

Please note that this survey requires at least a 13" monitor and cannot be completed on a mobile phone.

Consent Form

By completing this online questionnaire (i.e., clicking “Next”), you agree to participate in the study.

The study focuses on investigating the type and amount of information needed for debugging program failures. The study is structured in the form of an online questionnaire with 12 questions, which are split into three parts. Each part includes questions about the same piece of code. The entire survey is expected to take around 20-30 minutes of your time.

As a “thank you”, we will randomly select 10 participants who finished the study to receive a **\$30 Amazon gift card**. Moreover, your participation in this study will help the academic and industrial community gain valuable insight into debugging approaches. The results will be fully anonymized and will be reported in an academic paper, which will be openly available to the community.

Participation is completely voluntary and you may withdraw from the study at any time before the final reports are made public.

We do not collect identifying information in this study. However, **if you would like to be considered for the award, withdraw your data from the study at a later stage, and/or be notified when the results of the study are available, you should provide your email address as contact information**. We will also appreciate it if you provide optional demographic information, which will only be shared in an aggregated form and will not be associated with any individual responses.

Thank you in advance for your time.

Your email address (optional):

Experience and Demographics

1. Which of the following describes you the best? (pick all that apply)

- ☐ Software developer or engineer working in industry
- ☐ Software tester working in industry
- ☐ Researcher working in industry (Research Staff Member, Research Fellow, Research Engineer)
- ☐ Researcher working in academia, non-student (Postdoctoral Fellow, Faculty Member)
- ☐ PhD student
- ☐ Master's student
- ☐ Undergraduate student
- ☐ Other

2. How many years of programming experience while in **school/university** do you have? (in any programming language)

- ☐ No experience
- ☐ Less than 1 year
- ☐ At least 1 but less than 3 years
- ☐ At least 3 but less than 5 years
- ☐ At least 5 but less than 10 years
- ☐ More than 10 years

3. How many years of programming experience **outside of school/university** do you have?

- ☐ No experience
- ☐ Less than 1 year
- ☐ At least 1 but less than 3 years
- ☐ At least 3 but less than 5 years
- ☐ At least 5 but less than 10 years
- ☐ More than 10 years

4. How would you rate your programming skill level?

- ☐ Novice: developed a few small programs
- ☐ Intermediate: developed a few large programs
- ☐ Advanced: developed several large software systems

5. What is your software development area? (e.g., web developer, full stack developer, embedded systems developer, ML data analyst)

6. How do you debug your code? (Pick all that apply)

- ☐ I do not debug my code
- ☐ Program logging (e.g., print)
- ☐ Assertions
- ☐ IDE debugger utilities (e.g., breakpoints and stepping)
- ☐ Other

7. What is your country of employment or studies?

8. What is your age?

- ☐ <25
- ☐ 25-34
- ☐ 35-44
- ☐ 45-54

- ☐ 55-64
- ☐ >64
- ☐ Prefer not to answer

9. What is your gender?

- ☐ Female
- ☐ Male
- ☐ Non-binary person
- ☐ Prefer not to answer

Part I

Part 1/3: Comparing code views

In this part of the survey, you will be given three different views, named A, B, and C. Each view contains two versions of a code snippet: the old version (V1), where an assertion passes, and a new version (V2), where the same assertion fails due to changes made in the code.

Each code snippet includes only a subset of statements deemed relevant to the failure. Views A, B, and C differ by the subset of statements their code snippet includes. You will be asked to rank the views based on their:

- **Completeness:** include all essential information needed to explain and debug the failure.
- **Conciseness:** do not include unnecessary information, unneeded to explain and debug the failure.

The goal is to identify views that are most helpful to explain and debug the failure.

Note: Clicking on the picture below opens its larger version.

Figure 1 illustrates the changes between three versions of a code snippet (V1, V2, and V3). The diagram shows the code snippets for each version and highlights the differences using color coding and annotations.

Version V1:

```

1 public static void main(String[] args) {
2     // ...
3     Scanner scanner = new Scanner(System.in);
4     // ...
5     // ...
6     // ...
7     // ...
8     // ...
9     // ...
10    // ...
11    // ...
12    // ...
13    // ...
14    // ...
15    // ...
16    // ...
17    // ...
18    // ...
19    // ...
20    // ...
21    // ...
22    // ...
23    // ...
24    // ...
25    // ...
26    // ...
27    // ...
28    // ...
29    // ...
30    // ...
31    // ...
32    // ...
33    // ...
34    // ...
35    // ...
36    // ...
37    // ...
38    // ...
39    // ...
40    // ...
41    // ...
42    // ...
43    // ...
44    // ...
45    // ...
46    // ...
47    // ...
48    // ...
49    // ...
50    // ...
51    // ...
52    // ...
53    // ...
54    // ...
55    // ...
56    // ...
57    // ...
58    // ...
59    // ...
60    // ...
61    // ...
62    // ...
63    // ...
64    // ...
65    // ...
66    // ...
67    // ...
68    // ...
69    // ...
70    // ...
71    // ...
72    // ...
73    // ...
74    // ...
75    // ...
76    // ...
77    // ...
78    // ...
79    // ...
80    // ...
81    // ...
82    // ...
83    // ...
84    // ...
85    // ...
86    // ...
87    // ...
88    // ...
89    // ...
90    // ...
91    // ...
92    // ...
93    // ...
94    // ...
95    // ...
96    // ...
97    // ...
98    // ...
99    // ...
100   // ...

