INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

AN INTERACTIVE APPROACH FOR SOFTWARE SYSTEM RE-MODULARIZATION BASED ON CHRONICLE DATA

Rajalakshmi M[1], Bright Gee Varghese. R[2]

[1] *PG Student, Department of Computer Science and Engineering, Karunya University, Tamilnadu, India, itisrebecca@gmail.com*
[2] *Assistant Professor, Department of Computer Science and Engineering, Karunya University, Tamilnadu, India, brightfsona@yahoo.co.in*

Abstract

Re-modularization of software systems is a key technique used in testing and maintenance phase of the software development lifecycle. This process helps the developer in reviewing the modules and the interaction between those modules of the developed software systems. Re-modularization process also allows the developer to make the software system more efficient by adding additional features according to their future requirement and enhancement. The use of Interactive Genetic Algorithms is to integrate developer's knowledge in a re-modularization task. Specifically, the proposed algorithm uses a fitness composed of automatically-evaluated factors accounting for the modularization quality achieved by the solution and a human-evaluated factor, penalizing cases where the way re-modularization places components into modules is considered meaningless by the developer. An observation regarding these applications is that they generate a large amount of intermediate data, and this abundant information is thrown away after the processing finish. We propose a module aware of cache framework for software re-modularization, which is called Mache. In Mache, tasks submit their intermediate results to the cache manager. A task, before initiating its execution, queries the cache manager for potential matched processing results, which could accelerate its execution or even completely saves the execution. The proposed approach has been evaluated to re-modularize two software systems, SMOS and GESA. The obtained results indicate that IGA is able to produce solutions and Mache mechanism that, from a developer's perspective, are more meaningful than those generated using the full-automated GA.

Keywords: Software Re-Modularization, Genetic Algorithm, Cache Management.

1. Introduction

Software systems in general consist of modules and methods that interact with each other in order to accomplish the purpose for which those systems are actually developed. The unchanging fact is that these developed software systems are exposed to modifications or changes. This may be done in the view of detecting and correcting errors or in need of improving the efficiency of software systems by introducing additional features based on their future requirements. This is termed to be the re-modularization process of the software systems. (Fowler .M et al. 1999) says the modifications made in the developed software may however reduce the cohesiveness of the modules and increase the coupling between various modules and thus making the resultant software system to be harder to maintain and possibly be more fault-prone.

To overcome this situation (i.e.) to improve cohesion with the module in the software system and to reduce coupling between various modules within the software systems, various re-modularization techniques are used.

Hierarchical clustering based techniques, automated clustering approaches and clustering using Genetic Algorithm (GA) are some ways in which re-modularization process of software systems can be done. The resultant package or modules of the software system will possess high cohesiveness and low coupling characteristics.

Software module clustering is the problem of automatically organizing software units into modules to improve program structure. There has been a great deal of recent interest in search based formulations of this problem, in which module boundaries are identified by automated search, guided by a fitness function that captures the twin objectives of high cohesion and low coupling in a single objective fitness function. Finally we move for the technique of software system re-modularization, which involves the designer to know all about the system in order to re-modularize it efficiently.

We also utilize the intermediate data obtained during the re-modularization process. This is implemented based on the MapReduce framework. (Yaxiong Zhao et al. 2014) says an observation of the MapReduce framework is that the framework generates a large amount of intermediate data. Such abundant information is thrown away after the tasks finish, because MapReduce is unable to utilize them. In this paper, we propose Mache, a cache framework for big-data applications. In Mache, tasks submit their intermediate results to the cache manager. A task queries the cache manager before executing the actual computing work.

2. Proposed Method

The real world is not static: new laws are created, concurrent offer new functionalities, users have renewed expectation toward what a computer should offer them, memory constraints are added, etc. As a result, software systems must be continuously updated or face the risk of becoming gradually out-dated and irrelevant.

The single-objective GA uses as fitness function (to be maximized) the MQ metric, while the multi-objective GA implemented as Non- Dominating Sorting Genetic Algorithm (NSGA-II)considers five different objectives, related to maximizing MQ, intra-cluster connectivity and number of clusters, and minimizing the inter-cluster connectivity and the number of isolated clusters(Deb k et al2001.

