Improvised Software Code Comprehension Using Data Mining

Ram Gopal Gupta
Department of Computer Science and Engineering, VMSB Uttarakhand Technical University, India
rgmail@rediffmail.com

Ankur Dumka
Department of Computer Science and Engineering, Women Institute of Technology-UTU Campus, India
ankurdumka2@gmail.com

Bireshwar Dass Mazumdar
School of Computer Science Engineering and Technology, Bennett University India
bireshwardm@gmail.com

Abstract: *Millions of lines of code are used to create the modern software applications, which are more complicated in terms of their structure, behaviour, and functionality. The rapid advancement of supporting and enabling technologies, for example, is one reason why the development life cycles of these applications show a propensity to get shorter. As a result, a growing amount of the expense associated with software development moves from the generation of new artefacts to their adaption. Understanding the layout, functionality, and behaviour of current code artefacts is essential to this activity. The task of understanding code is crucial to software maintenance. We employed data mining techniques including clustering, classification, and associative rules to improvise software code comprehension*.

Keywords: *Software code comprehension, code mining, software maintainability, association, classification, correlation, coupling, cohesion.*

Received December 22, 2023; accepted May 9, 2024
https://doi.org/10.34028/iajit/21/3/15

1. Introduction

Understanding software source code is crucial to performing software maintenance activities [2, 18, 19, 28, 32]. For this, a variety of traditional and intelligent computing techniques have been used. However, it has been demonstrated that data mining techniques are quite beneficial and effective in retrieving necessary information at various levels of abstraction. In order to extract data about a specific C++ software from its source code, correlation must be determined at the class and function levels. It governs how difficult it would be to alter a function within a class and how that would affect the other correlated classes. We give a succinct overview of data mining techniques with a focus on source code comprehension in this paper.

Numerous elements of software maintenance, including corrective maintenance, adaptive maintenance, preventive maintenance and perfective maintenance, and approximation of maintenance effort time and cost, have made substantial use of data mining techniques [5, 17]. The deployment of computing methods and the software design perspective are the two fundamental perspectives on software maintenance. Data mining has been shown to be a very flexible and valuable way for analyzing software from the level of the source code to the level of the program [1]. The interpretation and analysis of source code frequently use clustering techniques from the data mining perspective. Here, we discuss some of the pertinent research and methods in this area.

2. Literature Review

Software is being utilized and developed in greater quantities as a result of the rising demand. Software is getting significantly bigger. Software maintenance tasks including repair, expansion, and improvement start once the software is in use [38]. The process of maintaining software involves understanding, modifying, and reconfirming the software [34]. The present software business is dealing with significant concerns related to program maintenance. The secret to effective software maintenance is having a precise, quick, and thorough grasp of the program. The best way to maintain software is therefore to analyse and comprehend the program.

Clustering method is applied, in order to extract high level subsystems [15, 26]. They have defined linkages: single, combined, weighted and unweighted average among various individual clusters, to measure the detachment and likeness between the new cluster and the other objects (A, B, C). They have demonstrated that their correlation measure operates similarly to the Jaccard metric by using the detachment and correlation metrics for dual features.

Classified the issues in the software alteration record using machine learning and data mining approaches [27]. For the cataloguing of relevant and irrelevant files using a cohesiveness measure, they have combined syntactical and text-based features. A relationship that may be used to anticipate if an alteration in a source file may need variable alteration in other file has been demonstrated using the induction approach of machine learning.

The difficulty of the program assembly, a solution to the problem of clustering software segments has been offered [25, 29]. By constructing the complicated network module in accordance with the software system, they convert the software module clustering problem into a graph clustering problem. The software module clustering problem is then solved using the SPS software module clustering algorithm, and the best clustering scheme is obtained by dynamic adjustment and optimization that adheres to the "high cohesion" and "low coupling" software design principles.

Software segmentation, recovery, and restructure utilizing clustering approaches in Kunz and Black [12] and Lung *et al*. [14]. They sought to incorporate software applications using numerical taxonomy clustering techniques. The approach has been used in:

- Software division during the design phase.
- In the process of reverse engineering.
- Software reorganization to facilitate evolution throughout the maintenance phase.
- Enhancing the coupling and cohesion source code.

In order to find modification patterns-a method which uses data mining over frequently modified source files to analysis the alteration history of the code [37]. They used frequent pattern-mining algorithm for association rule mining and tested their methodology on different sizable projects named Mozilla, Eclipse. Their approach has been contrasted in a Comma Separated Values (CSV) dataset with an association rule to recommend program-code that could be related to a specified source-code portion which describes changing associations between files or functions [3, 36].

An association rule mining system based on Meta rules [18] for pattern interpretation in software system supplied source files. On five legacy systems, association mining restrictions have been put in place. Support, Confidence, and Coverage were utilised as three indicators to assess the success of the association rules [30].

A strategy based on data mining was developed to understand the source code of Object-Oriented System (OOS) for maintainability [9, 10]. In order to more clearly describe this class entity, presented the code in Extensible Markup Language (XML) format and taken into account a variety of Object Oriented Programming (OOP) metrics. A data mining technique called K-means clustering uses different input sets for classes based on metric values and classes based on structural criteria, such as the packages or subclasses to which they belong. A program called JBoss, which includes 569 packages, 4,717 Java files, 6,448 classes and 1,615,289 lines of code has been taken into account.

The time to extract both static and dynamic dependency graphs from the vast Mozilla.browser.1.3v software system in an effort to evaluate the efficacy of employing dependencies between software components as input to a clustering approach [35]. Experimented with various software clustering algorithms in a number of different ways. By gathering data on function calls made during runtime and outputted as output, they were able to identify the dynamic dependency. The file that defines the relation between the files substitutes functions in a specially built script. From a total of 3,559 source files, they discovered a dynamical dependency between 1,023 of them in this context. Then, they have employed several filtering techniques to extract relevant information.

The ability to understand software code is crucial for code analysis [4]. Based on a review of the literature, we have approached this issue by employing data mining techniques to extract data at the file, class, and method levels in order to compute coupling and cohesion using various methods in this situation.

Creating new correlation metrics for calculating cumulative coupling and cohesion indices is the main goal of our work [22]. Source code understanding was used to extract file, class, method, and other parameter level information from source code [21]. Data mining techniques such as C5.0, clustering and association were used to examine the data at the file, class, and method level. A knowledge base was created using this information. In this work, we use standard class-level calculation methods to relate the numbers of a particular class to the different number of classes at various levels (Very Low (VL), Low (L), Medium (M), High (H) and Very High (VH)). Correlations between classes were established based on shared functions within and between classes. This article covers the following topics: Section 1 describes the purpose of the work and section 2 defines the literature review. Problem description is stated in section 3 and solved in section 4 with k-means data mining techniques and the association algorithm Apriori. Major contributions are described in section 5 commutative coupling and cohesion are computed through correlation matrices. Section 6 describes the discussion. Section 7 defines the conclusion of the work.

3. Problem Description

We have worked on software source-code of GitAhead C++ project. It is an open-source Git client Graphical User Interface (GUI) and is accessible to all. It is a Medium-Large size application consists 494 classes, 9,902 functions and around 4,312 parameters.

In our research, we went through the steps of getting the software code data from the GitAhead project then extracting all pertinent data about the parent classes, classes, data members, functions, function parameters using one of the program code comprehension tools sci-understand [33]. The segmental problem-solving approaches have been used. The output from one module is passed on to the next in a sequential order. Each module's performance can be checked again. Block diagrams are used in this method to depict the different key phases and descriptions, as seen in Figure 1.

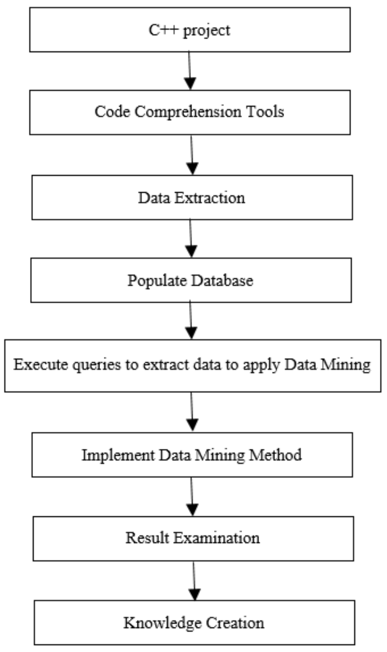


Figure 1. Process work flow.

3.1. Explanation of Block Diagram in Figure 1

- C++ Project: in first phase's, used GitAhead project with open source code developed in the C++ programming language.
- Code Comprehension Tools: the sci-understand tools are now used to parse source code projects. They parse the entire project and provide information on its classes, functions, member functions, and other important components. Other programming languages supported by this tool include Java, C#, Python, Ada, etc. It has the capacity to parse any project type and provides important details for understanding source code.
- Data Extraction: this step involves taking the parsed output data from the tool, modifying it for the input model, and saving it as an excel file. Now, we'll utilise this excel file as an input to fill out a database with all the information required for our job.
- Populate Database: the populated database using the excel data that was used as input during this step. The database was created using import and export functionality of MySQL Service.
- Execute Queries to Extract Data to Apply Data Mining: after building the database, we have executed Structured Query Language (SQL) to obtain the information we need. The next phase used this resultant data which is stored into the excel file to apply the data mining techniques.
- Implement Data Mining Methods: to the result produced from the preceding phase, put on various data mining techniques, such as classification, association and clustering then analyse each technique, which will be covered under section 4 below.
- Result Examination: the results will include association method-Apriori, the rule set-C5.0, and clustering with K-means of the class, functions, etc., which will be covered in further detail in sections 4 through 5.
- Knowledge Creation: In this stage, we will build a knowledge base with information about the job that is both quantitative and qualitative, as well as a correlation matrix and a qualitative correlation matrix. This block derives coupling, the National Correct Coding Initiative (NCCI), Correct Coding Initiative (CCI) and cohesion using diverse levels of notion.

