0. MILESTONE 1

Project Preparation/Proposal

1. TEAM FORMATION

Team Member	Tanisha Shah	Jemish Paghadar	Raj Patel	Deep Patel	Dolly Modha	Hardik Vora
Student ID	40088985	40080723	40085012	40087798	40084358	40087606
Project Proposal	✓	✓	✓	✓	✓	✓
Related Work study	✓			✓	✓	✓
Data Collection	✓	✓	✓		✓	
Code Development		✓	✓	✓		✓
Evolution Analysis	✓		✓			✓
Final Report		✓		✓	✓	
Project Presentation	✓	✓	✓	✓	✓	✓

2. TYPE OF STUDY

In software development, evolution of software is the most important part of it. Evolution of software means developing new versions with changes that leads to corrective and adaptive maintenance. Change proneness is an external quality attribute. It depicts the changes in classes across different version of the system. If changes were predicted earlier than errors and bugs can be prevented. Change proneness can affect the understandability, functionality, reusability of the software.

In this proposal, we propose to compare different versions of software systems like Eclipse, Intellij and NetBeans and co-relate the change proneness in the software during their evolution using the C&K metrics, QMOOD metrics. Additionally, we also focus on analysing the attributes like understandability, functionality, reusability, extendibility that are affected due to change proneness. We also plan to calculate the maintainability index that will tell us how easy to support and change the source code is.

3. RELATED STUDIES

Massimiliano Di Penta, Luigi Cerulo, Yann-Ga¨el Gu´eh´eneuc, and Giuliano Antoniol in their paper [1] ‘An Empirical Study of the Relationships between design pattern roles and Class Change Proneness’ presents an empirical study to understand whether there are roles that are more change-prone than others and whether there are changes that are more likely to occur to certain roles. Design pattern suggest design motifs, which are prototypical solutions to design problems. They used Design Motif Identification Multi-layered Approach (DeMIMA) to identify occurrences of design motifs.

Yasutaka Kamei, Ahmed E. Hassan, Naoyasu Ubayashi in their paper [2] ‘Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density’ have proposed a method for developers what kind of code is good or bad by using the metric’s values. They have analysed the defect quality and defect density of Eclipse, NetBeans and Mylyn. They have concluded that size is a better threshold of defect-proneness than complexity.

J.M. Bieman ; A.A. Andrews ; H.J. Yang [4] in ‘Understanding change-proneness in OO software through visualization’ have identified and visualize classes and class interaction that are most change prone by finding patterns and using change proneness measure like local-change-proneness(LCP), Pair change coupling (PCC), Sum of pair coupling(SPC).

Puneet Kumar Goyal and Gamini Joshi’s [5] ‘QMOOD metric sets to assess quality of Java program’ have designed a hierarchical model that defines equations between quality attributes and design properties. The authors have implemented the QMOOD metrics with java programs. They have defined a TQI (Total Quality Index) that helps developers to directly decide the best design.

Bharti Suri and Shweta Singhal in their paper [6] ‘Investigating the OO characteristics of software using CKJM metrics’ have calculated and analysed OO software metrics to improve the software development process and quality. They grouped the metrics into coupling, cohesion, and inheritance and yielded the values of this metrics for 10 different projects written in JAVA.

4. PROJECTS

We plan to work on Eclipse (JAVA IDE), Intellij and NetBeans software projects. Eclipse, NetBeans and Intellij are IDE’s for java. Our main reason for selecting this software is that all three projects are written in the same language. In addition, all this software has more than 200 files or more. These systems completely relate to our goal of co-relating change proneness with different versions using different metrics as this software have a vast range for selecting versions.

4.A. Project Versions

- **ECLIPSE-Platform**
  https://github.com/eclipse/eclipse.platform.ui (source code)
  Releases: eclipse.platform.ui-master released on 8 March,2019
  eclipse.platform.ui-I20180601-0915 released on 1 June,2018
  eclipse.platform.ui-I20170501-2000 released on 1 May,2017
  eclipse.platform.ui-I20160601-1000 released on 30 May,2016

- **NETBEANS**
  https://github.com/apache/incubator-netbeans (source code)
  Releases: incubator-netbeans-master released on 9 March,2019
  incubator-netbeans-9.0-vc3 released on Jul 8, 2018
  netbeans-releases-jshell_api9 released on Nov 17, 2017

- **INTELLIJ**
  https://github.com/JetBrains/intellij-community (source code)
  Releases: IntelliJ IDEA Community Edition released on 9 march2019
  webstorm/181.5540.36 released on Nov 19,2018
  webstorm/172.2465.2 released on May 23,2017
  webstorm/162.646.2 released on May 31,2016

4.B Tools

- **CodeMR:** CodeMR is a software quality and static code analysis tool that helps software companies developing better code, better quality.

- **Understand:** Understand is static analysis tool. Understand provides an efficient way of collecting metrics about the code and providing different ways for you to view it.

- **JHawk:** JHawk is a java metric tool. Using JHawk we will be calculating maintainability index, cumulative halstead bug and cyclomatic complexity.

- **JDeodrant:** This is an eclipse plugin used for identifying design problems and suggest applied refactoring.

- **Dependency Filter:** This is a suite of tools for analysing compiled Java code. JarJarDiff is important tool, which we will be using for finding the changes in different versions of software systems.

- **Metrics:** It is a plugin used for counting metrics of the java source code.

5. METRICS

As mentioned earlier, the metrics that we are planning to implement are C&K metrics and QMOOD metrics.

- **C&K Metrics:**

  It is a set of OO metrics based on complexity, coupling, cohesion and inheritance. It has 6 metrics which are listed below:

  - Weighted method per class (WMC): WMC is sum of complexities of methods defined in a class.
  - Depth of Inheritance Tree (DIT): DIT counts maximum length from node to the root of the tree. DIT would be very helpful in detecting the change in softwares.
  - Number of immediate subclasses (NOC): NOC is number of children subordinated to class in the class hierarchy. NOC will help us to analyse the reusability of the code before developing the new version.
  - Coupling between object classes: CBO represents the count of number of other classes to which a class is coupled. CBO tells us how much effort we need in maintenance.
  - Response for a class (RFC): RFC is a count of the set of methods that can potentially be executed in response to a message received by an object of that class.
  - Lack of Cohesion in Method (LCOM): LCOM considers two methods as cohesive if they access at least one common instance variable.

