Change request log

# Team

Team Badvision, consisting of Brendan Robert, the lead janitor.

# Change Request

JE3 – jEdit bug fix to improve highlighting search results for Hyper Search feature to support multiple found instances on a given line.

# Initiation

From assignment 1, I knew what a mess this program was. I also knew that the author grouped features next to each other in the packages, but despite that packaging the underlying processing logic and the user interface were tightly co-mingled. I picked this bug because it looked like a potential oversight (using single value instead of proper loop), but that was unfortunately not the case. Also, among the other reports this seemed like the most useful and impactful bug of the three to fix.

# Concept Location

Concept location is extremely difficult due to poor code organization and unmodulated programing style. The eventual result of concept location was a very overly-large monolithic method which was rather improperly named, with very informative variables such as "s".

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *I decided to use a project-wide search for "HyperSearch" using the project search feature of IntelliJ. Found several classes in a search package with HyperSearch in their names.* | *Code organization is highly suspect, and I was hoping that if anything I could find the corresponding code because the author tends to put the name of the feature in the class name.* |
| 2 | *Ran the system to reproduce the issue; made notes of text labels that appear in the UI.* | *Needed to gain familiarity and find labels which could show up as strings in the code responsible for generating the user interface.* |
| 3 | *Noted that the number of occurrences are counted correctly but as the bug indicates only one per line is shown* | *This confirms that the processing logic responsible for counting occurrences works, but some display code is at fault. But that doesn’t make finding the display code any easier.* |
| 4 | *Studying the data model, there is a linked-list stored in a property called "occur" in the HyperSearchResult class. Goal is to trace uses of this variable because it stores a list of occurrences of the search string on each line* | *Walking references to this type of data should help pinpoint what parts of the code are not interpreting it correctly.* |
| 5 | *Tried debugging the program to perform dynamic analysis on the getSelection method, which converts the linked list of occurrences in a line to an array of ranges* | *Dynamic analysis using a debugger reveals stack traces of all the methods that called each other up to that point, and allows inspection of the full application state. Unfortunately this method only triggers AFTER the UI has already rendered incorrectly, so it is a dead end!* |
| 6 | *Starting from scratch, I decided to deconstruct how the results window is rendered, found that HyperSearchResults generates the view in its constructor method (of all places…)* | *Only rationale here is I was desperate and out of elegant options. At this point I had to start thinking like the original developer to understand his thought process (or lack thereof.)* |
| 7 | *After many false starts, FOUND THE ISSUE in the HyperSearchResults.HighlightingTree.convertValueToText method* | *I didn’t have notes leading up to this discovery because this was probably one of dozens of things I evaluated until I finally had a eureka moment.* |
| 8 | *Set breakpoints at line 583 and 601 in HyperSearchResults.HighlightingTree.convertValueToText method. Ran the application with debugging activated to confirm behavior.* | *Author had written their own pattern matcher instead of using the one that Java provides. Unsurprisingly their code does not work correctly.* |
| 9 | *Concept as been located. Identified this as the location and there are no other side effects.* | *The author only committed this mistake once that I could determine, and this class isn’t called by anything else.* |

**Time spent (in minutes):** 240

# Impact Analysis

This code is a mess. I’ve provided a more detailed summary of this during Assignment 1, but basically it’s challenging enough to find an IDE that can even open the project; I still have to run Ant from the command line and I am unable to execute tests. Furthermore, the IntelliJ IDE is not able to successfully navigate the code for dependency lookups, such as finding references to existing methods so I have to use static searches to look for method calls; complicating analysis

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *Evaluated code and found that the monolithic method where the bug occurs fortunately is not called by anything other than the UI that appears once.* | *Eliminate any ripple effects expected during repair makes getting this over with much more painless.* |

**Time spent (in minutes):** 5

# Prefactoring (optional)

**Ideally, if this were not such a fragile software project and I were able to execute tests, I could assert a few behaviors in a unit test as working, make changes, and assert there are no side-effects. Instead, I am confronted with a fragile system and inoperable unit tests. Therefore, my goal is to make as minimal of a change as possible and get out of the system as quickly as I can before I break anything. The work of properly rectifying the code structure should be in its own feature branch with an entire epic of stories to address the many shortcomings of the program and its structure (or lack thereof.)**