3.2. SQL Queries to Extract Data

The exported data in excel file format through software code comprehension tools is imported into MySQL to populated the database such as clustermatrix, Filematrix, classmatrix, classdependencies, functionmatrix etc., then executed several SQL queries to extract the data from the database on different classes, function, it's types etc., so that the desired outcome can be derived. for example:

- To determine which cluster we have to consider in out study: The cluster that has high number of records.
  ```
  Select cluster_name, count(*)
  from clustermetrix
  group by cluster_name
  ```
- To find out which class is more valuable: Through getting the occurrence of the classes in cluster2
  ```
  Select count(*) cnt, ClassID, ClassName
  from ClassMatrix
  where ClassID in
  (Select ClassID from ClusterMatrix where
  Cluster_name=….)
  group by ClassID
  order by cnt desc
  ```
- To know the most appreciated function: by getting the frequently used function
  ```
  Select count(*) fcnt, functionname
  from FunctionMetrics fm, (select count(*) cnt,
  ClassID, ClassName
  from ClassMetrix
  where ClassID in
  (select ClassID from ClusterMetrix where
  Cluster_name=…)
  group by ClassID
  order by cnt DESC) clData
  where fm.ClassID = clData.ClassID
  group by FunactionName
  order by clData.ClassID
  ```
- To extract the function types used in majority to the given cluster

```
Select count(*), functiontype
From FucntionMetrix
where ClassID in
(Select ClassID from mfun
where mfun.cnt= (Select Max(cnt) from
(Select count(*) cnt, ClassID, ClassName from
ClassMatrix where ClassID in
(Select ClassID from ClusterMetrix where
Cluster_Name=….)
group by ClassID) mfun))
group by functiontype
………………………………..
………………………………..
```

4. Application and Result Analysis

4.1. K-means Clustering

The project comprises of 494 classes and 9,902 functions. In our attempt to cluster data using the K-means algorithm of data mining, we were successful in obtaining five clusters. Significant source code information is present in each cluster. In this study, we took into account the cluster2 data and examined the cluster data for several factors, as indicated in Table 1. This table reveals that cluster 2 accounts for the majority of the data and includes 2905 records out of 2994. Similar results may be found for clusters 1, 3, 4, and 5, which include 34, 13, and 23 records, respectively. These records show the percentage of frequency with which each clustered class, class id, file, function, and function type occurs. The class HunkWidget has the maximum percentage, or 9.35 percent of occurrence, among the additional classes in the identical cluster2, as shown in Table 1. In the same cluster, class ID, file, function, and function type percentage occurrences are 9.35 percent, 94.21 percent, 2.34 percent, and 100 percent, respectively. It also provides information on which functions belong to which classes, their function types, and how many times a function has been applied to a cluster, for example, it also provides information on how many specific functions are included in a given class.

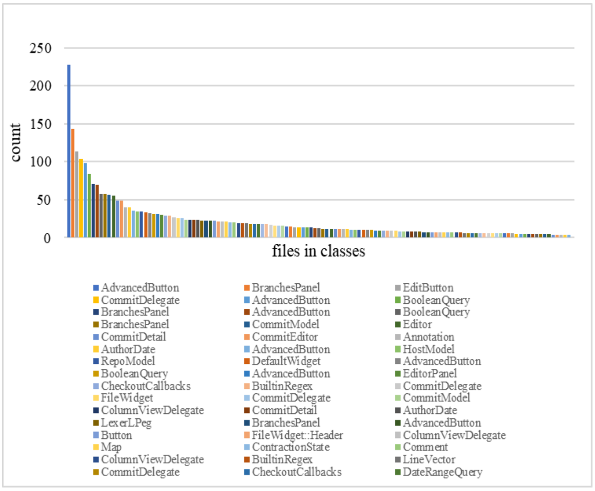
Table 1. Cluster-wise result where inputs are class, class ID, file, function and function type.

Cluster	Records	File	Class ID	Class	Function	Function type
Cluster 1	34	CommitList.cpp (38.23%)	225 (41.32%)	CommitModel (41.32%)	fetchMore (17.22%)	public virtual (31.25%)
Cluster 2	2905	DiffView.cpp (94.21%)	549 (9.35%)	HunkWidget (9.35%)	Invalidate (2.34%)	public (100%)
Cluster 3	13	StartDialog.cpp (88.34%)	902 (99.78%)	RepoModel (99.78%)	setShow (12.11%)	public (12.11%)
Cluster 4	19	Commit.cpp (92.45%)	133 (100%)	CommitDelegate (100%)	paint (24.56%)	public (24.56%)
Cluster 5	23	ConfigDialog.cpp (58.35%)	118 (58.35%)	GeneralPanel (58.35%)	qt_metacall (17.65%)	public virtual (50%)

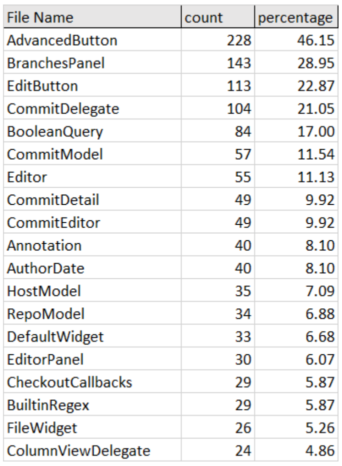
4.1.1. Class and File Level Examination in Cluster

The two files "AdvancedButton.cpp" and "BranchesPanel.cpp" have important contributions in cluster2 as we can see in Figure 2-a) and (b). This allows us to perform analysis at the file and class level.

Additionally, we observe that when software maintenance is concerned with files and classes, files with single connections--or, more precisely, links-to other classes play a significant role. Quantitative data about the number and proportion of classes in each File are shown in Figure 2-c). The file 'AdvancedButton.cpp' has the maximum percentage of 46.15 percent and count of 228; the lowest percentage for files like 'DiffCallbacks.cpp,' 'unpacked.cpp,' etc., is 0.20 percent and count of 1. It indicates that, among other files used in the project, AdvancedButton.cpp is the most frequently used and important file.

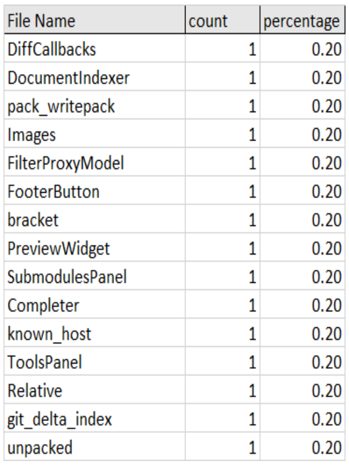


a) Depiction of file and classes.



File Name	count	percentage
AdvancedButton	228	46.15
BranchesPanel	143	28.95
EditButton	113	22.87
CommitDelegate	104	21.05
BooleanQuery	84	17.00
CommitModel	57	11.54
Editor	55	11.13
CommitDetail	49	9.92
CommitEditor	49	9.92
Annotation	40	8.10
AuthorDate	40	8.10
HostModel	35	7.09
RepoModel	34	6.88
DefaultWidget	33	6.68
EditorPanel	30	6.07
CheckoutCallbacks	29	5.87
BuiltinRegex	29	5.87
FileWidget	26	5.26
ColumnViewDelegate	24	4.86

b) Praportion of classes in their individual files.



File Name	count	percentage
DiffCallbacks	1	0.20
DocumentIndexer	1	0.20
pack_writepack	1	0.20
Images	1	0.20
FilterProxyModel	1	0.20
FooterButton	1	0.20
bracket	1	0.20
PreviewWidget	1	0.20
SubmodulesPanel	1	0.20
Completer	1	0.20
known_host	1	0.20
ToolsPanel	1	0.20
Relative	1	0.20
git_delta_index	1	0.20
unpacked	1	0.20

c) Percentage and count of classes in their individual files.

Figure 2. Percentage and count of classes in their individual files.

4.1.2. Function and Class Level Examination in Cluster

In cluster2, we can observe the correlation in great extent between class and function. The class "LexState" in Figure 3 has the most functions, close to 150 functions. There are numerous common functions that are connected to various classes; this correlation matrix will be covered in a later section. As indicated in Table 2, there is also a percentage and count of functions for each class of cluster 2. For example, the class "LexState" has 144 functions and a 5.16 percent frequency within cluster 2, which is a significant contribution to this cluster and the project code as a whole.

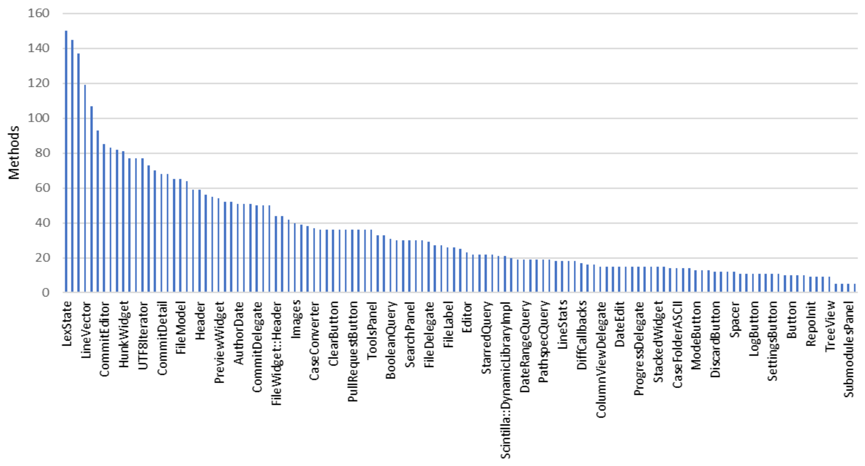


Figure 3. Representation of file and classes.

Table 2. Percentage and count of functions in class of cluster 2.

Class	Count	Percentage (%)
LexState	144	5.16
ContractionState	141	4.99
CommitModel	110	4.72
LineVector	115	4.10
….	….	….
….	….	….
ToolsPanel	25	1.24
DocumentIndexer	22	1.14
Indexer	22	1.14
….	….	….
TreeView	9	0.31
CommentWidget	5	0.17
….	….	….

4.2. Apriori Association Method

In data mining, the Apriori approach is mostly used for data association [13]. Before being utilized as a parameter in an association rule, specific terms must be defined and illustrated.

We have taken into account the 5 classes and 4 related functions in this situation, as in Table 3. As we can see in the MarginView class, there are two functions; if a function is in a class, indicated by 1, otherwise 0. The functions Compile and PaintMargin have been taken into consideration, whereas FindText and LayoutLine have not been. Four additional classes and their corresponding function combinations are present. The following association approaches' characteristic (metrics) will be easier to grasp with the aid of this table.

- Support: the portion of occurrences in the dataset that comprise the itemset is support or supp(A), of an itemset, for example,

  Supp(*Compile*, *PrintMargin*)=1/5= 0.2, *meaning that 20% of all occurrences involve it.*

- Confidence: the likelihood that an antecedent and a consequent will appear in the same transaction is indicated by a rule's confidence level. The conditional likelihood that something will happen given a specific antecedent, for example,

  Conf (=>A)=Supp(AUB)/Supp(A)
  Rule for $\{\underset{\displaystyle A}{\underset{\downarrow}{\text{PrintMargin}}}\}\Rightarrow\{\underset{\displaystyle B}{\underset{\downarrow}{\text{LayouLine}}}\}$

Hence, *Conf*(*A*=> *B*) = *Supp*(*AUB*)/*Supp*(*A*) => (1/5)/(2/5) = 1/2 = 0.5, means the rule has 50% confidence.

