An Improved Methodology on Information Distillation by Mining Program Source Code

Y. Kanellopoulos[1, 2], C. Makris[2] and C. Tjortjis[1]

[1] School of Informatics, University of Manchester

[2] Computer Engineering & Informatics Department, University of Patras

Contact author:

Christos Tjortjis,

School of Informatics, University of Manchester,

P.O. Box 88, Manchester, M60 1QD, UK.

Tel: +44 161 3063304, Fax: +44 161 3061281

Email: christos.tjortjis@manchester.ac.uk

Abstract

This paper presents a methodology for knowledge acquisition from source code. We use data mining to support semi-automated software maintenance and comprehension and provide practical insights into systems specifics, assuming one has limited prior familiarity with these systems.

We propose a methodology and an associated model for extracting information from object oriented code by applying clustering and association rules mining. *K-means* clustering produces system overviews and deductions, which support further employment of an improved version of *MMS Apriori* that identifies hidden relationships between classes, methods and member data. The methodology is evaluated on an industrial case study, results are discussed and conclusions are drawn.

1. Introduction

It is well understood and accepted that developing software systems of any size which do not need to be changed is unattainable [23]. Such systems, once in use, need to be functional and flexible in order to operate correctly and fulfil their mission, as new requirements emerge. As a result, software systems remain subject to changes and maintenance throughout their lifetime.

It is crucial to manage such changes, as a lot of effort and time are required in order to keep software systems operational and fit for purpose. Several studies investigating post delivery costs of changes have shown that such costs can be as high as 50-75% of the total system cost throughout the entire system life cycle [21], [26].

The discipline concerned with post delivery changes applied to software systems is known as *software maintenance*. Given the high costs, many organisations often consider their maintenance processes as an area of competitive advantage [16]. Documentation should, in theory, assist maintainers to identify problematic files or modules; alas, in practise, it is often outdated and unreliable [23], [26]. As a result, the best alternative for software maintainers is to comprehend source code, which is both *costly* and *time consuming* [4]. More specifically, 50-90% of the maintenance engineers' time is reported to be spent on program comprehension [26].

The work presented in this paper aims at developing a methodology for semi-automated program comprehension and maintenance incorporating data mining techniques. A fundamental underlying assumption is that the software maintainer may have little or no knowledge of the program which is analysed.

The research objectives of this work include the definition of an input data model, and an associated methodology to populate a database with elements extracted from

source code, the application of a selection of data mining techniques and the evaluation of results in consultation with domain experts.

More specifically, first we define the input model needed to extract data from C# source code; this requires defining entities and their related attributes. Then we propose a methodology which extracts data from source code based on the defined input model. Subsequently we employ *Clustering* in order to provide software maintainers with a quick and rough grasp of a software system so that they can operate with a level of confidence as if they had prior familiarity with the system. After that, we apply *Association Rules mining* in order to identify hidden relationships between classes, member data, and methods and we assess the feasibility of the proposed methodology, in producing valid, useful and novel patterns and knowledge about a software system.

C# was selected as the programming language to be examined as it is a new, widely used language associated with comprehension difficulties when compared to other programming languages [29]. It was also selected because it is reasonably expected that a significant part of future legacy systems will have been written in C#. C#, as an object oriented language, has the advantage that it can be analyzed at either a detailed, technical level in the structural domain, (member data analysis), or at a more abstract level in the behavioral domain (member methods analysis).

The contribution of this research work is twofold: first we propose a novel algorithmic framework which combines two different kinds of data mining algorithms, one for clustering and one for mining association rules from code; this provides maintenance engineers with a more comprehensive view of the system under maintenance, at various levels of abstraction. Secondly, the efficient implementation of this framework required a number of key algorithmic decisions including the following:

- The choice of appropriate data mining algorithms.
- The definition of novel algorithmic metrics.

- The way that the chosen clustering algorithm interfaces and connects to the chosen association rules mining algorithm.

Although both clustering and association rules have been used in isolation in the past in order to support software maintenance, this is the first time combining these techniques is attempted [14], [15], [17], [20]. The results are not only promising but also address a fundamental requirement that maintenance engineers should be able to switch focus between various levels of abstraction and partially understand a system using a combination of the bottom-up, top-down or the middle out approach [26]. Combining the overviews provided by high level clustering and the insights into inter and intra-cluster interrelationships provided by association rules are unique contributions to facilitating software maintenance and comprehension.

The remaining of this paper is organised as follows. Section 2 reviews the existing work in the area of data mining for program comprehension; and describes the data mining algorithms we employed in this research work. Section 3 outlines the proposed methodology for extracting data from C# source code, the input data model and the data mining techniques we used. Section 4 assesses the accuracy of the output of this methodology, analyses its results and outlines deductions from its application. Finally, conclusions and directions for future work are presented in Section 5.

2. Background

Software maintenance is the most difficult and expensive stage in software lifecycle, often performed with limited understanding of the design and the overall structure of a system because of commercial pressures [18]. Fast, unplanned modifications, based on partial understanding of a system, give rise to increased code complexity and deteriorated modularity, thus resulting in 50-90% of the maintainers' time to be spent on program comprehension [26]. Furthermore it is recognised that there are no explicit guidelines given a program understanding task, nor there are good criteria to decide how to represent knowledge derived by and used for it [3].

2.1 Data Mining for Program Comprehension

Data mining and its ability to deal with vast amounts of data, has been considered a suitable solution in assisting software maintenance, often resulting in remarkable results [2], [12], [14], [15], [17], [20], [27], [30]. Data mining can discover non-trivial and previously unknown relationships among records or attributes in large databases [7], [8]. This highlights the capacity of data mining to obtain useful knowledge about the structure of large systems. It has three features that make it useful for program comprehension and related maintenance tasks [17]:

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

- It can be used to expose previously unknown non-trivial patterns and associations between items in databases. Therefore, it can be used to reveal hidden relationships among program components.

- It 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.

Data mining has been previously used for identification of subsystems based on associations (ISA methodology) [17]. This approach provides a system abstraction up to the program level as it produces a decomposition of a system into data cohesive subsystems by detecting associations between programs sharing the same files.

Sartipi et al. used data mining for architectural design recovery [20]. They proposed a model for the evaluation of the architectural design of a system based on associations among system components and used system modularity measurement as an indication of design quality and its decomposition into subsystems. Three association views of a system were generated: a) control passing which represents system components correlation based on function invocation, b) data exchange which represents system components correlation based on aggregate data types and c) data

sharing which represents system components correlation based on functions sharing global variables. This approach models software systems as attributed relational graphs with system entities as nodes and data-control-dependencies as edges. Application of association rules mining decomposes such graphs into domains of entities based on the association property. This approach is based on the concept of the association between the components of a system. There are however other characteristics that can play a role in grouping system components, such as the number of member data or functions in a class. These can be discovered by using other data mining techniques like clustering.

This data mining technique has been used to support software maintenance and software systems knowledge discovery [19]. This work proposes a methodology for grouping Java code elements together, according to their similarity and focuses on achieving a high level system understanding. The methodology derives system structure and interrelationships as well as similarities among system components by applying cluster analysis on data extracted from source code. It consists of two main parts, the input model and the clustering algorithm. The input model takes into account five basic Java code elements: files, packages, classes, methods, and parameters. These elements form the entities to be stored in respective tables. Each entity also has a number of associated members. A Hierarchical Agglomerative Clustering (HAC) algorithm is employed to reveal similarities between classes and other code elements thus facilitating software maintenance and Java program comprehension. The methodology was evaluated on a small sized system (10-20 classes) only. It would be very interesting to see how this methodology scales up to deal with real industrial scale systems.

Understanding low/medium level concepts and relationships among components at the function, paragraph or even line of code level by mining C and COBOL legacy systems source code was addressed in [27], [28]. For C programs, functions were used as entities, and attributes defined according to the use and types of parameters and

variables, and the types of returned values. Then clustering was applied to identify sub-sets of source code that were grouped together according to custom-made similarity metrics [28]. For COBOL programs, paragraphs were used as entities, and binary attributes depending on the presence of user-defined and language-defined identifiers. In this case association rules were derived in order to establish inter-group and intra-group relationships [27]. Both approaches address software systems at medium and low level and confirm that data mining can produce structural views of source code thus facilitating legacy systems understanding. Their shortcoming is failing to capture correlations across system components such as programs and files [27], [28].

An approach for the evaluation of dynamic clustering is presented in [31]. The scope of this solution is to evaluate the usefulness of providing dynamic dependencies as input to software clustering algorithms. This method consists of three phases: The first is the analysis of dynamic dependencies by adding instrumentations when compiling the source code. The second is the analysis of static dependencies by extracting them with the use of *Swagkit* [9], a software architecture toolkit developed in the university of Waterloo. The last step is filtering in order to help weigh the dynamic dependencies graphs. The method was applied to *Mozilla*, a large open source software system with more than four million lines of C/C++ [31]. The conclusion of this work is that there is merit in clustering dynamic dependencies of a software system. That means that better and fuller dynamic dependencies graphs and clustering algorithms should be implemented in order to give better chances to dynamic clustering.

