training set (known as domain-adaptive pre-training [17]). The BERT model uses WordPiece tokenization [58], in which a word may be split into sub-words. For example, the word “validate” is composed of the sub-words “valid” and “ate”, each one with its own embedding vector. Therefore, when extracting embeddings of words that are split into sub-words, we need to aggregate the embeddings of the sub-words (e.g., by averaging the embedding vectors).

2.1.2 Sentence Embedding

Differently from word embedding, sentence embedding is the representation of a whole sentence with a real-valued (and usually high-dimensional) numeric vector. In this work, we use three different techniques to extract sentence embeddings (SBERT, USE, and TF-IDF). Figure 1b presents two examples of pre-processed test steps along with part of their sentence embeddings obtained by the SBERT, USE, and TF-IDF techniques. Next, we explain how each sentence embedding technique works.

Sentence-BERT (SBERT) is a BERT-based framework that allows us to directly extract numeric representations of full sentences [43]. The embeddings of sentences that

[1] https://github.com/asgaardlab/test-case-similarity-technique



Test step	Word2Vec	BERT
[verify item name]	[(-0.93, -0.16, ...), (0.57, 0.21, ...), (0.12, 0.85, ...)]	[(-0.12, -0.11, ...), (-0.59, -0.13, ...), (-0.24, -0.58, ...)]
[verify item description]	[(-0.93, -0.16, ...), (0.57, 0.21, ...), (-0.03, -0.27, ...)]	[(-0.12, 0.07, ...), (-0.61, -0.08, ...), (-0.24, -0.50, ...)]

(a) Examples of word embeddings for test steps.

Test step	SBERT	USE	TF-IDF
[verify item name]	[(0.32, 0.02, ...)]	[(0.46, 0.52, ...)]	[(0.0 ... 0.63, 0.67 ... 0.0)]
[verify item description]	[(0.31, -0.09, ...)]	[(-0.15, 0.81, ...)]	[(0.0 ... 0.76, 0.55 ... 0.0)]

(b) Examples of sentence embeddings for test steps.

Figure 1: Examples of test step embeddings. Note that we provide only the first two elements of the embedding vector due to space constraints as the actual vectors have a high dimension.

are semantically similar are close in the embedding space. We can use this information for different purposes, such as identifying paraphrases and clustering similar sentences. For instance, the SBERT embeddings of the two test steps presented in Figure 1b are close in the embedding space (i.e., have a small distance between them). Among several generic and task-specific SBERT pre-trained models that are available[2], three models are suitable for our task (identifying similar test steps): *paraphrase-distilroberta-base-v1*, *stsb-roberta-base*, and *stsb-roberta-large*. While the first model is optimized to identify paraphrases and was trained on large scale paraphrase data, the second and third ones are the base and large versions of a model that was optimized for semantic textual similarity.

Universal Sentence Encoder (USE) is an embedding technique that can be used to directly extract embeddings from sentences, phrases, or short paragraphs to be used in another task, such as textual similarity and clustering tasks [5]. With a similar behavior to SBERT, the two examples presented in Figure 1b have close embedding vectors.

Finally, we also used the **TF-IDF (Term Frequency–Inverse Document Frequency)** method to represent sentences. TF-IDF computes the importance of a word to a document by combining the word frequency in the document and the word frequency across all the other documents [21, 22, 46]. In our case, the test steps (i.e., sentences) are considered documents. We built a numeric vector for each test step using the word importance values. Words that are not present in the step are assigned a value of zero. We can observe a typical vector obtained with TF-IDF in the examples presented in Figure 1b, in which the values different from zero correspond to the importance of the words presented in the "verify item name" and "verify item description" steps.

2.2 Clustering techniques

Hierarchical Agglomerative Clustering (HAC) [44] is a clustering algorithm that works in a bottom-up manner. Initially, each data point corresponds to a single cluster itself, and as the algorithm iterates, different clusters are merged with the aim of minimizing a specific linkage criterion. The result of the iterative merging process is a tree structure that can represent the data points (and their clusters), known as a dendrogram. Although the dendrogram can be used to identify the number of clusters, in our work we determined that parameter empirically and used the number that maximizes our evaluation metric (as explained in Section 6.5). Different linkage criteria can be used, such as single-linkage (the algorithm uses the minimum of the distances between all data points of two sets) and average-linkage (the algorithm uses the average of the distances between all data points of two sets).

The **K-means clustering** [12] algorithm splits the data points into k different clusters. Different from HAC, no hierarchical cluster structure is generated with K-means. The goal of K-means is to group data points in order to minimize the distance between points belonging to the same cluster compared to the distance of points from different clusters. Using the Expectation-Maximization algorithm [36], K-means starts with k centroids. Then, the algorithm (1) assigns each data point to the closest cluster (in terms of the distance between the point and the centroids) and (2) computes the new centroids using the updated data point assignments. The execution finishes when there is no change to the allocation of data points.

2.3 Game testing

Video game testing is substantially different from traditional software (e.g., desktop or mobile) testing. While there have been advances in test automation for traditional software, games still rely mainly on high-level, black-box, manual testing, in which human testers play through the game to assert its expected behavior (which is known as gameplay testing) [37, 39, 42]. Furthermore, the focus of game testing is more related to the overall user experience than to the accuracy of the test [42]. The test cases in a test suite of a game must also verify different types of requirements compared to traditional software, such as fun, entertainment, gameplay and other user experience aspects that traditional testing cannot satisfy [42]. Test automation is significantly more difficult in games for a number of reasons, such as (1) the difficulty of separating the user interface from the rest of the game, (2) the difficulty to explore the often large state space in games, (3) the challenge in asserting what the expected behavior is, and (4) the non-determinism that games have (e.g., because of multithreading, distributed

[2] https://www.sbert.net/docs/pretrained_models.html

computing, and AI agents) [37]. Finally, the common scenario of manual testing and the difficulties to automate tests in games show the need for new methodologies that can support QA engineers and developers during the game testing and enable game test automation in the future [39].

3 Related Work

In this section, we discuss prior work that applied clustering techniques [4, 6, 28, 52, 60] and natural language processing (NLP) [28, 30, 32, 33, 49, 54, 55] to software testing.

3.1 Clustering techniques for software testing

Our work is based on the study of Li et al. [28], which proposed an approach to cluster test steps written in natural language based on the steps' similarities. The study used text embeddings (including embeddings obtained with the Word2Vec technique) together with the Relaxed Word Mover's Distance (RWMD) metric [23] to measure similarity between embeddings. The test steps were then clustered with the hierarchical agglomerative and K-means clustering techniques. The approach was evaluated on a large-scale dataset of a mobile app and achieved an F-score of 81.55% in the best case. The proposed approach also reduced the manual effort for implementing test-step API methods by 65.90%. Differently from Li et al.'s work [28], we evaluated more recent NLP techniques to obtain word and sentence embeddings (BERT, SBERT, and Universal Sentence Encoder). Furthermore, we extended Li et al.'s work [28] for the purpose of identifying similar test cases using the identified clusters of test steps.

Walter et al. [52] proposed an approach to improve the efficiency of test execution. The approach removes redundant test steps and uses clustering techniques to rearrange the remaining steps. To use the approach, the textual descriptions of test cases must be converted into a representation form of parameters concatenated by first-order logic operators (AND, OR, NOT). The approach was evaluated in a case study with a system from an automotive industry company. The results indicated a test load reduction of 18% due to the removal of redundant test steps and rearranging of the remaining steps. Chetouane et al. [7] proposed an approach to reduce a test suite by clustering similar test cases (based on their source code) with the K-means algorithm. 13 Java programs were used to evaluate if the approach could efficiently reduce the test suite and assess the impact on coverage metrics. The evaluation showed that the approach can reduce the test suite by 82.2% while maintaining the same coverage metric as the original test suite. Even though the work of Walter et al. [52] addressed the problem of test step redundancy, their approach requires all test steps to contain a formal description of their precondition, action and postcondition. Creating these formal descriptions requires a large amount of manual effort which causes scalability issues and reduces the applicability of the approach in practice. Our approach does not require such manual effort. The approach proposed by Chetouane et al. [7] requires test cases that have source code associated with them. The test cases on which our approach focuses consist of only natural language descriptions and do not have any source code associated with them.

Pei et al. [40] proposed distance-based Dynamic Random Testing (DRT) approaches with the goal of improving the fault detection effectiveness of DRT. The work clustered similar test cases based on their source code with three clustering methods: K-means, K-medoids, and hierarchical clustering. The information of distance between the test case groups was used to identify test cases that are closer to failure-causing groups. 12 versions of 4 open-source programs were used to evaluate the approaches. The evaluation showed that the proposed strategies achieve a larger fault detection effectiveness with a low computational cost compared to other DRT approaches. Arafeen and Do [3] investigated whether clustering of test cases based on similarities in their requirements could improve traditional test case prioritization techniques. The paper used TF-IDF and the K-means clustering algorithm to group test cases that have similar requirements. Two Java programs were used to evaluate the approach. The evaluation showed that the use of requirements similarity can improve the effectiveness of test case prioritization techniques but the improvements vary with the cluster size.

Differently from the works above, our study aims at finding similar test cases that are written in natural language and for which there is no associated source code. We experimented with different NLP and clustering techniques to find clusters of similar test steps, which are used with test case names to obtain similar test cases. Furthermore, differently from the work of Walter et al. [52], which converts natural language descriptions of test cases into a representation form of parameters concatenated by logic operators to be used with their approach, our proposed approach works in an unsupervised manner with the original test cases written in natural language.

3.2 Natural Language Processing techniques for software testing

Wang et al. [54] proposed an approach to automate the generation of executable system test cases. The approach applies NLP techniques (such as tokenization and part-of-speech tagging) to textual data obtained from use case specifications. Furthermore, a domain model of the system under analysis is necessary to generate test data and oracles. Wang et al. [54] performed an industrial case study with automative software to demonstrate the feasibility of the proposed approach. Wang et al. [55] extended their previous work [54] by further providing empirical evidence about the scalability of the approach to generate executable, system-level test cases for acceptance testing from natural language requirements. In addition, Wang et al. [55] focused on embedded systems and demonstrated the effectiveness of the proposed approach using two industrial case studies, in which the approach correctly generated test cases that exercise different scenarios manually implemented by experts, including critical scenarios not previously considered.

