Extending C & K metrics suite to Determine and Analyze the Maintainability of Java Projects

Thesissubmitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Software Engineering

Submitted By

Piyush Nagar

(Roll No. 801331018)

Under the supervision of

Mr. Vinod Kumar Bhalla

Assistant Professor

(CSED)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2015

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled," *Extending C & K metrics suite to Determine and Analyze the Maintainability of Java Projects*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Vinod Kumar Bhalla* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Piyush Nagar)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Mr. Vinod Kumar Bhalla

Assistant Professor, CSE Department

Countersigned by

(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

Thapar University

Patiala

(Dr. S. S. Bhatia)

Dean (Academic Affairs)

Thapar University

Patiala

Acknowledgement

First of all, I am extremely thankful to my respective guide ***Mr. Vinod k. Bhalla***. Assistant professor, CSED, Thapar University for his valuable guidance, advice, motivation, encouragement, moral support, sincere effort and positive attitude with which he solved my queries and provide delightful ambiance for learning, exploring and making this thesis possible. It has been a great pleaser and experience to work under his sanctuary.

I am also heartily thankful to ***Dr. Deepak Garg***, Associate Professor and Head, CSED for motivation. He sets high standards for his students and he encourages and guides them to meet those standards.

I would like to thank my family members and my friends who are dearest and precious to me for their love, encouragement, blessings and support in all respects. Most importantly, none of this would have been possible without the love and patience of my family. To my brother and friends for showing me right direction. They are constant source of love, concern, support and strength for me all these years.

Finally I would like to thank management of Thapar University for proving a great platform for learning, not just for academics but also for sports and many other creative things.

Date: July, 2015

Place: Thapar University, Patiala (Piyush Nagar)

ABSTRACT

Several studies have validated that various metrics are useful indicators of maintenance effort in the procedural paradigm. However, software metrics have rarely been studied in the object-oriented paradigm.The main aim of this research is to analyze the maintainability of java project on the basis of object oriented metrics. Methodology extends C & K metrics suite to predict maintainability of projects. Proposed research defines the two factors for each metrics first one is multiplying factor and second one is including factor. After that effect on maintainability is calculated using these factors. Finally methodology gives the maintainability percentage of java projects.

Table of Contents

List of Figures

List of Tables

Chapter 1

1. INTRODUCTION

In past, the functionality of software was the first demand of users, but by the time things are drastically changed. Big data came into the picture. Number of concurrent users is increasing day by day. Protection of software from fault has become the necessary thing. So the functional requirement is necessary but not sufficient for developing good software. Quality attribute of software is most crucial and demanded as well. In short quality attribute are the non-functional requirement of software system.Over its lifetime, the cost of a software product is determined more by how well it achieves its objectives for quality attributes such as performance, reliability, availability or maintainability than by its functionality. Mainly quality attribute are divided into four categories.

1.1 Categories of quality attributes

Quality Attributes are the nonfunctional requirement of project. Now a days non-functional is criteria for comparing products. So long list of quality is required for the software to be good. Mainly quality attribute are divided into three categories.

1.1.1 Design quality:

Design quality means whenever system undergoes to change, how easily software can be changed. It also includes the reusability of software. Design quality is helpful for the developer and tester team. User doesn't need to bother about design quality.

- *Maintainability*:Maintainability is one of the quality features of software product which defines how easily you can fix bugs and error, how easily you can extend further. Maintenance is the life time phase of software products. So it is most focused quality feature to work upon [1].
- *Reusability*: Reusability is about building a library of frequently used components, thus allowing new programs to be assembled quickly from existing components. Software reusability is the use of engineering knowledge or artifacts from existing software components to build a new system [2].

1.1.2 Runtime quality:

Runtime quality means are seen by user, because run time quality can be judge only when system will be in running condition.

- *Availability*:Availability of a system is a measure of its ability to provide a failure-free operation. For many practical situations, reliability of a system is represented as the failure rate [3].
- *Performance*:Performance of a system is measure of its ability to return result in less time for a user query. Performance depends on algorithm design as well as hardware platform on which software runs [4].
- *Reliability:* Reliability is the probability of success or the probability of a failure-free operation in a cloud in given time. Cloud gets many requests concurrently and will also give the similar results for some requests in minimum time possible with full success [5].
- *Scalability:* Scalability is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. Scalability also affects the performance of system, because if during heavy load system is enable to operate normally then definitely its performance is degraded [6].
- *Security:* Software security is to engineer software in such a way that the required application functions uninterrupted and is able to nicely handle the security threats during malicious attacks. Security ensures that application works in a desired manner and to provide defense against security threats [7].

1.1.3 System quality:

System Qualities gives idea about testing and easiness of uses.

- *Testability*:The ISO defines testability as ―attributes of software that bear on the effort needed to validate the software product [8].
- *Usability*:Usability of software is measure of easiness for user to operate software [9].

1.2 Importance of maintainability

By the way the entire quality feature is equally important, but the better maintainability is more required feature for developer and tester team. Maintenance phase is the life time phase of SDLC process. Maintenance phasemay be costly and effort intensive if software is not well design before [10].So it's very necessary to develop good maintainable software.

1.2.1 Types of maintainability

On the basis of purpose maintainability can be divided into various part.

- *Corrective Maintenance*: Corrective maintenance is the process of fixing the bugs after deployment of projects.
- *Perfective Maintenance*:Perfective maintenance is the process of enhancement in projects.
- *Adaptive Maintenance*:Adaptive maintenance is the process of changing hardware and software platform for projects.
- *Preventive Maintenance*: Preventive maintenanceis process of making good maintainable software for future purpose.

1.3 Traditional metrics

It has already been covered what maintainability is? But the problem is that how to make good maintainable software? How to check maintainability of already developed software? What to measure for controlling the maintainability? How to measure? To control features, measurement is the first step.Software complexity is an area of software engineering concerned with the measurement of factors that affect the cost of developing and maintaining software [24].Every object in a real world is associated with some properties and features. These properties and features can be measure quantitatively or qualitatively. On the basis of these measurements quality of object can be judged. As software is also an object than definitely it will have its own properties, which can be measured quantitatively or qualitatively. Metrics are the measure of features of software product [11].

1.3.1 Types of traditional metrics

Traditional metrics are the measurement of features which are not related to any methodology. Most of the traditional metrics are applicable to traditional system or function oriented system. But few of traditional metrics are also applicable to object oriented system. Our main concentration will be on object oriented metrics, but for just introduction there are some brief about traditional metrics.

Traditional metrics also divided into following categories.

- *Project metrics*:Project metrics measure the feature of deliverables. LOC and Function Point metrics comes under this category.
- *Process metrics*:Process metrics measure the process quality. It generally related with defect and effort to recover from defects.
- *Product metrics*:Product metric measure the cost of project and time to deploy the project. COCOMO modal is used for such metrics.
- *Organizational metrics*:Organizational metrics covers measurement related to employee satisfaction, communication and organizational growth factors of the project.

