Why are program analysis tools difficult to understand?

A tool (mis)communication theory and adaptive approach for supporting developers during tool use

Brittany Johnson

NC State University
bijohnso@ncsu.edu

Abstract.

Table of Contents

1 Introduction . . . 3
2 Research Significance . . . 3
2.1 What are program analysis tools? . . . 3
2.2 How do they communicate? . . . 4
2.3 What do they communicate about? . . . 4
2.4 How does tool communication relate to verbal communication? . . . 5
2.5 Why don't developers use program analysis tools? . . . 5
3 A Tool Miscommunication Theory . . . 5
4 An Approach for Modeling Developer Knowledge . . . 6
4.1 Using Concept Inventories to Assess Conceptual Knowledge . . . 6
4.2 Using Public Git Repositories to Predict Conceptual Knowledge . . . 8
5 A Proposed Approach for Notification Adaptation . . . 8
6 Experiments & Evaluations . . . 9
7 Related Work . . . 9
8 Project Plan . . . 10
9 References . . . 20

1 Introduction

If program analysis *tool use* is a form of *communication* and *inability to resolve* notifications is *miscommunication*, tools can adopt a theory similar to *constructivism* where tools can *approximate* individual developer's *conceptual knowledge* to adapt notifications, leading to *reduced time and effort* required for developers to resolve tool notifications.

2 Research Significance

2.1 What are program analysis tools?

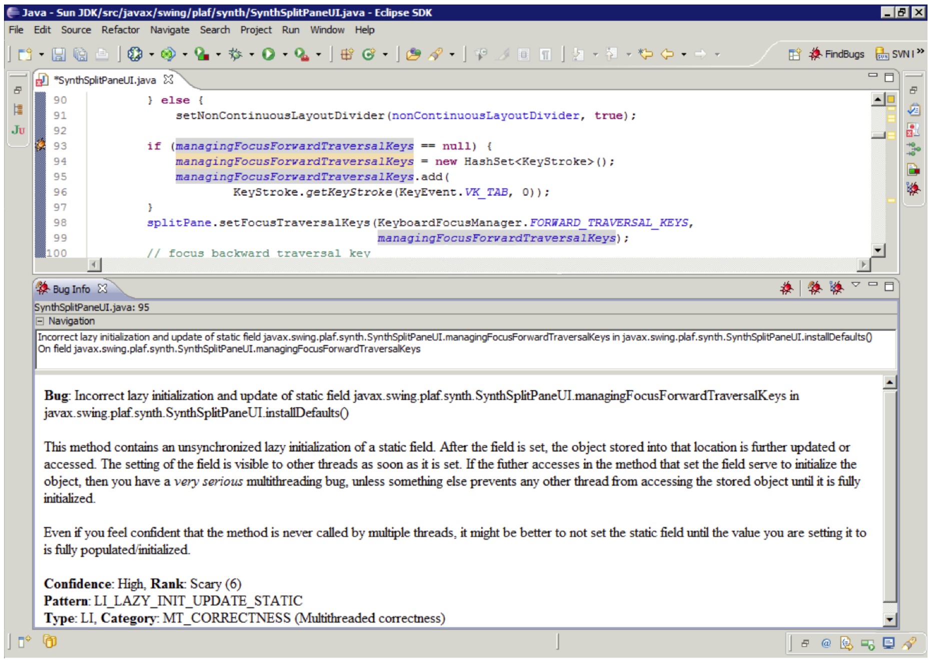
2.2 How do they communicate?

2.3 What do they communicate about?

2.4 How does tool communication relate to verbal communication?

2.5 Why don't developers use program analysis tools?

Motivating Example


Java - Sun JDK/src/javax/swing/plaf/synth/SynthSplitPaneUI.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
FindBugs
SVN
*SynthSplitPaneUI.java
Bug Info

```
        } else {
            setNonContinuousLayoutDivider(nonContinuousLayoutDivider, true);

        if (managingFocusForwardTraversalKeys == null) {
            managingFocusForwardTraversalKeys = new HashSet<KeyStroke>();
            managingFocusForwardTraversalKeys.add(
                    KeyStroke.getKeyStroke(KeyEvent.VK_TAB, 0));
        }
        splitPane.setFocusTraversalKeys(KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS,
                                        managingFocusForwardTraversalKeys);
        // focus backward traversal key
```

SynthSplitPaneUI.java: 95

Navigation

Incorrect lazy initialization and update of static field javax.swing.plaf.synth.SynthSplitPaneUI.managingFocusForwardTraversalKeys in javax.swing.plaf.synth.SynthSplitPaneUI.installDefaults()
On field javax.swing.plaf.synth.SynthSplitPaneUI.managingFocusForwardTraversalKeys

Bug: Incorrect lazy initialization and update of static field javax.swing.plaf.synth.SynthSplitPaneUI.managingFocusForwardTraversalKeys in javax.swing.plaf.synth.SynthSplitPaneUI.installDefaults()

This method contains an unsynchronized lazy initialization of a static field. After the field is set, the object stored into that location is further updated or accessed. The setting of the field is visible to other threads as soon as it is set. If the futher accesses in the method that set the field serve to initialize the object, then you have a *very serious* multithreading bug, unless something else prevents any other thread from accessing the stored object until it is fully initialized.

