Building a tool for clustering source code to facilitate program comprehension

2005

MSc Dissertation submitted to

The University of Manchester

Submitted By

Behram Khan

School of Informatics

Supervisor: Tjortjis, Christos

Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Christos Tjortjis for his immense support and guidance throughout this year in completing this dissertation.

I would also like to thank my friends for their constant affection and support. They have been a constant source of inspiration for me.

Most of all, I would like to specially thank my parents and my sister, for putting up with me and supporting me throughout my life.

Abstract

A well documented problem faced by software maintainers when understanding a software system is the lack of familiarity with it, combined with the lack of accurate documentation. Several problems in software maintenance occur because programs are modified with little understanding of the overall organization of the source code and of full impact of the modifications. Most programs are structured as a number of subsystems, consisting of code that collaborates to provide a common functionality to the program. An important aspect of program understanding in software maintenance is to perceive this subsystem structure.

Cluster analysis can be of potential use in deriving a meaningful subsystem structure of a program from its source code, thus facilitating program comprehension and understanding.

The product of this project is a clustering tool that is able to decompose a software code into meaningful subsystems (clusters). The tool uses input data extracted from a program and produces an abstraction of the program as a number of subsystems. The tool was evaluated against programs of various sizes and languages. Results showed that the clustering tool was able to derive accurate subsystem abstraction and identify interrelationships amongst the components.

Table of Contents

List of Figures

List of Tables

1 Introduction

The decomposition of a large software system into “meaningful” subsystems is essential for both the development and maintenance phases of a software project.

The definition of the term “large software” is constantly changing as the size of a software system continues to grow rapidly. Advances in hardware technology concerning the speed, size and cost of primary and secondary storage devices, as well as the introduction of object oriented technology in programming practice have significantly increased the size of software systems.

When the system become too large, it is very difficult to ensure that the structure of the present system is the intended one. System evolution over many years inevitably results in a very complex interconnected system structure. Due to this structure, minor changes in some part of the system may have unforeseen effects on other parts of the system which may not function correctly anymore without being changed themselves. Moreover, the original documentation, if it exists at all, becomes outdated as the system evolves, as keeping an updated documentation is mostly not the priority of the system developers who are much more concerned with meeting the deadlines. The fact that developers often discontinue their association with such large projects intensifies the problems, since they take a lot of knowledge about the system with them.

These factors contribute to the transformation of a piece of software into what is known as “legacy code”, (a piece of code that one uses but does not necessarily understand). The drawback of having legacy code in a software system become obvious when the time comes to alter its functionality, to adapt it to a new hardware platform or operating system, or to improve its performance. One needs to understand the code over all again.

To address the problems mentioned above researchers in reverse engineering community have been developing clustering tools. Software clustering techniques can break a software system into smaller parts, such as subsystems or modules. This can provide a starting point towards the recovery of the systems structure. Typical resources found in subsystems include modules, classes, and possibly, other subsystems. Subsystems facilitate program understanding by treating sets of source code resources as high-level entities.

1.1 Aim of the project

This project is about developing a clustering tool that is able to decompose a software code into meaningful clusters. The tool uses unsupervised clustering techniques that meaningfully decompose a software system to help maintainers to recognize parts of source code that have common characteristics, thus facilitating program understanding. Data extracted from source code (of different languages) are clustered, in order to identify logical, behavioral and structural correlations amongst program components.

1.1.1 Objectives

The major challenge of this work is to adapt clustering techniques to the peculiarities present in the application domain of a source code.
The main objectives of this work are

- **Specification of Input-Models:** The input-models are concerned with the specification of program entities and their attributes. It is these entities that are then grouped into subsystems. The program components that are used as entities are constructs within the source code. These entities must have several attributes that provide the means for measuring similarity between entities.

- **Specification of Similarity Metrics:** Similarity matrices determine the similarity between program entities based on their attributes. The choice of a proper similarity metric is a crucial part of the project, as it can have more influence on the result then the clustering algorithm itself **[1]**. Therefore, existing similarity metrics must be examined for use with different types of

attributes and the suitable ones selected, and tailored for the peculiarities of this domain.

- **Specification of the clustering algorithm:** Clustering algorithms meaningfully group program entities into subsystems (clusters) based on similarities provided by the previous step.

- **Implementation of the software tool:** The software tool is to be developed that takes input data extracted from a program's source code and produces an abstraction of the program as a number of subsystems.

The project is about producing a tool for automated approach to program understanding. Therefore it is assumed that the user has no expert knowledge of the program being analyzed.

1.2 Results

The results produced by the tool are very encouraging and constructive. Both primary and secondary requirements are achieved successfully.
Three case studies were carried out in order to evaluate the results produced by the software tool. The accuracy of the results was evaluated by comparing the sub-system abstractions with expert's mental models for two of the programs. The produced results were found to be meaningful in almost all the cases.

1.2.1 Achievements

The tool realizes the original design virtually in its entirety and effectively fulfils the majority of its requirements, including the core requirements. Major achievements of the project are

- **Specification of Input-Models:** Classes and functions in a source code are selected as entities to be clustered. Input models for both functions and classes are specified that completely fulfil all of the requirements expected of the input models in the requirements phase.

- **Specification of Similarity Matrices:** Customized Association coefficients are used to determine the similarity between program entities.

- **Specification of clustering algorithms:** Hierarchical agglomerative clustering algorithms (single-link and complete-link clustering) are used to cluster entities into meaningful subsystems.

- **Implementation of a software tool:** Software tool is implemented that incorporates the input models, similarity measures and the clustering algorithms. The tool takes input data extracted from a program's source code and produces an abstraction of the program as a number of subsystems. The results produced are then evaluated by comparing them with the expert's mental model of the program. The tool is responsive and has a user friendly Graphical User Interface. As the GUI is developed according to the conventional software structure, most of the users who participated in the evaluation learnt its usage very quickly.

1.3 Structure of the report

The main body of the report is spread across six chapters. A brief overview of these chapters is provided below.

Background

This chapter presents the background research that was deemed necessary to be carried out in order to acquire a full understanding of the problem domain. It comprises of a general review of software maintenance and the importance of program comprehension in this process. The role of data mining, with especial emphasis on the clustering analysis and its role in program comprehension is analyzed. The steps involved in clustering a software system are discussed along with some relevant work previously done in this regard.

Requirements Analysis

This Chapter presents the requirements analysis. Various functional and non functional requirements are discussed along with their relative priority in the project.

Design

This chapter explains the overall system architecture and design of the system components. It explains the whole process of creating the input models to extract data from the software, the similarity measures that are used to calculate similarities between different entities within the software, and the algorithms that are used to actually form the clusters for software remodulerizeation. All the aspects of the software system along with its interaction with the user are thoroughly discussed by using detailed UML and block diagrams. It also explains the data base design and mapping of the conceptual model to relational schema in Microsoft access database.

Implementation

Here, the realization of the design is described, in terms of both, the process and the implemented software system that resulted from it. An overview of all the modules (classes and the functions) along with some important code fragments that were used to implement the software system is given. Screen shots of the graphical user interface are also presented in order to show the vide range of options available to the user to perform the clustering process.

Testing and Evaluation

This chapter describes the internal testing of the product and evaluates the extent to which the product satisfies the stated requirements.

Conclusion and Future Work

The conclusion to the report presents overview of the project, some general conclusions about the project and its outcome and a consideration of the future work that stems from work that has already been undertaken.

2

Background

Background investigation covers a thorough review of all the relevant material and related approaches in the area of software maintenance (with especial emphasis on the topics more related to this particular project). At first the concept of *Software Maintenance* and the importance of *Program Comprehension* in this process are discussed. Then, it discusses the idea of *Data Mining* (a process in KDD), with more emphasis on *Clustering*, which is one of the data mining techniques and is the basis of this project. Finally some of the previous works done using data mining for program comprehension are summarized.

2.1 Software Maintenance

Software maintenance is defined as ***"Modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment"*** **[2].**

The maintenance of the software is motivated by a number of factors **[2].** These factors involve ***providing continuity of service,*** which includes fixing bugs, recovering from failures, and accommodating changes in the operating system and hardware. ***Supporting mandatory upgrades,*** which are needed because of things like amendments to the government regulation, to maintain competitive edge over the rivals, and so on? ***Supporting user requests for improvements*** includes supporting requests such as enhancement of functionality, better performance and customization to local working patterns. ***Facilitating future maintenance work*** involves code, database restructuring, and updating documentation.

In order to achieve the objectives of maintenance discussed above a wide spectrum of changes to the software product may be necessary. There are four categories of

changes to the software system: *Corrective, Adaptive, Perfective and preventive changes* **[3] [2].**

- **Corrective change** refers to modifications initiated by defects in the software. A defect can result from design errors, logic errors and coding errors.

- **Adaptive maintenance** is performed to make a computer program usable in a changed environment. Different environment may involve moving software to a different hardware or software platform.

- **Perfective Change** improves the performance, maintainability or other attributes of a computer program.

- **Preventive maintenance** is undertaken to prevent malfunctions or to improve maintainability of the software.

2.2 Program Comprehension in Software Maintenance

The above discussion looked at the type of possible changes in the process of software maintenance. However, an area, which is fundamental to effective changes in the software, is Software understanding. **[2]** Explains some of the important steps that are carried out during the maintenance process, which are

- Having a general knowledge about the functionality of the software system and how it relates to its environment.
- Identify where in the system changes need to be made.
- Having an in-depth knowledge how the parts to be corrected or modified work.

The ultimate aim of program comprehension is to be able to successfully implement requested changes. This entails acquiring information about certain aspects of the software system such as the problem domain, execution effect, cause-effect relation, product-environment relation and decision support features of the software.

Program comprehension is a demanding task comprising up to 80% of the total time spent on software maintenance, which in turn is the most expensive process in the lifetime of software **[2].** No commonly accepted framework exists that can be used to guide comprehension in the absence of familiarity with the code or the domain. Neither there is a well-defined set of metrics for measuring properties of the code such as structure, modularity and so on **[4]**. A major research challenge therefore is to understand key objectives in the program comprehension process and to provide an efficient automated (semi-automated) support for it.

The sections that follow, explain the concept of the data mining process and its tasks, and explain the use of data mining techniques to formulate an automated method of program comprehension and understanding.

2.3 Data Mining

***"Data Mining is the use of algorithms to extract the information and patterns derived by the KDD (*Knowledge Discovery in databases*) process"* [5]**.

Data mining is an important step in the KDD process. Data mining involves many different algorithms to accomplish different tasks. All these algorithms attempt to fit a model to the data. The algorithms examine the data and determine a model that is closest to the characteristics of the data being examined.

Data mining techniques can discover non-trivial and previously unknown relationships among records or attributes in large databases **[6]**. This observation highlights the capability of data mining to induce useful knowledge about the design of large legacy systems. Data mining has three fundamental features that make it a valuable tool for program comprehension and related maintenance tasks **[7].**

- It can be applied to large volumes of data. This implies that data mining has the potential to analyze large legacy systems with complex structure.

- It can be used to expose previously unknown non-trivial patterns and associations between items in a data base. Therefore, it can be utilized in

order to reveal hidden relationships between system or program components.

- Its techniques can extract information regardless of any previous domain knowledge. This feature is ideal for maintaining software with poor knowledge about its functionality or implementation details.

2.3.1 Basic Data Mining Tasks

There are several components of data-mining algorithms discussed in literature. A detailed description of Clustering is given below (as this component is most related to this particular project), while the other components are discussed briefly **[8] [5] [2] [9]**.

- **Classification** is the process of mapping a data item into one of several predefined categorical classes. It is often referred to as supervised learning because the classes are determined before examining the data.

- **Regression** is used to map a data item to a real valued prediction variable. The regression involves the learning of the function that does this mapping.

- **Summarization** provides a compact description for a subset of data. Summarization is also called characterization or generalization.

- **Rule generation** extracts classification rules from the data. Association rule mining refers to discovering association relationship among different attributes.

- **Dependency modeling** describes significant dependencies among variables. Dependency models exist at two levels, the structural and quantitative level **[6].** The structural level of the model specifies which variables are locally dependent on each other, whereas the quantitative level of the model specifies the strengths of the dependencies using some numerical scale.

- **Sequence analysis** models sequential patterns, like time-series analysis, gene sequence and so on. These patterns are similar to associations in that data are related, but the relationship is based on time.

- **Clustering** is a function that groups a set of data items into clusters. Cluster is a collection of data items similar to one another within the same cluster and dissimilar to the items in other cluster. It is often referred to as un-supervised learning because the clusters are not determined before examining the data.

The major challenge of this work is to use cluster analysis, to meaningfully decompose software into subsystems. Therefore the rest of the sections in this chapter extensively deal with the process of clustering and its use in software comprehension and understanding.

2.4. Clustering

Clustering techniques are used in many different areas for a wide spectrum of problems. Among the areas in which cluster analysis is used are graph theories, business area analysis, information architecture, information retrieval, resource allocation, image processing, software testing, galaxy studies, chip design, pattern recognition, economics, statistics and biology.

Cluster analysis or clustering is "a generic term for set of techniques which produce classifications from initially unclassified data." **[10]**. It aims to solve the problem of classification, namely to divide a set of data objects into a set of classes, the objects of each of which are considered to be similar to others within the same class and dissimilar from those in others. According to **[11]**, the two main approaches to cluster analysis are probabilistic and deterministic approaches, the latter of which often gives rise to what are classed as hierarchical techniques.

Each data object, or individual may have any number of attributes, which may be either quantitative or qualitative. A deterministic approach involves measuring the *similarity* or *distance* (dissimilarity) between each pairing of data objects, based on the values of these attributes, via *similarity/distance measures*. A great number of

different examples of such measures are commonly used. The values arising from the pair-wise comparison of all individuals can be displayed within a *symmetric distance* or *symmetric similarity matrix*. The values in a distance matrix can be any non-negative value, while the values in a similarity matrix are in the range 0 to 1, with 0 representing a pairing of two completely dissimilar individuals and 1 representing that the two are identical. The values within such a matrix can be used as the basis for grouping together of individuals, the result of which is referred to as *cluster*.

2.5 Issues in Clustering

2.5.1 Data Model

The first step in the clustering process is to produce a data model. That is, to build an abstraction of the real world in which the entities to be clustered are described according to some scheme.

Algorithms for clustering traditionally use data model shown in the table 2.1.

Entities	Attribute1	Attribute2	Attribute3	Attribute4
----------	----------	----------	----------	----------
----------	----------	----------	----------	----------

Table 2.1 Data Model traditionally used by clustering algorithms.

Entities are specified in the entity column and the attributes which describe the entities constitute rest of the columns. Attributes which do not apply to the entity are left blank, or some other scheme is used depending on the requirement of the clustering process.

It is these attribute on which similarity between the entities is calculated by the similarity metrics explained below.

2.5.2 Similarity and Distance Metrics

The most important property of a clustering algorithm is that items within one cluster should be more similar to the items in the same cluster than to the items outside it. The similarity and distance measures define the criteria for measuring the similarity or dissimilarity among the items in the input data.

There are a lot of different similarity measures which compute the similarity between items based on the similarity between the selected features. A similarity measure always yields a value between 0 and 1. Two items are more similar if their similarity measure is closer to 1.

There are different categories of similarity/dissimilarity measures **[12]**: *distance measures, association coefficients, correlation coefficients* and *probabilistic similarity measures.*

- **Distance measures** measure the dissimilarity of items. The greater the outcome the more dissimilar the items are. If two items have the same value for all features, the distance between them should be zero. The most popular distance measures are the (squared) Euclidean distance and the Manhattan distance.

- **Association coefficients** calculate similarity based on the number of features present and absent in items and are suited to binary attributes. **[13]** Uses the Table 2.2 to summarize this.

	X_j		
X_i	1	0	**Sum**
1	a	b	a + b
0	c	d	c + d
Sum	**a + c**	**b + d**	**N (Grand Total)**

Table 2.2 Contingency table for binary attributes

In this table ***a*** represents the number of features present in both the entities (X_i and X_j). Analogously, **d** corresponds to the number of features absent in both the entities (X_i and X_j). **b** is the number of features present in X_i but absent in X_j, and **c** is the number of entities present in X_j but absent in X_i.

Different coefficients treat the values of **a** and **d** differently and also put different weightings on any of the four entries of the table. The most common association coefficients are.

The *simple matching coefficient*, defined as: **b+c / a+b+c+d.**

The *Jaccard coefficient*, defined as: **b+c / a+b+c.**

The *matching coefficient* treats **a** and **d** equally, both contribute to the similarity. The *Jaccard coefficient* does no take **d** matches into account at all. Therefore this measure is very well suited for asymmetric binary features. There are many association coefficients in the literature, depending on how they treat *a, b, c, d* matches and how they weight different similarities.

- **Correlation coefficients** are originally used to correlate features. For clustering source code it is inappropriate to determine similarity between attributes as apposed to entities, because clustering relies on the latter rather than the former **[14]**.

- **Probabilistic measures** are based on the idea that agreement on rare feature contributes more to the similarity between two entities than agreement on features that are frequently present. Probabilistic coefficients take into account the distribution of the frequencies of the features present over the set of entities. Probabilistic coefficients are developed to include feature distributions into similarity calculations. Given that probability coefficients are complex and must be tailored to suit a particular application, a similar function may be achieved by incorporating attribute-weight into association coefficients, which would be much simpler **[14].**

Figure 2.1 shows the similarity matrix that stores a collection of proximities that is available for all pairs of n items.

$$\begin{bmatrix} 1 & & & & \\ S(2,1) & 1 & & & \\ S(3,1) & S(3,2) & 1 & & \\ \vdots & \vdots & \vdots & & \\ S(n,1) & S(n,2) & \cdots & \cdots & 1 \end{bmatrix}$$

Figure 2.1 Similarity Matrix

Were S(i, j) is the measured similarity between items i and j. In general S(i,j) is a non-negative number that is close to 1 when items i and j are highly similar, and becomes smaller the more they differ. Since S(i,j) = S(j,i) and S(i,i) = 1, we have the matrix as shown in the figure.

2.5.3 Clustering Algorithms

There exist large numbers of clustering algorithms in the literature. The choice of clustering algorithm depends both on the type of data available and on the particular purpose and application. Traditional clustering algorithms can be broadly categorized into two main types **[15]**, *Hierarchical clustering and Partitional clustering. Hierarchical clustering* is discussed in detail as this type of clustering is more relevant to this project.

- **Hierarchical method** creates hierarchical nested partitions of the dataset, using a tree structured dendogram and some termination criterion. A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed.

  The divisive approach, also called the top-down approach, starts with all the items in the same cluster. In a successive iteration, a cluster is split up into smaller clusters, until eventually each item is in one cluster, or until a termination condition holds.

  The agglomerative approach, also called the bottom-up approach, starts with individual items at the leaves as separate clusters. It then successively merges the items close to one another, until all of the items are merged into one (the top most level of the hierarchy), or until a termination condition holds. This result's in a binary tree clusters. One advantage of the agglomerative algorithms is that they are unsupervised, they do not need any extra information such as the number of cluster expected and possible region of the search space where to look for each cluster.

Another issue of importance with hierarchical agglomerative clustering algorithm is that of updating similarity measures. The similarity metrics discussed earlier are used to determine similarity between individual entities, but there is also the need to determine the similarity between clusters containing a number of entities. The main methods that are used in such cases are ***single-linkage rules***, ***complete linkage rules, weighted-linkage rules*** and ***un-weighted linkage rules*** **[16]**.

Each one differs in the way the new similarity is computed. For example, if A, B, C are clusters, after B and C are joined, one wishes to determine the similarity between A and BUC (B union C). Assuming the similarity between A and both B and C is known, the ***single linkage rule*** would set the new similarity as the maximum of Sim(A,B) and Sim(A,C). The ***complete-linkage rule*** would set the new similarity as the minimum of these. The ***weighted-linkage rule*** would calculate the new similarity as a weighted average of these, depending on the number of objects in cluster B and C, or some other criteria such as size or importance of objects in each cluster. The ***un-weighted-linkage rule*** would calculate the new similarity as the average of these similarities.

The agglomerative hierarchical algorithm utilizes the similarity metrics described above. Entities are grouped together depending on the similarities calculated by similarity measures. This process can be visualized by a dendogram shown in figure 2.2.

The leaves represent the starting point where all the clusters contain a single entity. Moving upwards the tree depicts an increasing aggregation of entities into clusters. The finishing point is the root of the tree where all entities are in the same cluster. Entities and clusters are merged with other entities and clusters, one at a time.

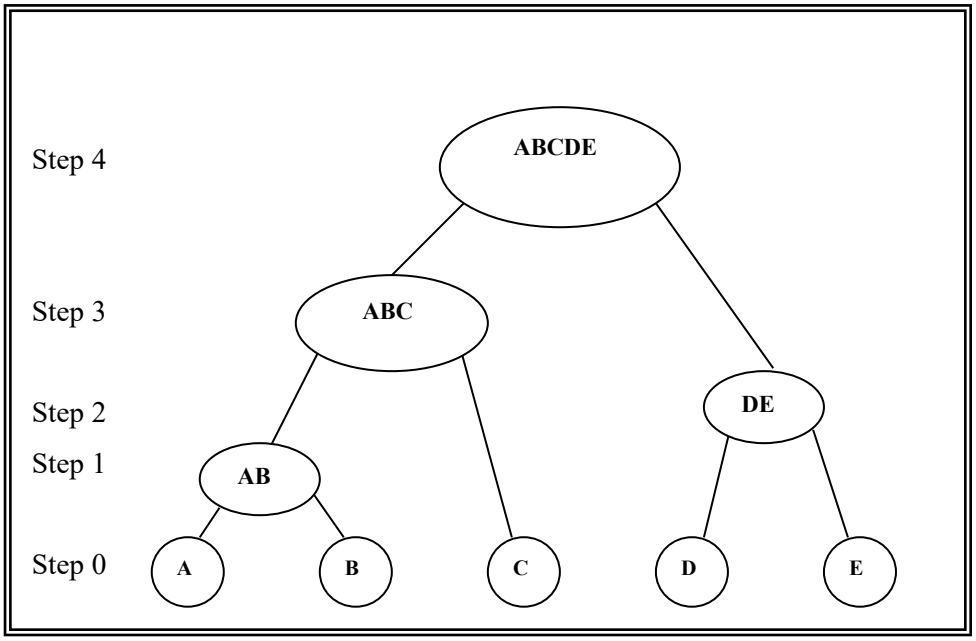


Figure 2.2 A dendogram for hierarchical clustering.

In practice, if the clustering process is allowed to continue until all the entities are combined in one big cluster, then this may result in *forced clustering*, where entities may be forced into a cluster even though they are highly dissimilar to the entities in that cluster. This unwanted result is avoided by specifying a *similarity threshold* or a *cut-point,* where the clustering process is stopped if the maximum similarity between two clusters does not exceed the threshold value.

- **Partitioning method** starts with an initial partition (clustering), in which entities are moved to other clusters in order to improve the partition according to some criterion. This relocation goes on until no further improvement of this criterion takes place. The final clustering is largely based on the initial partitioning.

2.6 Clustering a software system

Reverse engineering tries to help software engineers understand a presumable large piece of software. A key activity in reverse engineering consists of gathering the software entities (modules, routines, and so on.) that compose the system, into meaningful (highly cohesive) and independent (loosely coupled) groups. This activity is called clustering of software system. **[1]**.

There are two common ways to approach the problem of identifying clusters in a software system **[17]**:

1. **Knowledge-based approach**

   Using such an approach, one attempts to understand what different pieces of source code do by utilizing reverse engineering techniques and pre-existing domain knowledge. Program modules that implement similar or complementary functionality can then be grouped together, for example, procedures implementing mathematical functions can be assumed to be part of the same library.

2. **Structure-based approach**

   In this case, the decomposition of a software system is determined by looking at syntactic interactions (such as "call" or "fetch") between entities (such as procedures or variables). The problem of clustering a software system can be thought of as the partitioning of the vertex set of a graph, where the nodes are defined as procedures or variables, and the edges as relations between these entities.

Although knowledge-based approaches have been shown to work well with small systems, they do not perform as effectively when dealing with large ones. Various reasons, such as the size of the knowledge base becoming prohibitively large, the lack of problem domain specific semantics, and knowledge spreading in the source code contribute to this phenomenon. The majority of software clustering researchers has concentrated on structure-based techniques.

Most of the structure-based clustering activities in reverse engineering are based on the following four issues.

1. **Input Model**

   This is concerned with the specification of program entities and their attributes. It is these entities that are then grouped into clusters (subsystems).

   For software remodularization, entities may be files, routines, classes, processes, and so forth. Multiple attributes can be used to describe entities, for example, for functions the attributes can be their names, return types, parameters, variables they

use and so on. Similarly for classes the attributes can be their base class, the data types and the functions they have and so on. Similarity measures use these types of attributes (depending on the entities) to calculate the similarity between entities.

2. Similarity Metrics

After defining the input model, the next issue is to define the criteria for similarity between the entities. That is, what should be the measure of determining the similarity between the program entities?

Similarity matrices are used to define those criteria and to calculate the similarity between the entities.

3. Clustering algorithm

The next step is to apply a given clustering algorithm to meaningfully group entities into subsystem.

4. Evaluation

After performing the clustering the last step is to evaluate the clusters produced.

One of the most important requirements for any software clustering algorithm is that the clusters produced should actually represent the system. Ideally the results of the clustering algorithm should match the decomposition of the software presented by an expert (the partitioning done by an expert of the code).

Precision, recall **[1]**, *MoJo distance* **[18]** *and Edge similarity measurements* **[19]** are some of the evaluation techniques used by the software clustering community in order to evaluate their clustering results. A brief description of these techniques is given below. For further details consult the given references.

- *Precision* is the percentage of intra-pairs* proposed by the clustering method, which are also intra in the expert partition. *Recall* is the percentage of intra-pairs in the expert partition, which are also intra in the partition produced by clustering algorithm.

*Intra-pair: If two entities are in the same cluster they are called intra-pair.

In other words *precision* of a subsystem is the percentage of entities in the subsystem (produced by clustering algorithm) that are also in the subsystem according to the expert mental model and the *recall* of a subsystem is the percentage of entities in the subsystem (of expert's mental model) that are also in the subsystem according to the clustering tool.

- *MoJo* measures the distance between the two decompositions (decomposition by the software and the one done by an expert) of a software system by calculating the number of operations needed to transform one decomposition into the other. The transformation process is accomplished by executing a series of *Move* and *Join* operations.

Both the measurements described above only consider the assignment of source code components to clusters as the only criterion for similarity. The *Edge similarity measurements* also take into account the relations between the components as well.

2.7. Related Work

The use of data mining for system comprehension, and eventually for software maintenance has been considered in the past. Considerable amount of experiments have been conducted, often resulting in notable results. Some of the work done in this field is summarized below.

[20] [21] approaches addressed C/C++ and Java code respectively. These systems aimed at understanding low/medium level concepts and relationships involving components at class and function level.
The basic steps followed by these two approaches are:

- Definition of an input model, to extract code and populating a database**.**
- Clustering application is then applied to the pre-processed data**.**
- Result evaluation (comparison of the result with experts).

[20] uses functions, classes, function-parameters and member data as entities to be clustered for C/C++ programs. Similarly **[21]** uses classes, packages, functions, and function parameters as entities to be clustered for java programs.

There are few shortcomings of the above two approaches. Both of them use very few attributes to be used for function or class clustering. There are many other features of functions and classes that need to be considered in order to enhance the functionality of the clustering process. (These features are discussed in later chapters of this report). Furthermore, **[21]** Uses a commercial clustering tool rather then custom designed algorithms.