```

Version V2:

```

1 public static void main(String[] args) {
2     // ...
3     Scanner scanner = new Scanner(System.in);
4     // ...
5     // ...
6     // ...
7     // ...
8     // ...
9     // ...
10    // ...
11    // ...
12    // ...
13    // ...
14    // ...
15    // ...
16    // ...
17    // ...
18    // ...
19    // ...
20    // ...
21    // ...
22    // ...
23    // ...
24    // ...
25    // ...
26    // ...
27    // ...
28    // ...
29    // ...
30    // ...
31    // ...
32    // ...
33    // ...
34    // ...
35    // ...
36    // ...
37    // ...
38    // ...
39    // ...
40    // ...
41    // ...
42    // ...
43    // ...
44    // ...
45    // ...
46    // ...
47    // ...
48    // ...
49    // ...
50    // ...
51    // ...
52    // ...
53    // ...
54    // ...
55    // ...
56    // ...
57    // ...
58    // ...
59    // ...
60    // ...
61    // ...
62    // ...
63    // ...
64    // ...
65    // ...
66    // ...
67    // ...
68    // ...
69    // ...
70    // ...
71    // ...
72    // ...
73    // ...
74    // ...
75    // ...
76    // ...
77    // ...
78    // ...
79    // ...
80    // ...
81    // ...
82    // ...
83    // ...
84    // ...
85    // ...
86    // ...
87    // ...
88    // ...
89    // ...
90    // ...
91    // ...
92    // ...
93    // ...
94    // ...
95    // ...
96    // ...
97    // ...
98    // ...
99    // ...
100   // ...

```

Version V3:

```

1 public static void main(String[] args) {
2     // ...
3     Scanner scanner = new Scanner(System.in);
4     // ...
5     // ...
6     // ...
7     // ...
8     // ...
9     // ...
10    // ...
11    // ...
12    // ...
13    // ...
14    // ...
15    // ...
16    // ...
17    // ...
18    // ...
19    // ...
20    // ...
21    // ...
22    // ...
23    // ...
24    // ...
25    // ...
26    // ...
27    // ...
28    // ...
29    // ...
30    // ...
31    // ...
32    // ...
33    // ...
34    // ...
35    // ...
36    // ...
37    // ...
38    // ...
39    // ...
40    // ...
41    // ...
42    // ...
43    // ...
44    // ...
45    // ...
46    // ...
47    // ...
48    // ...
49    // ...
50    // ...
51    // ...
52    // ...
53    // ...
54    // ...
55    // ...
56    // ...
57    // ...
58    // ...
59    // ...
60    // ...
61    // ...
62    // ...
63    // ...
64    // ...
65    // ...
66    // ...
67    // ...
68    // ...
69    // ...
70    // ...
71    // ...
72    // ...
73    // ...
74    // ...
75    // ...
76    // ...
77    // ...
78    // ...
79    // ...
80    // ...
81    // ...
82    // ...
83    // ...
84    // ...
85    // ...
86    // ...
87    // ...
88    // ...
89    // ...
90    // ...
91    // ...
92    // ...
93    // ...
94    // ...
95    // ...
96    // ...
97    // ...
98    // ...
99    // ...
100   // ...

```

Annotations:

- Deletion:** Indicated by a red arrow pointing to the line number in V1.
- Addition:** Indicated by a green arrow pointing to the line number in V2 or V3.
- Change:** Indicated by a blue arrow pointing to the line number in V2 or V3.

The diagram shows that V2 and V3 are derived from V1. V2 contains changes to lines 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100. V3 contains changes to lines 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,

1. Please rank views A, B, and C (1 being the best; 3 being the worst). You can drag the view names into the box and then rank them internally. When ranking views, please consider:

- **Completeness:** include all essential information needed to explain and debug the failure.
- **Conciseness:** do not include unnecessary information, unneeded to explain and debug the failure.

The diagram illustrates the relationship between a set of items and their corresponding ranking views. On the left, a vertical list of items is shown: "Items", "View A", "View B", and "View C". On the right, a large rectangular box represents the "Ranking Views" space. The top of this box is labeled "Ranking Views". The box is divided into two main sections: a top section labeled "Ranking Views" and a bottom section labeled "Ranking Views". The bottom section is further divided into two sub-sections: "Ranking Views" and "Ranking Views".

