# Software Metrics Assignment
# A Review on Object Oriented Code Metrics and Object Oriented Code Metrics Extraction Tools

Koyyada Sai Pranav
*990429-9311*
*saky19@student.bth.se*

Vishruta Dochibhotla
*990807-1609*
*vido19@student.bth.se*

***Abstract*—This report encapsulates various Object Oriented Code Metrics proposed, analysed and or validated in numerous research studies in conjunction with topics of Software Cohesion, Complexity and Maintainability. The Object system, metric extraction tools, visualisation techniques and statistical methods used in these research works have been discussed. A personal reflection on using Object Oriented Code Extraction Tools Understand and Jmetric 1.3.6 are presented in section 8 of this report.**

***Index Terms*—Cohesion, Complexity, Maintainability, Object Oriented Code Metrics**

## I. INTRODUCTION

Software metrics are measurable representations of attributes and process associated with a system. [8] They are critical resource for evaluation and decisions in dynamic business infrastructures. [9]. Inconsistent metrics translates to unreliable assumptions and understanding. [8]. Hence, numerous studies have proposed, examined, analysed and validated metrics for their applicability. [10] Metrics are classified based on their individual subject of representation.

Cohesion defines intra-module dependency and association between data objects and member functions of a class defined to deliver a requirement. This contrasts with coupling and inherently mean high cohesion minimises inter-modules dependency.[7]

The Software Complexity is measured at finer generalities to address its broad nature. Essentially, it defines the degree of difficulty in multiple aspects of code which directly or indirect transitions to the overall quality. [9]

Maintainability is the ability of code to reproduce and adapt to changes in a restricted time. These changes are made to fix faults, and improve the overall system.[3]

This report briefly review the research works on Object Oriented Code Metrics of the above mentioned classifications. Summaries of the papers are presented in section 2 followed by identified object systems in section 3, Metric extraction Tools used in section 4, Visualisation techniques used in section 5 and Statistical methods used in section 6. All the Object Oriented Metrics in these research papers are listed in section 7. A personal reflection on using Object Oriented Code Extraction Tools Understand and Jmetric 1.3.6 on an Open Source System resourced from GitHub are present in section 8 of this report.

## II. SUMMARIES

### A. *Complexity - Paper1: Critical Analysis of Object Oriented Metrics in Software Development*

***Context:*** Software Metrics are quantifiable numeric or symbolic representation of attributes associated with a software preserving the defined rules. They reflect the quality and complexity of the software. The increasing surge in Object oriented paradigm for software development demand metrics relevant to quantify class and design attributes. Software metrics require critical resources including effort and time for evaluation. Inconsistent metrics direct fabricated representation of software resulting in faulty analysis and understanding of its quality.

***Objectives:*** The primary objective of the research study is to review and analyse Object Oriented Metrics proposed by researchers to validate their applicability. Additionally, identify a cluster of Object Oriented Metrics showcasing high correlation with quality and complexity of Software.

***Findings:*** The Weighted Methods per Class (WMC), Number of Children (NOC), Coupling Between Objects (CBO) and Response For a Class(RFC) of Chidamber and Kemerer (CK)[14] suite are applicable to quantify complexity and quality discarding Depth of Inheritance Tree of a class (DIT) and Lack of Cohesion Of Methods (LCOM) for measuring Object Oriented Design. Also, NOC cannot be considered as a measure for fault proneness. The 5 out of 6 metrics of Mood suite [15] are valid measures of quality except for Coupling Factor (CF). Lorenz and Kidd suite [16] is completely rejected for measuring Object Oriented design.

***Limitations:*** The research work only reports previous empirical and analytical studies. Also, the scope of validation is restricted to 3 suites of metrics. The conclusions are drawn solely based on the results from other studies without replication or verification. There is no convincing justification for rejecting Lorenz and Kidd suite when no previous validation study exists as per the author's knowledge. The secondary objective to identify a limited set of metrics is under-delivered with more or less encompassing the complete approved suite as a relevant measure to represent quality and complexity without any analysis.

### B. *Complexity - Paper2: Applicability of Three Cognitive Complexity Metrics*

***Context:*** The constant requirement for up scaling and improvisation of services have mandated organisations to adapt business structures governed by software. The intricacy of software development has evolved parallely to mitigate critical cost, schedule and quality concerns in these ecosystems. Diverse variety of complexity metrics are proposed to evaluate the software structures with motive to optimise the construction, deployment and maintenance cost. A few research studies have reviewed these complexity metrics to map their relevance with softwares. Also these metrics have been theoretically verified but not evaluated in real time environments.

***Objectives:*** The objective of the research study is to draws a relationship between three cognitive complexity metrics: Shao and Wangs' Cognitive Functional Size (CFS), Kushwaha and Misra Cognitive Information Complexity Measure (CICM) and Misra's Cognitive Weight Complexity Measure (CWCM) and identify the metric relevant for real world application.