[14] deals with clustering C/C++ source code. It uses functions as program entities for clustering purposes. The attributes used for calculating similarity between functions are: the use of global variables, function local variables, function parameter and return types.

This project uses many of the algorithms mentioned in this work with very few changes. All the algorithms devised in **[14]** are mentioned in the Appendix II.

[22] Uses a hill-climbing algorithm, trying to minimize an object function that subtracts the average "inter-connectivity" (Bunch's measure of coupling) of the partition to average "intra-connectivity" (Bunch's measure of cohesion). **[23]** Propose an approach, where architectural design recovery is based on design descriptions that are provided by the user in the form of queries. A language AQL (architectural query language) was formalized for this purpose.

[24] Present an approach for the evaluation of dynamic clustering. One of the main features of dynamic dependency graphs is that they are weighted. Although weights can be assigned to static dependencies as well, the variance of dynamic weights is significantly larger. This experiment attempts to evaluate the usefulness of providing dynamic dependencies as input to software clustering algorithms. Both static and dynamic dependencies are provided as an input to the clustering algorithms and the results produced by them are compared.

[17] Is a pattern-based software-clustering algorithm that attempts to recover subsystems commonly found in manually created decompositions of large software systems.

[25] Proposes a "shared neighbors" technique in order to capture items that appear commonly in software systems, also the "maveric analysis" enable the clustering tool to refine a partition by indicating components that happen to belong to wrong subsystem, and placing them in the correct one.

The clustering tool developed for this project resembles **[20]** **[21]** and **[14]** with considerable improvements. There are significant amount of changes made to the input model in order for this clustering tool to perform better and secondly unlike the **[21]**, this clustering tool uses a custom designed clustering algorithm instead of merely using a commercial tool.

In its future work, **[20]** suggests the use of weights for attributes to reflect their relative importance. The use of weights has been incorporated in this work in order to signify the relative importance of the attributes.

This work uses the attributes used by **[14]** (for function clustering) along with few new ones, which include function properties and function calls. The details of which are given in the design and implementation phase of the project. In addition to that, classes are also used as program entities for clustering, in order to make the tool more effective for object oriented software, as most of the object oriented programs use classes, and classes can encapsulate a single specialized functionality, perform a cohesive specialized task (the abstraction level may be higher, as compared to the functions), and are clearly defined with in a program.

3

Requirement Analysis

Software requirements express the requirements and constraints on a software product that contributes to the solution of a problem in the real world. The Requirements Gathering is a very important phase of any software product. This involves various techniques like “interviewing” or “market analysis” and so forth. In case of the clustering tool, the author has a well defined task with a set of fully elaborated requirements. Mostly, the requirements are collected through discussion with the project supervisor. However, to make the tool more useful and user-friendly, requirements have been collected from various discussion groups. The scope of the project has been continuously revised and changes have been made in it during the project design and implementation phase to enhance the functionality and usefulness of the project.

The aim of this project is to develop a clustering tool that is able to decompose a software code into meaningful clusters. The tool will try to offer ways of understanding and improving the source code. The design of the tool allows it to perform cluster analysis on the code written in many different languages (for example C++, C#, Java and so on) with little or no changes. The project is about producing a tool for automated approach to program understanding. Therefore it is assumed that the user has no expert knowledge of the program being analyzed.

A detailed description of requirements including functional and non functional requirements is given in this section.

3.1 Functional Requirements

Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform

[26]. The functional requirements of this project are: (The functional requirements are presented in the descending order based on their priority for this project).

1. **Specification of the input model**

   It involves defining an input model needed to extract data from the source code and populate a database. This is concerned with the specification of program entities (functions, classes and so on) and their attributes. Program entities must have several attributes that provide a basis for measuring similarity between entities.

   The input model must be able to satisfy the following requirement that could lead to satisfactory performance of the clustering methodology **[14]**.

   - The entities defined should be homogeneous in nature, thus allowing for description by a common set of attributes. This would facilitate entity comparison by use of their attributes, which is the basis of cluster analysis**.**

   - Entities must have a sufficient number of features to provide informative description about them and also present a means of comparison among the entities. A very small number of features may result in providing very little information content and possibly leading to deceptive conclusion. On the other hand (unnecessarily) increasing the specificity would lead to more detailed description of the entities but would lead to less effective cluster analysis**.**

   - The attribute values of the entities must be clearly defined for any given program. Additionally, the definition of an entity must be universally applicable to programs in order for the methodology to be widely effective. Entities, attributes and their values must be easily detected**.**

   - The choice of entities should be such that when the program is abstracted as a collection of entities, an appropriate majority of the program code is associated with an entity. This ensures that the majority of the program is

covered by the analysis, even though the analysis may show that some program entities do not belong to a subsystem**.**

The input model should be defined in such a way that it can be easily applied to many programming languages (like java, c# and so on.) with little or no changes.

2. Specification of the similarity metrics

The similarity metric determines the degree of similarity between program entities. The choice of a proper similarity metric can have more influence on the result then the clustering algorithm itself **[1]**. Since similarity is fundamental to the definition of a cluster, a measure of the similarity between two entities drawn from same feature space is essential to the clustering algorithm.

Because of the variety of feature types, scales, and difference in their relative importance, special care should be taken in specifying the similarity metric. The similarity metric should be customized in order for it to be best suited for the project's requirements (That is, the similarity metric should clearly and correctly define similarities and difference among the program entities that are being populated in the database). The main requirements regarding similarity metrics are **[14]**.

- The similarity metric must be suitable for comparison for binary, numerical and categorical data, as these are the types of attributes largely predominant in a source code application domain.

- The similarity metric must take into account the relative importance of attributes, as the presence of some features may be more significant than the presence of others.

- The similarity metric must consider the distribution and rarity of a feature throughout the whole set of entities.

- The similarity metric must normalize measure of similarity taking into account the probability of a match between attribute values of two entities. This is because the significance of a match on an attribute depends on the characteristics of the complete set of attributes.

3. Specification of the clustering algorithm

The clustering algorithms actually group the program entities into meaningful clusters, depending on the similarity values provided by the similarity metrics. The main requirements for the clustering strategy are as follows:

- The clustering technique must create several solutions to the problem, as apposed to a single clustering distribution. Ideally it should form clusters incrementally so that if the eventual result is not correct to any suitable extent, then the exact step for where the results started to deviate from the original solution can be pin pointed.

- The clustering technique must not require the final number of clusters to be specified.

- The clustering technique must not rely on an initial partitioning to be made to the entities by the user, as this is only feasible when the user has some idea of the nature of the required clustering. Furthermore, the initial partition may influence the final result in an undesirable manner.

4. Actual representation of the system

It is the one of the most important requirement for the clustering tool. The clustering tool should actually represent the system (and not the ideal view of the domain for example). Ideally the results of the clustering algorithm should match the decomposition of the software presented by an expert.

5. Design

Clustering should reflect a good design, a design that makes sense to the software designer, that is, the tool should extract modules that implement known concepts.

6. **Ability to handle different types of attributes**

   Many algorithms are designed that work well on particular type of data (for example, numerical data) only. However in this project, which involves source code, the clustering tool is required to handle other different types of data as well, such as binary, categorical (nominal), and ordinal data, or mixtures of these types of data.

7. **Insensitivity to the order of input records**

   Some clustering algorithms are sensitive to the order in which the input data is presented to them. They behave differently and produce different results (clusters) when given the same data in different order. The requirement is to develop a clustering tool that produces the same result regardless of the order in which the data is being provided.

8. **High dimensionality of the input data**

   Most Clustering algorithms generally work well on low dimensional data, that is, data items containing fewer (2 or 3) attributes. In the case of this project an item can have several attributes, depending on the structure and functionality of the item. (For example, if a *class* is taken as an item or an entity, it can have several attributes like “classes it inherits from”, its “protected data members” “private data members” and “public data members”, its “public “ “private” and “protected” member functions and so on). Therefore the challenge for the clustering algorithm in this work is to effectively handle data items in high-dimensional space.

9. **Prescribed Methodology**

   The software should aid the user to follow step-by-step data mining methodology to help avoid spurious results. The tool should form clusters incrementally, which will help the user to detect (in case of a problem) where the solution starts to deviate from the actual “expert decomposition”.

10. **Reporting**

    The output of the data-mining tool should be presented as detailed results in variety of ways (textually and graphically) that will help in understanding the

system in a better way. The tool should provide summary as well as detailed results.

11. Model exporting

It will really be facilitating that the clustering tool provides ways to export the results produced by it for on going use (for example, C program, SQL and so on.).

12. Model Validation:

Model validation and verification are important tasks in any clustering methodology. The tool should provide model validation in addition to model creation. There are number of techniques that should be used to validate the model produced by the clustering software.

13. Data Filtering

The tool should allow the selection of subsets of the data based on user-defined selection criteria.

14. Error reporting

Meaningful error reporting is important for the clustering tool. The software should report error messages in a way that should help in the debugging process.

15. Handling noise

It would be better to produce an algorithm that can detect and handle noisy data, so that the incorrect data may not result in the poor quality of the clusters being produced.

16. Data cleansing

The tool should allow the user to modify spurious values in the data set. The tool will be dealing with the entities extracted from the source code, therefore the quality of the data will be the primary requirement of the software.

3.2 Non-Functional Requirements

Non-functional requirements are constraints on the services and functions offered by the system. The non-functional requirements of this project are: (The non-functional requirements are presented in the descending order based on their priority for this project).

1. **Robustness**

   The tool should be able to run consistently without crashing. The tool should not require monitoring and intervention by the user.

2. **Cluster Size**

   Clustering should avoid solutions having many singleton clusters or only one huge cluster. Since hierarchical clustering algorithm is used for clustering in this project therefore special care must be taken in order to avoid situations where the algorithm tend to create one big cluster that grows regularly along the clustering process and drag all the entities to it one after the other, or the situation in which the algorithm tend to create very small clusters and then suddenly cluster all of them into one cluster at a higher height for a given similarity metrics.

3. **Automation**

   Many clustering algorithms require some type of user input, such as defining the initial cluster, mentioning the desired number of clusters and so on. The clustering tool developed for this project should be able to work well without requiring the user to have any domain knowledge of the system to be clustered.

4. **User Interface and learning curve**

   The user interface should be easy to navigate and uncomplicated, and the result should be produced in a meaningful way. Similarly the tool should be easy to learn and easy to use correctly.

5. **Stability**

   The clustering tool is stable, if the clusters obtained by it are not significantly affected by slight modification in its input, that is, the software system in question

[27]. In other words if there are slight changes in the software (being clustered), then the clusters obtained as a result should not be significantly different to the clusters obtained previously. The software tool should be checked for stability, so that the results produced by the tool can be of practical use.

6. **Scalability**

A clustering algorithm should be scalable. There are many different clustering algorithms that work well with the small data sets containing few items, but as the amount of the data increases their efficiency is severely affected. Numerous tests should be made on the clustering tool in order to check its efficiency and its ability to handle large data set.

7. **Interpretability and usability**

The clustering tool should be able to produce results in a way that are easily interpretable, comprehensible and usable. In the case of software comprehension this property of the clustering algorithm is essential, as the clustering tool should be able to present results that are understandable to both, an expert user and a novice.

8. **Efficiency**

A data-mining tool is efficient if the results it produces are in reasonable amount of time, its memory and hard disk space requirement are reasonable relative to the data size.

The clustering tool should be efficient in order for it to be useful in any actual maintenance work.

9. **Adaptability**

The clustering tool should be adaptable to the different needs posed by different users. For example it should help a novice understand the overall system, and help an expert to assert the consequence of a modification in the code. In addition the tool should be adaptable to different software, programming languages, and so on.

4

Design

This chapter gives a detailed explanation of the design of the software, the database, and the input models used for clustering of the source code. All the aspects of the software are discussed in details along with the UML and block diagrams for clear understanding of the project.

4.1 Input Models

One of the main challenges of this work is the definition of the input models that should be derived from a program code. Entities must be defined in such a way that provides useful and meaningful information for the clustering algorithms to perform their tasks.

Classes and functions in a source code are selected as entities to be clustered. They fulfil most of the requirements expected of the input model for code clustering.

The specification of the input models for both functions and classes is given in the sections below.

4.1.1 Function Input Model

Functions are considered to be an appropriate option to perform code clustering as they tend to encapsulate a single functionality, perform specialized cohesive tasks and they are clearly defined within a program.

The next step now is to derive attributes with which to describe this program entity. Figure 4.1 depicts the diagram for the proposed function input model.

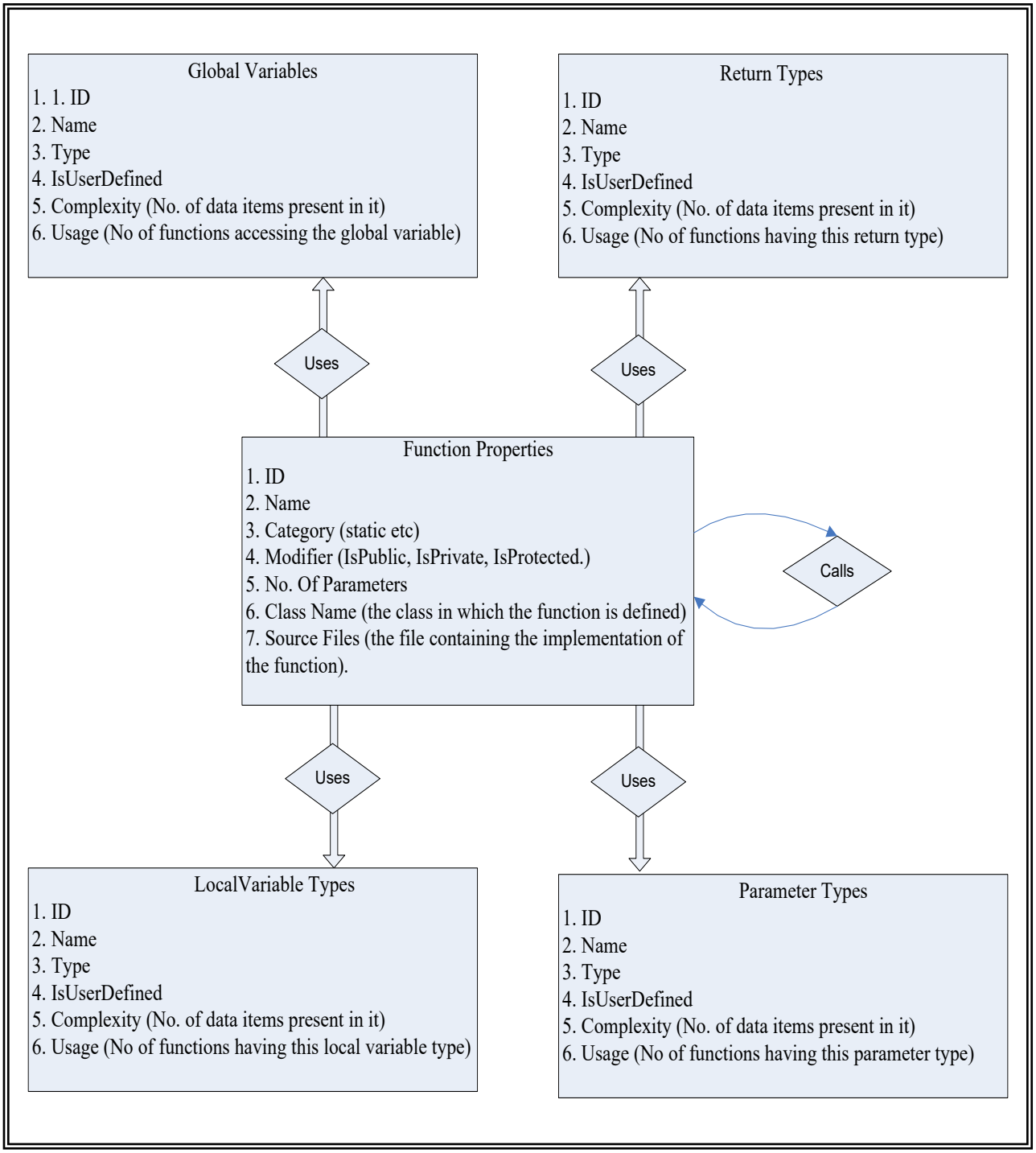


Figure 4.1 Function Input Model.

The similarity between functions is determined, using the attributes shown in the input model.

In brief, the similarity between the functions is calculated on the basis of

1. The function properties.
2. The functions use of global variables.
3. Use of the local variables.
4. Return types.
5. Parameter types.
6. The functions they call.

The rationale behind this selection is given in the following:

1. Functions use of global variable

The usefulness of a global variable as an attribute depends on its nature, which is largely influenced by its type. The most effective global variables are likely to belong to the user-defined types. The reason behind this statement is that most user-defined types are specified to model a particular aspect of the problem domain, and the functions that use these are likely to be associated with this aspect of the problem domain, indicating the membership of a common sub-system. Thus the differentiation whether the type of an attribute is user-defined or pre-defined is considered to be necessary and is also used for other types of attributes, where applicable.

2. Function use of local variable

The idea behind using local variables is that, the overall purpose of a function can be predicted by examining the nature of the data items on which it operates. The nature of the data items can be determined by their types. It follows that the functions that operate on several common variable types are likely to have similar purpose and hence are likely to belong to a same subsystem.

3. Function parameter types

The concept behind using parameter types for calculating function similarity is same as that for local variables. However it will not be useful for functions that do not have parameters.

4. **Functions return types**

   It follows the same concept as that for parameter types.

5. **Function calls**

   Function calls may also provide vital information in predicting the similarity between functions. It is hypothesised that the functions calling several common functions are likely to have a similar purpose, hence may belong to the same subsystem.

6. **Function Properties**

   The properties of functions like their names, Category (static and so on), modifier (like Public, Private, Protected), the class they belong to and the source file in which they implemented may play an important role in determining the similarities between any two functions. Therefore these properties were added to the input model in order to determine their importance in understanding the software system.

4.1.2 Class Input Model

Other entity used for clustering is the classes defined within the program. Figure 4.2 depicts the diagram for the proposed class input model.

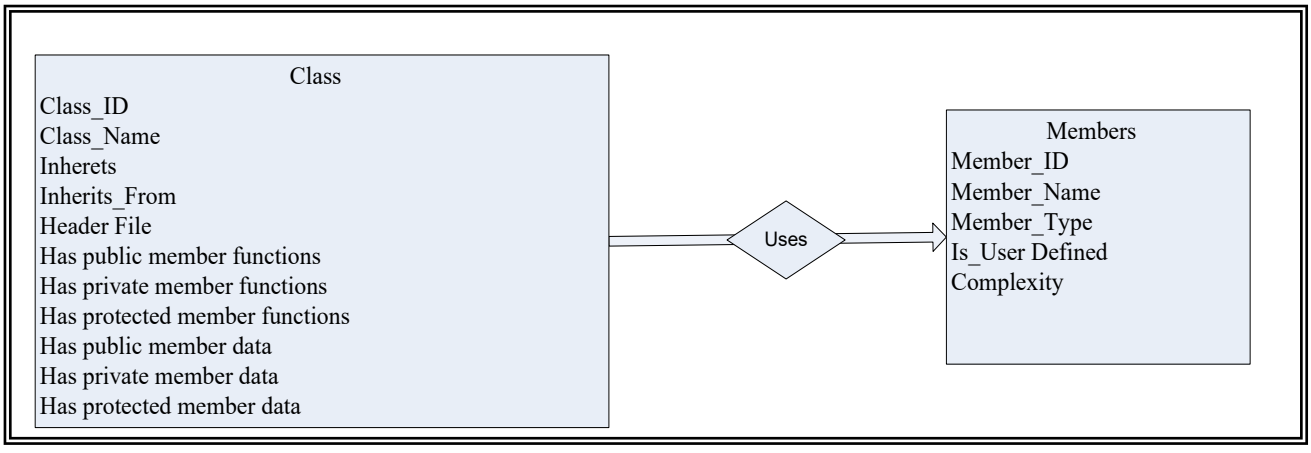


Figure 4.2 Class Input Model.

The similarity between the classes is on the bases of

1. Class Properties.
2. Class Member Variables.

1. **Class Member Variables**

   It is hypothesised that the classes having similar member data types are likely to have similar purpose and hence may belong to a same sub-system.

2. **Class Properties**

   The properties of class like their name, inheritance status, the class from which they inherit, the header file in which they are defined, their public, private, protected members and functions play an important role in determining the similarities between any two classes. Therefore these properties are chosen to be catered for when determining the similarity between classes.

4.2 Similarity Metrics

This section defines the formulas and the equations that are used to calculate similarity between program entities (that is, functions and classes), on the basis of their attributes.

4.2.1 Similarity Principles

There are three principals on the basis of which the similarities between entities are calculated **[14]**. They are

1. **Basic Principle**

   This principle uses the concept of Jaccard coefficient (explained in the background study) to calculate the similarity between two entities.

   This principle basis the similarity between the two entities on the number of attributes common between them. The greater the number of common attributes between two entities (relative to their total number of attributes) the greater is the similarity between them.

2. **Usage Principle**

   Usage principle is based on the relative importance of the attributes of the entities that are compared. The importance is based on the frequency-of-use of the attributes. Some attributes are more common in all the entities as compared to

others. For example, there may be some variables that are used by many functions in a program than the others. In that case the variables that are less common in the program are given more weight then the variables that are more frequently used.

3. **Complexity Principle**

   This principle is based on the relative importance amongst attributes, and depends on the type of the variables on which an attribute is based. The exact importance of any variable type can only be known to the programmer and therefore cannot be determined without the help of an expert. Therefore the concept used here is to give greater significance to more specialized or complex types. That is, greater the number of data items present in any complex variable type, the greater is its significance.

4.2.2 Function Similarity Metrics

The overall formula that is used for calculating the similarity between the two functions is.

Function Similarity S_{TOTAL}:

$$\frac{[W_{GV}.S_{GV} + W_{FC}.S_{FC} + W_{LV}.S_{LV} + W_{PU}.S_{PU} + W_{FP}.S_{FP} + F_{RT}.S_{RT}]}{[W_{GV} + W_{FC} + W_{LV} + W_{PU} + W_{RT} + W_{FP}]}$$

Where:

W_{XX} stands for the weight factor that is applied to show the relevant importance of the metrics specified by S_{XX}. These weights are provided by the user as there importance may vary from program to program and they also help in the evaluation and testing purposes. The software sets the default values in case the user does not fill the weights. The values of all the similarities are between 1 and 0 inclusive. Similarly the value of S_{TOTAL} is also between 1 and 0 and the division factor is applied just for this purpose.

The explanation of the terms used in the above equation is given below.

S_{GV} = Similarity based on the use of global variables.

S_{FC} = Similarity based on function-calls.

S_{LV} = Similarity based on local variable types.

S_{PT} = Similarity based on parameter types.

S_{RT} = Similarity based on return types.

S_{FP} = Similarity base on functions properties.

As previously mentioned, the similarities based on global variables, local variables, parameter types and return types are calculated by using equations very similar to **[14]** with minor changes. These equations are mentioned in the Appendix II.

Similarity Based on Function Properties (S_{FP})

The equation used for calculating the similarity between the two functions based on the function properties is given below.

$$S_{FP} = a_{fp} / a_{fp} + b_{fp} + c_{fp.}$$

Where

a_{fp} = Number of common properties among the functions.

b_{fp}, c_{fp} = Number of dissimilar properties among the functions.

This similarity is based on *basic similarity principle* (mentioned in 4.2.1).

Similarity based on function-calls (S_{FC})

Similarity based on function-calls is given as:

$$S_{FC} = \frac{S_{Basic} + S_{usage}}{2}$$

Where S_{Basic} and S_{usage} implement *basic* and *usage similarity principles* respectively. The division factor is to keep the similarity value between 0 and 1.

Basic similarity metric (S_{Basic}) is given as

$S_{Basic} = a_{fc} / a_{fc} + b_{fc} + c_{fc.}$

Where

a_{fc} = Number of function-calls made by both the functions being compared.

b_{fc} , c_{fc} = Number of function-calls not common in the two functions being compared.

This metric gives an elementary indication of the similarity between any two functions. It is simply the ratio of the function-calls common to the two functions (That is, the functions being compared) to the total number of their function-calls.

A similarity based on *Usage principle* is given as:

$$S_{usage} = \frac{\sum_{1}^{a_{fc}} 2 \,/\, \text{No. of functions calling the particular function.}}{a_{fc}}$$

Thus if a function is called predominantly by the two functions being compared, then it contributes more to the similarity metric as compared to the function that is used by many other functions. If a function is called exclusively by the two functions that are being compared, then this function will produce maximum contribution to the above metric.

4.2.3 Class Similarity Metrics

The overall formula that is used for calculating the similarity between the two classes is.

Similarity of Classes S_{TOTAL}:

$$\frac{[W_{MV}.S_{MV} + W_{CP}\,S_{CP}]}{[W_{MV} + W_{CP}]}$$

Where:

W_{XX} stand for the weight factor that is applied to show the relevant importance of the metrics specified by S_{XX}. The Values of the entire S is between 1 and 0 inclusive, similarly the value of S_{TOTAL} is also be between 1 and 0 and the division factor is applied exactly for this purpose.

S_{MV} = Similarity based on member variables type.

S_{CP}= Similarity based on the class properties.

Similarity Based on Class Properties (S_{CP})

The equation used for calculating the similarity between the two classes based on the class properties is given below.

$$S_{CP} = a_{CP} / a_{CP} + b_{CP} + c_{CP}.$$

Where

a_{CP} = Number of common properties among the classes.

b_{CP}, c_{CP} = Number of uncommon properties among the classes.

This similarity is based on *basic similarity principle* mentioned above.

Similarity based on Member variable types (S_{MV})

Similarity between classes based on member variable types can be calculated as:

$$S_{MV} = W_{UMV}.S_{UMV} + W_{PMV}.S_{PMV} / W_{UMV} + W_{PMV}.$$

Where:

W_{XX} stands for the weight factor that is applied to show the relevant importance of the metrics specified by S_{XXX}.

S_{UMV} = Similarity based on user defined member variable type.

S_{PMV} = Similarity based on predefined member variable type.

User Defined member variable types (S_{UMV}):

Similarity for user-defined variable types is given as

$$S_{UMV} = \frac{S_{Basic} + S_{complexity} + S_{usage}}{3}$$

Where S_{Basic}, S_{usage} and $S_{complexity}$ implement *basic, usage and Complexity similarity principles* respectively.

$$S_{Basic} = a_{umv} / a_{umv} + b_{umv} + c_{umv.}$$

Where

a_{umv} = Number of (user-defined) member variable types used by both the classes being compared.

b_{umv} , c_{umv} = Number of (user-defined) member variable types not common in the two classes (The classes being compared).

Similarity metric based on *complexity principal* is given by

$$S_{Complxity} = \frac{\sum_{1}^{a_{umv}} \text{No. of entities in a MVT / max. No. of entities in a MVT.}}{a_{umv}}$$

[MVT = Member variable type]

The contribution of an attribute to this metric is higher if the attribute is based on a more specialized type, that is, the one with more data members.

Similarly similarity based on *usage principle* is

$$S_{usage} = \frac{\sum_{1}^{a_{umv}} \text{2 / No. of classes using the member variable type.}}{a_{umv}}$$

Pre-Defined member Variable types (S_{PMV}):

Similarity for Pre-defined variable types is given as

$$S_{PMV} = \frac{S_{Basic} + S_{usage} + S_{complexity}}{3}$$

Where S_{Basic}, S_{usage} and $S_{complexity}$ implement *basic, usage and Complexity similarity principles* respectively.

$$S_{Basic} = a_{pmv} / a_{pmv} + b_{pmv} + c_{pmv.}$$

Where

a_{pmv} = Number of (pre-defined) member variable types used by both the classes being compared.

b_{pmv} , c_{pmv} = Number of (pre-defined) member variable types not common in the two classes (That is, the classes being compared).