Clustering over a Module Dependency Graph (MDG) [14] uses a collection of algorithms which facilitate the automatic recovery of the modular structure of a software system from its source code. The method creates a hierarchical view of system architecture into subsystems, based on the components and the relationships between components that can be detected in source code. First the system modules and the module-level relationships are presented as a module - dependency graph. Then this graph is partitioned, so that the high - level subsystem structure can be derived from

the component level relationships extracted from the source code. Based on the concepts of cohesion and coherence three parameters are introduced: intra-connectivity, inter-connectivity and modularisation quality. The basic goal of this modularisation technique is to automatically partition the components of a system into clusters (subsystems) so that the resultant organisation concurrently minimises inter-connectivity while maximising intra-connectivity. The underlying assumption is that a well-designed system is organised into cohesive clusters that are loosely interconnected. This approach provides a system abstraction up to the program level. The main drawback of this solution is that as the number of files exceeds 20, calculation time is greatly increased.

2.2 Data Mining Algorithms Selection

As clustering and association rule mining have been shown to be the most promising data mining techniques in the area of software maintenance [15], [20], [27], [30], we decided to use appropriate versions of *K-Means clustering* and *MMS Apriori* for this work. The algorithms are detailed in the following subsections. Details on how these algorithms were parameterised can be found in Section 3.

2.2.1 K-Means Description

K-means clustering is a commonly used partitioning algorithm. Each cluster is represented by the mean value of the objects in the cluster. As a result, cluster similarity is measured based on the distance between the object and the mean value of the input data in a cluster. It is an iterative algorithm in which objects are moved among clusters until a desired set is reached. The steps of the algorithm can be described as follows [6]:

```
Given a set of n objects t₁, t₂,…,tₙ
and a number k of desired clusters,
```

```
    assign initial values for means m₁, m₂,…, mₖ
repeat
    assign each item tᵢ to the cluster with the closest mean;
    calculate new cluster mean;
until means m₁, m₂,…, mₖ do not change
```

The squared-error criterion is used to measure the sum of the squares of the distance between each object and the mean. The sum should be minimized in order to obtain a good clustering result. It is obvious that the smaller the sum, the more tightly the objects are clustered around the mean value (centroid), and clustering is more precise. The squared-error criterion can be expressed by the formula in equation 1:

| $$SSE = \sum_{i=1}^{K} \sum_{p \in C_i} dist(c_i,p)^2$$ | Equation 1 |

where *dist* is the standard Euclidean (L_2) distance between two objects in Euclidean space; *p* is an object belonging to the ith cluster C_i, and c_i is the mean of the cluster.

The algorithm is suitable for discovering spherical-shaped clusters in small to medium size databases. However, its main problems are that it is sensitive to noise and to the initial partitioning. As many possible initial partitions lead to many different results, the final clustering is influenced by the initial partition, which is indicated by the user input [8].

2.2.2 MMS Apriori Description

*Multiple Minimum Support (*MMS) Apriori [11] is based on the combination of two algorithms: MSApriori [13] and DIC [5]. More specifically Liu et al. in [13] presented an algorithm called MSApriori, which was based on algorithm Apriori [1] and could find rules among items with different supports. The support for each different item was computed using the formula in equation 2:

$MIS(i) = \begin{cases} M(i) & M(i) > LS \\ LS & Otherwise \end{cases}$ $M(i) = \beta \cdot f(i)$	Equation 2

where $f(i)$ denotes the actual frequency of an item in the dataset, *LS* is a user defined lowest minimum item support allowed, and β is a parameter that controls how the Minimum Item Support (*MIS*) values should be related to their frequencies. The algorithm follows similar phases to the Apriori algorithm, with the difference that the minimum support for itemsets $I_1, I_2, \ldots, I_m$ is computed to be equal to min[MIS(I_1), MIS(I_2),…MIS(I_m)].

On the other hand, Brin et al. in [5] presented the algorithm DIC, which made use of a single minimum support and reduced the number of passes made over the data in comparison to the classic Apriori algorithm. The basic idea behind DIC is that one does not have to wait until a pass is complete before counting higher order itemsets; for example we can begin counting 2-itemsets even before we have finished counting all 1-itemsets.

MMS Apriori combines effectively the two approaches [11]. Hence, first it identifies all large and locally frequent 1-itemsets and assigns an *MIS value* to them. Then every itemset is marked with a different state in six different possible ways, which are:

- Dashed Circle (DC) – suspected small itemset – an itemset we are still counting and its count is below its MIS value – also the initial state of all itemsets.

- Solid Circle (SC) – confirmed small itemset – an itemset we have finished counting having its count is below its own MIS value, and found not to be locally frequent at any partition.

- Dashed Square (DS) – suspected large itemset – an itemset we are still counting, but its count already exceeds its MIS value.

- Solid Square (SS) – confirmed large itemset – an itemset we have finished counting through all the transactions and that exceeds its MIS value.

- Dashed Triangle (DT) – an itemset found locally frequent that we are still counting to see if its final count is above its MIS value.

- Solid Triangle (ST) – an itemset that we finished counting through all the transactions and that was found locally frequent at some partition, but its final count is below its MIS value.

Every 1-itemset begins to be counted with its state DC, except from the empty itemset, which is marked immediately with its state solid box. During traversals and at suitably defined periodic time intervals the counter of every counted itemset is checked against its MIS value. If its counter is larger or equal to its MIS value then its state is changed into DS. When an itemset has been counted through all the transactions we check again its counter against its MIS value. If its state was DC, and its counter is finally equal or larger than its MIS value we change its state to SS. If its state was DS, we simply change it to SS.

As a general rule, if any immediate superset of a k-itemset, stated DS has all of its subsets as solid or dashed squares, we make its state DC and begin counting it; this holds with the exception of some special cases handled by our algorithms. As we can easily understand only 1-itemsets can be stated DD or SD, since we keep an MIS ordering only for those itemsets. All the other higher order itemsets can be of state DC, DS, SS and SC. Upon termination of the algorithm we check and output all the Solid Square sets. We detail how the MIS values are defined and customised for this work in §3.3.3.

3. Proposed Methodology

A well documented problem faced by maintainers when comprehending a software system is the lack of familiarity with it, combined with the lack of accurate

documentation [18]. Facilitating and accelerating the time consuming activity of program comprehension motivated this research work.

Its main scope is to devise a methodology that can provide practical insights overviews to guide the maintenance engineer through the specifics of systems, assuming little familiarity with these. More specifically this work aims at helping the maintenance engineers to:

- Identify patterns in the source code

- Extract interrelationships between code elements

- Identify niches and potentially problematic classes

## 3.1 Systems Analysis Domain Models

We analyse software systems based on the following two domains:

- The behavioural domain, which concerns methods of the system's classes and their respective parameters [10].

- The structural domain, which concerns member data of classes.

We use classes, methods and member data as entities for clustering; as depicted in Fig. 1. These entities are characterised by *contains* relationships. Both types of relationships can be inverted to, contained by or used by to double the number of relationships [22]. These relationships are:

- Classes contain method definitions.

- Classes contain member data definitions.

- Methods contain parameters.

The behavioural domain includes the analysis of the correlations between the methods of the system's classes while the structural one is concerned with the interrelations among their member data.
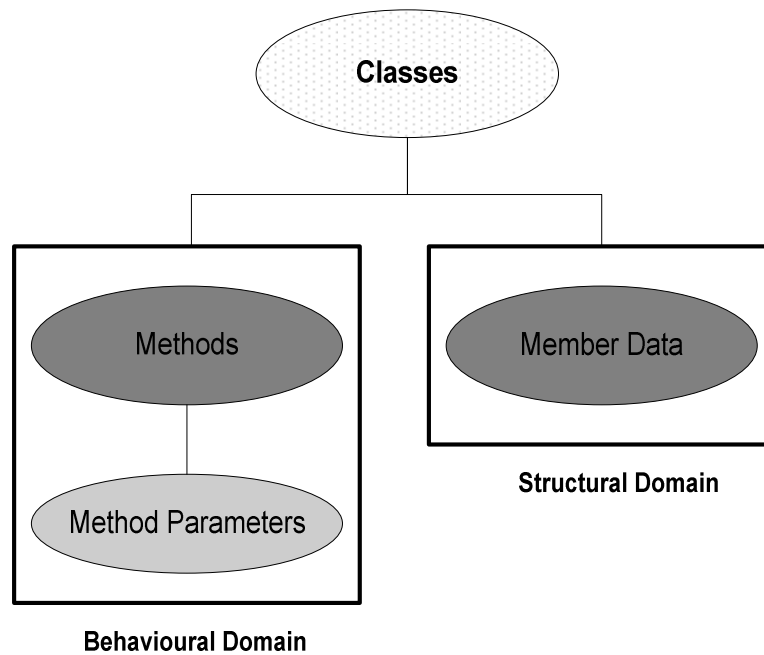
Figure 1-Entity Hierarchy and Analysis Domains

3.2 Data Extraction Process

As soon as the input model is formed, the next essential step is to design and implement a methodology for extracting data from source code. For this purpose a parsing engine which operates on source code at the lexical level was built.

