Generating API-usage Example for Project Developers

Zixiao Zhu[1,2], Yanzhen Zou[1,2,†], Yong Jin[1,2], Bing Xie[1,2]
1: Software Institute, School of Electronics Engineering and Computer Science, Peking University
2: Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, 100871, China
{zhuzx11, zouyz, jinyong11, xiebing}@sei.pku.edu.cn

ABSTRACT
Usage examples have been shown very helpful for API learning in software reuse. Nowadays, many approaches have been proposed to automatically extract usage examples from client code or web pages for API users. However, they overlooked the benefit of API developers in example publishing and few works paid attention to help API developers to generate usage examples automatically. In this paper, we proposed an approach to generate API-usage example based on test code before the project are released. It analyzed which parts in test code are important for indicating API-usage and summarized some test code patterns, then a heuristic slice algorithm are proposed to extract referential test code as API-usage example based on these patterns. In the experiments, we gave some case studies on the commons-lang3 open source software library. It proved that our approach can provide good assistance for developers in APIs usage example generation.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – *reusable libraries.*

General Terms
Algorithms, Documentation.

Keywords
Software reuse, usage example, test code, test scenario.

1. INTRODUCTION
Invoking the APIs provided by third parties (e.g. open source libraries) has been proved to be an important way of software reuse. To understand the functionality and usages of the APIs, users usually want to get good API-usage examples [1-2]. For this purpose, quite a few approaches have been proposed to automatically extract API-usage examples from API client code [3-8] or some web pages [9-11]. However, we found that these works are helpful to API users but not API developers: 1) They overlooked the benefit of API developers in providing examples. 2) They paid no attention to provide good assistance for developers in API usage example generation. Few client code and related web pages can be found before the API library is widely used. Therefore, API developers still have to write API-usage examples manually before these APIs are released.

To deal with this issue, we investigated the developing process of some API libraries and found that test code is a good resource for demonstrating API-usage. This idea is also proved by Nasehi in ICSM'10 [10]. There are some inherent advantages of extracting API-usage from test code. First, test code usually indicates executable and self-contained usages of invoked APIs. Second, test code are written by API developers and exactly show the typical and designed API usages meeting developers' expectations.

However, mining API usages from test code are different from mining client code repositories. In this paper, we use the concept of **test scenario** to refer to a self-contained code snippet that containing an independent testing process of API. In our pilot study, we found that a test method might contain many similar test scenarios that are difficult to distinguish. As shown in Fig. 1. there are two test scenarios in the test method of "*testKeySetByValue*". Lines 2-4 are the declaration of some data objects. Lines 5-13 depict a test scenario that contains a usage of some APIs such as *keySetByValue*, *put*, and *getKey*. Lines 14-22 depict another test scenario, which contains a usage similar to the usage in the previous scenario. It is necessary to separate the scenarios. But traditional code slicing techniques might fail in this case because those slicing strategies are data-flow based and the scenarios that share the same data objects are data-dependent.

```
1  public void testKeySetByValue() {
2     BinaryTree m = new BinaryTree();
3     LocalTestNode nodes[] = makeLocalNodes();
4     Collection c1 = new LinkedList();
      ...
5     m = BinaryTree.initial();
6     c1.clear();
7     for (int k = 0; k < nodes.length; k++){
8        m.put(nodes[ k ].getKey(), nodes[ k ]);
9        if (k % 2 == 1)
10          c1.add(nodes[ k ].getKey());
11    }
12    assertTrue(m.keySetByValue().retainAll(c1));
13    assertEquals(nodes.length / 2, m.size());
      ...
14    m = BinaryTree.initial();
15    c1.clear();
16    for (int k = 0; k < nodes.length; k++){
17       m.put(nodes[ k ].getKey(), nodes[ k ]);
18       if (k % 2 == 0)
19          c1.add(nodes[ k ].getKey());
20    }
21    assertTrue(m.keySetByValue().removeAll(c1));
22    assertEquals(nodes.length / 2, m.size());
      ...
23 }
```

Fig. 1. Two test scenarios in a test method

Interentware'13, October 23 - 24 2013, Changsha, China.

† Corresponding Author.

Table 1. Types of statements in test codes

Statement Type	Description	Examples
Declaration	Variable declaration statement that contains no method invocations.	`CharSet set;` `int index;`
Common Invocation	A statement that contains common method invocations (excluding the invocation of test target or assertion method). This is the most common type.	`set = CharSet.getInstance("a-d");`
Target Invocation	At least one method invocation in this statement is proved to be the test target of this test method.	`array = set.getCharRanges();`
Definition	Another statement type that contains no method invocations. Assignment expression is required.	`index = i - 1;`
Test Assertion	The statement that contains invocations of assertion methods. The assertion methods usually start with special words like "assert", "test", "check", "fail".	`assertEquals(array[index],` `set.getCharAt(i));`
Other	Any statement not belonging to above types.	`break;`

In this paper, we proposed a code slicing approach to identify proper API-usages from test code. It firstly built up the AST(Abstract Syntax Trees) of test code. Then applied an improved code slicing approach to separate different test scenarios in test methods and identified the test code segments as API-usages. Our improved code slicing approach is based on the patterns we summarized from the test code. In the experiments, we gave some case studies on the *commons-lang*3 open source software library. It indicates that our approach can extract good usage references for more than 60% of API methods in the commons-lang3 software libraries.

