Interpretable Test Case Recommendation based on Knowledge Graph

Wenjun Ke\(^1,2,3\), Chao Wu\(^3\), Xiufeng Fu\(^3\), Chen Gao\(^4\), Yinyi Song\(^3\)

\(^1\)Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
\(^2\)University of Chinese Academy of Sciences, Beijing 100049, China
\(^3\)Beijing Institute of Computer Technology and Application, Beijing 100854, China
\(^4\)Fifth Military Representative Office of the Chinese People’s Liberation Army Rocket Army in Beijing, Beijing, China

Abstract—Reproducing bugs and identifying causes is essential for the debugging of complex software systems. However, existing test case selection and recommendation technique diagnose bugs but failed to provide information to understand the cause. In this paper, we present an interpretable test case recommendation technique by building up knowledge graphs based on massive test cases, bug reports, code changes, and documents stored in software repositories. Specifically, it identifies correlations between new issue reports and historical information based on the knowledge graph and thus present test cases and corresponding documents to support the bug diagnosis. We conduct an empirical study on autonomous driving systems to show our technique is capable of identifying the proper test case. Further, we validate the effectiveness of recommended interpretation. The study shows that the recommended interpretation can help testers to comprehend bug reports and diagnose bugs efficiently.

Index Terms—Knowledge Graph, Test Case Recommendation, Interpretable Recommendation, Test Reuse

I. INTRODUCTION

With the rapid development of the software industry, software systems are becoming more and more complex. The system may be developed by a team, and every member here is only responsible for part of the system, so bugs appear. It is hard to find bugs which are caused by developers because of the thinking habits of developers. Therefore, professional software testing is of great importance. Although the software market is far from being saturated, software with various functions is emerging in an endless stream. As the iterative update of software accelerates and the scale of software increases dramatically, the maintenance of software becomes more and more difficult. These changes make quality assurance (QA) of software face a new challenge. Software testing plays an important role in troubleshooting software defects and ensuring software quality. In this process, the design and implementation of test cases occupy most of the Software Testing cycle because the quality of test cases directly affects the quality of Software Testing.

On December 20, 2019, a Boeing spacecraft starliner deviated from the channel shortly after launch and did not reach the scheduled orbit required by the space station. This aerospace giant with a century-old history and strong engineering strength was hit again after the 737 MAX air disaster, and the economic losses caused by it were countless. In the latest report, Boeing acknowledged that the cause of the accident was inadequate software testing of Starliner, and testers took shortcuts in order to shorten the test time, resulting in an obvious bug not being detected. Throughout the entire accident, it is difficult for testers to redesign sufficient test cases due to the time constraints, and finally, they chose to take the risk. If historical test cases can be fully utilized, the design efficiency of testers will be greatly improved, and such tragedy can also be avoided to happen again.

Knowledge graph (KG) and artificial intelligence (AI) are widely used in data mining and other fields in recent years [1]. The knowledge graph can well represent the correlation between things, so personalized recommendation based on Knowledge graph has been widely applied in various fields such as shopping, movies, and tourism. In addition, AI has also been introduced to the field of software testing, and software testing is moving towards intelligence [2]. At present, the intelligent testing of software has received widespread attention at home and abroad, and a large number of research results and application practices have been obtained. Software intelligent testing focuses on issues such as the combination of intelligent technology and test scenarios, automatic generation of test cases [3], regression of test cases [4], defect location technology based on artificial intelligence [5], and test reuse [6]. These practical applications have proved the feasibility and practicability of artificial intelligence in the field of software testing. AI can help testers to complete their testing tasks outstandingly.

Test data such as test cases are usually accumulated during the process of software testing, especially for third-party organizations specializing in software testing. Massive test data contains valuable test knowledge [7], so the field test knowledge can be well represented and mined with the knowledge graph. Meanwhile, it can help testers to see the nature of the problem combined with artificial intelligence technology, thus making testers design test cases more efficiently and completing their testing tasks outstandingly [8].