- Lift: denoted as *Lift*(*A*=>*B*)=*Supp*(*AUB*)/*Supp*(*A*)* *Supp*(*B*)=>0.2/(0.2 * 0.4)=0.2/0.08=2.5.
- Instances: the quantity of instances for a specific consequence and antecedent is known as instance.
- Rule provision*:* reflects IDs percentage for whose rule or consequent, antecedent is true. For instance, if FindText and Compile together exits 20% in exercise data, then the rule FindText=>Compile is supported by 20% IDs.
- Deployment: it measures the proportion of training data that meets the prerequisite for the antecedent and excluding consequent.

$$Deployment = \frac{(\,Antecedent\ support\ in\ \#of\ recors) - (Rule\ provision\ in\ \#records)}{Number\ of\ recods}$$

where

- Antecedent support: how many entries have the antecedent being true in them
- Rule provision: entries where the consequent and the

antecedent are both true.

Table 3. Class and functions relation.

Class \ Functions	FindText	Compile	PaintMargin	LayoutLine
Document	1	0	0	0
RESearch	1	0	0	0
MarginView	0	1	1	0
EditView	0	0	1	1
RepoView	0	1	0	0

4.2.1. Class and File Level Association

We identified a number of rules and a substantial correlation between the cluster2 data and the file AdvancedButton.cpp using the Apriori method. Because with 98.74% and 8.52% levels of confidence and support, class "LexState" is used 239 times in this consequence, Table4 demonstrates that it has a substantial association with five antecedent circumstances. The deployment and lift of the rule are also 0.107 and 1.04, respectively. The class "LexSate" has the most confidence, instances and support in a specific file called AdvancedButton.cpp, with 98.74 percent, 239 and 8.52 percent respectively, while the class "ContractionState" has the fewest confidence, instances and support, with 97.94 percent, 146 and 5.20 percent. To ensure that any changes to a class would need a correspondingly high number of changes, the information acquired would be helpful in enumerating the relevance and closeness of a class inside a file. As a result, class "LexState" undergoes the most changes, whilst class "ContractionState" undergoes the least changes.

Table 4 illustrates the comparison between Figures 2-c) and 3. We can see that the rule was mostly developed for the "AdvancedButton.cpp" file because it made a significant contribution.

Table 4. Association rules for file and class in cluster2 data.

Consequence (all are .cpp files)	Antecedent	Rule ID	Instances	Confidence%	Support%	Rule provision%	Deployment	Lift
AdvancedButton	Class=LexStae and ClassID=31 and FunctionTypp=Public Function	31	249	97.34	8.52	8.41	0.107	1.04
AdvancedButton	Class=LexStae and ClassID=26 and FunctionTypp=Public Function	30	170	98.23	6.06	5.95	0.107	1.04
AdvancedButton	Class=LexStae and ClassID=48 and FunctionTypp=Public Function	29	164	98.17	5.85	5.74	0.107	1.03
AdvancedButton	Class=LexStae and ClassID=53 and FunctionTypp=Public Function	28	149	97.98	5.31	5.20	0.107	1.03
BranchesPanel	Class=ContractionState and ClassID=38 and FunctionType=PublicFunction	27	146	97.94	5.20	5.10	0.107	1.03

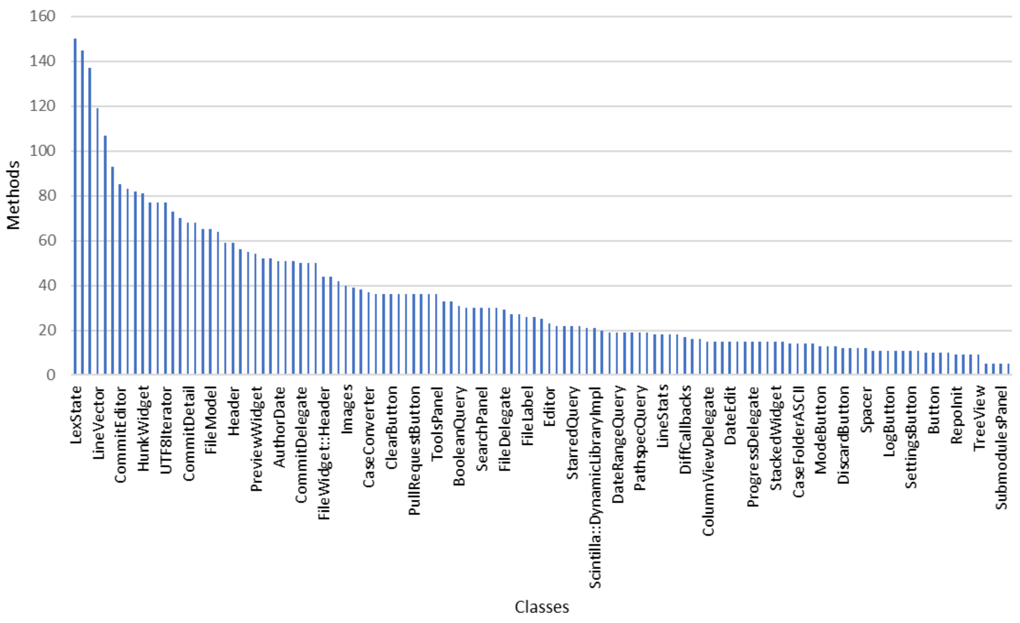


Figure 4. Functions association in classes within cluster2.

4.2.2. Association between Function and Class

The relationship among the classes and the functions defined for them in cluster2 is illustrated in Figure 4. There were 861 rules generated for 2905 entries. Table 5 displays the study of the association rule and observed how the functions are linked to their respective classes.

Table 5. Study of class and function of rule for cluster2.

Records	2,905
Rule produced	861
Maximum support	0.178%
Minimum support	0.036%
Maximum confidence	100%
Minimum confidence	20%
Maximum lift	2803.0%
Minimum lift	2.932%

4.3. C5.0 Method

As shown in Figure 5, the cluster2 data was also exposed to the data mining approach of rule generation, and interesting rules for the occurrences and significance of those classes and techniques were found. A class and several functions are associated within a rule. The "LexState" class has a lot of connectedness with functions, and most rules have been written for it, according to the graph. The rule here gives the overall number of functions pertaining to or linked to the particular class. We receive the confidence level for the rule generated and the number of occurrences of the function in each rule, for example, Rule-4 for the class DefaultWidget (1; 0.667).

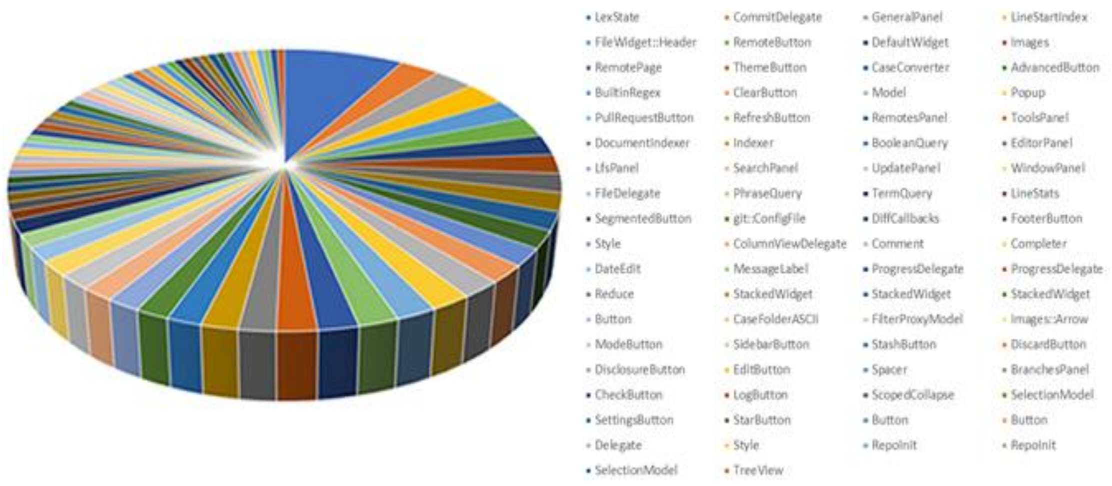


Figure 5. Class and functions for rule set with C5.0.

If Function=ClassInstance, then DefaultWidget: the function "ClassInstance" belongs to the class "DefaultWidget" using just one instance of their relationship, and the rule has a confidence level of 66.7 percent.

4.3.1. Rule Generations

The rule created for each class and its related functions was acquired when the C5.0 methodology was applied, as shown in Table 6. This table lists all of the rules that have been created for each class; for example, there have been 2, 1, and 121 rules generated for model, editor, and LexState, respectively. The chart also shows the total rule produced for the other classes.

Table 6. Class and total rules produced with C5.0.

Class	Total Rules Produced
Model	2
Editor	1
ListModel	3
……	……
……	……
DefaultWidget	45
Label	8
LexerPool	15
……	……
……	……
LexState	121
DiffCallbacks	9
..	…

The extended version of the rules is displayed in Table 7, where each associated rule's condition is shown in terms if-then, and the values inside the brackets are structured as (x; y), signifying the rule's confidence level and the number of instances. For instance, ListModel (3; 0.8) rule shows that the rule's confidence level is 0.8, or 80%, and that the rule has been applied to three occurrences of the condition.

Table 7. Rules produced with reference to Table 6.

Rules for Editor-contains 1 rule(s) Rule 1for Editor (1; 0.667) if Function=Editor then Editor
Rules for ListModel-contains 3 rule(s) Rule 1 for ListModel (3; 0.8) if Function=ShowList then ListModel Rule 2 for ListModel (9; 0.182) if Function=GetList then ListModel Rule 3 for ListModel (13; 0.133) if Function=SetModel then ListModel
Rules for Model-contains 2 rule(s) Rule 1 for Model (1; 0.667) if Function=SetModel then Model Rule 2 for Model (1; 0.667) if Function=OnOpenModel then Model ….. ….. ….

5. Knowledgebase Formation

Using the aforementioned beneficial outcomes for GitAhead project, On the basis of shared behaviour between classes, the knowledgebase formation has been taken place. The same process we have adopted for two other open-source projects Personal Software Process (PSP) and Jet. These are downloaded from GitHub and Thales [6, 23]. The actual structure of these projects has been compared to discuss the outcome validity, novelty and usefulness to the software maintainer.

5.1. First Open-Source Project Study

The GitAhead application knowledge formation has been explained here in detail, using the common behaviour between classes, we may create a correlation matrix. This correlation matrix shows the common functions found in the related classes; for instance, in the given below table; first row shows the C1 class, and the data in adjacent columns displays how many functions in C1, C3, C6, and C51 are shared. The count of shared functions between C1 and C3, C5, C6, C7, and the rest will be represented by equivalent column values in a second row, C3. An analytical numerical breakdown of the class and techniques is provided by this symmetric matrix. Here, the class name is represented by C1, C3, ...C53 as illustrated in Figure 6.