***Findings:*** All the three metrics were built on cognitive weights. The Shao and Wangs' Cognitive Functional Size (CFS) showed highest correlation coefficient of 0.903 based on Spearman's Rank correlation coefficient value derived from comparison of the correlation coefficient between the three complexity metrics and expert's score from 30 software practitioners opinions on 10 different java programs. Kushwaha and Misra Cognitive Information Complexity Measure (CICM) and Misra's Cognitive Weight Complexity Measure (CWCM) had correlation scores of 0.588 and 0.503 respectively, concluding Shao and Wangs' Cognitive Functional Size (CFS) to be the most relevant metric for real world usage.

***Limitations:*** The analysis of study lacks transparency. The individual score of each practitioner on each java program is not documented or cited. Thus questioning the author's choice of descriptive analysis to use the mean of these scores as a quantified value of expert score. Also, no description or repository on individual java programs are defined to replicate the study.

### C. *Complexity - Paper3: Studying the impact of dependency network measures on software quality*

***Context:*** Code reviews and unit testing are critical to identify and patch bugs and breakages. Prioritization of software improvement activities is crucial in environments subject to continuous change. Variety of metrics are evaluated, verified and proposed to quantify the quality of the software. Zimmerman et al. [11] analysed and reported improvement in performance of product metrics when used in combination with metrics derived from dependency graphs associated with the subject system. This study was evaluated on Windows Server 2003 operating system. Although generalizability of these results and the factors causing them are not studied.

***Objectives:*** The primary objective of the current study is to replicate the previous studies on the Eclipse Project to investigate the results of [1] and report the cause of improvement. Additionally, examine the practical significance of these results at different granularities of the system.

***Findings:*** There was a significant correlation evident between Social Network Analysis (SNA) metrics and post-release failures. Also SNA metrics improve prediction of post release failures. Both the findings were in accordance with the study [11]. Although, SNA metrics fail to identify more critical classes and packages almost twice in comparison with traditional complexity metrics as reported in [11]. It performs worse than the traditional complexity metrics. The analysis on practical significance showed better practical value of predictions at finer granularities. The improvement in performance is a result of improved explanation power added by the SNA measures to the Hierarchical model categorised metrics.

***Limitations:*** The current study evaluates the result on only one system again questioning the generalisability of the results beyond the scope of the systems evaluated in this research and [11]. Despite the complexity and effort required for replication of the study for another subject, there should have been a concise attempt made to concrete the results of generalisability study.

### D. *Cohesion - Paper1: Theoretical Analysis for the Impact of Including Special Methods in Lack-of-Cohesion Computation*

***Context:***Object Oriented systems are centered around classes and objects. The quality of a system is governed by the quality of class. Cohesion defines intra-module dependency and association between data objects and member functions of a class defined to deliver a requirement. Class cohesion metrics identify inconsistent and unoptimised class designs hence evaluating the quality of the system. Many studies have proposed and analysed different class cohesion metrics. But, there are limited research works reporting the outcomes of using various class cohesion methods in combination with special methods to calculate cohesion.

***Objectives:*** The primary objective of this research work is to examine the outcomes of including/excluding special methods in combination with metric values and thus, recommend productive combination of metrics and special methods.

***Findings:*** The outcome of combining special methods to cohesion metric values was confirmed to be dependent on the nature of special method using the lack-of-cohesion (LCOM) metrics. Also the special metrics are expected to impact the metric values negligibly.

***Limitations:*** The metrics set chosen to combine the special methods with was limited. A concrete conclusion could have been derived provided more metrics were examined.

### E. *Cohesion - Paper2: Towards a valid metric for Class Cohesion at Design Level*

***Context:*** The class cohesion metrics are built on the assumption where all member functions of the class process closely related information provided they have access to similar parametric types at the OO paradigm design Level. It is considered there exists a positive correlation between the number of similar type parameter accessible in member function and class cohesion.

***Objectives:*** The objective this research is to identify a valid metric relatively more valid in the design phase of cohesion by comparing existing metrics with proposed variant of NHD cohesion metrics in terms of definition and anomalies.

***Findings:*** The anomalies in existing NHD, CAMC, SNHD metrics were normalised by the proposed NHD modified(NHMD) metric and it satisfied cohesion metric properties both theoretically and empirically. Also the mean of NHDM was less in comparison with existing metrics.

***Limitations:*** Limited number of existing metrics were compared with the proposed NHDM metric.

### *F. Cohesion - Paper3: Class Cohesion Complexity Metric (C3M)*

***Context:*** The requirement to serve complex use cases have translated into exponential increase in size and complexity of softwares. With constant growth these attributes of software are multiplying. Reliability of software turn questionable in such ecosystem leading to its failure in the market.

***Objectives:*** The objective this research is to examine and justify that cohesion is positively correlated to reliability and thus with improved cohesion, reliability can be increased and complexity can be controlled. To achieve this, a new metric is proposed to evaluate the complexity and reliability of an algorithm.

***Findings:*** The proposed class cohesion complexity metric(C3M) derived that the software is less prone to faults and errors provided the class cohesion is high. Thus, explaining the negative correlation between cohesion and complexity to assure reliability.

***Limitations:*** The proposed metric was evaluated on a case study which is not always generalisable. Also the C3M was not evaluated against other metrics to present a comparison and its validity.