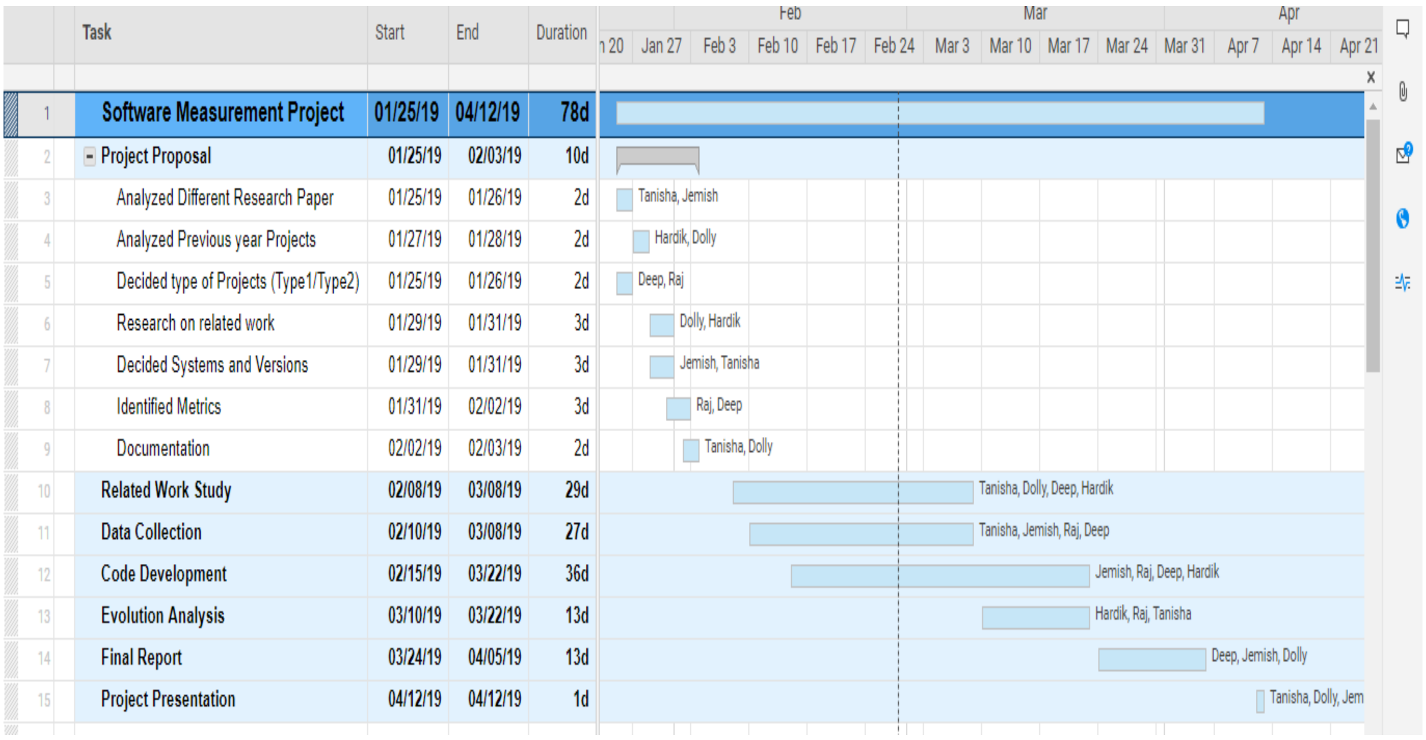
- **QMOOD Metrics:**

  QMOOD proposed by Bansiya and Davis in 2002, assesses the quality attributes (functionality, reliability, efficiency, usability, portability, maintainability) in object-oriented designs. Below is listed the QMOOD metrics (We may not be able to implement all the metrics).

  - Design size in classes (DSC): It calculates number of classes in design.
  - Number of hierarchies (NOH): It calculates number of hierarchies in the design.

- Average number of ancestors (ANA): The number of classes from which a class inherits information.
- Data access metric (DAM): The ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
- Direct Class Coupling (DCC): Count of different number of classes that a class is directly related to.
- Number of polymorphic methods (NOP): It counts number of abstract methods.
- Class Interface Size (CIS): It counts the number of public methods in a class.
- Number of methods (NOM): It counts all the methods defined in a class.

6. RESOURCE PLANNING



	Task	Start	End	Duration	
1	**Software Measurement Project**	**01/25/19**	**04/12/19**	**78d**	
2	- Project Proposal	01/25/19	02/03/19	10d	
3	Analyzed Different Research Paper	01/25/19	01/26/19	2d	Tanisha, Jemish
4	Analyzed Previous year Projects	01/27/19	01/28/19	2d	Hardik, Dolly
5	Decided type of Projects (Type1/Type2)	01/25/19	01/26/19	2d	Deep, Raj
6	Research on related work	01/29/19	01/31/19	3d	Dolly, Hardik
7	Decided Systems and Versions	01/29/19	01/31/19	3d	Jemish, Tanisha
8	Identified Metrics	01/31/19	02/02/19	3d	Raj, Deep
9	Documentation	02/02/19	02/03/19	2d	Tanisha, Dolly
10	Related Work Study	02/08/19	03/08/19	29d	Tanisha, Dolly, Deep, Hardik
11	Data Collection	02/10/19	03/08/19	27d	Tanisha, Jemish, Raj, Deep
12	Code Development	02/15/19	03/22/19	36d	Jemish, Raj, Deep, Hardik
13	Evolution Analysis	03/10/19	03/22/19	13d	Hardik, Raj, Tanisha
14	Final Report	03/24/19	04/05/19	13d	Deep, Jemish, Dolly
15	Project Presentation	04/12/19	04/12/19	1d	Tanisha, Dolly, Jem

7. REFERENCES

[1] An Empirical Study of the Relationships between Design Pattern Roles and Class Change Proneness, Massimiliano Di Penta1, Luigi Cerulo1, Yann-Ga¨el Gu´eh´eneuc2, and Giuliano Antoniol3

[2] Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density, Kazuhiro Yamashita, Changyun Huang, Meiyappan Nagappan, Yasutaka Kamei, Audris Mockus, Ahmed E. Hassan, and Naoyasu Ubayashi, 2016.

