







Research Article

A Novel Approach to Automate Complex Software Modularization Using a Fact Extraction System

Muhammad Zakir Khan ¹, Rashid Naseem ², Aamir Anwar ³, Ijaz Ul Haq,⁴
Ahmad Alturki ⁵, Syed Sajid Ullah ⁶, and Suheer A. Al-Hadhrami ⁷

¹James Watt School of Engineering, University of Glasgow, G12 8QQ Glasgow, UK

²Department of Computer Science, Pak Austria Fachhochschule Institute of Applied Sciences and Technology, Haripur, Pakistan

³School of Computing and Engineering, University of West London, London W5 5RF, UK

⁴Faculty of Education, Psychology and Social Work, University of Lleida, 25003 Lleida, Spain

⁵STC's Artificial Intelligence Chair, Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

⁶Department of Information and Communication Technology, University of Agder, Norway

⁷Department of Computer Engineering, College of Engineering, Hadhramout University, Hadhramout, Al Mukalla, Yemen

Correspondence should be addressed to Ahmad Alturki; ahmalturki@ksu.edu.sa, Syed Sajid Ullah; syed.s.ullah@uia.no, and Suheer A. Al-Hadhrami; s.alhadhrami@hu.edu.ye

Received 6 December 2021; Revised 25 January 2022; Accepted 7 February 2022; Published 30 March 2022

Academic Editor: Naeem Jan

Copyright © 2022 Muhammad Zakir Khan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Complex software systems that support organizations are updated regularly, which can erode system architectures. Moreover, documentation is rarely synchronized with the changes to the software system. This creates a slew of issues for future software maintenance. To this goal, information extraction tools use exact approaches to extract entities and their corresponding relationships from source code. Such exact approaches extract all features, including those that are less prominent and may not be significant for modularization. In order to resolve the issue, this work proposes an enhanced approximate information extraction approach, namely, fact extractor system for Java applications (FESJA) that aims to automate software modularization using a fact extraction system. The proposed FESJA technique extracts all the entities along with their corresponding more dominant formal and informal relationships from a Java source code. Results demonstrate the improved performance of FESJA, by extracting 74 (classes), 43 (interfaces), and 31 (enumeration), in comparison with eminent information extraction techniques.

1. Introduction

Software systems are essential in our daily lives, businesses, and governmental organizations, and they require an updated software system to meet their functional and nonfunctional requirements. Client requests or changes in the system's environment may cause changes in requirements [1–3]. Changes deteriorate the architecture of software systems, making it difficult to maintain them. In such situations, the software system must be designed in such a way that the negative effects of modifications to the software system are kept to a minimum. Updated documentation is

required for updated software. If the documentation is outdated, the software systems need to be retired or replaced.

Reverse engineering is the first step in re-engineering, and it involves understanding the system and acquiring the necessary information for software system maintenance [4, 5]. Information can be extracted from a software system through documentation, compiled code, development team members, and source code [6]. The most reliable source of information for restoring software architecture has been observed to be source code. The reason is the most recent version of the software, and this source code will be built and eventually run, the information gathered from it is the most reliable.

Information extracting from the source code of a software system, on the other hand, is a challenging task since no developer can fully know the code of a large and complex system. As a result, we need tools to automatically extract information from source code, which will help in the recovery of software architecture and the comprehension of software systems [7]. The understanding and recovery of architecture are crucial for software maintenance and evolution.

The first step in modularizing a software system is to analyze it since source code is the primary source of information for extracting artifacts. The need for a software modularization technique that transforms low-level artifacts (source code) into abstract views (high-level) [1, 5]. Two approaches can be utilized for code analysis: an exact approach and an approximate one. The exact approach utilized the parser to analyze the whole source code, whilst the approximate approach utilized it to extract the specified parts of the information. We propose a methodology for evaluating Java source code in order to discover entities and their relationships in a Java software system in this paper. The selection of an object-oriented system since it is a more realistic approach to software development. Object-oriented software systems developed in the 1990s are now legacy systems with an unstable structure due to changes made to them [5, 7]. Their documentation is either non-existent or obsolete. A comprehensive understanding of these software systems is necessary for future updates and maintenance.

The proposed system is named "Fact Extractor System for Java Application," or FESJA, which gives an approximate approach to automatic Java application software modularization. Entities such as classes, interfaces, and enumerations and their relationships (formal and informal) can be extracted using the tool. We utilized FESJA to extract formal relationships from classes in several Java software systems; interfaces and informal relationships are 74 and 43, respectively. We have extracted 31 relationships to analyze, which are divided into seven categories: folder-based, implements-based, composition-based, file-based, generality based, and router-based relationships. To the list of contributions, the following items could be added:

- (1) A framework for extraction of relationships
- (2) Introduction of enumeration is an entity
- (3) Introduction of additional formal and informal relationships

The organization of the paper is composed of the following sections. Section 2 focuses on related work. The source code entities and relationships are described in Section 3. The research methodology for the fact extraction system is described in Section 4. Section 5 discusses the experimental setup. Section 6 presents the results. Section 7 contains the outcome of the experiment as well as a discussion.

2. Related Work

In the literature, several fact extractor systems for extracting features from source code have been proposed. To

modularize software systems, Raimond and Lovesum [8] followed Anquetil and Lethbridge employed formal and informal features. They used files, routines, classes, and processes as entities. User-defined data types, procedure calls, file inclusion, macro use, and global variables are some of the formal features mentioned. Identifiers and comments are examples of nonformal features. They focused on how to modularize structured language software systems using hierarchical clustering algorithms. They came to the conclusion that identifiers provide good design results that the file has a good expert comparison, and those comments have a good expert comparison, but that bad design results and routine calls perform poorly. The authors in [9–12] proposed a method for improving the accuracy of autonomous software architecture reconstruction. The method uses a combination of graph clustering and partitioning. They considered classes as distinct entities and created eleven relationships between them. The following relationships were discovered: inheritance, implements an interface, members (A has at least one member of type B), method parameter (A has at least one method with a parameter of type B), local variable (A has a local variable of type B in a method), returned type (A has a method that returns type B), field access (A directly accesses at least one field of type B), static method call (A calls a static method of B), object instantiation (an object of type B is instant (inside A a cast to B is done). Their focus was on object-oriented systems (java).

Hussain et al. [13] adopted an agglomerative approach for clustering of structured software systems. They considered functions as entities. They took advantage of formal features like entity calling functions, global variables that entities refer to, and user-defined types that entities can access. Aghdasifam et al. [14] carried out work of software modularization using agglomerative hierarchical clustering algorithms. They targeted the structured system. They used functions as an entity. In their work, they utilized use global variables, user-specified data types, and function calls. Based on the results of the experiments, they determined that the type of feature outperforms the global and call features. Zahoor et al. [15] presented a tool called the WAFE tool (Web Application Fact Extractor) which extract features from web application. Similarly, a similar work presented by [16, 17], automate extraction dependency between web component and database resources in java web applications. The Web Page contains classes. The WAFE tool extracted the following information: a class called from a web page, a class function called from a web page, web page form submits on another web page or the same web page, web page link to another web page, web page redirect to another web page, and web page folder or directory. Custom Code Files are included in Web Pages, and Custom Code File functions are called by Web Pages. Web Pages' classes are derived from these classes. Classes derived from Web Page classes, as well as functions of classes derived from Web page classes.

Shah et al. [1] followed Abdul Qudus Abbasi's research work carried out a detailed study about features between entities. They developed a Fact Extractor that could extract twenty-six features from the source code of object-oriented

systems to extract the features. Classes, structs, unions, files, folders, global functions, global variables, and macros were all considered entities by them. Class to class relationships based on inheritance, class to class relationships based on containment, class to class relationships based on genericity, class to class relationships based on member access, class to class relationships based on source files and folders, class to class relationships based on a friend, class to global functions or data relationships or macro, and global function to global function are among relationships extracted by fact extractor. An experimental evaluation of relationships for the modularization of object-oriented software systems was reported by Thakur et al. [18]. Using Abbasi's Fact Extractor [13], they retrieved the twenty-six features. Direct and indirect relationships were the two types of relationships. They also found that indirect relationships give better modularization results than direct relationships based on the experimental data.

Aljarah et al. [19] improved on the work of Tzerpos and Andritso, who developed the LIMBO algorithm for software architecture recovery. They combined structural and non-structural features to determine the usefulness of non-structural features to the reverse engineering process. Developers' names, directory paths, lines of code, and times of the last update were among the nonstructural features they examined at. The experiments revealed that directory structure and ownership information, but not lines of code, are important factors in software clustering. They also concluded that weighing schemes are useful in breaking down software systems.