### *G. Maintainability - Paper1: Prediction of Software Maintainability Using Fuzzy Logic*

***Context:*** Object oriented metrics hold a non-linear and complex relationship with software maintainability. The prediction of software maintainability deals with issues like uncertainty and imprecision critical to minimize maintenance effort of the system. The fuzzy inference systems can support achieving better software maintainability predictions.

***Objectives:*** The paper aims to develop a prediction model using fuzzy logic for predicting software maintainability.

***Findings:*** The fuzzy interference system trained and tested on metric data collected from two datasets i.e., User Interface Management System (UIMS) and Quality Evaluation System (QUES) proved to perform well overall with Mean Magnitude of Relative Error (MMRE) on testing data of UIMS and QUES being 0.36 and 0.89 respectively.

***Limitations:*** The prediction model developed in this paper is built using the Mamadani fuzzy interface system [13] and no comparison has been made with the other machine learning techniques.

### *H. Maintainability - Paper2: Estimation of Responsibility Metrics to Determine Package Maintainability and Testability*

***Context:*** Software design is important part of software development process. Improper software designs lead to poor maintainability. Hence, choosing the right software design which leads to lesser maintenance and testing issues is highly significant.

***Objectives:*** The objective of this paper is to build modified R.C Martin metrics [17] that measures maintainability and testability during the design phase of the development which may help to reduce the maintenance and testing efforts put at later stages to a great extent. Also, compare these metrics and test them against 3 large open-source software for empirical validation.

***Findings:*** The paper performs the evaluation by calculating the correlation between the metrics(newly developed and Martin's metrics) and maintainability(number of revisions and RLOC(Revised Lines of Code)) and testability(number of lines of test code(TLOC)) per package. The coupling measures newly developed showed a positive moderate to high correlation to the two measures of package maintainability for both the TOMCAT and JHotDraw datasets. Also the newly developed metrics showed greater correlation to the measures when compared to Martin's metrics. The positive and significant correlation observed indicates that the more coupled and abstract are the classes, the greater is the maintenance cost. The efferent coupling metric indicates the testing effort and Martin's package coupling correlation with TLOC is observed to be not as strong as the newly developed metrics.

***Limitations:*** The paper limits its metric developed to direct dependencies neglecting indirect dependencies of the classes. Additionally, coupling is considered as the only factor for maintainability which may not be true.

### *I. Maintainability - Paper3: Assessment of Maintainability Metrics for Object-Oriented Software System*

***Context:*** Maintainability of the software is a crucial aspect of the entire software process. Maintainability of the software results in majority of the software cost, almost 80% of the whole budget. Therefore, the less effort and cost that goes into maintaining the software the more successful it will be.

***Objectives:*** The objective of the paper is to find a right metric system to calculate the maintainability .

***Findings:*** From the assessment, by keeping low values of CK metrics the system developers can definitely improve the maintainability of software systems for a qualitative utility in this crucial field. By collecting important data with regards to the maintainability measurements based on the values of the CK metrics, a highly efficient system may be created for practical use.

***Limitations:*** Limited metrics taken into consideration for assessment

### *J. Maintainability - Paper4: Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability*

***Context:***Software maintainability is one of the major quality factors defined for a software. Most of the development process on date emphasize on Object-Oriented paradigm. Prediction of the software maintainability using the various Object Oriented metric is considered as a usefulness of the metric and also significance lies in predicting the software maintainability which stands as an essential factor in software development process and for a software code.

***Objectives:*** The objective of the studied paper is to develop a prediction for software maintainability using Neuro-Genetic Algorithms(Neuro-GA) and ANN(Artificial Neural Networks) and analyse them in comparison to 4 categories(SIZE metrics, CK Metrics, Li and Henry Metrics [18], SIZE along with CK Metrics) of metrics.

***Findings:*** The analysis was made by taking metrics from the User Interface Management System (UIMS) and Quality Evaluation System (QUES) datasets. The estimation done using the SIZE metrics showed poor performance in comparison to CK Metric and Li and Henry metrics. When the overall metrics were considered, the performance was observed to be good. It is seen that the combination of the CK Metric, Li and Henry metric and the SIZE metric yielded better accuracy of prediction of the maintainability.

***Limitations:*** The study doesn't take into consideration many major Object Oriented based metrics for analysis and prediction of maintainability. The datasets taken are also developed in ADA language and popular Object-Oriented languages like Java and others were not evaluated.

## III. Identified Object System

**Windows Server 2003 operating system: Large proprietary enterprise/commercial utility system**
Windows Server 2003 operating system is a part of Windows NT family developed by Microsoft as a Server Operating system for enterprise use. The windows kernel is in C. The dynamic link layers (DLL) are sharable encapsulations of the object libraries used by the main executable of a system. Bugs are reported, managed and closed if the DLL is fixed. Hence, DLL are used as the module of study in [11].

**Eclipse Project: Large open source enterprise/personal utility system**
Eclipse project is an open source project defining a common set of frameworks required to support the Eclipse as a tool integration platform running on a component model. Eclipse is coded in Java and hence has classes that are grouped into packages. Thanh et al [10] replicated the study of [11] using Eclipse Project at two levels of granularity: package level where the module is a collection of classes grouped as package and class level where an individual class is considered as the module for evaluation.