Yue et al. [59] proposed a Test Case Specification (TCS) language, called Restricted Test Case Modeling (RTCM), and an automated test case generation tool, called *aToucan4Test*, to transform textual test cases into executable test cases. RTCM provides a template that combines natural language with restriction rules and keywords for writing TCS. Two case studies were performed to assess the applicability of RTCM and a commercial video conferencing system was used to evaluate the *aToucan4Test* tool. *aToucan4Test* could correctly generate 246 executable test cases from 9 test case specifications of subsystems of the video conferencing system. The study also evaluated the effort to use RTCM and *aToucan4Test* using the average time for deriving the executable test cases, which is 0.5 minutes. Mai et al. [30] addressed the problem of automatically generating executable test cases from security requirements in natural language. Mai et al. proposed an approach to generate security vulnerability test cases from use case specifications that capture malicious behavior of users. Similarly to previous work, Mai et al. evaluated the approach with an industrial case study in the medical domain. The evaluation indicated that the proposed approach can automatically generate test cases detecting vulnerabilities.

Prior work also used NLP techniques for test case prioritization and fault localization. Peng et al. [41] investigated program change-based test case prioritization using Information Retrieval (IR) techniques, in which the textual similarity between the program changes and the tests is used to rank tests for execution. Four techniques were used to compare and rank the tests, such as BM25, LDA, LSI, and TF-IDF, which transforms the text data into a numeric vector using bag-of-words. The proposed techniques were evaluated using cost-aware and cost-unaware metrics related to the Average Percentage of Faults Detected (APFD). Lachmann et al. [24] investigated test case prioritization of system-level, black-box test cases written in natural language. Test case textual descriptions were pre-processed (with techniques such as tokenization and stemming) and converted to numeric vectors using the frequency of terms occurring in the test cases. These vectors were combined with other test case meta-data (e.g., failures revealed by the test cases) to rank test cases based on their importance. Hemmati et al. [19] also studied test prioritization using natural language, black-box test cases. Three techniques were proposed (text diversification, topic modeling, and history-based test prioritization) and evaluated on Mozilla Firefox projects. The evaluation showed that, in rapid release environments, test case failure history can be used to effectively prioritize test cases for execution. DiGiuseppe and Jones [11] proposed a Semantic Fault Diagnosis (SFD), which automatically provides natural language descriptions of software faults. Using information extracted from the source code text (e..g, class names, comments, and other keywords), SFD can present developers not only with the pass and fail outcome of a test execution, but also a list of words that describe the topics related to the fault. Finally, Fry and Weimer [14] presented an approach that relies on textual features (e.g., term frequency vectors) from source code and defect report descriptions to localize defects in the source code. Using a similarity score to compare the representations of a defect report and the source code files, the approach ranks the source code files such that files at the top are more likely to contain the defect.

The aforementioned works used different NLP techniques to perform several tasks related to testing, such as to automatically generate different types of test cases, test case prioritization, and fault localization. In contrast, we propose an approach that leverages different NLP techniques to extract text embeddings and can automatically identify similar test cases. The approach can be used to identify and remove redundant test cases written in natural language.

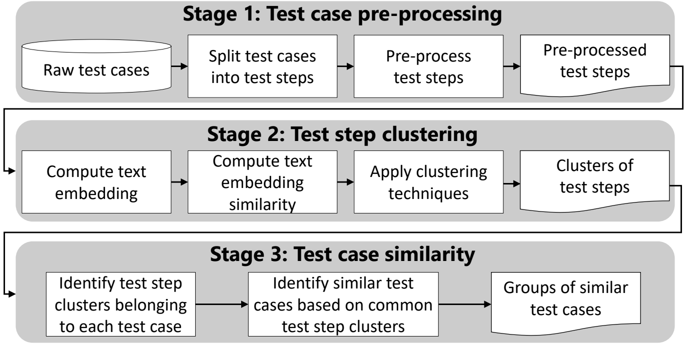

Stage 1: Test case pre-processing
Raw test cases
Split test cases into test steps
Pre-process test steps
Pre-processed test steps
Stage 2: Test step clustering
Compute text embedding
Compute text embedding similarity
Apply clustering techniques
Clusters of test steps
Stage 3: Test case similarity
Identify test step clusters belonging to each test case
Identify similar test cases based on common test step clusters
Groups of similar test cases

Figure 2: Overview of our proposed approach.

4 Proposed approach

In this section, we demonstrate our proposed approach through a running example. Our approach starts by clustering similar test steps, which are then used to identify similar test cases. We adopt a test step-based approach since test steps have a simpler grammatical structure compared to whole test cases. Also, a whole test case, which consists of all of its test steps aggregated, is not a coherent document as the test steps in a test case might be very different from each other. For example, in the same test case, one test step might be related to the 'login' functionality and another test step might be related to 'purchasing a membership'. Finally, using a test step-level approach provides more flexibility for recommending improvements not only related to whole test cases but also to individual test steps in the future. Figure 2 presents an overview of our approach, which consists of three stages: (1) pre-processing of test cases, (2) clustering of similar test steps and (3) identification of similar test cases. Next, we explain the stages of our approach, and we present an example that demonstrates the necessity of our approach.

4.1 Stage 1: Test case pre-processing

Our approach relies only on test cases that are written in natural language, which means that there is no source code available for our test cases. The input to our approach consists of unprocessed (raw) test cases. Table 1 presents three test cases (TC1, TC2, and TC3) that we use as a running example to describe how our approach identifies similar test cases. As we can observe, each test case contains an identifier, a name and a type. In addition, a test case has one or more test steps, which are instructions that the tester must perform in order to achieve the overall objective of the test case. Note that this objective is generally not explicitly specified. Test steps might be related to one or more game assets, which are the content of the game (e.g., in-game items, characters, and maps). The test steps that we collect to perform our experiments are explicitly identified (i.e., each test step has its own field within a test case). Therefore, we can directly collect the test steps and identify to which test

Table 1: Running example.

Test case identifier	Test case name	Test case type	Test step identifier	Test step (before pre-processing)	Test step (after pre-processing)
TC1	Log in to an existing account	Login	TS1.1	Login to the game using an existing account that has completed the tutorial	[login, game, using, existing, account, completed, tutorial]
			TS1.2	Select the Playing from School portal	[select, playing, school, portal]
TC2	Assignment with many students	Education	TS2.1	Update the assignment adding students	[update, assignment, adding, student]
			TS2.2	Request the next skill and question from the algorithm gateway for the 1st student on the assignment	[request, next, skill, question, algorithm, gateway, student, assignment]
			TS2.3	Request the next skill and question from the algorithm gateway for the middle student on the assignment	[request, next, skill, question, algorithm, gateway, middle, student, assignment]
TC3	Student has multiple assignments	Education	TS3.1	Request the next skill and question from the algorithm gateway for one of the students that was on the assignment	[request, next, skill, question, algorithm, gateway, one, student, assignment]
			TS3.2	Remove student from the first assignment	[remove, student, first, assignment]
			TS3.3	Request the next skill and question from the algorithm gateway for one of the students that was on the assignment	[request, next, skill, question, algorithm, gateway, one, student, assignment]
			TS3.4	Remove the student from the second assignment	[remove, student, second, assignment]

case they belong. Each test step is assigned a unique identifier and is pre-processed. Initially, we used tokenization to transform the step sentences into a list of words. To ensure that we have high-quality data, we obtained a list of the unique words in our data and manually inspected the list to identify misspelled words, which were used to build a list of [misspelled_word, fixed_word] tuples. The manually built tuple list was used to programmatically replace misspelled words with the corresponding fixed words across the entire dataset. We then removed stopwords (such as "a", "of", and "the") as they do not add meaning to the sentences. Also, we applied lemmatization to the words to have a consistent terminology across the data. Finally, similar to prior work [28], we removed words that occur only once in the whole dataset (507 out of 2,599 unique words) as they may result in incorrect embeddings due to the small amount of data for these words. Overall, a test case instance can be represented by the triple:

$$<test\_case\_name, test\_case\_type, test\_steps>$$

4.2 Stage 2: Test step clustering

In the second stage, our approach clusters similar test steps. Figure 3 shows how the steps of the three test cases are processed in this stage. Before applying a machine learning algorithm to text data, we need to transform the text into a numeric representation [56, 57]. Our approach starts by transforming each test step into one or more numeric vectors (text embedding). The pairwise similarity between steps (in terms of embedding distance) is then computed. The computed distances between the text embeddings of the test steps can be used to capture their similarity. In particular, embeddings that are close in the embedding space should represent similar steps.

Finally, our approach leverages the computed distances to identify clusters of similar test steps. While steps that have a small distance between them should belong to the same cluster, steps with larger distances should be in different clusters.

4.3 Stage 3: Test case similarity

In the last stage, our approach leverages the clusters of test steps identified in stage 2 together with the test case name to find similar test cases. Figure 4 shows how the TC1, TC2, and TC3 test cases are processed in this stage. The relationship between test cases and test step clusters is represented through a matrix in which each row is a test case (TC1, TC2, and TC3) and each column is a step cluster (C1, C2, C3, C4, and C5). Initially, for each test case (matrix row), the approach identifies the test step clusters (matrix column(s)) that contain one or more steps of the test case. Our approach supports the use of binary (which yields a matrix consisting of 0's and 1's) or numeric flags. Note that a numeric flag represents the number of test steps present in the identified cluster. After filling in the matrix, each test case is represented by the corresponding binary or numeric vector (a row in the matrix) with a length corresponding to the total number of test step clusters. Test cases are then compared to each other in terms of the similarity between their representation vectors. Finally, to incorporate knowledge from the test case name, the approach computes the pairwise similarity between test case name embeddings and combines this similarity metric with the one obtained from the test step clusters. The final test case similarity score is a weighted sum between the test step cluster and the test case name metrics. For the running example, our approach identifies the TC2 and TC3 test cases as similar but both are different from the TC1 test case. A QA engineer can then investigate those test cases to decide, for example, whether they are redundant or should be improved.