Krishnan preferred Koschke's Ph.D. thesis [20] to introduce a classification of component recovery techniques. They used several features for architectural component recovery of structured systems. The features are function calls, set (subprogram sets the value of a global variable), use (subprogram uses the value of object T), take-address-of (subprogram takes the address of object T), function parameter (subprogram has a formal parameter of user-defined data type), return (subprogram returns a value of user-defined type), local-obj-of-type (subprogram has a local object of user-defined type), actual-parameter of (object is an actual parameter in a call to subprogram), of type (S is of type T, where S is an object and T is user-defined type), same-expression (S and T occur in the same expression where S and T are objects) and part-type (S is a part type of T where both S and T are user-defined types).

Richner and Ducasse proposed an environment for generating high-level views of object-oriented systems from both static and dynamic information, and Alshuqayran et al. [21] followed suit. For modularization of a software system, they used composition, inheritance, invocation (method of sender invokes received method on one of the candidates), access (an attribute of class 1 is accessed by the method of class 2), and method (a class defines a method that belongs to another class) as well as dynamic features.

Aljarah et al. [19] proposed MULICsoft, a software clustering algorithm. For the modularization of software systems, they exploited both static and dynamic features. Source files are the objects to be clustered. Procedure calls

and variable references are static features, but dynamic features on a software system are the results of profiling the system's execution, indicating how many times each file called procedures in other files during the run time. In 2003, Trifu [22] proposed a technique that combines clustering with pattern-matching, to automatically recover subsystem decompositions. For the clustering process, they used inheritance, association, aggregation, call, and access features. They also proposed a method for assigning weights to these relationships.

Rathee and Chhabra [23] used a combination of static and dynamic features to modularize java software systems. They used inheritance, implementation, containment, calls to methods, and access to variables and assignment relationships along with dynamic features for the software modularization process. Eski and Buzluca [24] presented a comprehensive comparative study of six software clustering algorithms. They developed a lightweight C/C++ source code extractor called CTSX. CTSX is built on CTags and CScope. CTSX uses CTags to extract program entities (functions, variables, and data types) and CScope to retrieve features (function calls).

Teymourian et al. [25] presented an approach for the evaluation of dynamic clustering. They used both static and dynamic features for the modularization process. For feature extraction, they used the CPPX fact extractor system. From the experimental results, they concluded that static features perform better than dynamic features. Rafi et al. [26] introduced a systematic study to categorize the critical challenges associated with best practices of software implementation for organizations and Akbar et al. [27] discussed the challenges associated with successful execution of outsource software development. Using a combined algorithm, Alanazi [28] focused on clustering software systems. Functions were considered as entities in their study, and the following features were used: function call (functions called by an entity), global variables (global variables referred to by an entity), and user-defined types (user-defined types accessed by an entity). Tjortjis [29] proposed a method for mining association rules from code in order to capture program structure and achieve a better understanding of the system. To classify code chunks as entities, they used the following characteristics: Code blocks, variables, and data types are all entitled.

Yadav et al. [30] proposed an approach for analyzing Java code. They analyzed programs and built tables using a Java code analyzer. They also used a clustering engine, which works with such tables and finds relationships between code elements. They considered files, packages, classes, methods, and parameters as entities. The relationships they used in their study include entities ID, entities Name, imported packages, inheritance, implements relationship, arguments, return value, modifier, parameter type, and parameter used.

Rathee and Chhabra [6, 23] followed Tonella presented an approach of using concept analysis for module reconstruction. Accesses to global variables, dynamic location accesses, the presence of a user-defined structured type in the signature, and the presence of a user-defined structured

type in the return type are the relationships he used for module reconstruction.

3. Source Code Entities and Relationship

This section focused on entity and relationship relationships in source code. These source code entities and relationships have been created based on Java source code entities and relationships.

3.1. Entities in Java Source Code. Entities are the smallest significant elements at the architectural level [20]. They are part of the clustering process and become members of clusters during the automated software clustering and modularization process [1]. The proposed “FESJA” extracted three types of entities, these types are classes, interfaces, and enumerations.

A class and an interface can be an entity in object-oriented systems, it has been used widely in software architecture reconstruction and recovery [1, 4]. The approach used by [9] helps to create basic, fully automated tools that can help detect the core classes of a software system based on its code. The study done by [22] used a Model-Driven Engineering technique to provide support for Micro Service Architecture Recovery (MiSAR). In their work, they described an empirical study that uses manual analysis on eight microservice-based open-source projects to identify the core elements of the approach. The research helps software developers and maintainers in taking steps at the design level to create maintainable object-oriented software with classes [22, 23, 31]. “An enum type is a form of data that allows a variable to be a set of specified constants.” Enums are java reference types, much as classes. We can add methods, variables, and constructors to an enum in the same way that we would in a regular Java class of Java beat version issued in 2013.

3.2. Relationships. Similarities between entities are always calculated during the modularization process based on their connections. It establishes links between the application’s entities. However, the first step is to analyze feature extraction; afterward when, we applied clustering to group entities with similar features or attributes [15]. The relationship types extracted by the proposed fact extractor system, a java-based system, have been identified. Static, dynamic, informal, direct, and indirect relationships are examples of these types of relationships.

4. The Research Methodology of Fact Extraction System

The proposed methodology uses low-level artifacts (source code) to build high-level (abstract views) of the software system in the form of a Java-based Fact Extractor System. Extraction of features is the first stage in modularization, and FESJA has utilized an API named *java.util.regex* to search for necessary parts (approximated approach). Regular expressions were utilized to match patterns in this

Java API. FESJA has extracted three categories of entities, as follows:

```
\\bclass[\\s][\\s]*[_[a-z[A-Z]]][\\w]*[\\s\\{<
```

The regular expression above can be used to extract a class `myClass{, class myClass {, class myClass<`.

```
\\binterface[\\s][\\s]*[_[a-z[A-Z]]][\\w]*[\\s\\{<
```

For the extraction of interfaces, the below regular expression is used.

The regular expression above, for example, can retrieve interface `myInt {, interface myInt{, interface myInt<`.

```
\\benum[\\s][\\s]*[_[a-z[A-Z]]][\\w]*
```

The enumerations are extracted using a regular expression.

The regular expression above, for example, can extract the enum `myEnum`.

The Process of fact extraction in FESJA starts with some data being uploaded, folders being extracted, and folder names that are alike being removed. After the extraction process of FESJA, the system check whether the source folder (src) exists or not. The process will end if there is no src folder. The same entity names are removed if the src file exists. Figure 1 shows the whole process of the fact extraction system, whereas Figure 2 shows entities with relationships.

5. Experimental Setup

The experiments are conducted using data sets to evaluate the relationships. These datasets are being designed and implemented using java (object-oriented methodology). JFree Chart (an open-source library for graphs and charts), JUnit (an open-source java unit testing framework), and Weka (an open-source java testing framework) compensate our dataset (Machine learning algorithm for data mining tasks). All these datasets are taken from Github and the source (<https://www.grepcode.com>). Entities identified by the FESJA tool in the above systems are shown below in Tables 1–4

5.1. Comparative Analysis of Intradataset. This study explores the results of multiple versions of the same data set.

5.1.1. iText. The statistics of the iText software system are shown in Tables 5–9. It has been concluded from the statistics that:

- (i) Class is the most important entity in the iText software system, and enumerations are not utilized at all
- (ii) Folder-based relationships, composition-based relationships, and access-based relationships are the most common relations for classes
- (iii) In comparison to file-based and access-based relationships, folder-based relationships occur frequently in interfaces

