IOP Conf. Series: Materials Science and Engineering **392** (2018) 062194 doi:10.1088/1757-899X/392/6/062194

Research on Test Case Automatic Generation Algorithm Based on Scenario Method

Yue LI[1], HuaiBin WANG[1]

[1]Guangdong Communication Polytechnic, School of Information, Guangzhou Guangdong 510650, China;

Abstract. Scenario method is a common method in black-box testing. In the test of complex software, the amount of test case based on scenario method will be very large. This paper proposed a test case generation method that automatically calculates the similarity scenario, which can effectively reduce the test data while ensuring the test results and the validity of this algorithm is proved by experimental data.

1. Introduction

With the deepening of information technology in all industries, the quality of software is getting more and more important. As a significant means to ensure the software quality, software testing becomes more important too. Software testing refers to a series of processes that perform a system or program for discovering the errors. The process include a series of work such as setting up a test plan, designing test cases, executing testing process, reporting test results, and summarizing test work. In the traditional test work, most of the test work is mainly focused on the manual function test and the test work is mainly carried out by the tester to build the test environment, select the test case and execute the results manually. Under this test method, the test work has the following problems [1]:

- Due to the limitation of testers' ability, there may be missing part of the test content.
- Because test cases are generated by testers through test theory, there may be limitations in terms of quantity and completeness.
- In the case of large amount of test data, there may be errors due to fatigue or omission of the tester itself.

2. Automatic test

For software testing, the purposes of the test are as follows:

- Testing is the process of executing a program to discover errors in the program.
- A good test plan is a test plan that can detect errors that have not been found yet.
- A successful test is the discovery of an erroneous that has not been discovered.

A good test should be a reasonable test plan and through the design of good test case to find errors that others can not find. For manual testing, if the software is large in scale, or the application scenario is very complex, or the software iteration cycle is very fast, these external and objective factors can lead to a large number of repetitive test cases.The design and maintenance of these test cases requires a lot of human resources, and can not guarantee that the manual design test cases can achieve test coverage effective and find software errors as many as possible.

Automated testing is a concept opposite to manual testing. It mainly refers to the introduction of automated testing tools in the test work. The automated test tools can achieve a large amount of data, high intensity, and more extensive test coverage that cannot be achieved manually. In automated

testing, automation tools can be used to achieve automated functions, automated performance and many other test contents. This paper focuses on the design and automatic generation of test cases for automated function testing based on automated testing.

3. Scenario test case

3.1. Decomposition of test requirements

Software requirements are important benchmarks for developing test cases. In test work, we first need to extract test points according to software requirements, and then design test cases step by step based on test points[2].The original requirements can be obtained through software requirements specification, system design specification or user operation manual. By refining the original requirements, the point to be measured in the software can be expressed as the following model:

Test Requirements = {requirement ID, function name, test priority, test point, completion status}

- requirement ID: a unique code, which can uniquely identify a measured demand point from the requirement.
- function name: the core function corresponding to requirement identification to explain the corresponding demand point.
- test priority: the priority of the test execution in the actual test.
- test points: Briefly describe the functional test points corresponding to this requirement.
- completion status: Identifies the status of the demand point in the test work, which can be divided into algorithm selection, use case design, use case execution, test completion, etc.

3.2. Test Cases

Test case refers to a set of execution sequences during the software testing process [3]. The following points should be covered in the sequence, which can be expressed in the following model:

Test case = {use case number, function point, test environment, test data, test step, expected result, execution status}

- use Case Number: The unique code for this test case.
- function point: test case corresponds to which function point is to be tested.
- test environment: The environment required for the test.
- test data: Data needed for testing input.
- test procedure: the specific execution steps in the testing process.
- expected results: the expected response of the software under the data and steps.
- execution status: the state of the use case can be divided into non execution, execution and coincidence expectations, and execution is not consistent with expectations.

In manual function test, the testers focus on the selection of test data.

4. Automatic generation of scenario-based test cases

4.1. Test scenario model

Scenario method is a common test method in black box testing. The main point of the scenario method design test case lies in the process and data of a set of user operating software that may appear according to the user's possibility of using the software system operation sequence, combined with software functions. Application scenario method to design test cases, scenario based use case model is proposed for scenario based test cases:

The test scene = {scene number, scene description, the scene, the scene operation sequence, the subsequent scene}

- scene number: the unique number corresponding to the scene.
- scene description: simple text description of the scene.
- pre-order scenario: a possible previous operation for users to enter the scene in operation.