Additionally, we have removed any matrix members with values lower than 2, or filters above a threshold.

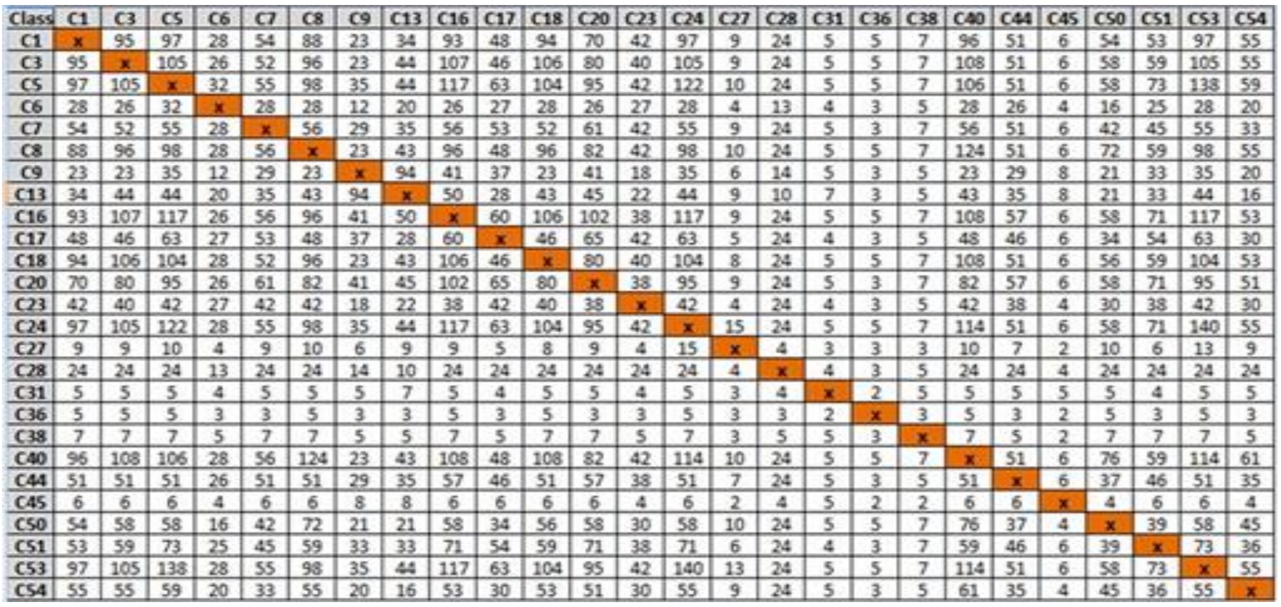
For simplicity, the various class names labelled as C1, C3, and C5, etc., Our database contains details on each Ci and the name of the related class.

Now, based on the five qualitative values (Very High-VH, Hight-H, Medium-M, Low-L, Very Low-VL) and the correlation matrix in Figure 6, we constructed a new qualitative matrix.

For each of the aforementioned qualitative characteristics, we have allocated a range of numerical values, as indicated in Table 8. The frequent functions used count shown in the Figure 6 correlation matrix.

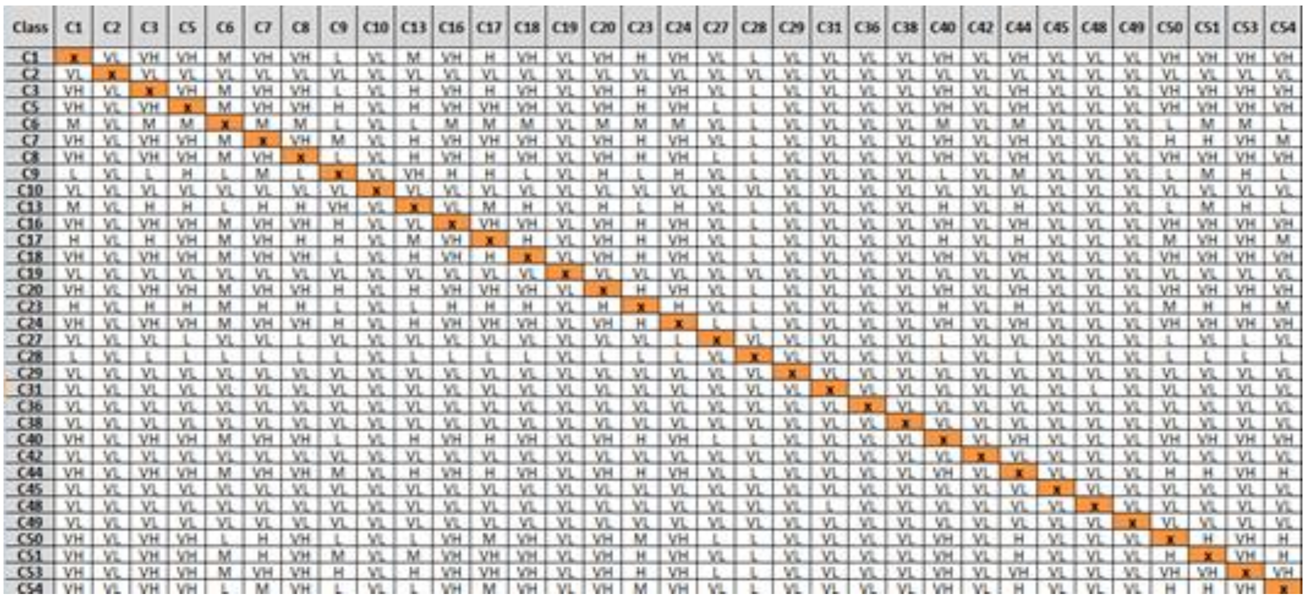
Table 8. The qualitative values' for quantitative range (X denotes the count of functions).

X<10	VL
X>=10 AND X<25	L
X>=25 AND X<35	M
X>=35 AND X<50	H
X>=50	VH



Class	C1	C3	C5	C6	C7	C8	C9	C13	C16	C17	C18	C20	C23	C24	C27	C28	C31	C36	C38	C40	C44	C45	C50	C51	C53	C54
C1	x	95	97	28	54	88	23	34	93	48	94	70	42	97	9	24	5	5	7	96	51	6	54	53	97	55
C3	95	x	105	26	52	96	23	44	107	46	106	80	40	105	9	24	5	5	7	108	51	6	58	59	105	55
C5	97	105	x	32	55	98	35	44	117	63	104	95	42	122	10	24	5	5	7	106	51	6	58	73	138	59
C6	28	26	32	x	28	28	12	20	26	27	28	26	27	28	4	13	4	3	5	28	26	4	16	25	28	20
C7	54	52	55	28	x	56	29	35	56	53	52	61	42	55	9	24	5	3	7	56	51	6	42	45	55	33
C8	88	96	98	28	56	x	23	43	96	48	96	82	42	98	10	24	5	5	7	124	51	6	72	59	98	55
C9	23	23	35	12	29	23	x	94	41	37	23	41	18	35	6	14	5	3	5	23	29	8	21	33	35	20
C13	34	44	44	20	35	43	94	x	50	28	43	45	22	44	9	10	7	3	5	43	35	8	21	33	44	16
C16	93	107	117	26	56	96	41	50	x	60	106	102	38	117	9	24	5	5	7	108	57	6	58	71	117	53
C17	48	46	63	27	53	48	37	28	60	x	46	65	42	63	5	24	4	3	5	48	46	6	34	54	63	30
C18	94	106	104	28	52	96	23	43	106	46	x	80	40	104	8	24	5	5	7	108	51	6	56	59	104	53
C20	70	80	95	26	61	82	41	45	102	65	80	x	38	95	9	24	5	3	7	82	57	6	58	71	95	51
C23	42	40	42	27	42	42	18	22	38	42	40	38	x	42	4	24	4	3	5	42	38	4	30	38	42	30
C24	97	105	122	28	55	98	35	44	117	63	104	95	42	x	15	24	5	5	7	114	51	6	58	71	140	55
C27	9	9	10	4	9	10	6	9	9	5	8	9	4	15	x	4	3	3	3	10	7	2	10	6	13	9
C28	24	24	24	13	24	24	14	10	24	24	24	24	24	24	4	x	4	3	5	24	24	4	24	24	24	24
C31	5	5	5	4	5	5	5	7	5	4	5	5	4	5	3	4	x	2	5	5	5	5	5	4	5	5
C36	5	5	5	3	3	5	3	3	5	3	5	3	3	5	3	3	2	x	3	5	3	2	5	3	5	3
C38	7	7	7	5	7	7	5	5	7	5	7	7	5	7	3	5	5	3	x	7	5	2	7	7	7	5
C40	96	108	106	28	56	124	23	43	108	48	108	82	42	114	10	24	5	5	7	x	51	6	76	59	114	61
C44	51	51	51	26	51	51	29	35	57	46	51	57	38	51	7	24	5	3	5	51	x	6	37	46	51	35
C45	6	6	6	4	6	6	8	8	6	6	6	6	4	6	2	4	5	2	2	6	6	x	4	6	6	4
C50	54	58	58	16	42	72	21	21	58	34	56	58	30	58	10	24	5	5	7	76	37	4	x	39	58	45
C51	53	59	73	25	45	59	33	33	71	54	59	71	38	71	6	24	4	3	7	59	46	6	39	x	73	36
C53	97	105	138	28	55	98	35	44	117	63	104	95	42	140	13	24	5	5	7	114	51	6	58	73	x	55
C54	55	55	59	20	33	55	20	16	53	30	53	51	30	55	9	24	5	3	5	61	35	4	45	36	55	x

Figure 6. Correlation matrix.



