Mining Source Code Elements for Comprehending Object-Oriented Systems and Evaluating Their Maintainability

Yiannis Kanellopoulos
The University Of Manchester, School Of Informatics, U.K.
Yiannis.Kanellopoulos@postgrad.manchester.ac.uk

Thimios Dimopulos
Deutsche Telekom Labs, Berlin, Germany
Thimios.Dimopulos@telekom.de

Christos Tjortjis
The University Of Manchester, School Of Informatics, U.K.
C.Tjortjis@manchester.ac.uk

Christos Makris
University Of Patras, Department of Computer Engineering & Informatics
makri@ceid.upatras.gr

ABSTRACT

Data mining and its capacity to deal with large volumes of data and to uncover hidden patterns has been proposed as a means to support industrial scale software maintenance and comprehension. This paper presents a methodology for knowledge acquisition from source code in order to comprehend an object-oriented system and evaluate its maintainability. We employ clustering in order to support semi-automated software maintenance and comprehension.

A model and an associated process are provided, in order to extract elements from source code; K-Means clustering is then applied on these data, in order to produce system overviews and deductions. The methodology is evaluated on *JBoss*, a very large Open Source Application Server; results are discussed and conclusions are presented together with directions for future work.

Keywords

Code mining, clustering, software maintenance, program comprehension, metrics, maintainability.

1. INTRODUCTION

Software maintenance is considered a very important and complex stage in software lifecycle typically consuming 50-70% of the total effort allocated to a software system [14], [17]. According to Sutherland [19], US companies spent more than $70 billion annually on software maintenance. There are several studies for evaluating a system's maintainability and controlling the effort required to carry out maintenance activities [1], [3], [20]. Additionally *program comprehension* is an important part of this stage, especially when program structure is complex, modularity has deteriorated and the documentation is not up to date. 50-90% of maintainers' time is reported to be spent on *program comprehension* [21].

The aim of this work is to facilitate maintenance engineers to comprehend the structure of a software system and assess its maintainability. For this reason elements from source code are collected, including:

- Entities that belong either to the behavioral (as classes, member methods) or the structural domain (as member data).
- Attributes that describe the entities (such class name, superclass, method name etc.).
- Metrics used as additional attributes that facilitate the software maintainer to comprehend more thoroughly the system under maintenance.

Our research objectives include: the definition of an input data model, the population of a database containing data extracted from source code, the application of clustering on the database and the evaluation of results and the usefulness of mining data from source code. For this purpose we used a large open source software system used in industrial real life applications: the *JBoss* Open Source Application Server, version 4.0.3SP1 [24]. Results have shown that clustering data extracted from source code and software metrics can uncover a lot of useful information about various aspects of software systems.

The remaining of this paper is organized as follows: Section 2 reviews existing work in the area of data mining for program comprehension and maintainability evaluation. Section 3 outlines the proposed method for extracting metrics and elements from Java source code, the input data model and the clustering method. Section 4 assesses the accuracy of the output of the proposed framework, analyses its results and outlines deductions from its application. Finally, conclusions and directions for future work are presented in Section 5.

2. BACKGROUND

Developing software systems of any size which do not need to be changed is unattainable [17]. Such systems, once in use, need to be functional and flexible in order to operate correctly and fulfill their mission, as new requirements emerge. Consequently, software systems remain subject to changes and maintenance throughout their lifetime. Program comprehension is required by maintenance engineers in order to identify problematic files or modules; and to assess their maintainability [14].

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 [12], [22], [23], [24].

Data mining techniques have been used previously, for identification of subsystems based on associations (ISA methodology) [12]. 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.

Clustering has also been used to support software maintenance and systems knowledge discovery. A method for grouping Java code elements together according to their similarity was proposed in [15]. It focuses on achieving a high level system understanding. The method derives system structure and interrelationships, as well as similarities among systems components, by applying cluster analysis on data extracted from source code. Hierarchical

Agglomerative Clustering was employed to reveal similarities between classes and other code elements thus facilitating software maintenance and Java program comprehension.

An approach for the evaluation of dynamic clustering was presented in [23]. The scope of this solution was to evaluate the usefulness of providing dynamic dependencies as input to software clustering algorithms. This method was applied to *Mozilla*, a large open source software system with more than four million lines of C/C++.

All these approaches employ data mining techniques only to recover the structure of a software system. On the other hand [24] is employing clustering for predicting software modules' fault proneness and potential noisy modules. K-Means and Neural – Gas algorithms were employed in order to group together modules with similar software measurements. A software engineering expert inspected the derived clusters and labelled them as fault prone or not.