2. APPROACH

In this section, we describe our approach on slicing API-usage examples from test code. First we introduce the statements types and basic idea of our slicing approach. Second, we present our code slicing algorithm. Based this, we separate the code segments of each test scenario as the candidate API-usage examples of the target API.

In general, a test scenario consists of three phases: data preparation(data declaration, data definition, e.g.), test execution (target invocation, common invocation, e.g.) and result assertion. In a test method containing multiple test scenarios, these three phases recur regularly. Therefore, the basic idea of our approach is to discover the recurrence of three test phases and identify each paragraph of a three-phase test scenario.

To characterize the role of each statement in the test method, we define some types for unit test code statements. As shown in table 1, we in total define 6 code statement types: *declaration, definition, common invocation, target invocation, test assertion,* and *others*. And we also give a brief description and some examples for each statement type.

Our slicing approach is proposed by combining the conventional code slicing method with some heuristic rules based on code patterns in test code. We need to deal with how to identify code characteristics in a test method and how to slice it with the pattern-based rules.

As shown in Fig. 2, we described the code statement sequence in a test method based of code statement types. Then we defined usage example code pattern on this statement sequence and provided a set of heuristic rules to identify the corresponding code segment. In this code statement sequence, *Declaration* is a special node at the beginning of a test method. Each statement type may repeat and we omit the arrow pointed to himself. In the statement sequence, transition @ means a new cycle of data preparation, test target invocation and result assertion. It indicates that the test scenario under this cycle should begin with some *Declaration statements*, followed closely by data *Definition* statement and a sequence of invocation statements contained at least one *Target Invocation*. Then the last part is one or more *Test Assertion* statements.

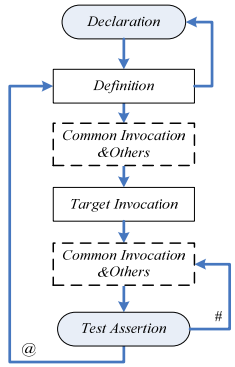

Fig. 2. Code state diagram in a test method

We use the following strategy for extracting the responding test scenario:

Step 1: Separate source code of the test method into groups by the edge @ and mark the nodes of target invocations in each code group.

Step 2: For each code group, slice out all the data-relevant statements. The pseudocode of this procedure is in Fig. 3. This algorithm takes the code group as input and produces all the relevant statements with the API usages.

Step 3: Extract the declaration statements for all the variables used in the output of Step 2. The variable declaration together with the data-relevant statements in a code group form a separated test scenario.

The output of the slicing algorithm is a set of code snippets from each test method. Each code snippet contains a separated and self-contained test scenario. As example candidates, these code snippets can be effectively handled with an existing approach to extracting API-usage examples from client code. For developers, they can adapt these candidate examples and add efficient API examples into official documents with less effort. In our approach, we also have implemented a prototype tool for embedding API usage examples into Javadoc.

Algorithm 1: Slice data-relevant statements in code group
Input: Code group *G*. List of *target invocation*s in *G*, de noted by *T*.
Output: List of data-relevant statements *S*.
Procedure:

```
1   let S←Ø
2   let V←Ø
3   for each statement s in T do
4       S←S∪{s}
5       R←getRelevantVariables(s)
6       for each variable v used in R do
7           V←V∪{v}
8   for each statement s' in G in reverse order do
9       R'←getRelevantVariables(s')
10      if R'∩V≠Ø then
11          S←S∪{s'}
12          for each variable v' in R' do
13              V←V∪{v'}
14  Output: S
```

Fig. 3. Pseudo code of slicing data-relevant statements

3. EVALUATION

In our experimental study, we want to investigate and answer the following two research questions.

✧ Whether our approach can slice the right segments from test code as usage example?

✧ Whether our approach can assist developers to produce sufficient examples for APIs?

Based on these ideas, we apply our approach to the open-source Java projects *commons-lang3* [11]. *Commons-lang3* is a library of common programming functionalities, an extension of standard Java library java.lang. In this library, there are totally *1704* test methods and *1346* APIs. We generate a raw data set by sampling *250* test methods because we cannot afford to manually creating a Golden Standard Dataset (GSD for short) for the project. If the separation result of the approach is more similar with the GSD, the approach is more effective.