Class	C1	C2	C3	C5	C6	C7	C8	C9	C10	C13	C16	C17	C18	C19	C20	C23	C24	C27	C28	C29	C31	C36	C38	C40	C42	C44	C45	C48	C49	C50	C51	C53	C54
C1	x	VL	VH	VH	M	VH	VH	L	VL	M	VH	H	VH	VL	VH	H	VH	VL	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C2	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
C3	VH	VL	x	VH	M	VH	VH	L	VL	H	VH	H	VH	VL	VH	H	VH	VL	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C5	VH	VL	VH	x	M	VH	VH	H	VL	H	VH	VH	VH	VL	VH	H	VH	L	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C6	M	VL	M	M	x	M	M	L	VL	L	M	M	M	VL	M	M	M	VL	L	VL	VL	VL	VL	M	VL	M	VL	VL	VL	L	M	M	L
C7	VH	VL	VH	VH	M	x	VH	M	VL	H	VH	VH	VH	VL	VH	H	VH	VL	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	H	H	VH	M
C8	VH	VL	VH	VH	M	VH	x	L	VL	H	VH	H	VH	VL	VH	H	VH	L	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C9	L	VL	L	H	L	M	L	x	VL	VH	H	H	L	VL	H	L	H	VL	L	VL	VL	VL	VL	L	VL	M	VL	VL	VL	L	M	H	L
C10	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
C13	M	VL	H	H	L	H	H	VH	VL	x	VL	M	H	VL	H	L	H	VL	L	VL	VL	VL	VL	H	VL	H	VL	VL	VL	L	M	H	L
C16	VH	VL	VH	VH	M	VH	VH	H	VL	VL	x	VH	VH	VL	VH	H	VH	VL	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C17	H	VL	H	VH	M	VH	H	H	VL	M	VH	x	H	VL	VH	H	VH	VL	L	VL	VL	VL	VL	H	VL	H	VL	VL	VL	M	VH	VH	M
C18	VH	VL	VH	VH	M	VH	VH	L	VL	H	VH	H	x	VL	VH	H	VH	VL	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C19	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
C20	VH	VL	VH	VH	M	VH	VH	H	VL	H	VH	VH	VH	VL	x	H	VH	VL	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C23	H	VL	H	H	M	H	H	L	VL	L	H	H	H	VL	H	x	H	VL	L	VL	VL	VL	VL	H	VL	H	VL	VL	VL	M	H	H	M
C24	VH	VL	VH	VH	M	VH	VH	H	VL	H	VH	VH	VH	VL	VH	H	x	L	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	VH	VH
C27	VL	VL	VL	L	VL	VL	L	VL	VL	VL	VL	VL	VL	VL	VL	VL	L	x	VL	VL	VL	VL	VL	L	VL	VL	VL	VL	VL	L	VL	L	VL
C28	L	VL	L	L	L	L	L	L	VL	L	L	L	L	VL	L	L	L	VL	x	VL	VL	VL	VL	L	VL	L	VL	VL	VL	L	L	L	L
C29	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
C31	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	L	VL	VL	VL	VL	VL
C36	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
C38	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
C40	VH	VL	VH	VH	M	VH	VH	L	VL	H	VH	H	VH	VL	VH	H	VH	L	L	VL	VL	VL	VL	x	VL	VH	VL	VL	VL	VH	VH	VH	VH
C42	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL
C44	VH	VL	VH	VH	M	VH	VH	M	VL	H	VH	H	VH	VL	VH	H	VH	VL	L	VL	VL	VL	VL	VH	VL	x	VL	VL	VL	H	H	VH	H
C45	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL
C48	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	L	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL
C49	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL
C50	VH	VL	VH	VH	L	H	VH	L	VL	L	VH	M	VH	VL	VH	M	VH	L	L	VL	VL	VL	VL	VH	VL	H	VL	VL	VL	x	H	VH	H
C51	VH	VL	VH	VH	M	H	VH	M	VL	M	VH	VH	VH	VL	VH	H	VH	VL	L	VL	VL	VL	VL	VH	VL	H	VL	VL	VL	H	x	VH	H
C53	VH	VL	VH	VH	M	VH	VH	H	VL	H	VH	VH	VH	VL	VH	H	VH	L	L	VL	VL	VL	VL	VH	VL	VH	VL	VL	VL	VH	VH	x	VH
C54	VH	VL	VH	VH	L	M	VH	L	VL	L	VH	M	VH	VL	VH	M	VH	VL	L	VL	VL	VL	VL	VH	VL	H	VL	VL	VL	H	H	VH	x

Figure 7. Qualitative correlation matrix.

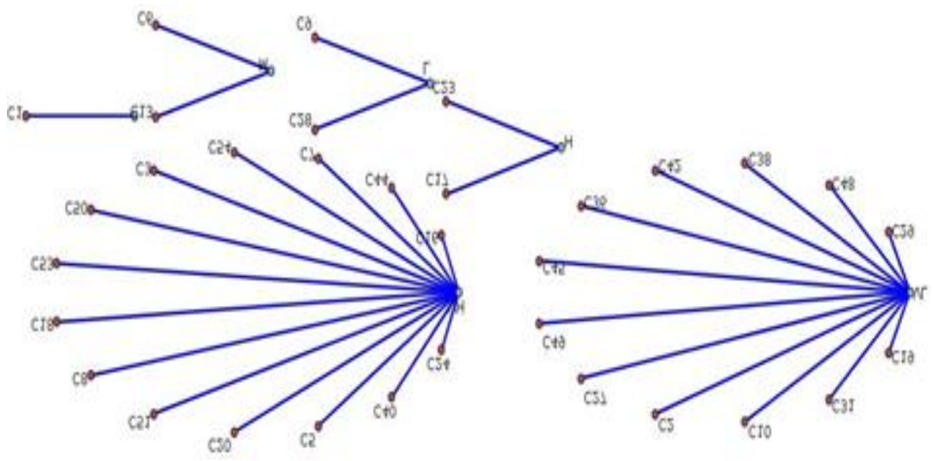


Figure 8. An illustration of class C1 using a qualitative correlation matrix.

Based on shared functions, table in Figure 7 illustrates the reasonable connection among C13 Class with others. For instance, the logical attribute (VH) linking Class C13 and Class C9 shows that they have more than 50 identical functions. Similar to this, logical relationships between classes C13, C3, and C5.., are linked with the logical attribute (H), which indicates that they share more than 35 but fewer than 50 functions in common. Similar to this, relationships between classes C13, C17 and C1 are connected using the attribute (M), indicating that they share more than 25 but fewer than 35 common functions; C13 and C6 and C28 classes are connected using the reasonable attribute (L), indicating that they share more than 10 but fewer than 25 common functions; and Classes C13 and C2, C10, C16, and others are connected by the attribute (VL), suggesting that there are a total shared functions between them. For further classes, including C1, C2, and others. We can also have web-graphical representations as shown in Figure 8 and explanations.

5.1.1. Normalised Coupling Computation

We standardise the VH's numerical value, which is represented in Table 9. Assuming that VH is equal to 0.8, in the row we find the occurrences of VH records, multiply by 0.8, After entering the outcomes into the matrix of column correlation, normalise the data using the procedure indicated below. In the column NCCI, we input the normalised value.

$$Normalisation(Ni) = [\{Ni - Minimum(i)\}/\{Maximum(i) - Minimum(i)\}] * [C + \{D - C\}]$$

which [C-D]=[0-1].

It has been discovered that C53, C24, C20, C16, and C5 classes are having a 100% coupling or correlation, but classes C9 and C13 have a minimal coupling of 0%.

Similar to this, by utilising the H, M, L and VL degrees of correlation by numerical value 0.6, 0.4, 0.2 and 0.1 respectively between the classes, we can also compute the normalised coupling index for certain classes, as shown in Tables 10, 11, 12, 13, and 14 appropriately.

In the example of "H" we can see that class C23 has a greatest coupling of 100% whereas classes "C16" and "C1" have a least coupling of 0% and similarly for M, L, VL is reflected in other tables.

Table 9. Highest and least coupling in Very High (VH).

Class	**C1**	**C3**	**C5**	**C7**	**C8**	**C9**	**C13**	**C16**	**C17**	**C18**	**C20**	**C24**	**C40**	**C44**	**C50**	**C51**	**C53**	**C54**	**CCI**	**NCCI**
C1	x	VH	VH	VH	VH			VH		VH	VH	VH	VH	VH	VH	VH	VH	VH	14*0.8-112	0.92
C3	VH	x	VH	VH	VH			VH		VH	VH	VH	VH	VH	VH	VH	VH	VH	112	0.92
C5	VH	VH	***x***	VH	VH			VH	VH	VH	VH	VH	VH	VH	VH	VH	VH	VH	12	1
C7	VH	VH	VH	x	VH			VH	VH	VH	VH	VH	VH	VH			VH		9.6	0.78
C8	VH	VH	VH	VH	x			VH		VH	VH	VH	VH	VH	VH	VH	VH	VH	11.2	0.92
C9						x	VH												0.8	0
C13						VH	x												0.8	0
C16	VH	VH	VH	VH	VH			x	VH	VH	VH	VH	VH	VH	VH	VH	VH	VH	12	1
C17			VH	VH				VH	x		VH	VH				VH	VH		5.6	0.42
C18	VH	VH	VH	VH	VH			VH		x	VH	VH	VH	VH	VH	VH	VH	VH	11.2	0.92
C20	VH	VH	VH	VH	VH			VH	VH	VH	***x***	VH	VH	VH	VH	VH	VH	VH	12	1
C24	VH	VH	VH	VH	VH			VH	VH	VH	VH	x	VH	VH	VH	VH	VH	VH	12	1
C40	VH	VH	VH	VH	VH			VH		VH	VH	VH	x	VH	VH	VH	VH	VH	11.2	0.92
C44	VH	VH	VH	VH	VH			VH		VH	VH	VH	VH	x			VH		8.8	0.71
C50	VH	VH	VH		VH			VH		VH	VH	VH	VH		x		VH		8	0.64
C51	VH	VH	VH		VH			VH	VH	VH	VH	VH	VH			x	VH		8.8	0.71
C53	VH	VH	VH	VH	VH			VH	VH	VH	VH	VH	VH	VH	VH	VH	x	VH	12	1
C54	VH	VH	VH		VH			VH		VH	VH	VH	VH				VH	x	8	0.64

Table 10. Highest and least coupling in High (H).

Class	C1	C3	C5	C7	C8	C9	C13	C16	C17	C18	C20	C23	C24	C40	C44	C50	C51	C53	C54	CCI	NCCI
C1	x								H			H								2*0.6=1.2	0
C3		x					H		H			H								1.8	0.08
C5			x			H	H					H								1.8	0.08
C7				*x*			H					H				H	H			2.4	0.16
C8					x		H		H			H								1.8	0.08
C9			H			x		H	H		H		H					H		3.6	0.33
C13		H	H	H	H		x			H	H		H	H	H			H		6	0.67
C16						H		x				H								1.2	0
C17	H	H			H	H			x	H		H		H	H					4.8	0.5
C18							H		H	x		H								1.8	0.08
C20						H	H				x	H								1.8	0.08
C23	H	H	H	H	H			H	H	H	H	*x*	H	H	H		H	H		8.4	1
C24						H	H					H	x							1.8	0.08
C40							H		H			H		x						1.8	0.08
C44							H		H			H			x	H	H		H	3.6	0.5
C50				H											H	x	H		H	2.4	016
C51				H								H			H	H	x		H	3	0.25
C53						H	H					H						x		1.8	0.08
C54															H	H	H		x	1.8	0.08

Table 11. Highest and least coupling in Medium (M).

