Understandable Test Generation Through LLM-generated summaries

Background
Unit test cases can be generated by several tools. One can achieve a high coverage, but it can be hard to understand because of the names of the variables and lack of documentation, for example. There exists for this several tools which serve as a solution to improve understandability.

This survey
In this survey, we will evaluate the impact on understandability by looking at summaries created by different tools. This survey does **not** aim to evaluate the automatically written test cases.

The first part is about your experience with Java and test generation frameworks. The second part is split into four rounds, with the first two rounds having 4 different test cases, and the last two rounds having 3 different test cases. It starts with the original test case without any summaries and/or comments for reference.

This survey contains 20 questions and will take ~15 minutes to complete. There are 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!

Part I - Background

Q1.1 What is your primary profession?*
- Student (Bachelor's)
- Student (Masters)
- Student (PhD)
- Software Developer

Q1.2 How many years of experience do you have with Java?*
- Industry (practical in the working life):
- Academic (for school):

Part II - Summaries

Round 1

In this section, you will be asked to evaluate the quality of the summaries created by different tools.

First, we present you the original automatically generated test case in each round, followed by the four summaries A, B, C, and D.

Original test

```java
@Test
public void test10() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();

    try {
        arrayIntList0.add(0, 0);
        arrayIntList0.add(0, 1);
        arrayIntList0.add(0, 2);
        assertEquals(3, arrayIntList0.size());
        arrayIntList0.removeElementAt((1));
        fail("Expecting exception: IndexOutOfBoundsException");
    } catch (IndexOutOfBoundsException e) {
        //
        // Should be at least 0 and less than 0, found -1
        //
    }
}
```

Methods being tested

```java
public int removeElementAt(int index) {
    checkRange(index);
    incrModCount();
    int oldval = _data[index];
    int numtomove = _size - index - 1;
    if(numtomove > 0) {
        System.arraycopy(_data,index+1,_data,index,numtomove);
    }
    _size--;
    return oldval;
    }
```

```java
private final void checkRange(int index) {
    if(index < 0 || index >= _size) {
        throw new IndexOutOfBoundsException("Should be at least 0 and less than " + _size + ", found " + index);
    }
}
```

Method called

```
114     public void add(int index, int element) {
115         checkRangeIncludingEndpoint(index);
116         incrModCount();
117         ensureCapacity(_size+1);
118         int numtomove = _size-index;
119         System.arraycopy(_data,index,_data,index+1,numtomove);
120         _data[index] = element;
121         _size++;
122     }
```

Here below are the summaries A, B, C, and D.

A:

```
1   /**
2    * OVERVIEW: The test case "test10" covers around 2.0% (low percentage) of
3    * statements in "ArrayIntList"
4    */
5   @Test
6   public void test10() throws Throwable {
7           // The test case instantiates a "ArrayIntList"  with the default
8           // configuration (initial capacity is 8)
9           ArrayIntList arrayIntList0 = new ArrayIntList();
10
11          try {
12              // The next method call removes the element at index -1 of
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: IndexOutOfBoundsException");
20          } catch (IndexOutOfBoundsException e) {
21              //
22              // Should be at least 0 and less than 0, found -1
23              //
24          }
25  }
```

B:

```java
/**
 * 1. Creates a new ArrayIntList
 * 2. Adds to "arrayIntList0" 3 times and checks if its size is
 * 3. Expects an IndexOutOfBoundsException when calling
 *    removeElementAt on "arrayIntList0" with argument 1
 **/
@Test
public void test10() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();

    try {
      arrayIntList0.add(0, 0);
      arrayIntList0.add(0, 1);
      arrayIntList0.add(0, 2);
      assertEquals(3, arrayIntList0.size());
      arrayIntList0.removeElementAt(1);
      fail("Expecting exception: IndexOutOfBoundsException");
    } catch (IndexOutOfBoundsException e) {
      //
      // Should be at least 0 and less than 0, found -1
      //
    }
}
```

