A Survey on Readability Improving Techniques on Automated Unit Test Case Generation

A. Sushma Deepthi
M. Tech. Student: dept. of CSE
JNTUACEA, Anantapur

***Abstract*--The goal of automated test generation tools is reducing the cost of testing activities. However generated tests are shown that are not helpful to developers in finding and detecting more bugs although they reach higher structural coverage compared to manual testing. The main reason is that generated tests are difficult to read, understand and maintain. Readability is a key factor to optimize in the context of automated test generation. For this problem various techniques are proposed to improve the readability of test cases. This paper presents a survey on readability improving techniques.**

Keywords— Test case; automated test generation; readability

I. INTRODUCTION

Software testing plays a key role in software development life cycle. However, testing is labor-intensive and expensive. It often accounts for more than 50% of total development costs. Various search-based techniques and tools have been proposed to reduce the time, developers need to spend on testing by automatically generating a set of test cases with respect to a specific test coverage criteria [1].Indeed, a recent study[2] reported that developers spend up to 50% of their time in understanding and analyzing the output of automatic tools. Automatically generated tests are not improving the ability of developers to detect faults when compared to manual testing [3]. Readability of test cases is a key factor to optimize in the context of automated test generation. However, the quality of the code composing the generated test cases e.g., input parameters, assertions, etc. is not the only factor affecting their comprehensibility. However, it is difficult to tell, without reading the contents of the target class, (I) what is the behavior under test, (ii) whether the generated assertions are correct, (iii) which if-conditions are finally traverse when executing the test (coverage).For this a solution is need to help developers to quickly understand both tests and code covered. Several techniques were proposed to improve the readability of the automated unit test case generation however in this paper focus only on few of the techniques. The following are the techniques

1. Incorporating a language model into search-based test input generation
2. Test case Adaptation
3. A technique to follow patterns of common object usage
4. The test describer approach

These are the some of the techniques used to improve the readability of automated unit test case generation, the remaining section of this paper will discuss in detail about the techniques of improving readability.

II. TECHNIQUES USED IN IMPROVING READABILITY

A. Incorporating A Language Model into Search-Based Test Input Generation

In this technique[4] applied a natural language model to the automatic generation of strings inputs, with the aim of generating readable tests that are easy for humans to comprehend. A language Model (LM) assigns a score to a string reflecting the "likeness" of a string. Once an input has been found to cover a branch, the LM component of the fitness function can be connected to improve string inputs from the perspective of human readability. An empirical study is conducted in that to evaluate the capabilities of the LM test input generation approach with human judgments. Programmers evaluate test cases for a series of 17 open source projects of java case studies. The results of this study shows that test inputs generated by the LM approach took significantly less time to evaluate for 10 java case studies, with medium to large effect sizes recorded in 6 case studies. For 3 java case studies, the accuracy of test input evaluation was significantly improved. finally the results of a human study in the language model technique is compared with a conventional, non-informed approach to generating branch-covering test suites, revealing cases where participants were both faster and more accurate in making oracle judgments and improves readability of strings without failing the test adequacy criterion

B. Test case Adaptation

Test case Adaptation [5] reuses the information available in existing test cases to automatically evolve test suites. A set of algorithms are proposed that can automatically evolve test suites.TCA requires inputs from software developers those are(1) the original and the modified version of the program(2) the test cases written for the original program and (3) the name of the test case to repair or the class to generate test cases for. TCA evolves the test suite by:
1) Analyzing the software changes by defining the original and modified version of software;

2) Adapting the test cases using the appropriate test evolution algorithms.
The following are the five algorithms are used for evolving test cases as a solution to support software developers. They are i. Repair Signature Changes ii. Test Class Hierarchies iii. Test Interface Implementations. iv. Test New Overloaded Methods. v. Test New Overridden Methods
The experimental results show that the approach can repair and generate many test cases, thus reducing the testing effort and improved the readability of test cases by repairing the test cases. TCA properly repairs 90% of the compilation errors, and generates test cases that cover the same amount of instructions of state of the art techniques. The test cases produced by TCA are complementary to the ones generated by other techniques, which indicates that TCA could be integrated with other approaches to improve testing results.

