Test Recommendation System Based on Slicing Coverage Filtering

Ruixiang Qian
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Yuan Zhao
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Duo Men
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Yang Feng
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Qingkai Shi
Hong Kong University of Science and Technology, Hong Kong, China
qingkaishi@gmail.com

Yong Huang
Zhenyu Chen
Mooctest Inc., Nanjing, China
chenzhenyu@mooctest.com

ABSTRACT

Software testing plays a crucial role in software lifecycle. As a basic approach of software testing, unit testing is one of the necessary skills for software practitioners. Since testers are required to understand the inner code of the software under test(SUT) while writing a test case, testers usually need to learn how to detect the bug within SUT effectively. When novice programmers started to learn writing unit tests, they will generally watch a video lesson or reading unit tests written by others. These learning approaches are either time-consuming or too hard for a novice. To solve these problems, we developed a system, named TeSRS, to assist novice programmers to learn unit testing. TeSRS is a test recommendation system which can effectively assist test novice in learning unit testing. Utilizing program slice technique, TeSRS has gotten an enormous amount of test snippets from superior crowdsourcing test scripts. Depending on these test snippets, TeSRS provides novices a easier way for unit test learning. To sum up, TeSRS can help test novices (1) obtain high level design ideas of unit test case and (2) improve capabilities(e.g. branch coverage rate and mutation coverage rate) of their test scripts. TeSRS has built a scalable corpus composed of over 8000 test snippets from more than 25 test problems. Its stable performance shows effectiveness in unit test learning.

Demo video can be found at https://youtu.be/xvrLdvU8zFA

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**.

KEYWORDS

Program Slice, Static analysis, Test recommendation, Test guidance

ISSTA '20, July 18–22, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8008-9/20/07.
https://doi.org/10.1145/3395363.3404370

ACM Reference Format:
Ruixiang Qian, Yuan Zhao, Duo Men, Yang Feng, Qingkai Shi, Yong Huang, and Zhenyu Chen. 2020. Test Recommendation System Based on Slicing Coverage Filtering. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18–22, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3395363.3404370

1 INTRODUCTION

With the development of software engineering in the past few decades, software has become more and more intelligent. With the architecture has become more complex, software is becoming more error-prone at the same time. Unit testing is an important white box test approach, which is almost a compulsory course to every software practitioners. When doing unit testing, testers are required to understand the inner code of the software under test(SUT) entirely in order to write unit test cases(test scripts) to detect latent bugs within the software. However, it is not easy for a novice to write test scripts in an effective and elegant way. It usually takes them a long-term to practise and learn to obtain the high level design idea of unit test cases. Therefore, how to efficiently guide a novice is of great significance.

According to the *State of Testing Survey 2019*[1] initiated by **Practitest** with about 1,000 participants from more than 80 countries, 65% participants learn testing by "just doing it", and more than 40% participants learn testing through formal training or certifications and courses. This survey implies that practice is a main approach of learning testing and testers are usually longing to improving test skills through formal software testing courses. However, these approaches are either time-consuming or too hard for a novice.

In addition to courses and practice, testers also utilize automated test generation tools(e.g. Evosuite[4]) as well as visualized coverage tools (e.g. Jacoco[5]) to assist testing. Automated test generation tools can generate test suites automatically with few or without human intervention. Coverage tools (e.g. Jacoco) can detect the coverage rate (e.g. branch coverage) of test suites, revealing the unreached parts of an SUT to testers in order to guide them refine test scripts. Although these tools can effectively help experienced testers improve test quality and reduce their workload, these tools are usually not suitable for novices due to lack of experience.