C:

```java
/**
 * Tests adding elements to an ArrayIntList and then removing an element at an invalid index,
 * expecting an IndexOutOfBoundsException to be thrown.
 */
@Test
public void test10() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();

    try {
        arrayIntList0.add(0, 0);
        arrayIntList0.add(0, 1);
        arrayIntList0.add(0, 2);
        assertEquals(3, arrayIntList0.size());
        arrayIntList0.removeElementAt((1));
        fail("Expecting exception: IndexOutOfBoundsException");
    } catch (IndexOutOfBoundsException e) {
        //
        // Should be at least 0 and less than 0, found -1
        //
    }
}
```

D:

```java
/**
 * The test method `test10` tests the `removeElementAt()` method of `ArrayIntList`.
 * The test case adds three elements to the list and asserts that the size is 3.
 * Then, it tries to remove an element at index 1, which is out of bounds, expecting an `IndexOutOfBoundsException`.
 * The test case verifies that the expected exception is thrown with the appropriate message.
 */
@Test
public void test10() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();

    try {
        arrayIntList0.add(0, 0);
        arrayIntList0.add(0, 1);
        arrayIntList0.add(0, 2);
        assertEquals(3, arrayIntList0.size());
        arrayIntList0.removeElementAt((1));
        fail("Expecting exception: IndexOutOfBoundsException");
    } catch (IndexOutOfBoundsException e) {
        //
        // Should be at least 0 and less than 0, found -1
        //
    }
}

```

Q2.1 How would you rate the **content** of the four summaries?*

	(1) Missing much important information or information mostly incorrect	(2)	(3) Missing some important information or information is somewhat incorrect	(4)	(5) Not missing any important information or no incorrect information
A					
B					
C					
D					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.2 How would you rate the **conciseness** of the four summaries?*

	(1) Contains mostly unnecessary information	(2)	(3) Contains some unnecessary information	(4)	(5) Contains no unnecessary information
A					
B					
C					
D					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.3 How would you rate the **naturalness** of the four summaries?*

	(1) The flow and tone are not smooth and appropriate at all	(2)	(3) The flow and tone are somewhat smooth and appropriate	(4)	(5) The flow and tone are smooth and appropriate

A					
B					
C					
D					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.4 Considering conciseness, content, and naturalness, which summary do you **prefer** to use?* (Multiple possible)

- Summary A
- Summary B
- Summary C
- Summary D
- None

Q2.5 What **aspects** do you like the most and the least in the four summaries? Please also write the corresponding tool with it. Examples are shown below, but you are encouraged to come up with your own.*

~ Example: I like using a numbered list like in summary B the most.

~ Examples of aspects: using a numbered list, length of summary, having the code coverage included, in-line comments, explaining what it does, step-by-step explanation of how it works, etc

- Aspect(s) liking the most: [Open]
- Aspect(s) liking the least: [Open]

(Optional) Feel free to leave any comments for the 4 proposed summaries here.

- [Open]

Round 2

As in *round 1*, we present you the original automatically generated test case in each round, followed by the four summaries A, B, C, and D. Please note that a specific summary from round 1, like summary A, is **not** necessarily made with the same tool as summary A from this round.

```
1  @Test
2  public void test3() throws Throwable {
3      Rational rational0 = new Rational(1L, 3215L);
4      Rational rational1 = rational0.abs();
5      assertEquals(1L, rational0.numerator);
6      assertEquals(3215L, rational0.denominator);
7      assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
8  }
```

Method being tested

```
68  public Rational abs() {
69      return new Rational((numerator < 0L) ? -numerator : numerator, (denominator < 0L) ? -denominator : denominator).reduce();
70  }
```

A:

```java
/**
* Tests the absolute value function of the Rational class to ensure it
* correctly handles positive values and returns the expected float representation.
*/
@Test
public void test3() throws Throwable {
    Rational rational0 = new Rational(1L, 3215L);
    Rational rational1 = rational0.abs();
    assertEquals(1L, rational0.numerator);
    assertEquals(3215L, rational0.denominator);
    assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
}
```