The value of our work that differentiates it from what presented above, is that we employ data mining techniques for both recovering the structure of a software artifact and assessing its maintainability. We do that by creating an input model which considers as program's entities' attributes both metrics (e.g. Chidamber and Kemerer metrics suite [5]) and elements from source code data (e.g. class name, method name, superclass etc.). This is presented in §3.2.

3. THE PROPOSED FRAMEWORK

This section presents the proposed framework for extracting metrics and elements from Java source code, the input data model and the clustering method.

3.1 Application Selection

In order to evaluate if and how mining elements from source code can facilitate software system comprehension and maintainability evaluation, we required a software system with the following required properties:

- Its size must be considerable, in order for clustering to be meaningful [23].
- Its source code must be publicly available.
- It should be Object Oriented as this paradigm is becoming increasingly popular [3].
- It should be used extensively.

Many open source systems have most of these properties; fewer have the last one. We selected *JBoss* as a test bed for this research work, as it does have these properties [24].

3.2 Input Model

Input model definition requires the specification of program entities and their attributes, which should be suitable for clustering. Clustering imposes requirements on the type and number of attributes, the lack of distinction between predictive and predicted attributes and so on [7], [9].

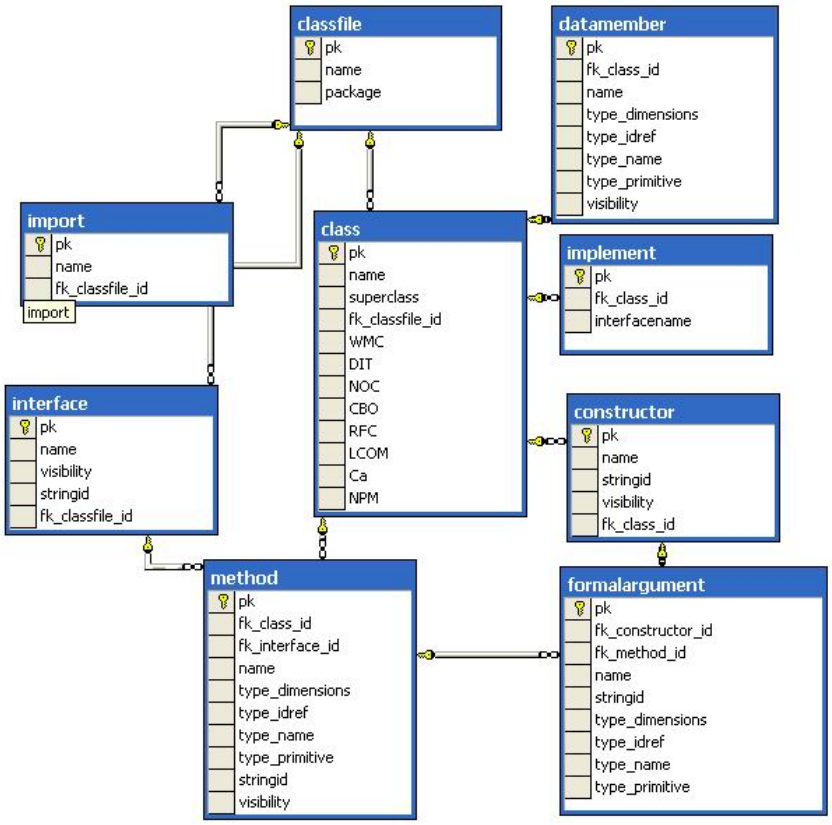

Figure 1: Input Model

Moreover entities need to contain a common set of attributes in order to achieve homogeneity; this allows for entities' comparison on the basis of their attributes, which is the main principle of clustering. The number of selected attributes needs also to be sufficient in order to avoid misleading comparisons or discovering accidental similarities. Selected attributes are both binary and qualitative, as they are predominant in a source code application domain [4].

The schema in Fig. 1 outlines the proposed input model in terms of entities and their respective attributes. Each entity is described by attributes thus formulating database tables.

The proposed input model consists of:

- Entities that belong either to behavioral (i.e. class, interface, member method) or structural domain (i.e. member data). In this research work we are mostly interested in working with classes, packages, classfiles, methods and member data.
- Attributes that describe the selected entities (class name, superclass, method name an so on).

Metrics which are considered as attributes that further describe an OO program's entities. More specifically and as depicted in Fig. 1, metrics are employed as attributes of the entity *Class*. In §3.4 we further describe the selected metrics and we justify why we think they are important in facilitating a maintenance engineer to comprehend and evaluate a system's maintainability.

3.3 Data Extraction Process

Source code representation in an abstract way is a key research topic [2]. Representing source code in an xml format can be useful for the following reasons:

- XML is a strict, well defined format that can be manipulated by a plethora of tools (i.e. parsers, editors etc) while it still remains a human readable format.
- An XML representation could also be abstract, independent of any programming language [6].