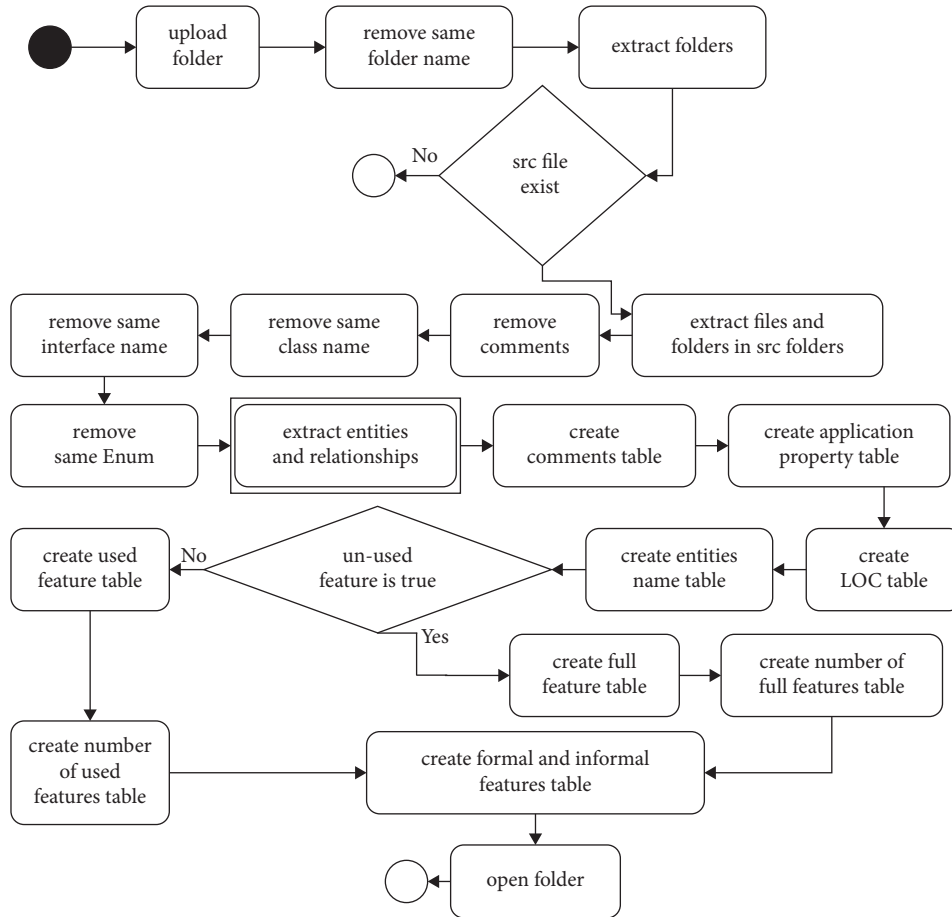


FIGURE 1: Process of fact extraction.

- (iv) Generic or outer implement relationships are not used in the iText software system
- (v) Figures 3 and 4 illustrate those formal relationships in iText are based on formal relationships of classes, while formal relationships of interfaces contribute to just 3% of formal relationships

Figures 3–7 provide graphical representations of iText software system statistics.

5.1.2. *JFreeChart*. Tables 10–14 summarizes statistics of the JFreeChart software system. From the statistics, it has been observed that.

- (i) Like iText the most dominant entity in the JFreeChart software system is class while enumerations are not used at all.
- (ii) The dominant relationships for classes are folder-based relationships, composition-based relationships, and access-based relationships while implements-based relationships and inheritance-based relationships have also good frequency.
- (iii) For interfaces the used relationships are folder-based, file-based, extend based and access-based

relationships while other relationships are infrequent. Also, among used relationships, folder-based relationships have the highest frequency.

- (iv) Same as iText generics-based relationships and outer implements-based relationships are not used in the JFreeChart software system.
- (v) By comparing Figures 8 and 9 it can be calculated that in JFreeChart 92% of formal relationships are based on formal relationships of classes while 8% of formal relationships are based on formal relationships of interfaces.

Figures 8–12 provides a graphical representation of statistics of the JFreeChart software system.

Due to the absence of enumerations, no relationships for enumeration exist in the JFreeChart software system.

5.1.3. *JUnit*. Tables 15–20 summarizes the statistics of a JUnit software system. From the statistics, it has been observed that.

- (i) In the JUnit software system enumerations are introduced in all versions except JUnit 4.8 but have a very low frequency of occurrence.

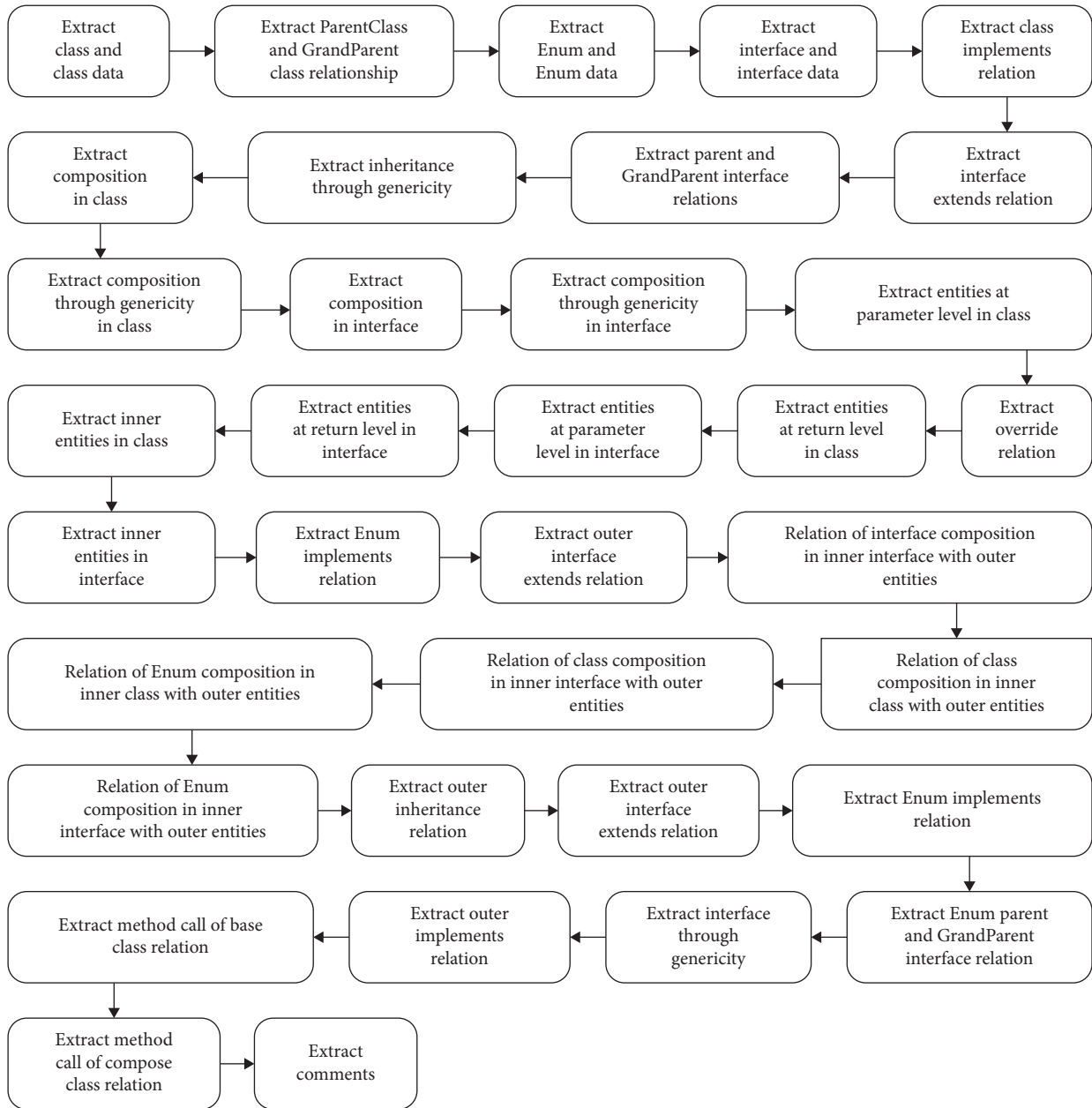


FIGURE 2: Process of Relationship extraction.

TABLE 1: Facts about entities in iText dataset.

Entity related information	iText 1.3	iText 1.4	iText 1.4.8	iText 2.1.7
Total no. of folders	29	32	34	69
Total no. of files	399	470	502	549
Total no. of entities	462	545	586	669
Total no. of classes	434	513	551	625
Total no. of interfaces	28	32	35	44
Total no. of enumerations	0	0	0	0
Total no. of functions in application	5330	6195	6517	7029
Total no. of functions in class	5258	6117	6431	6905
Total no. of functions in interface	72	78	86	124
Total no. of functions in enum	0	0	0	0
sLOC	76100	86858	90526	94073
Blank lines	12283	13802	14655	14811
LOC	88383	100660	105181	108884
Size in bytes	4938428	5704081	6103967	6696336

TABLE 2: Facts about entities in JFreeChart dataset.

Entity related information	JFreeChart 1.0.4	JFreeChart 1.0.7	JFreeChart 1.0.9	JFreeChart 1.0.14
Total no. of folders	61	39	39	40
Total no. of files	500	538	538	594
Total no. of entities	516	559	559	616
Total no. of classes	432	468	468	510
Total no. of interfaces	84	91	91	106
Total no. of enumerations	0	0	0	0
Total no. of functions in application	6705	7468	7522	8629
Total no. of functions in class	6209	6952	7006	8086
Total no. of functions in interface	496	516	516	561
Total no. of functions in enum	0	0	0	0
sLOC	73148	81003	81490	93430
Blank lines	16149	17615	17670	19332
LOC	89297	98618	99160	112762
Size in bytes	602995	6722638	6774669	7700413

TABLE 3: Facts about entities in Junit dataset.