B:

```java
/**
 * OVERVIEW: The test case "test3" covers around 6.0% (low percentage) of
 * statements in "Rational"
 **/
@Test
public void test3() throws Throwable {
    // The test case instantiates a "Rational" with numerator equal to 1L,
    // and denominator equal to 3215L.
    // The execution of this constructor implicitly covers the following 1
    // conditions:
    // - the condition " denominator equals to 0L" is FALSE;
    Rational rational0 = new Rational(1L, 3215L);
    // The test case declares an object of the class "Rational" whose value
    // is equal to the absolute value of "rational0"
    Rational rational1 = rational0.abs();
    // Then, it tests:
    // 1) whether the numerator of rational0 is equal to 1L;
    assertEquals(1L, rational0.numerator);
    // 2) whether the denominator of rational0 is equal to 3215L;
    assertEquals(3215L, rational0.denominator);
    // 2) whether the float value of "rational1" is equal to 3.11041E-4F
    // with delta equal to 0.01F;
    assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
}
```

C:

```java
/**
 * Creates a new instance of the "Rational" class with numerator 1 and denominator 3215.
 * Calls the "abs()" method on the created rational number, which returns a new rational number with the absolute value of the original number.
 * Asserts that the new rational number has a numerator equal to 1.
 * Asserts that the new rational number has a denominator equal to 3215.
 * Converts the new rational number to a float value and asserts that it is approximately equal to 3.11041E-4F, with a tolerance of 0.01F.
 */
@Test
public void test3() throws Throwable {
    Rational rational0 = new Rational(1L, 3215L);
    Rational rational1 = rational0.abs();
    assertEquals(1L, rational0.numerator);
    assertEquals(3215L, rational0.denominator);
    assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
}
```

D:

```java
/**
 * 1. Creates 2 Rational objects, "rational0" and "rational1"
 * 2. Checks the numerator and denominator of "rational0" and the
 *    float value of "rational1"
 **/
@Test
public void test3() throws Throwable {
    Rational rational0 = new Rational(1L, 3215L);
    Rational rational1 = rational0.abs();
    assertEquals(1L, rational0.numerator);
    assertEquals(3215L, rational0.denominator);
    assertEquals(3.11041E-4F, rational1.floatValue(), 0.0);
}
```

Q2.6 How would you rate the **content** of the four summaries?*

	(1) Missing much important information or information mostly incorrect	(2)	(3) Missing some important information or information is somewhat incorrect	(4)	(5) Not missing any important information or no incorrect information
A					
B					
C					
D					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.7 How would you rate the **conciseness** of the four summaries?*

	(1) Contains mostly unnecessary information	(2)	(3) Contains some unnecessary information	(4)	(5) Contains no unnecessary information
A					
B					
C					
D					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.8 How would you rate the **naturalness** of the four summaries?*

	(1) The flow and tone are not smooth and appropriate at all	(2)	(3) The flow and tone are somewhat smooth and appropriate	(4)	(5) The flow and tone are smooth and appropriate

A					
B					
C					
D					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.9 Considering conciseness, content, and naturalness, which summary do you prefer to use?* (Multiple possible)

- Summary A
- Summary B
- Summary C
- Summary D
- None

Q2.10 What **aspects** do you like the most and the least in the four summaries? Please also write the corresponding tool with it. Examples are shown below, but you are encouraged to come up with your own.*

~ Example: I like using a numbered list like in summary B the most.

~ Examples of aspects: using a numbered list, length of summary, having the code coverage included, in-line comments, explaining what it does, step-by-step explanation of how it works, etc

- Aspect(s) liking the most: [Open]
- Aspect(s) liking the least: [Open]

(Optional) Feel free to leave any comments for the 4 proposed summaries here.