In [2], the authors have used the follow three Object Oriented softwares to apply and study the new proposed metrics

- **TOMCAT 7.0.6:**
  TOMCAT is an open-sourced implementation of Java Servlets, JavaServer Pages, Java Expression Language and WebSocket technologies. TOMCAT is a java-based HTTP webserver environment. TOMCAT server had been developed by developers at the Apache Software Foundation and released
- **JHotDraw 7.5.1:**
  JHotDraw 7 is a Java framework for structured drawing editors and for document-oriented applications. The framework for structured drawing editors can be used to realize drawing editors for sketches, diagrams, and artistic drawings. The framework for document-oriented applications can be used to create applications which comply to platform-specific user interface guidelines.
- **Hadoop 2.2.0:**
  Hadoop is a collection of open-source software utilities that facilitates using network of many computers to process BigData.

**User Interface Management System (UIMS) and Quality Evaluation System (QUES):**
UIMS and QUES are two Object Oriented software datasets introduced by Li and Henry [12]. These softwares were coded in the Classical-AdaTM language. The metric data was collected from a total of 110 classes contained in both the softwares together with an individual count of 39 and 71 classes respectively. UIMS isolates system UI from application functionality whereas QUES evaluated internal and external quality of the software. Hamdi A. AI-Jamimi and Moataz Ahmed [1] used these systems to take the metrics that were used for the prediction model built using fuzzy logic to predict the software maintainability. Santanu Ku. Rath et. al.[4] used these systems for predicting the software maintainability by using a hybrid approach of the ANN and Neuro-GA algorithms.

## IV. Object Oriented Metric Extraction Tools

**UCINET** is a Social Network Analysis tool integrated with NetDraw for analysing and visualising network data. Thanh et al [10] extracted the Ego Network metrics of data for post-release failures of Eclipse 2.1 using UCINET.

**Structure 101** is an Agile Development Environment (ADE) for codebase visualisation and organisation. Thanh et al [10] used Structure 101 to understand the ties between modules of the Eclipse project. This network was fed to UCINET to compute metrics of individual modules in the network.

**Understand** is a static code analyser for source code investigation. Thanh et al [10] computed the traditional complexity metrics of the Eclipse project using Understand

and PROMISE dataset.

**CohMetric** is a computational cohesion and coherence metric system for written and spoken texts. CohMetric helps readers, writers, educators and researchers to quickly calculate the complexity of writing text to target audiences. The CohMetric tool is developed using the C programming language to collect metrics automatically.

**ObjectAidUML Explorer** is a Eclipse IDE tool for code visualisation. It is lightweight and graphically represents the java code in UML notions. Melton et al[2] used this tool was used to extract the UML diagrams from the java source code in their study.

**JHawk** is a general purpose metric collection tool that calculates the metrics from object oriented systems. In [2], JHawk was used to extract size and complexity metrics from each system and to validate the tool calculations.

**JDepend** is a java design quality metric tool used to measure the re-usability, maintainability and coverage of each java package and verify the relations between the packages. In [2], JDepend was used verify package relationships.

**Classycle** is a static and package dependency analyser for java applications. In [2], Classcycle was used to analyse the package dependencies.

## V. Visualisation Techniques

**Line Chart** or line graph plots data of series function by connecting individual markers mapped using coordinate values with straight lines. Trend analysis is a common utility out of line charts. De Silva et al[9] used line charts to draw comparison between three cognitive complexity metrics showcasing the variance in their complexity values. The markers were mapped using the coordinate values as 10 java programs on the x-axis corresponding to their respective complexity value using each metric as y-values. The chart reported CICM and CWCM to be closely associated and similar in comparison with the CFS complexity metric.

**NetDraw** is a visualisation tool for Social network data. Thanh et al [10] used NetDraw to visualise the dependency networks at two different granularities: class level where classes are the modules and package level where packages are the modules for evaluation.

**Cumulative Lift Chart/Lift** is a representation of effectiveness of a predictive model. It is calculated as the ratio of results obtained with and without the predictive model. Thanh et al [10] used Cumulative lift charts to visualise the potential number of bugs identified at class and package level using different combinations and standalone usage of metrics.

**Box plots** visualize numeric data as quartiles with whiskers on both ends describing the variability. Additionally, outliers can be represented using points. They are non-parametric hence show variation in data. Thanh et al [10] used box plots to visualise statistical measures associated with package and class level data of the Eclipse project. Santanu Ku. Rath et. al [4] uses box-plots to visualize the UIMS and QUES datasets.

**Histogram** is a graphical representation of data distribution among group or intervals. Santanu Ku. Rath et. al.[4] uses histograms to show the number of hidden nodes present in each fold in the cross-validation folds considered in the Neuro-GA prediction algorithm.

**Scatter Plot** maps Cartesian values between a combination of attributes of a set of data Santanu Ku. Rath et. al.[4] used scatter plot to visualize the number of No. of chromosomes contain same fitness value VS Iteration No. in the Neuro-GA prediction algorithm.