4.4 Motivational Example

Li et al. [28] proposed an approach to cluster similar test steps in natural language. Even though their work is supposed to be used for test steps only (and not entire test cases composed of one or more test steps), we evaluated an adaptation of their approach on our dataset using their

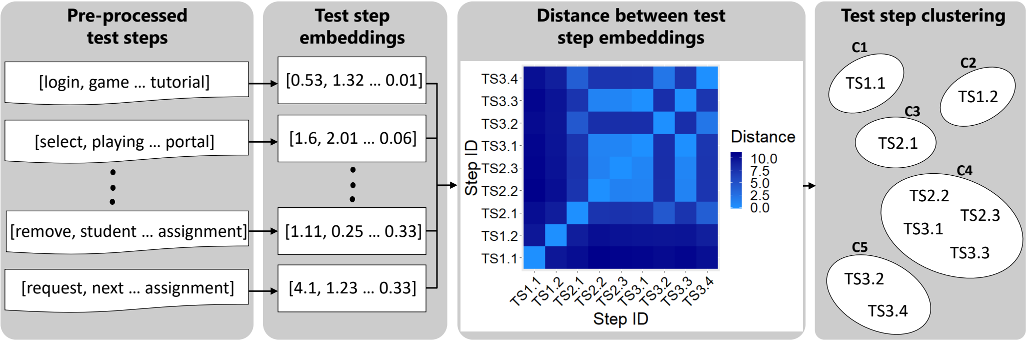


Figure 3: Overview of stage 2 of our approach with the running example.

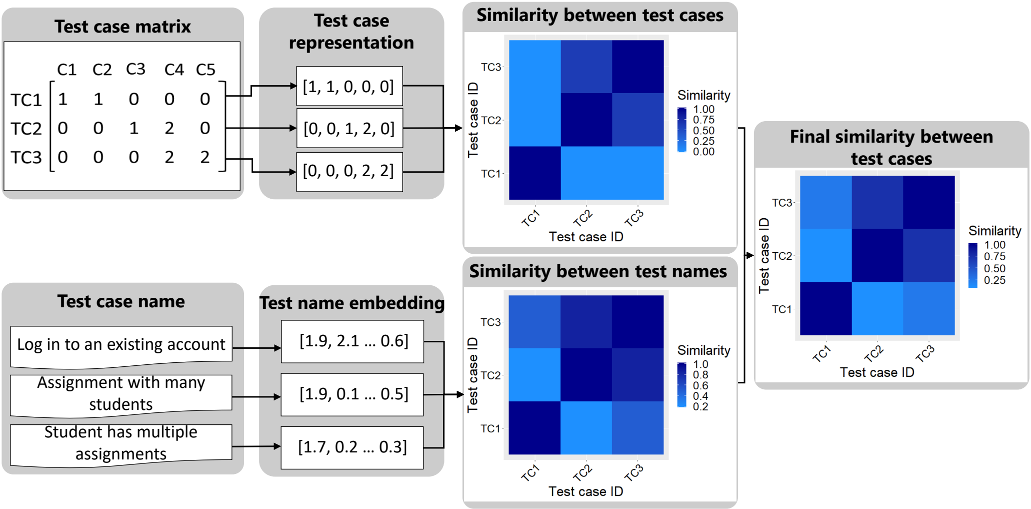


Figure 4: Overview of stage 3 of our approach with the running example.

Table 2: Motivational example of two similar test cases.

Test case name 1	**Test steps 1**
Boots - Ruin Dweller Boots (Got Item)	1. Verify item name 2. Verify item icon
Test case name 2	**Test steps 2**
Boots - Got Item	1. Verify item name 2. Verify item icon 3. Sanity check on wearable item - check ”Got item dialogue” for Boots

best-performing techniques. The adaptation approach consists of using Word2Vec for text embedding, Word Mover’s Distance (WMD) for text similarity, and hierarchical agglomerative clustering together with K-means for clustering. To be able to apply their approach, we considered a test case to be represented by either (1) the test case name concatenated with all the test steps or (2) all the test steps together. In both scenarios, the approach failed to cluster the two similar test cases presented in Table 2 (and there are many more examples in our dataset which could not be identified as similar by our adaptation of Li et al.’s approach).

The intended purpose of Li et al.’s approach is to cluster test steps (and not test cases). When considering a whole test case as a single test step, the granularity of Li et al.’s approach becomes too coarse and it considers the differences between the two test cases too large to cluster them together. However, when comparing the test cases step by step, instead of as a single blob of text, our approach detects that many of the steps overlap, hence clustering the two test cases in the same cluster.

5 Dataset and Ground Truth

We collected 3,323 test case descriptions written in natural language. The test cases under study were manually designed to test the *Prodigy Game*[3], a proprietary, educational math game with more than 100 million users around the world. Each test case is composed of one or more test steps and, in total, there are 15,644 steps. There is an average of 4.71 (and a median of 2) test steps per test case. We also collected the predefined type of the test case regarding the part of the game that is being tested. The test case type is available for 2,053 test cases (62% of the total number of test cases). All the test steps are pre-processed according to the pre-processing steps as explained in Section 4.1. Manual testing using test cases that are described only in natural language is still a common practice across several industries [18, 19, 28, 37, 39, 42, 52]. The test cases of the Prodigy game are similar in structure to natural language test cases

[3] https://www.prodigygame.com/main-en/

from other projects. Hemmati et al. [19] studied Mozilla Firefox projects, with manual test cases described only in natural language, with an objective and one or more test steps, similarly to our test cases. Li et al. [28] studied natural language test cases of a large industrial app (WeChat) with similar characteristics, such as 4.04 words per test step description on average (our average is 3.92) and test steps with simple grammatical structure. Walter et al. [52] studied automotive test cases in natural language, also with similar characteristics, such as an average of 3.57 test steps per test case for one of the studied systems (our average is 4.71).

To evaluate the performance of our approach (stage 2, for test step clustering, and stage 3, for test case similarity), we used our dataset to manually build a ground truth of similar test steps (stage 2) and similar test cases (stage 3), as we explain below.

Ground truth of similar test steps (stage 2 of our approach). We randomly selected a representative sample from all 15,644 test steps with a confidence level of 95% and a confidence interval of 5%, which corresponds to 394 steps. The test step samples were manually analyzed in an incremental manner: when analyzing test step *n*, we looked at all the (*n-1*) previously clustered steps to verify if step *n* should be assigned to an existing cluster or to a new cluster. To determine if two test steps are similar, we looked for two main characteristics of the data: (1) if the steps are textually similar or (2) if the steps give the same or similar instructions for testing, even if the textual descriptions are not similar. If any of those two characteristics are observed, we cluster the two samples together. Below, we show examples of pairs of test steps to demonstrate both characteristics:

(1) Textually similar test steps:

- “Play before 4pm and attempt to play video.”
- “Play before 8am and attempt to play video.”

(2) Test steps with similar instructions for testing:

- “Verify the game zones that can be selected by the student.”
- “Check which game zones are available to the student.”

The ground truth of similar test steps ended up with a total of 213 clusters and an average of 1.9 (standard deviation of 2.0) test steps per cluster. We also found that the largest cluster has 15 test steps. The fourth author independently validated the ground truth on a sample of 80 randomly selected pairs of test steps, which corresponds to a representative sample with a confidence level of 95% and a confidence interval of 10%. For each pair of test steps, the fourth author indicated if the two test steps should be in the same cluster (i.e., if they are similar) or not. We reached an agreement of 96.25% (which corresponds to a kappa coefficient [8, 25] of 0.89 or almost perfect agreement). The reached agreement demonstrates that the manual clustering process is straightforward (though time-consuming).

Ground truth of similar test cases (stage 3 of our approach). We randomly selected a representative sample of test cases with a confidence level of 95% and a confidence interval of 5%, which corresponds to 381 test cases. Similarly to the way that we built the ground truth of similar test steps, the test case samples were manually analyzed in an incremental manner: when analyzing test case *n*, we looked at all the (*n-1*) previously clustered test cases to verify if test case *n* should be assigned to an existing cluster or to a new cluster. To determine if two test cases are similar, we looked for the same characteristics (1) and (2) as for the test steps. If any of those two characteristics are observed, we cluster the two samples together. Note that, to analyze test cases, we consider the test case name, test case type, and all the steps that compose the test case. The ground truth of similar test cases ended up with a total of 242 clusters and an average of 1.6 (standard deviation of 1.9) test cases per cluster. For this ground truth, we found that the largest cluster has 21 test cases.

6 Evaluating our approach for clustering similar test steps

In this section, we discuss the experiments that we performed to evaluate our approach for clustering similar test steps in an industrial setting.

6.1 Evaluated techniques

Our approach consists of several steps that can be implemented through different techniques and models. To evaluate our approach, we performed experiments with combinations of five different text embedding techniques, two similarity metrics, and two clustering techniques. Figure 5 presents an overview of the experiments that we performed to address RQ1. Different NLP techniques can be used for text embedding at different granularities, such as words, sentences, and short paragraphs [5, 9, 26, 34, 35, 43]. As our test steps usually consist of a single sentence and the test steps are transformed into a list of words after preprocessing, we adopt word-level and sentence-level text embedding. We used two techniques to obtain text embeddings at the word-level (Word2Vec [34, 35] and BERT [9]) for the test steps and computed the embedding similarity using the Word Mover’s Distance (WMD) metric [23]. For text embeddings at the sentence-level, we used three techniques (SBERT [43], Universal Sentence Encoder [5], and TF-IDF [21, 46]) and used the cosine similarity to compare the embeddings. For both types of embeddings, we applied the hierarchical agglomerative [48] and K-means [12] clustering techniques to obtain clusters of similar steps.

