Block 5

.

Enhancing the Usability and Integration of Automatic Test Generation Tools

Background

There exist several tools that automatically generate unit test cases for source code. These tools can generate test cases in a resource efficient manner and augment existing manually written test cases. However, automatically written test cases tend to have poor readability due to obscure identifier names and a lack of documentation, which can incur a significant maintenance effort once these tests are integrated into a codebase.

Researchers have developed several tools and techniques to address this issue. In this survey, we will evaluate these approaches.

This Survey

The goal of this survey is to evaluate several research tools developed to enhance the readability of automatically written tests. Please note that this survey does **not** aim to evaluate the presented automatically written test cases.

This survey starts with a small section regarding basic demographic information. After this, you will be presented with several versions of automatically generated tests across 4 sections. You will be asked to evaluate each version on certain criteria. A version of the automatically generated test case in its original form will also be presented to you for reference.

The survey contains **17 questions** and will take **10-15 minutes** to complete. **16** of these questions are multiple choice. There are also optional feedback forms intended to allow you to elaborate on your answers if appropriate.

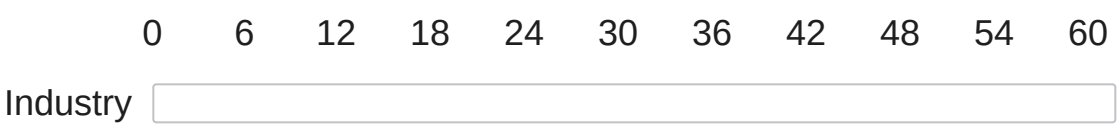
Thank you for taking the time to participate in this survey!

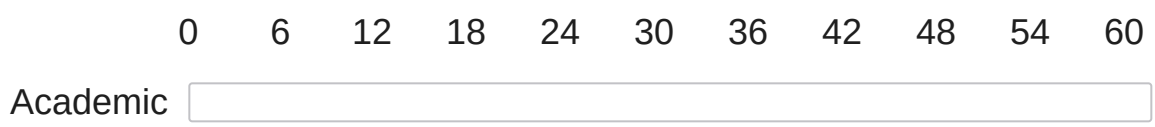
Default Question Block

Q1.1. What is your primary profession?

- ◯ Software Developer
- ◯ Student (Undergraduate)
- ◯ Student (Graduate)
- ◯ Academia (Professor or Researcher)

Q1.2. How many years of experience do you have with Java (in years)?





Q1.3. Do you use automatic test case generation frameworks, such as Evosuite?

- ◯ Yes
- ◯ No

Q1.4. Do you use such automatic test case generation tools in projects you develop?

- ◯ Yes
- ◯ No

Q1.5. Please elaborate on which tools you use. If you don't use any test case generation tools, why not?

Section 1

.

Section 1: Summaries

In this section, you will be asked to evaluate the quality of the test case summary generated by two different tools.

. First, we present to you the original automatically generated test case

.

```
1.
2. @Test
3. public void test10() throws Throwable {
4.     ArrayIntList arrayIntList0 = new ArrayIntList();
5.
6.     try {
7.         arrayIntList0.add(0, 0);
8.         arrayIntList0.add(0, 1);
9.         arrayIntList0.add(0, 2);
10.        assertEquals(3, arrayIntList0.size());
11.        arrayIntList0.removeElementAt((1));
12.        fail("Expecting exception: IndexOutOfBoundsExcept
13.    } catch (IndexOutOfBoundsException e) {
14.            //
15.            // Should be at least 0 and less than 0, foun
16.            //
17.    }
18. }
```

A). The following is the same unit test but along with the summary generated by the first tool (Listing **A**)

.

```
1. /**
2.  * OVERVIEW: The test case "test10" covers around 2.0% (l
3.  * statements in "ArrayIntList"
4.  **/
```

```
 5. @Test
 6. public void test10() throws Throwable {
 7.     // The test case instantiates a "ArrayIntList"  with
 8.     // configuration (initial capacity is 8)
 9.     ArrayIntList arrayIntList0 = new ArrayIntList();
10.
11.     try {
12.         // The next method call removes the element at in
13.         // "arrayIntList0"
14.         arrayIntList0.add(0, 0);
15.         arrayIntList0.add(0, 1);
16.         arrayIntList0.add(0, 2);
17.         assertEquals(3, arrayIntList0.size());
18.         arrayIntList0.removeElementAt((1));
19.         fail("Expecting exception: IndexOutOfBoundsExcept
20.     } catch (IndexOutOfBoundsException e) {
21.             //
22.             // Should be at least 0 and less than 0, foun
23.             //
24.     }
25. }
```