Class	C1	C3	C5	C6	C7	C8	C9	C13	C16	C17	C18	C20	C23	C24	C40	C44	C50	C51	C53	C54	CCI	NCCI
C1	x			M				M													2*0.4=0.8	0.07
C3		x		M																	0.4	0
C5			x	M																	0.4	0
C6	M	M	M	*x*	M	M			M	M	M	M	M	M	M	M		M	M		6	1
C7				M	x		M													M	1.2	0.14
C8				M		x															0.4	0
C9					M		x									M		M			1.2	0.14
C13	M							x		M						M					1.2	0.14
C16				M					x												0.4	0
C17				M				M		x							M			M	1.6	0.21
C18				M							x										0.4	0
C20				M								x									0.4	0
C23				M									x					M		M	1.2	0.14
C24				M										x							0.4	0
C40				M											*x*						0.4	0
C44				M			M									x					0.8	0.07
C50										M			M				x				0.8	0.07
C51				M			M	M										x			1.2	0.14
C53				M															x		0.4	0
C54					M					M			M							x	1.2	0.07

Table 12. Highest and least coupling in Low (L).

Class	C1	C3	C5	C6	C7	C8	C9	C13	C16	C17	C18	C20	C23	C24	C27	C28	C31	C40	C44	C48	C50	C51	C53	C54	CCI	NCCI
C1	x						L									L									2*0.2	0.06
C3		x					L									L									0.4	0.06
C5			x												L	L									0.4	0.06
C6				x			L	L								L					L			L	1	0.21
C7					*x*											L									0.2	0
C8						x	L								L	L									0.6	0.1
C9	L	L		L		L	x				L		L			L		L			L			L	2	0.47
C13				L				x					L			L					L			L	0.6	0.1
C16									x							L									0.2	0
C17										x						L									0.2	0
C18							L				x					L									0.4	0.6
C20												x				L									0.2	0
C23							L	L					x			L									0.6	0.1
C24														*x*	L	L									0.4	0.06
C27			L			L								L	x			L			L		L		1.2	0.26
C28	L	L	L	L	L	L	L	L	L	L	L	L	L	L		x		L	L		L	L	L	L	4	1
C31																	x			L					0.2	0
C40							L								L	L		x							0.6	0.1
C44																L			x						0.2	0
C48																	L			x					0.2	0
C50				L			L	L							L	L					x				1	0.21
C51																L						x			0.2	0
C53															L	L							x		0.4	0.06
C54				L			L	L								L								x	0.3	0.15

Table 13. Highest and least coupling in Very Low (VL).

Class	C1	C2	C3	C5	C6	C7	C8	C9	C10	C13	C16	C17	C18	C19	C20	C23	C24	C27	C28	C29	C31	C36	C38	C40	C42	C44	C45	C48	C49	C50	C51	C53	C54	CCI	NCCI
C1	x	VL							VL					VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					12*0.1=1.2	0.047
C2	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	3.2	1
C3		VL	x						VL					VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C5		VL		x					VL					VL						VL	VL	VL	VL		VL		VL	VL	VL					1.1	0
C6		VL			x				VL					VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C7		VL				x			VL					VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C8		VL					x		VL					VL						VL	VL	VL	VL		VL		VL	VL	VL					1.1	0
C9		VL						x	VL					VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C10	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	3.2	1
C13		VL							VL	x	VL			VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.3	0.095
C16		VL							VL	VL	x			VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.3	0.095
C17		VL							VL			x		VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C18		VL							VL				x	VL				VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C19	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	3.2	1
C20		VL							VL					VL	x			VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C23		VL							VL					VL		x		VL		VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C24		VL							VL					VL			x			VL	VL	VL	VL		VL		VL	VL	VL					1.1	0
C27	VL	VL	VL		VL	VL		VL	VL	VL	VL	VL	VL	VL	VL	VL		x	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL		VL		VL	2.6	0.71

Table 14. Highest and least coupling in Very Low (VL).

Class	C1	C2	C3	C5	C6	C7	C8	C9	C10	C13	C16	C17	C18	C19	C20	C23	C24	C27	C28	C29	C31	C36	C38	C40	C42	C44	C45	C48	C49	C50	C51	C53	C54	CCI	NCCI
C28		VL							VL					VL				VL	x	VL	VL	VL	VL		VL		VL	VL	VL					1.2	0.047
C29	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	1.2	1
C31	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	*x*	VL	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL	3.1	0.95
C36	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	*x*	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	3.2	1
C38	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	3.2	1
C40		VL							VL					VL						VL	VL	VL	VL	x	VL		VL	VL	VL					1.1	0
C42	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	VL	VL	3.2	1
C44		VL							VL					VL				VL		VL	VL	VL	VL		VL	x	VL	VL	VL					1.2	0.047
C45	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	VL	3.2	1
C48	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	VL	3.1	0.95
C49	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	x	VL	VL	VL	VL	3.2	1
C50		VL							VL					VL						VL	VL	VL	VL		VL		VL	VL	VL	x				1.1	0
C51		VL							VL					VL				VL		VL	VL	VL	VL		VL		VL	VL	VL		x			1.2	0.047
C53		VL							VL					VL						VL	VL	VL	VL		VL		VL	VL	VL			x		1.1	0
C54		VL							VL					VL				VL		VL	VL	VL	VL		VL		VL	VL	VL				x	1.2	0.047

The idea of logical classification rule-based model to put them in qualitative composition rule is being used in Mazumdar and Mishra [20]. Here in Table 15 specifies the classes name, class ID for various Levels which are extracted from above results.

Table 15. Class Names with their ID and quantitative level [20].

CLASS	Class label	Level
CommitDelegate	C2	VL
CommitDetail	C5	VH
CommitEditor	C6	M
BuiltinRegex	C10	VL
CommitDelegate	C16	VH
GeneralPanel	C19	VL
LexerLPeg	C20	VH
LfsPanel	C23	H
CommitModel	C24	VH
RemotePage	C28	L
RemotesPanel	C29	VL
Indexer	C36	VL
Model	C38	VL
ByteIterator	C42	VL
HistoryButton	C45	VL
PhraseQuery	C49	VL
HunkWidget	C53	VH

5.1.2. Class Level Cohesion

We have defined in our work a formula to find cumulative cohesiveness of a class in regard to the function and types of function parameter inside a cluster. For example, we have taken into consideration the class "LexState," which has 144 methods and 5 parameter types. As indicated in Table 16, we have counted the actual instances of each parameter type and added them, totaling 115 in this instance. So, the following formula will be used to determine the cumulative cohesion for class "LexState".

The total value of the column indicates the number of times the parameter type has been used; for example, Table 16's total value of 67 reveals that 67 out of 144 methods in the class "LexState" have used the int parameter type. Similar to this, every other number represents the entire amount of that parameter type. As a result, we are able to determine how many different parameter types are present in a class as:

$$[67+44+2+1+1=115]$$

We define intra-class cumulative cohesion, or cumulative cohesiveness within a class, based on heuristics as shown in Table 16 between functions and parameters.

$$Cumulative\ cohesion = \frac{(Total\ number\ of\ occurrences\ of\ parameter\ types\ in\ a\ class)}{(Number\ of\ types\ of\ parameters * total\ number\ of\ methods)}$$

$$=115/(5*144)=0.16$$

Table 16. Count of function's parameters type for class 'LexState'.

Functions	int	const char *	Document *	uptr_t	LexerModule *
AllocateSubStyles	1	0	0	0	0
DescribeProperty	0	1	0	0	0
DescriptionOfStyle	1	0	0	0	0
LexState	0	0	0	0	0
LineEndTypesSupported	0	0	0	0	0
NameOfStyle	0	0	0	0	0
NameOfStyle	0	0	0	0	0
NamedStyles	0	0	0	0	0
PrimaryStyleFromStyle	1	1	0	0	0
PrivateCall	0	1	0	0	0
…	…	…	…	…	…
PropGetExpanded	1	0	1	0	0
PropGetInt	1	0	0	0	0
PropSet	0	0	0	0	0
PropertyNames	0	1	0	0	0
PropertyType	0	0	0	0	0
SetIdentifiers	0	1	1	0	0
…	…	…	…	…	…
SetLexerLanguage	0	1	0	1	0
SetLexerModule	0	1	0	0	1
SetWordList	1	0	0	0	0
Total Count	**67**	**44**	**2**	**1**	**1**

5.2. Second Open-Source Project Study

PSP is a small-medium size open-source project. It consists of 107 classes and 2,703 functions. After applying clustering techniques, four clusters were found and we considered cluster 1 for our examination on the basis of majority number of records and this cluster includes 539 records out of 634. Similarly, records were found for cluster 2, 3 and 4, which are 11, 46 and 38 respectively. These records show the percentage of frequency with which each clustered class, it's ID, file, function, and function type occurs. The class 'MainFrameBase' has the maximum percentage, or 14.45 percent of incidence, between the additional classes in the identical cluster1, as shown in Table 17. In the same cluster, class Id, File, Function, and Function Type percentage occurrences are 14.45 percent, 95.14 percent, 9.14 percent, and 100 percent, respectively. It also provides information on which functions belong to which classes, their function types, and how many times a function has been applied to a cluster, for example, it also provides information on how many specific functions are included in a given class.

Table 17. Cluster-wise result where inputs are class, class ID, file, function and function type.

Cluster	Records	File	Class ID	Class	Function	Function type
Cluster1	539	MainFrameBase.cpp (95.14%)	59 (14.45%)	MainFrameBase (14.45%)	ResetVoltagesClick(9.14%)	public (100%)
Cluster 2	11	Renderer.cpp (83.21%)	87 (99.11%)	Renderer (99.11%)	Ortho2D (13.41%)	protected virtual (21%)
Cluster 3	46	SyncMotor.cpp (48.34%)	34 (51.45%)	SyncMotor (51.45%)	GetElectricalData (22.11%)	public (38.22%)
Cluster 4	38	Shunt.cpp (68.34%)	23 (100%)	Shunt (100%)	UpdateNodes (19.36%)	public (55%)

After applying the process flow in PSP open-source project which were used with GitAhead project, the correlation matrix has been derived. It shows the shared behaviour among the related classes.

5.2.1. Normalised Coupling Computation

It has been discovered VH degree of correlation or coupling is found in C15, C23, C67, and C84 classes are having a 100% coupling or correlation whereas C14 and C55 have least coupling of 0%. Similarly to this, by utilising the H, M, L, and VL degree of correlation by numerical value between classes, we found the C12 with 100% coupling and C2 has 0% coupling for H degree, C83 and C34 with 100% coupling for M and L respectively. C3 and C97 have VL degree with 100% coupling. Table 18 depicts the result.

Table 18. Class names with their ID and quantitative level [20].

CLASS	Class label	Level
AboutForm	C3	VL
Bus	C12	H
Branch	C15	VH
MainFrameBase	C23	VH
RateLimiter	C34	L
Renderer	C67	VH
Shunt	C83	M
Transformer	C84	VH
VertexBuffer	C97	VL

5.2.2. Class Level Cohesion

To find the class cumulative cohesion or cumulative cohesiveness, we applied same formula as in First application study. The capacitor class has 15 methods with 4 parameter types. As indicated in Table 19, these 4 parameters actual instance into the methods '0' represents parameter not used and '1' signifies parameter used in given method:

Cumulative cohesion=[5+4+2+1]/(4*15)
=12/(4*15)=0.2

Table 19. Count of function's parameters type for class 'capacitor'.