## VI. Statistical Measures

**Descriptive Statistics used:**

- **Mean:** Mean defines the centrality of discrete numeric set n with a value A computed by:

$$A = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{a_1 + a_2 + \cdots + a_n}{n} \tag{1}$$

- **Median:** Median shows the middle element of an ordered discrete numeric set n.
- **Min:** Minimum is lowest observed value for an attribute in a sample.
- **Max:** Maximum is highest observed value for an attribute in a sample.
- **Standard Deviation:** Standard deviation is the variance between the values of an attribute and their mean values.

$$\sigma = \sqrt{\mu_2} \tag{2}$$

De Silva et al [9] computed the average of expert scores to derive an expert opinion on individual complexities of 10 java programs later used to study the correlation with each complexity metrics. The results were used to mathematically justify the applicability of each metric in the real world.

**Spearman's rank correlation coefficient value** defines the strength of relationship between two variables on a completely non-increasing or non-decreasing function. It uses transforms to convert observations to rank and hence can be applied on discrete data. For sample size n, rXi and rYi rank of n Xi and Yi, Spearman's Rank rs is given by:

$$rs = Cov(rX, rY)/\sigma rX\sigma rY \tag{3}$$

De Silva et al [9] used Spearman's rank correlation coefficient value to compare strength of link between the expert scores and 3 complexity metrics to identify the most applicable one in the real world. Thanh et al [10] used Pearson correlation coefficient to measure accuracy of the Prediction.

**R-square** is a statistical method to determine the closeness of data points with respect to the fitted regression line. It explains the movement of the dependent variables with the movement of independent variables. It is calculated by:

$$R^2 = ExplainedVariation/TotalVariation \tag{4}$$

**Adjusted R-square** is a modification made to R square to encompass independent variables in multiple regression for complex models with many independent variables. Thanh et al [10] used R square to report the performance of the prediction model.

**Pearson correlation coefficient** measures the statistical associativity between two variables. Its value always lies between -1 and +1 with former being negatively correlated and later means positively correlated. It can be applied on normally distributed data. A0 signifies no relation between the variables. It is calculated by:

For sample size n, rXi and rYi rank of n Xi and Yi, Pearson's Rank rs is given by:

$$rs = Cov(rX, rY)/\sigma rX\sigma rY \tag{5}$$

Thanh et al [10] used Pearson correlation coefficient to measure accuracy of the Prediction.

**Precision** is the fraction of fitting instances among all the retrieved instances. **Recall** is the fraction of fitting instances among all instances. For fitting instance fi; retrieved instance riand total instances ti

$$Precision = firi \cap ri \tag{6}$$

$$Recall = firi \cap ti \tag{7}$$

Thanh et al [10] used Recall and precision to measure predictions.

**Analysis of Variance (ANOVA)** is a statistical models and procedures framework used to examine the divergence of group means in a sample. **F-tests** are a part of ANOVA and are any test using F-distribution. Thanh et al [10] used these tests to examine ego metrics extensively responsible for driving the prediction.

**Principle Component Analysis** combines and drops least relevant inputs for prediction. High Collinearity between independent variables depletes the prediction power. Hence PCA is employed to combine the input variables of a model into principal components and enables dropping the once which are least important. Thanh et al [10] used PCA to improve the prediction power of the model.

**Root-mean-square error (RMSE)** shows differences between values predicted by a model and the values actually observed from the thing being modelled.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\Big(\frac{d_i - f_i}{\sigma_i}\Big)^2} \tag{8}$$

In [1], this metric has been used to check the accuracy of the prediction.

**Normalized root-mean-square error (NRMSE)** to normalize the RMSE to the range of the observed data.

$$NRMSE = RMSE/f(x)_{max} - f(x)_{min} \tag{9}$$

In [1], this metric has been used to check the accuracy of the prediction.

**Mean Absolute Error** is the average measure of errors

$$mae = (\frac{1}{n})\sum_{i=1}^{n}|y_i - x_i| \tag{10}$$

Santanu Ku. Rath et. al. [4] calculated MAE for defining the accuracy of the predictions.

**Mean Absolute Relative Error**

$$RelativeError = AbsoluteError/MeasuredValue \tag{11}$$

Santanu Ku. Rath et. al. [4] uses MARE measure for calculating the accuracy of the predictions.

**Standard Error of the Mean** For N is Sample size

$$\sigma_M = \sigma/\sqrt{N} \tag{12}$$

Santanu Ku. Rath et. al. [4] has also been used in the study to understand the accuracy.