C. A Technique to follow Patterns of Common Object Usage

Test cases having a problem of methods, it can have implicit preconditions of which the developer might be aware but the test case generation tool is not [6]. Common object usage-patterns of object interaction as found in manually written code-to make generated test cases more similar to existing client code.
The general process is involved in this is start from the client node

i. Extract object usage models

ii. That reflects the usage of the software under test. These models are merged into API models

iii. Representing the usage of the entire API and then derive test cases

iv. That conforms to the API model and thus covers typical usage in the software under test.

By using this approach, the resulting test cases are shorter, reference fewer different classes, and violate fewer preconditions, making them altogether more understandable and more valuable. However no objective measurement for readability of test cases to date. Similarly it is not possible to directly measure violations of implicit preconditions, these preconditions are not explicit. Future work will include work on quantifying readability, and setting up benchmarks that allow evaluation of test generation techniques with respect to how they treat implicit preconditions.

D. The Test Describer Approach

Test describer[7] a novel approach to automatically generate natural language summaries of junit test cases and the portion of the target classes they are going to test and generates summaries for the portion of code exercised by each individual test case, thus providing a dynamic view of the code under test. In this approach it consists of four steps.

i. Test case generation -generate test cases using Evosuite [8]

ii. Test coverage analysis-Test Describer identifies the code implemented in each individual test case generated in the step one and collects the information that will be summarized in the following steps.

iii. Summary generation -Test Describer takes the collected information and generates a set of summaries at different levels of granularity: a global description of the class under test, each test case, a set of fine-grained descriptions of each test.

iv. Summary agreement-the extracted information and/or descriptions are added to the original test suite.

The results of this empirical study involving thirty human participants from both industry and academe. The impact of the generated test summaries on the number of bugs actually fixed by developers when assisted by automated test generation tools. Results of the study indicate that test describer substantially helps developers to find more bugs (twice as many) reducing testing effort and test case summaries do not intense how developers manage test cases in terms of structural coverage. It could be used to automatically document tests, improving their readability and understandability. Results of post-test questionnaire reveal that test summaries significantly improve the comprehensibility of test cases

III. CONCLUSIONS

This paper presents a survey on some of the techniques that are used to improve the readability of the test cases and each technique have been implemented with a different mechanism to reduce the effort of the testers and developers to understand the test cases and make them easy to read and maintain. The techniques discussed in this paper are very much worth for other testing aspects. However, these techniques are limited in these achievements so there is further work to be done on these techniques.

REFERENCES

[1] A. Cavarra, C. Crichton, J. Davies, A. Hartman, and L. Mounier. Using uml for automatic test generation. In Proc. of the International Symposium on Software Testing and Analysis (ISSTA). Springer-Verlag, 2002.
[2] G. Fraser and A. Arcuri. 1600 faults in 100 projects: automatically _finding faults while achieving high coverage with Evosuite. Empirical Software Engineering, 2013:611-639, 2015.
[3] M. Ceccato, A. Marchetto, L. Mariani, C. D. Nguyen, and P. Tonella. Do automatically generated test cases make debugging easier? An experimental assessment of debugging effectiveness and efficiency. ACM Trans. Software. Eng. Methodology, 25(1):5:1-5:38, 2015.
[4] S. Afshan, P.McMinn, and M. Stevenson, “Evolving readable string test inputs using a natural language model to reduce human oracle

cost," in Int. Conference on Software Testing, Verification and Validation(ICST), 2013,pp.352-361.

[5] M. Mirazaaghaei, F. Pastore, and M. Pezze,:Supporting test suite evolution through test case adaption," in IEEE Int, Conference on Software Testing, Verification and Validation(ICST) ,2012, pp.231-240.

[6] G. Fraser and A. Zeller, "Exploiting common object usage in test case generation," International Conference on Software Testing, Verification and Validation(ICST 2011), pp. 80–89.

[7] S Panichella, A Panichella, M Beller, A Zaidman H C Gall, "The Impact of test cases summaries on bug fixing performance: an empirical investigation", in Proc. 38th International Conference on Software Engineering (ICSE), 2016 , pp. 547-558.

[8] G. Fraser and A. Arcuri, "Evosuite: Automatic test suite generation for object-oriented software," in Symposium on the Foundations of Software Engineering,2011, pp 416-419.