The basic requirements for this parsing engine were:

- To operate without a need to pre-process or compile files; given that the aim of the methodology is to facilitate maintenance engineers, this step should be automated.

- To handle a wide variety of tasks; the parsing engine should identify patterns in the source code and then store these patterns in a database.

- To execute swiftly and handle arbitrary amounts of data; as the size of the applications to be analysed is not known in advance, the parsing engine should be able to handle a large variety of systems types with a wide range of sizes.

Based on these requirements we employed regular expressions in order to implement the parsing engine, as these can efficiently search for patterns in large code files or across many files [24]. Their use can also save time and yield results that might elude manual browsing.

3.3. A Framework for Using Data Mining Techniques

We propose here a framework for using data mining techniques in order to facilitate comprehension of systems under maintenance, as depicted in Fig. 2. First, we employ *K-means clustering* on data extracted from C# source code, as the maintenance engineer initially needs a quick and rough grasp of a software system in order to maintain it with a level of confidence as if he/she had this familiarity. Clustering is more suitable for this purpose because it produces overviews of systems by creating mutually exclusive groups of classes, member data or methods, according to their similarities, thus reducing the time required to understand the overall system. Clustering has also the potential to discover programming patterns and "unusual" or outlier cases which may require further attention.
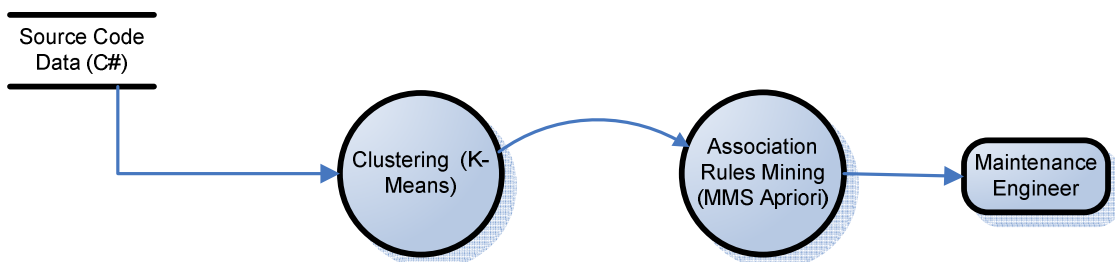
Figure 2-Data Mining Techniques for Program Comprehension

As soon as the maintenance engineer forms an overview of the software system and gains the required familiarity, identification of hidden relationships between classes, member data and methods is needed; this is achieved by using *Association Rules* mining. Based on the previous step of clustering, the maintainer can set the minimum support (*minsup*) and minimum confidence (*minconf*) thresholds in order to

gain the most accurate results from mining association rules. This can also assist

impact analysis and measuring cohesion and coupling of system modules, such as

files, classes, variables and methods [6]. Results from this analysis can then be

interpreted by the maintenance engineer.

In order to further explain the proposed framework we provide a small example that

illustrates how the proposed algorithms are used. This example is based on the

analysis of the Member Methods Parameters entity. Table 1 presents data as extracted

from C# source code, and are used as input for K-Means clustering.

Table 1
C# Source Code Data – Input for Clustering

Parameter ID	Method ID	Name	Type	Call Type
1	1	Sender	Object	ByValue
2	1	E	System_EventArgs	ByValue
3	2	Width	Int	ByValue
4	2	Height	Int	ByValue
5	3	callback	System_AsyncCallback	ByValue
6	3	asyncState	Object	ByValue
7	4	Ps	PAINTSTRUCT	ByReference
8	4	hWnd	IntPtr	ByReference
9	5	Pt	POINT	ByReference
8	5	hWnd	IntPtr	ByReference

Table 2:
Output data from clustering an input data for association rules mining

Parameter ID	Method ID	Parameter Name	MIS	Cluster	Record Score	Relevance
1	1	Sender	0.439120	2	0.951942	1.32534
2	1	e	0.464559	3	0.933708	1.21887
3	2	Width	0.811380	5	0.366234	0.866234
4	3	Height	0.811380	5	0.366234	0.866234
6	4	asyncState	0.802737	2	0.495631	0.750107
5	4	Callback	1.010638	8	0.244737	0.744737
7	5	Ps	1.924554	8	0.019354	0.500247
8	5	HWnd	1.372371	8	0.125161	0.603505
8	6	HWnd	1.372371	8	0.125161	0.603505
9	6	Pt	1.924554	8	0.019354	0.500247

Table 3:
Derived Association Rules

Condition	Consequence	Confidence	Occurrence
Sender	E	0.989	930
Callback	asyncState	1.000	93
Width	Height	1.000	6
Pt	hWnd	1.000	2
Ps	hWnd	1.000	2

As soon as clustering is completed, the following type of information is created as its output, which is in turn used as input for mining association rules:

- Parameter ID and Method Id are used as input data

- The Minimum Item Support (MIS) is used as an input parameter, as it is the support for each data-item. We further describe how the MIS values are defined in §3.3.3.

Table 2 presents the data produced as output from clustering, which in turn are used as input for association rules mining. Table 3 presents the derived association rules concerning the associations between the data-items of the Member Method Parameters entity.

3.3.1 K-Means Clustering Parameters

As described in §2.2.1, *K-Means clustering* algorithm was employed in this research work. Some of its key features include the need for the user to define the number of derived clusters and its sensitivity to noise.

To determine the number of clusters, we used a series of experiments and feedback from maintenance engineers. As detailed in the description of the case study of the Books Publication System presented in §4.3, the maintainers were asking for a number of clusters that would give them a system overview whilst providing opportunities to discover niches. After several trials we concluded that an appropriate number of derived clusters would be nine (9).

As far as noise sensitivity is concerned, this could have been addressed by eliminating outlier data either in advance or in a post-processing step. However, in this research work, we wanted to discover niches thus we did not remove outlier data. The input parameters we used for K-Means clustering are presented in Table 4, while the output parameters are shown in Table 5.

Table 4:

Input parameters for K-means clustering

Name	Description
Input Dataset [D]	The given dataset
Maximum Passes [P]	The maximum number of passes the algorithm goes through the source code data to perform clustering.
Maximum Number of Clusters [C]	The maximum number of clusters the algorithm generates. Limiting this number we avoid producing many small clusters and we save run time, while increasing it, improves the likelihood of finding niches.
Similarity threshold [S]	It limits the values accepted as best fit for the cluster.
Accuracy Improvement [A]	The minimum percentage of improvement on clustering quality after each pass through the data.

Table 5:

Output parameters for K-means clustering

Name	Description
Cluster ID [ID]	The identifier of the best fitting cluster for the corresponding input record.
Record Score Field Name, [RS]	Its values are the fitting quality of the corresponding input record to the best fitting cluster.
Relevance Field Name [REL]	Expresses the relevance of a record's assignment to a cluster.

3.3.2 MMS Apriori Association Rules Parameters

Association rules mining can be performed either inside each cluster (intra-cluster mining) or for all the clusters of each entity (inter-cluster mining). For this reason we employ the MMS *Apriori* algorithm [11], an improved version of the *Apriori* [13]. A database in which an association rule can be found is viewed as a set of records, where each of them contains a set of items. Market basket analysis is the most frequent application of association rules, where each item represents an item purchased, while each record is the list of items purchased at one time. In our case though, for analysing C# source code, we employ a different data model more suitable to our needs. More specifically, instead of transactions we use classes and member methods as "baskets", and member data, parameters and methods as "items" for the respective baskets; this is depicted in Table 6. For each type of "basket" and "item" we use a corresponding unique ID.

Table 6:
Code analysis analogy to market basket analysis

Basket	Itemset
Class	Member Data
Class	Member Methods
Member Method	Parameters

Table 7:
Parameters used for the *MMS Apriori*

Name	Description
Basket [B]	The given dataset (basket)
Output File [O]	The file containing the frequent Itemsets and the derived association rules
Minimum Item Support [MIS] File	The file containing the MIS applied for each item of the given Basket (B).
Minimum Confidence [MinConf]	The confidence threshold (value between [0…1]).