1.4 Object oriented metrics

Methodology used to develop the software decides the maintainability of software, for example it's easy to maintain software develop in object oriented methodology than procedural methodology. Now a day most of the software vender is using object oriented methodology to develop the software. Object oriented metrics are the measure of properties of software that are directly or indirectly related to object oriented methodology. Metrics which are not related with object oriented feature do not comes under category of object oriented metrics even though they are measured for object oriented project. There are so many features of object oriented methodology which affects the maintainability of software. Proper use of these features is necessary to develop good maintainable software. But how can you control the proper use of these features. Features can't be controlled without measuring them. Means quantitative measurements are required to judge the features. The result of such quantitative measurement of object oriented feature is known as object oriented metrics. So the best

understanding of object oriented feature is necessary to drive the conclusion, because each feature has its own significance. This paper meant to say that each quality attribute is affected by design of system. In object oriented system use of object oriented features decide the design quality of system. Design should fulfill the all required quality features. Good designed software reduces the effort required to maintain the software [12].Object-Oriented Analysis and Design of software provide many benefits such as reusability, decomposition of problem into easily understood object and the aiding of future modifications. But the OOAD software development life cycle is not easier than the typical procedural approach. Therefore, it is necessary to provide dependable guidelines that one may follow to help ensure good OO programming practices and write reliable code [22].

Features of object oriented methodology:

a) *Class*: Class is collection of similar type of objects.
b) *Object*: Object is instance of class with specific value for each variable.
c) *Methods*: Methods keep the procedure that can be applied on object of that class or derived class.
d) *Instance variable*: Instance variable declares inside the class, but non-sharable among the objects.
e) *Static variable*: Static variable also declares inside the class, but they are sharable among the object.
f) *Local variable*: Local variable are declares in the method and they have no scope outside the method.
g) *Encapsulation*: Process of keeping methods and variable in the class with proper use of "access specifiers".
h) *Abstraction*: Abstraction is guideline for derived classes. Abstraction declared the method which implements in derived class.
i) *Inheritance*: Inheritance concept gives two terms base class and derived class. In inheritance derived class can used the properties and method of base class. Inheritance is used to increase reusability in programs.

j) *Polymorphisms*: Polymorphism is about implementing the same name but different functionality. Polymorphism is practically implemented with help of method overloading and method overriding.

k) *Method Overloading*: Method overloading allows us to use methods with same in a single class, but with different parameters.

l) *Method overriding*: Method overriding allows us to write method with same name and same parameter, but it derived class of that class.

m) *Cohesion:* Cohesion is the measure of independency between the modules. High cohesion is necessary for the good design.

n) *Coupling*: Coupling is the measure of interdependency between the modules. Low coupling is necessary for the good design.

1.4.1 Type of object oriented metrics:

Object oriented metrics can be calculated for whole system or part of system. Part of system can be method, class and package. Collection of methods makes one class, collection of classes make one package and collection of all packages make software system.

1.4.1.1 Method level:

Method level metrics calculate for methods only. Method level metrics give idea about the method.

a) *Number of lines of code*: Line of code is oldest metrics. It is useful for measuring effort took by method. Higher value of line of code indicate method is more complex

b) *Cyclomatic Complexity*: McCabescyclomatic Complexity measures the number of independent path from entry point to exit point of method for the method. Cyclomatic complexity is completely not depends on line of code. On the contrary it depends on control statement. If-else statement, loops (for, while, do-while) and goto statements well known control statement used in programming language.

c) *Halstead length*: The Halstead length of method is count of operand and operator declared in method.

d) *Halstead vocabulary*: The Halstead vocabulary of method is unique count of operand and operator declared in method.
e) *Halstead difficulty*: Halstead Difficulty gives a formula to analyze the complexity base Halstead vocabulary.
f) *Halstead bugs*: Halstead Bugs attempts to estimate how much your method is prone to bugs. Its value should not be high.
g) *Number of Arguments*: Large value this metrics signifies that method is has lot of work to do. So designer should check all argument are useful or not.
h) *Number of Comments*: The number of Comments associated with the method. There is disadvantage of number of comment metrics. It does not tell about usefulness of comment. It only tells about size of comment.

1.4.1.2 Class level:

Line of code and all Halstead metrics can also be defined at class level. Class level metrics don't emphasize only on metrics which tells about single class. Class level metrics also emphasize on the measurement, which tells about interaction between the classes. All the metrics discussed above can also be used at class level.

a) *Un Weighted class Size*: The Un-weighted Class Size (UWCS) of the class is defined as the number of methods plus number of variables of the class.
b) *External method calls*: The external methods called by the class and by methods in the class. Higher value of this metrics increase coupling among the classes and makes the system complex.
c) *Local method calls*: The local methods called by the class and by methods in the class that are defined in the hierarchy of the class.
d) *Instance Variables*: The instance variables defined by the class
e) *Modifiers*: The modifiers metrics shows the protection level of that class. Public, private, protected are the common modifiers that uses in object oriented language.
f) *Interfaces Implemented*: The interfaces implemented by the class defines the level of abstraction used.
g) *Packages imported*: The packages imported metrics defines use of API by the class. Higher value of this metrics reduces the effort needed to develop the class.

h) *Fan In*:Fan In is the number of other classes that reference a class. Fan Out also known as Afferent Coupling.

i) *Fan Out*: The Fan Out for class is defined as the dependency of that class to other class. Fan Out also known as Efferent Coupling.

j) *Reuse Ratio*: The Reuse Ratio defines the reusability of class and mathematically it is calculated by ratio of number of ancestor in inheritance hierarchy to the total number of class in inheritance hierarchy.

k) *Specialization Ratio*:Mathematically it is calculated by ratio of number of descendent in inheritance hierarchy to the total number of class in inheritance hierarchy.

1.4.1.3 Package level:

Line of code, cyclomatic complexity, fan in, fan out and all Halstead metrics can also be defined at package level.

a) *Instability*: Instability is measure of how susceptible the package is to change. Zero value for this metrics indicates package is easier to change and it's preferable. One value for this metrics indicates package is difficult to maintain.

b) *Abstractness*: The value of the Abstractness can be between 0 and 1. It is calculated by ratio of total number of abstract classes and interfaces to total number of classes in package.

c) *Distance*: This measures the balance of a particular package between abstraction and instability.

1.4.1.4 System level:

As methods make class, classes make the package and package makes the system. So system level metrics totally depends on the method level, class level and package level metrics. All the metrics that are defined in class level and package level can be extended to system level.

Chapter 2

2 LITERATURESURVEY

Literature survey covers object oriented metrics proposed by different researcher. Main goal of literature survey is the understanding the relation of different suites with quality features. So that it can be decided that, which metrics suite is suitable to use for methodology. For literature survey various research papers on object oriented metrics have been studied in detail and gathered information about the research work of different authors by examining following questions

A) What work already has been done in field of object oriented metrics?

B) What work remains to be done?

2.1 Metrics for object oriented software engineering

This suite is well known as C & K metrics suite.

2.1.1 Weighted method per class

Method per class measures average distribution of method in each class. Quantitatively it is measured by ratio of total number of method to the total number of class [13].