Functions	int	bool	Element *	ElectricalUnit
GetCopy	1	0	0	0
AddParent	0	1	0	0
Draw	1	0	0	0
Capacitor	0	0	0	0
DrawDC	0	0	0	0
Contains	0	0	0	0
Intersects	0	0	0	0
Rotate	0	0	0	0
GetContextMenu	1	0	0	0
GetTipText	0	1	0	0
ShowForm	0	1	1	1
GetElectricalData	1	0	1	0
GetPUElectricalData	1	0	0	0
SetElectricalData	0	0	0	0
SaveElement	0	1	0	0
OpenElement	0	0	0	0
Total Count	**5**	**4**	**2**	**1**

5.3. Third Open-Source Project Study

Jet is a small size open-source project. It consists of 15 classes and 300 functions. After applying clustering techniques, three clusters were found and we considered cluster 1 for our examination on the basis of majority number of records and this cluster includes 137 records out of 185. Similarly, records were found for cluster 2 and 3 which are 33 and 15 respectively. These records show the percentage of frequency with which each clustered class, class ID, file, function, and function type occurs. The class 'Assembler' has the maximum percentage, or 45.21 percent of occurrence, among the additional classes in the identical cluster1, as shown in Table 20. In the same cluster, class ID, file, function, and function type percentage occurrences are 45.21 percent, 93.25 percent, 12.34 percent, and 100 percent, respectively. It also provides information on which functions belong to which classes, their function types, and how many times a function has been applied to a cluster, for example, it also provides information on how many specific functions are included in a given class.

The correlation matrix has been derived after applying the process flow in Jet open-source project which were used with GitAhead project. It shows the shared behaviour among the related classes.

Table 20. Cluster-wise result where inputs are class, class ID, file, function and function type.

Cluster	Records	File	Class ID	Class	Function	Function type
Cluster 1	137	Jet assembler.cpp (93.25%)	6 (45.21%)	Assembler (45.21%)	Write (12.34%)	Public (100%)
Cluster 2	33	Jet-lexer.cpp (88.53%)	13 (91.21%)	Lexer (91.21%)	Peek (23.41%)	Public (19.68%)
Cluster 3	15	Jet-diagnostic.cpp (56.12%)	9 (68.45%)	Diagnostic (68.45%)	Build (34.35%)	Public (45.32%)

5.3.1. Normalised Coupling Computation

It has been discovered VH degree of correlation or coupling is found in C3 and C12 classes are having a 100% coupling or correlation whereas C1 and C5 have least coupling of 0%. Similarly to this, by utilising the H, M, L, and VL degree of correlation by numerical value between classes, we found the C4 with 100% coupling and C11 has 0% coupling for H degree, C10 and C7 with 100% coupling for M and L respectively. C6 have VL degree with 100% coupling. The Table 21 shows the analysied results.

Table 21. Class Names with their ID and quantitative level [20].

CLASS	Class label	Level
Assembler	C3	VH
FileBuf	C4	H
Lexer	C6	VL
Option	C7	L
Parser	C10	M
Token	C12	VH

5.3.2. Class Level Cohesion

To find the class cumulative cohesion or cumulative cohesiveness, we applied same formula as in First application study. The 'Lexer' class has 8 methods with 2 parameter types. As indicated in Table 22, these 2 parameters actual instance into the methods '0' represents parameter not used and '1' signifies parameter used in given method.

Cumulative cohesion=[1+2]/(2*8)
=3/(2*8)=0.1875

Table 22. Count of function's parameters type for class 'Lexer'.

Functions	int	Size_t
lex lex_	1	0
int _float	0	0
lex_str	0	0
lex_ident	0	0
peek	0	1
curr	0	0
next	0	1
skip_new_line	0	0
Total Count	**1**	**2**

6. Discussions

Many academics and researchers have employed certain data mining methods for understanding source code. In this part, a comparative study on several methods is presented.

Comment-Mine a semantic search architecture [16], which extracts knowledge related to software design elements. The implementation and evolution results of method is in the form of a knowledge graph. This helps only to put a basic idea of program comprehension and analyse various comments exist in software.

Towards understanding Third-Party Library (TPL) dependency in the C/C++ ecosystem [31] a tool was developed to identify the dependency patterns of C/C++ projects and construct a comprehensive and precise C/C++ dependency detector.

A system was developed for identifying the identifiers names to code review methods [7]. This work was exclusively based upon naming practices.

A matrix developed by Husein and Oxley [8] and Liang *et al.* [13] depicts the interdependency of the elements. The elements are at the function or file level, and calling connection between the file and the functions depicted in a table. Model Predictive Control (MPC) explained in their work. Between clusters connections to other subsystem parts: Internal coupling is computed in our study based on class correlations to other classes inside a particular cluster at the cluster level. Similar to this, we computed coupling at inter and intra class level, through figuring out how the common methods in a class are connected to one another.

System strength is the ratio of a subsystem's internal cohesion to that subsystem's exterior connection to all other subsystems. This enables one to priorities or focus on a specific subsystem when rebuilding the system.

Mining Program Source (MMS)-apriori association rules have been implemented [11]. To demonstrate associations between various members, methods, method's parameters at inter-cluster and ultra-cluster associations.

The total of all clusters' internal edges ought to be greater than the sum of their outward edges [24]. To improve cohesiveness and reduce coupling both inside and across clusters of modules, or intra cluster and inter-cluster, they have developed an optimisation approach based on evolutionary algorithms. As a criteria for evaluation, they computed cohesiveness and coupling based on modularization quality attribute. We computed class-level NCCI, CCI, and cohesiveness in our work.

Using source code, framework of knowledge attainment is devised [9]. In order to facilitate partial-automated program understanding, maintenance easier and to offer beneficial perceptions of framework features. Their technique is evaluated using a case study from the industry. We have also extracted valuable data for maintenance using clustering and association.

Table 23 illustrates the superiority of our suggested work above other approaches found in the literature. This table provides a comparative perspective between our suggested method and other existing methods used to understand C++ code. The table presents results relating to Code Comprehension along with relevant observations about current practices and our suggested approach. The table's remark column demonstrates how our suggested work is superior to other current works that have been highlighted in the literature.

Following are the main characteristics (strengths) of our work:

As far as we are aware, no one has created correlation matrices for the two types of data using the common approaches used by the classes.

Additionally, the computation of the normalised and cumulative coupling indices is unusual.

The key distinguishing characteristics in our work are data mining techniques used for code comprehension and software code dependencies are derived in the form of a matrix of correlation among the classes by considering the common functions. On this basis, we have devised normalised cumulative coupling in terms of VH, H, M, L, and VL. Apart from this we have included the cumulative cohesion in consideration to function parameters. The most significant class of methods deriving from the clustering method's parameter types has been computed cumulative cohesion.

Table 23. Comparative analysis of different S/w code comprehension methodology.

SN	Work Descriptions	Data Mining Methodology	Code Comprehension related Result	Remarks
1	Comment-mine—a semantic search approach to program comprehension from code comments [16].	Knowledge graph.	Knowledge representation based on comments to aid program comprehension.	• No datamining method is proposed for identifying interdependency of code elements. • No computational model is proposed for measuring the cohesion or coupling. • No methods to determine size of each module.
2	Towards understanding TPL dependency in C/C++ ecosystem [31].	No standard data mining method is mentioned.	In order to comprehend TPL dependencies within the C/C++ ecosystem, the writers compile the TPL databases, package management, and dependency detection tools that are currently available, as well as explain the dependence patterns found in C/C++ projects.	• No datamining method is proposed for identifying interdependency of code elements. • No computational model is proposed for measuring the cohesion or coupling. • No methods to determine size of each module.
3	Naming practices in object-oriented programming: an empirical study [7].	No data mining method is mentioned.	Studies on naming identifiers demonstrate the importance of informative names in enhancing program readability and maintainability.	• No datamining method is proposed for identifying interdependency of code elements • No computational model is proposed for measuring the cohesion or coupling. • No methods to determine size of each module.
4	Applications of clustering techniques to software portioning, recovery, and restructuring [13].	The clustering techniques adopted in this work are based on agglomerative hierarchical approaches.	A matrix is developed for interdependency of the function or file level, and calling connection between the file and the functions.	• No model is proposed for measuring the cohesion or coupling. • No methods to determine size of each module.
5	An improved methodology on information distillation by MMS code [11].	Clustering and association rules mining.	Identifies hidden relationships between classes, methods, and member data.	• No computational model is proposed for measuring the cohesion or coupling. • No methods to determine size of each module.
6	Software module clustering as a multi-objective search problem [24].	Clustering techniques.	Identifies highest cohesion and the lowest coupling module.	• No computational model is proposed for measuring the cohesion or coupling. • No methods to determine size of each module.
7	Mining source code elements for comprehending object-oriented systems and evaluating their maintainability [9].	Clustering techniques .	Identifies interdependency of the function or file level.	• No computational model is proposed for measuring the cohesion or coupling. • No methods to determine size of each module.
8	Improvised software code comprehension using data mining. (our proposed work)	Classification, association, and clustering.	Identifies cohesion and the coupling in module with logical and deterministic value.	• Analyse each data mining techniques for each element of software project (class, function, etc.). • Computational models are proposed for measuring the qualitative and quantitative level of cohesion or coupling. • Logical method is proposed to determine qualitative size of each module.

7. Conclusions

We have used a variety of technologies in our work to extract information and create knowledge bases that can be used to maintain software or to help with other tasks.

Initially the class ID, class, method, its type and file are used for the purpose of clustering using K-means method. Using the web representation technique, we did class and file level study to display the association between classes and the file. Using a cluster diagram to display classes and methods, we discovered that the "LexState" class has the most methods (150), accounting for 5.16 percent of all the methods in the software.

By using the Apriori method, we were able to determine how a class and the file specified in Section 4 are related to one another. This study clarifies the importance of a function in a file evident. The data mining technique C5.0 allows to view all methods connected to a particular class. The number of rules that link a method's instances and confidence factor to a certain class may also be obtained by utilising the graphical style of representation.

We generate a quantitative and qualitative matrix from the aforementioned study to identify coupling and cohesiveness at various levels of abstraction. On the basis of standard techniques, we have determined the quantitative level. The coupling between various classes before giving various levels: VL, L, M, H, and VH to a range of numerical values. We utilised this level range for a filtered matrix. Then, as indicated in Tables 9 to 14, we arrive at the five matrices that simply reflect the coupling between classes at various levels: VL, L, M, H, and VH.