**Time spent (in minutes):** 0

# Actualization

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *Gutted the horribly written search loop like a fish.* | *The goal is to provide an array of numbers for start/end ranges of an occurrence of a string. The author can’t even do that right. I don’t want to use their code as a reference, there’s no point.* |
| 2 | *Needed to figure out what the original search string is, and it’s not immediately obvious where to get it from.* | *Once I have the search string, I can perform a search using Java’s pattern matching classes which are side-effect free and predictable.* |
| 3 | *Ran the program using breakpoints previously established so I could inspect variables in the scope of that method. Identified I could get the search string by a (VERY BADLY CONSTRUCTED) model object in the scope, node.getUserObject(); provides the object describing the search operation and its toString method returns the original search string.* | *If I can get the search string without changing the method signature then I have less chances of breaking anything.* |
| 4 | *With the search string, I rewrote the search portion of the monolithic method, removing 21 lines and replacing them with 9.* | *Garbage can is the best editing tool. (Argument could be made that there’s a lot more garbage here but that’s beyond the scope of this bug.)* |
| 5 | *Re-compiled and ran program to verify.* | *Works!* |

**Time spent (in minutes):** 10

# Postfactoring (optional)

The bug is fixed and the code is just as scary as it was before. Best path forward is to run away from it as quickly as I can before something else breaks. The temptation to gut half the program and restructure everything is overwhelming. It’s not worthwhile because the authors can’t even be bothered to respond to support requests for their broken Git repository, so I am in no mood to do them any favors either.

**Time spent (in minutes):** 0

# Validation

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *Run manual test to verify program works correctly using provided example as a test case* | *This is the regular expected behavior.*  *The test passed.* |

**Time spent (in minutes):** 5

# Timing

Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location | 240 |
| Impact Analysis | 5 |
| Prefactoring | 0 |
| Actualization | 10 |
| Postfactoring | 0 |
| Verification | 5 |
| Total | 260 |

# Reverse engineering

No. I’m not going to write a UML diagram because it’s going to look like a tangled ball of spaghetti. What I can say is that the bug existed within the vacuum of a single monolithic method (*HyperSearchResults.HighlightingTree. convertValueToText)* that had way too many concerns going on; and that’s where I fixed the bug. There are no other class interactions because this code is not designed well at all, it’s a big jumble! Trying to explore the application any further than necessary only reveals that the author was often frustrated and ultimately probably a very unstable and/or broken man; there’s nothing to be gained from exploring this code other than misery and heartbreak.

# Conclusions

The bug occurred because the author tries to reinvent many concepts that are already answered by the JDK runtime. For example, many places he creates his own linked list structures and uses while loops to iterate them, rather than just use java collections like a normal Java developer would! In this specific case of JE3, he recreated his own pattern matcher which has no good reason to exist. When the code was written, Apache ORO already was considered the norm for pattern matching. After JDK 1.4 (and that was a LONG time ago), Java added pattern matching as a first-rate feature so ORO was no longer needed either. Therefore, the find method he wrote should have been refactored out a long time ago. That is a much larger problem than this bug, but it is worthy of note because it indicates the vast amount of accrued technical debt.

My summary is as follows:

* The poor code organization as well as poor coding style made concept location very difficult.
  + This code was written by a C developer with next to no idea of how to use Java collections or iterators properly, so it makes code comprehension difficult.
  + There are uses of deprecated Java features (such as Vector) and a lack of leveraging java features introduced long before this code was heavily written; therefore this code carries a lot of needless technical debt that could be erased by refactoring it to follow modern java conventions and replace redundant methods/classes with any similar JRE-provided functionality.
  + The poorly-separated concerns in the code made impact analysis also very difficult. Only dynamic analysis (watching the program execute with a debugger active) was able to alleviate the invisibility perpetuated by many layers of obfuscated code.
* The developer is very unprofessional.
  + Not only is the code quality a critical issue, but the lack of professionalism shows in other ways, such as inappropriate language.
  + There are 8 instances of the f-word. The most entertaining is above a section of code written to resolve a UI Focus issue using a scheduled action; the comment above reads "f--- me dead." If the developer did more reading and less complaining they might have instead discovered how to mark a UI node as focus-transparent which also probably would have worked.
* Given the high level of technical debt and poor organization, evolving this software is not easily possible.
  + It is more feasible to write a new program that has better organization and newer java features that simplify the work.
  + There are several other editors written in Java which are designed properly, have managed to grow over time and evolve, and it’s unsurprising that this one has reached an evolutionary dead-end.