6.2 Configuration of the word embedding techniques

Word2Vec. We trained a Word2Vec model using all 15,644 test steps that we collected. Furthermore, to provide more context to the embedding model during training, we concatenated the test case type (available for 2,053 test cases) and test case name to each step. We used an embedding vector of length 300 (as in the original study that proposed the Word2Vec model [34]). We used the continuous bag-of-words (CBOW) model architecture of Word2Vec with two context words as this configuration provides the highest test step clustering performance, which was determined through an experiment in which we varied the number of context words from one to ten. We initialized the word embeddings with the weights from the large-scale pre-trained

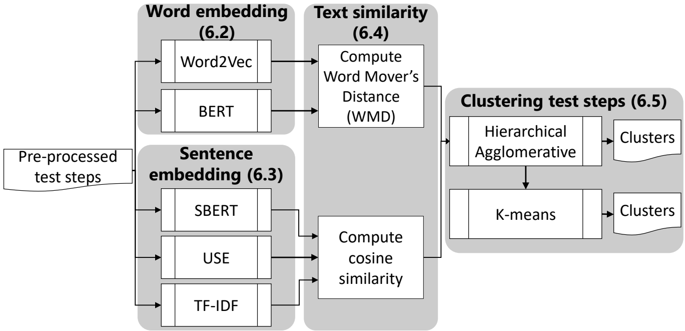


Figure 5: Overview of the experiments to identify clusters of similar test steps.

model released by Google.[4] This model contains 3 million word embeddings with dimension 300 and was trained on a Google News corpus with approximately 100 billion words. For words that are present in our dataset but not in the pre-trained model (and, therefore, cannot be initialized with pre-trained weights), we followed a process proposed by Li et al. [28] to initialize the word embeddings. We computed the mean and standard deviation of the initialized words and initialized the remaining words with samples of a normal distribution parameterized by the computed mean and standard deviation. Finally, the outcome of the training process is the word embeddings learned with our data.

BERT. In this work, we used the pre-trained model released by Google[5] (*pre-trained BERT*) to obtain contextual embeddings of the test steps. Furthermore, we used a model with additional pre-training using our own corpus of test steps (*domain-adaptive pre-trained BERT*) to obtain the contextual embeddings. We explain the configurations of both models below.

Pre-trained BERT. For the pre-trained model, we used the uncased (case-insensitive) version of the base model [9, 51]. We transformed the test step text into the BERT format by adding the *[CLS]* and *[SEP]* tokens respectively to the start and end of each test step text. The test step was then tokenized with BERT's own tokenizer. Finally, we used the tokenized steps to extract the contextual embeddings. As explained in Section 2.1.1, we can adopt different pooling strategies to obtain the embedding vector for a word. We performed experiments with four different pooling strategies to combine the layers (as suggested by the original paper's authors [9]): using only the second-to-last layer, summing the last four layers, averaging the last four layers, and concatenating the last four layers. We found that summing the last four layers achieves the best performance with our data. Finally, we used the average of sub-word embeddings (see Section 2.1.1) to obtain the original out-of-vocabulary word embedding.

Domain-adaptive pre-trained BERT. We also performed additional pre-training of BERT with our corpus. For the additional pre-training, after experimenting with the base and large models, we decided to use the uncased version of the BERT large model as the initial checkpoint (i.e., we performed the additional pre-training on top of the pre-trained large model). We followed the same process to configure the test step text to a BERT-friendly format. However, differently from the pre-trained model, using the second-to-last layer (instead of summing the last four layers) achieves the best results for the domain-adaptive pre-trained BERT model.

6.3 Configuration of the sentence embedding techniques

Sentence-BERT (SBERT). We performed experiments with three available pre-trained SBERT models suitable for our task (see Section 2.1.2): *paraphrase-distilroberta-base-v1*, *stsb-roberta-base*, and *stsb-roberta-large*. We decided to use the *paraphrase-distilroberta-base-v1* model since it achieves the best results with our data. To obtain the embeddings for the test steps, we just provided the test steps directly as parameters to the SBERT model.

Universal Sentence Encoder (USE). To obtain the test step embeddings with the USE model, we provided the steps directly as parameters to the USE model.

TF-IDF. Finally, we also used TF-IDF to extract the numeric vector representations of the test steps. For each word, we computed its importance in a single test step relative to all the other test steps. We used the *TfidfVectorizer* class provided by sklearn[6] with default parameters, which includes a smoothing parameter of 1 so that out-of-vocabulary words can be properly handled.

6.4 Computing the test step similarity

Word Mover's Distance (WMD). We used the Word Mover's Distance (WMD) [23] metric to measure the similarity between test step word-level embeddings. The WMD metric is suitable to be used together with the Word2Vec and BERT models because of the property that distances between embedded words in the embedding space are semantically meaningful, which is a property that WMD relies on [23]. Therefore, for word-level embeddings, we used the WMD metric instead of other metrics, such as the cosine similarity. We computed the pairwise WMD metric between any two test steps and built a distance matrix of dimension *[15,644 x 15,644]*. The more similar two steps are, the lower is the WMD metric, with the lowest bound being zero for exactly matching steps.

Cosine similarity. Since cosine is a widely used metric to measure similarity between text vectors [15, 20, 27, 47], we used the cosine to measure the similarity between test step sentence-level embeddings. Note that we cannot use the WMD metric for sentence-level embeddings since WMD requires the embeddings of each word individually instead of a whole sentence embedding. Similarly to the way we computed the WMD metric, we computed the pairwise cosine similarity between any two test steps and built a distance matrix of dimension *[15,644 x 15,644]*. As the cosine similarity score measures the cosine of the angle between the numeric vectors of two steps, the smaller the angle, the larger its cosine and the more similar the two test steps are.

[4] https://code.google.com/archive/p/word2vec/
[5] https://github.com/google-research/bert
[6] https://scikit-learn.org/stable/

6.5 Clustering test steps

Hierarchical Agglomerative Clustering. We applied the hierarchical agglomerative clustering technique to the distance matrix that we built in the previous step (Section 6.4). We used the average linkage criterion (with Euclidean distance), which means that the clustering algorithm merges pairs of test step clusters that minimize the average distance between each observation of the pairs.

K-means. To apply the K-means clustering technique, we used the test step embeddings obtained with the word/sentence embedding techniques (Sections 6.2 and 6.3). Note that, for word-level embeddings, we transformed the embedding vectors of the words of a test step into a single vector to represent the whole test step by computing the word embeddings' average [31, 61]. Furthermore, to speed up the execution of K-means, we used the centroids of the clusters obtained by the hierarchical approach as the initialization centroids, similarly to prior work [28, 29].

Regarding the number of clusters for both clustering techniques, we chose the number of clusters that maximized the F-score (which is our evaluation metric, as explained in Section 6.6). We performed a search by varying the number of clusters from 50 up to 15,000 with a step of 50, and for each value we executed both clustering approaches and computed the F-score. Finally, we selected the (optimal) number of clusters for which each clustering technique achieved the highest F-score. Note that the optimal number of clusters might be different for the hierarchical clustering and K-means.

Ensemble approach. Each text embedding technique that we used has different characteristics and properties to extract word or sentence embeddings, which leads to different clusters of test steps. Therefore, attempting to mitigate each model's specific weaknesses and based on prior work [10, 53] which showed that ensemble approaches might perform well for certain tasks (e.g., classification and clustering), we built an ensemble approach that uses majority voting. The approach uses the clusters generated by each previous single approach and starts by getting the set of all the test steps in the data. Then, it iterates through each test step and performs pairwise comparisons with all the other test steps. Suppose the approach (1) starts with test step TS$_n$. Then, (2) for each pair (TS$_n$-TS$_{n+1}$, TS$_n$-TS$_{n+2}$, etc.), the approach verifies if the majority of the single approaches (i.e., at least three out of five) assigned that pair to the same cluster or not and does the same assignment (i.e., puts the pair together if the majority did so or just skips the test step being compared to TS$_n$). After this first pass, we have all the test steps that are similar to TS$_n$. (3) The test steps that are clustered with TS$_n$ are removed from the main set of test steps (i.e., they will not be analyzed anymore). (4) We then repeat procedures (1), (2), and (3) for the next test step that is not part of the TS$_n$ cluster. When there is no test step left in the main set of test steps, the approach finalizes and we have a set of clusters of similar test steps.

Baseline. We used two baselines to evaluate the performance of our proposed approaches for test step (TS) clustering. The first baseline (*TS-Baseline 1*) assigns test steps to the same cluster only if those steps are exactly the same after pre-processing, similarly to Li et al. [28]. The second baseline (*TS-Baseline 2*) uses the Word2Vec technique together with the WMD similarity metric and only assigns two test steps to the same cluster if the WMD similarity of those steps is zero (i.e., their embeddings are the same).

6.6 Evaluation metric

We are interested in penalizing both the false positives (to avoid excessive suggestions of similar test steps when they are not similar) and false negatives (to avoid missing out many similar test steps). Therefore, we used the F-score metric (as shown in Equation 1) to evaluate the test step clustering approaches as this metric captures the trade-off between precision (related to false positives) and recall (related to false negatives). Even though we focus on the F-score, we also report the precision and recall of the proposed techniques along with the time necessary to execute the techniques. The executed time consists of the median time (in minutes) of five executions. Using the test steps present in the manually built ground truth of similar test steps, we analyzed all the pairs of test steps, similarly to prior work [28]:

- *True positive (TP)*: when a pair of steps is included in the same cluster by our approach and the steps indeed belong to the same cluster in the ground truth.
- *False positive (FP)*: when a pair of steps is included in the same cluster by our approach but the steps do not belong to the same cluster in the ground truth.
- *True negative (TN)*: when a pair of steps is not included in the same cluster by our approach and the steps do not belong to the same cluster in the ground truth.
- *False negative (FN)*: when a pair of steps is not included in the same cluster by our approach but the steps belong to the same cluster in the ground truth.

We then computed the F-score metric as follows:

$$\text{F-score} = 2 \times \frac{precision \times recall}{precision + recall} \tag{1}$$

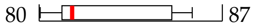
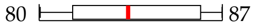
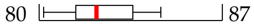
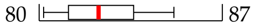
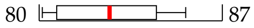
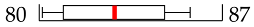
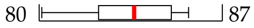
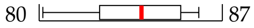
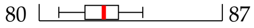
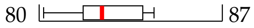
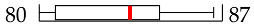
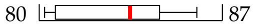
Where the precision corresponds to the proportion of true positives regarding all the pairs identified as positive ($\frac{TP}{TP+FP}$) and the recall corresponds to the proportion of true positives regarding all the existing positive instance ($\frac{TP}{TP+FN}$).