Entity related information	Junit 4.8	Junit 4.9	Junit 4.10	Junit 4.11
Total no. of folders	33	31	31	31
Total no. of files	175	160	162	164
Total no. of entities	180	165	168	171
Total no. of classes	148	134	137	137
Total no. of interfaces	32	30	30	32
Total no. of enumerations	0	1	1	2
Total no. of functions in application	1079	1013	1036	1143
Total no. of functions in class	1039	972	995	1099
Total no. of functions in interface	40	32	32	33
Total no. of functions in enum	0	9	9	11
sLOC	6727	6429	6579	7439
Blank lines	1466	1348	1381	1603
LOC	8193	7777	7960	9042
Size in bytes	340192	330167	338458	429090

TABLE 4: Facts about entities in weka dataset.

Entity related information	Weka-dev 3.7.5	Weka-dev 3.7.6	Weka-dev 3.7.7	Weka-stable 3.6.6
Total no. of folders	87	88	89	110
Total no. of files	1005	1020	1025	1050
Total no. of entities	1352	1378	1384	1379
Total no. of classes	1162	1185	1190	1236
Total no. of interfaces	149	151	152	119
Total no. of enumerations	41	42	42	24
Total no. of functions in application	17114	17501	17618	18746
Total no. of functions in class	16513	16896	17006	18273
Total no. of functions in interface	427	431	438	336
Total no. of functions in enum	174	174	174	137
sLOC	232297	237797	240029	255774
Blank lines	54985	56319	56633	61449
LOC	287282	294116	296662	317223
Size in bytes	13586516	13828049	13940620	15213130

TABLE 5: Entities in iText.

	iText 1.3	iText 1.4	iText 1.4.8	iText 2.1.7
Classes	434	513	551	625
Interfaces	28	32	35	44
Enumerations	0	0	0	0

TABLE 6: Formal and informal relationship in iText.

	iText 1.3	iText 1.4	iText 1.4.8	iText 2.1.7
Informal	43107	49474	52488	58601
Formal	7080	8587	9125	11245

TABLE 7: Formal relationship in iText.

Formal relationships	iText 1.3	iText 1.4	iText 1.4.8	iText 2.1.7
Formal relationships for class	6878	8350	8866	10849
Formal Relationships f or interface	202	237	259	396
Formal Relationships for enum	0	0	0	0

TABLE 8: Formal relationships for class in iText.

Formal relationships for class	iText 1.3	iText 1.4	iText 1.4.8	iText 2.1.7
Folder-based relationships	2228	2643	2859	3958
File-based relationships	434	513	551	625
Inheritance-based relationships	680	833	871	839
Implements-based relationships	171	203	208	213
Composition-based relationships	2115	2689	2847	3312
Generics-based relationships	0	0	0	0
Access-based relationships	1122	1318	1358	1630
Inner-based relationships	64	69	78	119
Outer inheritance-based relationships	14	14	17	25
Outer implements-based relationships	0	0	0	0
Outer composition-based relationships	50	68	77	128

TABLE 9: Formal relationships for interface in iText.

Formal relationships for interface	iText 1.3	iText 1.4	iText 1.4.8	iText 2.1.7
Folder-based relationships	140	164	182	283
File-based relationships	28	32	35	44
Extends-based relationships	6	6	6	12
Generics-based relationships	0	0	0	0
Composition-based relationships	0	0	0	0
Access-based relationships	28	35	36	57
Inner-based relationships	0	0	0	0
Outer extends relationships	0	0	0	0
Outer composition relationships	0	0	0	0

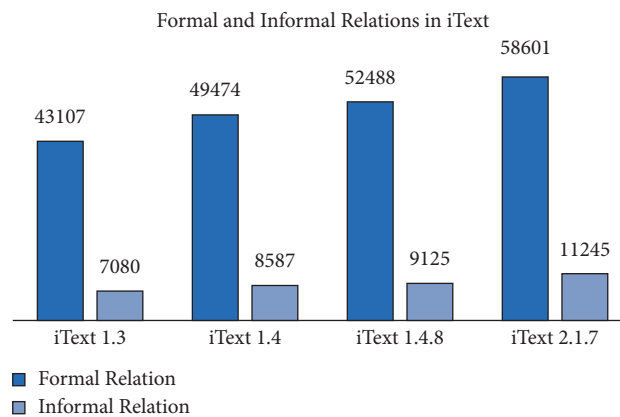


FIGURE 3: Count of formal and informal relationships in iText.

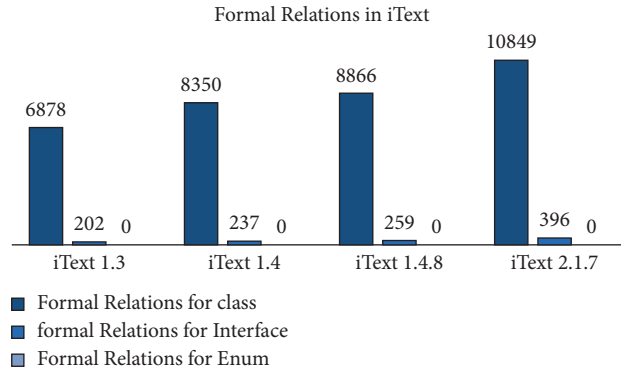


FIGURE 4: Count of formal relationships in iText.

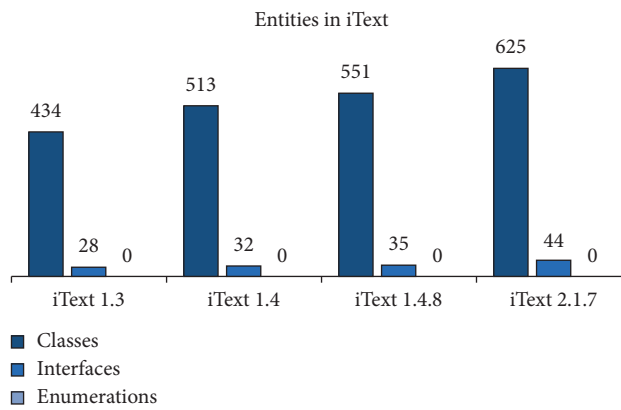


FIGURE 5: Count of entities in iText.

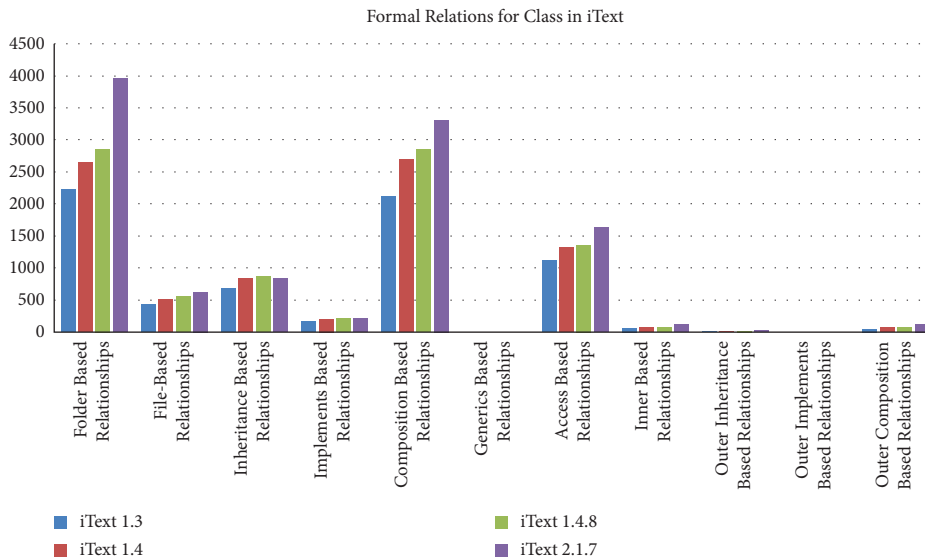


FIGURE 6: Count of formal relationships for class in iText.

(ii) For classes the most used relationships are folder-based relationships while access-based relationships, implements-based relationships, inheritance-based relationships, and file-based relationships have a good frequency of occurrence.

(iii) For interfaces outer extends-based relationships, outer composition-based relationships, and composition-based relationships are not used while extends-based relationships are only used in JUnit 4.8. Also, among used relationships, folder-based

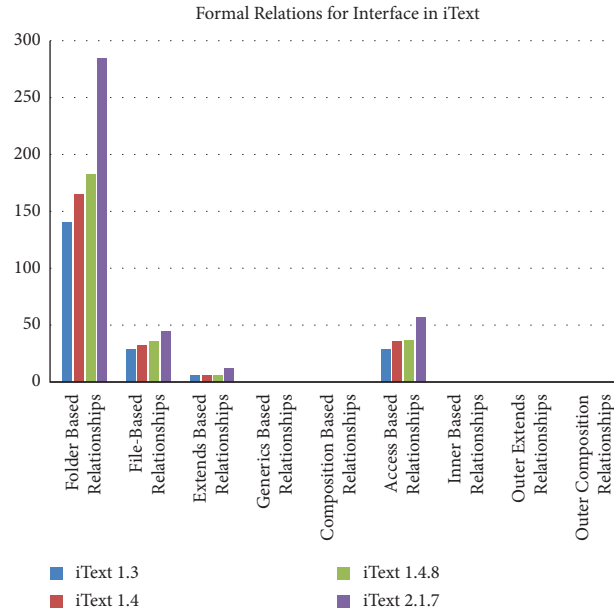


FIGURE 7: Count of formal relationships for interface in iText.