[3] A Review of Studies on Change Proneness Prediction in Object Oriented Software, International Journal of Computer Applications (0975 – 8887) Volume 105 – No. 3, November 2014.

[4] Understanding change-proneness in OO software through visualization, 11th IEEE International Workshop on Program Comprehension, 2003.

[5] QMOOD metric sets to assess quality of Java program, 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT).

[6] Investigating the OO characteristics of software using CKJM metrics, 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)

[7] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. ACM Commun., 27(1):42–52, 1984.

[8] D. Coleman, B. Lowther, and P. Oman. The application of software maintainability models in industrial software systems. J. Syst. Softw., 29(1):3–16, 1995.

[9] B. T. Compton and C. Withrow. Prediction and control of ada software defects. J. Syst. Softw. ,12(3):199–207, 1990.

[10] https://netbeans.apache.org/download/index.html

[11] https://www.jetbrains.com/idea/download/previous.html

[12] T. McCabe. A complexity measures. IEEE Trans. Softw. Eng., SE-2(4):308 – 320, 1976.

1. MILESTONE 2

RELATED WORK STUDY

1) An Empirical Study of the Relationships between Design Pattern Roles and Class Change Proneness

1. Summary

a. Research goal (What is the problem being solved/investigated?)

This paper presents a study of the evolution of classes playing different roles in design motifs to understand whether (i) there are roles that are more change-prone than others and (ii) there are some changes that occur more to certain roles than to others. Kinds of changes include: change to method implementation, method addition/removal, attribute addition/removal, and extension by sub classing.

b. Motivation (Why is the problem important?)

Their main motivation was that only considering occurrences of the motifs as a "whole" and inferring characteristics of the changes from all the classes playing roles in motif do not analyse whether those classes also exhibit different change frequency, along with different kinds and amount of changes.

c. Approach/Methodology (How did they solve/investigate the problem?)

The design motif identification was performed using the DeMIMA (Design Motif Identification Multi-layered Approach) approach and tool; DeMIMA not only identifies the classes, but also the different roles played by classes in occurrences of motifs. In DeMIMA each design motif to be identified is modelled using a meta-model, PADL. The instances of PADL describe each motif in terms of the roles, methods, associations and inheritance relationships among roles.

d. Empirical Findings/Experimental Results (What is the outcome of the study?)

Paper report new evidence on kind and frequency of changes; evidence often confirms the intuitive behaviour – e.g., concrete classes tend to be more subject to changes than abstract classes – but sometimes, in a few more complex cases, it contradicts the expected intuitive behaviour.

e. Conclusions (What did we learn from the study?)

Results suggest that developers must carefully design classes playing within a motif role that will likely be subject to frequent changes and kinds of changes (e.g., in the method interfaces) that will likely impact elsewhere in the system.

2. Comparison

a. What are the similarities with your study?

Main similarity is both are trying to examine the change proneness between systems.

b. What are the differences with your study?

We are not considering any design motifs and we are examining on the larger system.

c. What are the limitations of this related work?

There are, however, many cases where the actual, observed changes deviates from the intuition and form the common wisdom. For example, in the Composite motif classes playing the role of Composite can be more complex than expected and because of that undergoing a high number of changes.

d. What are the improvements you are planning to do in your study?

We will try to extend the empirical investigation to a larger set of systems, to investigate on different roles co-changes, and to find evidence of correlations between kind of changes, frequency of changes and defects in classes playing particular roles in the motifs.

2) "Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density"

1) Summary

a. Research goal (What is the problem being solved/investigated?)

The primary goal of this research is to be able to provide guidelines to practitioners on what kind of code has better quality. This goal has two purposes: (a) identifying the metrics that are most useful for the practitioner (often usefulness is measured by how accurately the metrics identify bad code, how easy is it to collect the metrics, and how actionable the metrics are), and (b) what values of these metrics indicate good code or bad code. They aim to observe if a consistent relationship between metric thresholds and software quality is present in Open Source Software and industrial projects.

b. Motivation (Why is the problem important?)

It is reasonable to expect the most complex code to be buggy. Structuring code into reasonably sized files and classes also appears to be prudent. Hence, they replicated a recently published technique for calculating metric thresholds to determine high-risk files based on code size (LOC and number of methods), and complexity (cyclomatic complexity) using a very large set of open and closed source projects written primarily in Java.

c. Approach/Methodology (How did they solve/investigate the problem?)

They evaluated software quality using two criteria - (a) defect proneness (probability of a file having a defect), and (b) defect density (the number of defects/LOC). They conducted a case study on three OSS and four industrial projects. The metrics that they have chosen to evaluate are size based (Total LOC in a file, and Module interface size in a file), and complexity based (cyclomatic complexity and module inward coupling).

d. Empirical Findings/Experimental Results (What is the outcome of the study?)

They related the threshold-derived risk to the probability that a file would have a defect, and the defect density of the files in the high-risk group. They found that files identified as very high-risk by size and complexity thresholds were associated highest fault-proneness, files identified as very high-risk by size threshold were several times more fault-prone than files identified as very high-risk by complexity threshold and files identified as very high-risk by size threshold have a lower (often significantly lower) defect density than the remaining files. In some cases, files identified as very low risk by size and complexity threshold had a low defect density value as well. In these cases, the files with medium to high risk size and complexity values had the highest defect density.

e. Conclusions (What did we learn from the study?)

These findings confirm earlier results that the size bench marks are associated with high defect-proneness. It also indicates that the size is a better threshold of defect-proneness than complexity.

2) <u>Comparison</u>

a. What are the similarities with your study?

In this project, we propose to compare different versions of software systems like Eclipse, Intellij and NetBeans and co-relate the change proneness in the software during their evolution using the C&K metrics. They have also used total LOC in a file and cyclomatic complexity to evaluate defect-proneness.

b. **What are the differences with your study?**
They have conducted a case study on three OSS and four industrial projects, but we compare different releases of software system to evaluate quality measures.

c. **What are the limitations of this related work?**
Their Results suggest that, as expected, less code is associated with fewer defects. However, the same amount of code in large and complex files was associated with fewer defects than when located in smaller and less complex files. Hence, they concluded that risk thresholds for size and complexity metrics must be used with caution if at all. Their findings have immediate practical implications as well like the redistribution of Java code into smaller and less complex files may be counterproductive.

d. **What are the improvements you are planning to do in your study?**
They have only considered metrics Lines of code, Module Interface size and cyclomatic complexity for software quality evaluation whereas we plan to work on more metrics and consider quality measures as well.

