Monitoring the Evolution of Software Systems Maintainability

P. Antonellis[1]D. Antoniou[1]Y. Kanellopoulos[1,2]C. Makris[1]E. Theodoridis[1]C. Tjortjis[2]N.Tsirakis[1]
1. University Of Patras, Department of Computer Engineering and Informatics, Greece
2. The University Of Manchester, School Of Computer Science, U.K.
{adonel, antonid, makri, theodori, tsirakis}@ceid.upatras.gr, Yiannis.Kanellopoulos@postgrad.manchester.ac.uk, christos.tjortjis@manchester.ac.uk

Abstract

This paper presents ongoing work on using data mining clustering to support the evaluation of software systems' maintainability. As input for our analysis we employ software measurement data extracted from Java source code.

1. Introduction

The scope of this work is to facilitate maintenance engineers to comprehend a software system and evaluate its evolution and maintainability. We attempt to address questions such as which classes are fault prone and more difficult to understand and maintain; how a system evolves from version to version and finally what are the dynamics of a system's classes through time.

For this reason we present a methodology which employs the clustering mining technique for the analysis of software measurement data. The k-Attractors algorithm which is tailored for software measurement data [4] was used for this purpose. The proposed methodology consists of two steps. At the first step, each version of a software system is analyzed separately in order to evaluate its maintainability. The second step comprises a macro-clustering analysis which investigates the derived clusters from all the versions of a software system. The aim of this step is to study a system's evolution by observing how clusters from each version grow up or shrink and how their centroids are moving in space from version to version.

2. Related work

Data mining [4], is the process which extracts implicit, previously unknown, and potentially useful information from data, by searching large volumes of them for patterns and by employing techniques such as classification, association rules mining, and clustering. More specifically, data mining has been previously used for identification of subsystems based on associations (ISA methodology) [2]. Sartipi et al. used it for architectural design recovery [11]. 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. Besides association rules, the clustering data mining technique has been used to support software maintenance and software systems knowledge discovery [13], [10]. The work in [10] proposes a methodology for grouping Java code elements together, according to their similarity and focuses on achieving a high level system understanding.

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 [9]. For C programs, functions were used as entities, and attributes were 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 [8]. An approach for the evaluation of clustering in dynamic dependencies is presented in [14]. The scope of this solution is to evaluate the usefulness of providing dynamic dependencies as input to software clustering algorithms. Additionally, Clustering over a Module Dependency Graph (MDG) [6] uses a collection of algorithms which facilitate the automatic recovery of the modular structure of a software system from its source code. This 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.

Moreover, in [5] an approach that examines the evolution of code stored in source control repositories is presented. This technique identifies *Change Clusters*, which can help managers to classify different code change activities as either maintenance or new development. On the other hand, [12] analyzes whether some change coupling between source code entities is significant or only minor textual adjustments have been checked in; in order to reflect the changes to the source code entities. An approach for analyzing and classifying change types based on code revisions has been developed.

In addition, Beyer and Noack in [3] presented a method based on clustering software artifacts, in order to organize software systems into subsystems and by this way make changes less expensive and less error prone. Towards the same goal of comprehending large software systems by creating abstractions of the software system's structure, Mitchell and Mancoridis in [8] presented the Bunch clustering system. In this work, clustering is implemented by search techniques and is performed on graphs that represent the system's structure. The subsystems are generated by partitioning a graph of entities and relations. Another approach in the context of software clustering is the Limbo algorithm, introduced by Tzerpos and Andritsos [1]. This scalable hierarchical algorithm focuses on minimizing the information loss when clustering a system, by applying weighting schemes that reflect the importance of each component.

Clustering algorithms are also used by Mancoridis et al. [7] in order to support the automatic recovery of the modular structure of a software system from its source code. The algorithms selected in this case are traditional hill-climbing and genetic algorithms. Towards program comprehension, a crucial step is detecting important classes of the system, since they implement the most basic and high level actions. Zaidman et al [15] introduced four static web-mining and coupling metrics in order to identify such classes and generally analyze a software system.

The work presented in this paper differs from the literature discussed above in means of performing clustering on the software measurement data, aiming at comprehending a software system and assessing its maintainability. More specifically, instead of applying clustering algorithms on graphs or directly on the source code, we employ the k-Attractors clustering algorithm on metrics that reflect the most important design aspects of a software system concerning its quality and maintainability. We employ a two-steps clustering analysis in order to provide a quick and rough grasp of a software system and depict its evolution by from version to version.