6.7 Findings

Similar test steps that are written in natural language can be identified with an F-score of 87.39% by applying the ensemble approach. Word2Vec (K-means), TF-IDF (HAC) and SBERT (K-means) also have high F-scores (86.99%, 86.67%, and 86.10%, respectively), but TF-IDF and SBERT considerably reduce the execution time (from 150 minutes to 2 minutes compared to Word2Vec). Table 3 presents the precision, recall, and F-score of all the approaches along with the execution time and the optimal number of clusters.

All the proposed approaches achieve a similar and high performance, with an F-score between 83.96% and 87.39%, except for both baselines, which have the same F-score of 70.40%. More specifically, the ensemble approach achieves

Table 3: Precision, recall, and F-score of the test step clustering approaches along with the execution time (in minutes) and the optimal number of clusters obtained using HAC and K-means. In the last column, we show the F-score distribution for a number of clusters between 2,150 and 3,000.

Text embedding technique	Clustering	Precision	Recall	F-score	Execution time (min)	Num. of clusters	F-score for num. of clusters between 2,150 and 3,000
TS-Baseline 1	Identical text	100.00	54.32	70.40	1	4,407	-
TS-Baseline 2	Identical embeddings	100.00	54.32	70.40	151	4,393	-
Word2Vec	HAC	93.74	79.19	85.85	149	2,650	
Word2Vec	K-means	94.24	80.77	86.99	150	2,650	
BERT	HAC	89.57	80.25	84.65	157	3,050	
BERT	K-means	91.14	79.89	85.15	160	3,050	
Domain-adaptive BERT	HAC	93.89	78.66	85.60	159	3,300	
Domain-adaptive BERT	K-means	94.29	78.66	85.77	162	3,300	
SBERT	HAC	94.67	78.30	85.71	2	3,350	
SBERT	K-means	95.09	78.66	86.10	2	3,350	
USE	HAC	90.26	78.48	83.96	1	3,050	
USE	K-means	86.91	82.01	84.39	1	2,900	
TF-IDF	HAC	91.90	82.01	86.67	2	2,500	
TF-IDF	K-means	91.80	80.95	86.03	2	2,500	
Ensemble approach	-	94.47	81.30	87.39	317	3,158	-

the highest performance, with an F-score of 87.39%. If we look at the performance of the single models, Word2Vec with K-means has the highest F-score (86.99%), which is very close to the ensemble approach performance. TF-IDF with HAC achieves the second highest F-score (86.67%) among the single models, followed closely by SBERT with K-means (86.10%) and Domain-BERT with K-means (85.77%). By analyzing the F-score obtained by all the approaches for all the searched number of clusters (from 50 up to 15,000), we observed that the F-score plateaus when we use a number of clusters of 6,000 or higher. This means that, in practice, we do not need to search for the optimal number of clusters with values above 6,000. We also noticed that the F-score is always above 80% when the number of clusters is between 2,150 and 3,000. We can therefore use a number of clusters in that range to avoid searching for the optimal number of clusters frequently.

Regarding the two versions of the BERT model, we observe that the domain-adaptive pre-trained BERT is a little better, with F-scores of 85.60% (using HAC) and 85.77% (using K-means), in comparison to the generic pre-trained BERT, with F-scores of 84.65% (using HAC) and 85.15% (K-means). One possible reason for the small gain is that we do not have large amounts of data for the domain-adaptive pre-training. However, our findings indicate that the additional pre-training is capable of improving the model performance and might be more helpful with larger datasets.

We can observe that for all the approaches except for TF-IDF, running K-means on top of HAC is beneficial as this increases the F-score. Note, however, that the gain in performance is minimal, such as 1.14% and 0.50% in absolute percentage point for Word2Vec and BERT, respectively. On average, applying K-means on top of hierarchical clustering increases the performance by 0.33% in absolute percentage point. The number of clusters obtained by the approaches with HAC does not necessarily need to be the same as the number of clusters obtained by the approaches with K-means. HAC and K-means are two different clustering techniques that we evaluate and, since they follow different procedures to cluster the data, they might achieve a different number of clusters. Note that, since we use the centers of the clusters obtained by HAC to initialize the K-means' centroids, K-means will converge fast as those initial cluster centers are often close to optimal or are in fact optimal in terms of F-score (which is the case when using HAC and K-means achieves the same number of clusters).

Table 3 also presents the precision and recall for all the approaches. Except for the baselines, we can observe that the precision varies from 86.91% (USE with K-means) up to 95.09% (SBERT with K-means) and the recall varies from 78.30% (SBERT with HAC) up to 82.01% (USE with K-means and TF-IDF with HAC). For the best performing models (ensemble approach and Word2Vec with K-means), both the precision and recall metrics are similar. Regarding the baselines, both of them present a very high precision (100%) but with a low recall (54.32%).

Finally, the presented execution time is the median time (in minutes) of five executions of the techniques. Even though the ensemble approach has the highest performance for clustering test steps, this approach is computationally expensive as it requires the implementation and execution of all the other approaches, which takes around 317 minutes (about 5 hours) in total (using six cores on an Intel i7-8700 CPU to compute the WMD metric and a single core for all the other computations). However, we can achieve a very close performance with a single technique, such as Word2Vec with K-means (which takes around 150 minutes to execute using six cores to compute the WMD metric).

TF-IDF (HAC) and SBERT (K-means) also achieve similar high F-scores (86.67% and 86.10%, respectively) but present much shorter execution times using a single core (2 minutes for both). Our experiments showed that both Word2Vec and BERT present a large execution time due to the large computational cost of computing the Word Mover's Distance, which makes SBERT a great alternative since it is considerably faster despite the slightly lower performance, and does not require further configurations or training as it uses a pre-trained model. The reported execution times are for the full test step clustering pipeline (test step preprocessing, word embedding training, test step similarity, and clustering) using the optimal number of clusters.

7 Evaluating our approach for identifying similar test cases

In this section, we discuss the experiments that we performed to evaluate our approach for identifying similar test cases that are specified in natural language. Below, we discuss four different techniques to identify similar test cases using the test step clusters obtained by the best-performing approach in Section 6 (ensemble approach).

7.1 Evaluated techniques

We performed experiments with three different techniques and variations of those techniques to identify similar test cases using the previously identified clusters of test step (with the ensemble approach). Figure 6 gives an overview of the experiments. To explain how each technique works, we use the two example test cases presented in Table 4.

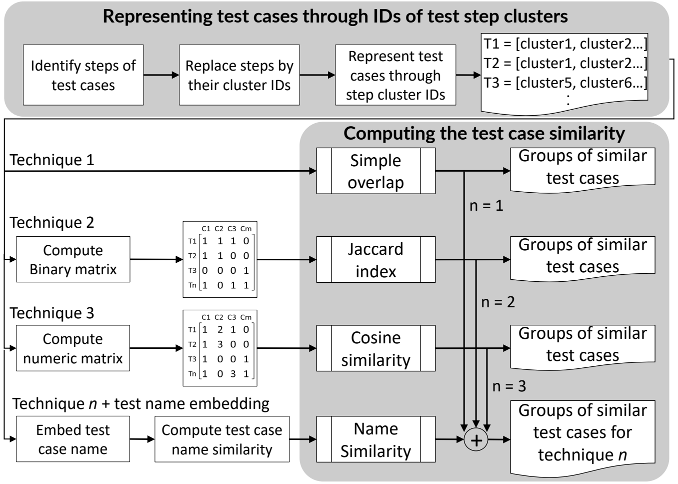

Representing test cases through IDs of test step clusters
Identify steps of test cases
Replace steps by their cluster IDs
Represent test cases through step cluster IDs
T1 = [cluster1, cluster2...]
T2 = [cluster1, cluster2...]
T3 = [cluster5, cluster6...]
Computing the test case similarity
Technique 1
Simple overlap
Groups of similar test cases
n = 1
Technique 2
Compute Binary matrix
C1 C2 C3 Cm
T1 1 1 1 0
T2 1 1 0 0
T3 0 0 0 1
Tn 1 0 1 1
Jaccard index
Groups of similar test cases
n = 2
Technique 3
Compute numeric matrix
C1 C2 C3 Cm
T1 1 2 1 0
T2 1 3 0 0
T3 1 0 0 1
Tn 1 0 3 1
Cosine similarity
Groups of similar test cases
n = 3
Technique n + test name embedding
Embed test case name
Compute test case name similarity
Name Similarity
Groups of similar test cases for technique n

Figure 6: Overview of the experiments to identify similar test cases.

In the example, there are two test cases (TC1 and TC2). TC1 contains four steps (TS1, TS2, TS3, TS4) and TC2 contains five steps (TS1, TS5, TS6, TS7, TS8). As we can see, only the TS1 step is shared between the test cases. In the test step cluster column, we can see the cluster ID to which each step belongs (TS1 belongs to the C1 cluster, TS2 belongs to the C2 cluster, and so on), where *Cn* is the ID of the cluster n. Note that different steps (such as TS2 and TS7) might belong to the same cluster (C2). Next, we explain each proposed technique using this example.