- scenario operation sequence: user's possible operation steps and operation contents in this scenario.
- follow up scenario: the next scenario that users may enter when they exit the scene.

The pre-order scene and subsequent scene are the collection of scene numbers, since the user may have multiple ways to enter a scene, there may be multiple possible scenarios after exiting a scene.

The scene operation series is a set of operation steps. In the scenario-based test case design, the test case should take into account all possible scenarios during the execution of the system, so the scenario operation series model is shown as follows:

Scene operation series = {basic operation stream sequence} + {optional operation stream sequence}

4.2. Automatic generation of test scenario algorithm

For a function that needs to be tested in the software system, it is assumed that there is a set N of all the pre-order scenes entering the function, N= $\{n_1, n_2.. n_n\}$, and a set S of pre-order scenarios that the user may appear on the function, S= $\{S_1, S_2,...S_{K+1}\}$, a set of follow up scenario M that may exist on this function, M= $\{m_1, m_2,..m_m\}$. for complete test view, all possible scenarios in the system should be tested. Therefore, there is a scenario-based test case set T, T=N×S ×M. After the calculation, all the test cases based on the scenario method can be obtained. The test case content is shown in the following table 1:

Table 1. Scenario-based test case

Scenario number	Scenario content
Scenario use case 1	Pre-order Scene 1 → full basic flow→subsequent Scene 1
Scenario use case 2	Pre-order Scene 1 → full basic flow→subsequent Scene 2
Scenario use case m	Pre-order Scene 1 → full basic flow→subsequent Scene m
……	Pre-order Scene 1 → basic flow + alternative flow 1 → subsequent scenario 1
……	Pre-order Scene 1 → basic flow + alternative flow 1 → subsequent scenario m
……	Pre-order Scene 1 → basic flow + alternative flow 2 → subsequent scenario 1
……	……
Scenario use case t	Pre-order Scene 1 → basic flow + alternative flow k → subsequent scenario m

In the case of a simple system function, scene composition and scene switching is not complicated, the above algorithm can be used to obtain a complete test of the scene combination that as far as possible to cover the user that may appear in the actual operation of the situation.However, for systems with complex system functions and complex scene switching scenarios, the use of this combination to obtain test cases will lead to an explosive growth in the number of tests, and will not be able to complete every test in a manual test environment. Scene testing, even if the introduction of automated testing tools to achieve automatic testing, will also lead to complex test scripts difficult to maintain, test workload and test results are not directly proportional to the impact. Therefore, this paper proposes a filtering algorithm based on the scene matrix, effectively reduces the test data through collaborative filtering, and achieves full test results as much as possible while reducing test overhead.

4.2.1. Similarity calculation of scene test case

In the actual testing work, the basic flow scene that does not contain the alternative flow is the core function that the system must complete. Therefore, the highest priority is found on the test priority, so the test case based on the scene needs to complete the complete test of the basic flow, without making filtering and filtering. The main source of the impact on test data is the presence of a large number of alternative streams in the test scenario, which will form a large number of test sets. Therefore, the focus of this algorithm is how to filter and filter the test scene containing the alternative flow. In the process of scene testing, each alternative flow is a branch generated on the basic flow path, which must be issued by a certain execution node on the basic stream, and the alternate flow is completed by the user executing a series of specific sequence of actions on the alternate stream. The two different alternate streams do not have exactly the same action between the nodes, but there may be a number of

different alternative flows from the same basic flow path node. For algorithm preparation, we need to refine the nodes on the basic flow and alternative streams.

For every test requirement, there is only one scenario based basic flow test case. Basic flow MR= {mr_1, mr_2... mr_n}, the basic flow is a set of node coded sets, where MR_N represents the operation node encoding on the complete basic flow under this scenario.

There are multiple scenarios based test alternatives for every test requirement. Alternative flow collection is expressed as $SR_{I,J}(K)$，I∈ {mr_1, mr_2... mr_n }, which indicates that the K alternative flow is issued by the mri node on the basic flow. J represents the operation node encoding on the alternate flow.

In the actual testing process, the basic flow contains multiple operation nodes, and the alternative flow also contains multiple operation nodes, but there may be a number of alternative streams with similar operation series between different alternative streams, that is, the similarity between the different alternative streams. The scenario based scenario combination automatically generates test cases, which will produce a large number of test data, which inevitably contains low quality test data that not only costs testing resources but also can not find more problems, and the test cases are filtered.