RQ1: Slicing Precision

As there exists quite a few approaches to extracting API examples with a code slicing procedure, we compared our approach with a state-of-art approach named eXoaDocs [3] [4] in our experimental study. We re-implemented the code slicing algorithm of the eXoaDocs approach for investigating the effectiveness of our approach. We also named our approach as UsETeC (Usage Examples from Test Cases) in Fig. 4.

In Fig. 4. , we show the percent of standard code snippets extracted at each min-EDs. ***Min-ED*** means the minimum edit distance between the standard snippet and any sliced snippet in the result set. It indicates that our approach, UsETeC, gains higher precision than eXoaDocs in slicing API-usage examples. In detail, more than 45% slicing results of our approach is completely same with GSD set. However, only about 2% slicing results of eXoaDocs approach is same with GSD set. This advantage lasts until the mid-ED reaches 5. When the minimum edit distance between the standard snippet and the sliced snippet increases to 5, the eXoaDocs approach exceed our approach for the first time. However, this situation just indicates the goodness of our approach in precision. *UsETeC* can extract nearly 80% of the human annotated slices in *GSD* when *min-ED* are less than 5. Compared with it, the results of *eXoaDocs* can only reach 25% of *GSD* set.

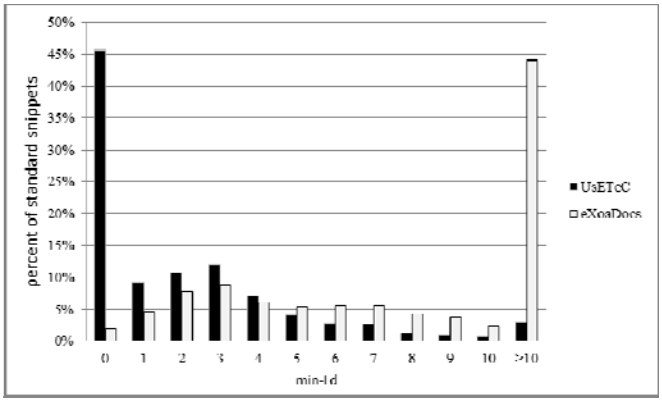

Fig. 4. Precision of test code slicing

RQ2: Is it useful for API-usage

Being motivated by providing API examples when libraries are just released, we want to evaluate the coverage of API examples that we can extract from unit tests.

```
1. public void testRandomStringUtils(){
2. String r1=RandomStringUtils.random(50);
3. assertEquals("random(50)
     length",50,r1.length());
4. String r2=RandomStringUtils.random(50);
5. assertEquals("random(50)
     length",50,r2.length());
6. assertTrue("!r1.equals(r2)",!r1.equals(r2));
7. r1=RandomStringUtils.randomAscii(50);
8. assertEquals("randomAscii(50)
     length",50,r1.length());
9. for (int i=0; i < r1.length(); i++) {
     assertTrue("char between 32 and
     127",r1.charAt(i) >= 32 && r1.charAt(i) <=
     127);
10.}
11.r2=RandomStringUtils.randomAscii(50);
12.assertTrue("!r1.equals(r2)",!r1.equals(r2));
     ...
13. }
```

```
String r1=RandomStringUtils.random(50);
assertEquals("random(50) length",50,r1.length());
```

```
String r1=RandomStringUtils.random(50);
String r2=RandomStringUtils.random(50);
assertEquals("random(50) length",50,r2.length());
assertTrue("!r1.equals(r2)",!r1.equals(r2));
```

```
String r1=RandomStringUtils.random(50);
r1=RandomStringUtils.randomAscii(50);
assertEquals("randomAscii(50) length",50,r1.length());
for (int i=0; i < r1.length(); i++) {
  assertTrue("char between 32 and 127",r1.charAt(i) >=
32 && r1.charAt(i) <= 127);
}
```

```
String r1=RandomStringUtils.random(50);
String r2=RandomStringUtils.random(50);
r2=RandomStringUtils.randomAscii(50);
assertTrue("!r1.equals(r2)",!r1.equals(r2));
```

Fig. 5. An example of our approach for in commons-lang3 project

For *commons-lang3*, we generate 1243 usage examples for *1346* APIs. These usage examples are related to *843* APIs in this project. That is to say, more than *62%* APIs in *commons-lang3* project can be automatically generated helpful usage examples with our approach. Notes that we treat the public method in source code as an API, this might lead to over count of designed programming interfaces. So the actual coverage of APIs is probably higher.