The scope of the proposed extraction process is to parse *javaml* xml files and extract elements and metrics from java code. The current version stores information like class name, package of the class, constructor names and their arguments, method names and arguments etc as depicted in Fig. 1. This information is subsequently stored in a relational database so that the Clustering Data Mining technique can be applied.

3.4 Metrics Selection and Description

According to Lehman [10] as software demonstrates regular behaviour and trends, these can be measured. Software evolution and maintenance require the collection of such metrics. This enables maintenance engineers to track status, control costs, and make decisions related to their maintenance tasks. These metrics can be useful as indicators for evaluating a system's maintainability and identifying potential problematic areas [13], [16]. We have to emphasise at this point that metrics can be used only as indicators for a system's maintainability, as this is a complex and largely subjective software attribute.

For this work we decided to employ the Chidamber and Kemerer metrics suite [5], as it can be applied to OO programs and can be used as a predictor and evaluator of maintenance effort according to [11]. More specifically we chose the following subset of this metrics suite:

- *Weighted Methods per Class (WMC)*, which is simply the sum of the complexities of its methods.
- *Coupling between Objects – Efferent Coupling (CBO)*, which represents the number of classes a given class, is coupled to.
- *Number of Children (NOC)*, which measures the number of immediate descendants of the class.
- *Depth of Inheritance Tree (DIT)*, which provides for each class a measure of the inheritance levels from the object hierarchy top.
- *Afferent (inward) Coupling (Ca)*, which measures the number of other classes that depend on the class under examination.
- *Number of Public Methods (NPM)*, which counts all the methods in a class that are declared as public.

Those selected metrics are used as additional attributes in order to further describe the *Classes* entity.

3.5 Clustering

At first the maintenance engineer 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 as it produces overviews of systems by creating mutually exclusive groups of classes, member data or methods, according to their similarities and hence reduces the time required to understand the overall system. Another contribution of this method is that it helps discovering programming patterns and "unusual" or outlier cases which may require attention.

For this purpose K-Means clustering was employed. Its popularity is largely due to its simplicity and low-time complexity. The input parameters we used for this algorithm are presented in Table 1.

Name	Description
Input Dataset [D]	The dataset used.
Max. Passes [P]	Maximum number of passes the algorithm goes through the data.
Max. No of Clusters [C]	Maximum number of clusters the algorithm generates. High values produce many small clusters facilitating outlier identification.
Similarity threshold [S]	It limits the values accepted as best fit for a cluster.
Accuracy Improvement [A]	Minimum % improvement on clustering quality after each pass.

Table 1: Input Parameters for *K-means Clustering*

4. RESULT EVALUATION

The proposed framework was evaluated in terms of ability to capture knowledge relevant to software maintenance activities, using the *JBoss* Open Source Application Server. 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 framework to a real software system, it is necessary to describe the experimental setup. The core idea was to extract elements and metrics from *JBoss* Open Source Application Server's source code and store these data in a database. The outcome of this step gave us the ability to:

1. Apply *K-Means Clustering* to the classes of *JBoss* based on their structural similarities, like the packages they belong to, or their respective superclasses. The derived clusters provide maintenance engineers with an overview of the basic parts of the system.
2. Apply *K-Means Clustering* to the classes of *JBoss* based on their metric values. The formed clusters can provide maintenance engineers with insights concerning the classes' characteristics described in §3.3. The input parameters for this task are presented in Table 2.
3. As soon as the clusters are formed, the maintenance engineer has the ability to mine inside each cluster separately, in order to find similarities concerning classes' structure (their respective methods and data for example).

Name	Description
Input Dataset [D]	A flat file containing records describing the Class Entity including metrics.
Max. Passes [P]	The chosen value is 2, as we wanted to increase the accuracy. More passes could improve quality.
Max. No of Clusters [C]	We used a range of values between 9 and 25, so that we could both extract system overviews and identify niches.
Similarity threshold [S]	The chosen value is 0.6, so that only records with 60% identical fields are assigned to the same cluster.
Accuracy Improvement [A]	The chosen value is 2; a trade off between clustering accuracy and limited processing time.

Table 2: Experimental Setup Parameters

4.2 Case Study: *JBoss* Application Server

JBoss is a free J2EE certified application server, and is one of the most widely used open source professional software [25]. Table 3 shows the statistics concerning its size.

Type	Value
Lines of Code (LOC)	1,615,289
Classes	6,448
Java Files	4,714
Packages	569

Table 3: *JBoss* Statistics

JBoss is built on a modular architecture on top of the JMX (Java Management Extension) infrastructure, which is a reusable framework that can expose applications to remote or local management tools. The major *JBoss* modules are manageable *MBeans* connected by the *MBean* server and are illustrated in Fig. 2.