3. Clustering Analysis

3.1. Objectives

The primary objective of the proposed clustering methodology is to provide a general but illuminating view of a software system that may lead engineers to useful conclusions concerning its maintainability. This data mining technique is useful for Similarity/Dissimilarity analysis; in other words it analyzes what data points are close to each other in a given dataset.

In the domain of software measurement data analysis for maintenance purposes, clustering can be formalized by the following function signature:

$$cluster : v^{M\times U} \rightarrow v^{M\times U\times G}$$

where:

- U: units in the system
- M: measurements performed on units
- G: groups of units

Thus, a matrix of measurement values is partitioned into a list of such matrices. Each matrix represents a cluster of items that are similar in terms of their measurement values.

In order to extract useful information for the maintenance engineers through the clustering analysis, it is very interesting to observe the form of each cluster over time. How each cluster grows up or shrinks and how its median is moving in space. In order to achieve that, a first task to be performed is the identification of each cluster in each version. An approach is to combine all the data sets (the data points corresponding to classes) into a large data set. Each point is marked with a different color in order to disentangle them later on. If we apply a clustering algorithm in this data set (k-Attractors in our case) we can make the assumption that a cluster will encompass data items of the same cluster through the versions. In each of these clusters will exist the same data items with different color and thus from different version. We can verify this by inventing an inner metric: the percentage of data points that exist in the cluster with all the possible colors (or a percentage respectively of them, for example 3 out of 5 of the versions). There are several ways to exploit this clustering by automated methods: we can trace the data items that have escaped the cluster and examine if they have gone to a better or a worse cluster, by examining in each cluster the sub-clusters, each one with a different color, and how their centroid is moving and the portion of their spatial overlap. By these panoramic observations, the sequence of centroids and proportion of the overlap, we can see if the data items of the corresponding cluster evolve to better or a worse state.

In order to quantify the cluster changes we define a metric *m(i)* of each cluster *i* which expresses how many variations, data items (from version to version) exist in the same cluster at the same time, and thus in the same quality space. This metric is expressed by the following formula:

$$m(i) = \sum_{\forall x \in ci} \sum_{j=1}^{n} ocj(x) \Big/ \sum_{j=1}^{j} pj \qquad (1)$$

Where n is the number of the formed clusters, $occ(x_i)$ is the number of occurrences of each data item x in cluster i, and p_i is the cardinality (population) of cluster i.

3.2. k-Attractors Algorithm

In the case of software maintainability evaluation, clustering produces overviews of systems by creating mutually exclusive groups of classes, member data or methods, according to their similarities in terms of technical (source code) measurements [16]. This helps reducing the time required to understand and evaluate the overall system. Another contribution of clustering is that it helps discovering programming patterns and "unusual" or outlier cases which may require attention. For this purpose the k-Attractors algorithm was employed which is tailored for numerical data such as measurements from source code [4]. The main characteristics of k- Attractors are:

- It defines the desired number of clusters (i.e. the number of k), without user intervention.
- It locates the initial attractors of cluster centers with great precision.
- It measures similarity based on a composite metric that combines the Hamming distance and the inner product of transactions and clusters' attractors.

The k-Attractors algorithm employs the maximal frequent itemset discovery and partitioning in order to define the number of desired clusters and the initial attractors of the centers of these clusters. The intuition is that a frequent itemset in the case of software metrics is a set of measurements that occur together in a minimum part of a software system's classes. Classes with similar measurements are expected to be on the same cluster. The term attractor is used instead of centroid, as it is not determined randomly, but by its frequency in the whole population of a software system's classes.

4. Evaluation –Case Study

The evaluation of the proposed methodology involved the study of Apache Geronimo Application Server. It is a fully certified J2EE 1.4 platform for developing and deploying Enterprise Java applications, Web applications and portals. Three publicly available versions of Apache Geronimo were evaluated employing a set of software evaluation metrics and their analysis using the k-Attractors clustering algorithm.

In the first step of the analysis we created overviews for each version of Apache in order to have an indication for their maintainability status. Then by studying the formed clusters for each version, we discovered classes which were fault prone. Those classes were members of the outlier clusters and examples are `CdrOutputStream` and `CdrInputStream`. These classes are used for streaming objects in Corba Common Data Representation format. These classes are used fairly widely within the application server, for, among others, serializing non-primitive data structures, hence the high complexity values. They should be of interest to the maintenance engineers, since they are at Geronimo's core and widely used, so for maintainability and runtime performance they will be important classes. Classes `KernelManagementHelper` and `MockGBean` can also be interesting from a maintenance engineer's perspective.