Similarity metric based on *complexity principal* is given by

$$S_{Complxity} = \frac{\sum_{1}^{a_{pmv}} \text{No. of entities in a MVT / max. No. of entities in a MVT.}}{a_{pmv}}$$

[MVT = Member variable type]

Similarly similarity based on *usage principle* is

$$S_{usage} = \frac{\sum_{1}^{a_{pmv}} 2 \text{ / No. of classes using the member variable type.}}{a_{pmv}}$$

4.3 Clustering Algorithms

Hierarchical (agglomerative) clustering algorithms are incorporated in the clustering tool to perform the clustering task in this project. There are different advantages of agglomerative hierarchical clustering algorithms over others, which make them a better choice for performing clustering in this type of application.

- Hierarchical clustering algorithms create several solutions to the problem, as opposed to a single cluster distribution**.**

- They form clusters incrementally, as opposed to a single step. The steps involved in the incremental formation of the clusters are small which helps in the debugging of the process in case the clustering algorithm does not

produce a suitable outcome (for example, it is easy to detect where the algorithm actually start to deviate from the actual result given by the software expert).

- Hierarchical clustering algorithms are unsupervised, they do not need any extra information such as some initial partitioning being made by the user or the mentioning of the number of clusters expected**.**

4.4 Block Diagram

Figure 4.3 shows the block diagram (of this project), that explains the overview of the system and indicates the overall design and functionality of the software. A brief description of this diagram is as follows:

The user interface provides the user with plenty of options (these options are discussed in the explanation of the user interface in the latter sections). At first the user has to select the database containing the (function or class) input model. The user can then choose to perform either function or class clustering (one at a time). Then comes the similarity matrices. The similarity between functions or classes (depending on the user selected options) is calculated by using the similarity principles and equations mention in the previous sections. These similarities are then passed on to the clustering algorithms. As mentioned before, agglomerative hierarchical clustering algorithms are used for clustering purposes. Single-Link clustering and Complete-Link clustering will calculate similarities between the clusters (see section 2.5.3 for further details). The clusters produced are then passed on to the module that is concerned with the display and storage of the results. The results are stored in text files, the details of which are explained in the latter sections.

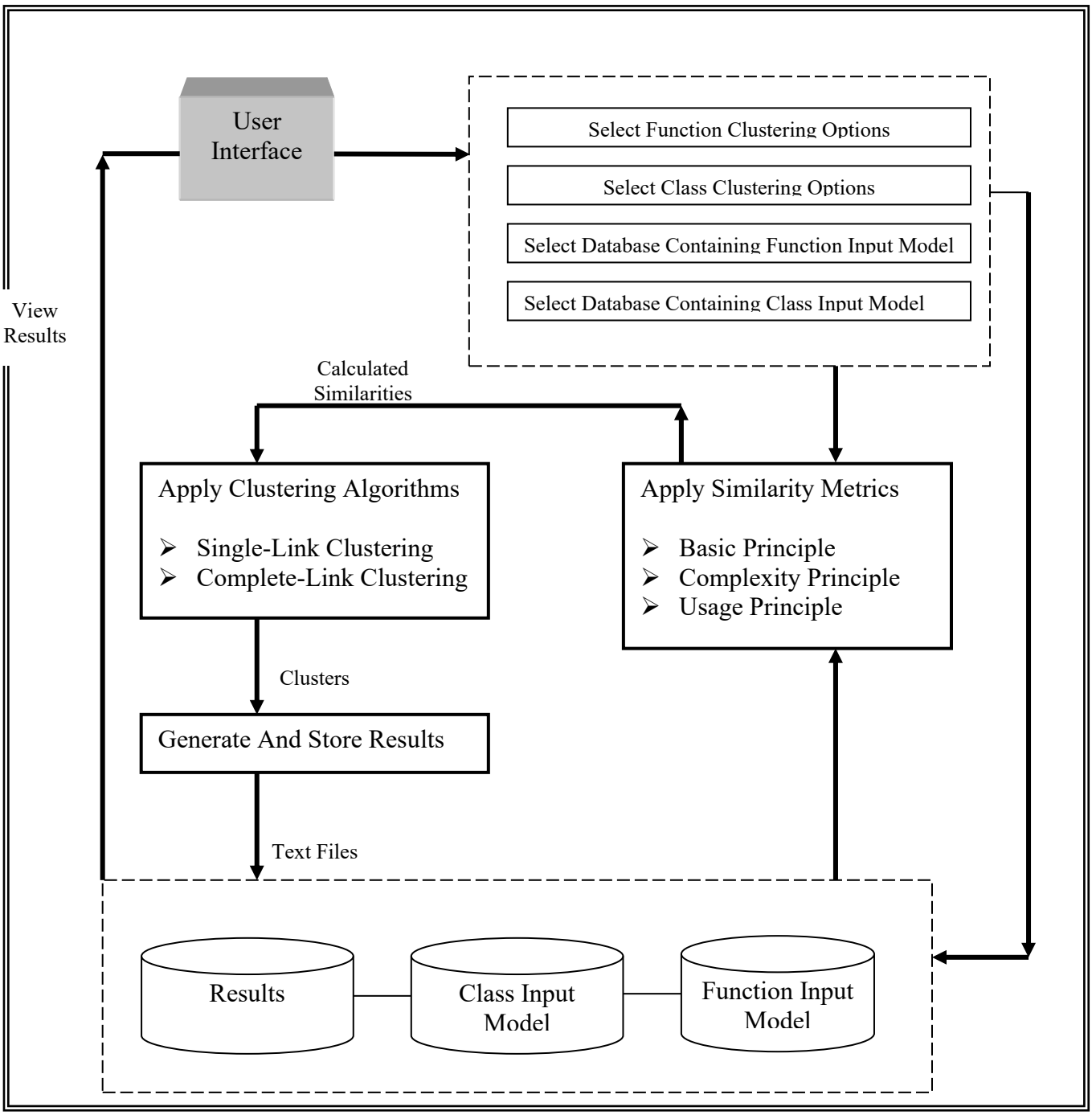


Figure 4.3 Block Diagram (Code Clustering Application)

4.5 Use Case Diagram

Figure 4.4 shows the Use case diagram of the project.

As mentioned previously the user has to select the database which contains the function or class information. The database contains the information about the functions and the classes in the format shown previously in the input models. The actual data fed into the database is through the parser which is not the part of this project. The code to be clustered is first parsed by the parser and the information is fed into the database with the help of this parser. The code clustering application then uses this information to perform clustering. Although the parser is not fully capable of parsing all the information need by the input model, there are some database operation that are done manually to fill all the information in the database, according to the input model created by the author.

The user now has the option to perform function and class clustering. There are numerous options given to the user in order to get the desired results. All these options are discussed in details, in the section dealing with the user interface of this project.

After the clustering process the user can see the results and then evaluate the clustering process by comparing its results with the results given by the expert.

The output generated by the clustering application is in the form of text files. Although these files contain thorough information of each step performed during the clustering process. It still is much easier to understand the process, if there is some sort of visualization tool that converts the text file into a visual form.

Generate Dendogram is another application that uses the text file generated by this application and then generates a visual form of the clustering process indicated in the text file.

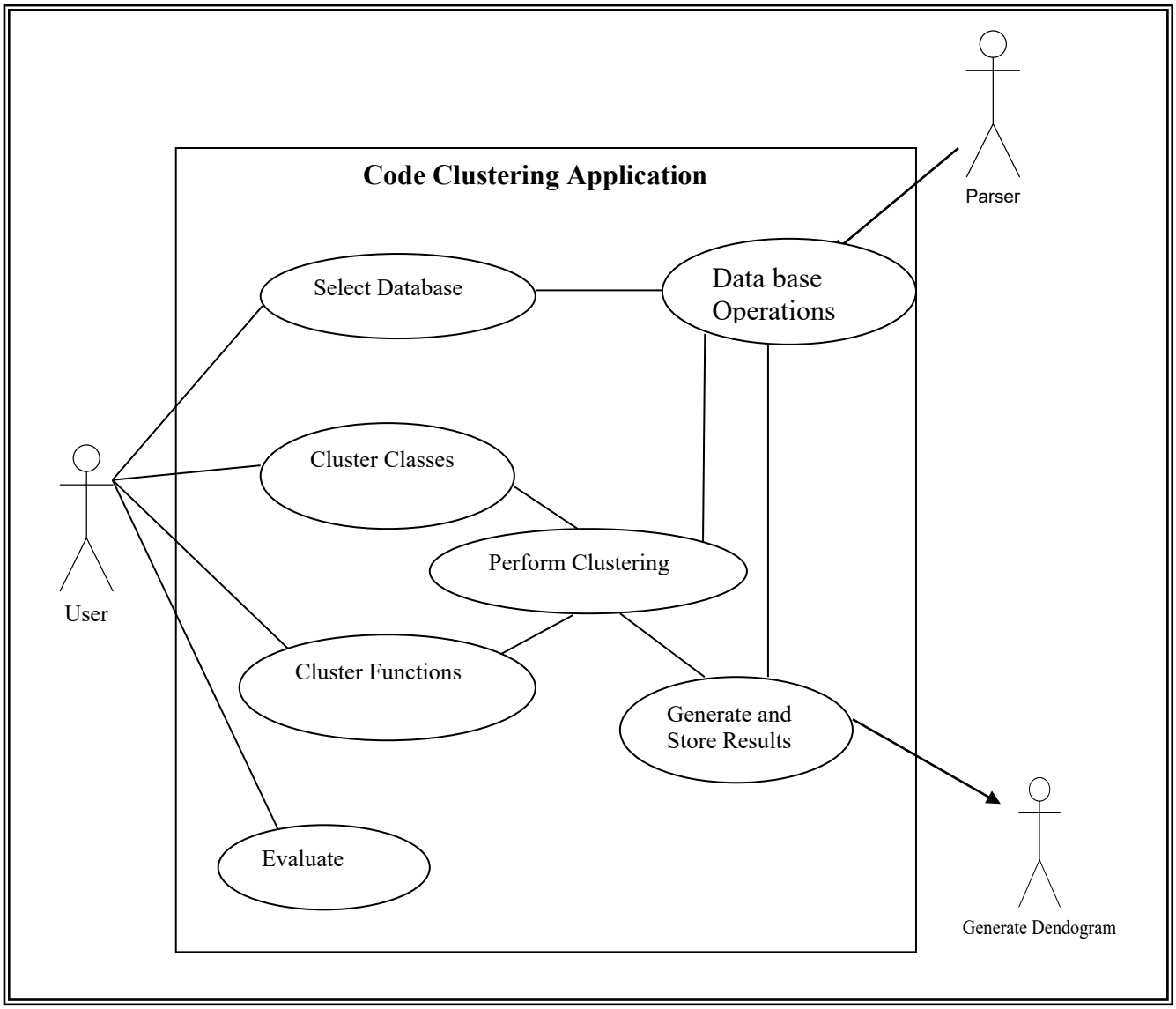


Figure 4.4 Use Case Diagram (Code Clustering Application)

USE CASES

Select Database This use case is activated when the user wants to select the database which contains the parsed information about the software to be clustered. The software tool then uses this data base as a source of information for clustering the software entities.

Cluster Classes This use case is activated when the user wants to perform Class clustering. Numerous options are provided to the user in order to perform clustering based on different similarity functions and principles.

Perform Clustering This use case is extension to the Cluster Classes and Cluster Functions use cases, and is responsible for actually performing the clustering operations on the basis of principle and similarities calculated by these two use cases.

Data base Operations This use case extracts the records from database and provides the required information from the database.

Cluster Functions This use case is activated when the user wants to perform function clustering. Numerous options are provided to the user in order to perform clustering based on different similarity functions and principles.

Generate and Store Results This use case actually saves all the resulting data in the format that is suitable for other applications to perform the visualization of the results produced.

Figure 4.4a Use Cases(Code Clustering Application)

ACTORS

User It represents the end user of the clustering tool.

Parser It represents another application that is actually used to partially fill the database for the code clustering tool. It is a parser that is used in order to parse the software (to be clustered) and to provide the information (in the database) required by the clustering application. Although the parser is used to partially fill the database, the rest of the work has to be done manually. This is one area where further work needs to be done

Generate Dendogram This is another application that uses the output provided by the clustering tool and provides a visual form of the output, in order to help better understand the clusters produced.

Figure 4.4b Actors (Code Clustering Application)

4.6 Class Diagram

Figure 4.5 shows the class diagram of the entire project. All the major classes and the relationships between them are indicated. The class responsible for user-interface (Form) is shown only to indicate its presence. The details and the relationships of this class are not shown in order to make the diagram simpler and easy to understand. Similarly the data type *"user provided info"* contains all the selections made by the user while interacting with the interface of the program. It is a very large data structure, the complete details of which are mentioned in the implementation section. Its details are not shown in this diagram for simplicity purposes.

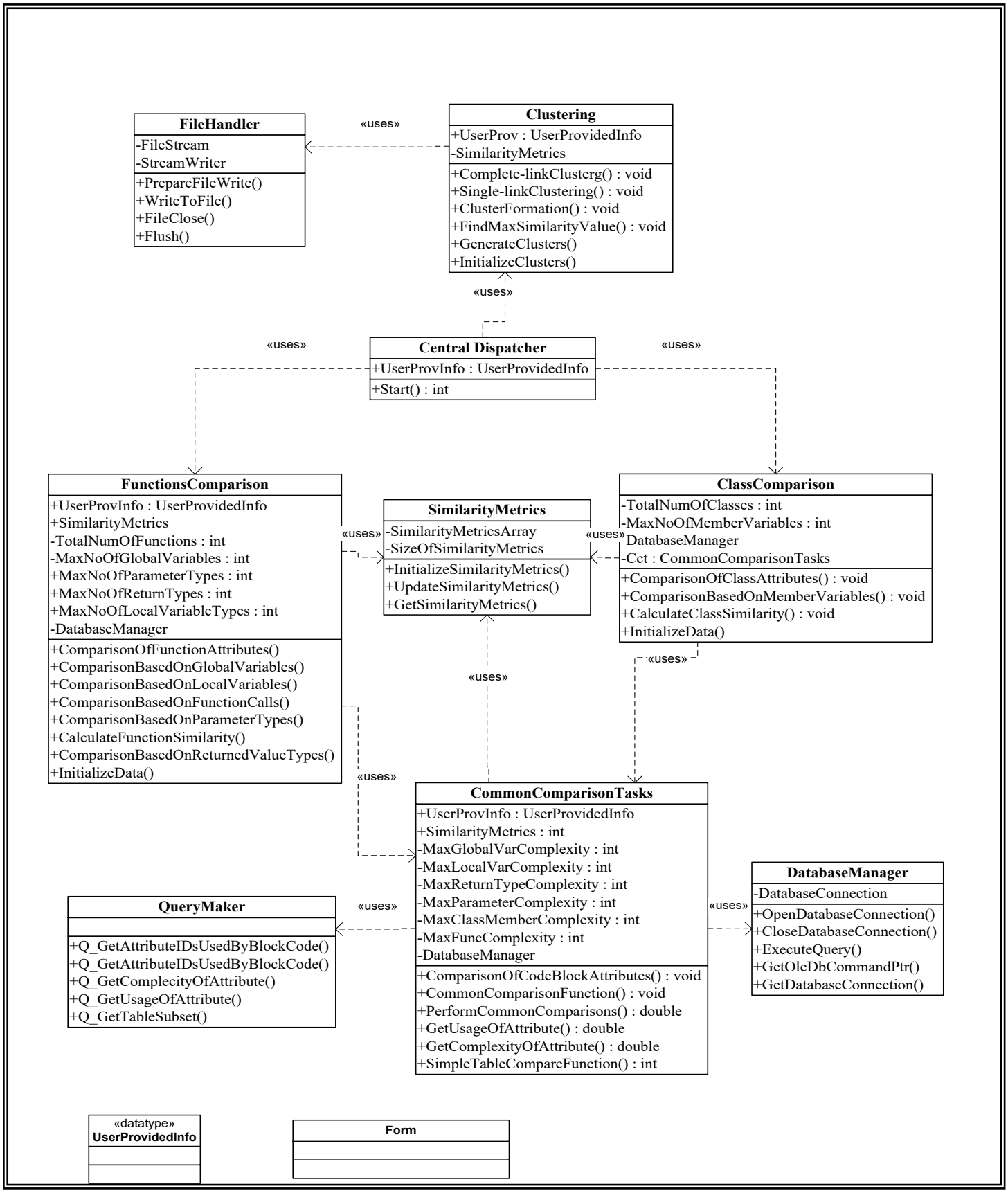


Figure 4.5 Class Diagram

The details of all the classes, their relationships and their functionalities are mentioned in the implementation section.

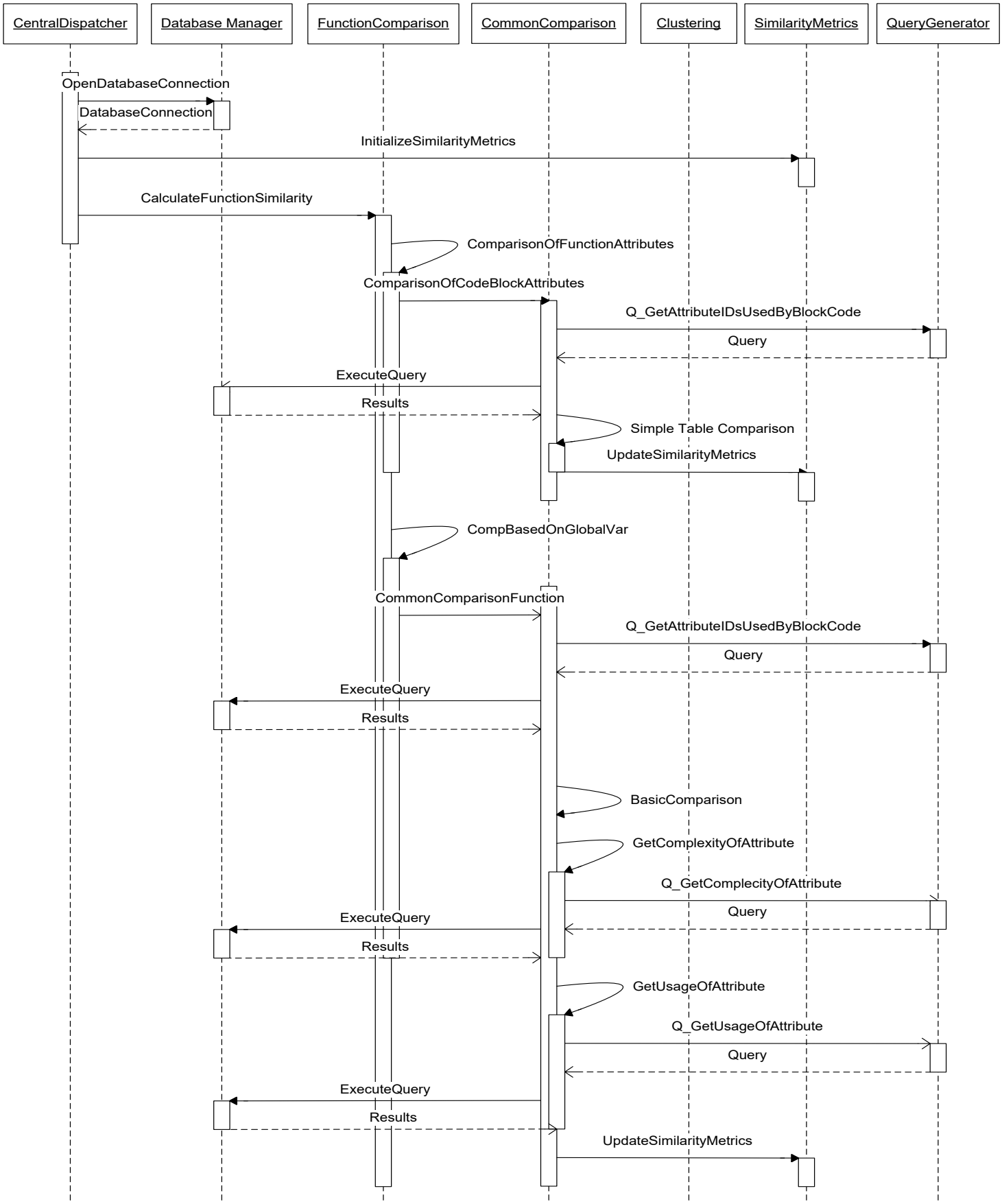
4.7 Sequence Diagram

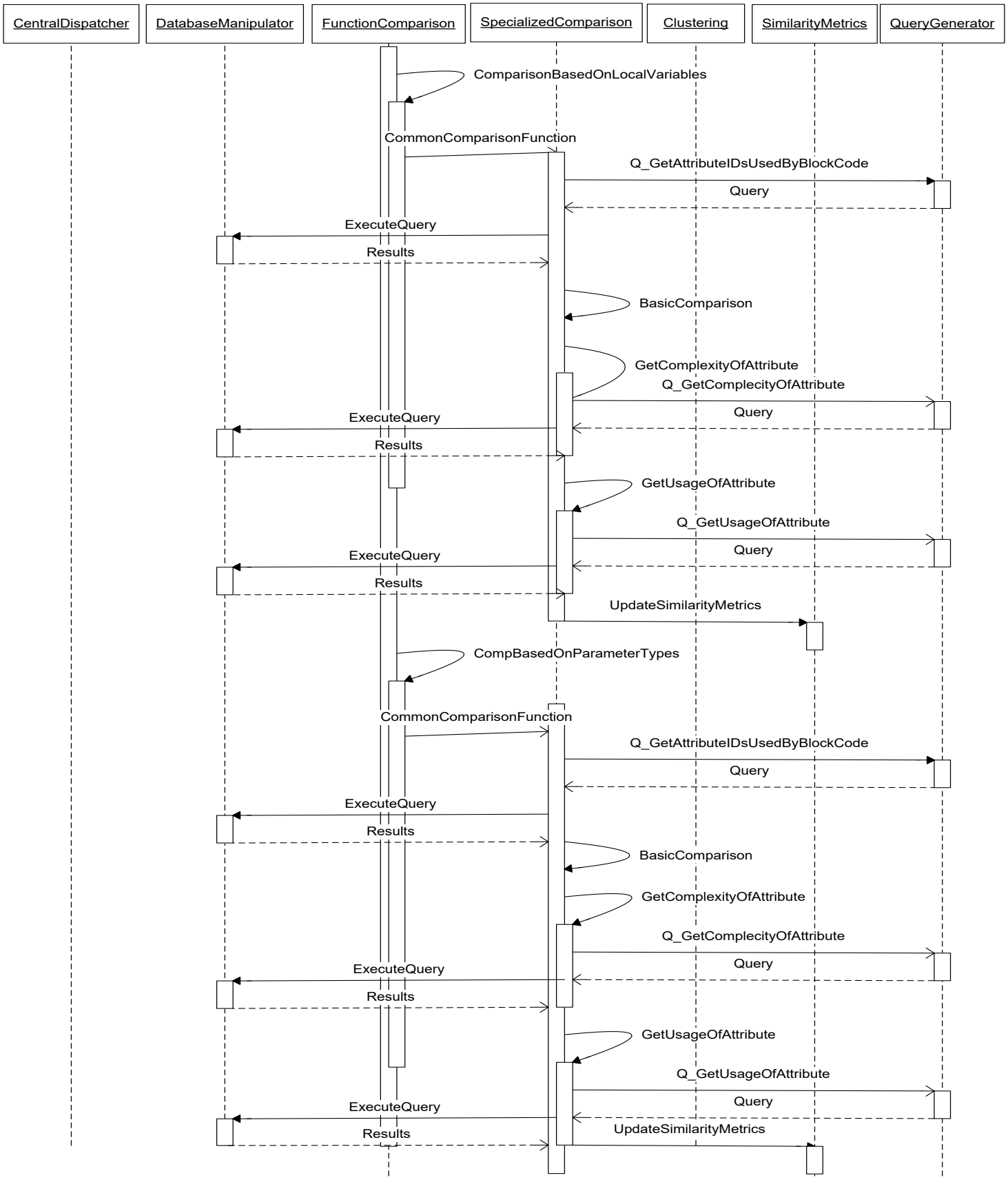
The figure 4.6 shows the sequence diagram that show the flow of the application, if the user chooses to perform function clustering on the basis of functions attributes, their use of global variables, local variables, parameter types, return types and the functions they call. The classes and the functions responsible for user interface are not shown in order to make the diagram much simpler, and easy to understand. By looking at the diagram it becomes obvious that many of the steps are repeated for every attribute, therefore an overview of all the steps is given in order to make the sequence diagram easier to understand. (All the names mentioned in inverted commas in the paragraph below are names of the classes taking part in the sequence diagram).

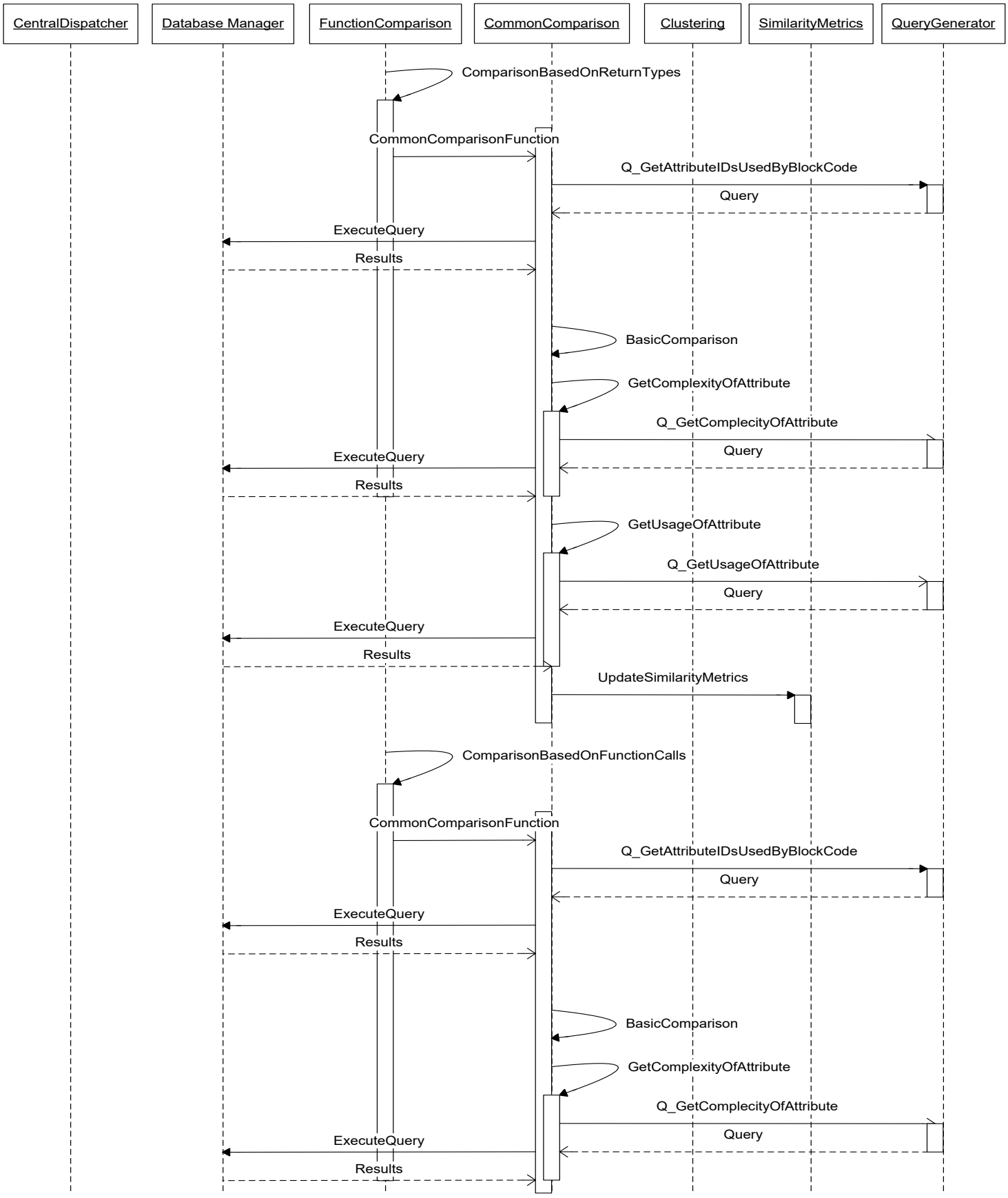
At first the “DatabaseManager” class opens the database connection (on the request of “CentralDispatcher”), in order to get the function information from the database. The “SimilarityMetrics” then initialise the array that will contain the similarities between all the function pairs.