The basic idea of the IGA is to periodically add a constraint to the GA such that some specific components shall be put in a given cluster among those created so far. Thus, the IGA evolves exactly as the non-interactive GA. Then, every nGens generations, the best individual is selected and shown to the developer. Then, the developer analyzes the proposed solutions and provides feedback indicating that certain components shall be moved from a cluster to another.

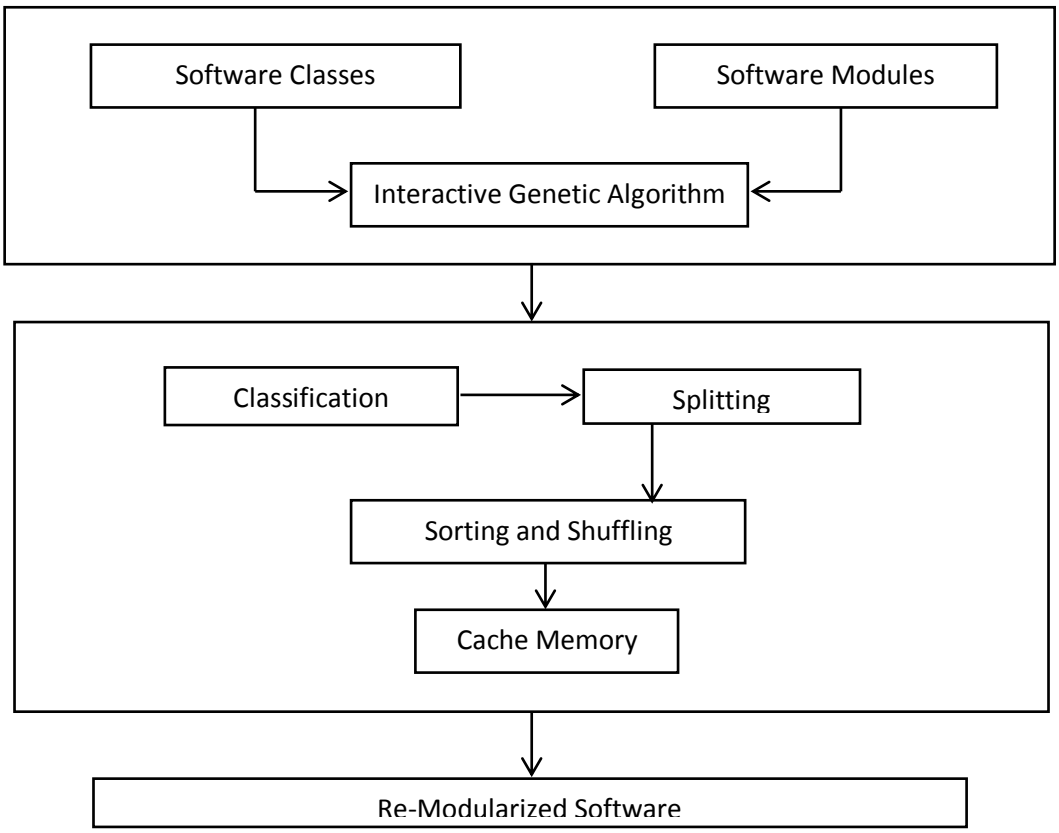

Figure 1: Architecture Diagram

The proposed method includes Interactive Genetic Algorithm (IGA) along with the chronicle data that is obtained during the process of re-modularization of the software system. The following are the steps involved in the proposed method: (i) Classification, (ii) Splitting, (iii) Cache Manager, (iv) Sorting and Shuffling, (v) Software Modularization.

2.1 Classification

Classification is a data technique used to predict group membership for data instances. The classification methodology used for Interactive genetic approach is to classify the modularization of class and modules. The property of iteratively assigning records to a cluster, calculating a measure, and re-assigning modules to classify until the calculated measures don't change much indicating that the process has converged to stable segments. Class and modules within a cluster are more similar to each other and more different from records that are in other classification. Depending on the particular implementation, there are a variety of measures of similarity that are chromosome values, fitness function but the overall goal is the approach has to converge to groups of related modules and class.