3) - “Understanding change-proneness in OO software through visualization”

1) Summary

a. Research goal (What is the problem being solved/investigated?)

Main goal of this research is to identify and visualize classes and class interactions that are the most change-prone. The method was applied to a commercial embedded, real-time software system. It is object-oriented software that was developed using design patterns. The problem that is being solved is in knowing where those change-prone clusters are can help focus attention, identify targets for re-engineering and thus provide product-based information to steer maintenance processes.

b. Motivation (Why is the problem important?)

Various ways have been developed to visualize changing systems. The system can identify changes in the explicit structural coupling between components. It has been found that low-level or module-level changes are more frequent than changes in the architecture. The evolution of data clustering, which is the grouping of functionality with associated data. This work tracks design changes based on common data interactions and the calling structure in COBOL programs. The aim is to see how the design structure of a system can affect the change-proneness of individual classes. They have examined the same case study data focusing on the classes that are the most change-prone, distinguishing between local changes and change coupling between classes.

c. Approach/Methodology (How did they solve/investigate the problem?)

The approach of this study is to analyse both the implementation structure of the case study software system and change logs. The analysis of the implementation structure provides a characterization of the individual classes in the system and identifies the design patterns. The change logs provide the information needed to develop a change-architecture for the system; they identify individual changes and all classes that were changed for each reported change.

d. Empirical Findings/Experimental Results (What is the outcome of the study?)

Finding Patterns

(i) Search for pattern names in the documentation of the system. (ii) Identify the context of the classes identified in step 1 by analyzing the class diagrams. Once they found the classes whose documentation specifies something relating to a pattern name/role, they looked at the class diagrams to identify all the classes required to constitute a pattern and looked for the links and interactions between classes that implement the pattern. (iii) Verify that the candidate pattern is really a pattern in-stance. They examined the pattern implementation to look for lower level details. (iv) Verify the purpose of the pattern. They examined each group of classes that represent a pattern candidate to confirm that the classes and relations have the same purpose as described by an authoritative pattern reference.

Change-proneness Measures

Local change-proneness (LCP): A change report associated with changes in a single class adds one point to the local change-proneness measure LCPi.
Pair change coupling (PCC): Pair change coupling is associated with class pairs. Two or more classes are pair-coupled if they are involved in the same change report.
Sum of pair coupling (SPC): A class may be involved in many pair couplings.

e. Conclusions (What did we learn from the study?)

Firstly, the identification and visualization of change-prone collections of classes in an object-oriented system. The second had to do with distinguishing local change-proneness from change-prone clusters of classes. It has been showed how to quantify the degree to which classes are change-prone both locally and in their interactions with others.

2) <u>Comparison</u>

a. What are the similarities with your study?

The similarities of both the study is that both emphasizes on the change proneness of the system and uses QMOOD metrics and C&K metrics.

b. What are the differences with your study?

In our study, we are emphasizing on the system built in java and have broaden the metrics while the paper was emphasized on system built in C++ and have taken limited metrics.

c. What are the limitations of this related work?

The use of color, overlays of change-architecture versus logical architecture, representation of other measures are the limitations of this related work.

d. What are the improvements you are planning to do in your study?

We are trying to improve the results by considering almost all metrics related to changes proneness leading to better results.

4) - "Investigating the OO Characteristics of Software using CKJM Metrics"

1) <u>Summary</u>

a. Research goal (What is the problem being solved/investigated?)

The objective of this paper is to validate software product metrics in software engineering using existing tools with the help of statistical analysis.

b. Motivation (Why is the problem important?)

Software metrics proposed by various researchers have been broadly classified as project based and design based. The rationale behind this classification is the open-source availability, platform independence, portability and compatibility with the new technology for the programs written in Java. This paper proposed that metrics calculated were grouped under three categories: coupling, cohesion and inheritance metrics.

c. Approach/Methodology (How did they solve/investigate the problem?)

This paper calculates and analyzes 12 object-oriented software metrics to improve the software development process and quality. These metrics were tested for 10 sample open-source programs written in java. The frequency and descriptive analysis were then performed on the obtained results using SPSS tool.

d. Empirical Findings/Experimental Results (What is the outcome of the study?)

The selected 10 projects were fed into the Ckjm and IntelliJIdea tools to yield the value of twelve metrics which have been analyzed in this paper. One of the result shown suggest that the projects with higher coupling have large value of CBO metric and this may effects on dependent variables (proneness). high variance/ standard deviation implies higher coupling which in turn implies higher fault proneness value. Using these results, we can derive properties of coupling, cohesion or inheritance.

e. Conclusions (What did we learn from the study?)

This empirical validation provides the practitioner with some empirical evidence indicating that most of the metrics can be helpful indicators of quality. Furthermore, most of the metrics were found to be redundant indicators even on being comparatively independent from each other. The results thus obtained endow with motivation for additional investigation and refinement of C&K OO metrics. Projects with more cohesion are considered better projects.

2) <u>Comparison</u>

a. What are the similarities with your study?

We are comparing different versions of software systems during their evolution using the C&K metrics. Additionally, we also focus on analysing the attributes like understandability, functionality, reusability, extendibility that are affected due to change proneness. In this paper they’ve also used C&K metrics to analyse mentioned attributed.

b. What are the differences with your study?

In our study we compare different versions of software systems like Eclipse, Intellij and NetBeans and co-relate the change proneness in the software during their evolution while in this paper they took 10 different java projects to perform frequency and descriptive analysis using SPSS tool.

c. What are the limitations of this related work?

In this paper they took 10 different projects on JAVA language, other language’s projects might not compatible with analyzing tools for descriptive or frequency analysis. Furthermore, in this paper, any type of project had no impact on the LCOM metric.

d. What are the improvements you are planning to do in your study?

We will apply descriptive and frequency analysis for all metrics which are part of our project.

5) - "QMOOD metric sets to assess quality of java program"

1) <u>Summary</u>

a. Research goal (What is the problem being solved/investigated?)

This paper describes the model to evaluate and grade the java programs, based on QMOOD (Quality Model for Object Oriented Design). QMOOD is the hierarchical model that defines relation between qualities attributes (like reusability, functionality, effectiveness, understand ability, extendibility, flexibility) and design properties with the help of equations. In this research they developed the system based on QMOOD which is using to evaluate the quality of JAVA programs.

b. Motivation (Why is the problem important?)