[1] https://qablog.practitest.com/wp-content/uploads/2019/06/STOT_2019_ver1_2.pdf

Therefore, we design and implement a system named TeSRS to save novice testers from the above situations. TeSRS is test recommendation system which is now deployed on MoocTest[2] platform. MoocTest has been devoted to education of software testing in the past few years. Through holding national software test competition[3], MoocTest has collected an enormous amount of test scripts. In order to get a higher score, participants usually need to modify their test scripts several times before final submission, which ensures the quality of these test scripts. Using these test scripts as raw material, we adopt static program slice[1, 10] technique to build a test snippets corpus. Since these test scripts are written for competition, some of them may more or less exist coupling. We slice coupled test scripts into fine-grained test snippets which are more suitable for recommendation. TeSRS associates each test snippet with test criteria (e.g. branch coverage) and provides online recommendation service. When test learners encounter a bottleneck while writing a test script, TeSRS will recommend associated test snippets to them. The recommended test snippets can help test learners improve test quality notably, which will eventually reflect in their score. Since test snippets are recommended while test learners are coding, it is much easier for them to understand the test logic behind these snippets. In this paper, we mainly make the following contributions:

- Building a test snippets corpus by slicing test scripts supplied by MoocTest, which consists of over 8000 test snippets from 25 testing problems.
- Proposing a test recommendation system, named TeSRS, to help test novices learn testing efficiently and effectively through interactive online test snippet recommendation.

2 TESRS

TeSRS is a test recommendation system now is embedded in the MoocTest WebIDE, which can recommend fine-grained and easy-to-understand test snippets to test learners online. The overview process of TeSRS is shown in Fig.1.

TeSRS is composed of two parts: (1) Test snippets corpus. We build the corpus in two phases: 1) Slice phase. Utilizing off-the-shelf program slice technique, we split the test scripts supplied by MoocTest into fine-grained and easy-to-understand test snippets. 2) Association phase. To associate each test snippet with test criteria, we compute test capability for each test snippet, and build the mapping relation between snippet and corresponding test criteria. (2) Interactive test recommendation component. When test learners encounter a bottleneck while writing test scripts, they can ask TeSRS for help. TeSRS will recommend top K test snippets to test learners according to the dissimilarity between test snippets and incomplete scripts. We will talk about these two parts elaborately in the following two subsections.

2.1 Test Snippet Corpus

Test snippet corpus is the basis of TeSRS. We have built a scalable corpus containing 8000 test snippets from 25 test problems. The building process can be divided into the following two phases:

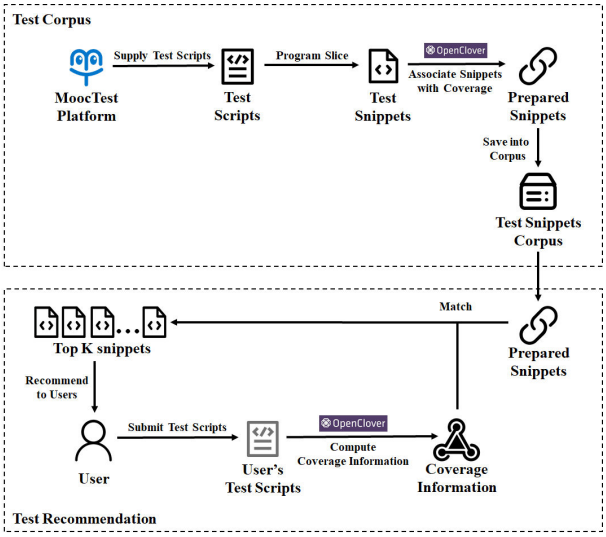

Figure 1: Overview of TeSRS

```
@Test(timeout = 4000)
public void test() {
    ALU alu = new ALU();
    assertEquals("00001001", alu.integerRepresentation("9", 8));
    assertEquals("11110111", alu.integerRepresentation("-9", 8));
    ...
}
```

Program Slice

```
@Test(timeout = 4000)
public void test0() {
    ALU alu = new ALU();
    assertEquals("00001001", alu.integerRepresentation("9", 8));
}
@Test(timeout = 4000)
public void test1() {
    ALU alu = new ALU();
    assertEquals("11110111", alu.integerRepresentation("-9", 8));
}
...
```