3.3.3 Defining MIS for the MMS Apriori

As mentioned above, before running the *MMS Apriori* mining algorithm, the

Minimum Item Support (*MIS*) has to be defined. The output parameters, `Record Score`

`Field Name (RS)` and `Relevance Field Name (REL)`, of the previous step (K-Means

Clustering) of our methodology will be used for this purpose. The value of the `RS`

parameter is the fitting quality of the corresponding input record to the best fitting

cluster, while the value of `REL` represents the relevance of the record's assignment to

this cluster.

Based on those two parameters, and whether we perform intra or inter cluster

association rules the following formulas in equations 3 and 4 are employed

respectively, in order to assign values to the `MIS` parameter:

*MIS(i) = f(i) * (1/(RS(i)+REL(i)))*	Equation 3

where `f(i)` is the percentage of the actual frequency of an item in the cluster, and

`RS(i)`, `REL(i)` are the item's Record Score and Relevance respectively. This formula is

used for intra-cluster association rules mining, because we want to examine the

relationships concerning the items inside each cluster.

*MIS(i) = f(i) * (1/(Avg(RS(i))+Avg(REL(i))))*	Equation 4

where $f(i)$ is the percentage of the actual frequency of an item in the dataset, and

$Avg(RS(i)), Avg(REL(i))$ are the average item's Record Score and Relevance

respectively. This formula is used for inter-cluster Association Rules mining as we are

interested in discovering relationships between the clusters of the system.
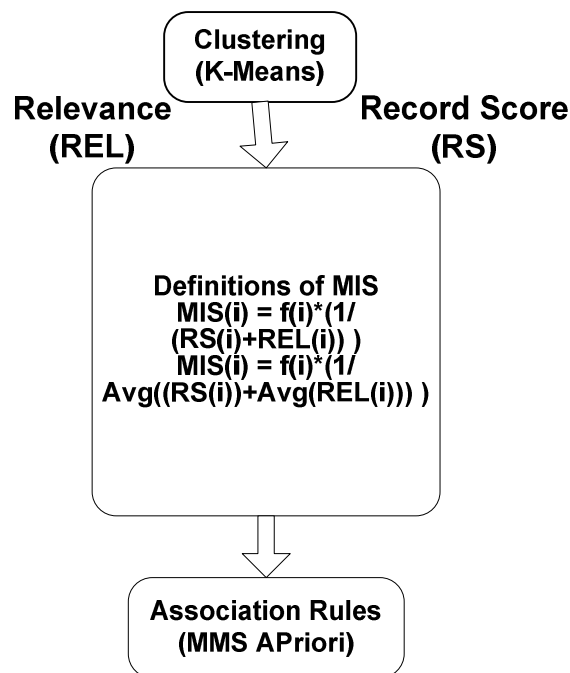
Figure 3 Process definition for MIS

The intuition behind the proposed formulae is that in order to favor items that are

significant we should lower their minimum support, and hence increase the chances

that the respective singleton itemsets, or itemsets containing important items, are

selected during the itemset generation phase. As the factors $RS(i)$ (record score) and

$REL(i)$ (record relevance) reflect the significance of item i, the respective minimum

support, used in the intra-cluster association rules mining, should be inversely proportional to them. On the other hand, when estimating the minimum support in inter-cluster association rules mining it is a natural choice to select the average significant factor of all the items in the same cluster as representing the significance of each item in the cluster. Fig. 3 depicts how the `MIS` parameter for each item in the given dataset is defined.

4. Result Evaluation

The proposed framework was evaluated in terms of accuracy and ability to capture knowledge relevant to software maintenance activities, using one industrial application. The actual structure of this application was compared, with the help of domain experts, to the outcome of the analysis of its respective input model. The output should be valid, novel and useful to the maintenance engineers. The following sub-sections discuss separately the outcomes of our empirical experimentation with this application.

4.1 Experimental Setup

Before describing the experimental application of our methodology to a real industrial software system it is necessary to describe how this experiment was set up. As presented in §3.3 clustering and association rules mining are employed in order to distil information from a software artefact's entities. The core idea is to perform clustering for each entity first. Based on clustering results we can:

- Define the `Minimum Item Support (MIS)` and `Minimum Confidence (MinConf)` parameters, for either intra-cluster either inter-cluster rule mining.

- Mine association rules from all the produced clusters of an entity (inter-cluster mining). This can help us discover the most important rules concerning the system overall.

- Mine association rules from separately for each produced cluster of an entity (intra-cluster mining). This can focus on specific rules concerning each cluster separately.

The analysis of entities was top – down. It started from classes, and then it analysed entities at the behavioural (member methods) and structural (member data) domains.

4.2 Case Study: Books Publication System

The Books Publication System was developed in order to cover the needs of a Greek Books Publishing Organisation. It is a Windows application developed with C# and is designed based on the 3-tier architecture. It consists of:

- 693 source code files

- 1242 classes

- 6919 member data

- 5976 member methods

- 5150 method parameters

4.2.1 Class Analysis

The Class entity is on the top of the described model at §3.1. As such, we can only apply clustering to it. Mining association rules at this top level would be meaningless.

Table 8:
Parameters used for the clustering Classes

Name	Description
Input Dataset [D]	A flat file containing records describing the Classes Entity.
Maximum Passes [P]	The chosen value is 5, as we wanted to increase the accuracy of clustering. Specifying more passes through the data improves the quality of the generated clusters.
Maximum Number of Clusters [C]	The chosen value is 9, as the maintenance engineers were interested in grouping the Classes Entity in a number of clusters that will give an overview of the system while on the same time providing the opportunity of finding niches.
Similarity threshold [S]	The chosen value is 0.6, as we wanted only records with 60% identical fields to be assigned on the same cluster
Accuracy Improvement [A]	The chosen value is 2, as we wanted clustering to be as accurate as possible and to limit the number of the processing time

4.2.1.1 Cluster Analysis

We applied K-Means to the Class entity using the parameters described in Table 8. The derived clusters were formulated based on the `BaseClass1` field, which describes the parent class from which each class inherits. The main base classes that are of importance in formulating clusters of the system are presented in Table 9.

Table 9:
The main base classes

S/N	Class Name
1	SQLHelper
2	DataComponent
3	ClientForm
4	ProppertyPage
5	ArrayList
6	CollectionWithEvents
7	FTPTestCase
8	ComponentNode
9	System.Windows.Forms.Form

4.2.1.2 Class Analysis Results

After analysing the Class entity we examined the derived results. First we verified that good design practices were followed in system models as all the base classes presented in the previous paragraph are abstract.

Table 10:
"God" classes

S/N	Class Name	Number of Methods
1	Reporting Service	269
2	HostPanel	127
3	DragProvider	93
4	CostCompForBooksWithoutYA	75

Moreover we identified potential problematic areas by discovering the existence of "god classes". That means there are classes that their number of methods is much above the general mean which is 5 methods per class. These classes in practice may be difficult to maintain and reuse. More specifically, we discovered a cluster that its elements (classes) have more than 70 methods. Those classes are shown in Table 10.

4.2.2 Member Methods Analysis

The Member Methods entity belongs to the behavioural domain as described at

§3.1. As such we can apply both clustering and association rules mining.

4.2.2.1 Cluster Analysis

Table 11 shows the values assigned to the input parameters in order to perform

clustering.

Table 11:
Parameters used for the clustering member methods

Name	Description
Input Dataset [D]	A flat file containing records describing the Member Methods Entity.
Maximum Passes [P]	The chosen value is 3, as we wanted to increase the accuracy of clustering. Specifying more passes through the data improves the quality of the generated clusters.
Maximum Number of Clusters [C]	The chosen value is 9, as the maintenance engineers were interested in grouping the Classes Entity in a number of clusters that will give an overview of the system while on the same time providing the opportunity of finding niches.
Similarity threshold [S]	The chosen value is 0.8, as we wanted only records with 80% identical fields to be assigned on the same cluster
Accuracy Improvement [A]	The chosen value is 4, as we wanted clustering to be as accurate as possible and to limit the number of the processing time.

The fields `Category, Modifier` and `ReturnType,` are the most important for forming

the clusters of this entity. The main characteristics of the derived clusters are presented

in Table 12

As derived from cluster analysis, almost half of the member methods return void.

Most of them also are Windows events that either draw objects or handle parameters

like the ones of clusters 7 and 8. Clusters 0, 6 and 5 reflect the niches of the Books

Publishing System. They are considered niches as:

- Cluster 0 is the only cluster that consists of static member methods which

  return `System.IAsyncResult` types.

- Cluster 5 consists of member methods that are responsible for the format and rounding of numbers. It is the only cluster of private methods and their Return Type is `ArrayList`.

- Cluster 6 is the only cluster with public external methods which have bool return type.