2.2. Splitting

Data splitting can be made even more effective by periodically retrieving and recombining the parts, and then splitting the data in a different way among different servers and using a different encryption key. Thus, even if a hacker makes progress towards obtaining split data, chances are that the data will have been reorganized before the hacker manages to obtain all the necessary components. It is based upon the number of classified records in the software modularization methodologies.

2.3. Cache Manager

A cache memory is a relatively smaller, easy-access, temporary memory that stores copies of data from frequently accessed memory locations for quick access by the processor. When the processor needs to read from a location in the main memory, it first checks whether a copy of that data is in the cache; if it is, the processor immediately uses this copy, which is much faster. The splitted data is stored into the cache memory.

2.4. Sorting and Shuffling

Sorting and shuffling makes the guarantee that the input to every reducer is sorted by key. The process by which the system performs the sort—and transfers the map outputs to the reducers as inputs—is known as the shuffle. The shuffle is an area of the codebase where refinements and improvements are continually being made, so the following description necessarily conceals many.

2.5. Software Modularization

A module is a software component or part of a program that contains one or more routines. One or more independently developed modules make up a program. An enterprise-level software application may contain several different modules, and each module serves unique and separate business operations. Software applications include many different tasks and processes that cohesively serve all paradigms within a complete business solution. Early software versions were gradually built from an original and basic level, and development teams did not yet have the ability to use prewritten code.

3. Result Analysis

When analysing results, we compare the ability of GAs and IGAs to reach a fair trade-off between the optimization of some quality metrics (that is the main objective of GAs applied to software re-modularization) and the closeness of the proposed partitions to an authoritative one. Note that we use the original design of the object systems as authoritative partition. This choice is justified by the good modularization quality of the

object systems that have been previously used as gold standard to assess other re-modularization approaches. On the one hand, to analyse the impact of provided feedback from the quality metrics point-of-view, we use four quality metrics previously adopted by (Praditwong et al., 2011) namely MQ, intra-edges, inter-edges, and number of isolated clusters. On the other hand, to measure the meaningfulness of the modularizations proposed by the experimented algorithms, we compute the MoJo

$$M_oJ_oFM(A,B)= 100- \left(\frac{mno(A,B)}{\max(mno(\forall A,B))} * 100 \right) \quad (1)$$

where mno(A; B) is the minimum number of Move or Join operations one needs to perform in order to transform the partition A into B, and max(mno(8 A; B) is the maximum possible distance of any partition A from the gold standard partition B.

Thus, MoJoF M returns 0 if a clustering algorithm produces the farthest partition away from the gold standard; it returns 100 if a clustering algorithm produces exactly the gold standard. We also statistically analyse whether the results achieved by different algorithms (interactive and not) significantly differ in terms of quality metrics or authoritativeness of the modularizations. In particular, the values of all the employed metrics (e.g., MQ) achieved in the 30 runs by two algorithms are statistically compared using the Mann-Whitney test.

Apart from this the use of chronicle data also increases the classification accuracy compared with those done with Simple interactive genetic algorithm(IGA).