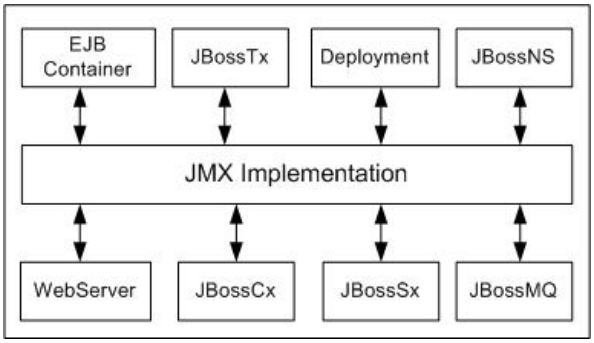

Figure 2: *JBoss* Model

4.2.1 System Overview

One of the main goals of this work was to provide the maintenance engineer 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. That reduces the required time to comprehend a system's structure. For this reason, we started our experiment by clustering *JBoss'* classes based on attributes like the package they belong to and their respective superclasses. Fig. 3 depicts the formed clusters of the classes of *JBoss* based on these attributes.

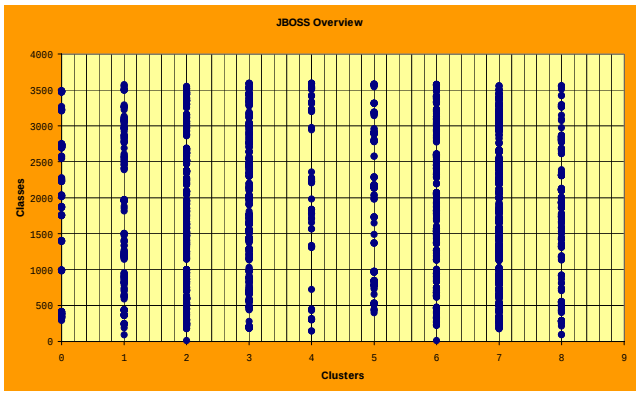

Figure 3: *JBoss* Clusters

In the X axis are the ids of the formed clusters while on the Y axis the ones of *JBoss'* classes. As we can see the clusters with the highest population are 2, 3, 6, 7, 8 while 0, 4, 5 represent a small part of the system. Based on Fig. 2 the parameter C (Maximum Number of Clusters) was set to 9 as we did not want to search for outliers but to give an overview of *JBoss'* main structure. Table 4 presents the formed clusters, and the predominant packages they are included in each cluster.

The first cluster (0) consists of packages defined by the JMX specification. `javax.management` and `org.jboss.mx` are related with *MBeans* which are Java objects that implement the standard *MBean* Interface. Another interesting cluster is the third one which consists of packages related to the `Aspect Oriented Programming` support that *JBoss* provides for AO middleware. Cluster 2 also consists of classes belonging to packages that are related to the administration of entity bean classes. More specifically `org.jboss.console` provides a simple web interface for managing the *MBean* server while `org.jboss.ejb3` is the most basic package for an *EJB* (Enterprise Java Beans) implementation.

Cluster S/N	Size %	Packages
0	5.92	`javax.management, org.jboss.mx.`
1	6.38	`org.jboss.ejb, javax.xml`
2	15.72	`org.jboss.console, org.jboss.ejb3`
3	15.75	`org.jboss.aop.deployment, org.jboss.aspects, org.jboss.mq`
4	3.07	`org.jboss.management`
5	5.77	`org.jboss.util, org.jboss.resource`
6	15.78	`org.jboss.ejb.plugins, org.jboss.invocation, org.jboss.metadata`
7	23.57	`org.jboss.axis, jboss.org.resource.adapter, .jdbc,org.jboss.webservice`
8	8.04	`org.jboss.security, org.jboss.tm`

Table 4: *JBoss* Clusters

4.2.2 Weighted Methods per Class (WMC - Class Complexity)

Fig. 4 depicts clusters formed of *JBoss* classes, based on their complexity. By examining these clusters, a maintenance engineer can identify classes with exceptional values and then examine them closely and consider refactoring to improve their design. More specifically, the X axis shows the cluster IDs while the Y axis shows WMC attribute values. The 14th, 12th and 22nd clusters (7% of the whole population) are formed by classes that exhibit high WMC values. In practice, these classes may be difficult to maintain and reuse. For example the `org.jboss.axis.message.SOAPElementAxisImpl` class, declares 141 methods in a 2100 lines java file. This class implements the base type of nodes of a SOAP message parse tree. Although such a size would usually imply low maintainability of a class, the purpose of this specific class (to represent a soap message) could justify its size.