TABLE 10: Entities in JFreeChart.

Entities	JFreeChart 1.0.4	JFreeChart 1.0.7	JFreeChart 1.0.9	JFreeChart 1.0.14
Classes	432	468	468	510
Interfaces	84	91	91	106
Enumerations	0	0	0	0

TABLE 11: Formal and informal relationships in JFreeChart.

Relationships	JFreeChart 1.0.4	JFreeChart 1.0.7	JFreeChart 1.0.9	JFreeChart 1.0.14
Informal	48877	53862	54105	61294
Formal	8321	9059	9058	10056

TABLE 12: Formal relationships in JFreeChart.

Formal relationships	JFreeChart 1.0.4	JFreeChart 1.0.7	JFreeChart 1.0.9	JFreeChart 1.0.14
Formal relationships for class	7636	8314	8313	9196
Formal relationships for interface	685	745	745	860
Formal relationships for enum	0	0	0	0

TABLE 13: Formal relationships for class in JFreeChart.

Formal relationships for class	JFreeChart 1.0.4	JFreeChart 1.0.7	JFreeChart 1.0.9	JFreeChart 1.0.14
Folder-based relationships	2184	2385	2385	2599
File-based relationships	432	468	468	510
Inheritance-based relationships	816	917	917	1013
Implements-based relationships	897	970	970	1052
Composition-based relationships	2069	2234	2233	2514
Generics-based relationships	0	0	0	0
Access-based relationships	1217	1308	1308	1475
Inner-based relationships	14	19	19	20
Outer inheritance-based relationships	2	3	3	3
Outer implements-based relationships	0	0	0	0
Outer composition-based relationships	5	10	10	10

TABLE 14: Formal relationships for interface in JFreeChart.

Formal relationships for interface	JfreeChart 1.0.4	JFreeChart 1.0.7	JfreeChart 1.0.9	JFreeChart 1.0.14
Folder-based relationships	411	450	450	526
File-based relationships	84	91	91	106
Extends-based relationships	81	91	91	95
Generics-based relationships	0	0	0	0
Composition-based relationships	0	0	0	0
Access-based relationships	109	113	113	133
Inner-based relationships	0	0	0	0
Outer extends relationships	0	0	0	0
Outer composition relationships	0	0	0	0

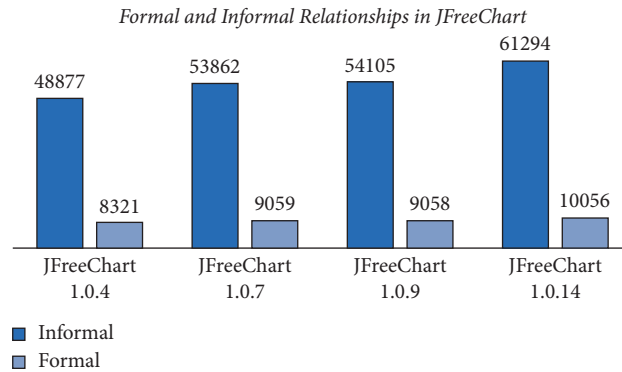


FIGURE 8: Count number of formal and informal relationships in JfreeChart.

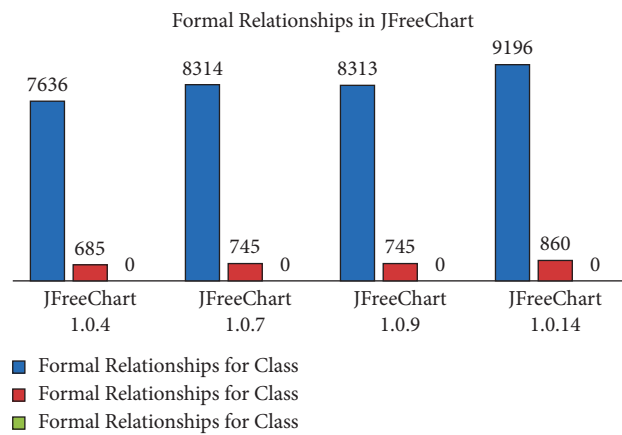


FIGURE 9: Count number of formal relationships in JfreeChart.

relationships have the highest frequency of occurrence.

- (iv) Generics bases relationships have been introduced for classes and interfaces.
- (v) The used relationships in enumerations are folder-based, file-based, and access relationships while composition-based relationships are only used in JUnit 4.11. Among them, the dominant relationship is folder-based relation.
- (vi) By comparing Figures 13 and 14 it can be calculated that in JUnit 89% of formal relationships are based on formal relationships of classes while 10% of formal relationships are based on formal

relationships of interfaces. Similarly, formal relationships in JUnit 4.9, JUnit 4.10, and JUnit 4.11 are based on 0.57%, 0.55%, and 0.92% of formal relationships of enumerations respectively.

Figures 15–18 provide a graphical representation of statistics of a Junit software system.

5.1.4. *Weka*. Table 21–26 summarizes the statistics of the Weka software system. From the statistics, it has been observed that.

- (i) In the Weka software system the dominant entity is class while enumerations have a low frequency of occurrence.

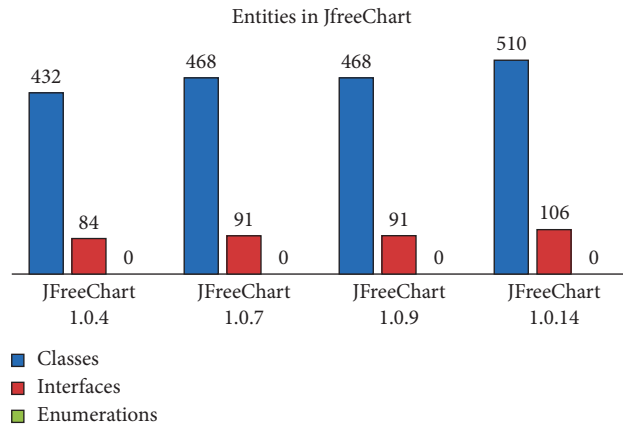


FIGURE 10: Count of entities in JfreeChart.

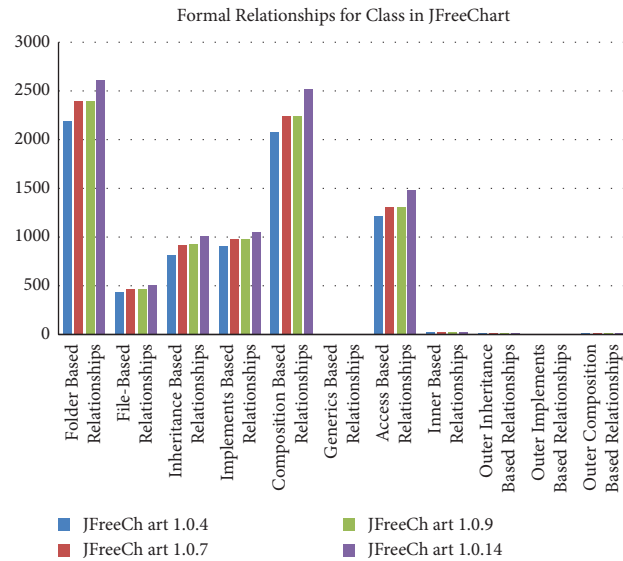


FIGURE 11: Count number of formal relationships for class in JfreeChart.

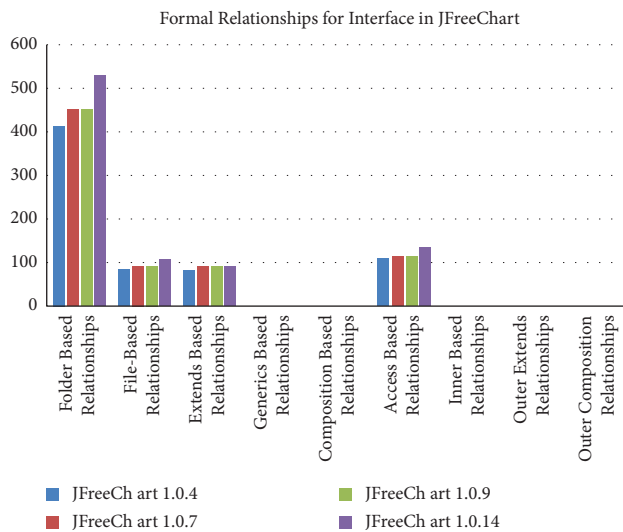


FIGURE 12: Count number of formal relationships for interface in JfreeChart.