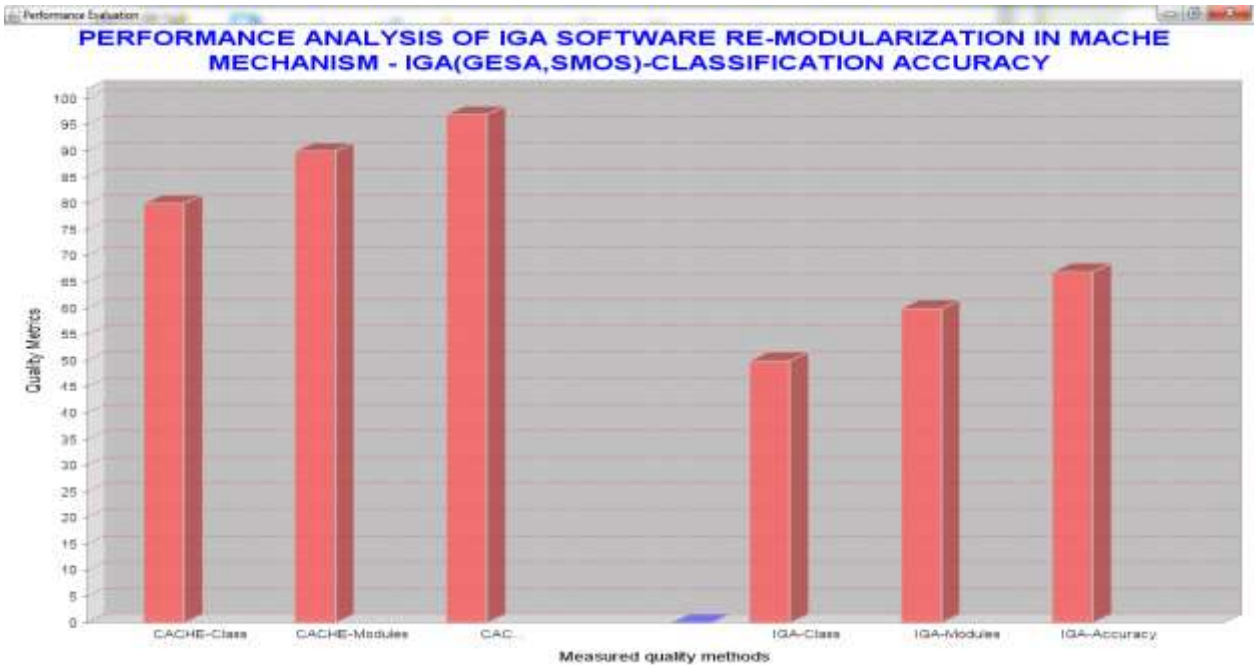

Figure 2: Comparison of Mache mechanism with IGA in terms of classification accuracy

4. Conclusion

The use of IGAs to integrate developers' knowledge during software re-modularization activities has been proposed in this paper. Both single- and multi-objective IGAs are implemented and experimented and their performances are compared with those achieved by their non-interactive counterparts. The achieved results show that the IGAs are able to propose re-modularizations are more meaningful from a developer's point-of-view and are better in terms of modularization quality, with respect to those proposed by the non-interactive Gas. Also the use of chronicle data accelerates the execution of the process by using some of the already classified modules. This will surely reduce the time taken for the re-modularization of software systems.

REFERENCES

Yaxiong Zhao, 2013, Dache: A data aware caching for big-data applications using the MapReduce framework, In proceedings of IEEE, pp.35-39.

Gabriele Bavota, 2012, Putting the developer in the loop: an interactive GA for software re-modularization, In proceedings of 4th International Symposium, SSBSE pp.75-89.

Praditwong K, Harman M, Yao X, 2011: Software module clustering as a multiobjective search problem,In proceedings of IEEE, pp.264-282.

Tonella P, Susi A, Palma F, 2010: Using interactive GA for requirements prioritization. In: SSBSE. pp. 57-66.

J. Dean and S. Ghemawat, Mapreduce, 2008: Simplified data processing on large clusters, Commun. of ACM, vol. 51, no. 1, pp. 107-113.

F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable, 2006: A distributed storage system for structured data, in Proc. of OSDI'2006, Berkeley, CA, USA.

Deb K, 2001: Multi-Objective Optimization Using Evolutionary Algorithms Wiley, 581 pages.

Fowler M, Beck K, Brant J, Opdyke W, Roberts D, 1999: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.

A Brief Author Biography

Rajalakshmi M is pursuing M.Tech (Computer Science and Engineering) degree from Karunya University, Tamil Nadu, India. She received her B.Tech (Information Technology) degree from RVS College of Engineering and Technology, Tamil Nadu, India in 2012.

Bright Gee Varghese R completed his B.E (Computer Science and Engineering) from Sun College of Engineering and Technology, Nagercoil, Tamil Nadu, India in 2003. He started his career as Lecturer in Sun College of Engineering and Technology, Nagercoil, from June 2003. He completed his M.E from Vinayaka Mission university, Salem, Tamil Nadu in 2011.Currently he is working as an Assistant Professor in Computer Science Department in Karunya University and pursuing part time Ph.D in Karunya University, Tamil Nadu, India. His main research area is Software Engineering.