It has been observed that Object-Oriented design is beneficial in software development environment and object-oriented design metrics is an important feature to measure software quality over the environment. Object oriented is an approach that is capable to classify the problem in terms of small object and it provides various paybacks on decomposition, reliability, reusability and adaptability of problem into easily under stood objects and providing some future modifications.

c. Approach/Methodology (How did they solve/investigate the problem?)

Compute all the design metrics from the java program taken as input and Normalize all the values of the design metrics. Find all the design properties from the design metrics and compute all the quality attributes. Then, Add the values of all quality attributes to compute the total quality index.

d. Empirical Findings/Experimental Results (What is the outcome of the study?)

The qualities of many programs that vary in their complexity and design have been assessed by the system. In each case the system has given different TQI (Total quality Index). It is observed from the above experimented figures that coupling and complexity are inversely proportional to the quality of software while others to some extent give the positive result that is directly proportional to quality.

e. Conclusions (What did we learn from the study?)

The system developed so far concludes that it's not necessary that every time increasing the positive parameter would result in better quality. Though QMOOD study is bit subjective in nature but still it provides us with the model for evaluating the values of the various parameters which define the quality of software. This system can be extended for other object-oriented languages (like C++). So, it can extend it by finding the minimum cut-off TQI, which will help the developers to directly decide the best design.

2) <u>Comparison</u>

a. What are the similarities with your study?

We both used QMOOD model which is developed to measure the quality with the help of various metrics. These metrics are the good indicators of quality of the software and are computable early in the design phase that helps in reducing complexity at the later stages.

b. What are the differences with your study?

The software quality evaluation is very important for software providers and their users. This paper provides a model based on QMOOD to assess large as well as small object-oriented java programs, while we will use QMOOD model to assess different versions of single JAVA software.

c. What are the limitations of this related work?

Some concepts presented in the standard are not clearly defined. So, whenever the standard was used it made difficult to return on investment and quality model reuse. The elements of software metric were not clear in the standard definition. For example, it was not clear what types of metric scales were allowed, or which classification to consider.

d. What are the improvements you are planning to do in your study?

In this research QMOOD metrics is used to assess the quality of java programs. It also describes the evaluation system which is used to evaluate java programs. In our study different versions of java systems are shown as input and result have been evaluated and featured with the help of 2D graph.

2. MILESTONE 3
DATA COLLECTION

1. INTERNAL METRICS IMPLEMENTATION:

In this section, we will be discussing about the internal metrics. Also, discuss about the projects and tools, which we used to collect the metric values. Later, we talk about how we collected the values, what challenges we faced during its implementation. We have implemented C&K metrics and QMOOD metrics for Eclipse, NetBeans and IntelliJ.
The tool that we have used for doing static code analysis is CodeMR.

TOOLS:

UNDERSTAND:

- Understand™ [8] is a powerful source code comprehension tool.
- It is used to quickly understand large or complex legacy code bases ... often with poor documentation.
- Programmers and team leaders regularly use it to visualize complex legacy code, perform impact analysis, and deliver powerful metrics.
- Perform impact analysis for changes and evaluates metrics values.
- Understand creates a repository of the structures and relations in a software project which it can help us to a better comprehension of the source code.

CODEMR:

- CodeMR[3] is a software quality and static code analysis tool which helps to developing better quality code and supports multiple programming languages.
- CodeMR is integrated with Eclipse and IntelliJ IDEA which Supports Java, Kotlin, Scala and C++ languages. CodeMR static code analysis tool extracts and visualises code in a big picture at Metric Distribution, Package Structure, TreeMap, Sunburst and Dependency views and Project Outline.
- It visualises high-level Object-Oriented quality attributes and low-level metrics.
- It extracts quality metrics and all types of relations directly from source code. You can see which parts need refactoring.
- It also Check code complexity, cohesion and coupling while investigating code quality. CodeMR has advanced modularization algorithms.
- It shows current package structure or extracts modules and services from source code. CodeMR helps to transform microservice architecture from their monolithic applications.

C&K Metrics

a) Weighted Method per Class (WMC): -

DEFINITION:

WMC[2] measures the overall complexity of a class. It sums the cyclomatic complexity of each method for all methods of a class.

$$WMC=\sum_{i=1}^{n} CCi$$

where, n is the number of methods in a class

CCi is the cyclomatic complexity of method i

A high value of WMC suggest that it is more prone to errors and less like to be reusable. WMC might be helpful to understand whether the class is change prone or not for future developments.

IMPLEMENTATION:

The logic behind calculating WMC is first counting the number of control statements in each method and then adding the value of all the methods. We have calculated WMC using the CodeMR tool, which is used for static code analysis. We have also calculated the data using Understand tool. WMC is calculated at package and class level for four releases of all the three software (Eclipse, NetBeans, and IntelliJ).

b) Depth of Inheritance Tree (DIT): -

DEFINITION:

DIT[2] is useful in calculating the inheritance of a class. DIT for a class X is the maximum path length from X to root of inheritance tree. The deeper a class is in the hierarchy, the more methods and variables it is likely to inherit, making it more complex. Deep trees as such indicate greater design complexity. The higher the value of DIT higher chances of change proneness due to inheritance related design violations. Nominal range of DIT is 0-4. If value of DIT is zero, which indicates that, there is no inheritance for that class. If the value is higher than 4 then it means that it will increase the encapsulation, which will eventually lead to problems when changes made to classes.

IMPLEMENTATION:

The logic behind calculating the DIT is by counting the number of super classes that extends a class. DIT has been calculated in understand and codeMR both the tools. In Understand, DIT is calculated as MaxInheritanceTree. From understand we have collected DIT as class level and package level.

c) **Number of immediate subclasses (NOC): -**

DEFINITION:

NOC[2] is the number of subclasses subordinated to a class in the hierarchy. The NOC for class X is the number of immediate subclasses of X in the inheritance tree. The higher the value of NOC, the higher reuse of the base class. However, if base class contains error, these will propagate to many subclasses. From NOC we will able to analyse the reusability of the code of class.