## VII. Various Object Oriented Metrics

TABLE I: Object Oriented Metrics identified

| **Object-Oriented Metrics** | **Metrics Description** | **Categorization** | **Resource** |
|---|---|---|---|
| Congnitive Function Size (CFS) | Proportional to product of Cognitive Weights of Basic Control Structure and Number of inputs and outputs. | Complexity | [9] |
| Cognitive Information Complexity Measure (CICM) | Product of Weighted information count and Cognitive weights | Complexity | [9] |
| Cognitive Weights Complexity Measure(CWCM) | Equal to cognitive weights | Complexity | [9] |
| Size/Ties/Pairs | Defines the Number of modules, dependencies and unique pairs of modules in the ego network to provide a descriptive overview of the system | Descriptive / Size | [10] |
| Density | It is the number of Ties occurring in an Ego network. These dependencies range between inter-module or third-party systems | Descriptive / Size | [10] |
| X2StepReach | It is the Number of modules reachable/callable in two steps or size. | Centrality | [10] |
| Broker | It is the number of indirectly associated or connected pairs of modules in a system. This relation is a cause of a mutually associated module between the pair. | Centrality | [10] |
| EgoBetween | It is the number of shortest traversable paths between a pair of modules | Centrality | [10] |
| EffSize | Defined as the difference between number of modules connected in the ego network and average number of ties or dependencies between those modules | Structural holes | [10] |
| Constraint | Restrictions on a module to access or reach another module in the ego network | Structural holes | [10] |
| Degree | Defined as the number of modules directly connected to a module in a global network | Centrality | [10] |
| Eigen. cent | Defined as the number modules connected to a module taking individual connectivities into account. When a module is connected to another module with many connections, Eigen Cent. is high. It describes the criticality of the module in the network and system | Centrality | [10] |
| Closeness | Summation of all traversable shortest paths from a module to all other modules in the network | Centrality | [10] |
| Betweenness | Occurrence of a module in the shortest path between another pair of modules. It identifies the carrier or pivoting points/modules in the system | Centrality | [10] |
| Information | Variant of closeness calculated as the harmonic mean of length of the shortest traversable paths of a module to all other modules in the network | Centrality | [10] |
| dwReach | It is the number of modules that can be reached from the current module assigned with a specific weight. The weight depends on the number of steps required to reach a required module | Centrality | [10] |
| FanIn | It is the number of other functions invoking the current function as a dependency or follow up to a task | Complexity | [10] |
| FanOut | It is the number of other functions invoked by the current function as a dependency or follow up to a task. | Complexity | [10], [1], [3], [4] |

| | | | |
|---|---|---|---|
| ClassMethods | Number of methods in class | Size | [10], [13],[1],[3] |
| SubClasses | Number of classes derived from a parent class | Size | [10], [3] |
| Coupling Between Object Classes(CBO) | An object of one class uses methods or properties of an object of another class is called coupling, keeping the count of these other classes that a class is coupled to is called as Coupling between Object classes(CBO). | Coupling | [10],[3] |
| Lack of Cohesion of Methods (LCOM) | Measures how closely the methods of one class are related to each other | Cohesion | [7] |
| Cohesion Among Methods of Class (CAMC) | Using the parameter types of methods we evaluate the level of cohesiveness (how well the methods are connected) among the methods when the system is in its design stage | Cohesion | [6], [3] |
| Normalized Hamming Distance (NHD) | A parametric operation matrix where all the rows are being compared and then columns with two rows having equal value are being counted then summed up over all row pairs and are then normalized to the max value of 1 | Cohesion | [6] |
| Similarity-based Class Cohesion (SCC) | This metric is used to calculate the value of cohesiveness on the design of object-oriented software by identifying the relationships present between methods, method-attribute, and interaction attribute-attribute. | Cohesion | [6] |
| DIT (Depth of the Inheritance Tree) | This Metric measures a path with maximum length from a node to a root node in a class hierarchy. A minimum value of DIT for a class is 1 and if the value is 0 i.e it is at the root. | Complexity | [1],[3] |
| MPC (Message-passing coupling) | It is the count of messages passing among objects of a class | Coupling | [1], [4],[3] |
| DAC (Data abstraction coupling) | Measures the coupling complexity caused by Abstract data types | Coupling, Complexity | [1], [4], [3] |
| WMC (Weighted method per class) | Sum of McCabe's cyclomatic complexity of all local methods | Size | [1],[3], [4] |
| SIZE2 (Number of properties) | Number of attributes + Number of local methods | Size | [1], [4] |
| Efferent Coupling (Ce) | # of external classes coupled to classes of a package due to outgoing coupling | Coupling | [2] |
| Afferent Coupling (Ca) | # of external classes coupled to classes of a package due to incoming coupling | Coupling | [2] |
| Instability | The ratio of efferent coupling (Ce) to total coupling is Instability | Maintainability | [2] |
| Revised Lines of Code (RLOC) | Evaluation of the size of code generated during its maintenance period | Maintainability, Size | [2] |
| Test Lines Of Code (TLOC) | # of lines of code of a test package | Maintainability, Size | [2] |
| MIF (Method Inheritance Factor) | Ratio of the sum of all inherited methods in all classes to the available methods in all classes | Complexity | [3] |
| COF (Coupling Factor) | This metric calculates the coupling only in between classes, not the one due to inheritance. It is the ratio between the number of actually coupled pairs and possible couple pairs. | Coupling | [3] |
| Cyclomatic Complexity (CC) | Number of independent linear paths | Complexity | [3], [9] |
| Comment Percentage (CP) | Ratio of number of commented lines to total number of lines | Size | [3] |
| Class Method Complexity (CMC) | This metric is defined as the summation of internal structure complexities of all local methods | Complexity | [3] |