The “CentralDispatcher”, now initiates the “FunctionComparison” class to perform the similarity comparisons and clustering tasks.

The “FunctionComparison” class dictates the “CommonComparison” class to calculate similarities between functions, by using the similarity principles and equations mentioned in the previous sections. After calculating the similarities, the information is passed on to the “Cluster” class to perform the clustering operation.
All the queries made to the database are through the “QueryGeneratorClass”, and the Queries are executed by the “DatabaseManager”.


CentralDispatcher
Database Manager
FunctionComparison
CommonComparison
Clustering
SimilarityMetrics
QueryGenerator
OpenDatabaseConnection
DatabaseConnection
InitializeSimilarityMetrics
CalculateFunctionSimilarity
ComparisonOfFunctionAttributes
ComparisonOfCodeBlockAttributes
Q_GetAttributeIDsUsedByBlockCode
Query
ExecuteQuery
Results
Simple Table Comparison
UpdateSimilarityMetrics
CompBasedOnGlobalVar
CommonComparisonFunction
Q_GetAttributeIDsUsedByBlockCode
Query
ExecuteQuery
Results
BasicComparison
GetComplexityOfAttribute
Q_GetComplecityOfAttribute
Query
ExecuteQuery
Results
GetUsageOfAttribute
Q_GetUsageOfAttribute
Query
ExecuteQuery
Results
UpdateSimilarityMetrics


CentralDispatcher
DatabaseManipulator
FunctionComparison
SpecializedComparison
Clustering
SimilarityMetrics
QueryGenerator
ComparisonBasedOnLocalVariables
CommonComparisonFunction
Q_GetAttributeIDsUsedByBlockCode
Query
ExecuteQuery
Results
BasicComparison
GetComplexityOfAttribute
Q_GetComplecityOfAttribute
Query
ExecuteQuery
Results
GetUsageOfAttribute
Q_GetUsageOfAttribute
Query
ExecuteQuery
Results
UpdateSimilarityMetrics
CompBasedOnParameterTypes
CommonComparisonFunction
Q_GetAttributeIDsUsedByBlockCode
Query
ExecuteQuery
Results
BasicComparison
GetComplexityOfAttribute
Q_GetComplecityOfAttribute
Query
ExecuteQuery
Results
GetUsageOfAttribute
Q_GetUsageOfAttribute
Query
ExecuteQuery
Results
UpdateSimilarityMetrics


CentralDispatcher
Database Manager
FunctionComparison
CommonComparison
Clustering
SimilarityMetrics
QueryGenerator
ComparisonBasedOnReturnTypes
CommonComparisonFunction
Q_GetAttributeIDsUsedByBlockCode
Query
ExecuteQuery
Results
BasicComparison
GetComplexityOfAttribute
Q_GetComplecityOfAttribute
Query
ExecuteQuery
Results
GetUsageOfAttribute
Q_GetUsageOfAttribute
Query
ExecuteQuery
Results
UpdateSimilarityMetrics
ComparisonBasedOnFunctionCalls
CommonComparisonFunction
Q_GetAttributeIDsUsedByBlockCode
Query
ExecuteQuery
Results
BasicComparison
GetComplexityOfAttribute
Q_GetComplecityOfAttribute
Query
ExecuteQuery
Results

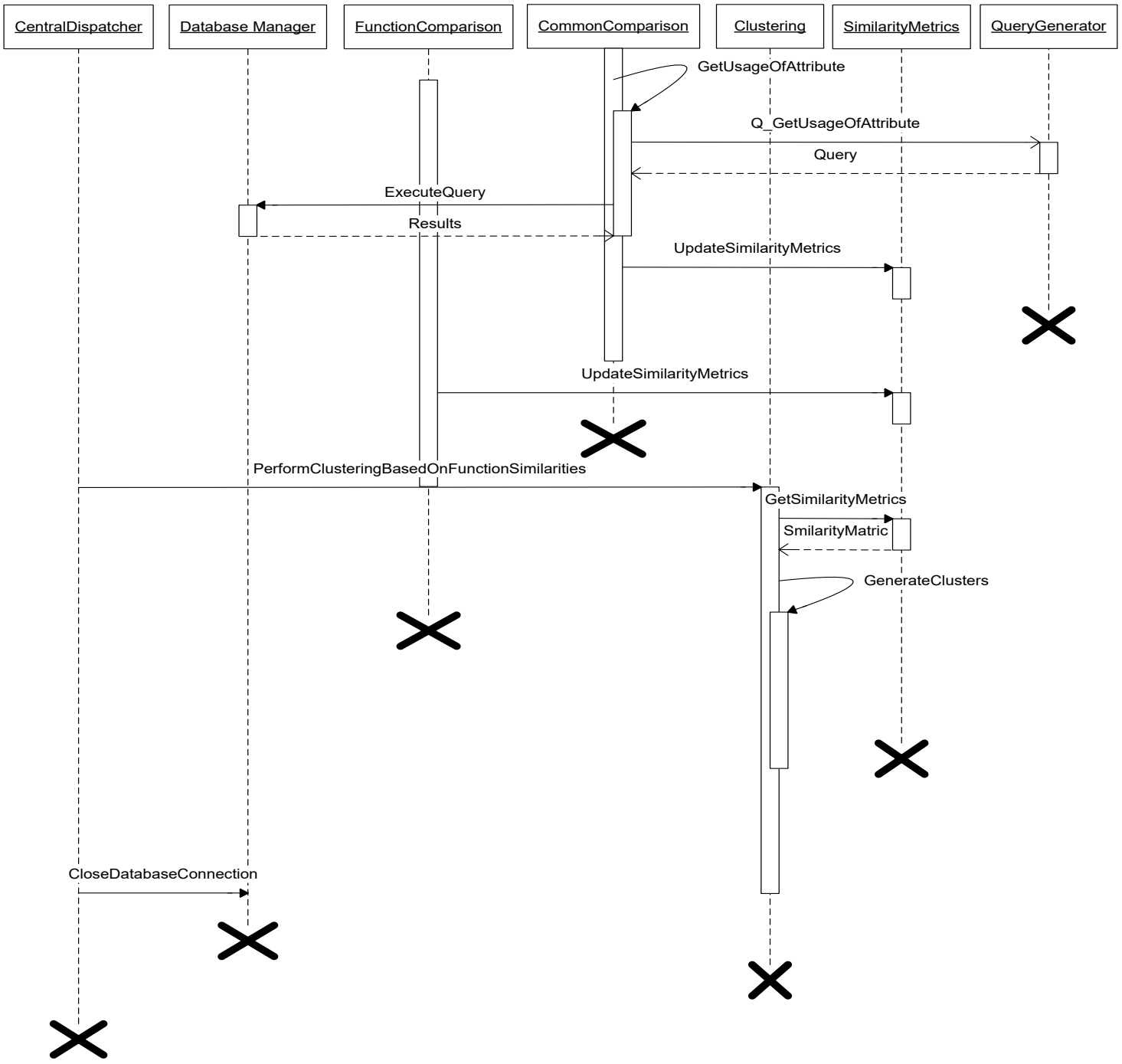


Figure 4.6 Sequence Diagram

4.8 Database Design

Access database is used to implement the function and class input models explained in section 4.1. This section explains the design and relationships of the database tables that are used as a data source by the application to perform code clustering.

4.8.1 Function Input Model

Figure 4.7 shows the relationship diagram of the function input model implemented in the access database.

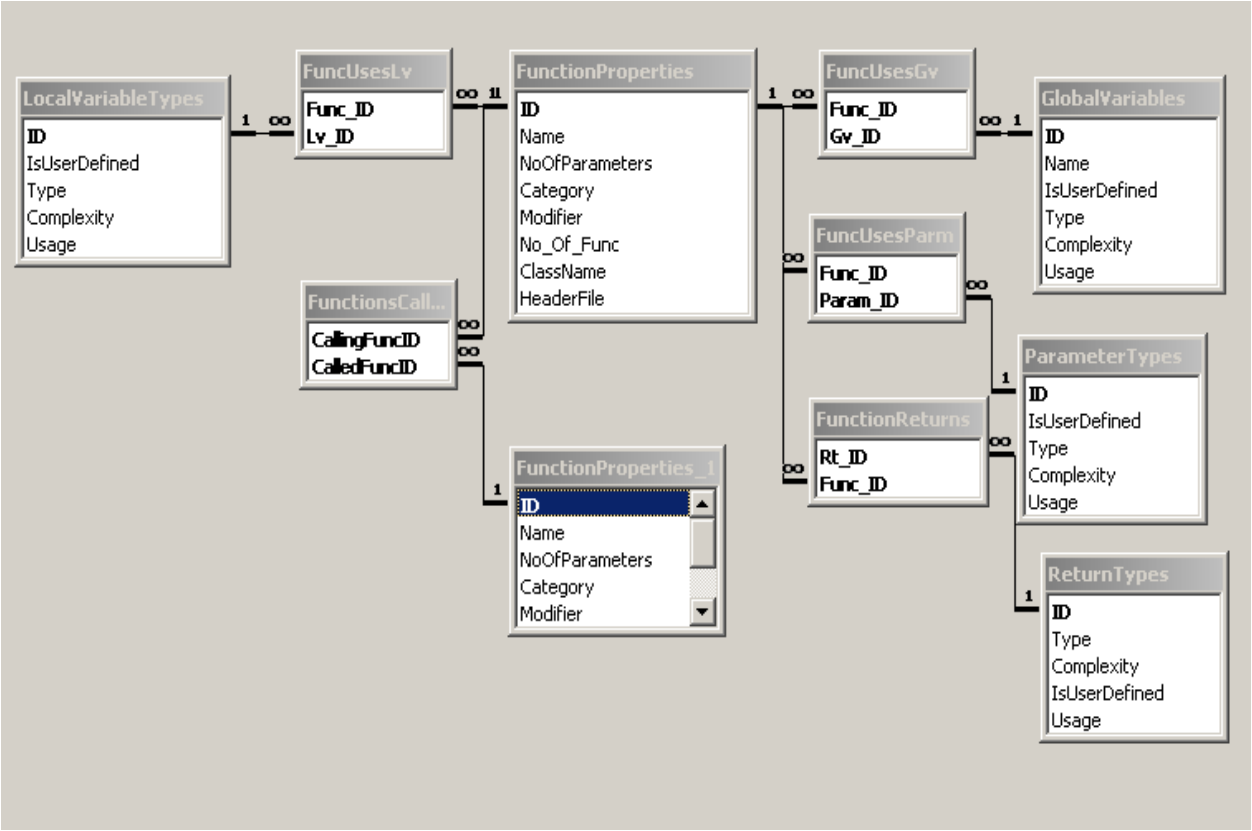
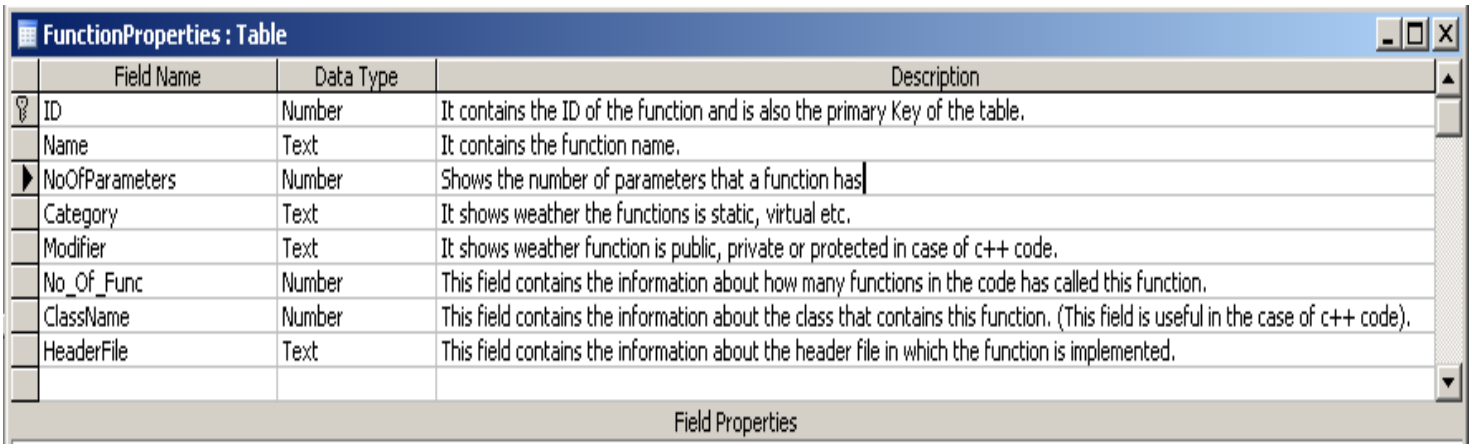


Figure 4.7 Function Input Model (Access Database)

The tables "FuncUsesLv", "FunctionsCalled", "FuncUsesGV", "FunctionUsesParm" and "FunctionReturns" are used as bridge entities that implement the "many-to-many" relationship between two tables.

Most of the tables contain similar fields, therefore the design of only a couple of tables is shown in order to explain the fields that were used to represent each of the attribute.

Figure 4.8 and 4.9 show the design-view of the FunctionProperties and GlobalVariables tables respectively. The description of all the fields is given in the description portion of the design.

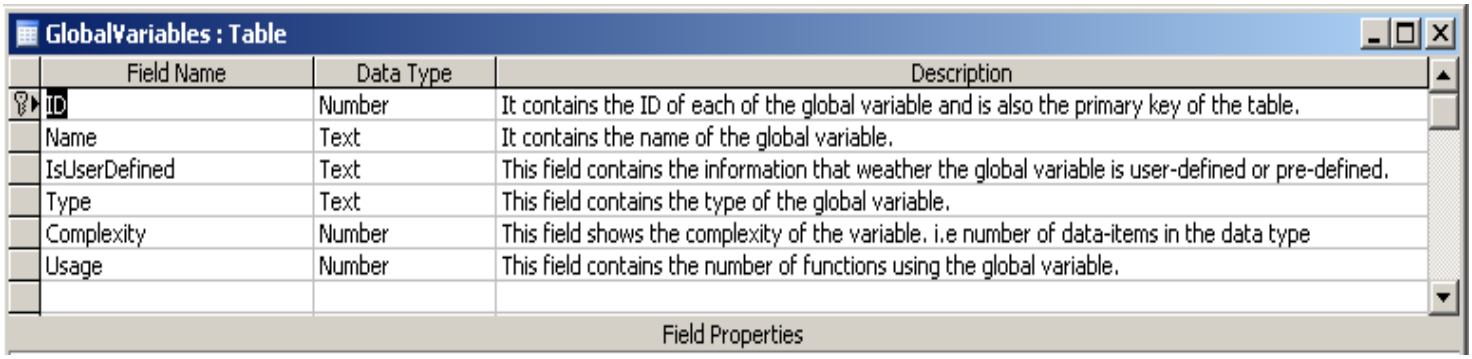


FunctionProperties : Table

Field Name	Data Type	Description
ID	Number	It contains the ID of the function and is also the primary Key of the table.
Name	Text	It contains the function name.
NoOfParameters	Number	Shows the number of parameters that a function has
Category	Text	It shows weather the functions is static, virtual etc.
Modifier	Text	It shows weather function is public, private or protected in case of c++ code.
No_Of_Func	Number	This field contains the information about how many functions in the code has called this function.
ClassName	Number	This field contains the information about the class that contains this function. (This field is useful in the case of c++ code).
HeaderFile	Text	This field contains the information about the header file in which the function is implemented.

Field Properties

Figure 4.8 Design of the FunctionProperties Table.



GlobalVariables : Table

Field Name	Data Type	Description
ID	Number	It contains the ID of each of the global variable and is also the primary key of the table.
Name	Text	It contains the name of the global variable.
IsUserDefined	Text	This field contains the information that weather the global variable is user-defined or pre-defined.
Type	Text	This field contains the type of the global variable.
Complexity	Number	This field shows the complexity of the variable. i.e number of data-items in the data type
Usage	Number	This field contains the number of functions using the global variable.

Field Properties

Figure 4.9 Design of the GlobalVariables Table.

4.8.2 Class Input Model

Figure 4.10 shows the relationship diagram of the class input model implemented in the access database.

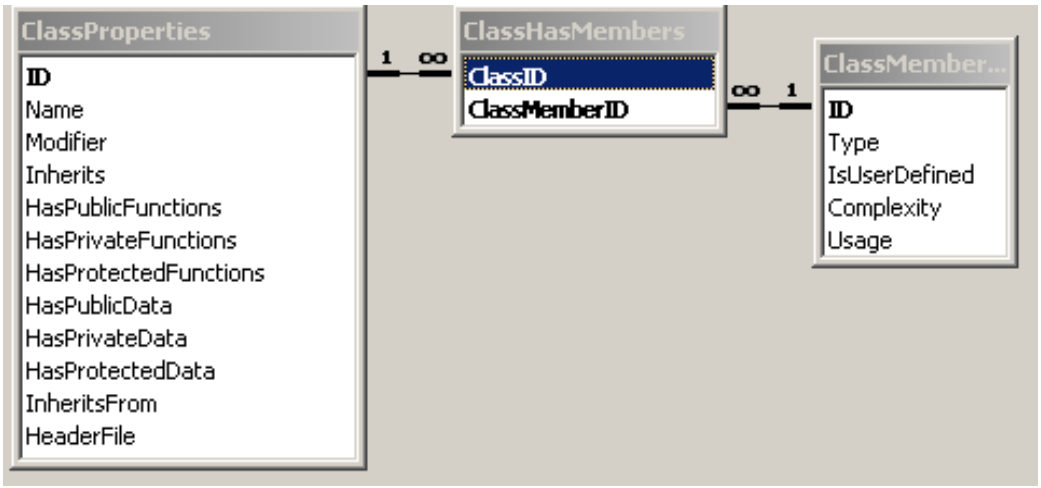


Figure 4.10 Class Input Model (Access Database)

The table "ClassHasMembers" acts as a bridge entity that implements the "many-to-many" relationship between the other two tables.

Figure 4.11 shows the design-view of the "ClassProperties" table. The description of all the fields is given in the description portion of the design.

ClassProperties : Table

Field Name	Data Type	Description
ID	Number	It contains the ID of the Classes and is also the primary Key of the table.
Name	Text	It contains the Class name.
Modifier	Text	It shows some other propertis of the class, like whether it is static, abstract etc
Inherits	Text	It shows weather the class inherits from any other class or not.
HasPublicFunctions	Text	It Contains the Boolean information about the class that weather the class has public functions or not.
HasPrivateFunctions	Text	It Contains the Boolean information about the class that weather the class has private functions or not.
HasProtectedFunctions	Text	It Contains the Boolean information about the class that weather the class has protected functions or not.
HasPublicData	Text	It Contains the Boolean information about the class that weather the class has public data or not.
HasPrivateData	Text	It Contains the Boolean information about the class that weather the class has private data or not.
HasProtectedData	Text	It Contains the Boolean information about the class that weather the class has protected data or not.
InheritsFrom	Text	It Contains the name of the class from which the class inherets.
HeaderFile	Text	It contains the header file in which the class is defined.

Field Properties

Figure 4.11 Design of the ClassProperties Table.

5

Implementation

5.1 Source Code Overview

A description of some of the important classes and their functions is given below. The class diagram shows all the functions that the classes have but in this section the author explains only the effects of some major functions. In the explanation the classes and the functions responsible for the user interface are not discussed in great details, as they are of little importance. Although screen shorts are shown in order to show the wide range of options that the user has while using the software.

1. **File Handler**

   The main purpose of this class is to write the results of the clustering algorithms to a text file. The eventual output of the software is a step by step representation of clusters formation in a text file. The text file is then used by another program, to produce its graphical representation.

   The File Handler class consists of three main functions

```
void PrepareFileWrite(String* FileName)
void WriteToFile(String *Str)
void FileClose()
```

   These three functions are responsible for opening the text file, writing the clustering results and then eventually closing it.

2. **Clustering**

   This class actually performs the clustering operations. Only single-link clustering and complete-link clustering are implemented due to lack of time. Major functions of this class are:

```
void ClusterFormation(DatabaseManager *DbManager)
void FindMaxSimValue(int* row, int* column)
void SingleLinkClustering(int column1, int column2)
void CompleteLinkClustering(int column1, int column2)
void GenerateClusters(Int32 Cluster1, Int32 Cluster2, Single max)
```

The ClusterFormation is basically a central dispatcher for this class, it gets the information about the choices made by the user and then calls the functions according to these choices.

The FindMaxSimValue function detects the most similar clusters at any particular instance. It detects the two most similar clusters at any particular stage and pass on their IDs and their similarities to the clustering algorithms.

The main purpose of the GenerateClusters function is to setup the format in which the results are written to the text file. The format is specifically chosen, so that the output file generated is in the format which can be used by another program to generate the graphical representation of the clustering process.

The SingleLinkClustering function actually performs the concept of single-link-clustering. After getting the most similar clusters and their similarities from the FindMaxSimValue function. This function combines the two clusters and updates the similarities of all the clusters by using the concept of single-link-clustering which is explained in the previous chapters.

The functionality of CompleteLinkClustering is the similar to that of SingleLinkclustering but it updates the similarities of all the clusters by using the concept of complete-link-clustering.

3. Central Dispatcher

It has only one function.

```
int Start(String* CodeType)
```

As the name of the class suggests, it is the central dispatcher for the application, it is this function that actually manages flow of the application. It initiates different operations depending on the input provided by the user.

4. **Function Comparison**

This class contains all the functions that setup the data and operations needed to perform function clustering. It contains following major functions.

```
void ComparisonBasedOnFunctionProperties();
void ComparisonBasedOnLocalVariableTypes();
void ComparisonBasedOnParameterTypes();
void ComparisonBasedOnGlobalVariables();
void ComparisonBasedOnReturnedValueTypes();
void ComparisonBasedOnFunctionCalls();
```

Names of all the functions clearly indicate the operations they perform. They set all the values to perform comparison between functions on the basis of function properties, local variables types, parameter types, global variables, return types and function calls respectively.

5. **Class Comparison**

This class contains all the functions that setup the data and operations needed to perform class clustering. It contains following major functions.

```
void ComparisonBasedOnClassProperties();
void ComparisonBasedOnMemberVariableTypes();
```

These functions set all the values to perform class comparisons on the basis of class properties and class member types respectively.

6. **CommonComparisonTasks**

This is the class that actually performs similarity measures. All the similarity matrices and equations discussed in the previous section are implemented by this class.

It has many functions but only few significant are discussed here, which are

```
void ComparisonOfCodeBlockAttributes(String *, String *, Single
Weight);
void CommonComparisonFunction(GeneralPerposeStruct *, Single
Weight);
Single PerformCommonComparisons(GeneralPerposeStruct *);
```

The ComparisonOfCodeBlockAttributes function calculates the similarities of functions and classes based on their properties and writes the data to the similarity metrics.

The CommonComparisonFunction calculates similarity between classes and functions based on the rest of the attributes, that is, local variables, global variables, function calls, return types and parameter types for the functions entities, and member data types for the class entities. This functions calls PerformCommonComparisons that calculate the similarities based on the three principles, that is, Basic principle, Usage principle and complexity principle. Code fragment for the PerformCommonComparisons is shown below.

```
Single
CommonComparisonTasks::PerformCommonComparisons(GeneralPerposeStru
ct * Gpc)
{
  ..................

   if(CommonV != 0)
   {
     UnCommon    = ((Gpc->NoOfVarUsedByCodeBlockA+Gpc-
     >NoOfVarUsedByCodeBlockB)-(2*CommonV));
     TotalV      = CommonV + UnCommon;
     if(UserProvInfo->Complexity == true)
     Complexity = Complexity/CommonV; //Implementing Complexity
     Principle

     if(UserProvInfo->Usage == true)
     Usage       = Usage/CommonV; //Implementing Usage Principle


     if(UserProvInfo->Basic == true)
     Basic      = CommonV/TotalV; //Implementing Basic Principle


     if(UserProvInfo->Basic == true){Devisor++;}
     if(UserProvInfo->Complexity == true) {Devisor++;}
     if(UserProvInfo->Usage == true){Devisor++;}

     return ((Complexity+Usage+Basic)/Devisor);}
     else return 0;}
```

7. Similarity Metrics

This class manages the Similarity metrics array used by the application to store similarities between all the entities. All the results after performing the similarity measures are stored by this class in an array and it constantly maintains all the updates in any similarity between entities.

Its major function is

```
void UpdateSimilarityMetrics(Single Weight, Single Size)
```

This function updates any changes in the similarity between functions or classes after calculating new similarities.

The similarity metrics array is defined in this class as

```
Single SimilarityMetricsArray [,];
```

It is a two dimensional array that contains the similarity between functions or classes.

8. Database Manager

As is clear from the name, this class maintains the opening, closing and access to the database. All the queries are executed and managed by this class.

Its functions are

```
OleDbConnection* OpenDatabaseConnection();
void CloseDatabaseConnection();
OleDbDataReader* ExecuteQuery(String* Query, OleDbConnection*
ProgramDB);
OleDbConnection * GetDatabaseConnection();
```

There functionality of the functions is clear from their names.

9. Query Maker

This class actually generates the queries to the database. It gets the requirement (the data needed for performing the similarity measures) and dynamically generates the query that will help in getting the exact information from the database that is required.

Its functions are

```
String* Q_GetAttributeIDsUsedByBlockCode(String *Attribute, Int32
ID)
String* Q_GetComplecityOfAttribute(String* Attribute, Int32 ID)
String* Q_GetUsageOfAttribute(String* Attribute, Int32 ID)
String* Q_GetTableSubset(String* TableName, Int32 ID, String*
Attributes)
```

These functions dynamically generate queries. The code of Q_GetUsageOfAttribute is shown to indicate how the queries are generated dynamically. This is one of the simplest query generator functions. Its code is shown in order to help in better understanding of dynamic query generation process.

```
String* Q_GetUsageOfAttribute(String* Attribute, Int32 ID)
{
  if(String::Compare(Attribute,"FunctionCalls",true) == 0)
  {
    return
    String::Concat("SELECT FunctionAttributes.No_Of_Func FROM
    FunctionAttributes WHERE (((FunctionAttributes.ID)=",
    ID.ToString(), "))" );
  }
......}
```

10. UserInterface (Namespace)

As the name suggests, this namespace contains all the classes and functions responsible for graphical user interface for the project. All the classes in this name space inherit from the Form class (Form is one of the classes provided by .NET framework for user interface purposes). These classes are discussed briefly in this sub-section.

