Assignment 1 Aayush Sapkota PRT 452

Question no 3:

Answer:

* ***Objective:***
  + To test whether the code Bad Smells are directly associated with faults.
  + To test whether the Code Bad Smells is directly associated with particular types of faults.

## *Targeted Code Bad Smells:*

* + Data Clumps, Message Chains, Middle Man, Speculative Generality, and Switch Statements
* ***Use the following data sets:***
  + Apache Common Packages (Common IO, Common Logging, Common Codec, Common DbUtils, Common DBCP, and Common Net )
* **Choose some bad smells.** 
  + Manually investigate 5 source code samples from the Apache commons
  + Zimmerman et al.’s (2007) fault identification approach:
  + Locate “bug”, “fix(ed)” and “update(d)” token in CVS comment messages.
  + If a version entry in CVS contains one or more above tokens and those tokens are followed by numbers, this version entry is seen as a bug fixing update.

Bugs:

1. DBCP – 80 Bugzilla ID – 24082

{ max = Math.max(max, Integer.parseInt((String)obj)); }

This code results into an error in code because the code assumes that the key it gets is an int, this is unnecessary delegation so, is speculative generality. This problem has this smell and has a bug because of this so, shows that code smells may result in bugs.

1. IO-448

The code waits too long to detect an exception so, it doesnot detect an interruption. The code has a bad smell of message chain:

Thread.currentThread.interrupt();

The smell doesn’t affect the execution of the code so, not confirming any relation between code smells and faults.

1. NET-551

The code re-uses the chars field for both the number of characters and the single characters. This is a type of data clumps and make code hard to read.

The existence of this bad smell might have led to the bug showing incorrect value for bytesTransferred.

1. Logging-147

Switch(type){

|  |
| --- |
| case SimpleLog.LOG\_LEVEL\_TRACE: buf.append("[TRACE] "); break; |
| case SimpleLog.LOG\_LEVEL\_DEBUG: buf.append("[DEBUG] "); break; |
| case SimpleLog.LOG\_LEVEL\_INFO: buf.append("[INFO] "); break; |
| case SimpleLog.LOG\_LEVEL\_WARN: buf.append("[WARN] "); break; |
| case SimpleLog.LOG\_LEVEL\_ERROR: buf.append("[ERROR] "); break; |
| case SimpleLog.LOG\_LEVEL\_FATAL: buf.append("[FATAL] "); break; |

}

Simplelog.log uses switch statements and data clumps. This update is not safe and uses fields written twice, log is called by two threads. These might be the results of having multiple bad smells in the code.

1. IO-389

FileUtils.java : Between the call to File.listFiles and the call to FileUtils.sizeOf, a file may be deleted. This can cause FileUtils.sizeOf to throw an IllegalArgumentException, indicating that the file does not exist.

This bug is caused a result of speculative generality so, no focus has been given to edge cases like this.

**Conclusion:**

From the analysis of these 5 different bugs, we can see that most of the time if a program has a bad smell then, it is prone to faults.