Let,

T: total no of classes

NM (i): number of method in class i

CF (i): complexity factor of class i

WMC: weighted method per class

$$\mathrm{WMC} = \frac{1}{\mathrm{T}} \sum_{\mathrm{k}=1}^{\mathrm{T}} [\mathrm{NM(k)} * \mathrm{CF(k)}] \qquad (2.1)$$

2.1.2 Level of inheritance

Level of inheritance of any class is the number of ancestor (number of its base class) but this point will fail when projects use the programming languages that support multiple inheritance like C++. Proper definition of level of inheritance is height of particular class in inheritance hierarchy. If any class doesn't have any base class than level of inheritance is zero for that class.[13].

LOI_i=level of inheritance of ith class [14].

Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree [23].

2.1.3 Coupling between the object

Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class [13].

CBO_i= number of classes to which class i is coupled [14].

2.1.4 Response for the class

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class [13].

N_i: number of method in class i.

R_i: number of remote methods directly accessed by local method of class i.

$$RC_i=N_i \text{ `}+ R_i \text{[14]} \qquad (2.2)$$

2.1.5 Lack of cohesion of method

It's a class level metrics let c1 is a class having n number of method. a^1, a^2, a^3…a^n and v^1, v^2, v^3…$v^n$is set of instance variable correspond to each method. Instance variable set is the collection of instance variable that are used by method. Lack of cohesion in method is difference between number of function pair for which intersection of instance variable set is empty and whose intersection of instance variable set is not empty[5]. Lack of cohesion in method for a class measures the dissimilarity of methods in a class via instanced variables[13].

$X=\{ (v_i, v_j)| v_i \cap v_j= \emptyset\}$

$Y=\{ (v_i, v_j)| v_i \cap v_j != \emptyset\}$

$LOCM_i=|x|-|y|$ if $|x|>=|y|$ otherwise zero[15].

Value of metrics equal to zero indicates that that the class is more cohesive. Value of metrics greater than zero Indicate that normalization or breaking of class into number of different class is necessary. High value of lack of cohesion metrics means class is faulty.

2.1.6 No of children

Number of immediate child classes of particular class. Higher value means higher use of reusability but also leads to more complex structure. So like level of inheritance this metrics also require balance in value.

NOC_i=number of immediate child classes of class I[15].

2.2 Metrics for object oriented design (MOOD)

Mood metrics is proposed by B.F. Abreu. In research work researcher has generated so many metrics to explain quality of the system. MOOD divides metrics in different categories based on object oriented feature, for example inheritance level, class level, method level, polymorphism level [16].

Let

T =total number of classes in source file
CL(i)=ith class of source file
M(i,j)=ith method of class j.
V(i,j)=ith variable of class j.
NM(i)=no of method in class i
NV(i)=no of variables in class i.

a) Accessibility of method:

is_accessible_m(M(i,j),CL(k)): *is_accesible_m* is the Boolean function which return 1 if i!=j and CL(k) could access M(i,j).otherwise it will return 0.Let accessibility of method is denoted by AS_M(i,j) [17].

$$AS\_M\ (M(i,j)) = \frac{\sum_{k=0}^{T} is\_accessible\ \_m\,(M\,(i,j),CL\,(K))}{T-1} \qquad (2.3)$$

Constraints:

- $\sum_{k=0}^{T} \text{is\_accessible\_m}(M(i,j), CL(k)) \leq T\text{-}1$
- $0 \leq AS\_M(M(i,j)) \leq 1$

b) Accessibility of variable

is_ accessible_v(V(i,j),CL(k)): *is_accessible_v* is the Boolean function which return 1 if i!=j and CL(j) could access V(i,j).otherwise it will return 0. Let accessibility of variableisdenoted byAS_V(i,j) [17].

$$AS\_V(V(i,j)) = \frac{\sum_{k=0}^{T} \text{is\_accessible\_v}(V(i,j), CL(k))}{T-1} \qquad (2.4)$$

Constraints:

- $\sum_{k=0}^{T} \text{is\_accessible\_v}(V(i,j), CL(k)) \leq T\text{-}1$
- $0 \leq AS\_V(V(i,j)) \leq 1$

The return value by *is_accessible_m* function can't have value greater then T-1 because at most all class can access that method except itself. It concludes that the accessibility of method can't be greater than 1. It can also be concluded that *is_accessible_m* function can have minimum value zero. The accessibility of method can't be less than 0. Same can be concluded about variable. Now it has been discussed what is *is _accesible* function and what is accessibility? What is the significance and importance of this function? These two metrics are not used directly, but for finding so many important metrics researcher used this metrics [16].

2.2.1 Encapsulation

Encapsulation allows us to hide variable and method of a class from outer world. Hide factor of variable or method is inversely proportional to accessibility of variable and method respectively. So magnitude of hide factor can be measure by 1 minus 'accessibility'.Two type of hide factor can be calculated, measure method hide factor and variable hide factor [17].

a) Method hide factor (MH)

$$MH=\frac{\sum_{k=1}^{T}\sum_{j=1}^{NM(k)}(1-AS\_M(M(j,k)))}{\sum_{k=1}^{T}NM(k)} \quad (2.5)$$

b) Variable hide factor

$$VH = \frac{\sum_{k=1}^{T}\sum_{j=1}^{NV(k)}(1-AS\_V(A(j,k)))}{\sum_{k=1}^{T}NV(k)} \quad (2.6)$$

2.2.2 Inheritance

Amount of inheritance used in source code show level of reuse. Now when class A inherits the class B then it can inherit function as well as variable [17].

a) Function inheritance factor(MIF)

NMI(i)=number of inherited function in class i(OVERINDING function is not Included in this), because it will comes in a set of method of class itself. Derived class can access method defined in itself and method of its base class [16].

NMA(i)=number of method accessed by i.

NMA(i)=NMI(i)+NM(i)

$$MIF = \frac{\sum_{k=1}^{T}NMI(k)}{\sum_{k=1}^{T}NM(k)} \quad (2.7)$$

b) Variable inheritance factor(VIF)

Derived class can access variable defined in itself and variable of its base class.

NVI(i)=number of inherited variable in class i

NVA(i)=total number of variable accessed in association with class CL(i).

NVA(i)=NVI(i)+NV(i)

$$VIF = \frac{\sum_{k=1}^{T}NVI(k)}{\sum_{k=1}^{T}NV(k)} \quad (2.8)$$

2.2.3 Coupling

Coupling show interdependency between classes .Inheritance also increase coupling but inheritance is not included in coupling metrics, because inheritance support reusability and reusability is most required quality feature in today era [16].

Coupling between the classes=CBC

is_related(CT(i),CT(j))=its will return 1 if class CT(i) is related with class CT(j).

$$CBC = \frac{\sum_{j=1}^{T}\sum_{i=0}^{T} is\_related\ (CT(i),CT(j))}{T*T-T} \qquad (2.9)$$

2.2.4 Polymorphism

Polymorphism may be compile time or may be run time. Run time polymorphism is more important than compile time, because now a day Runtime polymorphism is the first demandin generic software. Software should support differ binding time and it can be implemented by method overriding [16].