- Form1

  This is the main class, for providing the core user interface facilities to the user. The application starts by displaying a form (using this class) containing majority of the options that the user can select in order to perform clustering. The form that is initially displayed is explained the user interface section.

- AdvancedOptionChecks and AdvancedWeightOptions

  These two classes provide specialized options for more thorough results, and for better evaluation of the software being examined.

- AboutBox

  This class displays a dialog box that shows some general information about the software.

- ClusteringProgressBar

  This class indicates the progress made by the software (during the clustering process) to the user.

11. UserProvidedInfo (DataStructure)

This is the data structure that contains all the information that the user can give in order to calculate the similarity between functions or classes and the type of clustering that is to be performed. The data structure along with the comments is given below.

```
__gc struct UserProvidedInfo
{
      bool ParameterTypes; //Whether parameter types should be
      used //for calculating similarities
   bool ReturnTypes; //Whether Return types should be used for
      //calculating similarities
   bool UseOfLocalVariableTypes; //Whether local variable types
      //should be used for calculating similarities
   bool UseOfGlobalVariables; //Whether global variables should be
      //used for calculating similarities
   bool FunctionAttributes; //Whether function Attributes should
      //be used for calculating similarities
   bool FunctionCalls; //Whether function calls should be used
      //for calculating similarities
   bool Basic,Usage,Complexity; //Types of principles used
   String *FuncAttr[]; //contains the attributes selected for
      //clustering
   Single FaWeight; //weight given to function attributes
   Single FcWeight; //weight given to function calls
   Single RtWeight;//weight given to return types
   Single UlvWeight, UulvWeight, UplvWeight;//weights given to
      //User-defined and pre-defined local variables
   Single UgvWeight, UugvWeight, UpgvWeight;//weights given to
      //user-defined and pre-defined global variables
   Single PtWeight, PtuWeight, PtpWeight; //weights given to user-
      //Defined and pre-defined parameter types
   bool GvUPDiff, LvUPDiff, PtUPDiff; // whether to differentiate
      //between user-defined and pre-defined variables or not
      ........................
    };
```

5.2 User interface and options provided to the user

The software in itself is very dynamic. There are many options provided to the user so that the user can have as many opportunities to understand the code being examined. In the sections below the GUI and the options provided to the users are explained. Only the options provided for function clustering are explained as for class clustering the options are more or less similar (the GUI
for class clustering is shown in the Appendix III). Here one thing must be clarified that all the figures shown are to indicate all the functionalities of the software that a user can avail. If the user does not select many of these options, the software automatically fills the default values for the clustering process. Also there is error checking facility available to check whether the user has selected the correct options, and has entered correct values in the text boxes.

Figure 5.1 shows the initial screen that is displayed when the software starts. It shows the general information about the project.

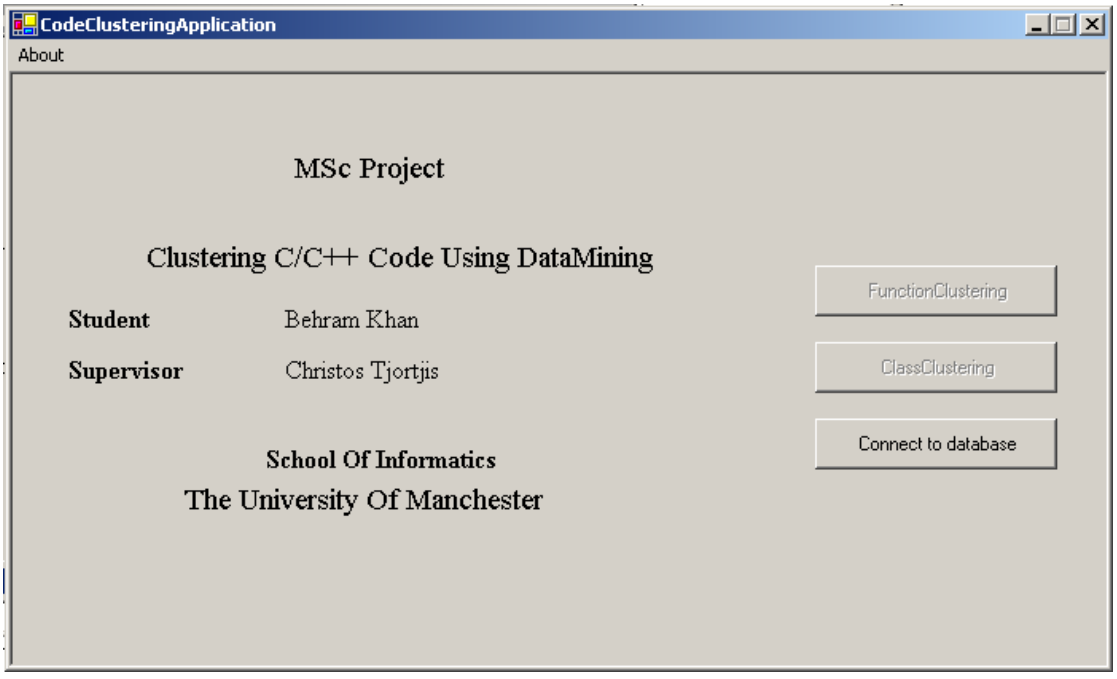


Figure 5.1 Initial Screen

The first thing the user has to do is to select the database which contains the parsed information about the code. For that the user selects the ***"Connect to database"*** button and then selects the database. A filter is applied in the software that only allows the

user to select the .mdb files only (as the software expects that the database containing the source code information is an MS Access data base).

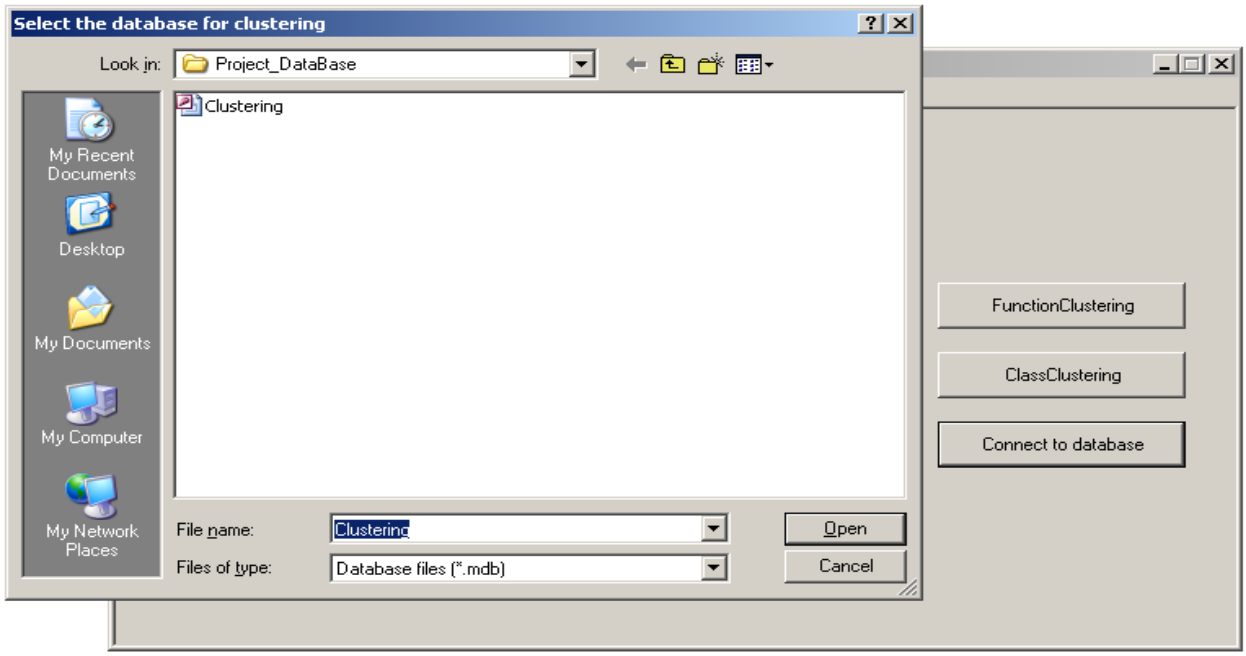


Figure 5.2 Select Database

All the queries to get the data are made to the database which the user selects from this dialog box.

Now the user has the option to select function or class clustering. The two buttons "***FunctionClustering"***, "***ClassClusteing"*** are for this purpose.

The discussion below explains the options given to the user in case of function clustering only (the options for class clustering are similar to that of function clustering and are shown in the Appendix III).

As explained earlier, the function clustering can be performed on the basis of function properties, used of global variables, local variables, return types, parameter types and function calls. The check boxes (in Figure 5.3) show exactly these options. The user can select any of the options to be used in the functions clustering process.

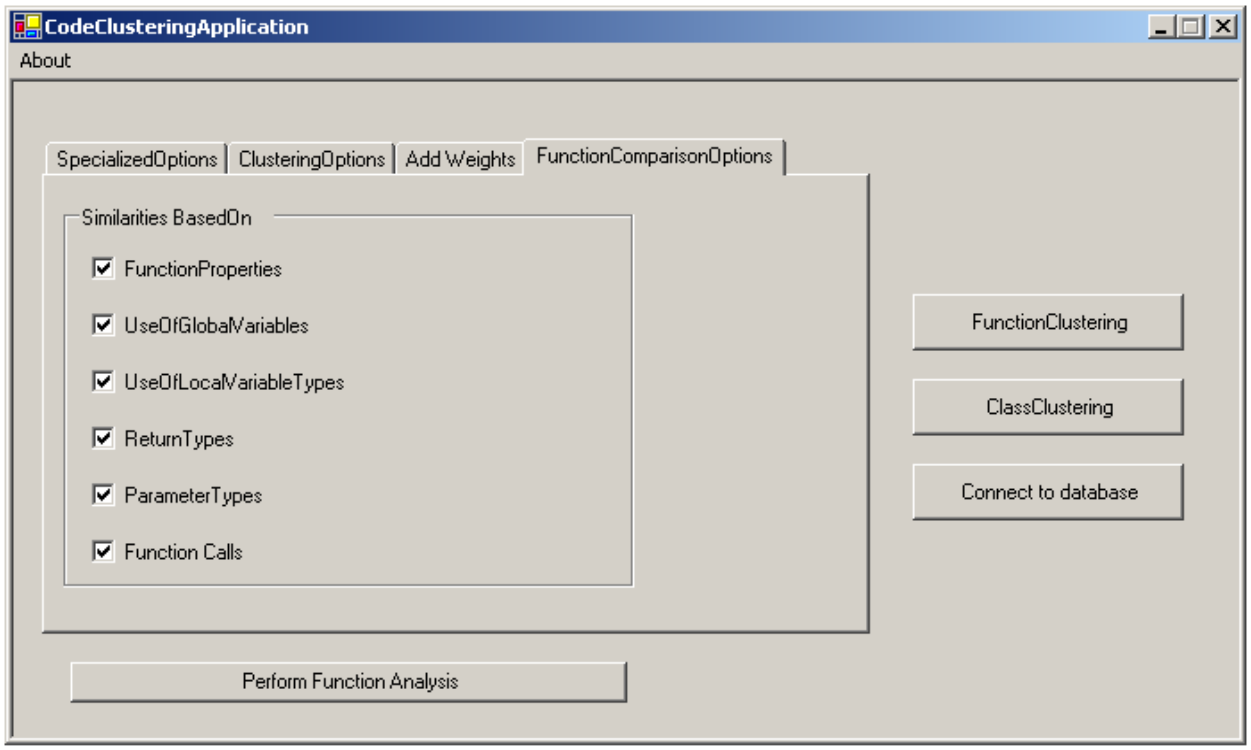


Figure 5.3 Select function attributes to perform clustering

Then is the ***"SpecialiedOptions"*** tab. Here as shown in the Figure (Figure 5.4) the user can select the function properties to be used in order to perform clustering. Function properties can only be selected if the user selects the ***"FunctionProperties"*** checkbox in the ***"FunctionComparisonOptions"*** tab. (shown in the Figure 5.3), otherwise this functionality is disabled.

As previously explained there are three principles (basic, usage and complexity) which are used to calculate similarity between functions (or classes). The checkboxes ***(Basic, Usage and Complexity)*** in Figure 5.4 give user the options to select any or all of the principles for clustering purposes.

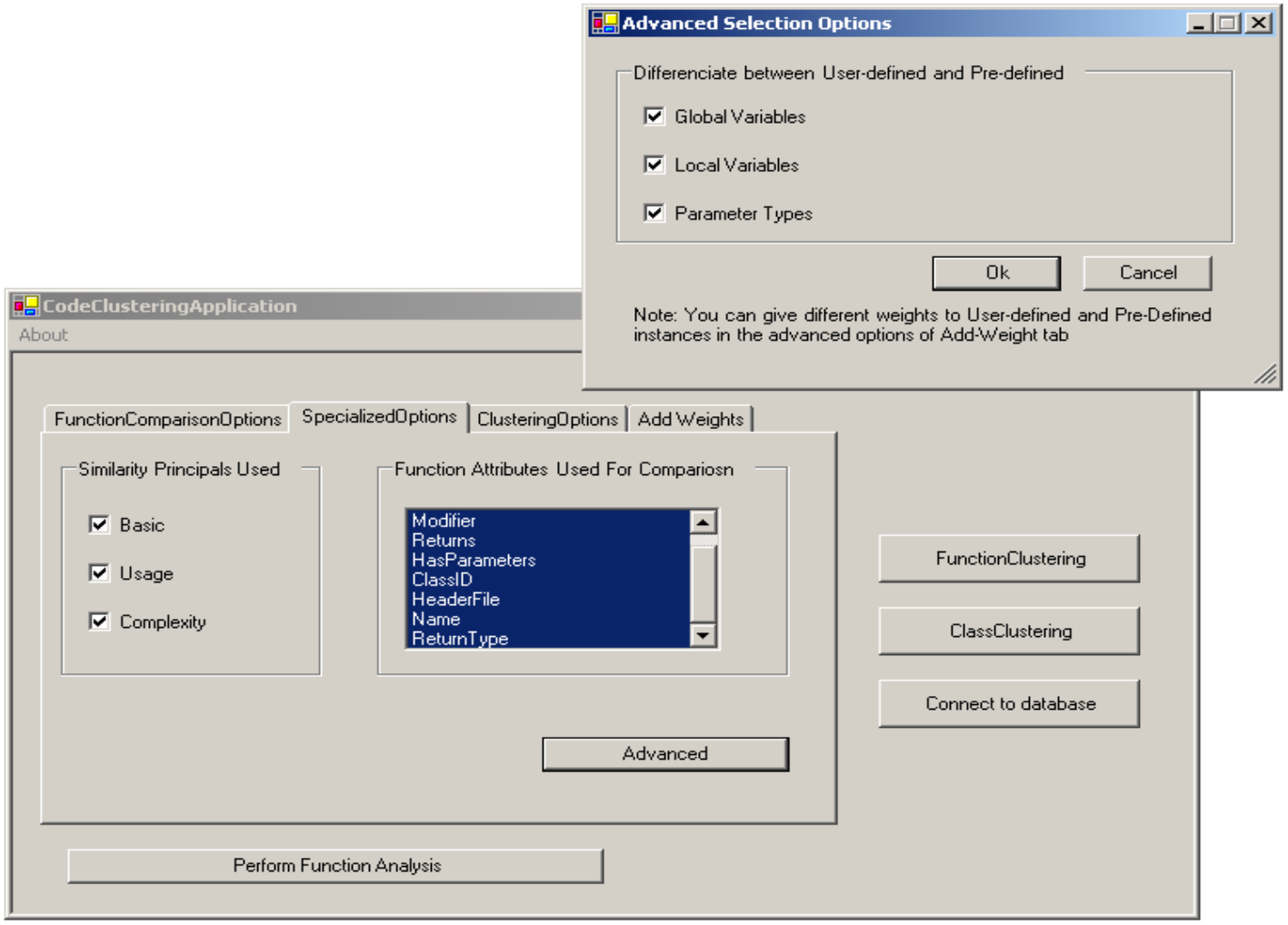


Figure 5.4 Specialized Options

There is an advanced button as well that helps the user to indicate whether he want to distinguish between user-defied and pre-defined types. The user can make this distinction in case of global variables, local variables, and parameter types only.
After selecting these options the user can give different weights to user-defined and pre-defined types as mentioned in the note of the ***"Advanced Selection Options"*** dialog box. The weights options are discussed in the latter screen shots.

Next is the Clustering Options (Figure 5.5). Here the user can select the type of clustering to be performed.
Due to shortage of time only single-link and complete-link clustering has been implemented.

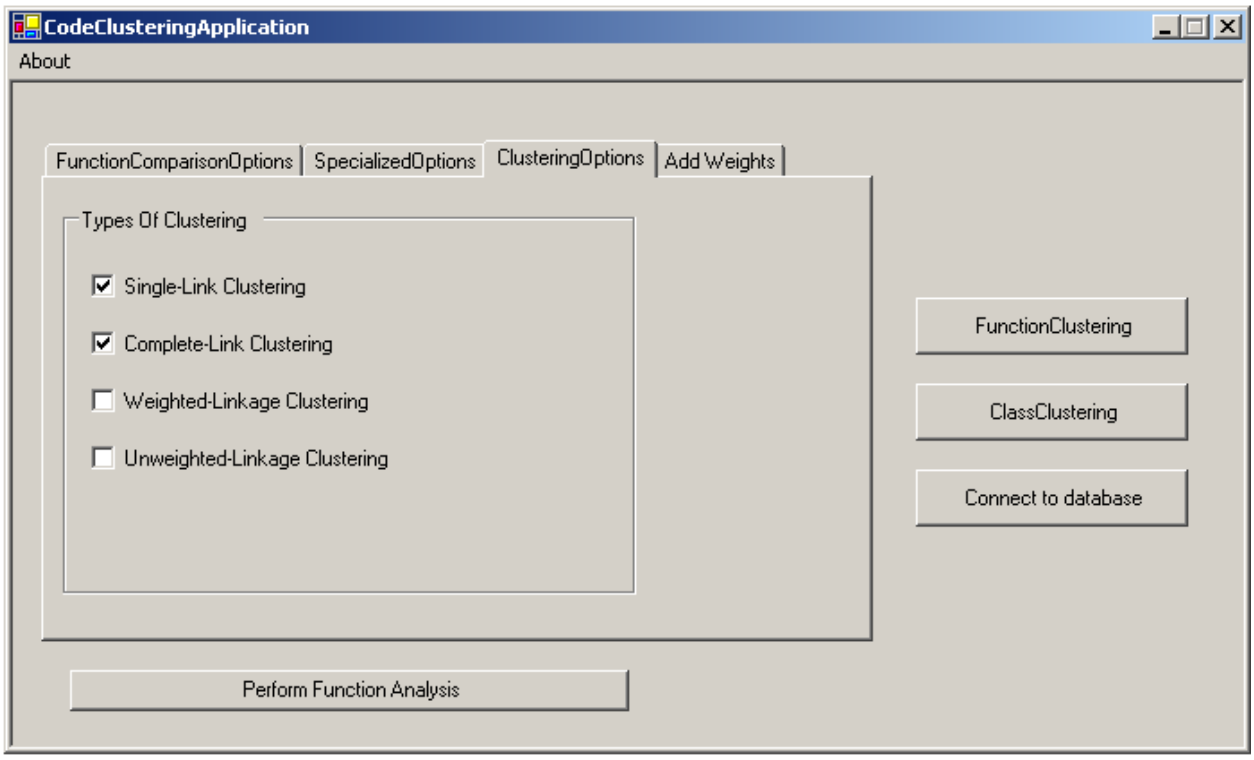


Figure 5.5 Clustering Options

In the end come the weight options (Figure 5.6). The user can select different weights for different attributes, in order to get valuable results and to understand the software being examined in a much better way. There is an advanced weight option available as well. As mentioned before, the advanced weight option is only available if the user chooses to distinguish between the user-defined and pre-defined types. In this dialog box the user can enter different weights for user-defined and pre-defined variables.

There is a complete error checking facility available at every stage in the software. The diagram below shows just one of error checking facility, That is, the error checked by software if the user enters an incorrect value for the weight for Pre-defined parameter type.

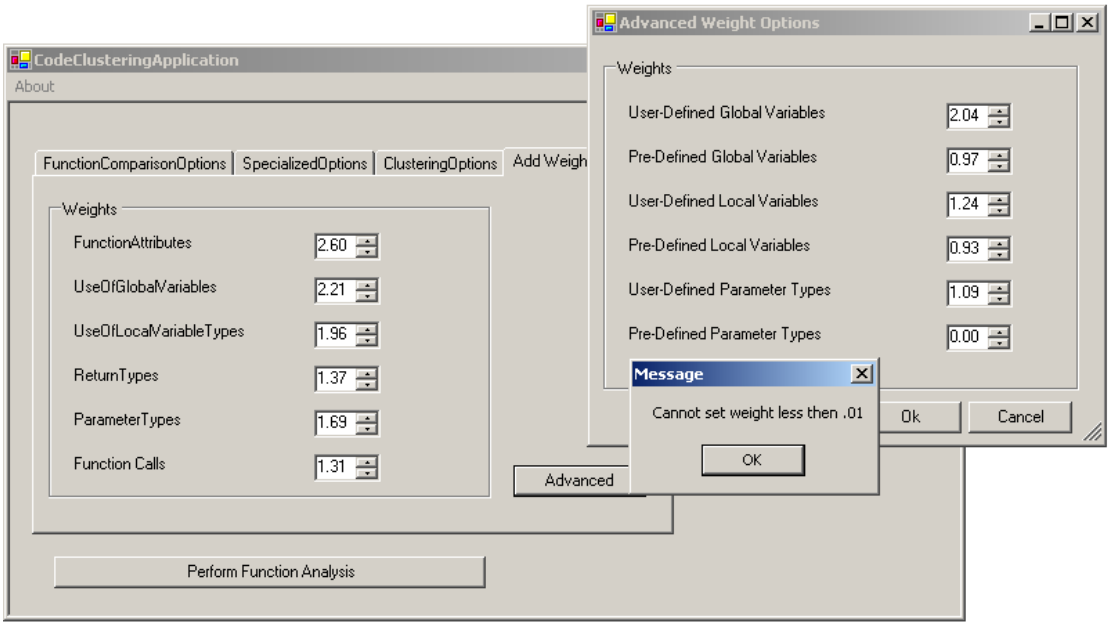


Figure 5.6 Weight Options

After selecting the options the user can now start the clustering procedure by pressing the Perform Function analysis button. After that, the processing starts and the clusters being generated are written to the text file.

As mentioned before, the user is not required to enter all the options provided by the software. For example, if the user does not enter the weights for any of the attributes, the software enters the default values for those weights and then starts performing the calculations.

Figure 5.7 indicates the progress made by the software while calculating the similarities between different attributes and notifies the user when performing a particular clustering process

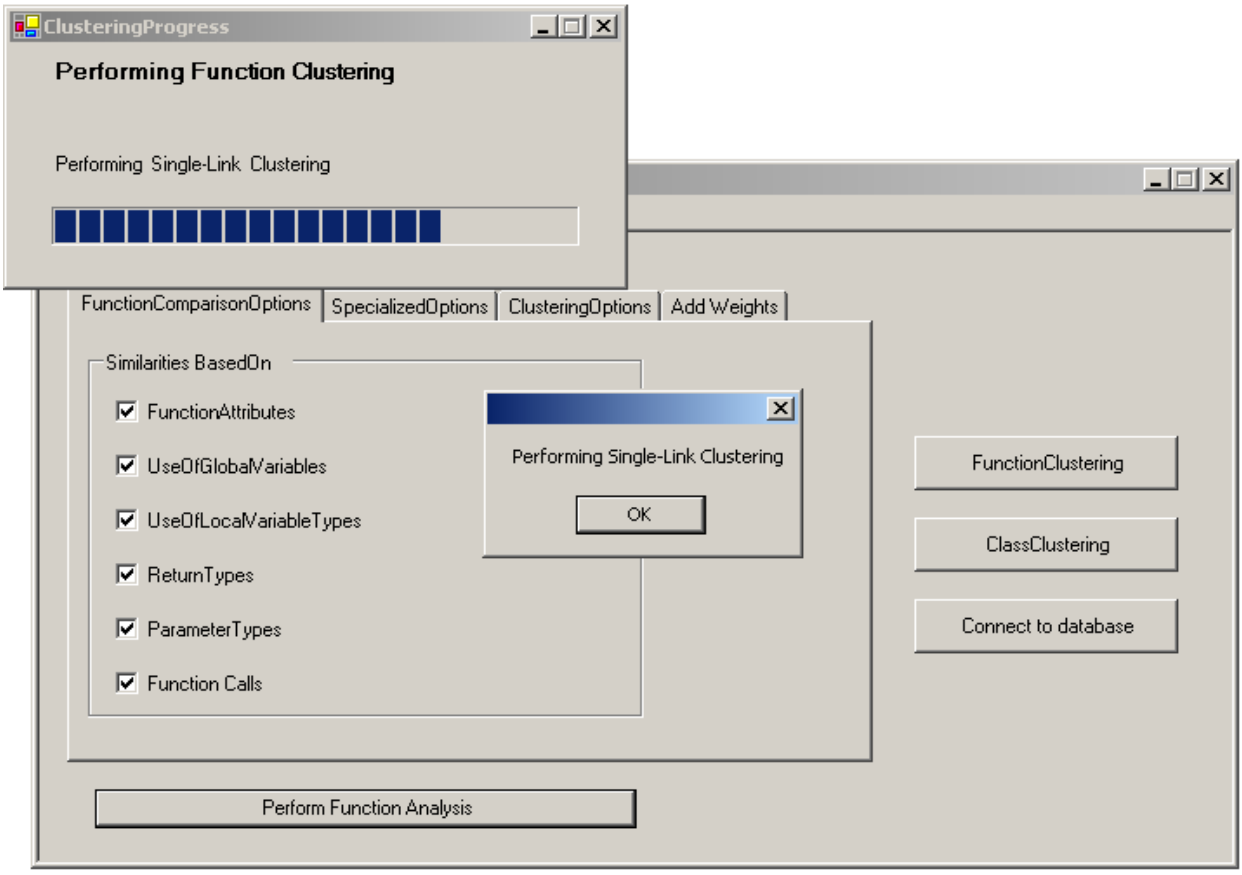


Figure 5.7 Progress made by the software tool.

5.4 Output

Figure 5.8 shows format of the text file that is generated by the software after performing function clustering. This text file contains the result of single-link clustering. The file shows step by step operations performed during the clustering operation and indicates which entities were clustered at which stage and what was the similarity between them. The numerical values within the brackets are the Ids of functions that are being clustered and the values outside the brackets represent the cluster number. The “sim” value shows the similarity between the two most similar clusters merged at any specific instance.