TABLE 15: Entities in Junit.

Entities	Junit 4.8	Junit 4.9	Junit 4.10	Junit 4.11
Classes	148	134	137	137
Interfaces	32	30	30	32
Enumerations	0	1	1	2

TABLE 16: Formal and informal relationships in Junit.

Relationships	Junit 4.8	Junit 4.9	Junit 4.10	Junit 4.11
Informal	6616	6171	6286	6705
Formal	1746	1572	1609	1629

TABLE 17: Formal relationships in Junit.

Formal relationships	Junit 4.8	Junit 4.9	Junit 4.10	Junit 4.11
Formal relationships for class	1568	1399	1436	1441
Formal relationships for interface	178	164	164	173
Formal relationships for enum	0	9	9	15

TABLE 18: Formal relationships for class in Junit.

Formal relationships for class	Junit 4.8	Junit 4.9	Junit 4.10	Junit 4.11
Folder-based relationships	653	602	617	615
File-based relationships	148	134	137	137
Inheritance-based relationships	161	153	154	157
Implements-based relationships	35	30	34	34
Composition-based relationships	207	178	182	188
Generics-based relationships	19	14	14	14
Access-based relationships	320	262	270	265
Inner-based relationships	16	16	17	18
Outer inheritance-based relationships	0	0	0	0
Outer implements-based relationships	0	0	0	0
Outer composition-based relationships	9	10	11	13

TABLE 19: Formal relationships for interface in Junit.

Formal relationships for interface	Junit 4.8	Junit 4.9	Junit 4.10	Junit 4.11
Folder-based relationships	125	120	120	127
File-based relationships	32	30	30	32
Extends-based relationships	1	0	0	0
Generics-based relationships	3	1	1	1
Composition-based relationships	0	0	0	0
Access-based relationships	15	12	12	12
Inner-based relationships	2	1	1	1
Outer extends relationships	0	0	0	0
Outer composition relationships	0	0	0	0

(ii) For classes composition-based relationships are the most dominant relationships. Other relationships having a good frequency of occurrence are folder-based and access-based relationships.

(iii) Folder-based relationships, file-based relationships, and access-based relationships have a high frequency of occurrence in interfaces while other relationships are either have a very low frequency of

occurrence or not used at all. Among used relationships, folder-based relationships have the highest frequency.

(iv) Generics-based relationships are used in classes and interfaces but have very low usage.

(v) Like JUnit the used relationships in enumerations are folder-based, file-based, access relationships, and composition-based relationships. Among them,

TABLE 20: Formal relationships for enum in JUnit.

Formal relationships for enumeration	JUnit 4.8	JUnit 4.9	JUnit 4.10	JUnit 4.11
Folder-based relationships	0	6	6	10
File-based relationships	0	1	1	2
Implements-based relationships	0	0	0	0
Composition-based relationships	0	0	0	1
Genericity-based relationships	0	0	0	0
Access-based relationships	0	2	2	2
Outer implements relationships	0	0	0	0

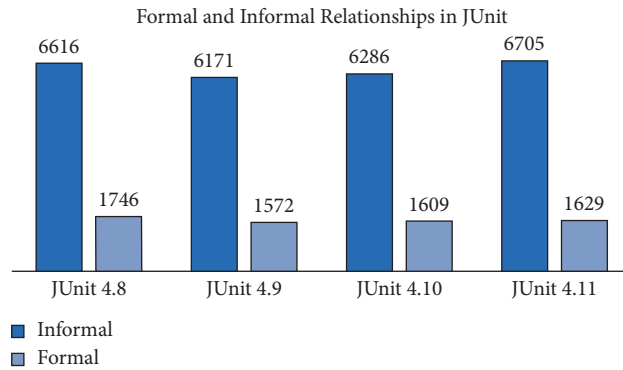


FIGURE 13: Count of formal and informal relationships in JUnit.

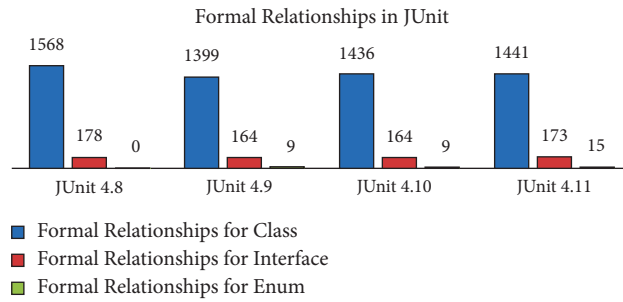


FIGURE 14: Count of formal relationships in JUnit.

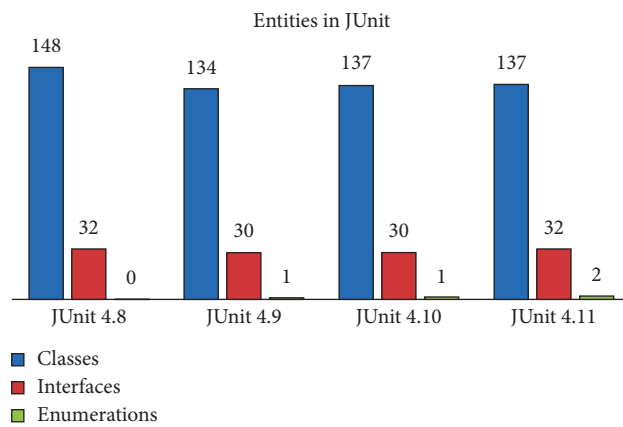


FIGURE 15: Count of entities in JUnit.

the dominant relationship is folder-based relation. Access-based and composition-based relationships have the same frequency of occurrence.

(vi) By comparing Figures 19 and 20, it can be calculated that in Weka 95% of formal relationships are based on formal relationships of classes, 4% of formal

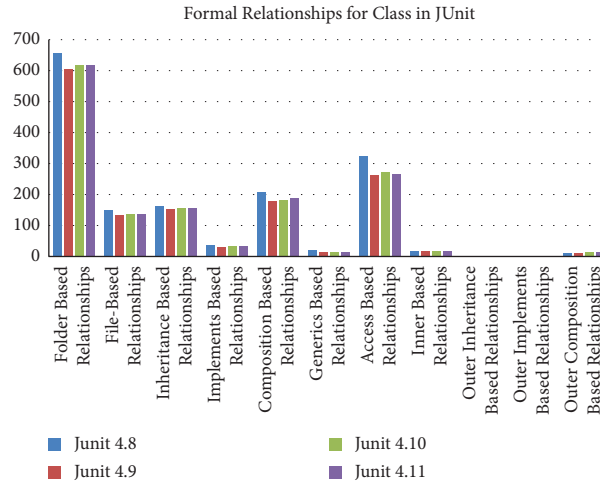


FIGURE 16: Count of formal relationships for class in JUnit.

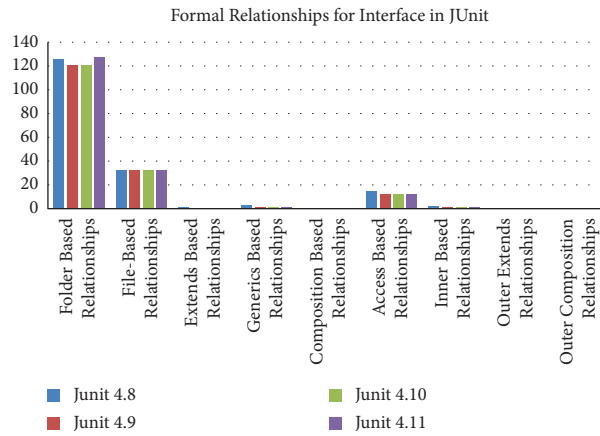


FIGURE 17: Count of formal relationships for interface in JUnit.

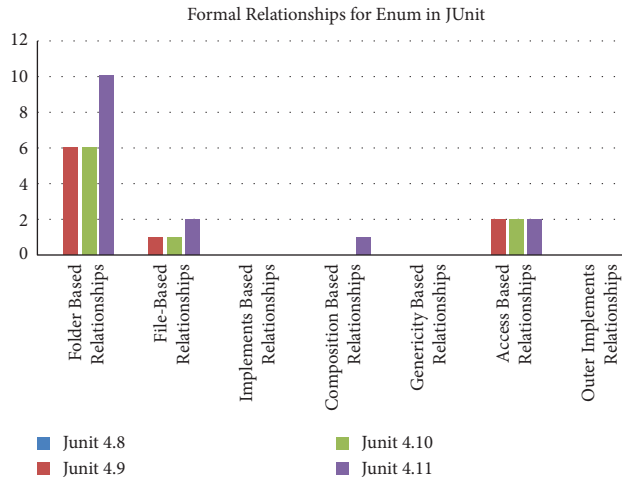


FIGURE 18: Count of formal relationships for enum in junit.