Figure 2: Test script to test snippets

Slice phase MoocTest has collected an enormous amount of superior test scripts(i.e. test scripts which got a high score in competition). All of these test scripts are Junit[4] tests from standard maven[5] projects. A example test script is shown at the top of fig.2. Although a test script may perform well in competition, it is usually coupled and hard to read, which are not suitable for test recommendation. Therefore, we use slice algorithm provided by Wala[1] to decouple test scripts and get more fine-grained test snippets. Wala is a static program analysis library for Java and Javascript, which also provide off-the-shelf backward and forward slice methods. Wala requires a statement, which is called "seed" in wala context, as the slice point to drive a slice process. Standard Junit test methods generally use static assert methods, such as *Assert.assertEquals*,

[2] http://www.mooctest.net
[3] http://www.mooctest.org/#/
[4] https://junit.org
[5] https://maven.apache.org/

to validate the correctness of outputs. An invocation of an assert method usually implies a completion of a testing phase. Therefore, we treat each assert method as a "seed" and slice test scripts in a backward style. The test snippets are shown at the bottom of fig.2.

Association phase This phase aims to bind each test snippet with some certain test criteria. In this paper, we bind each test snippet with branch coverage rate by OpenClover[6]. Openclover is a coverage report generation tool which can generate coverage report in xml format at runtime. Association is completed in four steps: (1) Construct blank test patterns. Since a test snippet are extracted from a test script, it must have the same dependency (e.g. outer classes which should be imported) as the corresponding script. For each test snippet, we firstly find its original project, and then locate the class where it comes from. After that, we construct a blank test pattern by deleting all the test methods within this class. Finally, we replace the pom.xml of this project by a pom.xml contains dependency for OpenClover. (2) Insert snippets into patterns. We firstly complete the method signature of each test snippet by JavaParser[6], and then insert it into blank test pattern accordingly. Through insertion, we get executable temporary test projects(test temp for short) and ready for collection. (3) Collect coverage information. We execute the test temps we get in (2) and run OpenClover to collect coverage report. (4) Associate test snippet with coverage rate. Since xml file is usually inconvenient for persistence and not suitable for recommendation, we transform coverage reports we get in (3) into JSON format, and then associate each test snippet with its JSON coverage information accordingly. These test composites will be finally saved into database.

2.2 Test Recommendation Component

Test recommendation component provide online recommendation service, the work flow is shown in fig.3. When test learners submit their test snippets on WebIDE, test recommendation component will collect coverage information from submitted test snippets and recommend test snippets to them timely. Test Snippets recommendation process consists of the following two phases:

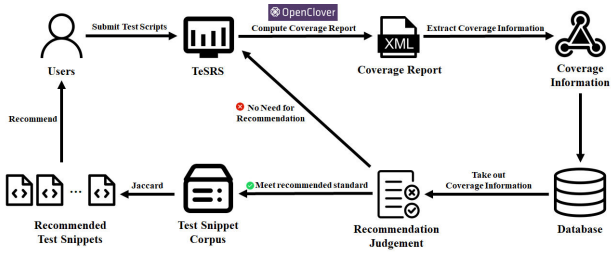

Figure 3: Recommendation Work Flow

Collection Phase When users submit and run their test scripts online, recommendation component will firstly utilize OpenClover to generate coverage report, which records coverage rate for each method under test(MUT). Then these coverage rates will be extracted respectively and stored into database according to MUTs.