B). The following is the same unit test but along with the summary generated by the second tool (Listing **B**)

.

```
1. /**
2.  * 1. Creates a new ArrayIntList
3.  * 2. Adds to "arrayIntList" 3 times and checks if its si
```

```
 4.  * 3. Expects an IndexOutOfBoundsException when calling
 5.  *         removeElement on "arrayIntList0" with argument
 6.  **/
 7. @Test
 8. public void test10() throws Throwable {
 9.         ArrayIntList arrayIntList0 = new ArrayIntList();
10.
11.         try {
12.                 arrayIntList0.add(0, 0);
13.                 arrayIntList0.add(0, 1);
14.                 arrayIntList0.add(0, 2);
15.                 assertEquals(3, arrayIntList0.size());
16.                 arrayIntList0.removeElementAt((1));
17.                 fail("Expecting exception: IndexOutOfBoun
18.         } catch (IndexOutOfBoundsException e) {
19.                 //
20.                 // Should be at least 0 and less than 0,
21.                 //
22.         }
23. }
```

Q2.1. How would you rate the **conciseness** of the two summaries?

	Contains no unnecessary information	Contains some unnecessary information	Contains mostly unnecessary information
Listing **A**	◯	◯	◯
Listing **B**	◯	◯	◯

. (Optional) Please elaborate on your answer above if appropriate.

Q2.2. How would you rate the **content** of the two summaries provided?

	Not missing any important information	Missing some important information	Missing some very important information
Listing **A**	◯	◯	◯
Listing **B**	◯	◯	◯

. (Optional) Please elaborate on your answer above if appropriate.

Q2.3. How would you rate the **readability** of the two summaries provided?

	Easy to read and understand	Somewhat easy to read and understand	Difficult to read and understand
Listing **A**	◯	◯	◯
Listing **B**	◯	◯	◯

. (Optional) Please elaborate on your answer above if appropriate.

Q2.4. Considering conciseness, content and readability, which summary do you prefer to use?

- ◯ Summary in Listing **A**
- ◯ Summary in Listing **B**
- ◯ Neither

. (Optional) Feel free to leave any comments for the two proposed summaries here.

Section 2

.

Section 2

In this section, you will be asked to evaluate the quality of the test case name suggested by two tools.

. We present the original test case here for reference

.

```
1.   @Test
2.   public void test13()  throws Throwable  {
3.       ClassWriter classWriter0 = new ClassWriter((-18));
4.       classWriter0.visitAnnotation("", false);
5.       classWriter0.toByteArray();
6.       assertEquals(3, classWriter0.index);
7.   }
```

3.1. How would you rate each of the **two suggested test case names** on the basis of their ability to convey the intent of the test case?

	Fully captures the intent of the test case	Mostly captures the intent of the test case	Somewhat captures the intent of the test case	ca ir c c
testVisitAnnotationWithNonEmptyStringAndFalse	◯	◯	◯	
testVisitAnnotation	◯	◯	◯	

. (Optional) Please elaborate on your answer above if appropriate.

3.2. How would you rate the two suggested test case names on the basis of their **naturalness**?

	Easy to read and to understand	Somewhat easy to read and understand	Difficult read an understa
testVisitAnnotationWithNonEmptyStringAndFalse	◯	◯	◯
testVisitAnnotation	◯	◯	◯

. (Optional) Please elaborate on your answer above if appropriate.

Q3.3. Which test case name do you prefer?

- ◯ **testVisitAnnotationWithNonEmptyStringAndFalse**
- ◯ **testVisitAnnotation**

◯ Neither

. (Optional) Feel free to leave any comments for the two proposed test case names here.

Section 3

.