Various Object Oriented Metrics are listed in Table 1. The Metric category descriptions is as follows:

- **Size:** This category of software metrics measure the size of the system or the software like the number of lines of code present in the system(LOC) or the number of classes etc.
- **Complexity:** Complexity metrics in software metrics is a representation of the interactions between the entities of the system.
- **Descriptive metric:** These metrics are based on Social Network Analysis metrics which are used to define the overall outline and aspects of the system.
- **Centrality:** The centrality metric are a concept of Social Network Analysis metric that defines the core or central entity of the software on which the other entities depend. This metric is a measure to understand the central component of the system.
- **Structural holes:** These metrics define interactions between components of a system in context of Social Network analysis Metrics.
- **Structural holes:** These metrics define interactions between components of a system in context of Social Network analysis Metrics.
- **Cohesion:** Cohesion metric in software engineering measures how well the classes are related to each other. It is a measure of the degree to which elements belong to each other inside a module.
- **Maintainability:** Maintainability is defined as the number of revisions made to a code during a maintenance period.

## VIII. Reflection on Object Oriented Metric Extraction Tools

The two metric extraction tools used to extract the metrics are

- Understand
- JMetrics 1.3.6

The system selected is an Object-Oriented system developed in Java. The Open Source System selected is a pet clinic application developed using the spring boot and thymeleaf java frameworks and is available on GitHub [18] .

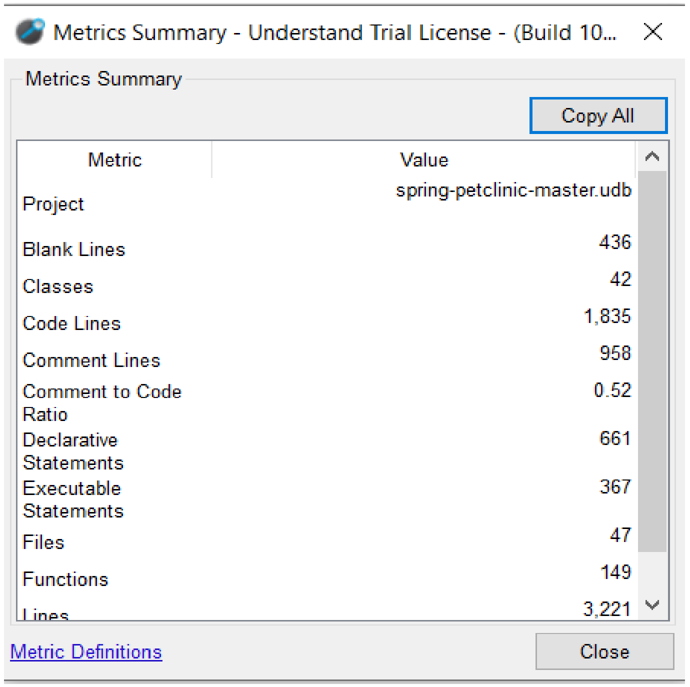
**Understand** is a static code analyzer developed by Scientific Toolworks. Inc. which was released on February 14, 2019. The environment had been originally developed using the Ada language. It is an Integrated Development Environment (IDE) with code analysis enabled through an array of visuals, documentation and the metric tools. The tool also provides metrics and reports, standard testing, documentation, searching, code knowledge and graphing. The tool supports an integration with the Eclipse development environment.
Understand supports most of the widely used languages like Java, Ada, Python, C/C++ etc., it runs on Windows, Linux, Solaris and Mac operating systems. The tools extracts various basic and custom metrics like class count, function count executable statement count, class coupling, knots, weighted methods per class etc. It also has additional features like instant search which allows easy searching through millions of lines of code, graphing and mapping the software and generation of different types of reports.
The different types of visualisations made by Understand are:

- Hierarchy graphs: maps the class inheritance of the system.
- Control flow graphs: visualises all paths traversed by the code during execution
- Dependency graphs: is a direct graph which identifies all the dependencies between the modules of the system.
- Tree Maps: represents hierarchical data using nested figures

The report generated by Understand are in HTML/Plaintext formats. The different types of reports generated by Understand are:

- Cross Reference Reports: show information similar to that in the References section of the Info Browser
- Structure Reports: document the structure of the system
- Quality Report: examines the quality of the system
- Metric Reports: provides metric values computed on the system

Fig1 shows the metric summary of this application extracted using Understand.



| Metric | Value |
|---|---|
| Project | spring-petclinic-master.udb |
| Blank Lines | 436 |
| Classes | 42 |
| Code Lines | 1,835 |
| Comment Lines | 958 |
| Comment to Code Ratio | 0.52 |
| Declarative Statements | 661 |
| Executable Statements | 367 |
| Files | 47 |
| Functions | 149 |
| Lines | 3,221 |

**Fig1. Metrics Summary of Pet clinic application extracted from Understand**

Understand is easy to install and use with complete GUI interactions. The visualisation techniques used to evaluate our system were Hierarchy graph and Dependency graphs.

These graphs provided in depth analysis on the structuring of the system to help us get accustomed with the packages and classes of the system.