NMO(i)=number of method overridden in class i.

NMN(i)=number of new method in class i.

NDC(i)=number of derived class for i.

Runtime polymorphism factor (RPF):

$$RPF = \frac{\sum_{k=1}^{T} NMO\ (k)}{\sum_{k=1}^{T}[NMN\ (k)*NDC\ (k)]} \qquad (2.10)$$

2.3 MORRIS metrics

2.3.1 Method per class

Average method per class is the ratio of total no of method to the total no of class. This metrics is used to check that which class is more deviated from central tendency with respect to average number of method [25].

Let

NC=total no of classes

NM (i) =number of method in class i.

AMC=average method per class

$$AMC = \sum_{k=1}^{T} NM(k) \qquad (2.11)$$

2.3.2 Inheritance dependency

As moose metrics suit already discussed about width of inheritance. Then what is difference between Inheritance dependency metric and width of inheritance metric. Both are different in the sense that width of inheritance show level of hierarchical inheritance

but what about multilevel inheritance. So to measure extend of multilevel inheritance Morris introduce a new category of metrics named depth of inheritance [25].

DOI=Depth of inheritance

DS(i)=DS(I is a set which contain length of all down path from class i to every leaf.)

$$DOI=\max (DS(i)) \quad (2.12)$$

Significance:

a) It shows more reusability than width of inheritance tree.

b) It has more sharing than width of inheritance tree.

2.3.3 Degree of coupling between the objects

It is related to class and object UML diagram [25].

NL= number of connected lines among the packages in UML diagram

TP=total number of packages

DC=degree of coupling b/w the package

$$DC=NL / TP \quad (2.13)$$

2.3.4 Cohesiveness of class

COB=cohesiveness of class.

FI(i)=no of incident line for class i in class diagram.

$$CBO = \frac{\sum_{k=1}^{T} FI(k)}{T} \quad (2.14)$$

2.3.5 Object library effectiveness

EOL=effectiveness of library

RO=number of object used from library.

TO=Total object in library.

$$EOL = RO/TO \quad (2.15)$$

2.3.6 Factoring effectiveness

Factoring effectiveness is the measure of run time polymorphism. The run time polymorphism in object oriented language is implemented by method overriding and method overriding helps in differ binding time [25].

FE=Factoring effectiveness
NMEO=number of methods excluding overriding.
NMIO=number of method include overriding (total)

$$FE = NMEO/NMIO \quad (2.16)$$

2.3.7 Average method complexity

It is a system level metrics because it value represent the complexity of whole system not the complexity of sub system [25].
CC(i)= cyclomatic complexity of ith method.
M=total no of methods

$$AMC = \frac{\sum_{k=1}^{M} CC(k)}{M} \quad (2.17)$$

2.3.8 Application granularity

T=total no of classes
TFP=Total function point
AG=application granularity

$$AG = \frac{T}{TFP} \quad (2.18)$$

2.4 LORENZE and KIDD metrics

Lorenze and Kidd proposed a metrics suit for object oriented design which is commonly known as LK suit. This suite proposed four metrics.

2.4.1 Length of class

Class in object oriented system is consist of methods and variables. Then its size will be definitely calculated from number of functions and number of variables defined in the class [19].

a) Length of class considering the variable(LCV)

LCV=number of variable embedded in it.

b) Length of class considering the method(LCM)

LCM=number of variable embedded in it.

2.4.2 Number of function used for run time polymorphism in derived class

Although runtime polymorphism is good for customizable product, but excess of it can make base class difficult to modify and test. So balancing of such metrics must be necessary. Run time polymorphism implements late binding. Late binding helps us in modifiability. Component can be replace at run time. [19].

NOMD=count of function used for run time polymorphism in derived class

2.4.3 Number of function specialized by derived class

There may be two type of function in a class, overridden and local function. In generalization and specialization hierarchy spatial properties of any sub class are those which are not in its base class. So specialized methods can be find by subtracting number of run time support method in derived and total method in derived class [19].

NSMD=number of specialized method in derived class

NM(d)=total number of method in derived class

NOMD=count of function used for run time polymorphism in derived class

$$NSMD = NM(d) - NOMD \text{ [20]} \qquad (2.19)$$

2.4.4 Specialization factor

NOMD=count of function used for run time polymorphism in derived class

DCI=depth of class in inheritance hierarchy

NM(i)=number of method in class i

SF(i)=specialization factor

$$SF(i) = (NOMD * DCI)/NM(i) \text{ [20]} \qquad (2.20)$$

2.5 EMOOSE metrics

2.5.1 Message pass coupling

It's common practice to use function of other class by function of other class to fulfill their task. Suppose there is a class 'A' which have three function and there is a class B it has 2 function and every function of class 'A' access every the function of class B, then total function call by A is 6 [21].

2.5.2 Data abstraction coupling

This metric measures the number of instantiations of other classes within the given class. This type of coupling is not caused by inheritance or the object orientedparadigm. For example there is a class A and another class B .Class B has five local variables out of which 2 are accessed in class A then data abstraction coupling will be 2 [21].

2.5.3 Local function

There may be some method which doesn't call any function or variable of other class. Such function are said to be local function. Number of local function shows how much your class is isolated from outer world [21].

2.6 Q-metrics for object oriented design

Q-metrics for object oriented design is explained by Bansiya's. Q-metrics divided into two main categories. Somemetrics represent whole system on other hand some metrics represent only specific components.

2.6.1 System level Q-metrics

Below is the table of metrics which are unique for whole system. Means by calculating these Quality of whole system can be judged. Here uniqueness means these metrics are not for a particular module [18].

Acronym	Description
DSC	Design Size in metrics
NOH	Number of Hierarchies
NIC	Number of independent classes
NSI	Number of single inheritance
NMI	Number of multiple inheritance
NNC	Number of internal classes
NAC	Number of abstract classes
NLC	Number of leaf classes
ADI	Average depth of inheritance
AWI	Average width of classes
ANA	Average number of ancestors

Table1. System level Q-metrics

2.6.2 Component level Q-metrics

These are the metrics which are specific to particular component of software system. Knowing these metrics we can judge quality of particular component of system [18].

Acronym	Description
MFM	Measure of functional modularity
MFA	Measure of functional abstraction
MAA	Measure of attribute abstraction
MAT	Measure of abstraction
MOA	Measure of aggregation
MOS	Measure of association
MRM	Modeled relationship measure
DAM	Data access metrics
OAM	Operation access metrics
MAM	Member access metrics

Table2. Component level Q-metrics

2.7 CHEN metrics

This metrics suit is closely related to metrics proposed by Morris. Chen metrics concentrate more on cohesion and coupling [18].