Collaborative filtering algorithm is a common filtering algorithm, which is often used in personalized recommendation system. In the recommendation system, because there is a lot of unknown data, it is necessary to create the most valuable TOP-N recommendation data for the current user through collaborative filtering algorithm [4]. Based on this idea, the collaborative filtering algorithm can be used to screen a large number of test cases automatically produced by combination. By calculating the similarity of different test cases, the most likely detection cases in similar test cases are selected for actual execution, thus reducing the actual cost of test work. Cosine distance is a commonly used method to calculate the similarity between different vectors by calculating the cosine values between vectors. It is the most widely used and relatively reasonable calculation method of computing overhead at present [5].

For each scene test case, cosine similarity values can be calculated, and the cosine similarity is calculated as follow:

$$\text{Sim(U,V)=Cos}(U,V) = \sum_{i \in R_{U,V}} r_{U,i} * r_{V,i} \left(\left(\sum_{i \in R_{U,V}} r_{U,i}^2 \sum_{i \in R_{U,V}} r_{V,i}^2 \right)^{1/2} \right)^{-1} \tag{1}$$

Pearson correlation coefficient is another common method to calculate the linear correlation between two variables, is calculated as follow:

$$\text{Sim(U,V)} = \sum_{i \in R_{U,V}} (r_{U,i} - \overline{r_U})(r_{V,i} - \overline{r_V}) \left(\left(\sum_{i \in R_{U,V}} (r_{U,i} - \overline{r_U})^2 \sum_{i \in R_{U,V}} (r_{V,i} - \overline{r_V})^2 \right)^{1/2} \right)^{-1} \tag{2}$$

For the calculated scene similarity data, the parameter K can be set. When the K is larger than a set value, it is considered that the scene use case under the similarity data can be merged, and the actual test work can be reduced and the test results are guaranteed as much as possible by the method of screening similar test scenes.

5. Experiment

The experiment uses an asset management system as the experimental platform, which includes 10 functional modules, such as personal information, asset management, department management and supplier etc, and 95 preposition defects.The ratios between the number of test cases filtered by the algorithm and the number of test cases in the fully combined scenario is the reduction rate. The lower value, the higher degree of reduction.The experiment uses the ratio of the number of defects found

B_{num} and the number of test cases T_{num} as the evaluation basis, called the defect detection rate. The larger the ratio, the smaller the workload to find a single defect, the higher the efficiency.

The experiment uses the ratio of the number of defects found and the number of preset errors in the module as the defect discovery rate. The larger the ratio, the more complete the defects found.

Experiments were performed on seven functional modules on the experimental platform. Cosine similarity calculation method and Pearson coefficient calculation method were used to screen the number of test cases. In the calculation process, the two similar algorithms used the same similarity screening parameters to test the reduction rate of different algorithms when calculating test cases.

From the perspective of reduction rate as figure 1, the reduction rates of the cosine similarity algorithm and the Pearson coefficient calculation method on the different modules are generally equivalent. The reduction rate of the cosine similarity algorithm is slightly lower than the Pearson coefficient calculation method, indicating the similarity degree. When the screening parameters are the same, the difference between these two calculation methods is not obvious, and a simpler cosine similarity algorithm can be used to achieve test case screening.

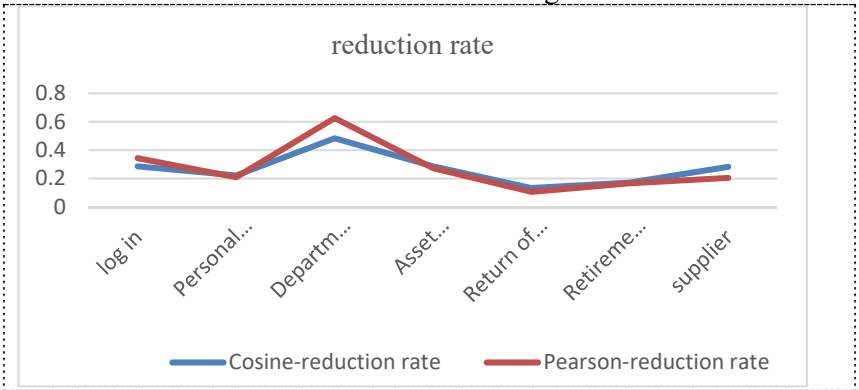

Figure 1. Comparison of reduction rate.

From the perspective of defect detection rate as figure 2, the overall performance of the Pearson coefficient calculation method on the seven different modules is slightly better than the cosine similarity algorithm, but the difference between these two algorithms is not obvious.