Technique 1: Test step cluster overlap. For this technique, we used only the identifiers of the test step clusters to represent test cases. For each test case, we gathered the unique list of cluster IDs that contain the test steps. For our running example, the TC1 test case is represented through the *[C1, C2, C3]* vector, while TC2 is represented through the *[C1, C2, C4, C5]* vector. Finally, we computed the pairwise similarity of any two test cases using a simple overlap metric, which indicates the proportion of overlap that test cases have in terms of test step cluster IDs, as shown below:

$$\text{Overlap} = \frac{length((TCn) \cap (TCm))}{max(length(TCn), length(TCm))} \qquad (2)$$

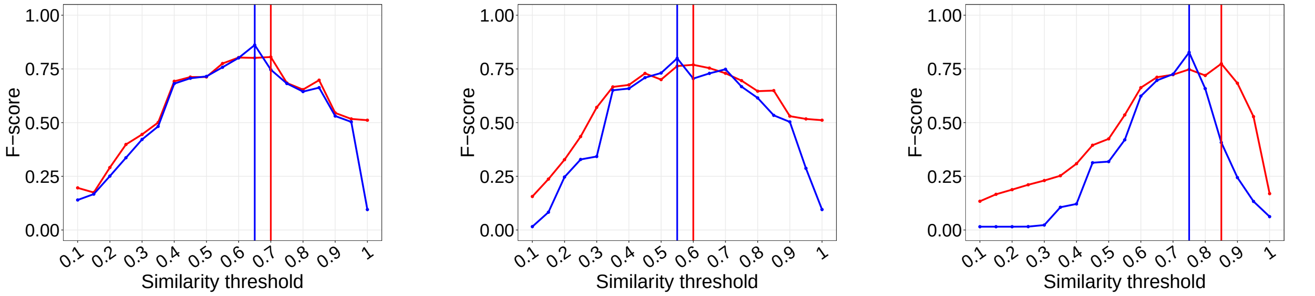
Where TCn and TCm correspond to the representations of the test cases n and m through the unique cluster IDs, respectively. Intuitively, test cases that have a large overlap of test step clusters (even if the test steps themselves are different) should be similar since test steps in the same cluster are (most of the time) similar. For our example, the length of TC1 is three (C1, C2, C3), the length of TC2 is four (C1, C2, C4, C5), and the length of the intersection between TC1 and TC2 is two (C1, C2). Therefore, the overlap between the TC1 and TC2 test cases is: $\frac{2}{max(3,4)} = \frac{2}{4} = 0.5$ (50%).

We used the computed overlap as the similarity metric to compare the test cases. Furthermore, in order to determine the optimal similarity threshold (i.e., with the optimal trade-off between false positives and false negatives) to be used to identify similar test cases, we performed a search by varying the threshold from 0.1 (10% of overlap) up to 1.0 (100% of overlap). Figure 7a shows how the F-score changes with the similarity threshold (the optimal threshold is indicated with the vertical red line). As we can see, our search showed that the threshold that provides the maximum F-score is 0.70, which means that two test cases should be considered similar if their overlap metric is at least 70%.

Technique 2: Binary representation of test cases. Similarly to Technique 1, for Technique 2 we used the test step clusters to represent test cases. However, instead of using the cluster IDs directly, we used a binary vector for each test case, in which we flagged the clusters that contain at least one test step of that case with a "1". Otherwise, we used "0". For our example, both test cases TC1 and TC2 are represented through a vector of length five because there are five different test step clusters in total (C1, C2, C3, C4, C5). TC1 is represented through the *[1,1,1,0,0]* vector (because TC1 has steps that belong to the clusters C1, C2, and C3, but no step belongs to the clusters C4 and C5), while TC2 is represented through the *[1,1,0,1,1]* vector. We built a matrix of dimension *[#test_cases x #test_step_clusters]*, where each row corresponds to a test case and each column corresponds to a test step cluster. Finally, we computed the pairwise similarity of any two test cases using the Jaccard index, as used in prior work [1, 2] to calculate the similarity between binary vectors. Our search (see Figure 7b) shows that the optimal lower threshold for the Jaccard index is 0.60 (vertical red line), which means that two test cases are similar if their Jaccard index is equal to or larger than 0.60.

Table 4: Examples of test case representations (through vectors) obtained with the experimented three techniques and their versions with test case name embedding (*Technique n + name embedding*).

Test case	Test step	Test step cluster	Technique 1	Technique 2	Technique 3	Technique *n* + name embed.
TC1	TS1, TS2, TS3, TS4	C1, C2, C3, C1	[C1, C2, C3]	[1, 1, 1, 0, 0]	[2, 1, 1, 0, 0]	Technique *n* + [TC1 Name embedding]
TC2	TS1, TS5, TS6, TS7, TS8	C1, C4, C5, C2, C5	[C1, C2, C4, C5]	[1, 1, 0, 1, 1]	[1, 1, 0, 1, 2]	Technique *n* + [TC2 Name embedding]



(a) F-score for different similarity thresholds for *Technique 1* and *Technique 1 + test name*.

(b) F-score for different similarity thresholds for *Technique 2* and *Technique 2 + test name*.

(c) F-score for different similarity thresholds for *Technique 3* and *Technique 3 + test name*.

Figure 7: F-score for different similarity thresholds for our proposed techniques. The vertical line indicates the threshold that maximizes the F-score (red for Techniques 1, 2, and 3 and blue for their versions with the test name).

Technique 3: Numeric representation of test cases. Using a binary vector to represent test cases might not be sufficient for situations where test cases have more than one step in a cluster. Therefore, we modified the previous technique so that, instead of representing test cases as a binary vector, we represent test cases as a numeric vector. This numeric vector corresponds to the number of test steps that the test case has in each cluster. For our example, TC1 is represented through the *[2,1,1,0,0]* vector (because TC1 has two steps in the C1 cluster, one step in each of the C2 and C3 clusters, and no step in the C4 and C5 clusters). TC2 is represented through the *[1,1,0,1,2]* vector. We found that using a threshold of 0.85 (see Figure 7c, vertical red line) achieved the best performance in terms of F-score. This means that all the pairs of test cases that have a cosine similarity equal to or larger than 0.85 are considered similar by Technique 3.

Including the test case name embedding. For each technique mentioned above (Techniques 1, 2, and 3), we evaluated their versions with the test case name embedding as well: *Technique 1 + test name*, *Technique 2 + test name*, and *Technique 3 + test name*. For our example, both test cases TC1 and TC2 are represented through the same vectors discussed above for Techniques 1, 2, and 3. For the versions with the test case name, we combined the test step clusters with the test case name embedding.

To obtain the embeddings for the name, we used the best-performing text embedding technique from the experiments for test step clustering (which is Word2Vec). Following a similar process as we did for the test step clustering, we computed the pairwise similarity for any two test case name embeddings. To compute the final similarity score for test cases, we used the weighted sum between the similarity score obtained with the test step clusters and the similarity score obtained with the test case name embeddings, as shown in Equation 3. Two test cases are considered similar if the final score is above a certain threshold.

$$\begin{aligned}\text{Final score} = \; &(\text{weight}) * (\text{test step cluster similarity}) \\ &+ (1 - \text{weight}) * (\text{test case name similarity})\end{aligned} \tag{3}$$

To determine the best weight and threshold for the final score, we performed a search similarly to the threshold search that we did for Techniques 1, 2, and 3. We varied the weight from 0.1 up to 1.0 with a step of 0.1. For each weight, we varied the threshold from 0.10 up to 1.0 with a step of 0.05. For each combination of weight and threshold, we obtained the clusters of similar test cases and computed the evaluation metrics.

For *Technique 1 + test name*, we found that the optimal weight is 0.9, i.e., the similarity score from the test step clusters contributes with 90%, while the similarity score from the test case name contributes with 10% to the final score. For *Technique 2 + test name*, the optimal weight is 0.8 and for *Technique 3 + test name*, the optimal weight is 0.5. Furthermore, as Figures 7a, 7b, and 7c show (vertical blue line), the optimal similarity thresholds for *Technique 1 + test name*, *Technique 2 + test name*, and *Technique 3 + test name* are 0.65, 0.55, and 0.75, respectively. Note that, due to space constraints and for a better visualization, Figures 7a, 7b, and 7c only display how the F-score changes with the threshold already using the optimal weights for each technique.

Baseline. We compared the performance of our proposed approaches with several baselines for test case (TC) similarity identification. *TC-Baseline 1* considers two test cases to be similar if they have the exact same steps (regarding the text of the step). *TC-Relaxed baseline 1* considers two test cases similar if they differ only in N test steps (with N=1), with the remaining test steps being exactly the same. Note that *TC-Baseline 1* has N=0. *TC-Baseline 2* considers two test cases to be similar if they have the same name. *TC-Relaxed baseline 2* considers two test cases similar if their test case name embedding vectors are close, for which we embedded test case

Table 5: Precision, recall and F-score of the test case similarity techniques along with the execution time (in seconds) and the optimal similarity threshold.

Technique	Technique name	Prec.	Recall	F-score	Exec. time (sec)	Thresh.
TC-Baseline 1	Identical steps	99.42	31.07	47.35	18.28	-
TC-Relaxed baseline 1	Close number of identical steps	93.91	38.57	54.68	23.36	-
TC-Baseline 2	Identical name	50.00	0.18	0.35	6.53	-
TC-Relaxed baseline 2	Close test name embedding	59.77	27.86	38.00	7.98	-
TC-Baseline 3	Aggregated test steps	68.87	72.68	70.72	325.82	-
TC-Baseline 3 + test name	Test case name + aggregated test steps	27.25	39.46	32.24	343.29	-
Technique 1	Test step cluster overlap	82.43	78.75	80.54	16.33	0.70
Technique 1 + test name	**Test step cluster overlap + test name embedding**	**83.19**	**89.29**	**86.13**	**33.63**	**0.65**
Technique 2	Binary repres. of test cases	78.34	75.53	76.90	144.09	0.60
Technique 2 + test name	Binary repres. of test cases + test name embedding	77.80	81.96	79.82	154.16	0.60
Technique 3	Numeric repres. of test cases	90.45	67.67	77.42	6.52	0.85
Technique 3 + test name	Numeric repres. of test cases + test name embedding	94.37	74.82	83.47	23.84	0.75

names with Word2Vec (as in Section 6.2) and searched for the optimal similarity threshold. *TC-Baseline 1*, *TC-Relaxed baseline 1*, *TC-Baseline 2*, and *TC-Relaxed baseline 2* are simple, intuitive and computationally cheap methods. *TC-Baseline 3* and *TC-Baseline 3 + test name* are based on existing document clustering techniques [20, 38]. We transformed the textual description of the test cases into a numeric vector (vectorization) using a traditional vectorization method (TF-IDF). We then computed the pairwise cosine similarity between the test cases and applied the HAC and K-means clustering algorithms. For both baselines, we consider the whole test case as a single document. For *TC-Baseline 3*, we represent a test case with all its test steps aggregated, while for *TC-Baseline 3 + test name* we consider the test case name concatenated to the aggregated test steps. For both cases, we followed a similar procedure as for the test step clustering (Section 6.5), in which we searched for the optimal number of clusters.

7.2 Evaluation metric.

To evaluate our approaches for finding similar test cases, we used the manually built ground truth of similar test cases to compute the precision, recall, and F-score. We followed the exact same process as we did previously for the test step clustering (Section 6.6).