- [Open]

Round 3

As in the previous rounds, we present you the original automatically generated test case in each round, but instead of followed by four summaries, these two last rounds will be followed by three summaries A, B, and C. Please note that a specific summary from previous rounds, like summary A, is **not** necessarily made with the same tool as summary A from this round.

Original test

```java
@Test(timeout = 4000)
public void testEncodedPath()  throws Throwable  {
    KeycloakUriBuilder uri = KeycloakUriBuilder.fromPath("x");
    HashMap<String, Integer> map = new HashMap<String, Integer>();
    URI result = uri.buildFromEncodedMap(map);
    assertEquals("x", result.getRawPath());
}
```

The summaries A, B, and C are below.

A:

```java
/**
 * The test method tests the `buildFromEncodedMap` method of the `KeycloakUriBuilder` class.
 * The test creates a `KeycloakUriBuilder` instance with the path "x" and adds an empty HashMap to it.
 * The `buildFromEncodedMap` method is called on the `KeycloakUriBuilder` instance,
 *  and the resulting URI is asserted to have the raw path "x".
 */
@Test(timeout = 4000)
public void testEncodedPath()  throws Throwable  {
    KeycloakUriBuilder uri = KeycloakUriBuilder.fromPath("x");
    HashMap<String, Integer> map = new HashMap<String, Integer>();
    URI result = uri.buildFromEncodedMap(map);
    assertEquals("x", result.getRawPath());
}
```

B:

```java
/**
 * Tests that KeycloakUriBuilder correctly builds a URI from an encoded map,
 * ensuring that the raw path remains unchanged.
 */
@Test(timeout = 4000)
public void testEncodedPath()  throws Throwable  {
    KeycloakUriBuilder uri = KeycloakUriBuilder.fromPath("x");
    HashMap<String, Integer> map = new HashMap<String, Integer>();
    URI result = uri.buildFromEncodedMap(map);
    assertEquals("x", result.getRawPath());
}
```

C:

```java
/**
 * 1. Creates a new KeyCloakUriBuilder "uri" from path
 * 2. Creates a new  HashMap and uses it to create a new URI "result" using
 *    method "buildFromEncodedMap" of "uri"
 * 3. Checks if the raw path of "result" equals "x"
 */
@Test(timeout = 4000)
public void testEncodedPath()  throws Throwable  {
    KeycloakUriBuilder uri = KeycloakUriBuilder.fromPath("x");
    HashMap<String, Integer> map = new HashMap<String, Integer>();
    URI result = uri.buildFromEncodedMap(map);
    assertEquals("x", result.getRawPath());
}
```

Q2.11 How would you rate the **content** of the three summaries?*

	(1) Missing much important information or information mostly incorrect	(2)	(3) Missing some important information or information is somewhat incorrect	(4)	(5) Not missing any important information or no incorrect information
A					
B					
C					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.12 How would you rate the **conciseness** of the three summaries?*

	(1) Contains mostly unnecessary information	(2)	(3) Contains some unnecessary information	(4)	(5) Contains no unnecessary information
A					
B					
C					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.13 How would you rate the **naturalness** of the three summaries?*

	(1) The flow and tone are not smooth and appropriate at all	(2)	(3) The flow and tone are somewhat smooth and appropriate	(4)	(5) The flow and tone are smooth and appropriate
A					
B					

C					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.14 Considering conciseness, content, and naturalness, which summary do you **prefer** to use?* (Multiple possible)

- Summary A
- Summary B
- Summary C
- None

Q2.15 What **aspects** do you like the most and the least in the three summaries? Please also write the corresponding tool with it. Examples are shown below, but you are encouraged to come up with your own.*

~ Example: I like using a numbered list like in summary B the most.

~ Examples of aspects: using a numbered list, length of summary, having the code coverage included, in-line comments, explaining what it does, step-by-step explanation of how it works, etc

- Aspect(s) liking the most: [Open]
- Aspect(s) liking the least: [Open]

(Optional) Feel free to leave any comments for the 3 proposed summaries here.