Acronym	Description
CCM	Class coupling metrics
OXM	Operating complexity metrics
OACM	Operating argument complexity metrics
ACM	Operating coupling metrics
OCM	Cohesion metrics
CM	Class hierarchy of method

Table3. Chen metrics suite

Chapter 3

3 PROBLEM STATEMENT

Growth of software in terms of quality features in last decade has changed the way of design of software. Object oriented methodology is become dominant over procedural methodology and user frequently demands for new functionality which force vender to develop new version with greater functionality. Use of object oriented design force the quality engineers to develop object oriented metrics.The main problem is to maintain the maintainability of software using object oriented metrics. How to check software developed by vender fulfills the maintainability criteria or not.

Proposed methodology for maintainability implements the following functionalities:

1. Methodology should have threshold value for each metrics to check with the actual value.
2. It will also provide a way to convert class level C & K metrics to system level metrics.
3. It will beable to perform an operation which can find the metrics, which are beyond the threshold value.
4. Research carried out proposes a formula which calculatesthe maintainability in percentage.

Chapter 4

4 IMPLEMENTATION

There are lotsof metrics available for object oriented system, but the problem is that among the all object oriented metrics proposed by different researcher, which metrics should be taken to analyze the maintainability. It is not possible to take all the metrics in methodology [6]. It is also senseless to take all the metrics, because metrics which are not related to maintainability are useless for our methodology. Cohesion and Coupling is the major feature which affects the maintainability directly. So for good maintainable software cohesion should be high as much as possible and coupling should be low as much as possible. So metrics which are closely related to these features should be taken. After studying different metrics suite it is found that C & K metrics suite is very closely related to maintainability. So for analyzing the maintainability C & K metrics suite is better than other suite. Proposed Methodology analyze the maintainability of whole project not for individual class or module, but most of the C & K metrics are class level metrics, so they can't be used directly. Methodology convert the class level metrics to system level metrics.Mapping of all class level metrics to project level is key feature of proposed methodology, because the methodology analyze maintainability of overall project not for individual classes. Before applying algorithm all the C & K metrics must be known for each class of the system and then take average of each metrics value. The average value of each metrics will be treated as system level metrics. The methodology uses average value calculated for each metrics. Methodology calculates the average value of all the metrics except weighted method per class metrics, because it is already a system level metric.

T=total number of classes.

Average level of inheritance= $\frac{\sum_{i=0}^{T} LOIi}{T}$

Average lack of cohesion of method=$\frac{\sum_{i=0}^{T} LOCMi}{T}$

Average number of children=$\frac{NOCi}{T}$

Average coupling b/w the object=$\frac{\sum_{i=0}^{T} CBOi}{T}$

Average response for the class=$\frac{\sum_{i=0}^{T} RCi}{T}$

4.1 Parameter proposed

This methodology defined following parameter to analyze maintainability.

4.1.1 Effect variable(E)

Effect variable defines the effect of each metrics on system. The methodology categories effects into three categories Low, Medium and High. These categories are given on the basis of relation of metrics with maintainability.

4.1.2 Multiplying factor (MF)

On the basis of effect variable each metrics have given a particular interval value.These values are called as interval value because it can be ranked and also provide a specific value for each category of effect variable. Value for Low effect is 0.25. Value for Medium effect is 0.50. Value for High effect is 1.0. So there is fix interval between the multiplying factors of each metrics. Interval between Low and Medium effect is 0.25 and interval between Medium and High variable is 0.50.These values are given on the basis of relation of metrics with maintainability. Metrics which highly affect the maintainability have 1 as multiplying factor. Metrics which moderately depends on maintainability have 0.50 as multiplying factor. Metrics which are loosely related to maintainability have 0.25 as multiplying.

4.1.3 Including factor (INF)

Including factor can have value either 0 or 1. Value to including factor will be given on the basis of calculated value of metrics and threshold value defined for metrics by different researcher. Means if value of metric is tolerable, then including factor will be zero and if value of metrics is not tolerable, then including factor will be one. Including factor decides that particular metric should be considered in calculation or not.

4.1.4 Effect on maintainability (EOM)

Effect on maintainability for each metrics is calculated by product of multiplying factor and including factor.

4.2 Proposed value of multiplying factor and effect variable

Metrics	M F	E
Weighted method per class	0.50	Medium
Level of Inheritance	0.25	Low
Lack of Cohesion of method	1.0	High
No of children	0.25	Low
Coupling b/w the object	1.0	High
Response for the class	0.50	Medium

Table4. Value of Multiplying Factor and Effect Variable

4.3 CKJM tool

CKJM tool is an open source tool for calculating C & K metrics suite. CKJM tool is very efficient tool available for C & K metrics suite.

4.3.1 How to use CKJM tool

CKJM is open source software. So it's available with source code and JAR file. First compile the java file and generate its .class file. Type the following command in command prompt to calculate C & K metrics for java file and press enter.

Command: Java –jar "path_of_CKJM.jar" "path of class file.class"

It takes too much time in processing file one by one. CKJM tool also provided facility to calculate result of C & K metrics for more than one Java file simultaneously.To calculate results for more than one class file a small change in command is needed. In command just replace the class name with '*'.

Command: Java –jar "path_of_CKJM.jar" "path of class file.class"

4.3.2 Result Pattern of Metrics

After executing the command result will be shown on command prompt. Result shows the complete path of file followed by list of all C & K metrics for that file. Result shows the metrics in order WMC, DIT, NOC, CBO, RFC, and LCOM. Result also shows list of all methods, its prototype and number of parameter.