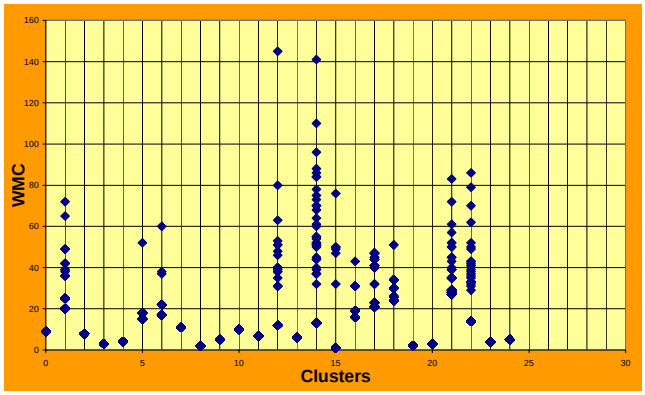

Figure 4: Weighted Methods per Class Clusters

The common characteristics of the classes that exhibit high WMC values are:

- Their methods return types are not primitive. Most of them are of the type `Logger`, `Document`, `WrappedStatement`, `CallableStatement`, `PreparedStatement`, and `ManagedEntityManagementFactory`.
- Most of them belong to the `jboss.org.axis`, `jboss.org.resource.adapter.jdbc`, `jboss.org.ejb3` packages.

On the other hand there are clusters like 3, 8 and 20 (25% of the whole population) that are formed by classes with very low WMC value. The common characteristic between these classes is that most of their methods return types are primitives like Boolean, long, int or void. Another interesting observation is that there are a significant number of classes having low WMC values that belong to the Aspect Oriented Programming package (`org.jboss.aop.deployment`).

4.2.3 Coupling Between Objects (CBO)

Fig. 5 depicts the formed clusters of the classes of *JBoss* based on their coupling. The 15th, 17th, 20th and 23rd (2.4% of the whole population) are formed by classes having high CBO values.

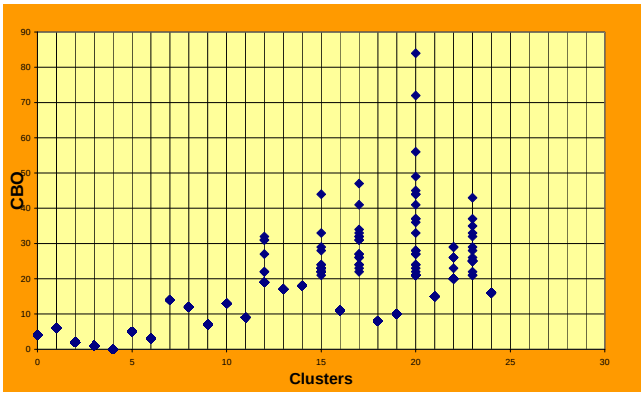

Figure 5: Coupling Between Objects Clusters

In practice these classes may be difficult to maintain and reuse as a change in a class may affect the classes it is coupled to. Inspecting the code for these classes a maintenance engineer can find out more. For example there is on the 20th cluster a class named `EJBQLToSQL92Compiler`, which implements `QLCompiler` and `JBossQLParserVisitor` interfaces and compiles `EJB-QL` and `JBossQL` into SQL using OUTER and INNER joins. It is a class of 1546 lines of code; has 84 couplings, and there are tens of methods like the following:

```
   public Object visit(ASTStringLiteral
node, Object data)
   {
      StringBuffer buf = (StringBuffer)
data;
      buf.append(node.value);
      return data;
   }
```

The common characteristics of classes like `EJBQLToSQL92Compiler` are:

- Their methods return types are not primitive. Most of them are of the type `Logger`, `Class`, `RemoteBinding`, `Thread`, `ThreadGroup`, `Emmiter`, and `Object`.
- Most of them belong to `org.jboss.ejb.plugins.cmp.jdbc`, `org.jboss.ejb.plugins.cmp.jdbc2`, and `org.jboss.ejb3.service` packages.

On the other hand there are clusters like 3 and 4, formed by classes with very low CBO value. Most of them belong to the `org.jboss.aop.deployment`, `javax.management`, `org.jboss.mx` and `org.jboss.mq` packages.

4.2.4 *Number of Children (NOC)*

Fig. 6 depicts the formed clusters of the classes of *JBoss* based on their number of children (NOC) attribute. Most of the classes (68.08% of the whole population) have no children, while another 30% have 1, 2 or 3 children.