Our contributions are as follows:

- We use the knowledge graph to store and represent software test assets and construct the knowledge base of software testing, which helps testers analyze historical data and design test cases better and faster.
- We propose a test case recommendation method which can quickly find suitable test cases from a large number...
of test cases, considering the ability to check errors in test cases. Certain auxiliary information is also provided to help testers understand the reason for recommending these test cases.

- We conduct a case study to demonstrate that our technique is capable of identifying the proper test case with an average accuracy value, and the recommended interpretation can help testers to comprehend bug reports and diagnose bugs efficiently.

II. RELATED WORK

A. Test Case Reuse

With the improvement of software quality awareness, the proportion of software test costs is getting higher and higher. In order to save the cost of software testing, the test reuse technology has received great attention. Test case design based on reuse technology can improve software testing efficiency [9].

The research on test case reuse has attracted a lot of scholars at home and abroad in recent years. Li et al. [10] designed and optimized the test case database and proposed a database model: According to the test task requirements. The tasks are decomposed into the corresponding test case group information, and then the database query function is used to query the current use case database and reuse the returned test cases directly. Nan et al. [11] proposed faceted classification of test cases, and constructed test case index trees to provide clues for test case retrieval, using test case index tree and tree matching model ideas to propose a test case retrieval method for different data patterns. Compared with the traditional database model retrieval method, this method improves the retrieval efficiency of test cases effectively. Guo et al. [12] combined with the collaborative filtering recommendation method to propose a test case recommendation reuse technology based on the radar software defect library. The related items are recommended to the users who have similar features and interests by analyzing user groups. Fraser et al. [13] proposed a test case reuse method based on commonality analysis. This method extracts commonality requirements by analyzing the commonality of different systems in the same field and then subdivides them to design reusable test cases. When testing a new system, you can select reusable test cases from the test case library.

The above research has promoted the development of test case recommendation technology effectively. However, they have the following problems: database-based reuse methods need to manually establish test task lists and classify test group information to which test cases belong. During the test case repetition is executed, the tester determines the specific test content, manually selects the test service number, and then returns the required test cases through the database matching method. It can be time expensive to build the test task table and cannot truly understand the user’s intention, so the accuracy of the results is not high. The collaborative filtering-based test case recommendation method needs to mine the characteristics and interest information of users, but the interest information is usually sparse. For projects that first appeared and were not evaluated, pure collaborative filtering could not predict and recommend it, resulting in poor recommendation results.

In addition, all current research methods ignore the consideration of test case error detection capabilities, but it is one of the main purposes of software testing to check the software defects by designing test cases. As a result, a large number of test cases are provided to the testers through the matching model with little help for testers.

B. Application of Knowledge Graph in Software Engineering

In the field of software engineering, the research on knowledge graph begins as early as 2008. Some researchers studied the construction of software ontology in software engineering, hoping to reduce the software maintenance cost by constructing the software ontology [14], [15]. In recent years, the existing entity recognition methods in the field of software engineering are mainly oriented to a single type of data sources such as error reporting, or a single type of entity such as API, and a single software project [16]. But the construction method is not universal and extendible, and the accuracy has not reached the level of entity recognition in the general domain. In the field of software security, researchers have designed a vulnerability mining algorithm, which analyzes and obtains the basic characteristics of software vulnerabilities by mining three databases [17]. Some researchers have also proposed a deep learning method that only uses the vulnerability descriptions in the CVE dataset to predict the severity of software vulnerabilities [18]. Based on the structured data in the CWE database, the knowledge graph of common software vulnerabilities is constructed and embedded to support various inference tasks related to software vulnerabilities. In the field of software testing, some researchers also use ontologies to define and structure the testing to make the process of maintenance testing easier [19].

In summary, this paper proposes a method for interpreting test cases based on knowledge graph, applying knowledge graphs and natural language processing technology to software testing. The strong correlation of knowledge graph is used to analyze the rules in the historical test cases, recommending test cases with high error checking rate for testers, and helping them to find hidden defects better. The sub-graph information is fed back to testers for helping them understand the reasons for the recommendation, thus improving the interpretability and credibility of the recommendation. As a result, we can reuse test cases better and improve the efficiency and quality of software test.