1{1} 2{2} 3{3} 4{4} 5{5} 6{6} 7{7} 8{8} 9{9} 10{10} 11{11} 12{12} 12 Clusters. Most similar clusters = 6 and 12, sim=0.658392

1{1} 2{2} 3{3} 4{4} 5{5} 6{6,12} 7{7} 8{8} 9{9} 10{10} 11{11} 11 Clusters. Most similar clusters = 3 and 6, sim=0.5997509

1{1} 2{2} 3{3,6,12} 4{4} 5{5} 6{7} 7{8} 8{9} 9{10} 10{11} 10 Clusters. Most similar clusters = 1 and 3, sim=0.5477273

1{1,3,6,12} 2{2} 3{4} 4{5} 5{7} 6{8} 7{9} 8{10} 9{11} 9 Clusters. Most similar clusters = 1 and 2, sim=0.543332

1{1,3,6,12,2} 2{4} 3{5} 4{7} 5{8} 6{9} 7{10} 8{11} 8 Clusters. Most similar clusters = 1 and 5, sim=0.4539995

1{1,3,6,12,2,8} 2{4} 3{5} 4{7} 5{9} 6{10} 7{11} 7 Clusters. Most similar clusters = 2 and 3, sim=0.3181818

1{1,3,6,12,2,8} 2{4,5} 3{7} 4{9} 5{10} 6{11} 6 Clusters. Most similar clusters = 4 and 6, sim=0.3181818

1{1,3,6,12,2,8} 2{4,5} 3{7} 4{9,11} 5{10} 5 Clusters. Most similar clusters = 4 and 5, sim=0.3181818

1{1,3,6,12,2,8} 2{4,5} 3{7} 4{9,11,10} 4 Clusters. Most similar clusters = 1 and 3, sim=0.2727273

1{1,3,6,12,2,8,7} 2{4,5} 3{9,11,10} 3 Clusters. Most similar clusters = 1 and 2, sim=0.2727273

1{1,3,6,12,2,8,7,4,5} 2{9,11,10} 2 Clusters. Most similar clusters = 1 and 2, sim=0.2272727

Figure 5.8 SingleLinkClustering.txt: an example output text file generated by the clustering tool.

This text file is then used as an input to another program, that converts this information into a visual form. (Note: The visual form shown in figure 5.9 is not the output for the text file shown in the previous diagram.).

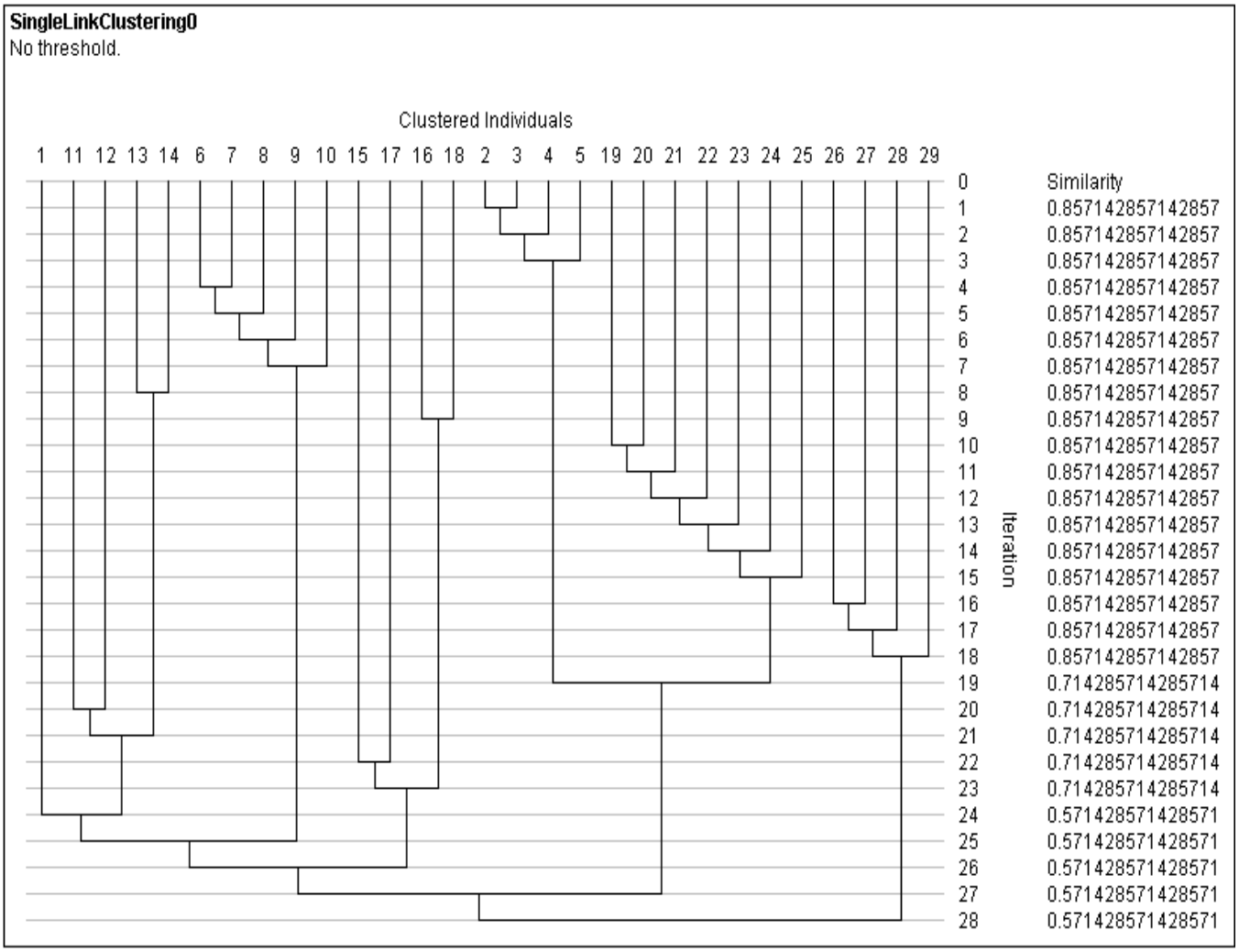


Figure 5.9 A Dendogram produced by the software, after taking as input, the text file produced by the CodeClusteringApplication.

This figure actually shows a Dendogram, to indicate the clustering that takes place at each step in the clustering process. (Dendogram was explained in section 2.5.3).

6

Evaluation

A methodology for deriving the input model from the source code by extracting entities and the relevant attributes is described in the previous chapters. The way that these entities could be clustered in order to retrieve a potentially meaningful modularization of a system is formulated. The approach needs to be evaluated in practice, assessing its effectiveness and possible improvements. A clustering tool is built for this reason, in order to experiment with real programs.

The source code of three programs has been used for experimentation. First the author used ***"CodeClusteringApplication"*** (CCAP), a small/medium C++ program with known mental model, which contains 13 classes and 35 functions, to test the applicability of the approach. Secondly the author used ***"Credit scoring e-service"***, a small/medium C# program with known mental model, which contains 25 classes and 76 functions, to evaluate the suitability of the approach for C# programs. A medium/large C# program ***"Administration of Books Publishing"*** without a known mental model, consisting of 5976 functions in 1242 classes was used to assess the scalability of the approach and its suitability with dealing with unfamiliar software.

The methodology comprises the following main steps.

1. Derive the input model in a semi automatic fashion, using a parser and the facility of a .NET class view and solution explorer.

2. Feed the input model to the tool.

3. Derive a subsystem abstraction of the program using single and complete linkage methods. It is shown that the choice of the method affects the nature of the clustering produced.

4. Compare the derived subsystem abstraction to an expert's mental model of the system if available.

Precision and Recall were used as quantitative elements in judging the correctness of the results of the tool (refer to section 2.6 for explanation about precision and recall). The term accuracy is used in some places during the explanation in order to refer to both precision and recall.

There was no similarity threshold imposed to stop the clustering process, so that all possible clustering could be derived and convergence or divergence towards the expert's mental model could be observed.

6.1 Case Study I (CodeClusteringApplication)

In this case study, the code of CodeClusteringApplication (CCAP) was used for the evaluation purposes. CCAP is a small application with 13 classes and 35 functions.

6.1.1 Function Clustering

This section discusses function clustering done by the software tool, with different options selected by the user. The best results in each case are shown and compared with the expert's mental model (the grouping of functions done by the expert).

Figure 6.1 shows the function clustering of the system done by the expert.
The numbering given to each of the function is equal to its Id in the actual database which was used as a source (of information) by the clustering tool. These numberings are used to represent the functions in the discussion below in order to make the clustering process easier to explain.

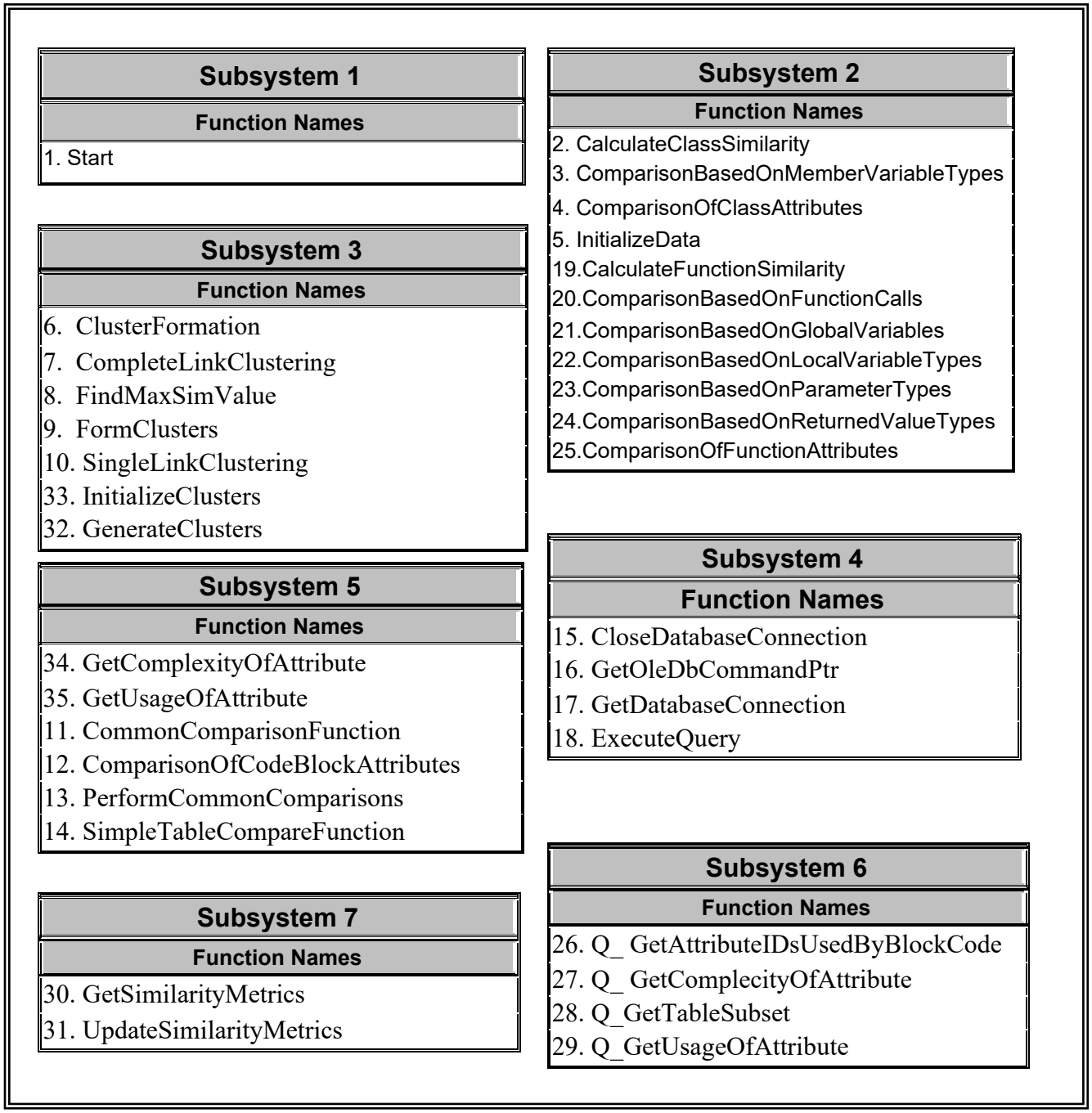


Figure 6.1 Case Study I: Expected results (Function Clustering)

[CodeClusteringApplication]

Subsystem 1: This subsystem contains only one function. This function is like a central dispatcher of the application. It determines the flow of the whole application depending on the selections done by the user.

Subsystem 2: This subsystem contains all the function that performs similarity measures on functions and classes based on the selections done by the user.

Subsystem 3: The functions in this subsystem perform the clustering operations that form clusters of functions and classes on the basis of similarities provided by the previous subsystem.

Subsystem 4: This subsystem contains all the functions that are responsible for performing operations on the data base of the system.

Subsystem 5: This subsystem carries out the similarity principles (that is basic, complexity, and usage principle).

Subsystem 6: This subsystem contains all the functions that generate queries dynamically in order to get the data needed from the database.

Subsystem 7: Functions in this subsystem manage the similarity matrix that holds all the similarities among all the entities in the data base.

Figure 6.1a Case Study I: Expected results (Function Clustering)
[CodeClusteringApplication]
Explanation of the Clusters suggested by the expert

Figure 6.1 can be represented in a simpler format by showing the functions in groups, only in terms of their IDs. The representation then becomes.

1{1}, **2**{2,3,4,5,19,20,21,22,23,24,25}, **3**{6,7,8,9,10,32,33}, **4**{15,16,17,18}, **5**{11,12,13,14,34,35}, **6**{26,27,28,29}, **7**{30,31}.

Case study I: Expected results (Function Clustering)
[CodeClusteringApplication]

Numerous results were produced by selecting different options and were compared with the expert's mental model. These results are as follows

Results I

The first results were produced by comparing functions only on the basis of function properties. Both, single-link and complete-link clustering algorithms were used to produce the results. The results generated by both these algorithms are explained below.

Single-Link-Clustering

Single-link method produced numerous clusters, and the results continuously improved at each step of the clustering process until the similarity fell below .7142857. After that point, the accuracy (Precision and Recall) started to decrease when clusters having smaller value of similarity were forced to merge because there was no threshold value set to stop forced clustering. The greatest accuracy (Precision and Recall) was obtained after 27 steps, and the clusters generated are shown below

1{1} **2**{2,3,4,5,19,20,21,22,23,24,25} **3**{6,7,8,9,10,32,33} **4**{11,12,13,14,34,35} **5**{15,17,16,18} **6**{26,27,28,29} **7**{30} **8**{31}.

Case Study I: Results Produced (Single-Link Clustering) [Results I]

The values of precision and recall after comparing these results with the results of the expert are shown in table 6.1

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 100%	Precision = 100%	Precision = 100%	
Recall = 100%	Recall = 100%	Recall = 50%	

Table 6.1 Precision and Recall

The value of recall was significantly less for subsystem 7 because of very small number of functions in that subsystem, but over all, the results were quite encouraging.

Complete-Link-Clustering

The results produced by complete link clustering were quite accurate as well. The results improved continuously until the similarity fell below .7142857. After that the accuracy (Precision and Recall) deteriorated significantly due to the effect of forced clustering. The greatest accuracy (Precision and Recall) was obtained after 25 steps, and the clusters generated are shown below

1{1,16,18,26,27,28,29**}** **2{**2,19,3,20,21,22,23,24,4,25,5**}** **3{**6,7,8,9,10,32,33**}**
4{11,14,12,13,34,35**}** **5{**15,17**}** **6{**30,31**}**

Case Study I: Results Produced (Complete-Link Clustering) [Results I]

The values of precision and recall after comparing these results with the results of the expert are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 14%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 100%	Precision = 57%	Precision = 100%	
Recall = 50%	Recall = 100%	Recall = 100%	

Table 6.2 Precision and Recall

The results for single-link clustering were better then complete-link clustering, due to small value of minimum entity-entity similarity across the clusters which would have otherwise been merged.

Results II

These results were obtained by comparing functions on the basis of function properties and function-calls. The experiment was done a couple of times by giving different weights to function-calls and function properties. Although the best results obtained by selecting these options were the same as that of the 'Results I', there were some interesting findings. The similarities between functions in the same cluster were greater then the similarity between them in 'Result I'. This was due to the fact that most of the functions that were combined in the previous results, called the same

functions, and there were very few instances where dissimilar functions (as given by the expert) called the same functions. Thus the results confirming the fact the function-calls can play an important in determining correct function clusters.

The most accurate clusters generated by selecting these options are indicated below

Single-Link-Clustering

1{1} **2**{2,3,20,21,22,23,24,4,25,5,19} **3**{6,7,8,9,10,32,33} **4**{11,12,13,14,34,35} **5**{15,17,16,18} **6**{26,27,28,29} **7**{30} **8**{31}

Case Study I: Results Produced (Single-Link Clustering) [Results II]

The values of precision and recall for the each of the Subsystem (in Figure 6.1) based on these results are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 100%	Precision = 100%	Precision = 100%	
Recall = 100%	Recall = 100%	Recall = 50%	

Table 6.3 Precision and Recall

Complete-Link-Clustering

1{1,16,18,26,27,28,29} **2**{2,5,3,20,21,22,23,24,4,25,19} **3**{6,7,8,9,10,32,33} **4**{11,12,13,14,34,35} **5**{15,17} **6**{30,31}

Case Study I: Results Produced (Complete-Link Clustering) [Results II]

The values of precision and recall are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 14%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 100%	Precision = 57%	Precision = 100%	
Recall = 50%	Recall = 100%	Recall = 100%	

Table 6.4 Precision and Recall

Results III

The most interesting point in the evaluation came when the functions were compared on the bases of parameter types and function properties. This particular experiment clearly showed how important it was to distinguish between the user-defined and pre-defined data types.

Another importance of this test was that most of the functions in the code had parameters, so this test was always going to produce some important information regarding program understanding and particularly the testing of the software, as to whether it generated the correct results or not.

Initially when the test was made, there was no distinction made between user-defined and pre-defined parameter types. The results produced were quite bad as compared to results produced earlier.

Single-Link Clustering

For single-link clustering, the first few steps produced correct results, after that, the clustering accuracy greatly reduced and it was resulting in a one large cluster into which all other entities were merged one-by-one. The best results produced are

1{1,7,8,9,10,32,11,13,12,14,31,34,35,16,18,26,27,28,29,6} **2**{2,3,4,5} **3**{15,17} **4**{19,20,21,22,23,24,25} **5**{30} **6**{33}

Case Study I: Results Produced (Single-Link Clustering) [Results III]

The values of precision and recall are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 5%	Precision = 100%	Precision = 30%	Precision = 100%
Recall = 100%	Recall = 63%	Recall = 85%	Recall = 50%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 30%	Precision = 20%	Precision = 100%	
Recall = 100%	Recall = 100%	Recall = 50%	

Table 6.5 Precision and Recall

Complete-Link Clustering

The results produced by complete-link clustering were much better then the single-link clustering, but still, the accuracy was quite low as compared to the results in the earlier experiments.

1{1,26,27,28,29,11,13,12,14,34,35} **2**{2,3,4,5,19,20,21,22,23,24,25}

3{6,7,8,9,10,32} **4**{15,17,16,18} **5**{30,31} **6**{33}

Case Study I: Results Produced (Complete-Link Clustering) [Results III]

The values of precision and recall are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 9%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 85%	Recall = 100%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 54%	Precision = 36%	Precision = 100%	
Recall = 100%	Recall = 100%	Recall = 100%	

Table 6.6 Precision and Recall

If the weight of the functions properties was made two to three times more then the parameter types, then the contribution of the parameter types in determining the clusters was significantly reduced and the accuracy of the results became much better. But the important issue was to understand why the results deviated so much form the correct solution when parameter types were contributing more in finding the results.

After carefully examining the code, it became clear that most of the functions that had parameters, predominantly used “int” and “string” (both are pre-defined parameter types) as their parameter types along with other parameter types if any. Therefore most of the functions were using similar pre-defined parameter types. There were very few user-defined parameter types to make any difference, therefore the functions could not be distinguished correctly on the basis of their parameter types.

The next reasonable step was to distinguish between user-defined and pre-defined parameter types. The weight given to the user-defined parameter types was twice that of the pre-defined parameter types.

The best results produced are

Single-Link Clustering

1{1} **2**{2,3,4,5,19,20,21,22,23,24,25} **3**{6,7,8,9,10,32,33}
4{11,13,12,14,34,35,31} **5**{15,17,16,18} **6**{26,27,28,29} **7**{30}

Case Study I: Results Produced (Single-Link Clustering) [Results III]

The accuracy improved tremendously as shown below

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 86%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 100%	Precision = 100%	Precision = 100%	
Recall = 100%	Recall = 100%	Recall = 50%	

Table 6.7 Precision and Recall

Complete-Link Clustering

1{1,26,27,28,29} **2**{2,3,4,5,19,20,21,22,23,24,25} **3**{6,7,8,9,10,32,33}
4{11,13,12,14,34,35} **5**{15,17,16,18} **6**{30,31}

Case Study I: Results Produced (Complete-Link Clustering) [Results III]

The accuracy (Precision and Recall) of the results is

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 20%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5	**Subsystem 6**	**Subsystem 7**	
Precision = 100%	Precision = 100%	Precision = 100%	
Recall = 100%	Recall = 100%	Recall = 100%	

Table 6.8 Precision and Recall

The results produced were significantly improved as compared to the results when user-defined and pre-defined parameter types were not distinguished or when they were given equal weights.

Results IV

These results were obtained when function properties were selected along with local variable types. The results showed exactly the same pattern as that of the 'Results IV'. The clusters produced in both the cases (Single-Link clustering and Complete-Link Clustering) were significantly improved when differentiation was made between user-defined and pre-defined types, and the user-defined types were given more weight as compared to pre-defined types.

6.1.2 Class Clustering

There are 13 classes in the CodeClusteringApplication. Only three major clusters were suggested by the expert. The expert mental model for the classes is given in the figure 6.2.

As in the case of functions, the numbering given to each of the class is equal to its Id in the actual database, which was used as a source (of information) by the clustering application. These numberings are used to represent the classes in further discussions below in order to make the clustering process easier to explain.

Subsystem 1	Subsystem 2	Subsystem 3
Class Names	**Class Names**	**Class Names**
2. Clustering	9. AboutBox	4. DatabaseManager
3. CommonComparisonTasks	10.AdvancedOptionChecks	7. QueryMaker
6. FunctionComparison	11.Form1	
12. ClassComparison	13.AdvancedWeightOptions	
Subsystem 4	**Subsystem 5**	**Subsystem 6**
Class Names	**Class Names**	**Class Names**
1. CentralDispatcher	5. FileHandler	8. SimilarityMetrics

Figure 6.2 Case study I: Expected results (Class Clustering)

[CodeClusteringApplication]

Subsystem 1: All the classes in this group are responsible for calculating the similarity matrices and performing clustering of the code.

Subsystem 2: All the classes in this group are responsible for user interface of the project.

Subsystem 3: The classes in this group are responsible for data base related activities.

Subsystem 4: There is only one class in this group, and it is the central dispatcher of the application. It determines the flow of the whole application depending on the selections done by the user.

Subsystem 5: This subsystem is responsible for all the file handling capabilities of the system, all the results of the system are written to file in specific format that helps other applications to use the files for further processing.

Subsystem 6: The class in this subsystem is responsible for handling the similarity matrix that contains all the information about the similarities between entities in the data base.

Figure 6.2a Case Study I: Expected results (Function Clustering)

[CodeClusteringApplication]

Explanation of the Clusters suggested by the expert

The above diagram can be represented in a simpler format by showing the classes in groups, only in terms of their IDs. The representation then becomes.

1{2,3,6,12}, **2**{9,10,11,13}, **3**{4,7}, **4**{1}, **5**{5}, **6**{8}

Case study I: Expected results.
[CodeClusteringApplication]

Numerous results were produced by selecting different options and were compared with the expert's mental model. These results are as follows

Results I

These results show the single-link clustering and complete-link clustering behaviour when only class properties were selected for cluster formation.

Single-Link Clustering

The best results produced by single-link clustering algorithm are indicated below

1{1} **2**{2,3,6,12} **3**{4} **4**{5} **5**{7} **6**{8} **7**{9,11,10,13}

Case Study I: Results Produced (Single-Link Clustering) [Results I]

The values for precision and recall are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5	**Subsystem 6**		
Precision = 100%	Precision = 100%		
Recall = 100%	Recall = 100%		

Table 6.9 Precision and Recall

The algorithm was unable to detect the similarity between two classes 4 and 7 as indicated by the expert. The major reason behind it was that, most of the properties of class 4 and 7 (That is, Query Maker and DatabaseManager) had null values (That is, the classes did not have any values for those properties), as a result the algorithm was

unable to detect any significant similarity between the two classes to join them in the same cluster.

Complete-Link Clustering

The results of complete-link clustering were similar to that of single-link clustering with few changes in the similarities between the classes (due to the difference in the way the two algorithms form clusters).

The best results produced by Complete-link clustering algorithm are

1{1} **2**{2,3,6,12} **3**{4} **4**{5} **5**{7} **6**{8} **7**{9,11,10,13}

Case Study I: Results Produced (Complete-Link Clustering) [Results I]

The values for precision and recall are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5	**Subsystem 6**		
Precision = 100%	Precision = 100%		
Recall = 100%	Recall = 100%		

Table 6.10 Precision and Recall

Results II

These results were obtained by selecting class member types along with the class properties to determine clusters.

The results produced by both form of clustering algorithms (Single-link and Complete-link) were similar to the ones in 'Results I'. The only difference was that the similarity between the classes with in same clusters increased due to the fact that the classes in same clusters used similar types of variables.

Single-Link Clustering

The best results produced by single-link clustering algorithm are as follows

1{1} **2**{2,3,6,12} **3**{4} **4**{5} **5**{7} **6**{8} **7**{9,11,10,13}

Case Study I: Results Produced (Single-Link Clustering) [Results II]

The values for precision and recall are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5	**Subsystem 6**		
Precision = 100%	Precision = 100%		
Recall = 100%	Recall = 100%		

Table 6.11 Precision and Recall

Complete-Link Clustering

The best results produced by Complete-link clustering algorithm are

1{1} **2**{2,3,6,12} **3**{4} **4**{5} **5**{7} **6**{8} **7**{9,11,10,13}

Case Study I: Results Produced (Complete-Link Clustering) [Results II]

The values for precision and recall are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5	**Subsystem 6**		
Precision = 100%	Precision = 100%		
Recall = 100%	Recall = 100%		

Table 6.12 Precision and Recall

6.2 Case Study II (Credit scoring e-service)

Credit scoring e-service contains 25 classes and 76 functions. This case study helped in getting some idea about the scalability of the software, but the real test of scalability is dealt with in the third case study.

6.2.1 Function Clustering

According to the expert mental model, functions are divided into five clusters.

The division of the functions in five groups is shown in Figure 6.3.

The numbering given to each of the function follows the same concept as previously explained.

The only information that the author have about the functions in this code is their properties and their return types, therefore only these two attributes are used in calculating the similarities between the functions.

Subsystem 1
Function Names
38. Page_Load
39. btnLogin_Click
40. Page_Load
41. fillGrid
42. btnSubmit_Click
43. Page_Load
44. populateColorList
45. populateLayoutList
46. Page_Load
47. lnkLogout_Click
48. lnkUpload_Click
49. lnkScore_Click
50. lnkCustomize_Click
51. Linkbutton1_Click
52. Page_Load
53. Page_Load
54. checkAddedParams
55. setMethodList
56. uploadData
57. verifyXML
58. btnReset_Click
59. btnUpload_Click
60. Page_Load
61. populateList
62. lstID_SelectedIndexChanged
63. btnSubmit_Click
64. Page_Load
65. populateTable
66. populateDataGrid
67. Page_Load
68. Page_Load

Subsystem 2
Function Names
25. LoanDetails
24. LoanDetails
37. PersonalDetails
36. PersonalDetails
5. setAccountNumber
8. setAccountType
9. setCreditLimit
26. setCustID
3. setCustID
21. setDate
7. setFIName
19. setID
29. setLoanAmount
6. setOpeningDate
20. setPassword
27. setPurpose
30. setRepaymentPeriod
4. setSortCode
28. setTotalCostOfItem
2. BankDetails
1. BankDetails
17. FI
18. FI

Subsystem 4
Functions Names
76. invokeWebservice

Subsystem 3
Function Names
14. getAccountNumber
11. getAccountType
10. getCreditLimit
35. getCustID
16. getCustID
22. getDate
12. getFIName
32. getLoanAmount
13. getOpeningDate
23. getPassword
34. getPurpose
31. getRepaymentPeriod
15. getSortCode
33. getTotalCostOfItem

Subsystem 5
Function Name
69. XMLHelper
70. getInstance
71. verifyXML
72. generateCustID
73. parseXML
74. riteXMLPreferencesFile
75. getQuery

Figure 6.3 Case Study II: Expected results (Function Clustering)
[Credit scoring e-service]

Subsystem 1: interface (user defined and generated both)
Subsystem 1 contains user-interface related functions. These functions include both user-defined functions as well as functions generated by the interface. The basic purpose of these functions is to change/ modify interface.

