Homework 2

ARCHITECTURE RECOVERY AND ANALYSIS OF LOG4J 2.7 BY: MUKESH DANGI

System: log4j_2.7_1.0a

Change: Added one import PerformanceSensitive.java(line of code) file in LoggerRegistry.java:

1. **Adding a line of code** in the system: **Relax** recovery method:

(a) Adding a line of code in existing code would not create any change in directory structure of log4j 2.7. When we look in log4j_2.7-api_directories.pdf which becomes more evident as new code was not introduced by creating a new directory.

(b) When we analyze log4j-api_relax_clusters_Concerns, we can see there is an arrow from *LoggerRegistry* cluster to *PerformanceSensitive* which clearly indicates a recently added dependency of PerformanceSensitive.java in LoggerRegistry.java.

(c) As we can in log4j-api_relax_clusters_Concerns .pdf that *LoggerRegistry* is one of the populous cluster having so many incoming and outgoing edges. The new change was introduced by an engineer in *LoggerRegistry.java* by importing extra file would impact *LoggerRegistry* and 9 other clusters connected to it.

2. **Adding a line of code** in the system: **ARC** recovery method:

(a) When we analyze log4j-api_deps.rsf generated from ARC recovery method, we can see that we have created a dependency of PerformanceSensitive.java. The new relationship between these two cluster is indicated by the comment in *.rsf* file as – *"depends apache.logging.log4j.spi.LoggerRegistry- apache.logging.log4j.util.PerformanceSensitive"* which substantiates our changes.

(b) Similarity, adding extra line of code is reflected when we inspected log4j-api_deps_clusters.pdf. This recovery method gives more insight on cluster dependencies—here 9 other clusters -- are dependent on *PerformanceSensitive.java* unlike original log4j 2.7 where we have 8 other clusters connected to it. Moreover *PerformanceSensitive.java* is not dependent on any other cluster as there is not outgoing edge from it.

3. **Adding a line of code** in the system: **ACDC** recovery method:
   (a) This recovery method doesn't give much information about dependency graph of new changes in a single file.

   Conclusion: By comparing these 3 recovery methods, I would prefer to use ARC method for architecture recovery to find the recent changed made in system's descriptive architecture specially if we know that changes are made in a single file and engineer has not made any change in package structure or directory. ARC gives information about clusters which are directly and indirectly affected by adding code in existing system architecture. Moreover, In ARC both *log4j-api_deps_clusters.pdf* and log4j-api_deps.rsf tell us more about architecture inconsistencies when we compare new model with existing log4j 2.7.

System: log4j_2.7_1.0b

Change: Added **one file *PerformanceSensitiveRet.java*** and same **import is added in one existing file** named as *ReusableMessageFactory.java*—in the same directory:

1 **Adding a new file** in the same directory: **Relax** recovery method:

(a) Let's dive into log4j_2.7-api_directories.pdf, Here in directory tree we are able to find our changes. though there is no change in the directory structure of existing log4j 2.7 system but

(b) At leaf level of this directory tree, we can see our newly added file *PerformanceSensitiveRet.java* So, we can say that this recovery method makes a tree structure of all packages and files. File information will be at the leaf level and middle node of the tree would be upper level package name.

(c) When we analyze log4j-api_relax_clusters_Concerns, we can see that a new cluster node has been formed named as *PerformanceSensitiveRet*. *PerformanceSensitiveRet* cluster is connected to only one other node which is *ReusableMessageFactory* with an outgoing edge from *ReusableMessageFactory* cluster. Moreover, *PerformanceSensitiveRet* is isolated from whole system except one node so making any reckless change would not impact much.

2. **Adding a new file** in the same directory: **ARC** recovery method: -

   (a) When we analyze log4j-api_deps.rsf generated from ARC recovery method, we can see that we have created a dependency of *PerformanceSensitiveRet.java* in *ReusableMessageFactory.java*. The new relationship between these two cluster is indicated by the a comment in *.rsf* file as – *"depends log4j.message.ReusableMessageFactory - log4j.util.PerformanceSensitiveRet"* which substantiates our changes.

(b) Similarity, adding extra file is reflected when we observe changes in log4j-api_deps_clusters.pdf. This recovery method gives more insight on cluster dependencies—here 9 other clusters -- are dependent on *PerformanceSensitive.java* unlike original log4j 2.7 where we have 8 other clusters connected to it. Moreover *PerformanceSensitive.java* is not dependent on any other cluster as there is not outgoing edge from it.

3. **Adding a new file** in the system: **ACDC** recovery method:

(a) This recovery method doesn't give information about any dependency graph of new file However *log4j-api_acdc_clustered.rsf* suggests that *PerformanceSensitiveRet.java* has dependency on *message.ss* cluster with " *org.apache.logging.log4j.message.ss org.apache.logging.log4j.util.PerformanceSensitiveRet"* comment in the *.rsf* file.

Conclusion: To document the changes of systems' descriptive architecture, I'd prefer to use Relax method. Relax result gives us a clear picture of changes made by an engineer. By analyzing only two generated files -- *log4j_2.7-api_directories.pdf(looking at leaf node)* and *log4j-api_relax_clusters_Concerns.pdf* – we can point out that a new file has been added and how many other clusters would be affected if anyone makes any change in the newly added file.