**JMetric 1.3.6**[19] was a research project started in April 1998 after the developers found very less extraction tool availability for the object-oriented languages. This tool had been developed specially for Java language based developed project. The tool is an outcome of an on-going research in the School of Information Technology at Swinburne University. The metrics that can be extracted using this tool are:

- Code line
- Amount of control structure
- Amount of control structure
- LCOM (Lack of Cohesion of Methods)
- Cyclomatic Complexity
- Number of classes, packages and methods

This tool can display the results in both tabular as well as graphical formats. It is an Eclipse IDE plugin and easy to install. All the feedback are integrated with the Eclipse UI. The visualisations used to analyse our system was graphical dependency analyser which shows dynamic hyperbolic package dependencies. Thus making it simple to identify the critical packages in the system.

Both Understand and JMetric 1.3.6 are robust and efficient tools but comparatively, Understand offers more flexibility and feedback.

## References

[1] Al-Jamimi,H.A.,Ahmed,M.,2012.Prediction of software maintainability using fuzzy logic, in: 2012 IEEE International 186 A.S. Nuñez-Varela et al. / The Journal of Systems and Software 128 (2017) 164–197 Conference on Computer Science and Automation Engineering. IEEE, pp. 702–705. doi:10.1109/ICSESS.2012.6269563

[2] Almugrin,S., Melton, A., 2015. Estimation of Responsibility Metrics to Determine Package Maintainability and Testability, in: 2015 Second International Conference on Trustworthy Systems and Their Applications. IEEE, pp. 100–109.doi:10.1109/TSA.2015.25

[3] Dubey, S.K., Rana, A., 2011. Assessment of maintainability metrics for object-oriented software system. ACM SIGSOFT Softw. Eng. Notes 36, pp. 1–7. doi:10.1145/2,020,976.2020983.

[4] Kumar, L., Naik, D.K., Rath, S.K., 2015. Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability, in: 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015). Elsevier Masson SAS, pp. 798–806. doi:10.1016/j.procs.2015.07.479.

[5] Yadav, A., Khan, R.A., 2011. Class cohesion complex- ity metric (C3M), in: 2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011). IEEE, pp. 363–366. doi:10.1109/ICCCT.2011.6075152

[6] Kuljit Kaur and Hardeep Singh. Towards a valid metric for class cohesion at design level. In 2011 Second International Conference on Emerging Applications of Information Technology, pages 351–354. IEEE, 2011.

[7] Dallal, J. Al, 2012. Theoretical Analysis for the Im- pact of Including Special Methods in Lack-of-Cohesion Computation, in: First World Conference on Innovation and Computer Sciences (INSODE 2011). pp. 167–171. doi:10.1016/j.protcy.2012.02.031.

[8] Bansal, M., Agrawal, C.P., 2014. Critical Analysis of Object Oriented Metrics in Software Development, in: 2014 Fourth International Conference on Advanced Com- puting  Communication Technologies. IEEE, pp. 197–201. doi:10.1109/ACCT.2014.106.

[9] De Silva, D.I., Weerawarna, N., Kuruppu, K., Ellepola, N., Kodagoda, N., 2013. Applicability of three cognitive complexity metrics, in: 2013 8th International Conference on Computer Science  Education. IEEE, pp. 573–578. doi:10.1109/ICCSE.2013.6553975.

[10] Nguyen, T.H.D., Adams, B., Hassan, A.E., 2010. Studying the impact of dependency network mea- sures on software quality, in: 2010 IEEE International Conference on Software Maintenance. IEEE, pp. 1–10. doi:10.1109/ICSM.2010.5609560.

[11] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in Proc. of the 30th inter. conf. on Soft. Engi. Leipzig, Germany: ACM, 2008, pp. 531–540.

[12] W.Li and S.Henry, Object Oriented Metrics that Predict Maintainability, Journal of Systems and Software, vol 23 no.2, 1993, pp.III-I22.

[13] E.H.Mamdani and S.Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, International Journal of Man-Machine Studies, 7(1):1-13,1975.

[14] Chidamber, Shyam R., and Chris F. Kemerer. "A metrics suite for object oriented design." Software Engineering, IEEE Transactions on 20.6 (1994): 476-493

[15] Fernando Brito e Abreu and Carapuca R. 1994. Object-Oriented Software Engineering: Measuring and Controlling the Development Process, Proceedings of the 4th International Conference on Software Quality, ASQC, McLean, VA, USA, 1994.

[16] Lorenz, Mark, and Jeff Kidd. Object-oriented software metrics: a practical guide. Prentice-Hall, Inc., 1994.

[17] Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Alant Apt Series. Prentice Hall, Upper Saddle River, NJ, USA (2002)

[18] Wei Li and Sallie Henry. 1993. Object-oriented metrics that predict maintainability. J. Syst. Softw. 23, 2 (November 1993), 111–122. DOI:https://doi.org/10.1016/0164-1212(93)90077-B

[19] Open Source System - Pet clinic application developed using the spring boot and thymeleaf java frameworks https://github.com/spring-projects/spring-petclinic

[20] JMetric 1.3.6 ( https://prezi.com/1oef4j-yrp4l/jmetric-tool/ )