Subsystem 2: (constructor and set)
Every table in database have a corresponding class in implementation with the same parameters as found in the corresponding database table. Subsystem 2 contains the functions that set the value of these parameters to the argument provided in the function. These functions are of type setXX() as well as constructors of each class (as constructor also initializes the value of each parameter).

Subsystem 3: (get)
This group contains functions of type getXX(). These functions are used to retrieve the value of database entity.

Subsystem 4: (invoke web service)
This group contains only one function and its functionality is different from all other functions of the system because it invokes the web service which will apply credit scoring algorithm.

Subsystem 5: (business methods)
This group contains the functions which implement business logic of the system. These functions form the core functionality of the system and are different from rest of the functions of system.

Figure 6.3a Case Study II: Expected results (Function Clustering)
[Credit scoring e-service]
Explanation of the Clusters suggested by the expert

The above diagram can be shown in a simpler format by representing the functions in terms of their Ids. The representation then becomes.

1{38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68**}**

2{25,24,37,36,5,8,9,26,3,21,7,19,29,6,20,27,30,4,28,2,1,17,18**}**

3{14,11,10,35,16,22,12,32,13,23,34,31,15,33**}** **4{**76**}** **5{**69,70,71,72,73,74,75**}**

Case study II: expected results

[Credit scoring e-service]

The clustering results obtained by the software tool are as follows

Results I

These results were calculated by only selecting the function properties. Both, single-link and complete-link clustering algorithms were used. The results obtained are shown below.

Single-link Clustering

This method produced encouraging results. From the beginning, each step was continuously improving upon the results of the previous step. The greatest similarity between any two pair of clusters was 1. This similarity was between constructors of the same class. The best results were produced after 70 steps, and the similarity (between clusters) at that point was reduced to 0.5714286. Six clusters were left at that stage. After that similarity between clusters was significantly reduced because of forced clustering.

The best results produced are shown below

1{38,39,40,42,43,46,52,53,60,64,67,68,47,48,49,50,51,58,59,62,63,41,44,45,54,55,56,57,61,65,66**}**

2{1,2,3,4,5,6,7,8,9,19,20,21,26,27,28,29,30,17,18,36,37,24,25**}**.

3{10,11,12,13,14,15,16,22,23,31,32,33,34,35**}**

4{69**}** **5{** 70,72,71,73,75,74**}** **6{**76**}**

Case Study II: Results Produced (Single-Link Clustering) [Results I]

The values of precision and recall for the subsystems are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5			
Precision = 100%			
Recall = 85%			

Table 6.13 Precision and Recall

The only function that was out of its cluster was the function No.69 (that is, XMLHelper), which was a constructor of the XMLHelper class. After examining the function, it was discovered that almost none of the properties (which were used for clustering) of this function resembled any other function in the same cluster.

Complete-link clustering

The results produced by complete-link clustering were almost identical to the single-link clustering. Although the sequence in which the functions were added to the clusters was a little different (because of the difference in the concept the two techniques follow, see section 2.5.3), the eventual best results were exactly the same to that of single-link clustering. The similarity value at which the best results were achieved was less then that of single-link clustering. The best results were achieved at sim=0.4285714.

The best results produced are

1{38,39,40,42,43,46,52,53,60,64,67,68,47,48,49,50,51,58,59,62,63,41,44,45,54,55,56,57,61,65,66**}**

2{1,2,3,4,5,6,7,8,9,19,20,21,26,27,28,29,30,17,18,36,37,24,25**}**.

3{10,11,12,13,14,15,16,22,23,31,32,33,34,35**}**

4{69**}** **5{** 70,72,71,73,75,74**}** **6{**76**}**

Case Study II: Results Produced (Complete-Link Clustering) [Results I]

The values of precision and recall for the subsystems are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5			
Precision = 100%			
Recall = 85%			

Table 6.14 Precision and Recall

Results II

These results were generated by using function properties along with their return types.

Single-Link Clustering

As expected the results produced by selecting these options were identical to the ones produced above. The only difference was that the similarities between the functions in the same cluster were greater at each step then the similarity between them in the Result 1. The reason behind these observations was that most of the functions that belong to the same group (as mentioned by the expert) have similar return types. Therefore the similarity between them increased as the return types were included for evaluating the similarity between the functions. Similarly the difference between two different clusters also increased because the return types of the functions belonging to different clusters were different. So over all results obtained were stronger then the previous results.

The best results produced are

1{38,39,40,42,43,46,52,53,60,64,67,68,47,48,49,50,51,58,59,62,63,41,44,45,54,55,56,57,61,65,66**}**

2{1,2,3,4,5,6,7,8,9,19,20,21,26,27,28,29,30,17,18,36,37,24,25**}**.

3{10,11,12,13,14,15,16,22,23,31,32,33,34,35**}**

4{69**}** **5{** 70,72,71,73,75,74**}** **6{**76**}**

Case Study II: Results Produced (Single-Link Clustering) [Results II]

The values of precision and recall for the subsystems are given below

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5			
Precision = 100%			
Recall = 85%			

Table 6.15 Precision and Recall

Complete-Link Clustering

The results produced by complete-link clustering were identical to the single-link clustering. The sequence in which the functions were added to the clusters was a little different because of the difference in the concept the two techniques follow (section 2.5.3). The eventual best results were exactly the same to that of single-link clustering.

The best results produced are

1{38,39,40,42,43,46,52,53,60,64,67,68,47,48,49,50,51,58,59,62,63,41,44,45,54,55,56,57,61,65,66**}**

2{1,2,3,4,5,6,7,8,9,19,20,21,26,27,28,29,30,17,18,36,37,24,25**}**.

3{10,11,12,13,14,15,16,22,23,31,32,33,34,35**}**

4{69**}** **5{** 70,72,71,73,75,74**}** **6{**76**}**

Case Study II: Results Produced (Complete-Link Clustering) [Results II]

The values of precision and recall for the subsystems are

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 50%	Recall = 100%
Subsystem 5			
Precision = 100%			
Recall = 85%			

Table 6.16 Precision and Recall

6.2.2 Class Clustering

According to the expert mental model, classes were divided onto five clusters.
The division of the classes in five groups is shown in Figure 6.4.
The numbering given to each of the class follows the same concept as previously explained.

The only information that the author has about the classes in this code is their properties, therefore only the class properties were used in calculating the similarities between classes.

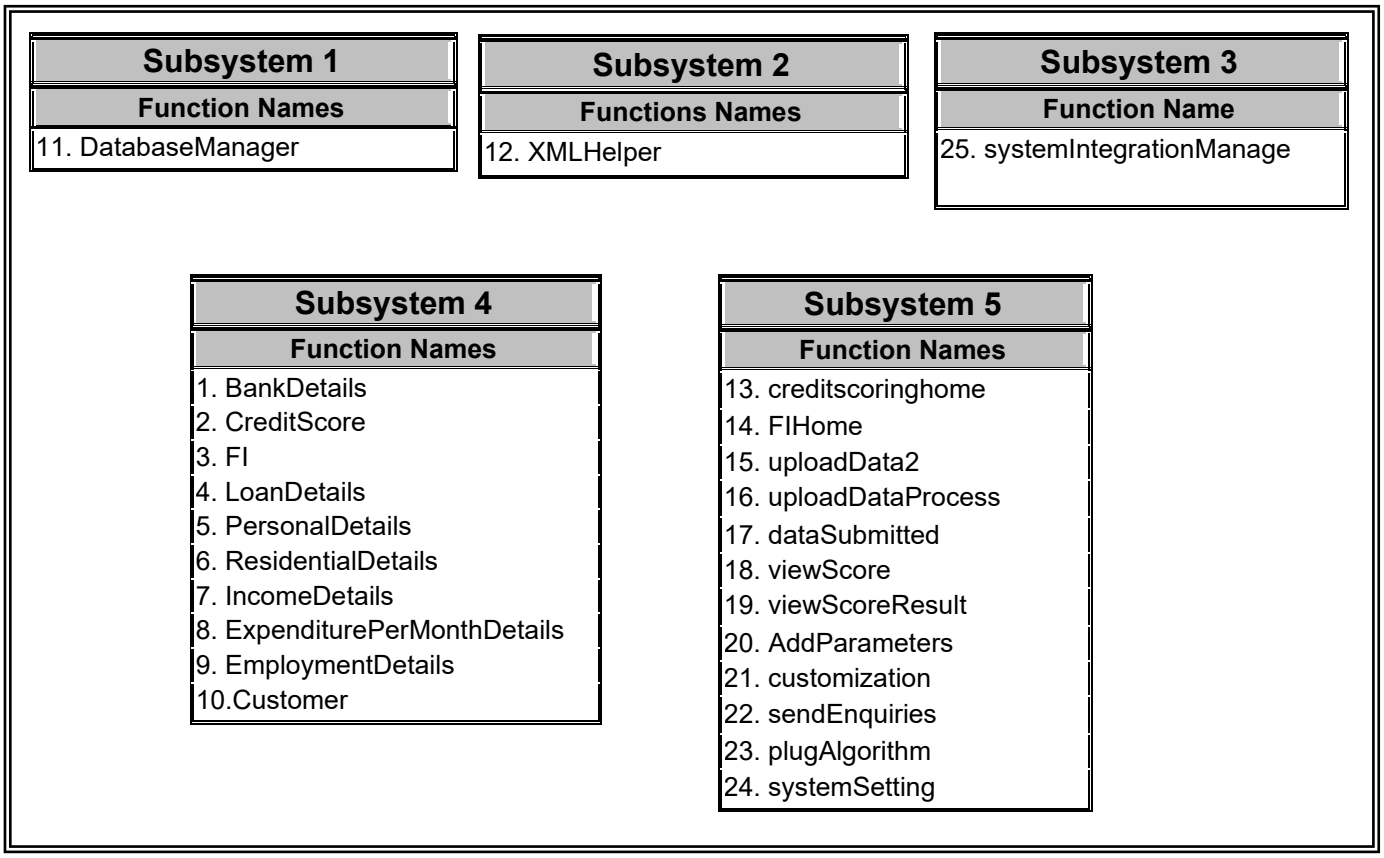


Figure 6.4 Case Study II: Expected results (Class Clustering)
[Credit scoring e-service]
Explanation of the Clusters suggested by the expert

Subsystem 1: This subsystem contains only one class. And it is responsible for all the actions performed on the database of the system.

Subsystem 2: This group contains the classes which implement business logic of the system. These classes form the core module of the system and are different from rest of the classes of the system.

Subsystem 3: This subsystem contains only one class and its purpose is different from all other classes of the system because it invokes the web service which will apply credit scoring algorithm.

Subsystem 4: Every table in database have a corresponding class in implementation with the same parameters as found in the corresponding database table. Subsystem 4 contains the classes that deal with setting and getting the value of these parameters.

Subsystem 5: Subsystem 5 contains user-interface related classes. The basic purpose of these classes is to change/ modify interface

Figure 6.4a Case Study II: Expected results (Class Clustering)
[Credit scoring e-service]
Explanation of the Clusters suggested by the expert

The above diagram can be shown in a simpler format by representing the classes in terms of their class Ids. The representation then becomes.

1{11} **2**{12} **3**{25} **4**{1,2,3,4,5,6,7,8,9,10}
5{13,14,15,16,17,18,19,20,21,22,23,24}

Case study II: Expected results
[Credit scoring e-service]

The clustering results obtained by the software tool are as follows

Results I

These results were calculated by selecting the class properties only. Both, single-link and complete-link clustering algorithms were used. The results are shown below.

Single-link Clustering

This method resulted in a perfect match. The best results produced by single-link clustering were exactly the same as that produced by the expert. The similarity value at which the best results were achieved was sim=0.7272727.

The results are shown below

1{11} **2**{12} **3**{25} **4**{1,2,3,4,5,6,7,8,9,10}
5{13,14,15,16,17,18,19,20,21,22,23,24}

Case Study II: Results Produced (Single-Link Clustering) [Results I]

The values of precision and recall for the subsystems are given as

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5			
Precision = 100%			
Recall = 10%			

Table 6.17 Precision and Recall

Complete-link clustering

The results produced by complete-link clustering were identical to the single-link clustering. The similarity value at which the best results were achieved was sim=0.7272727.

The best results produced are

1{11} **2**{12} **3**{25} **4**{1,2,3,4,5,6,7,8,9,10}

5{13,14,15,16,17,18,19,20,21,22,23,24}

Case Study II: Results Produced (Complete-Link Clustering) [Results I]

The values of precision and recall for the subsystems are given as

Subsystem 1	**Subsystem 2**	**Subsystem 3**	**Subsystem 4**
Precision = 100%	Precision = 100%	Precision = 100%	Precision = 100%
Recall = 100%	Recall = 100%	Recall = 100%	Recall = 100%
Subsystem 5			
Precision = 100%			
Recall = 100%			

Table 6.18 Precision and Recall

6.3 Case Study III (Administration of Books Publishing)

This case study is carried out solely to test the scalability of the software tool. The software that was tested contains large number of functions and classes. There are 5976 functions and 1242 classes. Expert mental model for this software was not available, for this reason there is no way to check the results produced by the tool. This cases study is only to measure the suitability of the tool for large software systems.

6.3.1 Class Clustering

Results I

The first test was made by clustering classes only on the basis of their properties.

The results were obtained within a couple of minutes (1 to 1.5 minuets, depending on the specification of the computer with which the tests were carried out). These results were quite encouraging as the tool was able to cluster large systems in very less time.

Results II

This test was made by clustering classes on the basis of class member types. The time the software tool took (8 to 12 minutes depending on the specification of the computer with which the tests were carried out) to cluster the classes based on their

member types was quite long as compared to that of class properties. The reason behind this outcome is the design of the database that implemented the input model. It can be seen in the database design (Figure 4.7) that there are two tables whose combined information specifies which class has which type of member data. The two tables are “ClassHasMembers” and “ClassMemberType”.

The “ClassHasMembers” table only has two fields ClassID and ClassMemberID. This table only indicates the link between the classes and their member data types. The “ClassMemberType” table contains all the information about the member data types. This arrangement was made in order to avoid unnecessary repetition of data in the tables and as a result the space requirement of the data base was significantly reduced. The side affect of this arrangement was that the tool now had to access data from two tables to get information about member types of any class (that is “ClassHasMembers” table and “ClassMemberType” table). As a result the complexity and the number of queries made to the database are greater for member data types as compared to function properties (The function Id and its properties were placed in the same table, because it was not causing any repetition of the entries in the tables).

This is one major area where further work needs to be done in order to improve the scalability of the software for other attributes, while keeping the space requirement of the database to its minimum.

6.3.2 Function Clustering

The only information that is known about the functions (in this code) is their properties, therefore the results were obtained only on the basis of function properties.

Results I

The test was made by clustering functions only on the basis of their properties.
The results were obtained within 15 to 20 minutes, depending on the specification of the computer with which the tests were carried out.

The number of functions is approximately five times that of classes but the time to calculate function clusters was in some cases 15 times more then that of classes. This observation can be explained by the equation 6.1.

No. Of Comparisons Required = **(n(n-1) / 2)** ———————— **Equation 6.1**

This equation indicates the number of comparisons required between n numbers of entities, to calculate all the similarities before starting the clustering process. It is evident from the equation that the number of comparisons required is directly proportional to the square root of total number of entities. Therefore as the number of entities grow the number of comparison between the entities increase at a much higher rate, and as a result takes much more time then in cases where entities are fewer.

6.4 Requirements Fulfilled

Numerous functional and non-functional requirements were specified in the requirement analysis section of this report. This section discusses all the requirements one by one with a little description of whether the requirement is completely fulfilled, partially fulfilled or unfulfilled. The reasons for the non-fulfilment of requirements are also provided.

6.4.1 Functional requirements

1. **Specification of input model**

   Function and class input model are specified and implemented that cover all the four requirements imposed on the input models.

   Status: Fulfilled

2. **Specification of the similarity metrics**

   Similarity matrices are specified (majority of them with the help of **[14]**), that completely cover all the four requirements imposed on the similarity measures in the requirement analysis section.

   Status: Fulfilled

3. **Specification of the clustering algorithm**

   Hierarchical (agglomerative) clustering algorithms are incorporated in the clustering tool, and the reasons for this selection are given in sections 4.3. These algorithms cover majority of the requirements imposed on the clustering algorithms.

   Status: Fulfilled

4. **Actual representation of the system**

   The two case studies carried out in this chapter clearly indicate that the clustering tool clustered the program entities according to the expert mental model with high degree of accuracy. Although there are cases where the tool do not entirely produce the correct results but still the results are quite encouraging and satisfactory.

   Status: Fulfilled.

5. **Design**

   The tool specified a high-quality design, and extracted modules that implemented a known concept.

   Status: Fulfilled.

6. **Ability to handle different types of attributes**

   As shown in the design in section 4.8 the attributes are defined by numerical, binary, and categorical fields. And the application is able to handle all the three types of data. Therefore this requirement was completely fulfilled.

   Status: Fulfilled.

7. **Insensitivity to the order of input records**

   The clustering tool is not affected by the change in the order in which data is provided to it.

   Status: Fulfilled.

8. **High dimensionality of the input data**

   Most of the tables have dimensionality greater then 5. The class and function properties tables have even greater dimensionality (7 and 11 respectively). The functionality of the software does not reduce notably with the increase in the dimensionality of the input data.

   Status: Fulfilled.

9. **Prescribed Methodology**

   The results produced by the clustering tool are incremental and step-by-step. The results clearly help in the user to detect (in case of a problem) where the solution starts to deviate from the actual "expert decomposition".

   Status: Fulfilled.

10. **Reporting**

    The results are produced in a textual format and the software is not developed to produce a graphical representation of the results. Instead another tool is used to produce the graphical output of the results obtained.

    Status: Partially Fulfilled.

11. **Model exporting**

    This tool produces a result in a text file that can be easily used by other programs for ongoing use. In fact the results produced by the tool were used by software to produce graphical outputs.

    Status: Fulfilled.

12. Model Validation

The process of validation and verification of the results is not automated. The software for automatically evaluating the results was not developed due to lack of time.

Status: Not Fulfilled.

13. Data Filtering

The user can select a subset of data and not all the data to produce clustering.

Status: Fulfilled.

14. Error reporting

The software notifies the user if the user selects some incorrect options or specifies some incorrect values during the clustering process. However in case of the clustering results obtained, the software does not indicate where the results actually start to deviate from the actual representation. Similarly the software does not provide any feed back messages for the debugging process, this has to be done manually by checking the results produced by the software.

Status: Partially Fulfilled.

15. Handling noise

The software has the functionality to cope up with the missing or null values. But it is not able to detect or correct values entered incorrectly.

Status: Partially Fulfilled.

16. Data cleansing

The tool allows the user to modify spurious values in the data set, so that the data that the tool work on is of high-quality

Status: Fulfilled.

6.4.1 Non-Functional requirements

1. **Robustness**

   The clustering tool developed runs consistently without crashing and does not require any monitoring or intervention by the user.

   Status: Fulfilled.

2. **Cluster Size**

   The clustering algorithm clusters the program entities on the basis of their similarities. In majority of the cases it does not form huge singleton clusters. How ever the nature of the output depends on the selection of the attributes and the information provided to the clustering tool. This requirement is too vague for the tool to be assessed against it.

   Status: Cannot be justified.

3. **Automation**

   The clustering tool is completely automated and does not require any user intervention. The options provided to the user are optional, if the user does not wish to use these options, the tool selects them automatically.

   Status: Fulfilled.

4. **User Interface and learning curve**

   The user interface of the tool is very easy to understand and work with and the user can easily adjust to its usage.

   Status: Fulfilled.

5. **Stability**

   This requirement is not tested and evaluated against the clustering tool because of lack of time.

   Status: Not Fulfilled.

6. **Scalability**

   The third case study was carried out solely to check the clustering tool against scalability. The results produced by the clustering tool are quite encouraging. The

clustering is done quite fast when working with only the function and class properties.

In that situation, the tool clustered nearly 1300 classes and 6500 functions in just a couple of minuets. The time requirement of the tool increases significantly when other attributes are added for the clustering purposes (in the case of such large soft wares), and it is this area that need further research.

Status: Partially Fulfilled.

7. **Results Interpretability and usability**

   The results are produced in a form of a text file. Although these results are satisfactory in case of small applications, but the understanding of the results get quite difficult as the size if the software increases. Therefore the output was produced in such a format that could be used by another software application to produce its visual form that could be easily understood by the user.

   Status: Partially Fulfilled.

8. **Efficiency**

   As the third case study suggest, the efficiency of the tool is quite satisfactory, though there is a great room for improvement and greater amount of testing is also required. The results produced by the software are in text files which are of normal size, considering the format of the output (that is, to save each and every step of the clustering process).

   Status: Partially Fulfilled.

9. **Adaptability**

   The tool was used to cluster C++ and C# code. It can also be used easily with very few or no changes to work with other languages. So the adaptability of the tool is quite satisfactory.

   Status: Fulfilled.

7

Conclusion & Further Work

The report concludes, firstly, with a retrospective overview of the work that has been undertaken. This is followed by some conclusions which have been made about the project as a whole. After that a consideration is made of what further work should stem from that performed here.

7.1 Review

Before any work could be commenced on formulating any kind of solution, a through investigation of the domain needed to be performed. To appreciate the current situation, background research was carried out in the field of data mining with special emphasis on the clustering techniques and cluster analysis. All the aspects of the clustering process and especially the use of clustering techniques for software maintenance were thoroughly investigated.

To correctly capture the project requirements, an assessment of the needs of software maintainers, what problems they currently faced and the use of the clustering tools to help in the process of software maintenance was carried out. A list of 16 functional and 9 on-functional requirements was defined. These requirements were then ranked by importance, so that the focus of the solution design could be shifted towards the core needs of the user and that the resulting implementation would be sure to at least deliver the more important features, should time run short.

A design that would attempt to satisfy each and every requirement was then planned and constructed. The design was expressed through a combination of formal and informal techniques, including block diagram, use cases, class diagrams, sequence diagrams and natural language, so as to consider the tool's functionality, use and internal operation.

A visual C++ implementation for this design was then produced over an intensive 10 week period. Each aspect of the functionality was implemented in an order that reflect the priority of the requirements that it fulfilled. The more complex, difficult or interesting components of the source code were presented.

After doing some testing in order to ensure that the resulting software behaved as desired, the tool had reached a finalised state. After that, an extensive evaluation of the tool was made in order to ensure that it fulfilled most of the requirements that were initially set in the requirements phase of the project.

7.2 Conclusions

This work addresses the issue of the feasibility of clustering functions and classes of programs, depending on the types and use of number of parameters. It also investigated the appropriateness of different clustering strategies. Finally it explored the suitability of the approach for different types of programming languages and sizes of programs. A software tool was developed for this reason.

The tool was used to obtain the subsystem abstraction of 3 programs. The accuracy of the results was evaluated by comparing the sub-system abstractions with expert's mental models for two of the programs. The produced results were found to be meaningful in most cases.

In addition to evaluating the accuracy of the results, the feasibility of this approach was evaluated by assessing the complete process of obtaining a subsystem abstraction of a program. Overall, the results of this initial investigation demonstrated the potential of this approach for obtaining a subsystem abstraction of a program and for identifying interrelations between classes and functions.

The tool realises the original design virtually in its entirety and effectively fulfils the majority of its requirements, including the core requirements. The project is deemed to have been successful, overall, but the tool cannot be considered to be truly complete until some handful of issues with it has been removed. The process that this

project adhered to is considered to be satisfactory, but not without faults or room for improvements.

The author has found this project to be an invaluable opportunity to consolidate and expand upon his skill set, in all areas, and has acquired a confidence that will allow his skills to be applied effectively to future projects.

7.3 Further Work

This section provides a number of possible enhancements to the approach and the tool in particular, that should be performed in order to achieve a complete solution to the problem of source code analysis.

- **Automatic derivation of input model**

  The process of extracting the input model from program and performing the relevant pre-processing on these data is time consuming as there is no automated method for supporting it. Only a subset of the input data is acquired using a parser, rest of the information has to be filled manually by using different editors search facilities. The precise method needs to be further specified and enriched to facilitate as much automation of the derivation of the input model as possible.

- **Further testing on larger programs**

  Evaluation was carried out for two small/medium size programs with few thousand lines of code. Evaluation of the tool in the third case study for larger software was carried out only to test the scalability of the software. The results could not be validated as there was no mental model available for the program. In order to achieve a better evaluation of the proposed results, more tests should be performed on larger programs to asses how method scales to deal with real industrial scale systems. However, in order to proceed to these tests, the previous enhancement should be implemented first, so as to get the input data from these programs automatically.

- **Use of incremental Clustering technique**

  The clustering technique used is not incremental. That is, the entire process has to be conducted all over again on all entities, if there are changes made to the software (which results in addition of new entities) being examined. It would be desirable if this method could be incremental in a manner accommodating changes in software (that is, the method could update clusters only by processing newly added entities) as the assumption of static software is unrealistic.

- **Clustering based on dynamic dependencies**

  This work attempts to cluster large software systems based on static dependencies between software artefacts. However, it would be highly desirable to cluster software entities based on dynamic dependencies. For example, taking into consideration the number of times a particular procedure call or data reference took place during the runtime of the software. It is quite likely that dynamic clustering will also provide interesting insight into a system's structure.

- **Testing other clustering algorithms**

  Hierarchical agglomerative clustering is used with Single-link and complete-link algorithms only. The effects of weighted-linkage rules and un-weighted linkage rules would also be interesting to analyse. Additional data mining techniques like classification, association rules could also be used for experimentation purposes.

- **Source code analysis at the statement level**

  The suggested approach concentrated on a model consisting of modules (functions, classes and procedures) and the ultimate aim was to create program decomposition in groups containing interrelated modules. On the other hand, the tool can also be used in order to perform a source code analysis at the statement level. To do so, the input data should consist of records representing statements and the final results will produce groups of related statements. This kind of analysis is useful, when local parts of program need to be understood.

- **Using other evaluation techniques**

  In this work, only Precision and Recall are used as quantitative elements in judging the correctness of the results of the tool (refer to section 2.6 for explanation about precision and recall). There are other evaluation techniques like *MoJo distance* **[18]** *Edge similarity measurements* **[19]** and many others that need to be used in order to get better understanding of the results produced by the software tool.

References

1. Anquetil, N.; Lethbridge, T.C.; *Experiments with clustering as a software remodularization method.* Reverse Engineering, 1999. Proceedings. Sixth Working Conference on , 6-8 Oct. 1999 Pages:235 – 255.

2. Armstrong, A. T; Grubb, P. A; *Software maintenance : concepts and practice.* 2003. Singapore : World Scientific.

3. Kajko-Mattsson,M ; *Preventive maintenance! Do we know what it is?* Software Maintenance, 2000. Proceedings. International Conference on , 11-14 Oct. 2000 Pages:12 – 14.