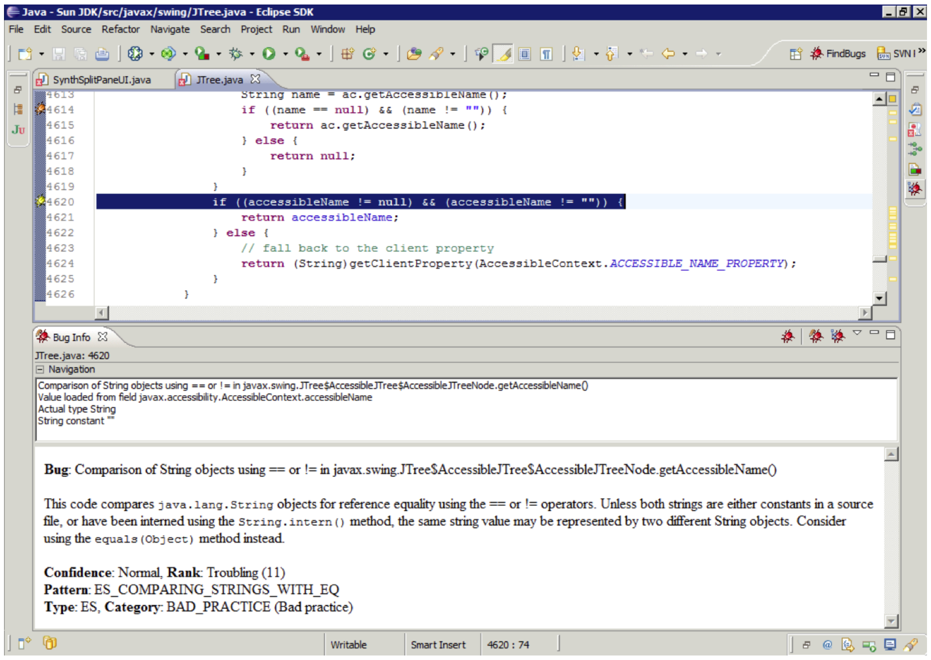
Even if you feel confident that the method is never called by multiple threads, it might be better to not set the static field until the value you are setting it to is fully populated/initialized.

Confidence: High, **Rank**: Scary (6)
Pattern: LI_LAZY_INIT_UPDATE_STATIC
Type: LI, **Category**: MT_CORRECTNESS (Multithreaded correctness)

Fig. 1. Findbugs notification in the Eclipse IDE concerning multi-threading.

Valerie is a software developer at a start-up company. She primarily writes Java code, though she did not learn to program in Java, and uses the Eclipse Integrated Development Environment (IDE). In her spare time, she builds her knowledge of Java programming concepts by contributing to open source software. While modifying code in the Sun JDK source code repository, she contributes code that results in the notification shown in Figure 1. She has experience using FindBugs, so she is familiar with some of the ways FindBugs communicates. For example, she knows that an orange bug icon indicates a *scary* bug and that by clicking the bug icon she can gain access to more information about the bug.

As she explores the information provided by FindBugs, she realizes that despite her experience with FindBugs, she is having difficulty determining how to resolve the notification. She first attempts to use what knowledge she does have regarding multi-threading, which she accrued from resolving compiler synchronization warnings, to better understand the problem; however, she is unfamiliar with the concept central to the notification in Figure 1 (lazy initialization). Though the notification tells her.... , she is unable to make a connection between her knowledge regarding multi-threading and the message FindBugs is attempting to communicate and therefore cannot resolve the notification without outside help.


Java - Sun JDK/src/javax/swing/JTree.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
FindBugs
SVN I
SynthSplitPaneUI.java
JTree.java
Bug Info
Writable
Smart Insert
4620 : 74

```
                    String name = ac.getAccessibleName();
                    if ((name == null) && (name != "")) {
                        return ac.getAccessibleName();
                    } else {
                        return null;
                    }
                }
                if ((accessibleName != null) && (accessibleName != "")) {
                    return accessibleName;
                } else {
                    // fall back to the client property
                    return (String)getClientProperty(AccessibleContext.ACCESSIBLE_NAME_PROPERTY);
                }
            }
```

JTree.java: 4620

Navigation

Comparison of String objects using == or != in javax.swing.JTree$AccessibleJTree$AccessibleJTreeNode.getAccessibleName()
Value loaded from field javax.accessibility.AccessibleContext.accessibleName
Actual type String
String constant ""

Bug: Comparison of String objects using == or != in javax.swing.JTree$AccessibleJTree$AccessibleJTreeNode.getAccessibleName()

This code compares `java.lang.String` objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the `String.intern()` method, the same string value may be represented by two different String objects. Consider using the `equals(Object)` method instead.

Confidence: Normal, **Rank**: Troubling (11)
Pattern: ES_COMPARING_STRINGS_WITH_EQ
Type: ES, **Category**: BAD_PRACTICE (Bad practice)

Fig. 2. Findbugs notification in the Eclipse IDE on checking string equality.

Valerie's goal when using tools like FindBugs is to help her learn more about Java programming concepts, however she finds that some notifications are better at communicating problems while contributing to knowledge than others. For example, when first learning how to work with strings in Java, Valerie encountered the notification in Figure 2. The first time she encountered the problem she was able to understand and resolve the notification. Looking back, she realizes this is because the notification in Figure 2 filled in gaps in her own knowledge of the concept by ...

Because the tools she uses are not familiar with what she does and does not know, the notifications the tools use run the risk of miscommunication with the developer.

3 A Tool Miscommunication Theory

4 An Approach for Modeling Developer Knowledge

4.1 Using Concept Inventories to Assess Concept Knowledge

4.2 Using Public Git Repositories to Predict Concept Knowledge

5 A Proposed Approach for Notification Adaptation

6 Experiments & Evaluations

7 Related Work

8 Project Plan

Acknowledgments

References

Clarke, F., Ekeland, I.: Nonlinear oscillations and boundary-value problems for Hamiltonian systems. Arch. Rat. Mech. Anal. 78, 315–333 (1982)