- [Open]

Round 4

As in the previous rounds, we present you the original automatically generated test case in each round. This last round will be followed by three summaries A, B, and C as it was in *round 3*. Please note that a specific summary from previous rounds, like summary A, is **not** necessarily made with the same tool as summary A from this round.

Original test

```
1   @Test
2   public void test8() throws Throwable {
3       ArrayIntList arrayIntList0 = new ArrayIntList();
4       arrayIntList0.ensureCapacity(9);
5       assertEquals(0, arrayIntList0.size());
6   }
```

Method being tested

```
124     public void ensureCapacity(int mincap) {
125         incrModCount();
126         if(mincap > _data.length) {
127             int newcap = (_data.length * 3)/2 + 1;
128             int[] olddata = _data;
129             _data = new int[newcap < mincap ? mincap : newcap];
130             System.arraycopy(olddata,0,_data,0,_size);
131         }
132     }
```

Methods called

```
68      public ArrayIntList() {
69          this(8);
70      }
```

```
90      public int size() {
91          return _size;
92      }
```

The summaries A, B, and C are below.

A:

```java
/**
 * OVERVIEW: The test case "test8" covers around 3.0% (low percentage) of
 * statements in "ArrayIntList"
 **/
@Test
public void test8() throws Throwable {
    // The test case instantiates a "ArrayIntList" with the default
    // configuration (initial capacity is 8)
    ArrayIntList arrayIntList0 = new ArrayIntList();
    // The next method call the ensure capacity in min cap 9 to
    // arrayIntList0
    // The execution of this method call implicitly covers the following 1
    // conditions:
    // - the condition " min capacity greater than _data.length" is TRUE;
    arrayIntList0.ensureCapacity(9);
    // Then, it tests:
    // 1) whether the size of "arrayintlist0" is equal to 0;
    assertEquals(0, arrayIntList0.size());
}
```

B:

```java
/**
 * Create a new instance of ArrayIntList
 * Ensure the list has enough capacity for 9 elements
 * Assert that the size of the list is 0 (empty)
 */
@Test
public void test8() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();
    arrayIntList0.ensureCapacity(9);
    assertEquals(0, arrayIntList0.size());
}
```

C:

```java
/**
* Ensures that the capacity of an ArrayIntList can be increased to a specified value without affecting its size.
*/
@Test
public void test8() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();
    arrayIntList0.ensureCapacity(9);
    assertEquals(0, arrayIntList0.size());
}
```

Q2.16 How would you rate the **content** of the three summaries?*

	(1) Missing much important information or information mostly incorrect	(2)	(3) Missing some important information or information is somewhat incorrect	(4)	(5) Not missing any important information or no incorrect information
A					
B					
C					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.17 How would you rate the **conciseness** of the three summaries?*

	(1) Contains mostly unnecessary information	(2)	(3) Contains some unnecessary information	(4)	(5) Contains no unnecessary information
A					
B					
C					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.18 How would you rate the **naturalness** of the three summaries?*

	(1) The flow and tone are not smooth and appropriate at all	(2)	(3) The flow and tone are somewhat smooth and appropriate	(4)	(5) The flow and tone are smooth and appropriate
A					
B					

C					

(Optional) Please elaborate your answer if appropriate.

- [Open]

Q2.19 Considering conciseness, content, and naturalness, which summary do you **prefer** to use?* (Multiple possible)

- Summary A
- Summary B
- Summary C
- None

Q2.20 What **aspects** do you like the most and the least in the three summaries? Please also write the corresponding tool with it. Examples are shown below, but you are encouraged to come up with your own.*

~ Example: I like using a numbered list like in summary B the most.

~ Examples of aspects: using a numbered list, length of summary, having the code coverage included, in-line comments, explaining what it does, step-by-step explanation of how it works, etc

- Aspect(s) liking the most: [Open]
- Aspect(s) liking the least: [Open]

(Optional) Feel free to leave any comments for the 3 proposed summaries here.

- [Open]

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

- [Open]