IMPLEMENTATION:

The logic behind calculating NOC is counting the number of classes inheriting the given class. NOC is implemented by at class level and package level in understand. In understand NOC is implemented as CountClassDerived. In CodeMR, NOC is calculated at class level.

d) **Coupling between objects (CBO): -**

DEFINITION:

Coupling between objects (CBO) [2] is a count of the number of classes that are coupled to a class i.e. where the methods of one class call the methods or access the variables of the other. These calls need to be counted in both directions, so the CBO of class A is the size of the set of classes that class A references and those classes that reference class A. Since this is a set - each class is counted only once even if the reference operates in both directions i.e. if A references B and B references A, B is only counted once. Ideally the value of CBO must as low as possible.

IMPLEMENTATION:

The logic behind calculating the CBO is implemented by computing the various calls made from class to other classes then counting the other classes using that class. In understand CBO is calculated as CountClassCoupled. The higher value of CBO indicated that class is more change prone thus increasing difficulty for maintenance.

e) **Response set for class (RFC): -**

DEFINITION:

The RFC [2] is a set of methods that can be potentially executed in a response to a message received by an object of that class.

- $RFC = |RS|$
- $RS = M \cup R$
- $R = \bigcup_{i=1}^{|M|} R_i$

Where, M is the set of methods in a given class,

Ri is the set of remote methods directly called from method Mi.

IMPLEMENTATION:

The logic behind implementing the RFC is firstly by calculating the total number of methods and then methods that are invoked from these methods are counted. In understand, RFC cannot be calculated thus we have calculated RFC only using CodeMR tool. Thus, RFC is implemented only at class level. The higher value of RFC indicates more understandability.

f) Lack of Cohesion in method (LCOM): -

DEFINITION:
In original incarnation of C&K metrics, defined LCOM [2] based on the numbers of pairs of methods that shared references to instance variables. Let Ii be set of instance variables accessed by method Mi. For all possible pairs of methods (Mi, Mj) compute the intersection of Ii ∩ Ij.

- $P = \{(M_i, M_j) \mid I_i \cap I_j = \emptyset\}$
- $Q = \{(M_i, M_j) \mid I_i \cap I_j \neq \emptyset\}$
- $LCOM = \begin{cases} |P| - |Q|, & if |P| > |Q| \\ 0, & otherwise \end{cases}$

IMPLEMENTATION:

The logic behind implementing LCOM would be that, for each class we get the list of all the methods, the size of the list is L, then we have L(L-1)/2 pairs of methods. Then for each method, we have the set called attributes, which stores all the class names of fields the method accessed, and of methods, the method calls. For each of the other methods, we use a set called attributesJ that stores the same data corresponding to the that class, then if the intersection of the two sets is not empty, these two methods share attributes and we put the pair into a set Q. After checking with all the methods, the LCOM of the class equals to the total number of method pairs minus 2 times the size of Q. In understand, LCOM is implemented as PercentLackOfCohesion.

QMOOD Metrics

a) Number of methods (NOM) [1]: -

DEFINITION:
It will count all the methods defined in a class. It’s design property is complexity.

IMPLEMENTATION:

We have implemented the NOM using CodeMR tool. This tool calculates values at class level.

b) **Number of polymorphic method (NOP) [1]: -**

DEFINITION:

It will count abstract methods. It's design property is polymorphism.

IMPLEMENTATION:

We have implemented NOP using CodeMR tool. In CodeMR, NOP is implemented as norm at class level.

c) **Design size in class (DSC) [1]: -**

DEFINITION:

DSC means to count number of classes in the design. It's design property is design size.

IMPLEMENTATION:

We have implemented DSC using CodeMR tool as #c (number of classes). It is implemented at class level.

d) **Cohesion among methods of class (CAMC) [1]: -**

DEFINITION:

It computes the relatedness among methods of a class based upon the parameter list of methods.

IMPLEMENTATION:

We have implemented CAMC using CodeMR tool as LCAM (Lack of Cohesion among methods). It is implemented at class level.

Other Metrics

a) **Number of interfaces: -**

We have also calculated number of interfaces using CodeMR tool at class level.

b) **Abstractness: -**

In software engineering and computer science, abstraction is a technique for arranging complexity of computer systems [5]. It works by establishing a level of simplicity on which a person interacts with the system, suppressing the more complex details below the current level. It can be applied to control data.

We have computed abstractness using CodeMR tool. It is implemented at class level. By measuring abstractness, we can define the complexity of the system.

c) **Line of code: -**

This metric represents the number of lines in the file including comments, empty lines, and actual code.
LOC = # lines in file/class

IMPLEMENTATION:

We have calculated line of code using understand and CodeMR tools. In understand it is implemented as CountLineCode while in CodeMR as LOC.

CHALLENGES DURING IMPLEMENTATION:

We calculated the C&K metric using Understand, but it does not calculate the QMOOD metrics, so we found another tool called CodeMR, which calculates few QMOOD metrics. When calculating the metrics using CodeMR we were not able to calculate it at package level, so we first calculated it at class level and later we calculated the values at package level with Understand. We also faced problems while exporting the collected data values into .csv format. CodeMR only exports the data into .xml format so we first generated the tabular format of the data and then copy pasted it in excel. Additionally, since the source code of our projects is very large while calculating the values the system kept slowing down.

CORRECTNESS OF METRIC:

We checked the correctness of C&K metric by evaluating the metric's values using Understand and CodeMR tool. We have compared class level metrics values for both the tools.

2. EXTERNAL METRICS IMPLEMENTATION:

In this section, we will discuss about the external metrics. We will also talk about the tool, which we have used to calculate the metrics. Lastly, there is how we implemented the metrics.

EXTERNAL METRICS:

External metrics are like dependent variables in a linear equation. External metric is dependent on the internal metrics. The external metrics as of now which we have computed are number of changes, maintainability index and halstead bug.

NUMBER OF CHANGES:

By computing, number of changes we aim to check the changes between two releases of a system by using “git diff” command and we have generated different text files indicating changes between different releases that we have included in our git repository(URL: https://github.com/Jemish27121997/SOEN6611_Team7.git).

MAINTAINABILTY INDEX:

Maintainability index is a software metric which measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula consisting of lines of code (LOC), cyclomatic complexity and halstead volume [7].

HALSTEAD BUG:

Halstead bug metric estimates how many bugs you are likely to find in the system [6].