In the second step, the macro-clustering analysis, we traced classes that their quality was either degraded or upgraded. Such classes are `RefContext` and `AbstractWebModuleBuilder`

5. Future Work

Our findings indicate that the proposed methodology has considerable merit in facilitating maintenance engineers to monitor how a system's maintainability evolves. On the other hand though, it lacks the ability to predict the maintainability of an upcoming version of a system. Another data mining technique with prediction capabilities (such as classification) could be additionally employed in order to enhance our methodology.

Moreover and apart from this, we consider the following various alternatives in order to further develop the proposed methodology: *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. *Employ an alternative approach for monitoring cluster changes from version to version*. Another approach for monitoring cluster changes is to perform the clustering procedure for each one of the versions. We use all these clusters (each one with a different color according to the version that belongs) in a second clustering phase, using the corresponding centroids, in order to produce clusters of clusters in a hierarchical way. We can assume that each one of the level two clusters consists of the same cluster of data item through versions.

6. Conclusions

In this research work, the development of a methodology based on the clustering data mining technique was presented. It consists of two steps:

i. a separate clustering step for every version of a system to assist software system's evaluation in means of maintainability.

ii. a macro-clustering analysis in order to study the system's dynamics from version to version.

The scope of the proposed methodology is to facilitate maintenance engineers to identify classes which are fault prone and more difficult to understand and maintain as well as to study the evolution of a system from version to version, and its classes' dynamics. We chose to employ the k-Attractors clustering algorithm as it is tailored for the analysis software measurement data.

Our work is different than [7], which employs clustering in order to produce a high-level organization of the source code. Additionally, instead of applying clustering algorithms directly on the source code [7], we clustered software metrics that reflect the most important aspects of a system concerning its quality and maintainability. Moreover the study of the classes' evolution through versions differentiates this work from [15] which only detects the most important classes on a single version of the system.

Acknowledgements

This research work has been partially supported by the Greek General Secretariat for Research and Technology (GSRT) and Dynacomp S.A. within the program "P.E.P. of Western Greece Act 3.4".

References

[1] Andritsos, P. and Tzerpos, V. "Information-Theoretic Software Clustering". I*EEE Trans. Software Eng.* vol. 31(2), 2005, pp. 150-165

[2] Bandi, R. K., Vaishnavi, V. K. and Turk, D. E. "Predicting Maintenance Performance Using Object Oriented Design Complexity Metrics", *IEEE Transactions on Software Engineering*, vol. 29(1), January 2003, pp. 77-87.

[3] Beyer, D. and Noack, A. "Clustering software artifacts based on frequent common changes". *In Proc. IWPC*, IEEE, 2005, pp. 259–268.

[4] Kanellopoulos Y., Antonellis P. Tjortjis C., Makris C., "k-Attractors, A Clustering Algorithm for Software Measurement Data Analysis", In Proceedings of IEEE 19th International Conference on Tools for Artificial Intelligence (ICTAI 2007), IEEE Computer Society Press 2007

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

[5] Lawrie, D. J., Feild, H. and Binkley, D. "Leveraged Quality Assessment using Information Retrieval Techniques," *14th IEEE International Conference on Program Comprehension (ICPC'06)*, 2006, pp. 149-158.

[6] Mancoridis, S., Mitchell, B.S., Chen, Y. and Gansner, E.R. "Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures", *Proc. Int'l Conf. Software Maintenance (ICSM 99)*, 1999, pp.50-59.

[7] Mancoridis, S., Mitchell, B. S., Rorres, C. "Using Automatic Clustering to Produce High-Level System Organizations of Source Code", (1998) *IEEE Proceedings of the 1998 Int. Workshop on Program Understanding (IWPC'98)*, 1998

[8] Mitchell, B. S. and Mancoridis, S. "On the Automatic Modularization of Software Systems Using the Bunch Tool". *IEEE Trans. Software Eng.*, vol. 32(3), 2006, pp. 193-208

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

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

[11] Sartipi, K., Kontogiannis, K. and Mavaddat, F. "Architectural Design Recovery Using Data Mining Techniques", *Proc. 2nd European Working Conf. Software Maintenance Reengineering (CSMR 00)*, 2000, pp. 129-140.

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

[13] Tzerpos, V. and Holt, R. "Software Botryology: Automatic Clustering of Software Systems", *Proc. 9th Int'l Workshop Database Expert Systems Applications (DEXA 98)*, 1998, pp. 811-818.

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

[15] Zaidman, A., Du Bois, B. and Demeyer, S. "How Webmining and Coupling Metrics Improve Early Program Comprehension." *ICPC*, 2006, pp. 74-78

[16] 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.