Table 12:
Member methods clusters' main characteristics

Name (Number)	Size %	Characteristics
7	26.42	The methods belonging to this cluster are public, override and return nothing (void). The predominant names are AssignParameters, GetParameters and Dispose.
8	20.00	The methods belonging to this cluster are private, extern and return nothing (void). The predominant name is InitializeComponent.
2	13.91	The methods belonging to this cluster are public, override or static and return nothing (void). The predominant names are OnSelected, Unload, MenuCommand and Refresh.
4	13.10	The methods belonging to this cluster are public, virtual or delegate and return nothing (void). The predominant names are Remove, Insert and AddRange.
1	10.66	The methods belonging to this cluster are protected, override and return bool or object. The predominant names are Execute, OnCancel and OnOK.
3	10.04	The methods belonging to this cluster are public, override and return string. The predominant names are Add, GetInfo and Remove.
6	2.89	The methods belonging to this cluster are public, virtual or extern and return Boolean. The predominant names are Contains, and OnClose.
0	2.61	The methods belonging to this cluster are public, static and return int or System.IAsyncResult. The predominant names are IndexOf, LoadBitmap and LoadBitmapStrip.
5	0.37	The methods belonging to this cluster are private, have no modifier and return an ArrayList, double and string. The predominant names are MoneyFormat, roundingTypographics and roundingTypographicsforMont.

4.2.2.2 Inter-Cluster Association Rules Mining

In order to find the associations between different clusters we performed the association rules mining for all the clusters. The result was to generate rules and to discover frequent itemsets concerning all the clusters in total. Table 13 shows the values assigned to the input parameters in order to perform the rules mining task.

Table 13:

Parameters used for Methods Inter – Cluster Association Rules

Name	Description
Basket [B]	Member Methods Basket
Output File [O]	Output.txt
Minimum Item Support [MIS] File	Weights.txt
Minimum Confidence [MinConf]	0.60

Table 14 presents the most important rules derived by the application of the *MMS Apriori*.

Table 14:

Member methods Inter – Cluster Association Rules

Condition	Conse-Quence	Confi-dence	Occur rence	Description
Add	Remove	1	8	Those two functions are used in the FormTemplates library that supports the forms the user interacts with.
Assign Parameters	Get Parameters	1	237	Those methods are used mostly in the SQLStubs library which is responsible for the communication with the database.
OnOK	OnCancel	1	125	These methods are typical Windows events.
Initialize Component	Dispose	1	170	Those two methods are used in the FormTemplates library that supports the forms the user interacts with.

4.2.2.3 Intra-Cluster Association Rules Mining

Intra-Cluster Association Rules Mining aims at discovering associations between items that reside within the same cluster. It can help the maintenance engineer to discover interesting rules focused on the member methods of each cluster separately. Table 15 shows the values assigned to the input parameters in order to perform this type of mining task.

Table 15:

Parameters used for Methods Intra – Cluster Association Rules

Name	Description
Basket [B]	Member Methods Basket
Output File [O]	Output.txt
Minimum Item Support [MIS] File	Weights.txt
Minimum Confidence [MinConf]	0.10

The most interesting rules generated for the Member Methods intra-cluster

Association Rules Mining are shown in Table 16.

Table 16:
Member methods Intra – Cluster Association Rules

Condition	Conse-quence	Confi-dence	Occur rence	Cluster	Description
updateList View	InitializeCo mponent	1	13	8	Those two methods are used in the FormTemplates library that supports the forms the user interacts with.
groupingFi eldCb_Sele ctedIndexC hanged	InitializeCo mponent	1	17	8	Those two methods are used in the FormTemplates library that supports the forms the user interacts with.
groupComb oSelectedI ndexChang ed	InitializeCo mponent	1	20	8	Those two methods are used in the FormTemplates library that supports the forms the user interacts with.
OnSelected	Unload	0.846	11	2	These methods are typical Windows events
Refresh	Unload	0.875	7	2	These methods are typical Windows events
Insert	Remove	1	19	4	Those two methods are used in the FormTemplates library. Insert inserts a value from a Windows component in a specific position.
Clear	Remove	1	8	4	Those two methods are used in the FormTemplates library. Clear is responsible for the initialisation of the components of the form.
GetEnumer ator	Add	1	8	3	Those two methods are used in the RemoteComponentInterface library, which is the only common library between the server and the client.
Remove	Add	1	8	3	Those two methods are used in the SQLStubs library, which contains the prototypes of the stored procedures of the database
FetchFrom	CreateFor	1	3	3	Those two methods are used in the SQLStubs library. FetchFrom retrieves a code statement and CreateFor creates a code statement.
Receive	Send	1	2	0	Those two methods are found in the AuxBooksSystemRuntime, which is a utility responsible for updating automatically the clients with the new versions reside on the server.

4.2.2.4 Member Methods Analysis Results

We performed Member Methods entity analysis by using clustering and association rules mining; we discuss here the derived results.

At first maintenance engineers of the Books Publishing System were given an overview of the most important groups of types of Member Methods. This is useful when the maintenance engineer starts reading the code at an initial level in order to find the piece of code that requires enhancement or fixing. By clustering the Member Methods in particular the maintenance engineer has the opportunity to understand easier the behavioural domain of the system.

On the other hand we identified associations between the Member Methods of the system. This can facilitate the maintenance engineers to identify areas that it is very likely to be affected in a potential refactoring. For example if an engineer wants to remove the Receive method from a class, then he has to examine what will happen with Send method which coexists with it in every class it appears. Another example also is for the maintenance engineer to examine if he can create a single method by combining the two methods together.

4.2.3 Method Parameters Analysis

The Method Parameters entity belongs to the behavioural domain like its parent entity (Member Methods) as described at §3.1. As such we can apply both the Clustering and Association Rules data mining tasks.

4.2.3.1 Cluster Analysis

Table 17:
Parameters used for the Clustering of member methods parameters

Name	Description
Input Dataset [D]	A flat file containing records describing the Methods Parameters Entity.
Maximum Passes [P]	The chosen value is 3, as we wanted to increase the accuracy of clustering. Specifying more passes through the data improves the quality of the generated clusters.
Maximum Number of Clusters [C]	The chosen value is 9, as the maintenance engineers were interested in grouping the Classes Entity in a number of clusters that will give an overview of the system while on the same time providing the opportunity of finding niches.
Similarity threshold [S]	The chosen value is 0.8, as we wanted only records with 80% identical fields to be assigned on the same cluster
Accuracy Improvement [A]	The chosen value is 3, as we wanted clustering to be as accurate as possible and to limit the number of the processing time.

Table 17 shows the values assigned to the input parameters in order to perform the Clustering mining task.

The fields `ParamName, ParamType` and `ParamUse` are the most important for forming the clusters of this entity. Table 18 presents the main characteristics of the derived clusters.

Table 18:
Member method parameters clusters' main characteristics

Name (Number)	Size %	Characteristics
3	21.92	The parameters belonging to this cluster are objects passed by value. The predominant names are Sender, asyncState and Value.
4	20.54	The parameters belonging to this cluster are EventArgs passed by value. The predominant name is e.
9	18.14	The parameters belonging to this cluster are the types of System.AsyncCallbacks, IntPtr and Content and are passed by value. The predominant names are callback, hWnd and Value.
1	9.38	The parameters belonging to this cluster are strings passed by value. The predominant names are Report, remoteFile and Description.
5	9.30	The parameters belonging to this cluster are ISQLManagers passed by value. The predominant name is Conn.
2	7.46	The parameters belonging to this cluster are bools and Points passed by value. The predominant names are disposing, screenPos, show and xmlOut.
6	7.28	The parameters belonging to this cluster are Int and Graphics passed either by value or by reference. The predominant names are Index, g, Parameters and xmlln.
8	4.47	The parameters belonging to this cluster are System.IAsyncResult, Rectangle, DataRow and Control passed either by value or by reference. The predominant names are AsyncResult, row, control and itemRow.
7	1.51	The parameters belonging to this cluster are decimal, uInt, FileObject and RemFileObject passed either by value or by reference. The predominant names are obj, contributor, flags, BookId and OpId.

As derived from the clustering analysis most of the parameters are passed by Value and the majority of them belong to Windows built-in methods. The parameters also with the highest frequency, like the ones of clusters 3 and 4 don't have explanatory names (i.e. `System.EventArgs e`). On the other hand clusters 1, 5 and 7 include parameters that have more comprehensible names, a fact that makes the task of a maintenance engineer easier.

4.2.3.2 Inter Cluster Association Rules Mining

In order to find the associations between different clusters we performed the Association Rules Mining for all the clusters. The result was to generate rules and to discover frequent itemsets concerning all the clusters in total. Table 19 shows the values assigned to the input parameters in order to perform the Association Rules mining task.