Recommendation Phase Test recommendation component won't recommend test snippets to users at the very beginning. Recommendation service will be triggered when 1) the user has successively submitted for several times 2) and the number of submissions has excelled our predefined threshold N(which is one in this paper) and 3) the quality of test scripts, which is reflected by test score, has no improvement. Recommendation service will be completed in three steps. (1) Firstly, recommendation component finds all related test snippets according to the signature of each MUT. (2) Secondly, recommendation component builds test vectors for each test script which is submitted by user and test snippet which is taken out from corpus. A test vector v is a representation of coverage rate and each v correspond to a MUT. At present, we only adopt branch coverage as the accordance for recommendation. Therefore, the test vector of a MUT m is defined as $v_m = [b_1, b_2, b_3, ..., b_n]$ where n denotes the number of branches and $b_i (1 \leq i \leq n)$ denotes the evaluation result of a branch of m. The value of $b_i (1 \leq i \leq n)$ is $true$ or $false$, representing whether the ith branch of m has been covered or not. We use v_{msc} and v_{msn} to represent test vector of a script and a snippet respectively. (3) Finally, we compute Jaccard distance between v_{msc} and v_{msn} and recommend top k test snippets according to it. The math formula of Jaccard distance is shown below:

$$d_j = 1 - \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

A and B are both sample sets. Jaccard distance measures the dissimilarity between two sets. We treat each test vector as a set, then the Jaccard distance of v_{msc} and v_{msn} is:

$$d_j = 1 - \frac{|v_{msc} \cap v_{msn}|}{|v_{msc} \cup v_{msn}|} = 1 - \frac{l}{n}$$

Note that l denotes the number of branches shared by v_{msc} and v_{msn}, then d_j can be simplified to the ratio of the number of different branches and the number of total branches. Recommendation component will recommend top K test snippets to user to help them to improve test score. We set K to 2 at present. If there are more than 2 test snippets got the highest d_j, which means they are worthy recommending equally, component will randomly select 2 test snippets to recommend. The insight behind this strategy is that each of these test snippet is able to improve test score by covering new branches, so it doesn't matter which 2 will be recommended to users.

3 EXPERIMENT

3.1 Research Questions

To evaluate the effectiveness of TeSRS, we summarize key problems into the following research questions and design experiment to answer them respectively:

- **RQ1:** Can test snippets maintain the test quality of their origin test scripts?
- **RQ2:** Can TeSRS help test novice with test learning effectively?

3.2 The Reliability of Test Snippets

We select four test problems(**AStar**, **ALU**, **Triangle** and **RBTree**) to answer RQ1. To verify whether test snippets maintain the high quality of their original test scripts, we measure the test criteria(i.e. branch coverage and mutation coverage) of each test script(as

[6] https://openclover.org

shown in "origin" column) and test snippet(as shown in "slice" column) respectively. We make test snippets executable through the following operations: (1) Wrap test snippets into independent test methods. (2) Add outer dependencies (i.e. import statements like *import java.lang.**) and (3) add inner dependencies (e.g. inner classes and static methods). These operations are mainly done by a simple program. However, loss of several lines of code will inevitably occur when program is relatively complicate(e.g. has exception handler statements). We mended these loss manually. The experiment result is shown in table.1 We can find test quality doesn't change in terms

Table 1: Test criteria before and after slicing

	Origin		Slice	
Problem	Branch	Mutation	Branch	Mutation
Astar	81.0%	75%	81.0%	72%
ALU	81.7%	79%	81.7%	79%
RBTree	92.0%	83%	92.0%	82%
Triangle	81.2%	78%	81.2%	78%

of branch coverage, while tiny decrease in mutation coverage can be observed on some test problems(i.e **Astar** and **RBTree**). This is because program slice may destroyed some test logic while decoupling original test scripts. Some latent bugs need specific test sequences to trigger. However, since the decrease in percentage is not that huge and users will reunion these test snippets while they are coding with help of TeSRS, we will create similar test sequence so that the tiny loss of mutation coverage rate won't affect the quality of recommended test snippets.