JHAWK TOOL (Java Metric Tool):

JHawk [7] is a static code analysis tool - i.e. it takes the source code of your project and calculates metrics based on numerous aspects of the code - for example volume, complexity, relationships between class and packages and relationships within classes and packages.
Using JHawk tool we have calculated maintainability index, halstead bug, number of classes, average cyclomatic complexity, number of methods etc.

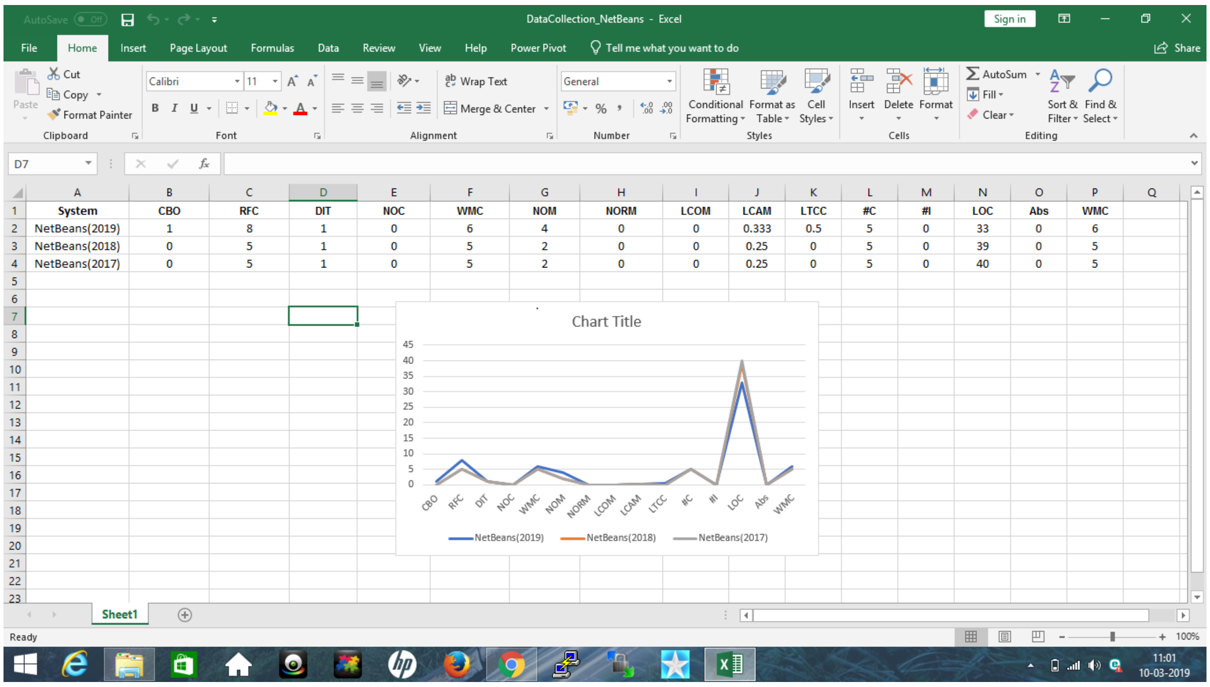
IMPLEMENTATION DETAILS:

The **Maintainability Index (MI)** [6] is a set of polynomial metrics, using Halstead's effort and McCabe's cyclomatic complexity, plus some other factors relating to the number of lines of code (or statements in the case of JHawk) and the percentage of comments. The MI is used primarily to determine if code has a high, medium or low degree of difficulty to maintain. It is language independent and was validated in the field by Hewlett-Packard (HP). HP concluded that modules with a MI less than 65 are difficult to maintain, modules between 65 and 85 have reasonable maintainability and those with MI above 85 have excellent maintainability. MI can be calculated at method, class, package and system level.

3. COLLECTED DATA:

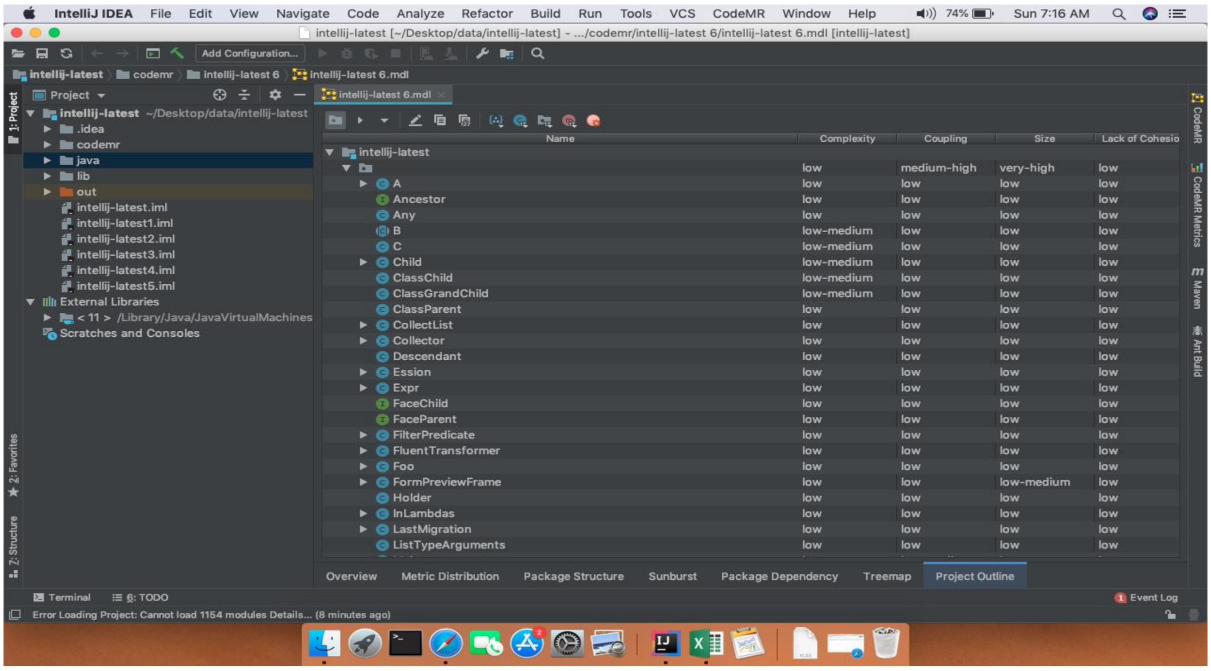
In this section, we have put snapshot of the data collected. Since there is lot of data we are just keeping few snapshots and for the rest of the data collected here is URL of our git repository (https://github.com/Jemish27121997/SOEN6611_Team7.git).