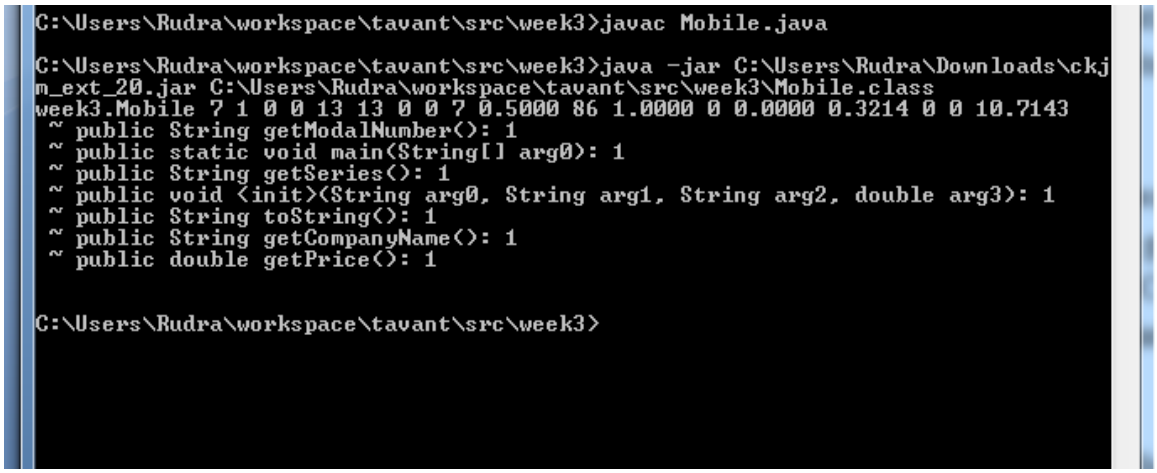


Figure 1 Showing C & K metrics as an output for single java class

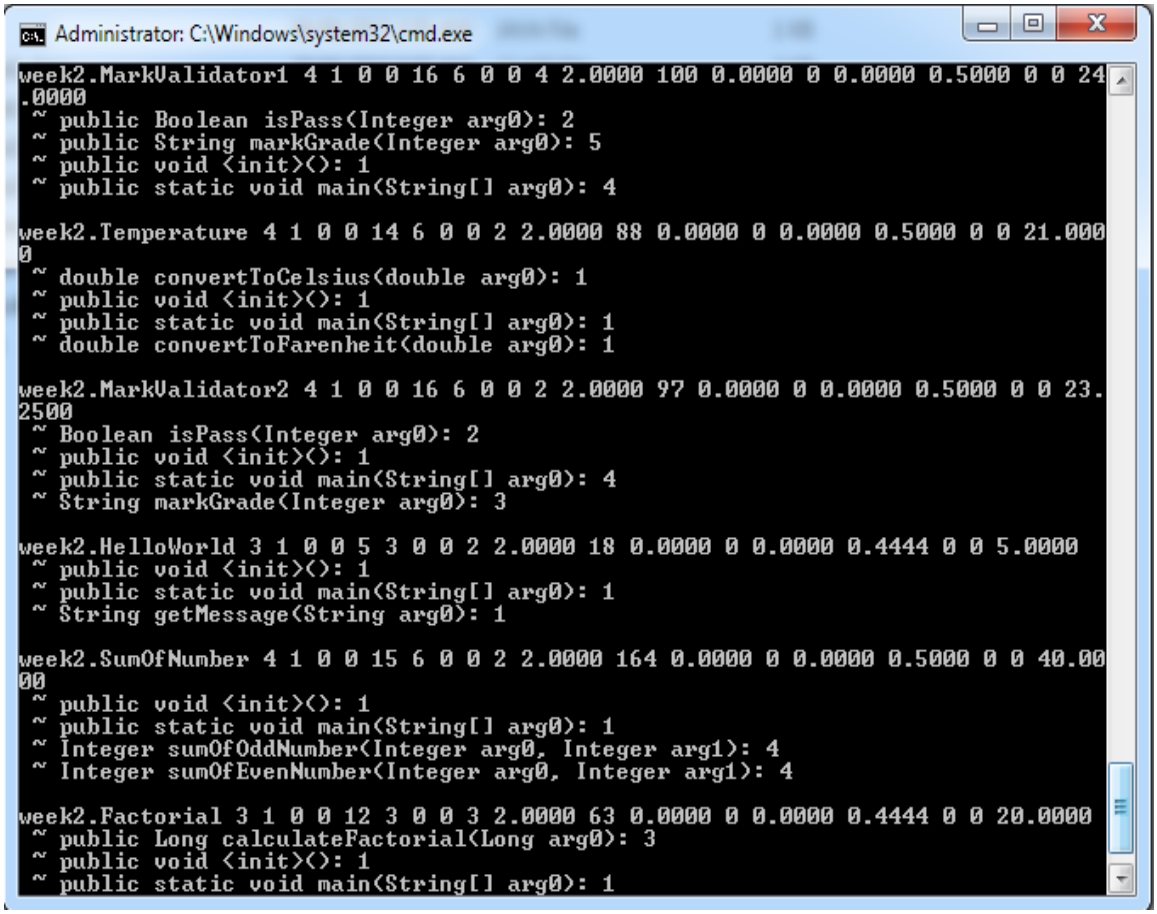


Figure 2. Showing C & K metrics as an output for multiple java class simultaneously

4.4 Algorithm proposed

This research already has discussed the different variables and metrics that are going to be used in algorithm.

Step1: Metrics for each classwill be calculate using CKJM tool and then calculate the average value for all the metrics.

Step2: Compares each calculated metrics with threshold value. If metrics value is less than threshold value then set including factor for that metrics to zero. If metrics value is greater than threshold value then set including factor for that metrics to one.

Step3: After assigning value to including factor, methodology calculates the effect on maintainability for each metrics. It is product of multiplying including factor and multiplying factor particular metrics.

Step4: Sum up the effect on maintainability for all the metrics.

Step5: Normalize the effect on maintainability between the 0 and 1.

Step6: Calculate maintainability.

Effect on maintainability and maintainability are the two different terms and both are inversely proportional to each other. Effect on maintainability just tells about deprecation of maintainability in percentage. So remaining part is itself maintainability. After calculating the effect on maintainability, maintainability can be calculated by formula (1-*effect on maintainability*). It has been discussed that how including factor and multiplying factor effect the maintainability of projects.

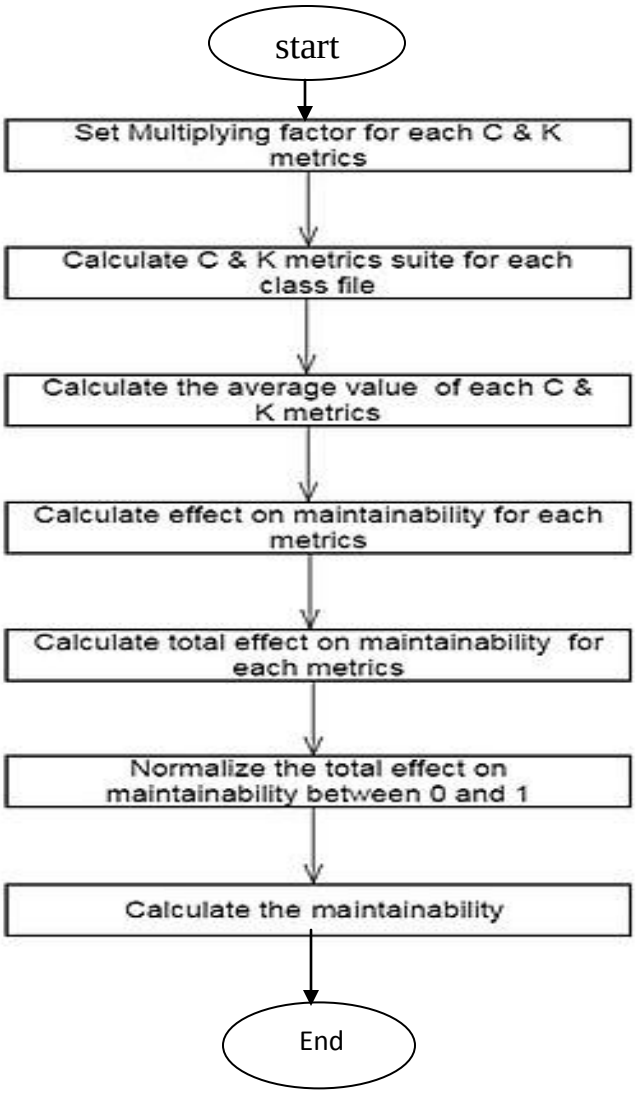


Figure 3.Flow Graph of algorithm

Notation	Description
MF_i	Multiplying Factor of i^{th}metrics.
IF_i	Including Factor of i^{th}metrics.
CV_i	Calculated value of i^{th} metrics.
TV_i	Threshold value of i^{th}metrics.
EOM_i	Effect on maintainability of i^{th}metrics.
TEOM	Total effect on maintainability.
TM	Total maintainability