In Fig. 5. , we illustrate some sliced code results from the test method of *testRandomStringUtils*. Considering the limit of space, we only give the first 12 lines in this test method and 4 usage examples are extracted from these test codes. It can be seen from the results that the *Declaration*/*Definition* of "*r2*" are correctly extracted out and included in the second usage example. In our work, these sliced results will be clusted and ranked further and only the top ones are recommend to users

4. DISCUSSION AND FUTURE WORK

The main strength of heuristic code slicing algorithm is to be more specific to different code patterns and achieve more accurate result. However, they also have some weakness: the most important one is the completeness of code patterns. Obviously we are not able to enumerate all the code patterns existed in different test methods. Our code patterns have an overall coverage of 81%. We do not aim to achieve a high coverage of patterns because over-detailed patterns may lead to the second weakness: the mismatch between test code and patterns. Some test code differs with each other quite slightly, but sometimes the slight difference might be the key part in a code snippet. The refinement of our approach will be an important part in our future work.

In future work, we plan to further investigate the following issues.

First, we plan to implement the other modules of extracting API-usage examples for developers. We will focus on the ranking of code snippets, human-interaction with developers and the representation of examples. Second, we plan to refine the code slicing algorithm by improve the separating strategies for each code pattern. Third, inspired by the result from test code, we plan to survey the feasibility of extracting API-usage examples from source code of software libraries.

5. RELATED WORK

Our work is closely related to the following works: mining usage example from client code or web pages.

Base on client code, many API specification mining approaches were proposed to help programmers use API. Buse et al. [5] proposed an approach of automatically synthesizing human-readable usage examples by mining API's client code repository. Kim et al. [3] extract code examples of a specific API from source repository of open source projects and code examples are embedded into API's documentations.

Some approaches harvesting API examples from online technical web pages were also proposed, such as Assieme [9], and APIExample [8]. Such an approach collects API's related web pages by leveraging a general search engine. Then automatically identify and extract sample code snippets and related textual descriptions from pages as well as.

These related works provide great improvement on reducing the burden faced by programmers when using APIs. However, the approaches that are based on client code or technical documentations can hardly provide support for the APIs from newly released frameworks or libraries. The work in our paper fills in such gap and makes usage examples available even when APIs are fresh.

Nasehi et al. [10] conducted an empirical study to verify the practicality of extracting API-usage examples from test code. Their work does provide us much inspiration. But in their paper, how to generate easy-to-use usage examples was not presented. In this paper, we provide a series of techniques to effectively extract usage examples based on separating test scenarios.

6. CONCLUSION

In this paper, we have proposed an approach to extracting API-usage examples for developers' document purpose. We summarize the code patterns of test method and give a heuristic slicing algorithm. In an experimental study, we compared our approach with eXoaDocs in precision. It indicates that our approach is more accurate than eXoaDocs. Furthermore, our approach can provide considerable amount of usage examples for APIs in libraries.

7. ACKNOWLEDGMENT

This research is sponsored by the National Natural Science Foundation of China under Grant No. 61103024, the High-Tech Research and Development Program of China (Grant No. 2012AA011202), the National Basic Research Program of China (973) under Grant No. 2009CB320703.

8. REFERENCES

[1] M. P. Robillard, "What makes apis hard to learn? answers from developers," IEEE Software, vol. 26, no. 6, pp. 27-34, November 2009.

[2] C. Scaffidi, "Why are APIs difficult to learn and use," ACM Crossroads, Vol. 12, No. 4, pp. 4-9, May 2006.

[3] J. Kim, S. Lee, S. W. Hwang, S. Kim. "Adding examples into java documents," *in Proc.* ASE, 2009, pp. 540-544.

[4] J. Kim, S. Lee, S. W. Hwang, S. Kim. "Enriching Documents with Examples: A Corpus Mining Approach," ACM Transactions on Information Systems, 2013, 31(1), pp. 1.

[5] R. P. L. Buse, W. Weimer, "Synthesizing API-usage examples," in Proc. ICSE, 2012, pp. 782-792.

[6] L W Mar, Y C Wu, H C Jiau. "Recommending Proper API Code Examples for Documentation Purpose," in Proc. APSEC, 2011, pp. 331-338.

[7] J Kim, S Lee, S Hwang, S Kim, "Towards an intelligent code search engine," in Proc. AAAI Conference on Artificial Intelligence. 2010.

[8] L Wang, L Fang, L Wang, G Li, B Xie, "APIExample: An effective web search based usage example recommendation system for java APIs," in Proc. ASE, 2011, pp. 592-595.

[9] R. Hoffmann, J. Fogarty, and D. S. Weld. "Assieme: finding and leveraging implicit references in a web search interface for programmers," in Proc. UIST, 2007, pp. 13-22.

[10] S M Nasehi, F. Maurer, "Unit tests as API-usage examples," in Proc, ICSM, 2010, pp. 1-10.

[11] http://commons.apache.org/proper/commons-lang/, accessed in March 2013