Table 19:
Parameters used for Member Method Parameters Inter – Cluster Association Rules

Name	Description
Basket [B]	Method Parameters Basket
Output File [O]	Output.txt
Minimum Item Support [MIS] File	Weights.txt
Minimum Confidence [MinConf]	0.60

The most important rules concerning the methods' parameters, derived by the application of the *MMS Apriori* are presented in Table 20.

Table 20:
Member method parameters Inter – Cluster Association Rules

Condition	Conse-quence	Confi-dence	Occur rence	Description
Sender	E	0.989	930	Those parameters are used from Windows events like btnAdd_Click.
Callback	asyncState	1	93	Those parameters are used from methods of the Printing_DTCMP library that incorporates functionality regarding the reporting services of the system.
Index	Value	0.620	49	Those parameters are used from methods that draw Windows forms like InsertWindowValue.
Width	Height	1	6	Those parameters are used from methods that draw or move Windows forms like MoveWindow.
Operator	Row	1	18	Those parameters are used from methods of the PublicationDep_DTCMP library that incorporates functionality regarding the tasks of this department.
HistoryID	Report	1	7	Those parameters are used from methods of the Printing_DTCMP library that incorporates functionality regarding the reporting services of the system.
Rank	hsClasses	0.75	3	Those parameters are used from method AddClasses wich belongs to the PublicationDep_DTCMP library.
Username	Password	0.667	2	These parameters are used from methods like Login of the AuxBooksSystemRuntime or LogonUser of the Printing_DTCMP library.

4.2.3.3 Intra Cluster Association Rules Mining

Intra-Cluster Association Rules Mining aims at discovering associations between items that reside within the same cluster. It can help the maintenance engineer to discover interesting rules focused on the member methods of each cluster separately. Table 21 shows the values assigned to the input parameters in order to perform this type of mining task.

Table 21:
Parameters used for Member Method Parameters Intra – Cluster Association Rules

Name	Description
Basket [B]	Method Parameters Basket
Output File [O]	Output.txt
Minimum Item Support [MIS] File	Weights.txt
Minimum Confidence [MinConf]	0.10

The most interesting rules generated for the Method Parameters intra-cluster Association Rules Mining are presented in Table 22.

Table 22:
Member Method parameters Intra – Cluster Association Rules

Condition	Conse-quence	Confi-dence	Occur-rence	Cluster	Description
Pt	hWnd	1	2	8	Those parameters are used from methods that are responsible for the position or drawing of Windows forms, like MoveWindow.
Ps	hWnd	1	2	8	Those parameters are used from methods that are responsible for the drawing of Windows forms, like BeginPaint.
Height	Width	1	6	5	Those parameters are used from methods that are responsible for the drawing or positioning of Windows forms, like SetWindowPos, DrawShadowVertical.

4.2.3.4 Member Method Parameters Analysis Results

We performed Member Method Parameters entity analysis by using clustering and association rules mining; we discuss here the derived results.

At first the maintenance engineers of the Books Publishing System were given an overview of the most important groups of types of Member Method Parameters. This

can also help the maintenance engineer to start inspecting the code for potential changes.

On the other hand we identified associations between the Member Method Parameters of the system. By studying those associations that further describe the signatures of the Member Methods of the system; a maintenance engineer can easier understand their scope and functionality. This can also help him to assess how easy it is to refactor a method. For example a method that its signature consists of *Callback* and *asyncState* which appear always together is difficult to change by either removing one of them or by creating two separate methods with one parameter each.

4.2.4 Member Data Analysis

The Member Data entity belongs to the structural domain as described at §3.1. As such we can apply both the Clustering and Association Rules data mining tasks.

4.2.4.1 Cluster Analysis

Table 23 shows the values assigned to the input parameters in order to perform the Clustering mining task.

Table 23:
Parameters used for the Clustering of Member Data

Name	Description
Input Dataset [D]	A flat file containing records describing the Methods Parameters Entity.
Maximum Passes [P]	The chosen value is 5, as we wanted to increase the accuracy of clustering. Specifying more passes through the data improves the quality of the generated clusters.
Maximum Number of Clusters [C]	The chosen value is 9, as the maintenance engineers were interested in grouping the Classes Entity in a number of clusters that will give an overview of the system while on the same time providing the opportunity of finding niches.
Similarity threshold [S]	The chosen value is 0.8, as we wanted only records with 80% identical fields to be assigned on the same cluster
Accuracy Improvement [A]	The chosen value is 2, as we wanted clustering to be as accurate as possible and to limit the number of the processing time.

The fields `DataCategory`, `DataType` and `DataName`, are the most important for forming the clusters of this entity. The main characteristics of the derived clusters are presented in Table 24.

Table 24:
Member Data parameters clusters' main characteristics

Name (Number)	Size %	Characteristics
8	16.49	The data belonging to this cluster are private, and their type is System.Int32. The predominant names are _RETURN_VALUE, _book_id, _published_book_id and _contract_id.
4	15.33	The data belonging to this cluster are private, and their types are FormTemplates_CustomButton, FormTemplates_CustomGroup, FormTemplates_CustomList, and System_Decimal. The predominant names are customGroup1, customGroup2, customGroup3, contractorInfoGb and booklist.
2	12.98	The data belonging to this cluster are private, and their type is System.Windows.Forms.Label. The predominant names are label1, label2, nameLbl, cityLbl and addressLbl.
9	12.85	The data belonging to this cluster are private, and their types are System.Windows.Forms.ColumnHeader and System.String. The predominant names are columnHeader1, columnHeader12, _title, _paperSize, and _Phones.
7	11.63	The data belonging to this cluster are private, and their types are System.Windows.Forms.TextBox,System.Windows.Forms.ComboBox, System.Windows.Forms.Button and System.Byte. The predominant names are groupCombo, nameTxt, cityTxt, button1, and _type.
1	11.42	The data belonging to this cluster are private, and their types are System.Double,System.ComponentModel.IContainer, System.ComponentModel.Container,System.Windows.Forms.CheckBox and System.Windows.Forms.RadioButton. The predominant names are components, _typographics, _xrFive, _xrFour, and _xrThree.
3	9.91	The data belonging to this cluster are protected, and their types are int, bool, color, string and Rectangle. The predominant names are _style, _manager, _style, _direction, and _redocker.
5	9.38	The data belonging to this cluster are public, and their types are int, string, bool, uInt and System.DateTime. The predominant names are Name, IParam, Login, Pwd, and OpId and SessionId.

As derived from the clustering analysis the Member Data of this system have explanatory names, a fact that makes them more comprehensible to the maintenance engineers. The most important entity of the system is Book as the most frequent Ids are `book_id` and `published_book_id` in cluster 8. An interesting observation is that in cluster 5 there are public Ids (i.e. `OpId`, `_uid`, `TaskID`, `SubscriprtionID`, `SessionId`), which makes more possible the creation of couplings among the classes of the system.

4.2.4.2 Inter-Cluster Association Rules Mining

In order to find the associations between different clusters we performed the Association Rules Mining for all the clusters. The result was to generate rules and to discover frequent itemsets concerning all the clusters in total. Table 25 shows the values assigned to the input parameters in order to perform the Association Rules mining task.

Table 25:

Parameters used for Member Data Inter – Cluster Association Rules

Name	Description
Basket [B]	Member Data Basket
Output File [O]	Output.txt
Minimum Item Support [MIS] File	Weights.txt
Minimum Confidence [MinConf]	0.50

The most important rules derived by the application of the MMS Apriori are presented in Table 26.

Table 26:

Member Data Inter – Cluster Association Rules

Condition	Consequence	Confidence	Occurrence	Description
_RETURN_VALUE	_book_id	0.778	105	Those member data are used mostly in classes of the SQLStubs library.
_RETURN_VALUE	_published_book_id	0.7	63	Those member data are mostly used in classes of the SQLStubs library.

4.2.4.3 Intra-Cluster Association Rules Mining

Intra-Cluster Association Rules Mining aims at discovering associations between items that reside within the same cluster. It can help the maintenance engineer to discover interesting rules focused on the member data for each cluster separately. Table 27 shows the values assigned to the input parameters in order to perform this type of mining task.

Table 27:
Parameters used for Member Data Intra – Cluster Association Rules

Name	Description
Basket [B]	Member Data Basket
Output File [O]	Output.txt
Minimum Item Support [MIS] File	Weights.txt
Minimum Confidence [MinConf]	0.50

The most interesting rules generated for the Member Data intra-cluster Association

Rules Mining are shown in Table 28.

