Wiki

All Sites Advanced Search

Wiki > Wiki Pages > Good Coding Practice

Good Coding Practice

LOKI Coding Standards

<<document our coding standards here>>

Fortify

Fortify is a tool used to find potential vulnerabilities in code. Scans are performed after each release.
Find a presentation on the most prevalent Fortify findings here: LOKI Dev Team > Shared Documents > Fortify

Fortify Finding Prevention

Always Close Streams in Finally

CAT II Finding
Explanation from Fortify
The program can potentially fail to release a system resource.

Recommended Fix
com.cdmtech.icdm.util.file.FileWrapper has a safeClose method.

```
public void doStuff(DataHandler dataHandler) {
   InputStream stream = null;
   try {
      stream = dataHandler.getInputStream();
      ...
   }
   finally {
      FileWrapper.safeClose(stream);  //This will properly null check.
   }
}
```

Remove Empty Catch Blocks

CAT II Finding
Explanation from Fortify
Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break.

Recommendation from Fortify
At a minimum, log the fact that the exception was thrown so that it will be possible to come back later and make sense of the resulting program behavior.

Avoid Throws Exception

CAT II Finding
Explanation from Fortify
Declaring a method to throw Exception or Throwable makes it difficult for callers to do good error handling and error recovery. Java's exception mechanism is set up to make it easy for callers to anticipate what can go wrong and write code to handle each specific exceptional circumstance. Declaring that a method throws a generic form of exception defeats this system.

Recommended Fix
List any Exceptions that may be thrown from your method.

Avoid Catch NullPointerExcptions

CAT II Finding
Explanation from Fortify

Programmers typically catch NullPointerException under three circumstances:

1. The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem.
2. The program explicitly throws a NullPointerException to signal an error condition.
3. The code is part of a test harness that supplies unexpected input to the classes under test.

Of these three circumstances, only the last is acceptable.

Recommendation from Fortify

The program should not dereference null pointers. If you cannot eliminate the cause of the null pointer dereference, you must carefully review the code to make sure that the exception is handled in such a way that the program does not enter into an unexpected or illegal state.

If the NullPointerException is being thrown explicitly, change the program to throw an exception derived from RuntimeException or Error instead.

Do not throw in a Finally

CAT II Finding
Explanation from Fortify
Using a throw statement inside a finally block breaks the logical progression through the try-catch-finally.

Recommendation from Fortify
Never throw exceptions from within finally blocks. If you must re-throw an exception, do it inside a catch block so as not to interupt the normal execution of the finally block.

Synchronization parse() and format() methods

CAT II Finding
Explanation from Fortify
The methods parse() and format() in java.text.Format contain a design flaw that can cause one user to see another user's data.

Recommendation from Fortify
Synchronize any format or parse.

```
public class Common {
   private static SimpleDateFormat dateFormat;
   ...
   public synchronized String format1(Date date) {
      return dateFormat.format(date);
   }
   public String format2(Date date) {
      synchronized(dateFormat) {
         return dateFormat.format(date);
      }
   }
}
```

Remove unused Private Methods and Variables

CAT II Finding
Explanation from Fortify
Dead code is defined as code that is never directly or indirectly executed by a public method.
Recommended Fix
Delete the unused method/variable. You can alway comment it out, if you **must** hang on to it.

Remove main Methods Used for Debug

CAT II Finding
Explanation from Fortify
A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to

be shipped or deployed with the application. When this sort of debug code is accidentally left in the application, the application is open to unintended modes of interaction. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application.

Recommended Fix

Put debug code in a package used for testing, that will not be depolyed with the software.

Do not use System.out.println

CAT II Finding

Explanation from Fortify

Using System.out or System.err rather than a dedicated logging facility makes it difficult to monitor the behavior of the program.

Recommended Fix

Use the com.cdmtech.icdm.web.util.LoggingUtil methods.

Other Coding Tips

Intro to Google's Guava libaray. This library is used in many places within SLP.

The NMI series of presentations are available on **CorpNet** at **\\tapslo034\Public\TUG Presentations**
There you will find the videos taken during Ian's presentations on the Google Guava Libraries and the Finder-DAO pattern we employ, as well as the discussion that took place after the Finder-DAO presentation.

Last modified at 7/9/2013 3:04 PM by Briano Planeta