SNAPSHOT FROM CODEMR FOR NETBEANS RELEASES:


DataCollection_NetBeans - Excel

System	CBO	RFC	DIT	NOC	WMC	NOM	NORM	LCOM	LCAM	LTCC	#C	#I	LOC	Abs	WMC
NetBeans(2019)	1	8	1	0	6	4	0	0	0.333	0.5	5	0	33	0	6
NetBeans(2018)	0	5	1	0	5	2	0	0	0.25	0	5	0	39	0	5
NetBeans(2017)	0	5	1	0	5	2	0	0	0.25	0	5	0	40	0	5

SNAPSHOT of CodeMR Tool:

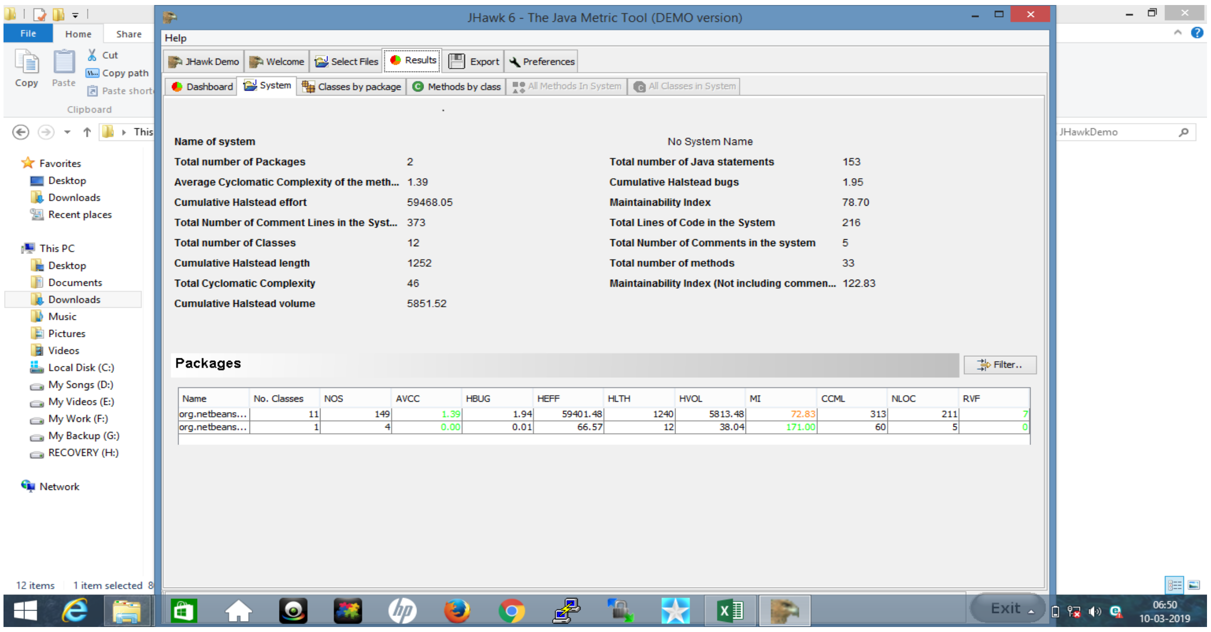

intellij-latest [~/Desktop/data/intellij-latest] - .../codemr/intellij-latest 6/intellij-latest 6.mdl [intellij-latest]

intellij-latest ~/Desktop/data/intellij-latest
.idea
codemr
java
lib
out
intellij-latest.iml
intellij-latest1.iml
intellij-latest2.iml
intellij-latest3.iml
intellij-latest4.iml
intellij-latest5.iml
External Libraries
< 11 > /Library/Java/JavaVirtualMachines
Scratches and Consoles

Name	Complexity	Coupling	Size	Lack of Cohesio
intellij-latest				
	low	medium-high	very-high	low
A	low	low	low	low
Ancestor	low	low	low	low
Any	low	low	low	low
B	low-medium	low	low	low
C	low-medium	low	low	low
Child	low-medium	low	low	low
ClassChild	low-medium	low	low	low
ClassGrandChild	low-medium	low	low	low
ClassParent	low	low	low	low
CollectList	low	low	low	low
Collector	low	low	low	low
Descendant	low	low	low	low
Ession	low	low	low	low
Expr	low	low	low	low
FaceChild	low	low	low	low
FaceParent	low	low	low	low
FilterPredicate	low	low	low	low
FluentTransformer	low	low	low	low
Foo	low	low	low	low
FormPreviewFrame	low	low	low-medium	low
Holder	low	low	low	low
InLambdas	low	low	low	low
LastMigration	low	low	low	low
ListTypeArguments	low	low	low	low

Error Loading Project: Cannot load 1154 modules Details... (8 minutes ago)

SNAPSHOT of JHawk Tool:



Name of system		No System Name	
Total number of Packages	2	Total number of Java statements	153
Average Cyclomatic Complexity of the meth...	1.39	Cumulative Halstead bugs	1.95
Cumulative Halstead effort	59468.05	Maintainability Index	78.70
Total Number of Comment Lines in the Syst...	373	Total Lines of Code in the System	216
Total number of Classes	12	Total Number of Comments in the system	5
Cumulative Halstead length	1252	Total number of methods	33
Total Cyclomatic Complexity	46	Maintainability Index (Not including commen...	122.83
Cumulative Halstead volume	5851.52		

Packages

Name	No. Classes	NOS	AVCC	HBUG	HEFF	HLTH	HVOL	MI	CCML	NLOC	RVF
org.netbeans...	11	149	1.39	1.94	59401.48	1240	5813.48	72.83	313	211	7
org.netbeans...	1	4	0.00	0.01	66.57	12	38.04	171.00	60	5	0

4. REFERENCES:

[1] QMOOD metric sets to assess quality of Java program, 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT).

[2] Investigating the OO characteristics of software using CKJM metrics, 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions).

[3] www.codemr.co.uk

[4] www.virtualmachinery.com

[5] www.portablecontacts.net

[6] http://www.virtualmachinery.com/jhawkmetricsmethod.htm

[7] http://www.virtualmachinery.com/jhawkprod.htm

[8] https://scitools.com/legacy-code-tool/