Table 28:
Member Data Intra – Cluster Association Rules

Condition	Consequence	Confidence	Occurrence	Cluster	Description
_yp_aps_id	_school_year, _published_book_id	1	1	8	Those member data are mostly used in classes of the SQLStubs library. Those classes represent the stored procedures that reside on the database.
_atelie_id	_book_id	0.75	6	8	Those member data are mostly used in classes of the SQLStubs library. Those classes represent the stored procedures that reside on the database.
_school_year	RETURN_VALUE	1	14	8	Those member data are mostly used in classes of the SQLStubs library. Those classes represent the stored procedures that reside on the database.
_contract_id	_published_book_id	0.66	6	8	Those member data are mostly used in classes of the SQLStubs library. Those classes represent the stored procedures that reside on the database.
_pages	_typeCover	1	3	6	Those member data are mostly used in classes of the SQLStubs library. Those classes represent the stored procedures that reside on the database.
Dsbooks	rbAll, rbPrimary, rbHighSchool, rbCollege, rbTEE	1	4	5	Those member data are used from classes that reside on the client part of the application.
Periodic_check_id	txbParatasi	1	3	5	Those member data are components used from the classes of the TE_TECHNOLOGY_H library which incorporates processes concerning the

Condition	Consequence	Confi-dence	Occur-rence	Cluster	Description
					technology department of the organisation.
_name	_value	0.83	5	3	Those member data are mostly used in classes of the SQLStubs library. Those classes represent the stored procedures that reside on the database.
_aclass	_cclass	1	4	3	Those member data are mostly used in classes of the SQLStubs library. Those classes represent the stored procedures that reside on the database.
supervisor NameTxt	addressTxt	0.916	11	0	Those member data are components used from the classes of the TE_GRAPHICS_H library which incorporates processes concerning the graphics department of the organisation.
vatTxt	cityTxt	1	12	0	Those member data are components used from the classes of the TE_CONTRACTS_H library which incorporates processes concerning the contracts department of the organisation.
photoCB	montazCB	1	2	0	Those member data are components used from the classes of the TE_GRAPHICS_H library which incorporates processes concerning the graphics department of the organisation.

4.2.4.4 Member Data Analysis Results

We performed Member Data entity analysis by using clustering and association rules mining; we discuss here the derived results.

At first the maintenance engineers of the Books Publishing System were given an overview of the most important groups of the types of Member Data entity. This is useful when the maintenance engineer starts reading the code at an initial level in order to identify pieces of code that require either to evolve or to get corrected. By clustering the Member Data in particular the maintenance engineer has the opportunity to understand easier the behavioural domain of the system.

We also identified a potential problematic area, as a cluster with public IDs was found. In general the use of public member data is not considered as a good programming technique as it violates the principle of encapsulation. It also makes more possible the creation of couplings among the classes of the system.

On the other hand we identified associations between the Member Method Parameters of the system. Those associations can help the maintenance engineer to better understand how the classes of the system are structured; and what are the input and output data that affect their behaviour.

5. Conclusions and future work

This section presents conclusions drawn by evaluating the proposed methodology. Directions for future work are also discussed here.

5.1 Conclusions on the Proposed Methodology

The main goal of this research work was to provide the maintenance engineer an overview of the main aspects of the software system in order to facilitate its comprehension. For this reason a framework which employs Clustering and Association Rules Mining techniques, and the respective input model that supports them, was developed. It was designed for object oriented languages and more specifically for C# and was tested in an industrial application of a substantial size (i.e. 693 source code files, 1242 classes, 5976 member methods, 5150 parameters and 6919 member data). That differentiates it from [14], [17], [27], [28] that are designed for procedural programming languages like C and COBOL.

The proposed solution is semi automated, unlike [14], [26], [27] and [28], as the parsing engine extracts the data from the source code and stores them on a database. It is also more complete than [19], [20] because as soon as the data extraction finishes, the maintenance engineer has the ability to use both Clustering and Association Rules in order to comprehend the system under maintenance. More specifically by using this methodology he/she has the ability to get a quick and rough grasp of a software system

at first, and then to try to identify hidden relationships between classes, member data, methods and method parameters.

On the other hand our methodology analyses only the static dependencies of system's entities unlike [31] which uses clustering in order to study the dynamic dependencies of a system under maintenance. We also use the K-Means clustering algorithm which has the drawback that the user has to define the number of the derived clusters. On the contrary the work in [19] employs the Hierarchical Agglomerative Clustering (HAC) algorithm which automatically defines the number of the derived clusters.

5.1.1 Conclusions from Clustering Analysis

The scope of using the K-Means clustering technique was to provide the maintenance engineers with an overview of the most important groups of the types of system's entities as described in §3.1. This methodology is useful especially when the maintenance engineer starts inspecting the code at an initial level in order to identify pieces of code that require changes either for evolving or getting corrected, an activity which is time consuming. Having seen at first an overview with the main aspects of the system the maintenance engineer gains a familiarity with it and then it is easier for him to start inspecting the system.

A good example of this was the discovery of the existence of "god classes" as there were classes that their number of methods is much above the average which is 5 methods per class.

We also discovered patterns that reflected the niches of the Books Publishing System. For example the 5th cluster of the Member Methods consists of methods that are responsible for the format and rounding of numbers.

The most important entity of the system seems to be Book as the most frequent Ids are `book_id` and `published_book_id` in the 8th cluster of member data. Another interesting observation is that in cluster 5 there are public Ids (i.e. `OpId, _uid, TaskID,`

`SubscriprtionID, SessionId`), which makes more possible the creation of couplings among the classes of the system.

By using also the clustering technique we provided the maintenance engineers the ability to verify if good design techniques are followed along the development of the system. For example we checked and we found that all the base classes found were abstract, which is a sign of a good design technique.

5.1.2 Conclusions from Association Rules Mining

The aim of using the Association Rules mining was trying to discover the relationships between the elements of a software system. For this reason we employed an innovative algorithm, *MMS Apriori* which was based on algorithm Apriori and could find rules among items with different supports. We used both inter and intra cluster association rules techniques in order to discover the most important rules concerning the system in total at first and then each cluster separately.

At first we showed that the application of the Association Rules technique can facilitate the maintenance engineer to identify areas that it is very likely to be affected in a potential refactoring. That was presented in §4.2.2.4 and §4.2.3.4.

At §4.2.2.2-3, §4.2.3.2-3, §4.2.4.2-3 rules characterising system's member data, methods and their parameters were presented. Some of them like `supervisorNameTxt->addressTxt, contract_id->_published_book_id, Assign Parameters->Get Parameters` were characterised by system's developers interesting as they provided them a clearer picture with the interrelationships between system's entities.

5.2 Future Work

We consider the following various alternatives in order to enhance the proposed methodology:

Systems' components clustering, based on their dynamic dependencies

This research work presented the analysis of static dependencies between the components of the system. It would be of great interest to attempt to evaluate the usefulness of analysing the dynamic dependencies of a software system's artefacts [31].

Integration of more data mining algorithms

The proposed methodology integrates the K-Means and MMS Apriori algorithms. However it may be useful if more custom data mining algorithms were integrated in this framework. This would result in a complete system for automated program and system comprehension. A good example is the application of Spectral Clustering in order to investigate the dependencies based on the objects that are invocated in each file, and on Hierarchical Agglomerative Clustering (HAC) algorithm which has the advantage that automatically defines the number of the derived clusters.

Enriching the input model

Based on our methodology we extracted information that described the behavioural and structural domains of a software system. This input model can be enriched by investigating also the logical domain of a software system, which includes the analysis of the dependencies between the files of the system under examination. This can be done by analysing either the `using` (C#) or the `include` (Java, C++) statements of the beginning of each file that defines a class. In other words this domain is consisted of the analysis of the correlations between the files of the system's sections. Fig. 4 suggests such a more detailed model.