The average coupling normalised index was calculated by multiplying the number of levels in the row by 0.8 and assigning each level a numerical value, such as VH equates to 0.8 in Table 9 of VH. The cumulative coupling column is provided as a consequence. We derive the normalised coupling using the usual normalisation formula to produce NCCI. This provides us with an index that ranges from 0 to 100%, which we can use to determine the classes with the highest and lowest NCCI. We have determined a class's cohesiveness, or the methods inside a class that take the same sorts of arguments.

This technique can be used in the future to identify cohesion and coupling while performing adaptive maintenance that takes into account local or global variables. It is possible to calculate weighted cohesiveness by assigning a value to each class member. In adaptive maintenance, these findings can be completely applied, if a modification in one place affects other resources of the source-code.

References

[1] Anquetil N. and Lethbridge T., “Experiments with Clustering as a Software Remodularization Method,” *in Proceedings of the 6th Working Conference on Reverse Engineering*, Atlanta, pp. 235-255, 1999. DOI:10.1109/WCRE.1999.806964

[2] Balmas F., Wertz H., and Singer J., “Understanding Program Understanding,” *in Proceedings of the 8th International Workshop Program Comprehension*, Washington (DC), pp. 256, 2000. https://dl.acm.org/doi/10.5555/518049.856959

[3] Chen K., Tjortjis C., and Layzell P., “A Method for Legacy Systems Maintenance by Mining Data Extracted from Source Code,” *in Proceedings of the IEEE 6th European Conference Software Maintenance and Reengineering*, Washington (DC), pp. 54-60, 2002. https://www.ihu.edu.gr/tjortjis/publications.htm

[4] Eisenbarth T., Koschke R., and Simon D., “Locating Features in Source Code,” *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 210-224, 2003. DOI:10.1109/TSE.2003.1183929

[5] Fayyad U., Piatetsky-Shapiro G., and Smyth P., “From Data Mining to Knowledge Discovery: An Overview,” *Advances in Knowledge Discovery and Data Mining*, pp. 1-34, 1996. https://dl.acm.org/doi/10.5555/257938.257942

[6] GitHub, The-NextGen-Project/jet, https://github.com/The-NextGen-Project/jet, Last Visited, 2024.

[7] Gresta R., Durelli V., and Cirilo E., “Naming Practices in Object-Oriented Programming: An Empirical Study,” *Journal of Software Engineering Research and Development*, vol. 11, no. 1, pp. 1-16, 2023. https://doi.org/10.5753/jserd.2023.2582

[8] Husein S. and Oxley A., “A Coupling and Cohesion Metrics Suite for Object-Oriented Software,” *in Proceedings of the International Conference on Computer Technology and Development*, Kota Kinabalu, pp. 421-425, 2009. DOI:10.1109/ICCTD.2009.209

[9] Kanellopoulos Y., Dimopulos T., Tjortjis C., and Makris C., “Mining Source Code Elements for Comprehending Object-Oriented Systems and Evaluating their Maintainability,” *SIGKDD Explorations*, vol. 8, no. 1, pp. 33-40, 2006. https://doi.org/10.1145/1147234.1147240

[10] Kanellopoulos Y. and Tjortjis C., “Data Mining Source Code to Facilitate Comprehension: Experiments on Clustering Data Retrieved from C++ Program,” *in Proceedings of the 12th IEEE International Workshop on Program Comprehension*, Bari, pp. 214-223, 2004. DOI:10.1109/WPC.2004.1311063

[11] Kanellopoulos Y., Makris C., and Tjortjis C., “An Improved Methodology on Information Distillation by Mining Program Source Code,” *Data and Knowledge Engineering*, vol. 61, no. 2, pp. 359-383, 2007. https://doi.org/10.1016/j.datak.2006.06.002

[12] Kunz T. and Black J., “Using Automatic Process Clustering for Design Recovery and Distributed Debugging,” *IEEE Transactions on Software Engineering*, vol. 21, no. 6, pp. 515-527, 1995. DOI:10.1109/32.391378

[13] Liang X., Xue C., and Huang M., “Improved Apriori Algorithm for Mining Association Rules of Many Diseases,” *in Proceedings of the 5th International Symposium, ISICA*, Wuhan, pp. 272-279, 2010. https://link.springer.com/chapter/10.1007/978-3-642-16388-3_30

[14] Lung C., Zaman M., and Nandi A., “Applications of Clustering Techniques to Software Portioning, Recovery and Restructuring,” *The Journal of Systems and Software*, vol. 73, no. 2, pp. 227-244,

2004. https://doi.org/10.1016/S0164-1212(03)00234-6

[15] Maione C., Nelson D., and Barbosa R., "Research on Social Data by Means of Cluster Analysis," *Applied Computing and Informatics*, vol. 15, no. 2, pp. 153-162, 2019. https://doi.org/10.1016/j.aci.2018.02.003

[16] Majumdar S., Papdeja S., Das P., and Ghosh S., *Advanced Computing and Systems for Security*, Springer, 2020. https://link.springer.com/chapter/10.1007/978-981-15-2930-6_3

[17] Mancoridis S., Mitchell B., Chen Y., and Gansner E., "Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures," *in Proceedings of the IEEE International Conference on Software Maintenance for Business Change*, Oxford, pp. 50-59, 1998. DOI:10.1109/ICSM.1999.792498

[18] Maqbool O., Babri H., Karim A., and Sarwar M., "Metarule-Guided Association Rule Mining for Program Understanding," *IEE Proceedings-Software*, vol. 152, no. 6, pp. 281-296, 2005. DOI:10.1049/ip-sen:20050012

[19] Mayrhauser A., Vans A., and Howe A., "Program Understanding Behaviour during Enhancement of Large-Scale Software," *Journal of Software Maintenance: Research and Practice*, vol. 9, no. 5, pp. 299-327, 1997. https://doi.org/10.1002/(SICI)1096-908X(199709/10)9:5<299::AID-SMR157>3.0.CO;2-S

[20] Mazumdar B. and Mishra R., "Customer Orientation Based Multi-Agent Negotiation for B2C e-Commerce," *International Journal of Agent Technologies and Systems*, vol. 2, no. 2, pp. 24-48, 2010. https://www.igi-global.com/article/customer-orientation-based-multi-agent/43867

[21] Moreira G. and Santos J., "Applying Coupling and Cohesion Concepts in Object-Oriented Software: A Controlled Experiment," *in Proceedings of the 19th Brazilian Symposium on Software Quality*, Sao Luis, pp. 1-10, 2020. https://doi.org/10.1145/3439961.3439969

[22] Offutt J., Abdurazik A., and Schach S., "Quantitatively Measuring Object-Oriented Couplings," *Software Quality Journal*, vol. 16, no. 4, pp. 489-512, 2008. https://link.springer.com/article/10.1007/s11219-008-9051-x

[23] Oliveira T., Thales1330/PSP, https://github.com/Thales1330/PSP/tree/master, Last Visited, 2024.

[24] Praditwong K., Harman M., and Yao X., "Software Module Clustering as a Multi-Objective Search Problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264-282, 2011. DOI:10.1109/TSE.2010.26

[25] Rathee A. and Chhabra J., "Improving Cohesion of Software System by Performing Usage Pattern Based Clustering," *in Proceedings of 6th International Conference on Smart Computing and Communication*, Kurukshetra, pp. 740-746, 2018. https://doi.org/10.1016/j.procs.2017.12.095

[26] Saeed M., Maqbool O., Babri H., Hassan S., and Sarwar S., "Software Clustering Techniques and the Use of Combined Algorithm," *in Proceedings of the 7th European Conference on Software Maintenance and Reengineering*, Benevento, pp. 301-306, 2003. DOI:10.1109/CSMR.2003.1192438

[27] Shirabad J., Lethbridge T., and Matwin S., "Mining the Maintenance History of Legacy Software System," *in Proceedings of the International Conference on Software Maintenance*, Amsterdam, pp. 95-104, 2003. DOI:10.1109/ICSM.2003.1235410

[28] Standish T., "An Essay on Software Reuse," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 494-497, 1984. DOI:10.1109/TSE.1984.5010272

[29] Sun J. and Ling B., "Software Module Clustering Algorithm Using Probability Selection," *Wuhan University Journal of Natural Sciences*, vol. 23, no. 2, pp. 93-102, 2018. https://link.springer.com/article/10.1007/s11859-018-1299-9

[30] Supriyamenon M. and Rajarajeswari P., "A Review on Association Rule Mining Techniques with Respect to their Privacy Preserving Capabilities," *International Journal of Applied Engineering Research*, vol. 12, no. 24, pp. 15484-15488, 2017. https://www.ripublication.com/ijaer17/ijaerv12n24_216.pdf

[31] Tang W., Xu Z., Liu C., Wu J., Yang S., Li Y., and Liu Y., "Towards Understanding Third-Party Library Dependency in C/C++ Ecosystem," *in Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, Michigan, pp. 1-12, 2022. https://doi.org/10.1145/3551349.3560432

[32] Tiarks R., "What Programmers Really Do: An Observational Study," *Softwaretechnik-Trends*, vol. 31, no. 2, pp. 36-37, 2011. https://api.semanticscholar.org/CorpusID:17226304

[33] Understand by SciTools, http://www.scitools.com/, Last Visited, 2024.

[34] Wedyan F. and Abufakher S., "Impact of Design Patterns on Software Quality: A Systematic Literature Review," *IET Software*, vol. 14, no. 1, 1-17, 2020. https://doi.org/10.1049/iet-sen.2018.5446

[35] Xiao C. and Tzerpos V., "Software Clustering Based on Dynamic Dependencies," *in Proceedings of the 9th European Conference on Software Maintenance and Reengineering*, Manchester, pp. 124-133, 2005. DOI:10.1109/CSMR.2005.49

[36] Yadav V., Singh R., and Yadav V., "Estimation Model for Enhanced Predictive Object Point Metric in OO Software Size Estimation Using Deep Learning," *The International Arab Journal of Information Technology*, vol. 20, no. 3, pp. 293-302, 2023. https://doi.org/10.34028/iajit/20/3/1

[37] Ying A., Murphy G., Ng R., and Chu-Carroll M., "Predicting Source Code Changes by Mining Change History," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574-586, 2004. DOI:10.1109/TSE.2004.52

[38] Zhang M., Hall T., and Baddoo N., "Code Bad Smells: A Review of Current Knowledge," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 3, pp. 179-202, 2011. https://doi.org/10.1002/smr.521

Ram Gopal Gupta research scholar, Department of Computer Science and Engineering, VMSB Uttarakhand Technical University, Dehradun, Uttarakhand-India.

Ankur Dumka associate professor, Department of Computer Science and Engineering, Women Institute of Technology, Dehradun, Uttarakhand-India.

Bireshwar Dass Mazumdar associate professor, School of Computer Science Engineering and Technology (SCSET), Bennett University, Greater Noida, Uttar Pradesh-India.