7.3 Findings

Clusters of similar test steps and test case name embeddings together can be used to identify similar test cases with an F-score of 86.13%. Table 5 presents the F-score of all the techniques that we evaluated along with precision, recall, execution time and the optimal similarity threshold.

We observe that *Technique 1 + test name* achieves the highest performance in terms of F-score (86.13%), followed by *Technique 3 + test name*, which achieves an F-score of 83.47%. We also observe that, even though *Technique 3* achieves a higher performance than *Technique 2*, the improvement is very small (0.52 in absolute percentage point). This indicates that using the number of test steps in each cluster (instead of just flagging whether the cluster contains a test step) slightly improves the performance of the test case similarity technique. Further incorporating the test case name information considerably improves the performance for all the three techniques. *Technique 1 + test name* improved the F-score in 5.59 absolute percentage point from *Technique 1*. The absolute percentage point improvements for Techniques 2 and 3 were 2.92 and 6.05, respectively.

Regarding the baselines, all the experimented techniques perform considerably better than all the baseline methods. We observe that *TC-Baseline 2* achieves an extremely low F-score (0.35%) and that *TC-Baseline 3* presents the best F-score among the baseline methods (70.72%). Note that for *TC-Baseline 3*, HAC performed better than K-means, while for *TC-Baseline 3 + test name*, K-means performed better. We only report the results using the best-performing clustering algorithms. We also see that the precision of the *TC-Baseline 1* is very high (99.42%), but the recall is very low (31.07%). Two main reasons explain why all our proposed approaches perform better than the baseline methods. First, *TC-Baseline 1 and 2* and *TC-Relaxed baseline 1 and 2* are too simple to capture all the different types of similar test cases (e.g., test cases that have a different name and number of test steps, which, despite describing similar testing activities, are written differently). Second, *TC-Baseline 3* and *TC-Baseline 3 + test name* consider a whole test case as a single document. However, a whole test case is not a coherent document as test steps might be very different from one another. For example, in the same test case, one test step might be related to the 'login' functionality and another test step might be related to 'purchasing a membership'. Therefore, using whole test cases as documents for the similarity detection is a much more difficult task.

The presented execution time is the median time (in seconds) of five executions of the techniques and reflect the time necessary to run the test case similarity techniques considering that the clusters of test steps (obtained in the previous stage, as described in Section 6) are available. Apart from the *TC-Baseline 3* and *TC-Baseline 3 + test name* and *Technique 2* and *Technique 2 + test name*, all the other techniques present a similar execution time, which ranges from 6.52 seconds (*Technique 3*) up to 33.63 seconds (for the best-performing *Technique 1 + test name*). Using the optimal similarity threshold, the best technique (*Technique 1 + test name*) found 427 groups of similar test cases with two or more test cases in each group. The 427 groups contain a total of 2,193 test cases (65.9%), i.e., there are 2,193 test cases in the test suite that have at least one similar test case, according to this technique. This leaves 1,130 test cases for which there is no other similar test case. On average, each group has two similar test cases, with a standard deviation of four.

To understand the output produced by our best technique, we manually inspected a representative sample of 100 of the obtained groups of similar test cases. The sample was randomly selected with a confidence level of 95% and a confidence interval of 10% from the 427 groups of similar test cases obtained by the best technique (*Technique 1 + test name*). We identified four main types of similar test cases (shown in Table 6). While Type 1 corresponds to test cases with the same steps for different game assets, Type 2 regards test cases that have slightly different steps to indicate the asset being tested (e.g., **backpack hat** and **backpack wand**). Type 3 refers to test cases with a large overlap of steps but one of them has more/fewer steps, which might indicate unnecessary or missing steps. Finally, Type 4 regards redundant test cases, which are written differently and may have a different number of steps, but the testing objective is the same. The last type of similarity helps to identify and remove redundant test cases from the test suite.

8 Discussion

In this section, we revisit the research questions and discuss the validation of our approach.

RQ1: How effectively can we identify similar test steps that are written in natural language?

Our experiments demonstrate that we can identify similar test steps with a high performance in terms of F-score. We showed that an ensemble approach using a combination (majority voting) of different techniques (five text embedding techniques with two similarity metrics and two clustering algorithms) achieves the highest performance. Such ensemble approach has a large computational cost as it requires the execution of several different techniques. However, we showed that using a single technique (such as Word2Vec or TF-IDF) can also provide a high performance while being less computationally expensive.

RQ2: How can we leverage clusters of test steps to identify similar test cases?

Our experiments demonstrate that we can use the clusters of similar test steps identified in the first part of the study to represent test cases and identify the similar ones. More specifically, representing test cases through a vector that captures the number of test steps in each cluster boosts the similarity technique performance. Furthermore, we showed that combining the clusters of similar test steps with the embedding of the test case name achieves an even higher performance. Our experiments showed that the optimal weight (for our data) for the test step clusters and the test case names is 90% (i.e., the similarity score from the test step clusters contributes with 90%, while the similarity score from the test case name contributes with 10% to the final similarity score). In addition, we can use a threshold of 0.65 for the final similarity score to decide whether two test cases are similar (i.e., two test cases are similar if their final similarity score is equal to or larger than 0.65).

Validation with developers. To validate the results of our approach, we did an informal interview with a QA expert at Prodigy Education to discuss whether our results are valid and how they can be used in practice and improve the testing process. We selected a purposive sample [13, 16, 50] to explicitly select test cases that cover the different types of similar test cases that we identified.

Overall, the expert validated the different types of test case similarity that we identified and mentioned that our approach can help the QA to improve the quality of the test cases. More specifically, the QA expert pointed out four practical usages of our approach, as we explain next.

- **Identification of redundant test cases**, which are test cases that are described differently (e.g., because they were written by different professionals) but test exactly the same aspect/asset of the game.
- **Reuse of existing test cases** when creating new ones for new features of the game. In this case, existing descriptions of test cases can either be fully or partially (e.g., a few test steps) reused. The reuse can be full (e.g., when a new test case instructs the tester to perform the exact same steps of an existing test case but for a new game asset, such as a new consumable item in the game) or partial (e.g., when a new test case performs a similar test as an existing test case but with a few differences, such that only part of the test steps of the existing similar test case can be reused). By reusing test cases, the overall quality of the test suite improves with more consistent and homogeneous descriptions in terms of terminology. Furthermore, reusing test cases reduces the manual effort and time required for designing and creating new test cases.
- **Identification of test cases with missing steps**. A few test case samples that we discussed with the expert were indeed groups of similar test cases which perform the same task, but some of the cases had fewer steps than what is actually performed by a tester. We further investigated those cases with the QA expert and found out that the missing steps were scattered across the test suite (in different cases) and should be merged with the steps of the main test case.
- **Identification of test cases which are redundant but one of the cases has additional steps.** This occurs when a new test case is created based on existing ones, but some steps are added for clarification purposes and the older test case is not removed from the test suite.

9 Threats to Validity

External validity relates to the generalizability of our findings. One threat is that our findings rely on the test case descriptions of an educational game company. Test cases of organizations from different domains might be different (e.g., in terms of the used terminology, grammar complexity and structure, and characteristics of the data, such as the distribution of test steps across the test cases) and might affect the results. However, as we explained in Section 5, our test cases are similar in structure to natural language test cases from other domains studied by prior work, such as WeChat [28], automotive systems [52], and Mozilla Firefox

Table 6: Examples of the four types of test case similarity. Differences between test cases' steps are highlighted in bold.

Similarity type	Test case name	Test steps
(1) Same steps for different game assets	Check Hat - In Backpack	1. Verify item name 2. Verify item icon
	Check Wand - In Backpack	1. Verify item name 2. Verify item icon
(2) Slightly different steps for different game assets	Equip Hat	1. Trigger equip functionality via **backpack hat** item slot 2. Trigger unequip functionality via **backpack hat** item slot
	Equip Wand	1. Trigger equip functionality via **wand backpack** item slot 2. Trigger unequip functionality via **wand backpack** item slot
(3) Test cases with additional/missing steps	Check Consumables (Water Resist)	1. Use in battle 2. Check battle bonus 3. **Check item card name** 4. **Check item card stats**
	Check Food (Popcorn)	1. Use in battle 2. Check battle bonus
(4) Redundant test cases	Catch Firefly in Forest	1. **Catch firefly in forest**
	Firefly Forest - Catch Firefly	1. **Catch a firefly**

projects [19]. Our approach can be applied to natural language test cases from other industrial projects with similar characteristics as the test cases from our industry partner, such as the test cases from the projects and companies discussed above. Furthermore, our approach can be applied to well-maintained open-source projects which have test cases described in natural language that are composed of one or more individual test steps. Note that our approach consists of using clusters of similar test steps together with the similarity of test case name embeddings, which might be computationally expensive for large datasets. In addition, specifying a ground truth to be used with our approach can be challenging and time-consuming. Finally, further investigation is necessary to apply our approach to projects with different test case characteristics, such as different distributions of test steps across test cases or different test case structures (e.g., test cases which are not composed of individual test steps). Another threat is that our thresholds for optimal values (e.g., the number of clusters and the similarity score) likely do not apply to other systems. However, our method for searching for these values is generalizable.

Internal validity concerns the bias and errors due to the experimental design. One threat concerns the methods that we used for text embedding, text similarity and clustering. Even though we mitigated this threat by studying several different types of techniques (five different text embedding techniques, three similarity metrics and two clustering algorithms), different results might be achieved with other techniques. Future studies should further investigate additional methods and algorithms for text embedding, text similarity and clustering. Another threat is related to the manual analysis of the samples of test steps and test cases performed by one author to build the ground truth. The manual analysis is subject to error and bias because of human factors. To mitigate this threat, another author independently cross-validated a subset of the 20 randomly selected test steps, which achieved an agreement of 100%. Furthermore, a QA engineer with more than 5 years of experience in the company further validated the output produced by our technique.