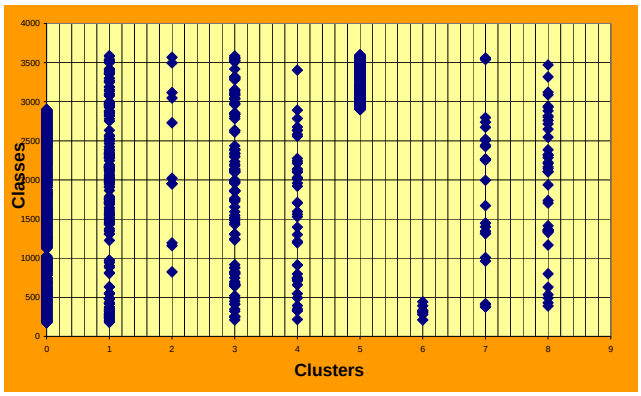

Figure 6: Number of Children Clusters

On the other hand there are clusters like 6, 7 and 8 (1.8% of the whole population) that contain classes with more than 8 children. In the 8th cluster a class named `ServiceMBeanSupport` that has 152 children was discovered. This class is a warning for a maintenance engineer to inspect for subclassing misuse as it has an abnormally number of children. The common characteristic of the methods that belong to the classes of clusters 6, 7 and 8 is that their return types are not primitive. Most of them are `loggers`, `nodes` or `vectors`. Classes that belong to those clusters may also be fundamental elements in JBoss's structure.

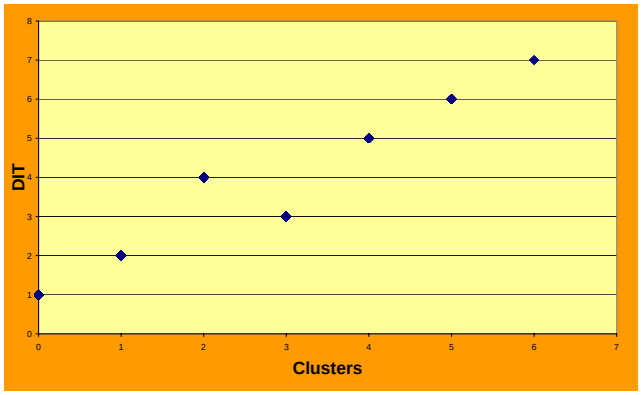

Figure 7: Depth of Inheritance Tree Clusters

4.2.5 *Depth of Inheritance Tree (DIT)*

Fig. 7 depicts the formed clusters of the classes of *JBoss* based on their depth of inheritance (DIT) attribute. It is obvious that as in many object-oriented systems inheritance is not really used a lot [18]. In *JBoss*, classes (clusters 5 and 6) with the highest DIT value are subclasses of the org.jboss.axis.message and org.jboss.varia.scheduler packages. The tree of inheritance leading to these classes has a depth of 6. A good example is class `org.jboss.varia.scheduler.XMLScheduleProvider` which has one of the deepest inheritance trees we found.

4.2.6 *Afferent Couplings (Ca):*

Fig. 8 depicts the formed clusters of the classes of JBoss based on their inward coupling. We can see here that on the 11th cluster there is the Logger class that depends on 2 only other classes, while 1200 other depend on it. That is why the maintenance engineer has to be very cautious when changing its interface and behaviour.

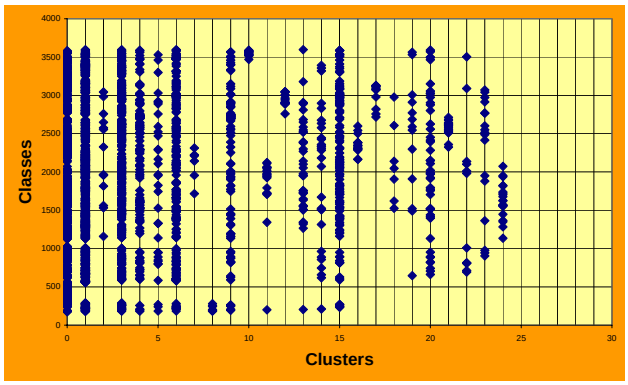

Figure 8: Afferent (Inward) Coupling (Ca) Clusters

4.2.7 *Number of Public Methods (NPM)*

Fig. 9 depicts the formed clusters of the classes of *JBoss* based on their number of public methods.

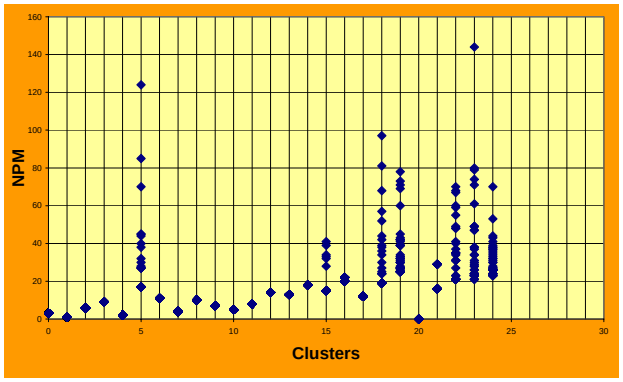

Figure 9: Number of Public Methods Clusters

Exposing too much of the class logic through public methods can be a sign of poorly designed code as it may need to split the class into two. However, there are cases where this is inevitable. For example, there are 144 public methods in class `org.jboss.resource.adapter.jdbc.WrappedResultSet`. This class implements a wrapper to access the results of a `sql` query using the java database connectivity `API`. Its internal logic is not complicated (almost all of its methods consist of one

or two lines), but it provides many public methods to access the result of the query in many ways.