III. INTERPRETABLE RECOMMENDATION OF TEST CASES BASED ON KNOWLEDGE GRAPH

During software testing, only a small number of crucial test cases can generate problem lists, so the importance of different test cases are different [20]. Using a guided testing method to help testers analyze the test requirements, and clarify the test focus is a major problem to be solved in this paper. It is necessary to measure the importance of test cases based on
knowledge graph so that the important cases can be prioritized to achieve the guided testing [21].

In this paper, we propose a test case interpretable recommendation method based on the knowledge graph. A reusable set of test cases can be found from the knowledge graph based on test items and software information, which can recommend test cases easier to find defects and ensure the quality of software testing [22].

A test case is composed of many keywords (interruption, communication, cycle time, etc.) [23]. The probability of causing problems is different under different keywords. Therefore, we need to first find the historical test cases in the software test knowledge graph based on the keyword information, and then sort test cases according to the keywords combination so that test cases which easy to find problems are recommended first. So the recommendations are interpretable in our method.

The method in this paper is mainly divided into two parts: 1. technology to filtering candidate test cases with similar requirements from historical data by using knowledge graph, 2. evaluating and predicting the error detection capabilities of test cases by using artificial intelligence. As a result, software defects are found more easily for testers. The following sections introduce the knowledge graph in software testing and the intelligent recommendation for test cases.

A. Knowledge Graph in software testing

In software testing, software requirements are decomposed into multiple test items, which in turn are decomposed into multiple test cases [24]. Therefore, test cases corresponding to similar software requirements have some relevance. Historical software requirements which similar to the current software requirements can be calculated, and test cases corresponding to similar requirements can be obtained from the knowledge graph as recommendations. The following figure 1. is a new extracted test item A, which contains test content and requirement, test constraint, test method and judgment criteria.

![Fig. 1. Test item example](image)

According to the common knowledge extraction principle when constructing the knowledge graph, the above test items are extracted into the following test item A partial graph, which is shown in Figure 2.

The software information (software model, software type, software described information, programming language, software requirements, etc.) which test item A belongs to is also the key information for querying the test item in the knowledge graph. More test items can be found by querying similar test items to test item A in test knowledge graph based on the above four description information of test item A and the software information described in test item A. As shown in the figure 3., test item B and C are similar to test item A, which consider both the software information, and test constraint and test method related to the test items. Using partial graph matching method to find similar test items embodies the advantage of strong similarity in similar knowledge in the knowledge graph. If we use keyword matching method without knowledge graph, it is difficult to take into account knowledge chains such as Software → Requirements → TestItems → TestCases, which will reduce the accuracy of recommendation of subsequent test cases. Secondly, for the final recommended test cases, returning the sub-graph containing the knowledge for testers can increase the interpretability of the test case recommendation greatly and assist the tester to understand the reason for the test case decomposition quickly.

B. Intelligent Recommendation Method for Test Cases

After finding similar test items, a set of candidate test cases can be obtained. How to evaluate the test cases and filter the final recommended test cases is the focus of the rest of this paper. Different from the collaborative filtering method, we propose a new recommendation model based on keyword discovery to filter the test cases that are most likely to detect problems from the set of candidate test cases.

First, organize the existing test items, test cases, and corresponding problem list to build a sample set; next, use a named entity recognition model to identify the key information in the test items and test cases, such as interruption and data collection. Then, use the information gain to calculate the information gain of each keyword pair in the category of whether or not a problem list is generated. Finally, keywords with large information gain are selected as features, and a vector representation is generated for each test item in the sample set (one dimension in the vector corresponding to each keyword or keyword combination), and the classifier is trained in a supervised manner. The probability of each test item which can test defects can be predicted by using the trained classifier, and test cases that are prone to test defects can be recommended for testers, thereby helping testers design test cases that are more likely to find defects. In addition, because the key features for classification are calculated in our method, it can output partial graph of test item, test cases, and problem list when recommending test cases. Meanwhile, the keyword information of the recommended test cases is marked to help testers understanding.
The overall framework of the test case recommendation model proposed in this paper is shown in the figure 4.