Table5. Different acronym used in algorithm

Algorithm 1: Maintainability Analyzer

Input: Java files

Output: Maintainability of java project

1. setMF_i for each metrics i.
2. for each metrics I
3. calculate CV_i ;
4. end for
5. for each metrics i
6. if (CV_i<TV_i) then
7. INF_i=0;
8. Else
9. INF_i=1;
10. end for
11. for each metrics i
12. EOM_i=INF_i*MF_i;
13. end for
14. $TEOM=\frac{\sum_{i=0}^{5} EOMi}{\sum_{i=0}^{5} MFi}$
15. TM=1-TEOM;
16. stop

Chapter 5

5 RESULTS

Methodology used open source CKJM tool to calculate the value of eachmetrics. This methodology has been applied on three different projects Online Examination System, Multimedia Steganography and ERP System. All three projects are almost equally hard to develop. So it will be good to compare them. First three columns show the value of C & K metrics for different project. Third column shows threshold value corresponds to each metric. Then including factor is calculated for each project by comparing the threshold value and actual value of metrics. Then Effect on Maintainability is calculated for each project with help of including factor and multiplying factor. Effect of maintainability will be between 0 and 1. Methodology normalize effect on maintainability between 0 and 1. So that it can be also convert it into percentage multiplying it by 100. Now at this point it has been known that how much maintainability is affected. So maintainability is calculated by simple probability rule (1-effect on maintainability). After applying methodology to these projects, the maintainability of projects are compared on the basis of result. The methodology result not only shows the maintainability but also shows the importance of multiplying factor in methodology. It has already been discussed that all the metric does not affects maintainability equally. So result will also explain how multiplying factor affects the maintainability of project. It can be observed from result thatOnline Examination System and Multimedia Steganographyboth have 2 metrics which is out of threshold but maintainability of ERP System is higher than Online Examination System. This shows that multiplying factor place an important role in maintainability.

Let,

P1: Online Examination System

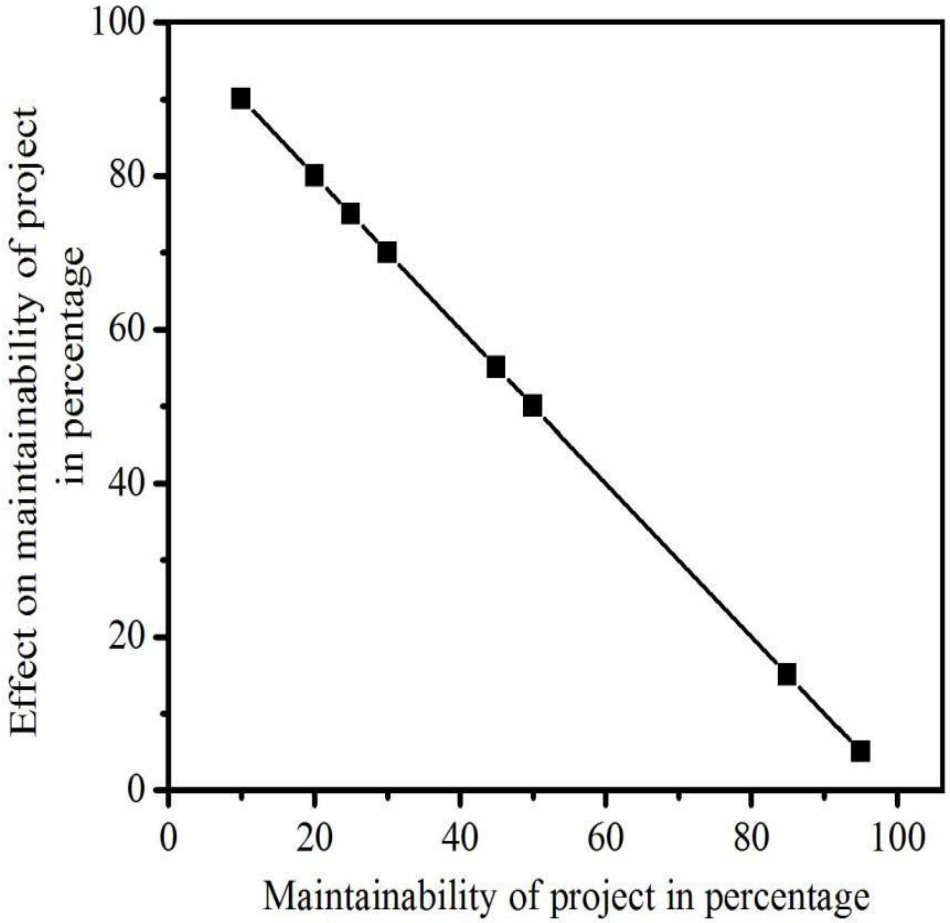
P2: Multimedia Steganography

P3: ERP System

INF1: Including Factor of Online Examination System

INF2: Including Factor of Multimedia Steganography

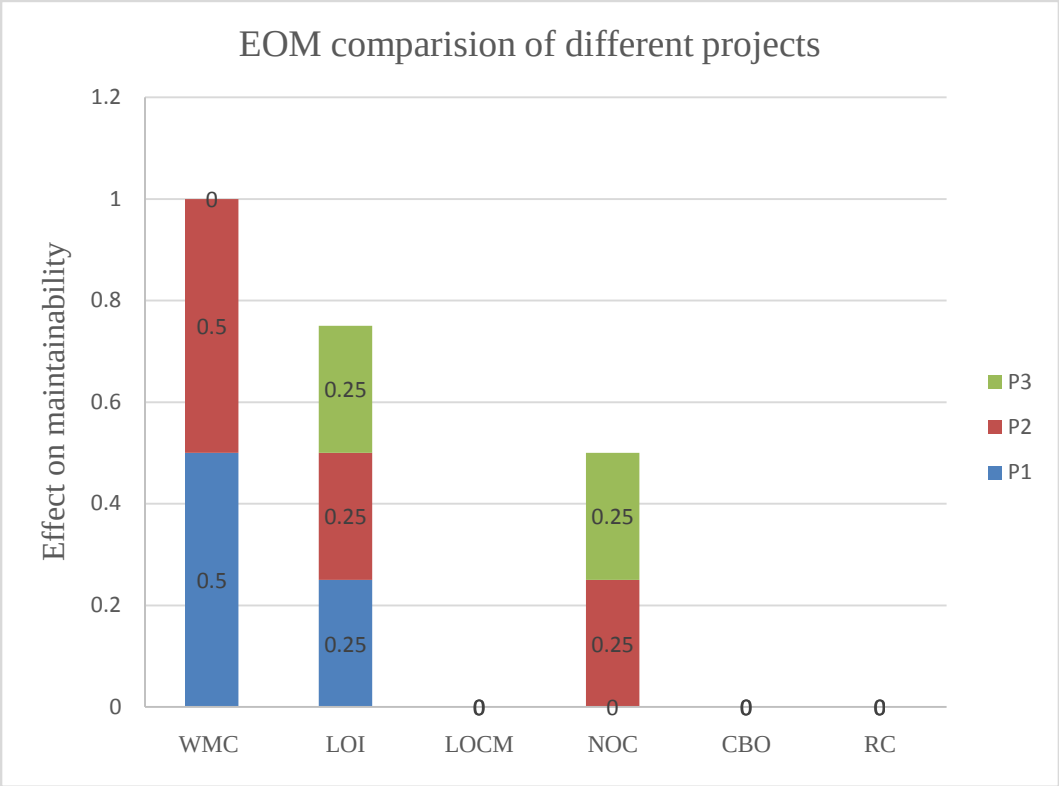
INF3: Including Factor of ERP System



"***Note***. This graph shows the relationship between maintainability and effect on maintainability.