4.3 Result Interpretation

As derived from the clustering (based on its structural characteristics), *JBoss* is an OO system that is built based on the JMX (Java Management Extension) infrastructure and supports also Aspect Oriented Programming middleware. We consider that packages which play the most significant role in formulating clusters would be the main packages of JBOSS. For example Cluster 3 is formed by classes that belong to `org.jboss.console` which provides a simple web interface for managing the *MBean* server and org.jboss.ejb3 which is the most basic package for an EJB implementation. We also discovered a very interesting cluster (Cluster 4) which comprises of packages that implement AOP (Aspect Oriented Programming) – based services.

On the other hand most of the classes of *JBoss* have very low complexity values and they do not expose too excessive either afferent or efferent couplings. They also have no children and they do not use inheritance very much. An interesting observation was that classes related to the implementation of (AOP) support have very low complexity (WMC) values. We also observed that most of the classes with high WMC and CBO values have member methods that their return types are not primitives.

The results presented in this section were derived without having prior knowledge of the *JBoss* OS Application Server. We would expect that domain experts could better support the evaluation of the proposed method.

5. CONCLUSIONS AND FUTURE WORK

This section presents conclusions drawn by evaluating the proposed framework and comparing it to similar ones. Directions for future work are also discussed.

5.1 Conclusions

The aim of this work was to facilitate maintenance engineers to comprehend the structure of a software system and assess its maintainability.

The first step towards that was to develop an extraction process that incorporates an xml representation of the code so that we can leverage the effort already put on this field by others. This goal was accomplished by using the *javaml* representation. 1,615,289 LOC or 4,714 Java code files were parsed in less than an hour in order to extract elements and metrics from *JBoss*' source code.

The application of this extraction method differentiates this work from [12] which is designed for procedural languages like COBOL and C. The proposed solution is also semi automated unlike [4] as the parsing engine extracts the data from the source code and stores them on a database.

The second step towards our goal to facilitate a maintenance engineer to comprehend a software system and assess its maintainability was the application of clustering for both recovering the structure of a software artifact and assessing its maintainability. That makes our work more complete than [4], [9], [12], [15], [23], which use data mining techniques only for structure recovering and [24] which uses clustering for predicting a software modules' fault proneness.

On the other hand our method analyses only the static dependencies of system's entities unlike [23] 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 [15] employs the Hierarchical Agglomerative Clustering (HAC) algorithm which automatically defines the number of the derived clusters.

5.2 Future Work

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

- **Systems' components clustering based on their dynamic dependencies**

It would be of great interest to attempt to evaluate the usefulness of analysing the dynamic dependencies of a software system's artefacts.

- **Integration of more data mining algorithms**

The proposed framework integrates the K-Means algorithm. 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. An example is the integration of hierarchical clustering algorithms that they do not need the user to define the number of the desired output clusters [8].

- **Enhance the Extraction Method**

The proposed method processes information derived only from Java source code files (*.java). It is of great interest to extract data from other OO languages like C++, C# and Borland Delphi. It can also be more flexible. For example, logic could be incorporated in java objects in order to compute software engineering metrics directly from the *javaml* files. These metrics could later be stored in the database using hibernate, just like the rest of the information.

- **Enrich the Input Model with more metrics and meta-metrics**

Based on this work we extracted elements and metrics from source code. The proposed Input Model can be enriched by employing more metrics except these in Chidamber and Kemerer Object Oriented suite [5]. We can also use meta-metrics in order to evaluate how accurate and right are the metrics we employ to assess a system's maintainability.

6. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for the constructive feedback on the original manuscript.

7. REFERENCES

[1] E. Arisholm, L.C. Briand, A. Foyen, "Dynamic Coupling Measurement for Object-Oriented Software", *IEEE Transactions Software Engineering*, Vol. 30, No. 8, 2004, pp. 491-506.

[2] G.J. Badros, "JavaML: A Markup Language for Java Source Code", *Computer Networks*, Vol. 33, No 1-6, 2000, pp 159-177.