Fig. 4. Overall Model Framework of Use Case Recommended

1) Domain Segmentation Based on Named Entity Recognition Model: A deep learning-based named entity recognition model to transform the entity recognition problem into a sequence labeling problem is used in our method [25], marking each word in a sentence as four types of BIEO labels (B indicates the beginning of the word, I indicates the middle part of the word, E indicates the end of the word, and O indicates unimportant words). The input of the model is a sequence of characters and the output is a label sequence, so it is an end-to-end sequence label problem.

We use a main current deep labeling framework for sequence tagging: BiLSTM (Bi-directional Long Short-Term Memory) + CRF (Conditional Random Field). BiLSTM integrates two sets of LSTM layers with opposite learning directions (one in sentence order and one in reverse order), which can theoretically learn the overall sentence information and is more conducive to increase the accuracy of labeling sentences [26]. The domain segmentation model based on deep learning is shown in the figure 5.

Fig. 5. BiLSTM + CRF model

The NER model is used to segment the collected description text in test items and test cases into words. Then we remove words that have no meaning to the semantic expression by NER model, so as to extract key information (named entities) in the sentence. Finally the key information is input in the information gain model to calculate core keywords.

2) Combination Feature Mining Based on Apriori: Apriori algorithm was first used to mine association rules [27], in order to find the hidden relationship between different data items in the data set, and finding frequently occurring items or subsets in massive data sets, and the correlation between items [28]. In order to find out the impact of the combination of features between core keywords on classification, Apriori algorithm is used to further process the core keywords.

The algorithm can be described as follows:

**Input:** NER segmentation result \( S = [s_1, s_2, s_3, \ldots, s_n] \), where \( s_i = [w_1^i, w_2^i, w_3^i] \) is the NER segmentation result of the \( i \)-th sentence (\( i = 1, 2, \ldots, n \)), \( w_j^i \) is recorded as the \( j \)-th word (\( j = 1, 2, \ldots, m \)) in the result of the \( i \)-th question segmentation.

**Output:** Frequent itemsets \( L = [L_1, L_2, L_3, \ldots, L_k] \), Where \( L_i \) is the frequent \( i \) itemset (itemset which has \( i \)
Step 1: Generate candidate set $C_1 = \{c_1, c_2, c_3, \ldots, c_n\}$ by using NER segmentation results: Extract all words of all sentences from the word segmentation result of the test cases and issue reports, then deduplicate it to get frequent item set $L_1 = C_1$.

Step 2: Use frequent item set $L_1 = [l_1, l_2, l_3, \ldots, l_n]$ to get candidate set $C_2 = \{c_1, c_2, c_3, \ldots, c_q\}$: Take out all the items in $L_1$, then arrange and combine the items in $L_1$ to obtain $c_0$, and the number of words contained in $c_0$ is 2. Finally, put $c_0$ into the candidate set $C_2$, and the complete candidate set $C_2$ is obtained until the new $c_p$ cannot be generated.

Step 3: Use candidate set $C_2 = \{c_1, c_2, c_3, \ldots, c_q\}$ to get frequent binomial set $L_2 = [l_1, l_2, l_3, \ldots, l_x]$ : Get item $c_t$ from candidate set $C_2$ first, and calculate the optimized support $E(c_t)$ corresponding to $c_t$:

$$E(c_t) = \frac{\sum_{i=1}^{m} \prod_{j=1}^{y} f_{new-tf-idf}(c^j_t)}{N}$$

(Where $N$ is the total number of NER segmentation results, $m$ is the number of segmentation results containing $c_1, c_2$ is the spacer word between words in a single field in $c_0$, candidate set $c_0$ has $r$ spacer words, $f_{new-tf-idf}(c^j_t)$ is the normalized value of TF-IDF for each spacer.)

If the corresponding optimized support $E(c_t)$ meets the minimum support requirement $E_1$, then represent it as $l_y$ and add it to $L_2$. Finally put $L_2$ into $L$.

Step 4: Use candidate set minimum support $E_{n+1}$ for frequent itemsets $n + 1$:

$$E_{n+1} = q \ast E_n = q^n \ast E_1$$

(Where $q$ is the coefficient of variation for minimum support, $E_1$ is the minimum support corresponding to the frequent item set set during algorithm initialization, $E_1$ is the minimum support for frequent $n$ itemsets.)