Construct validity concerns the choices made during the construction of our experiments. One threat is related to the chosen parameters for the embedding techniques. We mitigated this threat by using well-studied or recommended values for such parameters. For example, our chosen length for the Word2Vec embedding vector is 300. This is a popular choice and was used in the original study that proposed the Word2Vec model [34], but models with different lengths might achieve different performances for the clustering task. Another threat concerns the pooling strategies that we used to combine the layers of the BERT models and extract the embedding vectors. Different strategies provide different word embedding vectors. Even though several strategies can be used, we compared the four strategies recommended by the original paper on BERT [9]. The use of the clusters obtained by the HAC algorithm to initialize K-means' centroids is another threat to validity. Even though we adopted this process to speed up the execution of K-means, using different initialization methods might produce different clusters and achieve a different performance. Future studies should investigate how the initialization of the K-means' centroids affects the performance of the test step clustering. Finally, another threat is regarding the evaluation of the test step clustering and test case similarity identification. We evaluated our proposed techniques using a random sample of test steps/cases, which may not reflect the characteristics of the entire population. To mitigate this threat, we randomly sampled the data with a confidence level of 95% and a confidence interval of 5%, which yielded a statistically representative sample.

10 Conclusion

Test cases written in natural language are often defined by different people who may use different terminology to refer to the same concept. As a result, many similar or redundant test cases may exist in the test suite, which increases the manual testing effort and the usage of development resources. Since manually identifying similar test cases is a time-consuming task, an automated technique is necessary.

In this paper, we propose an approach to identify similar test cases specified in natural language. First, we evaluated different text embedding techniques, similarity metrics, and clustering algorithms to identify clusters of similar test steps (which compose test cases). We then leveraged the identified test step clusters together with the test case name to identify similar test cases. To evaluate the approach, we used test cases from an educational game company. We manually built a ground truth of similar test steps and test cases and computed the F-score metric. The approach evaluation shows that similar test steps can be identified with a high performance (an F-score of 87.39%) using an ensemble approach which consists of different NLP techniques. We can also achieve a similar performance (an F-score of 86.99%) using a single technique (Word2Vec). Furthermore, we identified similar test cases with a high performance (an F-score of 83.47%) using clusters of similar test steps combined with the similarity between test case names.

In this work, we show how we can identify similar test cases based only on their description in natural language with an unsupervised approach, which requires no labelled data nor human supervision. As indicated in an informal interview with a QA engineer, our approach has several usages in practice, such as supporting QA and developers to identify and remove redundant test cases from the test suite. Furthermore, existing groups of similar test cases can be leveraged to create new test cases and help to maintain a more consistent and homogeneous terminology across the test suite.

Acknowledgments

The research reported in this article has been supported by Prodigy Education and the Natural Sciences and Engineering Research Council of Canada under the Alliance Grant project ALLRP 550309.

References

[1] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "On the accuracy of spectrum-based fault localization," in *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 2007, pp. 89–98.

[2] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[3] M. J. Arafeen and H. Do, "Test case prioritization using requirements-based clustering," in *IEEE sixth international conference on software testing, verification and validation*. IEEE, 2013, pp. 312–321.

[4] T.-D. B. Le, D. Lo, C. Le Goues, and L. Grunske, "A learning-to-rank based fault localization approach using likely invariants," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 177–188.

[5] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

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

[7] N. Chetouane, F. Wotawa, H. Felbinger, and M. Nica, "On using k-means clustering for test suite reduction," in *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 380–385.

[8] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[10] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[11] N. DiGiuseppe and J. A. Jones, "Semantic fault diagnosis: automatic natural-language fault descriptions," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 1–4.

[12] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification and scene analysis*. Wiley New York, 1973, vol. 3.

[13] I. Etikan, S. A. Musa, and R. S. Alkassim, "Comparison of convenience sampling and purposive sampling," *American journal of theoretical and applied statistics*, vol. 5, no. 1, pp. 1–4, 2016.

[14] Z. P. Fry and W. Weimer, "Fault localization using textual similarities," *arXiv preprint arXiv:1211.2858*, 2012.

[15] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," *International Journal of Computer Applications*, vol. 68, no. 13, pp. 13–18, 2013.

[16] J. M. Guarte and E. B. Barrios, "Estimation under purposive sampling," *Communications in Statistics-Simulation and Computation*, vol. 35, no. 2, pp. 277–284, 2006.

[17] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, "Don't stop pretraining: Adapt language models to domains and tasks," *arXiv preprint arXiv:2004.10964*, 2020.

[18] H. Hemmati, Z. Fang, and M. V. Mantyla, "Prioritizing manual test cases in traditional and rapid release environments," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–10.

[19] H. Hemmati, Z. Fang, M. V. Mäntylä, and B. Adams, "Prioritizing manual test cases in rapid release environments," *Software Testing, Verification and Reliability*, vol. 27, no. 6, p. e1609, 2017.

[20] A. Huang *et al.*, "Similarity measures for text document clustering," in *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, vol. 4, 2008, pp. 9–56.

[21] T. Joachims, "A probabilistic analysis of the rocchio algorithm with TF-IDF for text categorization." Carnegie-mellon univ, Pittsburgh, PA - Dept of computer science, Tech. Rep., 1996.

[22] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

[23] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From word embeddings to document distances," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. PMLR, 2015, pp. 957–966.

[24] R. Lachmann, S. Schulze, M. Nieke, C. Seidl, and I. Schaefer, "System-level test case prioritization using machine learning," in *In Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2016, pp. 361–368.

[25] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[26] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning (ICML)*. PMLR, 2014, pp. 1188–1196.

[27] B. Li and L. Han, "Distance weighted cosine similarity measure for text classification," in *International conference on intelligent data engineering and automated learning*. Springer, 2013, pp. 611–618.

[28] L. Li, Z. Li, W. Zhang, J. Zhou, P. Wang, J. Wu, G. He, X. Zeng, Y. Deng, and T. Xie, "Clustering test steps in natural language toward automating test automation," in *Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020, pp. 1285–1295.

[29] J.-F. Lu, J. Tang, Z.-M. Tang, and J.-Y. Yang, "Hierarchical initialization approach for k-means clustering," *Pattern Recognition Letters*, vol. 29, no. 6, pp. 787–795, 2008.

[30] X. P. Mai, F. Pastore, A. Göknil, and L. Briand, "A natural language programming approach for requirements-based security testing," in *Proceedings of the 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018.

[31] M. Marcińczuk, M. Gniewkowski, T. Walkowiak, and M. Bedkowski, "Text document clustering: Wordnet vs. TF-IDF vs. Word embeddings," in *Proceedings of the 11th Global Wordnet Conference*, 2021, pp. 207–214.

[32] S. Masuda, F. Iwama, N. Hosokawa, T. Matsuodani, and K. Tsuda, "Semantic analysis technique of logics retrieval for software testing from specification documents," in *Proceedings of the 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2015, pp. 1–6.

[33] S. Masuda, T. Matsuodani, and K. Tsuda, "Automatic generation of UTP models from requirements in natural language," in *Proceedings of the 9th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2016, pp. 1–6.

[34] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*.

[35] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[36] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60.

[37] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, "Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?" in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 2014, pp. 1–11.

[38] M. P. Naik, H. B. Prajapati, and V. K. Dabhi, "A survey on semantic document clustering," in *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, 2015, pp. 1–10.

[39] L. Pascarella, F. Palomba, M. Di Penta, and A. Bacchelli, "How is video game development different from software development in open source?" in *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018, pp. 392–402.

[40] H. Pei, B. Yin, M. Xie, and K.-Y. Cai, "Dynamic random testing with test case clustering and distance-based parameter adjustment," *Information and Software Technology*, vol. 131, p. 106470, 2021.

[41] Q. Peng, A. Shi, and L. Zhang, "Empirically revisiting and enhancing ir-based test-case prioritization," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 324–336.

[42] C. Politowski, F. Petrillo, and Y.-G. Guéhéneuc, "A survey of video game testing," *arXiv preprint arXiv:2103.06431*, 2021.

[43] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[44] L. Rokach and O. Maimon, "Clustering methods," in *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[45] G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing, Verification and Reliability*, vol. 12, no. 4, pp. 219–249, 2002.

[46] G. Salton, "Developments in automatic text retrieval," *science*, vol. 253, no. 5023, pp. 974–980, 1991.

[47] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[48] P. H. Sneath, R. R. Sokal *et al.*, *Numerical taxonomy. The principles and practice of numerical classification.*, 1973.

[49] S. H. Tan and Z. Li, "Collaborative bug finding for android apps," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 1335–1347.

[50] M. D. C. Tongco, "Purposive sampling as a tool for informant selection," *Ethnobotany Research and applications*, vol. 5, pp. 147–158, 2007.

[51] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," *arXiv preprint arXiv:1908.08962*, 2019.

[52] B. Walter, M. Schilling, M. Piechotta, and S. Rudolph, "Improving test execution efficiency through clustering and reordering of independent test steps," in *IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 363–373.

[53] X. Wan, "Using bilingual knowledge and ensemble techniques for unsupervised chinese sentiment analysis," in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 2008, pp. 553–561.

[54] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 24th International Symposium on Software Testing and Analysis (ISSTA)*, 2015, pp. 385–396.

[55] C. Wang, F. Pastore, A. Goknil, and L. Briand, "Automatic generation of acceptance test cases from use case specifications: an nlp-based approach," *IEEE Transactions on Software Engineering*, 2020.

[56] S. M. Weiss, N. Indurkhya, T. Zhang, and F. Damerau, *Text mining: predictive methods for analyzing unstructured information*. Springer Science & Business Media, 2010.

[57] S. M. Weiss, N. Indurkhya, and T. Zhang, *Fundamentals of predictive text mining*. Springer, 2015.

[58] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[59] T. Yue, S. Ali, and M. Zhang, "Rtcm: a natural language based, automated, and practical test case generation framework," in *Proceedings of the 2015 international symposium on software testing and analysis*, 2015, pp. 397–408.

[60] M. Zalmanovici, O. Raz, and R. Tzoref-Brill, "Cluster-based test suite functional analysis," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 962–967.

[61] Y. Zhang, J. Lu, F. Liu, Q. Liu, A. Porter, H. Chen, and G. Zhang, "Does deep learning help topic extraction? a kernel k-means clustering method with word embedding," *Journal of Informetrics*, vol. 12, no. 4, pp. 1099–1117, 2018.