Section 3

In this section, you will be asked to evaluate the quality of the suggested variable names

. We present the original test case here for reference

.

```
1. @Test(timeout = 4000)
2. public void test03()  throws Throwable  {
3.     AnsiStyle ansiStyle0 = AnsiStyle.NORMAL;
4.     String string0 = AnsiOutput.encode(ansiStyle0);
5.     Object[] objectArray0 = new Object[5];
6.     String string1 = AnsiOutput.toString(objectArray0);
7.     assertTrue(string1.equals((Object)string0));
8. }
```

. Here is a version of the same test but with renamed variables:

.

```
1.   @Test(timeout = 4000)
2.   public void test03()  throws Throwable  {
3.       AnsiStyle style = AnsiStyle.NORMAL;
4.       String encoded = AnsiOutput.encode(style);
5.       Object[] data = new Object[5];
6.       String result = AnsiOutput.toString(data);
7.       assertTrue(result.equals((Object)encoded));
8.   }
```

Q4.1. How would you rate the **variable names** used for the code snippet above?

	Fully conveys the intended usage of the variable	Somewhat conveys the intended usage of the variable	Does not convey the intedend usage of the variable	Does not convey the intended usage of the variable, and is misleading
style	◯	◯	◯	◯
encoded	◯	◯	◯	◯
data	◯	◯	◯	◯
result	◯	◯	◯	◯

. (Optional) If you think that the above variable names need improvements, please suggest more appropriate names.

Section 4

. **Section 4**

In this last section of the survey, you will be presented with a version of the test case that includes the generated summaries, method names and variable names.

A). We present the original version of the test case here for reference:

.

```
1. @Test(timeout = 4000)
2. public void test3()  throws Throwable  {
3.     LoggerGroups loggerGroups0 = new LoggerGroups();
4.     HashMap<String, List<String>> hashMap0 = new HashMap<
5.     LinkedList<String> linkedList0 = new LinkedList<Strin
6.     linkedList0.add(">9z<ZrVT\"dk");
7.     hashMap0.put("", linkedList0);
8.     loggerGroups0.putAll(hashMap0);
9.     LoggerGroup loggerGroup0 = loggerGroups0.get("");
10.    assertNull(loggerGroup0.getConfiguredLevel());
11. }
```

B). Here is the version of the test case using all the transformations:

.

```
1. /**
2.  * 1. Creates a new LoggerGroups "logger", new HashMap
3.  *    a new LinkedList.
```

```
 4.  * 2. Adds to the LinkedList and puts into the
 5.  *    HashMap.
 6.  * 3. Puts all HashMap into "logger" and checks if
 7.  *    the configured level of "logger" is null.
 8.  */
 9. @Test(timeout = 4000)
10. public void testGetGroup()  throws Throwable  {
11.     LoggerGroups logger = new LoggerGroups();
12.     HashMap<String, List<String>> expected = new HashMap<
13.     LinkedList<String> string = new LinkedList<String>();
14.     string.add(">9z<ZrVT\"dk");
15.     expected.put("", string);
16.     logger.putAll(expected);
17.     LoggerGroup result = logger.get("");
18.     assertNull(result.getConfiguredLevel());
19. }
```

Q5.1. How would you rate the overall **improvement in readability** in the code snippet 4-B (enhanced using the proposed tool) over code snippet 4-A (original automatically generated test)?

- ◯ Significant increase in readability
- ◯ Minor increase in readability
- ◯ No change in readability
- ◯ Minor decrease in readability
- ◯ Significant decrease in readability

. (Optional) Please elaborate on your answer above if appropriate.

Q5.2. If you were to use automatically generated unit tests, how likely are you to also use the proposed tool to transform the generated test cases?

- ◯ Extremely likely
- ◯ Somewhat likely
- ◯ Neither likely nor unlikely
- ◯ Somewhat unlikely
- ◯ Extremely unlikely

. (Optional) Please elaborate on your answer above if appropriate.

Q5.3. With the existence of the proposed tool, how likely are you to utilize automatically written tests in projects you work on?

- ◯ Extremely likely
- ◯ Somewhat likely
- ◯ Neither likely nor unlikely
- ◯ Somewhat unlikely
- ◯ Extremely unlikely

. (Optional) Please elaborate on your answer above if appropriate.

Q5.4. Please rank the features of the tool that you found useful (if any) in the context of automatically generated test cases, in order of relative importance.

Items	Useful
Test Case Summaries	

Variable Renaming	
Method Renaming	

Not Useful

. (Optional) Please elaborate on your answer above if appropriate.

Block 4

. (Optional) Feel free to leave any final comments here.

. We appreciate you taking the time to participate in our survey. Please go to the next page to finalize and submit the survey.