[3] R.K. Bandi, V.K. Vaishnavi, D.E. Turk, "Predicting Maintenance Performance Using Object Oriented Design Complexity Metrics", *IEEE Transactions Software Engineering*, Vol. 29, No. 1, 2003, pp. 77-87.

[4] K. Chen, C. Tjortjis and P.J. Layzell, "A Method for Legacy Systems Maintenance by Mining Data Extracted from Source Code", *Case Studies IEEE 6th European Conf. Software Maintenance Reengineering* (CSMR 02), 2002, pp. 54-60.

[5] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. *IEEE Transactions Software Engineering*, Vol.20, No.6, 1994, pp.476-493.

[6] Al-Ekram, R., Kontogiannis, K., "An XML-Based Framework for Language Neutral Program Representation and Generic Analysis", *Proc. IEEE 9th European Conference Software Maintenance Reengineering* (CSMR 2005), 2005, pp. 42-51.

[7] U. M. Fayyad, G. Piatetsky-Shapiro and P. Smyth. "From Data Mining to Knowledge Discovery: An Overview", *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1996.

[8] A. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review", *ACM Computing Surveys*, Vol. 31, No. 3, 1999, pp.264 - 323.

[9] Y. Kanellopoulos, C. Tjortjis, "Data Mining Source Code to Facilitate Program Comprehension: Experiments on Clustering Data Retrieved from C++ Programs", *Proc. IEEE 12th Int'l Workshop Program Comprehension* (IWPC 2004), 2004, pp. 214-223.

[10] M.M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", *Proc. IEEE*, Vol. 68, No 9, 1980, pp. 1060 - 1076.

[11] W. Li, S. Henry, D. Kafura and R. Schulman, "Measuring Object Oriented Design", *J. Object– Oriented Programming*, Vol. 8, No. 4, 1995, pp. 48-55.

[12] 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), 1998, pp.16-23.

[13] T. Pearse and P. Oman, "Maintainability Measurements on Industrial Source Code Maintenance Activities", *Proc. Int'l Conference Software Maintenance*, (ICSM 95), 1995, pp. 295-303.

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

[15] D. Rousidis, C. Tjortjis, "Clustering Data Retrieved from Java Source Code to Support Software Maintenance: A Case Study", *Proc. IEEE 9th European Conf. Software Maintenance Reengineering* (CSMR 05), 2005, pp. 276-279.

[16] H.M. Sneed, "Applying Size Complexity and Quality Metrics to an Object Oriented Application", *Proc. European Software Control and Metrics Conference* - Software Certification Programme in Europe, 1999.

[17] I. Sommerville, *Software Engineering*, 6th ed., Harlow, Addison-Wesley, 2001.

[18] D. Spinellis. *Code Quality*, Addison-Wesley, 2006.

[19] J. Sutherland, "Business Objects in Corporate Information Systems", *ACM Computing Survey*, Vol. 27, 1995, pp 274-276.

[20] Y. Tan, V.S. Mookerjee, "Comparing Uniform and Flexible Policies for Software Maintenance and Replacement", *IEEE Transactions Software Engineering*, Vol. 31, No. 3, 2005, pp. 238-255.

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

[22] C. Tjortjis, L. Sinos and P.J. Layzell, "Facilitating Program Comprehension by Mining Association Rules from Source Code", *Proc. IEEE 11th Int'l Workshop Program Comprehension* (IWPC 03), 2003, pp. 125-132.

[23] C. Xiao, V. Tzerpos, "Software Clustering on Dynamic Dependencies", *Proc. IEEE 9th European Conf. Software Maintenance Reengineering* (CSMR 05), 2005, pp. 124-133.

[24] S. Zhong, T.M. Khoshgoftaar, and N. Seliya, "Analyzing Software Measurement Data with Clustering Techniques", *IEEE Intelligent Systems*, Vol. 19, No. 2, 2004, pp. 20-27.

[25] JBoss Community home page http://www.jboss.org

About the authors:

Yiannis Kanellopoulos is a PhD candidate at the School of Informatics, University of Manchester, U.K. His research interests are in the areas of Data Mining, Program Comprehension and Maintenance, and Software Engineering Quality.

Thimios Dimopulos is a Software Engineer at Deutsche Telekom Labs, Berlin, Germany. His research interests are in the areas of and Software Engineering and Maintenance.

Christos Tjortjis is a Lecturer at the School of Informatics, University of Manchester, and a part time Lecturer at the Nottingham Business School, Nottingham Trent University, U.K. His research interests are in the areas of data mining, software comprehension and maintenance where he has published widely.

Christos Makris is an Assistant Professor at the Department of Computer Engineering and Informatics, University of Patras, Greece. His research interests include Data Structures, Web Algorithmics, Computational Geometry, Data Bases and Information Retrieval.