Step 5: Go to step2, get candidate set $C_1, C_2, \ldots, C_i$ and frequent item set $L_1, L_2, L_3, \ldots, L_i$ in order, and add $L_1, L_2, L_3, \ldots, L_i$ to $L$, until frequent $i$ itemsets do not meet the minimum support requirements.

3) Core Features Extraction with Information Gain: The selection of classification features is to select the features that have classification capabilities for training data, and information gain is a criterion for describing the classification capabilities of features [29].

Information gain introduces the concept of entropy, which means the degree of chaos of a system [30]. The higher the uncertainty of the system, the greater the entropy.

We assume the variable set $X = \{x_1, x_2, \ldots, x_n\}$ and its corresponding probability set $P = \{p_1, p_2, \ldots, p_n\}$ . Then the entropy $H(X)$ is expressed as:

$$H(X) = - \sum_{i=1}^{n} p_i \log_2 p_i$$

(1)

The information gain $G(D, A)$ of feature $A$ to training set $D$, is equal to the empirical entropy $H(D)$ of set $D$ minus the conditional empirical entropy $H(D|A)$ of $D$ under the given conditions of feature $A$, that is

$$G(D, A) = H(D) - H(D|A)$$

(2)

The domain vocabulary obtained by NER word segmentation is regarded as a feature. After the calculation of its information gain, we can get the importance of the classification of the overall data set (whether it will cause defects) , so that it can filter some core features which have a meaningful explanation for the recommendation.

4) Classification Model Training: According to the core keywords obtained in step 2, we encode the training set by using the bag-of-words model to turn each sentence into a vector $X = (x_1, x_2, x_3, \ldots, x_n)$, where $n$ is the number of core keywords. Then put $X$ into the classification models’ network for training. There are many kinds of commonly used classification algorithms. In this paper, we use Naïve Bayesian classifier which is based on Bayes’ theorem and the independence of characteristic conditions [31]. $P(A)$ is the prior probability, which represents the probability of each category distribution; $P(B|A)$ is a conditional probability, which means the probability of something happening under the premise of a certain category; the conditional probability can be obtained by counting samples. $P(A|B)$ is the posterior probability, which means that something happened and it belongs to a certain category. With this posterior probability, the samples can be classified. The greater the posterior probability, the more likely something belongs to this category, the more reason to put it under this category. According to the Bayesian formula, the posterior probability of a sample in different classification categories can be calculated and compared, then take the largest posterior probability among them to calculate the category to which it belongs.

5) Test Case Recommendation: The test case recommendation process is shown in figure 6. By inputting each candidate test case into the training classification model trained in step 3, we can predict the probability of defects in each test case and recommend key test cases with high defecting probability for users first, so that we can return the core keywords used to classification to users. Then the knowledge chain such as Software $\rightarrow$ Requirements $\rightarrow$ Test Items $\rightarrow$ Test Cases is returned to the testers to help the testers design test cases, thereby improving the efficiency and quality of software test.

IV. CASE STUDY

In order to demonstrate the effectiveness of the method, we implemented our method based on on Scikit-learn 0.21.2, Neo4j-3.3.5, Python3.7. And this study includes three processes: data collection, knowledge graph construction and interpretable recommendation.
A. Knowledge Graph Construction

We selected Apollo 1, which is a large-scale open-source autonomous driving system, as the target program. Apollo is written with C++ and Python programming language. It contains 5130 lines of python code and 57003 lines of C++ code. Apollo consists of a number of primary modules, including control, decision, drivers, localization, perception, planning, prediction and so on.