TABLE 21: Entities in weka.

Entities	Weka-dev 3.7.5	Weka-dev 3.7.6	Weka-dev 3.7.7	Weka-stable 3.6.6
Classes	1162	1185	1190	1236
Interfaces	149	151	152	119
Enumerations	41	42	42	24

TABLE 22: Formal and informal relationships in weka.

Relationships	Weka-dev 3.7.5	Weka-dev 3.7.6	Weka-dev 3.7.7	Weka-stable 3.6.6
Informal	124080	129103	129948	136926
Formal	20358	20847	20932	22695

TABLE 23: Formal relationships in weka.

Formal relationships	Weka-dev 3.7.5	Weka-dev 3.7.6	Weka- dev 3.7.7	Weka-stable 3.6.6
Formal relationships for class	19250	19720	19800	21852
Formal relationships for interface	887	900	905	720
Formal relationships for enum	221	227	227	123

TABLE 24: Formal relationships for class in weka.

Formal relationships for class	Weka-dev 3.7.5	Weka-dev 3.7.6	Weka-dev 3.7.7	Weka-stable3.6.6
Folder-based relationships	4705	4794	4816	5003
File-based relationships	1162	1185	1190	1236
Inheritance-based relationships	1955	1976	1984	2141
Implements-based relationships	1532	1595	1604	1697
Composition-based relationships	6289	6457	6482	7671
Generics-based relationships	8	10	10	6
Access-based relationships	2775	2858	2868	3300
Inner-based relationships	357	368	369	335
Outer inheritance-based relationships	97	97	97	78
Outer implements-based relationships	14	14	14	6
Outer composition-based relationships	356	366	366	379

TABLE 25: Formal relationships for interface in weka.

Formal relationships for interface	Weka-dev 3.7.5	Weka-dev 3.7.6	Weka-dev 3.7.7	Weka-stable3.6.6
Folder-based relationships	556	563	566	442
File-based relationships	149	151	152	119
Extends-based relationships	43	45	45	40
Generics-based relationships	6	7	7	6
Composition-based relationships	0	0	0	0
Access-based relationships	132	133	134	113
Inner-based relationships	1	1	1	0
Outer extends relationships	0	0	0	0
Outer composition relationships	0	0	0	0

TABLE 26: Formal relationships for enum in weka.

Formal relationships for enumeration	Weka-dev 3.7.5	Weka-dev 3.7.6	Weka-dev 3.7.7	Weka-stable3.6.6
Folder-based relationships	176	181	181	99
File-based relationships	41	42	42	24
Implements-based relationships	0	0	0	0
Composition-based relationships	2	2	2	0
Genericity-based relationships	0	0	0	0
Access-based relationships	2	2	2	0
Outer implements relationships	0	0	0	0

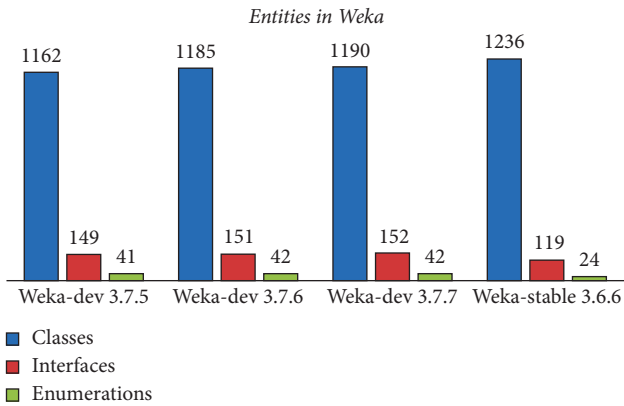


FIGURE 19: Count of entities in weka.

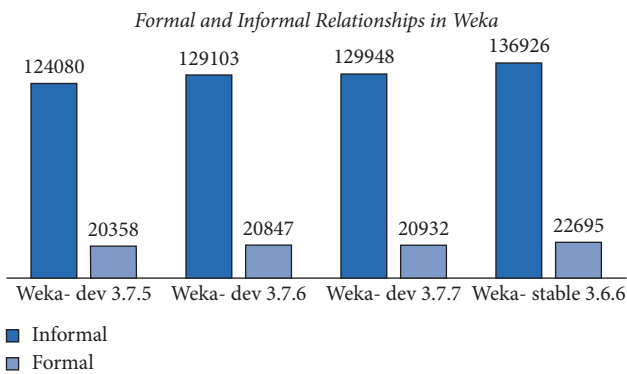


FIGURE 20: Count of formal and informal relationships in weka.

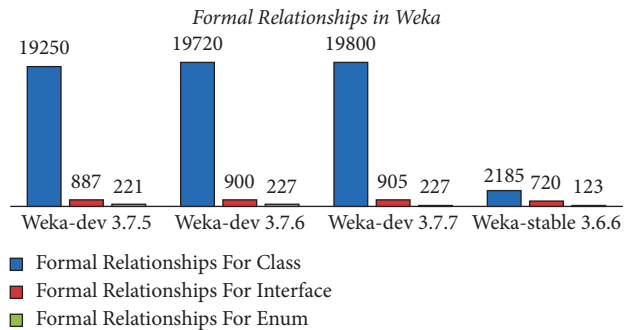


FIGURE 21: Count of formal relationships in weka.

relationships are based on formal relationships of interfaces and, 1% of formal relationships are based on formal relationships of enumerations.

Figures 19–24 provide a graphical representation of statistics of the Weka software system.

6. Discussion and Analysis

The above Tables 5–26, we have provided the result and analysis of datasets that we have used to conduct our experiment. Different tables and graphs are provided for result analysis, and it can be concluded that the most dominant entity is class, informal relationships have higher occurrence

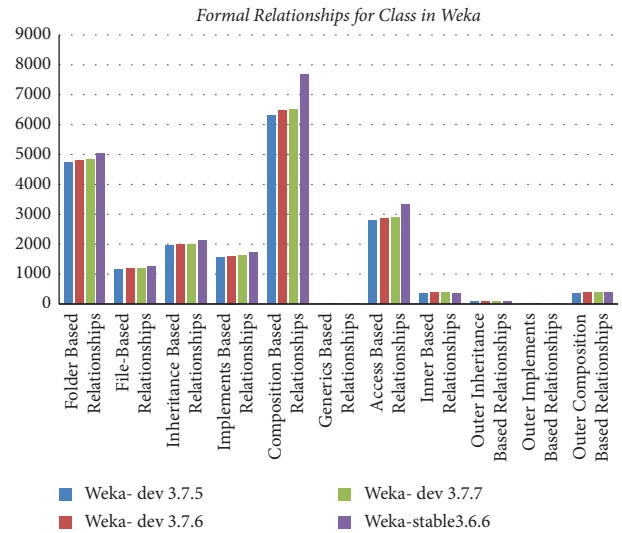


FIGURE 22: Count of formal relationships for class in weka.

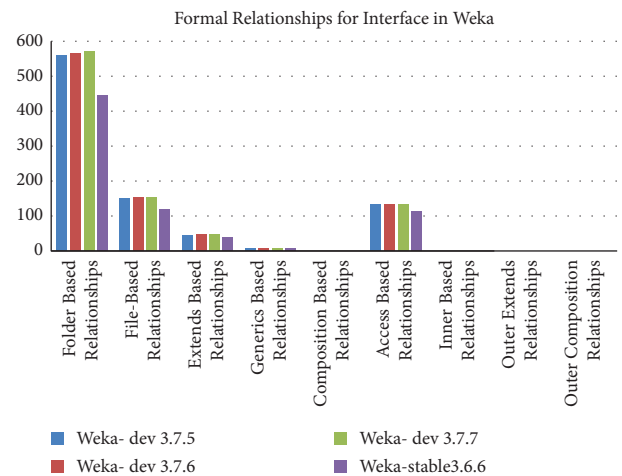


FIGURE 23: Count of formal relationships for interface in weka.