2. Explain your ranking by describing the advantages and disadvantages of each view (given in their sequential order below).

3. Explain the failure in your own words. Specifically, please describe why changes made in this code resulted in the assertion failure.

Part II

Part 2/3: Comparing textual explanations

In this part of the survey, you are given three different **textual** explanations of the failure corresponding to the three code views in the previous part. You will need to rank these explanations based on their:

- **Completeness:** include all essential information needed to explain and debug the failure.
- **Conciseness:** do not include unnecessary information, unneeded to explain and debug the failure.

Note: Clicking on the picture below opens its larger version.

Explanation A

The method `parseSQL(query)` returns an SQL expression parsed for a given query.
In line 4, the code obtains the SQL keyword of the expression returned by `parseSQL(query)`.
The assertion in line 6 checks if keyword has the same value as the variable expected.
However,
Internally, in line 11, the method computes the value of keyword as a function of the input query.
However,
1. In V1, if the value of keyword is equal to "select" (line 12), the variable is reassigned to its uppercase version, i.e., "SELECT" (line 13).
2. In V2, lines 12 and 13 are deleted and, thus, keyword is not reassigned to its uppercase version.
Then, in both versions, the variable keyword is appended to the variable `expr` (line 14). Next, `expr` is further appended with function, table, and predicate in lines 16, 18, and 20, respectively.
In summary, the deletion of lines 12-13 leads to the difference in the value of keyword in V1 and V2. This difference causes the assertion to fail.

Explanation B

The method `parseSQL(query)` returns an SQL expression parsed for a given query.
The input query is initialized in line 3 as "select sum(c1) from sales where c2>1".
In line 4, the code obtains the SQL keyword of the expression returned by `parseSQL(query)`.
The assertion in line 6 checks if keyword has the same value as the variable expected.
However,
Internally, in line 11, the method computes the value of keyword as a function of the input query.
However,
1. In V1, if the value of keyword is equal to "select" (line 12), the variable is reassigned to its uppercase version, i.e., "SELECT" (line 13).
2. In V2, lines 12 and 13 are deleted and, thus, keyword is not reassigned to its uppercase version.
Then, in both versions, the variable keyword is computed as a function of both keyword and query (line 20).
In summary, the deletion of lines 12-13 leads to the difference in the value of keyword in V1 and V2. This difference causes the assertion to fail.

Explanation C

The method `parseSQL(query)` returns an SQL expression parsed for a given query.
In line 4, the code obtains the SQL keyword of the expression returned by `parseSQL(query)`.
The assertion in line 6 checks if keyword has the same value as the variable expected.
However,
Internally, in line 11, the method computes the value of keyword as a function of the input query.
However,
1. In V1, if the value of keyword is equal to "select" (line 12), the variable is reassigned to its uppercase version, i.e., "SELECT" (line 13).
2. In V2, lines 12 and 13 are deleted and, thus, keyword is not reassigned to its uppercase version.
Then, in both versions, the variable keyword is computed as a function of both keyword and query (line 20).
In summary, the deletion of lines 12-13 leads to the difference in the value of keyword in V1 and V2. This difference causes the assertion to fail.

Notations: Colored backgrounds highlight the differences between the views.
FYI: Views are given below again, for your reference.

View A

V1

1 public static void main(String[] args){
2 SQLExpr expr = parseSQL(query);
3 String result = expr.getSQLKeyword();
4 String expected = "SELECT";
5 assertEquals(result, expected);
6 public SQLExpr parseSQL(String query){
7
8
9
10
11
12 if(keyword.equals("select"))
13 keyword = keyword.toUpperCase();
14 expr = expr.append(keyword);
15
16
17 String function = parsed[1].toUpperCase();
18 expr = expr.append(function);
19
20 String table = parsed[3];
21 expr = expr.append(table);
22
23 String predicate = parsed[5];
24 expr = expr.append(predicate);
25 return expr;
26 }
27 }

V2

1 public static void main(String[] args){
2 SQLExpr expr = parseSQL(query);
3 String result = expr.getSQLKeyword();
4 String expected = "SELECT";
5 assertEquals(result, expected);
6 public SQLExpr parseSQL(String query){
7
8
9
10
11
12
13
14 expr = expr.append(keyword);
15
16
17 String function = parsed[1].toUpperCase();
18 expr = expr.append(function);
19
20 String table = parsed[3];
21 expr = expr.append(table);
22
23 String predicate = parsed[5];
24 expr = expr.append(predicate);
25 return expr;
26 }
27 }

