# Discussion of approach.

When reading the spec, it says re factor the code. It does not say the code produces incorrect results.

**Steps taken to refactor the code:**

* Having the solution under source control is good practice. I have created a GIT repository and added the solution.
* Created a unit test project. (This is using the FakeItEasy mocking framework and MSTest).
* Created 3 test methods based on the acceptance criteria outlined in the spec.
* The GetSnag method, news up instances of ArchivedDataService, FailoverRepository, SnagDataAccess. The classes should be injected using IOC. I created interfaces for the three classes.
* Due to the amount of classes in the project grouping the classes makes it easier to maintain. (DataAccess, Services) to make the project more manageable.
* The program flow of the GetSnag method has many nested if statements.

This makes it more difficult to read and debug.

The program flow can be better organized with a helper method.

The method has been broken into one public method and a private overloaded method “GetSnag”.

* The spec mentioned a snag rule to determine if the system is in failover mode.

Abstracting the check for determining if the system was in failover mode to a separate class made sense.

* The method access some app settings in the web config by means of the configuration manager.

This is messy. Moving the call to the appsettings into a separate Config class makes it cleaner.

Moving the number of minutes and the number of errors used to determine if the system is in failover mode makes it easy to change these values without redeploying the Dll.

* The FailoverSnagDataAccess class has a static method “GetSnagById”. This makes it hard to fake using unit tests. It makes more sense to have it as an instance method. This means when it is implemented it can be disposed of more safely closing any data access connections.