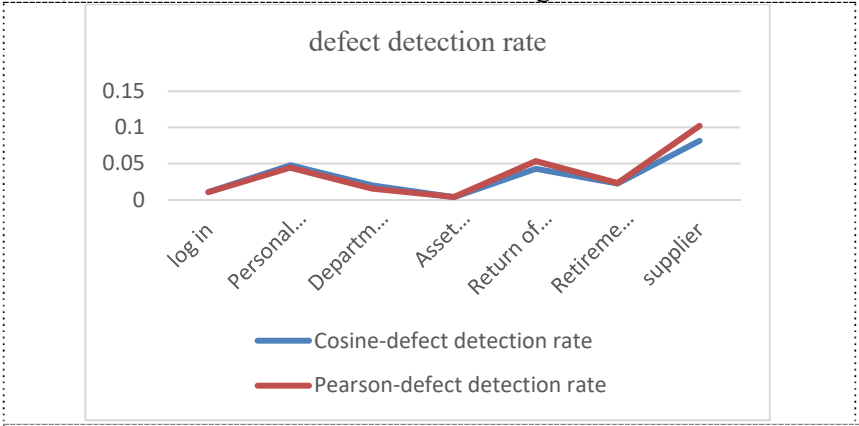

Figure 2. Comparison of defect detection rate.

From the defect discovery rate as figure 3, these two algorithms can find the software defects better in the system. The cosine similarity algorithm performs better on individual modules than the Pearson correlation coefficient calculation method.

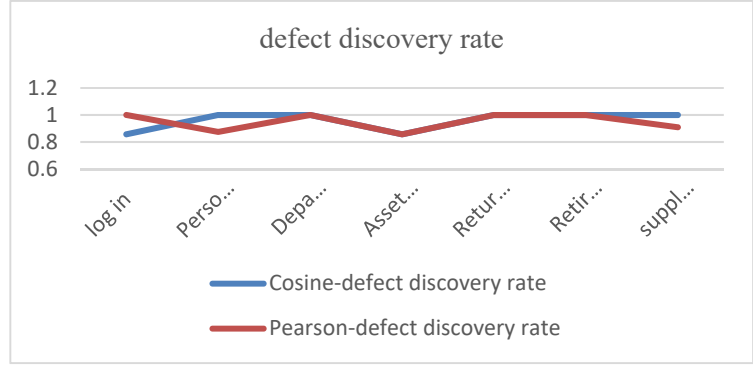

Figure 3. Comparison of defect discovery rate.

Through the above experimental data, we can see that through this scenario-based test case automatic generation algorithm, we can reduce the test work greatly and can discovery the software defects well at the same time. In the test case screening algorithm, the cosine similarity algorithm and the Pearson correlation coefficient calculation method are equivalent. Considering the three evaluation indexes such as approximate simplicity, defect detection rate and defect detection rate, cosine similarity can be selected as the final selection algorithm.

6. Conclusion

The number of defects that can be found in the system is directly related to the quality of test case. Although the system defects can be found as much as possible in complete combination method, there is also a large amount redundant use cases. Through modeling the scenario test case and calculating the similarity of scenario test use cases, it can remove a number of similar scenes and ensure testing quality.This algorithm can further study how to choose the value of similarity filtering parameter K to get more improved algorithm quality.

Reference:

[1] REN Lixia.Black box testing process of computer interlocking software [J].Railway Computer Application, 2018(02):39-43.

[2] YANG Jun,LU Caixia ,HUANG Chen ,WANG Ting.On the Application of Test Case Reusing in the Electronic Procurement Trading Platform[J].Computer & Digital Engineering, 2018(01):108-113.

[3] ZHANG Zhiyi, CHEN Zhenyu, XU Baowen, et al.Research Progress on Test Case Evolution [J]. Journal of Software, 2013, 24(4):663-674.

[4] Zhang-Hong, Wang Hui.Research on collaborative filtering algorithm based on user score and common score [J/OL]. Application Research of Computers. 2019, 36(1). [2018-01-10]. http://www.arocmag.com/article/02-2019-01-032.html.

[5] Qian Ren, Wu Yun, Kong GuangQian. Research on collaborative filtering algorithm based on sparse weighted[J/OL].Computer Technology and Development.2018,(06).[2018-02-24.]http://kns.cnki.net/kcms/detail/61.1450.TP.20180224.1521.074.html.