View B

V1

1 public static void main(String[] args){
2 String query = "select sum(c1) from sales where c2>1";
3 SQLExpr expr = parseSQL(query);
4 String result = expr.getSQLKeyword();
5 String expected = "SELECT";
6 assertEquals(result, expected);
7 public SQLExpr parseSQL(String query){
8
9
10
11
12
13
14
15 String keyword = Func(keyword, query);
16 if(keyword.equals("select"))
17 keyword = keyword.toUpperCase();
18 expr = expr.append(keyword);
19
20
21
22
23
24
25
26
27 }
28 }

V2

1 public static void main(String[] args){
2 String query = "select sum(c1) from sales where c2>1";
3 SQLExpr expr = parseSQL(query);
4 String result = expr.getSQLKeyword();
5 String expected = "SELECT";
6 assertEquals(result, expected);
7 public SQLExpr parseSQL(String query){
8
9
10
11
12
13
14
15 String keyword = Func(keyword, query);
16
17
18
19
20
21
22
23
24
25
26
27 }
28 }

View C

V1

1 public static void main(String[] args){
2 SQLExpr expr = parseSQL(query);
3 String result = expr.getSQLKeyword();
4 String expected = "SELECT";
5 assertEquals(result, expected);
6 public SQLExpr parseSQL(String query){
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27 }
28 }

V2

1 public static void main(String[] args){
2 SQLExpr expr = parseSQL(query);
3 String result = expr.getSQLKeyword();
4 String expected = "SELECT";
5 assertEquals(result, expected);
6 public SQLExpr parseSQL(String query){
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27 }
28 }

4. Please focus on the textual explanation now (upper part of the picture). Please rank explanations A, B, and C (1 being the best; 3 being the worst). You can drag the explanation names into the box and then rank them internally.

Items

Explanation A

Explanation B

Explanation C

Ranking Explanations

5. Explain your ranking by describing the advantages and disadvantages of each explanation (given in their sequential order below).

Explanation A:
+
+
-
-
Explanation B:
+
+
-
-
Explanation C:
+
+
-
-

6. Which of the following do you prefer to see when understanding and debugging the failure?

- ☐ Code views
- ☐ Textual explanations
- ☐ Both

7. Explain your selection.

Part III

Part 3/3: Analyzing complete code snippets

In this part of the survey, you are given the complete code snippets of versions V1 and V2. You will be asked whether seeing the complete code helps your understanding of the failure.

Notations: The code is represented in a way similar to the code in Part 1. For simplicity, each “if” statement (e.g., in line 12) is annotated with a label showing whether the “if” condition is evaluated to *true* or *false*. Gray lines indicate statements that are not executed because they are encapsulated by an 'if' statement that evaluates to *false*. Like in Part 1, Changes between versions (“Add”, “Update”, “Delete”) are shown on arrows between the code snippets.

8. Please select statements you deem important for understanding and debugging the failure (in addition to the changed statements that are clearly important and, thus, already pre-selected below). Click on a statement to select it and click again to deselect it.

V1

1 public static void main(String[] args){
2 String query = "select sum(c1) from sales where c2>1";
3 SQLExpr expr = parseSQL(query);
4 String result = expr.getSQLKeyword();
5 String expected = "SELECT";
6 ☒ assertEquals(result, expected);
7 }
8 public SQLExpr parseSQL(String query){
9 SQLExpr expr = new SQLExpr();
10 String[] parsed = query.split(" ");
11 String keyword = parsed[0];
12 if(keyword.equals("select")) [true]
13 keyword = keyword.toUpperCase();
14 expr = expr.append(keyword);
15 String function = parsed[1].toUpperCase();
16 expr = expr.append(function);
17 String table = parsed[3];
18 expr = expr.append(table);
19 String predicate = parsed[5];
20 expr = expr.append(predicate);
21 return expr;
22 }

Delete

V2

1 public static void main(String[] args){
2 String query = "select sum(c1) from sales where c2>1";
3 SQLExpr expr = parseSQL(query);
4 String result = expr.getSQLKeyword();
5 String expected = "SELECT";
6 ☒ assertEquals(result, expected);
7 }
8 public SQLExpr parseSQL(String query){
9 SQLExpr expr = new SQLExpr();
10 String[] parsed = query.split(" ");
11 String keyword = parsed[0];
12
13
14 expr = expr.append(keyword);
15 String function = parsed[1].toUpperCase();
16 expr = expr.append(function);
17 String table = parsed[3];
18 expr = expr.append(table);
19 String predicate = parsed[5];
20 expr = expr.append(predicate);
21 return expr;
22 }

9. Explain your selection.

10. Has seeing the complete code snippet changed your understanding of the failure and why? Would you now augment the explanation you gave in Part 1, Question 3? (you can navigate to your explanation by pressing the back button twice)

11. Please list any suggestions for how the views and textual explanations you liked the most can be improved even further.

12. Do you have any other comments related to this survey?