We made NLP(Natural Language Processing) word segmentation, BiLSTM + CRF-based entity labeling and relationship extraction based on the technical requirements and test documents of Apollo project. In these documents, we can achieve the function extraction in the technical requirements, function description extraction in the test documents, and also construct function graph, test items and test case graph respectively; For example, the description of the perception module and the entity relationship extraction are as follows:

```
Fig. 7. Functional description example 1
```

Then, based on text semantic similarity and graph structure similarity calculation, we associate functions with test items and test cases to form a test knowledge graph by using 1https://github.com/apolloauto entities, relations and sub graphs above. The graph is built as shown(partial):

```
Fig. 8. Knowledge Graph(partial)
```

Finally, a many-to-many relationship is established between “software projects (red entities in the figure 8)-software functions (purple entities in the figure 8)-software test cases (yellow entities in the figure 8)” by the knowledge graph. In the later stage of experimental verification, we can locate historical test cases quickly with the same or similar requirements using link reasoning of the knowledge graph based on this association. As a result, it can provide interpretable evidence for recommending the source of the test case and the sufficiency of the test.

B. Interpretable Recommendations

Taking the perception module as an example, there are 4 test cases in the knowledge graph. The test cases are described in Table I:

For the same sensing module, it can be disassembled into 4 test cases, including the identification of signal lights, people, objects and vehicles. In order to determine which test case is more likely to find hidden defects in the perception module test, the test case description of the perception module is used as the input and the test result is used as the output, then train the test case classification model with supervised learning method, which is used to accurately recommend failing test cases for later test cases.(test cases that make it easier to spot problems).

1) Test Case Combination Feature Mining Based on Apriori Algorithm: In terms of input data processing, in order to improve the accuracy of classification, we segmented the description sentence into words and made feature mining based on the Apriori algorithm to figure out features which can reflect the characteristics of test cases better. In this paper,
we set support confident value into 0.7, and uses the Apriori algorithm to perform the association mining results of the combined words on the segmentation results.

As can be seen from the figure, words such as pair, judgment, perceive, and proceed often appear together. Therefore, these words can be combined as a one-dimensional feature to improve the discrimination of the feature.

2) Information Gain Calculation: The selected combined words are used as features to calculate the information gain, and the words with high information gain are selected as the core words for training the classification model.

3) Classification Recommendation Model Test: After model training, if the task requirements in the perception module is to recognize pedestrians, traffic lights, road vehicle and make perceptual judgments, we can extract the entity relations of the words and phrases, then make subgraph matching with the knowledge graph and test item disassembly, and find possible recommended test cases in the knowledge graph respectively based on the disassembled test items. Finally, we can make predictions based on the trained classification model to recommend test cases that failed in history, that is, Pedestrians Identify.

4) Result Analysis: The system will prioritize the test case exception-004, which has been tested for problems in pedestrian recognition in history because in the historical test cases, this test case is more prone to defects than the traffic light and road vehicle test case.

V. DISCUSSION

We constructed the knowledge graph in the field of software testing. In the process of the graph construction, NER is a crucial step, occupying a lot of our work. In order to improve the accuracy of the NER model, we need to manually label the data in the early stage.

Compared with the traditional keyword-based matching model, the method of this paper incorporates semantic matching, and achieves the ranking of test cases according to the importance of the defect-finding. Therefore, the test cases recommended with our method are more accurate and useful for testers. Compared with the deep learning model, which improves the recommendation accuracy only by tuning the model parameters that are meaningless for testers, the knowledge graph simulates the way humans think, and discovers the hidden rules through abstracting concepts, which has better interpretability and can better assist testers to summarize the rules of testing and complete the design of test cases.

As verified, with the method proposed in this paper, test cases with similar functions can be recommended to testers finally. However, the number of projects verified in the method of this paper is small, and the project scale is small. Therefore, it is necessary to improve the construction of the domain test knowledge graph in the future to fully verify the accuracy of the recommended test cases.

VI. CONCLUSION

Facing an increasingly complex software environment, time constraints and heavy tasks have become the norm in software test. This paper proposes a test case recommendation method which can quickly find suitable test cases from a large number of test cases, and also provides certain auxiliary information to help testers understand the reason for recommending these test cases. In addition, the ability of checking errors in test cases is considered in our method, which can be pushed to testers who easily find defects in test cases. Unlike other methods, the method presented in our article is a content-based recommendation. By analyzing the key features that cause defects, programmers can better understand the mechanism of bug generation, and improve test efficiency. In addition, we can find out those special inputs, which are difference with the right, and this can help us generate more edge test cases.

REFERENCES