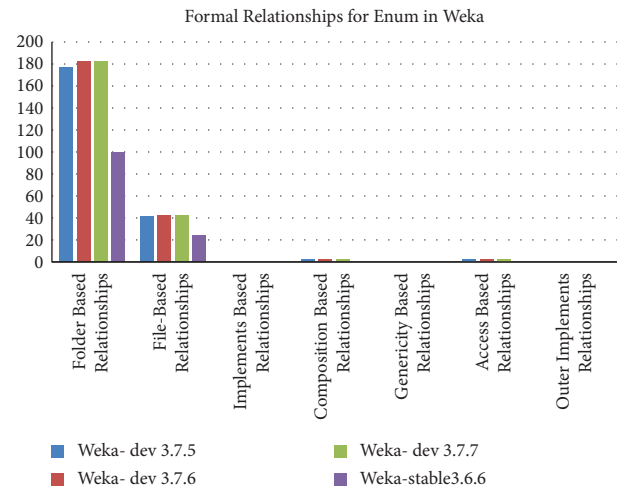


FIGURE 24: Count of formal relationships for enum in weka.

than formal relationships, and formal relationships are mostly based on formal relationships for classes. We provided a framework for the extraction of entities and relationships that exist t in a Java software system. Our Fact Extractor can extract three types of entities which are classes, interface, and enumerations. The fact extractor can be used to extract both formal and informal relationships. For classes, interfaces, and enumerations, the total number of formal relationships retrieved is 74, 43, and 31, respectively. Similarly, the fact extractor extracted a total of 73 informal relationships.

7. Conclusion

Fact Extractor System for Java Applications (FESJA) is an automatic software modularization tool, used to extract entities and relationships from java source code. The entities extracted by the fact extractor system are classes, interfaces, and enumerations. The Fact Extractor can extract both formal and informal relationships. The formal relationships are categorized into three parts which are formal relationships for classes, formal relationships for interfaces, and formal relationships for enumerations. For evaluation of relationships, we performed our experiment on four systems (dataset). The systems are iText, JFreeChart, Junit, and Weka software systems. We have provided different graphs and tables for analysis of results and presented our observations which can help researchers to carry out tasks related to software modularization process, software architecture recovery, and software clustering.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

The authors are grateful to the Deanship of Scientific Research, King Saud University for funding through Vice Deanship of Scientific Research Chairs.

References

- [1] Z. Shah, R. Naseem, M. A. Orgun, A. Mahmood, and S. Shahzad, "Software clustering using automated feature subset selection," in *Proceedings of the International Conference on Advanced Data Mining and Applications*, pp. 47–58, Hangzhou, China, December 2013.
- [2] I. Candela, G. Bavota, B. Russo, and R. Oliveto, "Using cohesion and coupling for software remodularization," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 3, pp. 1–28, 2016.
- [3] P. Behnamghader, D. M. Le, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, "A large-scale study of architectural evolution in open-source software systems," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1146–1193, 2017.
- [4] S. Muhammad, *Experimental Evaluation of Relationships for the Modularization of Object-Oriented Software Systems*, Quaid-e-Azam University Islamabad, Islamabad, Pakistan, 2010.
- [5] Q. Alsarhan, B. S. Ahmed, M. Bures, and K. Z. Zamli, "Software module clustering: an in-depth literature analysis," *IEEE Transactions on Software Engineering*, 2020.
- [6] R. Naseem, M. M. Deris, O. Maqbool, and S. Shahzad, "Euclidean space-based hierarchical clusters combinations: an application to software clustering," *Cluster Computing*, vol. 22, no. 3, pp. 7287–7311, 2019.
- [7] R. Naseem, M. B. M. Deris, O. Maqbool, J.-p. Li, S. Shahzad, and H. Shah, "Improved binary similarity measures for software modularization," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 8, pp. 1082–1107, 2017.
- [8] K. Raimond and J. Lovesum, "A novel approach for automatic modularization of software systems using extended ant colony optimization algorithm," *Information and Software Technology*, vol. 114, pp. 107–120, 2019.
- [9] I. Şora and C. B. Chirila, "Finding key classes in object-oriented software systems by techniques based on static analysis," *Information and Software Technology*, vol. 116, Article ID 106176, 2019.
- [10] H. Li, T. Wang, W. Pan et al., "Mining key classes in java projects by examining a very small number of classes: a complex network-based approach," *IEEE Access*, vol. 9, pp. 28076–28088, 2021.
- [11] W. Jiang and N. Dai, "Identifying key classes algorithm in directed weighted class interaction network based on the structure entropy weighted leaderrank," *Mathematical Problems in Engineering*, vol. 2020, Article ID 9234042, 12 pages, 2020.
- [12] X. Du, T. Wang, W. Pan et al., "COSPA: identifying key classes in object-oriented software using preference aggregation," *IEEE Access*, vol. 9, pp. 114767–114780, 2021.
- [13] I. Hussain, A. Khanum, A. Q. Abbasi, and M. Y. Javed, "A novel approach for software architecture recovery using particle swarm optimization," *The International Arab Journal of Information Technology*, vol. 12, no. 1, pp. 32–41, 2015.
- [14] M. Aghdasifam, H. Izadkhah, and A. Isazadeh, "A new metaheuristic-based hierarchical clustering algorithm for software modularization," *Complexity*, vol. 2020, Article ID 1794947, 25 pages, 2020.
- [15] I. Zahoor, O. Maqbool, and R. Naseem, "Web application fact extractor (WAFE)," in *Proceedings of the Eighth International Conference on Digital Information Management (ICDIM 2013)*, pp. 379–384, IEEE, Islamabad, Pakistan, September 2013.
- [16] J. Oh, S. Lee, A. H. Kim, and W. H. Ahn, "An automatic extraction scheme of dependency relations between web components and web resources in Java web applications," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 22, no. 3, pp. 458–470, 2018.
- [17] J. Oh, W. H. Ahn, and T. Kim, "Automatic extraction of dependencies between web components and database resources in java web applications," *Journal of Information and Communication Convergence Engineering*, vol. 17, no. 2, pp. 149–160, 2019.
- [18] M. Thakur, K. Patidar, S. Chouhan, and R. Kushwah, "A clustering based on optimization for object oriented quality prediction," *International Journal of Advanced Technology and Engineering Exploration*, vol. 5, no. 41, pp. 62–69, 2018.
- [19] I. Aljarah, H. Faris, S. Mirjalili, N. Al-Madi, A. Sheta, and M. Mafarja, "Evolving neural networks using bird swarm

- algorithm for data classification and regression applications,” *Cluster Computing*, vol. 22, no. 4, pp. 1317–1345, 2019.
- [20] M. Krishnan, *Feature-Based Analysis of the Open-Source Using Big Data Analytics*, University of Missouri-Kansas City, Kansas City, MO, USA, 2015.
- [21] N. Alshuqayran, N. Ali, and R. Evans, “Towards microservice architecture recovery: an empirical study,” in *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*, pp. 47–4709, IEEE, Seattle, WA, USA, April 2018.
- [22] M. Trifu, “Architecture-aware, adaptive clustering of object-oriented systems,” Diploma Thesis, Forschungszentrum Informatik Karlsruhe, Karlsruhe, Germany, 2003.
- [23] A. Rathee and J. K. Chhabra, “Software remodularization by estimating structural and conceptual relations among classes and using hierarchical clustering,” in *Proceedings of the International Conference on Advanced Informatics for Computing Research*, pp. 94–106, Jalandhar, India, March 2017.
- [24] S. Eski and F. Buzluca, “An automatic extraction approach: transition to microservices architecture from a monolithic application,” in *Proceedings of the 19th International Conference on Agile*, pp. 1–6, Porto, Portugal, May 2018.
- [25] N. Teymourian, H. Izadkhah, and A. Isazadeh, *A Fast Clustering Algorithm for Modularization of Large-Scale Software Systems*, IEEE Transactions on Software Engineering, Piscataway, NJ, USA, 2020.
- [26] S. Rafi, W. Yu, M. A. Akbar, S. Mahmood, A. Alsanad, and A. Gumaei, “Readiness model for DevOps implementation in software organizations,” *Journal of Software: Evolution and Process*, vol. 33, no. 4, Article ID e2323, 2021.
- [27] M. A. Akbar, S. Mahmood, C. Meshram, A. Alsanad, A. Gumaei, and S. A. AlQahtani, *Barriers of Managing Cloud Outsource Software Development Projects: A Multivocal Study*, Multimedia Tools and Applications, Berlin, Germany, 2021.
- [28] R. N. Alanazi, *Software Analytics for Improving Program Comprehension*, University of Missouri-Kansas City, Kansas City, MO, USA, 2021.
- [29] C. Tjortjij, “Mining association rules from code (MARC) to support legacy software management,” *Software Quality Journal*, vol. 28, no. 2, pp. 633–662, 2020.
- [30] V. K. Yadav, S. Kumar, and M. Mittal, “Prediction of software maintenance effort of object-oriented metrics based commercial systems,” *African Journal of Computing & ICTs*, vol. 8, no. 1, pp. 163–172, 2015.
- [31] M. A. Akbar, W. Naveed, A. A. Alsanad et al., “Requirements change management challenges of global software development: an empirical investigation,” *IEEE Access*, vol. 8, pp. 203070–203085, 2020.