System: log4j_2.7_1.0c

Change: Added **one file** ***PerformanceSensitiveRet.java*** and same **import is added in one existing file** named as ***SimpleMessageFactory.java*****—in the new directory(org.apache.logging.log4j.utility)**:

1 Adding a new directory with a **new file**: **Relax** recovery method:

(a) Let's dive into log4j_2.7-api_directories.pdf, As per code we have added a new directory and a new file in it which has been referred by a file-- *log4j.message.SimpleMessageFactory*-- in existing code. Now we can see that same implementation level changes are reflecting in directory tree with new directory named as '**utility'** and this is not at leaf level node because we have added new directory so now package structure will change in tree as well.

(b) As we added a directory at the very end of the package, our new directory-- **org.apache.logging.log4j.utility--** will be one above the leaf level in the directory tree. we can say that this recovery method makes a tree structure of all packages and files. File information will be at the leaf level and one level above leaf will be out new directory.

(c) This recovery method generates two types of graph—directories and files-- of a particular system. When we analyze log4j-api_relax_clusters_Concerns, we can see that we have two graphs one bigger graph contains all the concerns or related files as nodes and other colored contains major directories and their types like io, sql, networking, graphics etc. In colored graph a new node named as *org.apache.logging.log4j.utility* has been connected to the '*io'* cluster. Moreover '*io'* cluster has 45 incoming and outgoing links to it so, making any change in *PerformanceSensitiveRet.java would* impact other 45 communication links.

2. Adding a new directory with a **new file**: **ARC** recovery method:

(a) Unlike other ARC recovered system, this time we see that other part of system is dependent on the whole directory instead of a particular file. When we analyze log4j-api_deps.rsf generated from ARC recovery method, we can see that we have created a dependency of *log4j.utility.PerformanceSensitiveRet.java*. The new relationship between these two clusters is indicated by a comment in *.rsf* file as – *"depends org.apache.logging.log4j.message.SimpleMessageFactory org.apache.logging.log4j.utility.PerformanceSensitiveRet"* which substantiates our directory level changes.

(b) Similarity, let's inspect log4j-api_deps_clusters.pdf. This recovery method gives more insight on cluster dependencies—here 9 other clusters -- are dependent on *PerformanceSensitiveRet.java* unlike original log4j 2.7 where we have 8 other clusters connected to it. Moreover *PerformanceSensitive.java* is not dependent on any other cluster as there is not outgoing edge from it.

3. Adding a line of code in the system: **ACDC** recovery method:

(a) Let's inspect *log4j-api_acdc_clustered.rsf* file. Content in the *log4j-api_acdc_clustered.rsf* file suggests that *PerformanceSensitiveRet.java* has dependency on *message.ss* cluster with *" org.apache.logging.log4j.message.ss org.apache.logging.log4j.**utility**.PerformanceSensitiveRet"* comment in the *.rsf* file. This concludes that we have added a directory level changes which could impact the most of the system if something goes wrong in *PerformanceSensitiveRet.java*

Conclusion: By comparing these 3 recovery methods, I would prefer to use Relax method for architecture recovery to find the recent changed made in system's descriptive architecture specially if specially any change in package structure or directory. Relax gives multiple facets of Relax gives information about clusters which are directly and indirectly affected by adding new directory and new file in that directory. Moreover, In Relax both *log4j-api_relax_clusters_Concerns.pdf* and log4j-api_relax_clusters_fn.rsf tell us more about architecture inconsistencies (files, directories, connectivity(graph) of new changes with existing system) when we compare new model with existing log4j 2.7.

System: log4j_2.7_1.0d

Change: Added **one comment in an existing file** ***PerformanceSensitive.java***

1. **Adding a in** a **file**: **Relax** recovery method:
2. **Adding a in** a **file**: **ARC** recovery method:
3. **Adding a in** a **file**: **ACDC** recovery method:
   Adding a comment in the code doesn't create an impact on the system which also reflects in output generated by these three recovery methods.

(Bonus Qs) Architecture Visualization analysis of ARC and Relax:

From Above 4 analyses of different versions of a same system, we can conclude that both ARC and Relax gives a very good insight about architecture drift, erosion/ decay. We included a new comment, new line of code, new file and a new directory in the existing system. Relax architecture recovery methods stand out a winner in this race as it gives information about new line of code, new file, new directory which could impact the system's behavior. Relax gives two special graphical overview whole system one as tree structure where newly added file is at leaf level of the tree and new directory at upper level of leaf nodes. The second—colored-- graph gives information about directories and their association with each other like *sql, io, graphics, networking* etc. Relax groups all the directories into these 5 types so, we have multiple perspectives of a single system.

We can make many improvements like highlighting the new changes made in existing system. Moreover, it is always good to have to matrix of nodes which are getting affected by new changes with percentage of changes. More importantly we should have simulation model of architecture recovery method which can capture the dynamic behavior of the system so, we can estimate about the effects of new changes like whether the new changes are going to affect the core system or just a less used additional functionality? Depending of simulation results we can prepare the testing plans and importance of the testing. Moreover, we can make future prediction about system's stability and architecture decay.