7.5/10 - Good start David. I think you understand refactoring better than this assignment demonstrates. The bad smells were a bit confusing and there were a number of things you overlooked and your trade-offs (the key to this assignment) were nearly non-existent.

I know that you are better at this than this assignment shows and I gave some benefit of the doubt in some of the cases. If you want to redo this assignment to imrpove the grade, please let me know.

David Pick
CM 2403
CSSE 375

Bad Smells:

Smell	Code	Refactorings
Long Method	The entire run method	Extract Method
Long Parameter List	`fileDetails.put( file.getName(), new FileDetail(new Date(), file.getName()));`	Replace Parameters with Method, Introduce Parameter Object, Preserve and Introduce Object
Switch Statements there are no switch statements...	`fileDetail == null`	Introduce Null Object
Feature Envy I don't see what you mean here	The code many methods from other classes	Move Method, Extract Method, Move Field
Inappropriate Intimacy	Parts of other objects are accessed throughout the code	Move Method, Move Field, Hide Delegate
Switch Statements there are no switch statements... Perhaps complex conditional?	If statements are used throughout the code when they don't need to be	Replace Conditional with Polymorphism
Temporary Variable	`String directoryName = reader.readLine();`	Replace temp with Query

3 params is a bit of a stretch to call a long param. list...

Also message chaining, complex conditionals, nested try-catch blocks, and varying levels of abstraction ...

Analysis of Refactorings:

Extract writeFileDetails:

This refactoring addresses the issue that writing to a file is done several times throughout the method, and each time it requires multiple lines of code or more specifically the long method smell. This refactoring allows writing to a file to be done with one method call which makes the original code easier to read as well as debug since the developer can now debug each method separately. They can also write separate test cases for each method which allows them to have a higher confidence level in each of the methods.

doesn't make sense to have one method call and debug "each" separately...

This refactoring accomplishes the goals of the developer well, however an alternative to this would be to extract all I/O functionality into a single class. This way reading and writing to a file could be accomplished from a singleton.

Good!

While extracting this method does make the original code easier to understand on a whole, it does add another layer of abstraction into the original code. This means that while debugging this code the developer may have to jump around the code to see all of it.

do you mean indirection?

What are the trade-offs?

Extract sendModifiedFiles:

This refactoring addresses an issue very similar to the previous one. By preforming this refactoring the run method becomes clearer and is doing less work. This means that debugging the run method will be easier and the developer will be able to test the run method and the sendModifiedFiles method seperately.

This refactoring does accomplish the developers goals however an alternative this would be again to extract the I/O methods into their own class.

While extracting this code does make the original method easier to understand it adds another layer of abstraction into the code. This can make the original code hard to follow since the developer may have to jump around the file or files.

What are the trade-offs?

again, do you mean indirection?

Extract readFileDetails:

This refactoring addresses the long method bad code smell. By preforming this refactoring the run method becomes clearer and may fit on one page. This means the developer can gain a good sense of what the method is doing without having to read through several pages of code to understand one method.

This refactoring does accomplish the developers goals however an alternative this would be again to extract the I/O methods into their own class.

While extracting this code does make the original method easier to understand it adds another layer of abstraction into the code. This can make the original code hard to follow since the developer may have to jump around the file or files.

What are the trade-offs?

again, do you mean indirection?

Extract Try Clause to an Abstract Method:

This refactoring addresses the try-clauses that were scattered throughout the original code. By extracting everything that was in the original try clause into a single method the developer has again made the run method much easier to understand.

I think you misunderstood this... the sendModifiedFiles becomes the abstract method with Map fileDetails while the try-catch pairs are dealt with in the run().

While this refactoring does accomplish the developers goals an alternative would be to extract the things inside of the try-catch block into their own method. This way the exceptions would be handled by that class and would not clutter up the run method.

I don't particularly like this refactoring since it does little to improve the readability of the original code. It however does make the run method slightly shorter which is good. But, it does not help the problem of having a try-catch block which makes code harder to read.

what would be better?

alternatives and trade-offs?

Summary:

The refactorings that the author of this page completed were all well reasoned and performed well. However, there were many more refactoring opportunities left in the code that were not

addressed. In order for this article to be a greater help to the readers the author needs to analyze the code further. This will lead to refactoring the methods that were extracted as well as helping to deal with the try-catch issue still in the run method.