4. Tjortjis, C.; '*Expert Maintainers' Strategies and Needs when Understanding Software: A Qualitative Empirical Study.* IEEE 8th Asia-Pacific Software Engineering Conference, (APSEC 2001), Proc., Macao, China, 4-7 December 2001, pp281-287, 2001 ISSN/ISBN:0-7695-1408-1.

5. Dunham, M, M; *Data Mining introduction and advanced topics.* 2003. Prentice Hall.

6. Fayyad, U; Piatetsky-shapiro, G.; Uthurusamy R; *Advances in knowledge discovery and data mining.* 1996 London ; Menlo Park, Calif. : AAAI Press/MIT Press.

7. C.M. De Oca and D.L Carver, "Identification of Data Cohesive Subsystems Using Data Mining Techniques", *Proc, Int'l Conf. Software Maintenance* (ICSM 98), IEEE Comp. Soc. Press, 1998, pp.16-23.

8. Mitra, S.; Pal, S.K.; Mitra, P; *Data mining in soft computing framework: a survey*.Neural Networks, IEEE Transactions on Volume: 13 , Issue: 1 , Jan. 2002 Pages:3–14.

9. Han, J.; Kamber, M; *Data Mining: Concepts and Techniques.* 2001. London: Academic Press.

10. Evveritt, B.S. (1980) cluster Analysis. (2nd ed.) London: Heinemann Educational Books Ltd.

11. du Toit S.H.C., Steyn, A.G.W., Stumpf, R.H. (1986) Graphical Exploratory Data Analysis. New York: Springer-Verlag New York Inc. Scanalytics, Inc. (n.d.), Gene Profiler. "Dendrogram Analysis." [online] Fairfax, Virginia. Available from: http://www.scanalytics.com/products/genePro/apps.shtml.

12. Wiggerts, T.A; *Using clustering algorithms in legacy systems remodularization.* Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on 6-8 Oct. 1997 Page(s):33 – 43.

13. Mitra, S; Acharya, T; *Data mining : multimedia, soft computing, and bioinformatics.* 2003 New York ; Chichester : Wiley.

14. Tjortjis, C; Samee, U.; Keane, J.; Layzell, P; *Clustering source code as a means to program comprehension.* School of Informatics, University of Manchester.

15. Jain, A. K.; Dubes, R.C; *Algorithms for clustering data.* 1998. Englewood Cliffs: Prentice-Hall.

16. Jain, A. K.; Murty, M. N.; Flynn P. J; *Data clustering: a review.* September 1999. ACM Computing Surveys (CSUR), Volume 31 Issue 3.

17. Tzerpos, V.; Holt, R.C; *ACDC: an algorithm for comprehension-driven clustering.*Reverse Engineering, 2000. Proceedings. Seventh Working Conference on 23-25 Nov. 2000 Page(s):258 – 267.

18. Tzerpos, V.; Holt, R. C; *Mojo: a distance metric for software clustering.* Reverse gineering, 1999. Proceedings. Sixth Working Conference on 6-8 Oct. 1999 age(s):187 – 193.

19. Mitchell, B.S.; Mancoridis, S; *Comparing the decompositions produced by software clustering algorithms using Similarity measurements.* Software Maintenance, 2001. Proceedings. IEEE International Conference on 7-9 Nov. 2001 Page(s):744 – 753.

20. Rousidis, R.; Tjortjis, C.; *A java code analyzer for data mining.* School of Informatics, University of Manchester.

21. Kanellopoulos, Y; Tjortjis, C.; *Data mining source code to facilitate program comprehension: experiments on clustering data retrieved from c++ programs.* IEEE 12th International Workshop on Program Comprehension (IWPC 2004), pp 214-223, 2004, ISSN/ISBN: 0-7695-2149-5.

22. Mancoridis, S.; Mitchell, B.S.; Rorres, C.; Chen, Y.; Gansner, E.R; *Using automatic clustering to produce high-level system organizations of source code.* Program Comprehension, 1998. IWPC '98. Proceedings., 6th International Workshop on 24-26 June 1998 Page(s):45 – 52.

23. Sartipi, K.; Kontogiannis, K.; Mavaddat, F; *Architectural design recovery using data mining techniques.* Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European, 29 Feb.-3 March 2000 Pages:129 – 139.

24. Chenchen, X.; Tzerpos, Y; *Software Clustering Based on Dynamic Dependencies.* Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on 21-23 March 2005 Page(s):124 – 133.

25. Schwanke, R. W; *An intelligent tool for re-engineering software modularity.* Software Engineering, 1991. Proceedings., 13th International Conference on 13-16 May 1991 Page(s):83 – 92.

26. Malan, R; Bredemeyer, D; *Functional Requirements and Use Cases*. June 1999. Resources For Software Architects. *www.bredemeyer.com*

27. Tzerpos, V.; Holt, R.C; *On the stability of software clustering algorithms.* Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on 10-11 June 2000 Page(s):211 – 218.

Bibliography

1. Collier,K.; Carey,B.; Sautter,D.; Marjaniemi,C.; *A methodology for evaluating and selecting data mining software.* System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on, Volume: Track6, 5-8 Jan. 1999 Pages: 11 pp.

2. Fayyad, U.; Shapiro, G, P.; Smyth**,** P; *The KDD process for extracting useful knowledge from volumes of data.* November 1996. Communications of the ACM, Volume 39 Issue 11.

3. Ganti, V.; Gehrke, J.; Ramakrishnan, R; CACTUS—clustering categorical data using summaries. August 1999. Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining.

4. Gibson, A.; Kleinberg, J.; Raghavan, P; Clustering categorical data: an approach based on dynamical systems. February 2000. The VLDB Journal — The International Journal on Very Large Data Bases, Volume 8 Issue 3-4.

5. Guha, S., Rastogi, R., Shim, K.: *ROCK: A robust clustering algorithm for categorical attributes.* Information Systems 25 (2000) 345—366.

6. Agrawal, R.; Imieliński, T.; Swami, A; *Mining association rules between sets of items in large databases.* June 1993 ACM SIGMOD Record , Proceedings of the 1993 ACM SIGMOD international conference on Management of data, Volume 22 Issue 2.

7. Tjortjis, C; Gold, N; Layzell, P; *Bennett, K.; From system comprehensioni to program comprehensioin.* Proc. IEEE 26th Int. Computer Software Applications Conf. COMPSAC 02), pp 427-432 ISSN/ISBN: ISBN 0-7695-1727-7.

8. Chen, K; Tjortjis,C; Layzell, P; *A method for legacy systems maintenance by mining data extracted from Source code.* Case studies of IEEE 6th European Conf. Software Maintenance and Reengineering, (CSMR 2002), pp 54-60.

9. Tjortjis,C; Sinos, L; Layzell, P; *Facilitating program comprehension by mining association rules from source Code.* Proc. IEEE 11th Int. Workshop Program Comprehension (IWPC 03), pp 125-132 ISSN/ISBN: ISBN 0-7695-1883-4.

10. Davey, J.; Burd, E; Clustering and concept analysis for software evolution. September 2001. Proceedings of the 4th International Workshop on Principles of Software Evolution.

11. King, M.A.; Elder, J.F., IV; Gomolka, B.; Schmidt, E.; Summers, M.; Toop, K.; *Evaluation of fourteen desktop data mining tools.* Systems, Man, and Cybernetics, 1998 IEEE International Conference on, Volume: 3 , 11-14 Oct. 1998 Pages:2927 – 2932 vol.3.

12. Vehvilainen, R; What *is preventive software maintenance?* Software Maintenance, 2000. Proceedings. International Conference on , 11- 14 Oct. 2000 Pages:18 – 19.

13. Tzerpos, V.; Holt, R.C; *Software botryology. Automatic clustering of software systems.* Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on 26-28 Aug. 1998 Page(s):811 – 818.

14. C.M. De Oca and D.L Carver, “Identification of Data Cohesive Subsystems Using Data Mining Techniques”, *Proc. Int’l Conf. Software Maintenance and Reengineering* (CSMR 2002), IEEE Comp. Soc. Press, 2002, pp. 54-60.

Appendix I

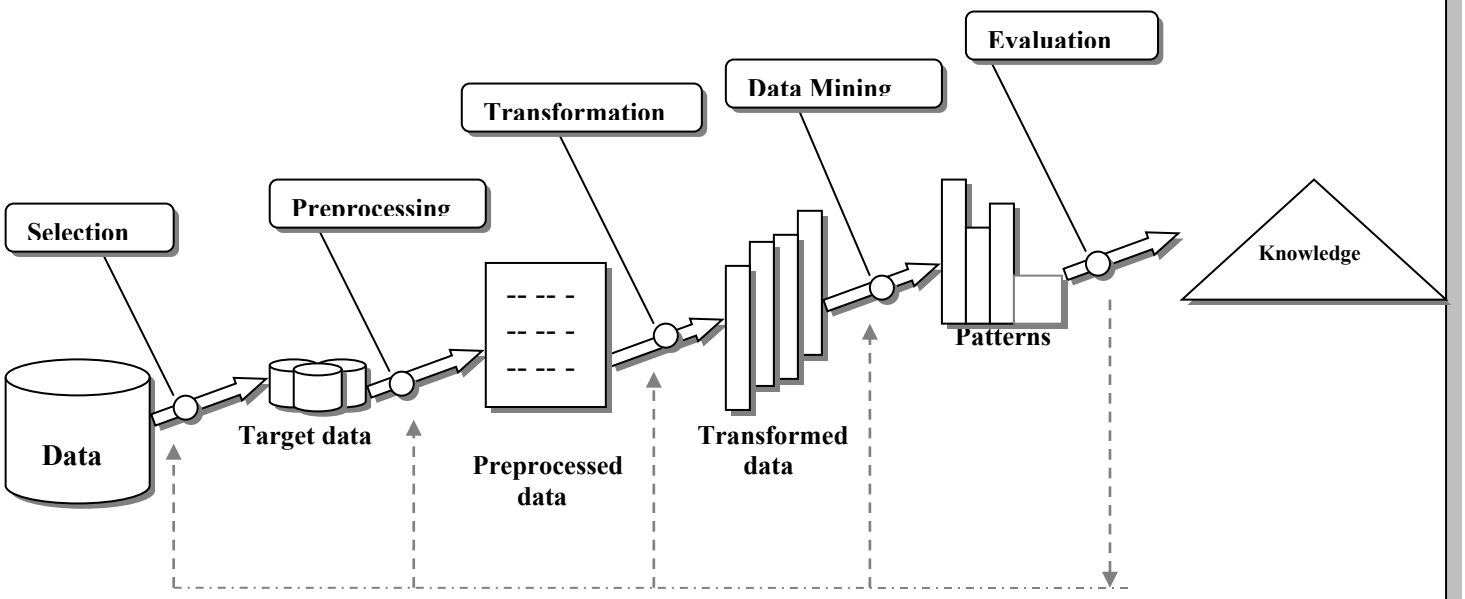


Figure AI: An overview of the steps comprising the KDD process.

General Algorithm for Hierarchical Clustering

- **Step 1.** Compute the proximity matrix containing the distance between each pair of patterns. Treat each pattern as a cluster.

- **Step 2.** Find the most similar pair of clusters using the proximity matrix. Merge these two clusters into one cluster. Update the proximity matrix to reflect these merge operations.

- **Step 3.** Stop the clustering process if all patterns are in one cluster or some termination criteria is met. Otherwise, go to step 2.

Appendix II

This appendix shows all the equations used in this project for calculating similarities between functions based on ***global variables, local variables, parameter types*** and ***return types*.** All these equation are similar to **[14]** with few changes made to better suite the equations for testing purposes.

Similarity Based on Global Variables (S_{GV}):

The equation for finding the Similarity between two functions based on the use of global variables is.

$$S_{GV} = W_{UGV}.S_{UGV} + W_{PGV}.S_{PGV} / W_{UGV} + W_{PGV}$$

Where

W_{XXX} stands for the weight factor that is applied to show the relevant importance of the metrics specified by S_{XXX}.

S_{UGV} = Similarity based on user-defined global variables.

S_{PGV} = Similarity based on pre-defined variables.

User Defined Global Variable (S_{UGV}):

The similarity based on user-defined global variables can be calculated by the following equation.

$$S_{UGV} = \frac{S_{Basic} + S_{Complexity} + S_{usage}}{3}$$

$$S_{Basic} = a_{ugv} / a_{ugv} + b_{ugv} + c_{ugv}.$$

Where

a_{ugv} = Number of (user-defined) global variables used by both the functions being compared.

b_{ugv} , c_{ugv} = Number of uncommon (user-defined) global variables used by the two functions.

$$S_{Complxity} = \frac{\sum_{1}^{a_{ugv}} \text{No. of data itmes in a GV / max. No. of data items in any GV.}}{a_{ugv}}$$

Thus a global variable will contribute more to this metric if it is based on more specialized type, that is one with more data members, whereas an elementary global variable will have a smaller contribution.

$$S_{usage} = \frac{\sum_{1}^{a_{ugv}} \text{2 / No. of functions using the global variable.}}{a_{ugv}}$$

Thus if a global variable is used predominantly by the two functions being compared, then it will contribute more to the similarity metric as compared to the global variable that is used by many functions.

Pre-Defined Global Variable (S_{PGV}):

$$S_{PGV} = \frac{S_{Basic} + S_{usage} + C_{Complexity}}{3}$$

$$S_{Basic} = a_{pgv} / a_{pgv} + b_{pgv} + c_{pgv}.$$

Where

a_{pgv} = Number of (pre-defined) global variables common in the two functions.

b_{ugv} , c_{ugv} = Number of (pre-defined) global variables not common in the two functions.

$$S_{usage} = \frac{\sum_{1}^{a_{pgv}} \text{2/No. of functions using the global variable.}}{a_{pgv}}$$

$$S_{Complxity} = \frac{\sum_{1}^{a_{pgv}} \text{No. of data itmes in a GV / max. No. of data items in any GV.}}{a_{pgv}}$$

Similarity based on local variable types (S_{LV}):

The equation for finding the Similarity between two functions based on the use of local variables is.

$S_{LV} = W_{ULV}.S_{ULV} + W_{PLV}.S_{PLV} / W_{ULV} + W_{PLV}.$

Where:

W_{XXX} stands for the weight factor that is applied to show the relevant importance of the metrics specified by S_{XXX}.

S_{ULV} = Similarity based on user defined local variable type.

S_{PLV} = Similarity based on predefined local variable type.

User Defined local Variable types (S_{ULV}):

$$S_{ULV} = \frac{S_{Basic} + S_{complexity} + S_{usage}}{3}$$

Where:

$S_{Basic} = a_{ulv} / a_{ulv} + b_{ulv} + c_{ulv}.$

$$S_{Complxity} = \frac{\sum_{1}^{a_{ulv}} \text{No. of dataitems in a LV / max. No. of dataitems in a LV.}}{a_{ulv}}$$

$$S_{usage} = \frac{\sum_{1}^{a_{ulv}} 2 / \text{No. of functions using the local variable type.}}{a_{ulv}}$$

Pre-Defined local Variable type (S_{PLV}):

$$S_{PLV} = \frac{S_{Basic} + S_{usage} + S_{complexity}}{3}$$

Similarity based on formal parameter type (S_{PU}):

$S_{PT} = W_{UPT}.S_{UPT} + W_{PPT}.S_{PPT} / W_{UPT} + W_{PPT}.$

S_{UPT} = Similarity based on user-defined formal parameter types.

S_{PPT} = Similarity based on pre-defined formal parameter types.

User Defined formal parameter type (S_{UPT}):

$$S_{UPT} = \frac{S_{Basic} + S_{complexity} + S_{usage}}{3}$$

$S_{Basic} = a_{upt} / a_{upt} + b_{upt} + c_{upt.}$

$$S_{Complxity} = \frac{\sum_{1}^{a_{upt}} \text{No. of entities in a PT / max. No. of entities in a PT.}}{a_{upt}}$$

$$S_{usage} = \frac{\sum_{1}^{a_{upt}} 2 \text{ / No. of functions using the parameter type.}}{a_{upt}}$$

Pre-Defined Parameter type (S_{PPT}):

$$S_{PPT} = \frac{S_{Basic} + S_{usage} + S_{complexity}}{3}$$

Similarity based on return type (S_{RT}):

$$S_{RT} = \frac{S_{Basic} + S_{complexity} + S_{usage}}{3}$$

Where S_{Basic} = 1 if the two parameters are same type otherwise its zero,

$$S_{Complxity} = \frac{\text{No. of data items in a RT}}{\text{Max. No. of data item in a return type}}$$

$$S_{usage} = \frac{2}{\text{No. of functions using the parameter types.}}$$

Appendix III

The contents of this appendix are related to the User interface part of the project.

User interface Options for class clustering are:

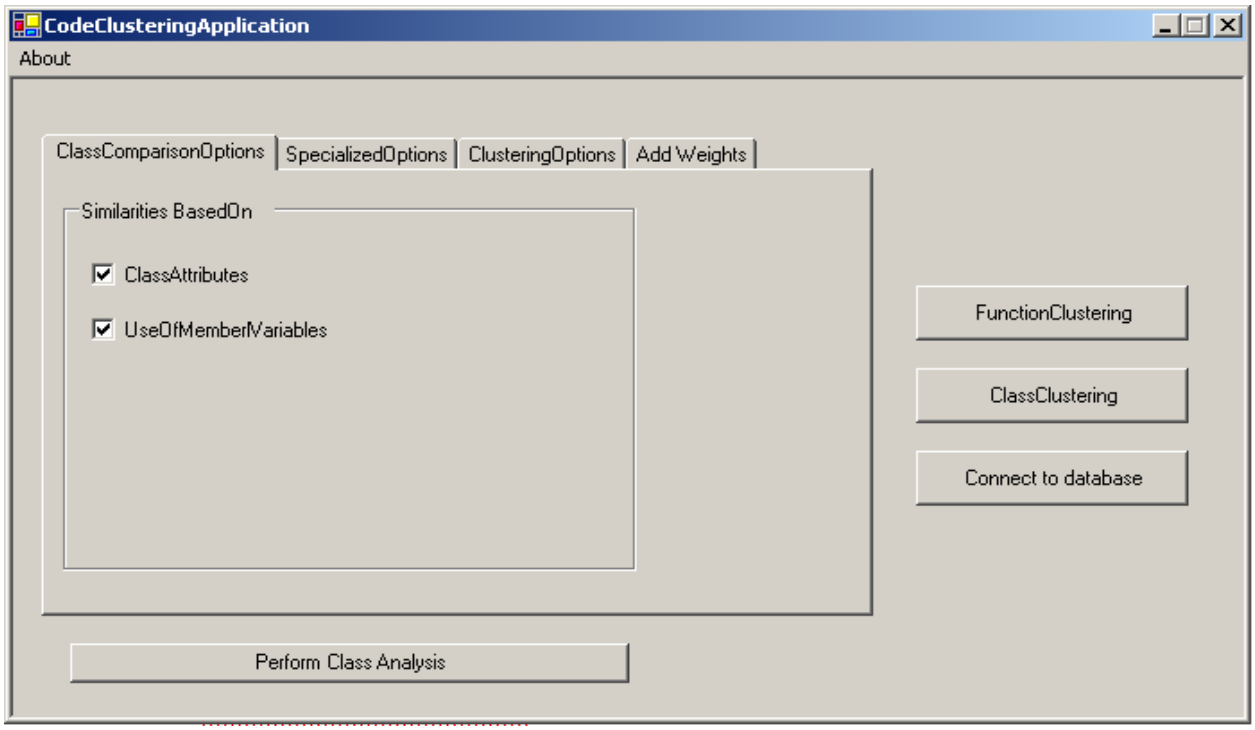


Figure AIII.1 Select class attributes to perform clustering

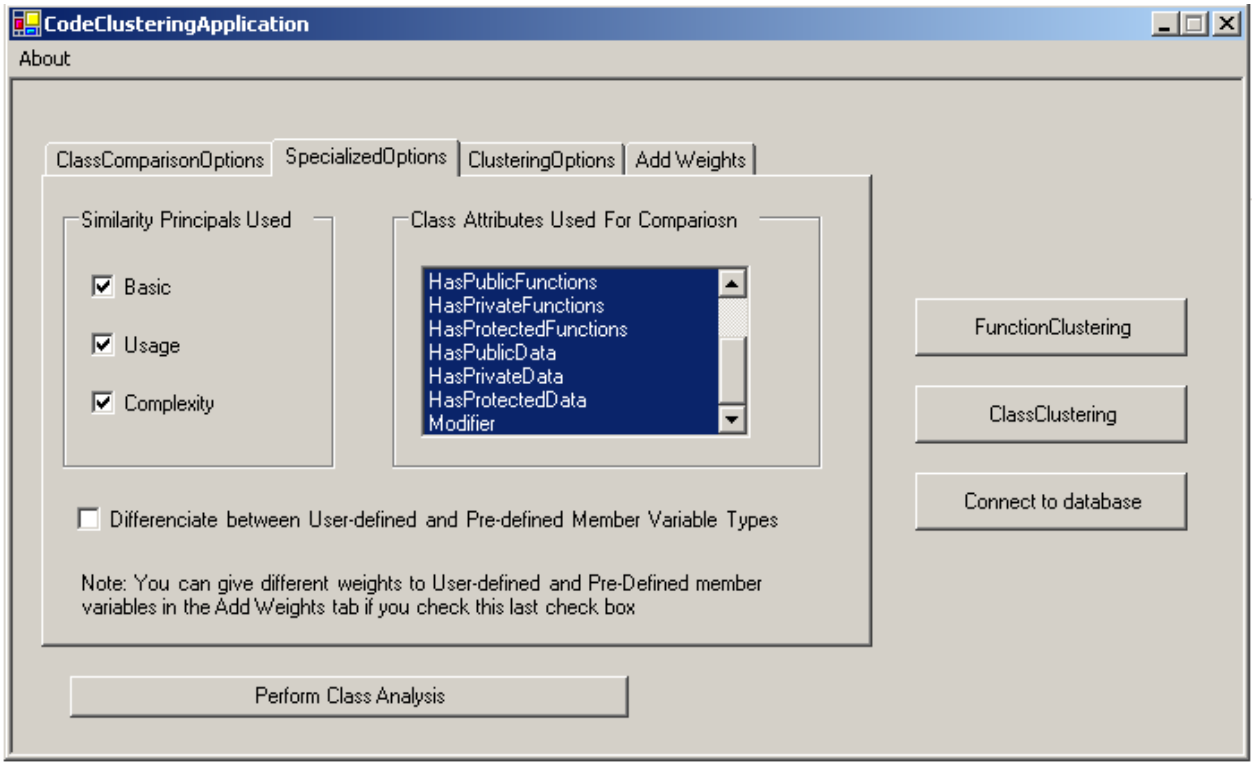


Figure AIII.2 Specialized Options for class clustering

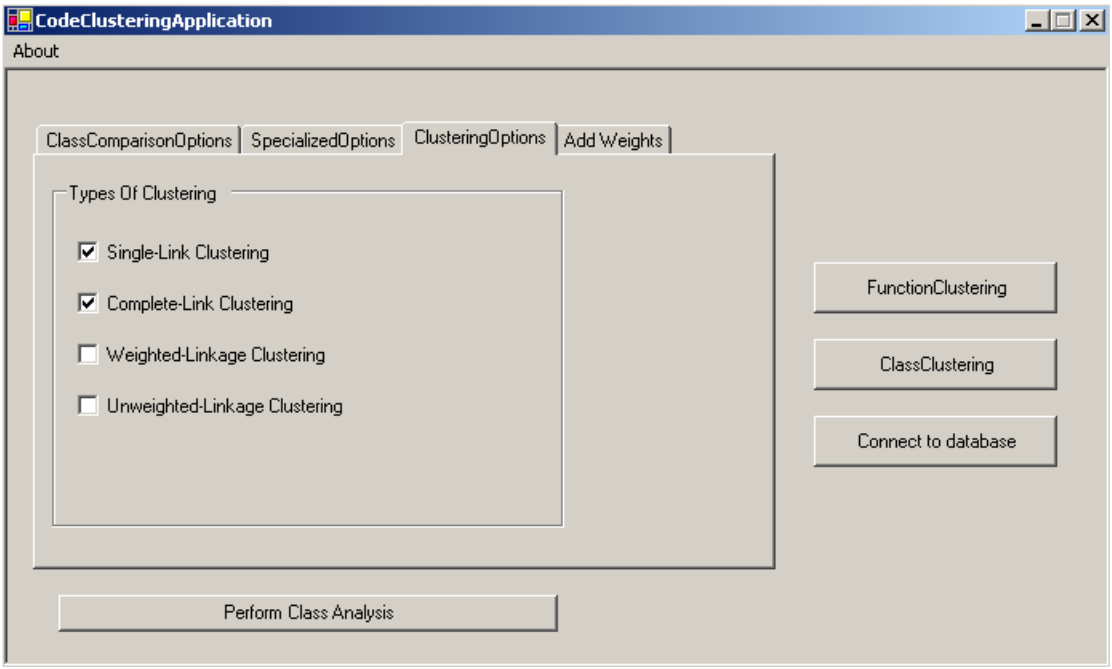


Figure AIII.3 Class Clustering Options

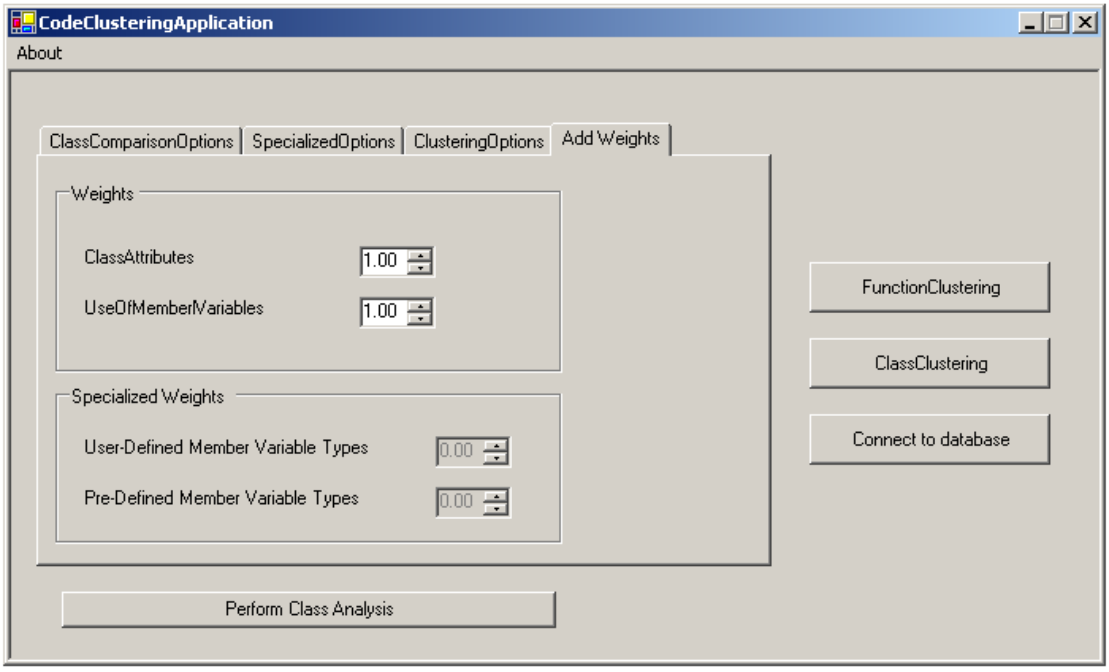


Figure AIII.4 Class Weight Options