3.3 The Effectiveness of TeSRS

We ask 20 students to respectively finish one test problem with the help of TeSRS and collect their feedback in the form of questionnaire We mainly investigated the following aspects: (1) Test script and test snippet, which one is more readable, and (2) user experience. In terms of the first aspect, we give out 3 test scripts and 3 test snippets. For each script or snippet, participant should give out a score about its readability from 1 to 5. Higher score implies a higher readability. As a result, test scripts got **180** in total while test snippets got **253**. This indicates program slicing does improve the readability of recommendation content by filtering out less important code. In terms of the second aspect, we ask every participants to evaluate TeSRS in two aspects: (1) To what extent can TeSRS widen their test idea? (2) To what extent can TeSRS help them with unit testing learning. Participants also need to give out a score in 1 to 5. As a result, all participants give out a score larger than 3 for both of these two aspects. Moreover, half of the participants give 5 point when evaluate the first aspect, 12 participants give a 5 at the second evaluation. Both of these two evaluation result indicates the effectiveness of our system.

4 RELATED WORK

Code recommendation can help developers find desired code from thousands of software artifacts and improve users' productivity. This problem is generally treated as an context-based query problem in information retrieval(Antunes et al. [3]). Rahman and Roy[9] exploit code examples collected from *GitHub* to help developers build a software with robust exception handle mechanism. In 2019, Ai et al.[2] proposed a code statement sequence information based recommendation system, named SENSORY, to help developers to implement unfamiliar programming tasks. Instead of directing developers to write a test script properly, all of the above techniques focus on assisting developers to use APIs better.

In terms of test code recommendation, Janjic and Atkinson.[7] leverage lessons learned from traditional software reuse to proactively make test recommendation. The core of their recommendation system is SENTRE(Search-ENhanced Testing with REuse), which can make reverse search for Junit test cases. Pham et al.[8] assist test novices in test learning by strategically showing them contextual test code examples from a project's test suite. Although these two techniques share similar insight with our approach, they did not utilize program slice technique to refine the code for recommendation or provide test learners a online platform for practice.

5 CONCLUSION

In this paper, we propose a test recommendation system, named TeSRS, to help test novices learn unit tests effectively and efficiently. We built a test snippet corpus, which comprises over 8000 test snippets from more than 25 test problem, by slicing the test scripts supplied by MoocTest. We also implemented a test recommendation component which can interactively recommend suitable test snippets to users. The reliability of test snippets and the effectiveness of recommendation component has been demonstrated through experiments. In the future, we plan to enlarge our test snippet corpus and further complete recommendation mechanism.

ACKNOWLEDGEMENTS

This work is partially supported by the National Natural Science Foundation of China(61802171) and the Key Project of Natural Science Research in Anhui Higher Education Institutions(KJ2019ZD67).

REFERENCES

[1] 2019. *T.J. Watson Libraries for Analysis.* https://github.com/wala/WALA/wiki
[2] Lei Ai, Zhiqiu Huang, Weiwei Li, Yu Zhou, and Yaoshen Yu. 2019. SENSORY: Leveraging Code Statement Sequence Information for Code Snippets Recommendation. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 27–36.
[3] Bruno Antunes, Barbara Furtado, and Paulo Gomes. 2014. Context-based search, recommendation and browsing in software development. In *Context in Computing*. Springer, 45–62.
[4] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.
[5] Marc R Hoffmann, B Janiczak, and E Mandrikov. 2011. Eclemma-jacoco java code coverage library.
[6] Roya Hosseini and Peter Brusilovsky. 2013. Javaparser: A fine-grain concept indexing tool for java problems. In *CEUR Workshop Proceedings*, Vol. 1009. University of Pittsburgh, 60–63.
[7] Werner Janjic and Colin Atkinson. 2013. Utilizing software reuse experience for automated test recommendation.
[8] Raphael Pham, Yauheni Stoliar, and Kurt Schneider. 2015. Automatically recommending test code examples to inexperienced developers. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 890–893.
[9] Mohammad Masudur Rahman and Chanchal K Roy. 2014. On the use of context in recommending exception handling code examples. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 285–294.
[10] Mark Weiser. 1984. Program slicing. *IEEE Transactions on software engineering* 4 (1984), 352–357.