Another way to further develop the input model is to take into consideration and further investigate the object and method invocation, the instance-class relations, the definitions of the constants and the `enum` structures etc. This enrichment of the input model could help us understand in more depth the domains of a software system.
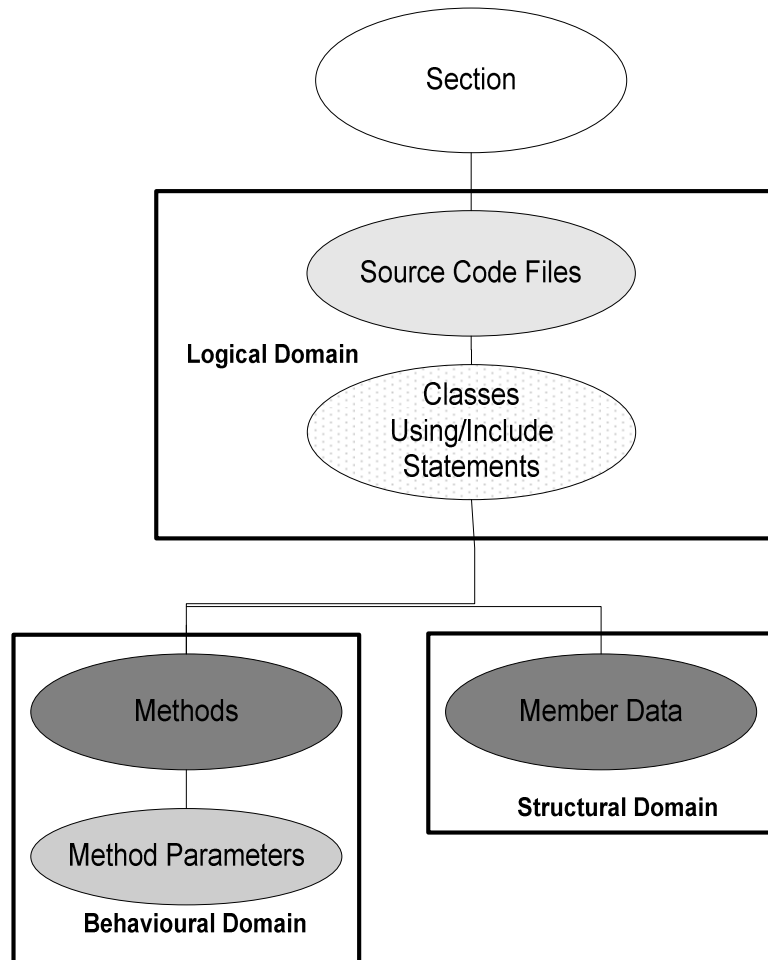
Figure 4-Enriched Input Model

Using Metrics for OO Programs as indicators for evaluating a system's maintainability

The main goal of this research work is to provide the maintenance engineer with an overview of the main aspects of the software system in order to facilitate its comprehension. The proposed methodology extracts information from the source code that describes the main entities of an OO program. It seems also promising to employ object oriented metrics that can be used as indicators for either evaluating, either predicting a system's maintainability [25]. Such metrics are the Maintainability Index and the Chidamber and Kemerer Metrics (Weighted Methods per Class (WMC), the

Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object Classes (CBO), Response for a Class (RFC), and the Lack of Cohesion in Methods (LCOM).

Tune the methodology for other OO languages

The proposed methodology processes information derived only from C# source code files (cs). It is of great interest to extract information from other OO languages like C++, Java and Borland Delphi.

Employ more advanced extraction techniques

This research work employed the regular expressions technology in order to parse the source code and extract the required data. A more advanced technique like using a lexical parser and store its output either in a database or in XML files is under consideration.

Acknowledgements

References

[1] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases, in: Proc. 20th Int'l Conf. Very Large Data Bases, (1994) 487-499.

[2] N. Anquetil and T. C. Lethbridge, Experiments with Clustering as a Software Remodularization method, in: Proc. 6th Working Conf. Reverse Engineering (WCRE 99), IEEE Comp. Soc. Press, (1999) 235-255.

[3] F. Balmas, H. Wertz and J. Singer, Understanding Program Understanding, in: Proc. 8th Int'l Workshop Program Comprehension (IWPC 00), IEEE Comp. Soc. Press, (2000) 256.

[4] W. B. Boehm, Software Engineering Economics, (Prentice Hall PTR1981).

[5] S. Brin, R. Motwani, J.D. Ullman and S. Tsur,Dynamic Itemset Counting and Implication Rules for Market Basket Data, in: Proc. ACM SIGMOD, (1997) 265-276.

[6] M. H. Dunham, Data Mining, Introductory and advanced topics, (Prentice Hall 2002).

[7] U. Fayyad, G. Piatetsky-Shapiro, and R. Uthurusamy, From Data Mining to Knowledge Discovery: An Overview, Advances in Knowledge Discovery and Data Mining, (AAAI Press/The MIT Press 1996).

[8] J. Han and M. Kamber, Data Mining: Concepts and Techniques, (Academic Press 2001).

[9] http://www.swat.uwaterloo.ca/~swagkit .

[10] Y. Kanellopoulos and C. Tjortjis, Data Mining Source Code to Facilitate Program Comprehension: Experiments on Clustering Data Retrieved from C++ Programs, in: Proc IEEE 12th Int'l Workshop Program Comprehension (IWPC 2004), IEEE Comp. Soc. Press, (2004) 214-223.

[11] I. N. Kouris, C. Makris, and A. Tsakalidis, An Improved Algorithm for Mining Association Rules Using Multiple Support Values, in: Proc 16th Int'l FLAIRS Conf., (2003) 309-314.

[12] T. Kunz and J. P. Black, Using Automatic Process Clustering for Design Recovery and Distributed Debugging, IEEE Transactions on Software Engineering, 21(6), (1995) 515-527.

[13] B. Liu, W. Hsu, and Y. Ma, Mining Association Rules with Multiple Minimum Supports, in: Proc. ACM SIGKDD Conf. on Knowledge Discovery & Data Mining, (1999) 337-341.

[14] S. Mancoridis, B.S. Mitchell, Y. Chen and E.R. Gansner, Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures, in: Proc.

Int'l Conf. Software Maintenance (ICSM 99), IEEE Comp. Soc. Press, (1998) 50-59.

[15] O. Maqbool, H.A. Babri, A. Karim, and M. Sarwar, Metarule-guided association rule mining for program understanding, Software, IEE Proceedings, 152(6) (2005) 281- 296.

[16] J. Moad Maintaining the Competitive Edge, Datamation, 36(4) (1990) 61-66.

[17] C. M. de Oca and D. L. Carver, Identification of Data Cohesive Subsystems Using Data Mining Techniques, in: Proc. Int'l Conf. Software Maintenance (ICSM 98), IEEE Comp. Soc. Press, (1998) 16-23.

[18] T. M. Pigoski, Practical Software Maintenance: Best Practices for Managing your Software Investment, (Wiley Computer Publishing 1996).

[19] D. Rousidis and C. Tjortjis, Clustering Data Retrieved from Java Source Code to Support Software Maintenance: A Case Study, in: Proc IEEE 9th European Conf. Software Maintenance and Reengineering (CSMR 05), IEEE Comp. Soc. Press, (2005) 276-279.

[20] K. Sartipi, K. Kontogiannis and F. Mavaddat, Architectural Design Recovery Using Data Mining Techniques, in: Proc. 2nd European Working Conf. Software Maintenance Reengineering (CSMR 00), IEEE Comp. Soc. Press, (2000) 129-140.

[21] J. Singer, Practices of Software Maintenance, in: Proc. Int'l Conf. Software Maintenance (ICSM 98), Comp. Soc. Press, (1998) 139-145.

[22] H. Sneed and T. Dombovari, Comprehending a Complex, Distributed, Object-Oriented Software System a Report from the Field, in: Proc. 7th Int'l Workshop Program Comprehension (IWPC 99), IEEE Comp. Soc. Press, (1999) 218-225.

[23] I. Sommerville, Software Engineering, 6th edition, (Addison-Wesley 2001).

[24] D. Spinellis, Code Reading: The Open Source Perspective, (Addison Wesley 2003).

[25] D. Spinellis, Code Quality: The Open Source Perspective, (Addison-Wesley 2006).

[26] C. Tjortjis and P.J. Layzell, Expert Maintainers' Strategies and Needs when Understanding Software: A Qualitative Empirical Study, in: Proc. IEEE 8th Asia-Pacific Software Engineering Conf. (APSEC 2001), IEEE Comp. Soc. Press, (2001) 281-287.

[27] C. Tjortjis C., L. Sinos and Layzell P.J., Facilitating Program Comprehension by Mining Association Rules from Source Code, in: Proc. IEEE 11th Int'l Workshop Program Comprehension (IWPC 03), IEEE Comp. Soc. Press, (2003) 125-132.

[28] C. Tjortjis, N. Gold, P.J. Layzell and K. Bennett, From System Comprehension to Program Comprehension, in: Proc. IEEE 26th Int'l Computer Software Applications Conf. (COMPSAC 02), IEEE Comp. Soc. Press, (2002) 427-432.

[29] A. Turtschi, S. Nandu, dotthatcom.com, G. Hack, J. Werry, J. Albahari, and W.M. Lee, C# .NET, Web Developer's Guide, (Syngress Publishing 2002).

[30] V. Tzerpos and R. Holt, Software Botryology: Automatic Clustering of Software Systems, in: Proc. 9th Int'l Workshop Database Expert Systems Applications (DEXA 98), IEEE Comp. Soc. Press, (1998) 811-818.

[31] C. Xiao and V. Tzerpos, Software Clustering on Dynamic Dependencies, in: Proc. IEEE 9th European Conf. Software Maintenance and Reengineering (CSMR 05), IEEE Comp. Soc. Press, (2005) 124-133.