Metrics	**P1**	**P2**	**P3**	**INF1**	**INF2**	**INF3**	**EOM1**	**EOM2**	**EOM3**
WMC	21	28	12	1	1	0	0.50	0.50	0
LOI	3	5	5	1	1	1	0.25	0.25	0.25
LOCM	2	3	4	0	0	0	0	0	0
NOC	15	2	3	0	1	1	0	0.25	0.25
CBO	4	5	3.5	0	0	0	0	0	0
RC	16	32	20	0	0	0	0	0	0

Table6. Intermediate values of different variables for each project



"***Note***. This graph shows the effect on maintainability by C & K metrics fordifferentprojects.

Project	**∑EOM**	**∑MF**	**TEOM**	**TM**
P1	0.75	3.5	0.21	0.79
P2	1.0	3.5	0.28	0.72
P3	0.50	3.5	0.14	0.86

Table7. Final result showing maintainability of each project

Chapter 6

6 CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

Overall aim of this research is to analyze maintainability of project on the basis of C & K metrics suite. Mapping of class level metrics to project level metrics is initial change in C & K metrics which helps us in obtaining maintainability at project level. Results show the dependency of multiplying factor with maintainability. If result shows the very low percentage of maintainability, then the metrics whose value are beyond the threshold value are collected. Then features which affect those metrics need to be correct. Methodology itself does not correct the maintainability of projects, but it guides developers and maintainers what to correct? And how to correct? The result of methodology will help the maintainer to understand the effort required in maintaining the project and also helps the developer to know that design is good or not, because if effect on maintainability increases the effort require to maintain the project also increase.

6.2 Future scope

These limitations can be removed with better approach. On studying different project of different size better threshold value can be defined for each category of project.Projects can be categorized in various category like small projects,medium projects and large project and then threshold can be defined for each categories separately. Practical approach can be defined to calculate multiplying factor. This specification will provide better accuracy in result.

REFERENCES

[1] Oman, Paul, and Jack Hagemeister. "Metrics for assessing a software system's maintainability." *Software Maintenance, 1992. Proceerdings., Conference on*. IEEE, 1992.

[2] Prieto-Díaz, Rubén. "Status report: Software reusability." *software, IEEE* 10.3 (1993): 61-66.

[3] Boehm, Barry W. "Software risk management: principles and practices."*Software, IEEE* 8.1 (1991): 32-41.

[4] Palmer, Jonathan W. "Web site usability, design, and performance metrics."*Information systems research* 13.2 (2002): 151-167.

[5] Schneidewind, Norman F. "Software Reliability Metrics." *Systems and Software Engineering with Applications* (2009): 129-151.

[6] Jogalekar, Prasad, and Murray Woodside. "Evaluating the scalability of distributed systems." *Parallel and Distributed Systems, IEEE Transactions on*11.6 (2000): 589-603.

[7] Bellovin, Steven M. "On the brittleness of software and the infeasibility of security metrics." *IEEE Security & Privacy* 4 (2006): 96.

[8] Voas, Jeffrey M., and Keith W. Miller. "Semantic metrics for software testability." *Journal of Systems and Software* 20.3 (1993): 207-216.

[9] Kirakowski, Jurek, and Mary Corbett. "SUMI: The software usability measurement inventory." *British journal of educational technology* 24.3 (1993): 210-212.

[10] D. Sjøberg, B Anda and A. Mockus, "Questioning Software Maintenance Metrics: A Comparative Case Study."*ESEM*, Lund, Sweden, 2012.

[11] Tegarden, David P., Steven D. Sheetz, and David E. Monarchi. "Effectiveness of traditional software metrics for object-oriented systems." *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*. Vol. 4. IEEE, 1992.

[12] Jamali, Seyyed Mohsen. "Object oriented metrics." *A survey approach Technical report, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran* (2006).

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

[14] Hitz, Markus. "Chidamber and Kemerer's metrics suite: a measurement theory perspective." *Software Engineering, IEEE Transactions on* 22.4 (1996): 267-271.

[15] Etzkorn, Letha, Carl Davis, and Wei Li. "A practical look at the lack of cohesion in methods metric." *Journal of Object-Oriented Programming*. 1998.

[16] Harrison, Rachel, Steve J. Counsell, and Reuben V. Nithi. "An evaluation of the MOOD set of object-oriented software metrics." *Software Engineering, IEEE Transactions on* 24.6 (1998): 491-496.

[17] Abreu, Fernando Brito, Rita Esteves, and Miguel Goulão. "The design of eiffel programs: Quantitative evaluation using the mood metrics." *Proceedings of TOOLS'96*. 1996.

[18] Gomathi. S, Edith Linda. P." An overview of Object Oriented Metrics A complete Survey." *International Journal of Computer Science & Engineering Technology (IJCSET).* ISSN: 2229-3345,2013.

[19] Harrison, R., S. Counsell, and R. Nithi. "An overview of object-oriented design metrics." *Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering]*. IEEE, 1997.

[20] Genero, Marcela, Mario Piattini, and Coral Calero. "Early measures for UML class diagrams." *L'objet* 6.4 (2000): 489-505.

[21] Li, Wei, and Sallie Henry. "Maintenance metrics for the object oriented paradigm." *Software Metrics Symposium, 1993. Proceedings., First International*. IEEE, 1993.

[22] Jamali, Seyyed Mohsen. "Object oriented metrics." *A survey approach Technical report, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran* (2006).

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

[24] Tegarden, David P., Steven D. Sheetz, and David E. Monarchi. "Effectiveness of traditional software metrics for object-oriented systems." *System Sciences, 1992.*

Proceedings of the Twenty-Fifth Hawaii International Conference on. Vol. 4. IEEE, 1992.

[25] Morris, Kenneth Lee. *Metrics for object-oriented software development environments*. Diss. Massachusetts Institute of Technology, 1989.

List of Publications

Title	Calculation of Maintainability for java project using C & K metrics site
Authors	Piyush Nagar, V.K. Bhalla
Publication	Second International Symposium on Computer Vision and the Internet (VisionNet'15)
Status	Communicated

VIDEO LINK

https://www.youtube.com/watch?v=K-kP5AQ_2Z0