Research Paper Engineering

Software System Re-Modularization using Interactive Genetic Algorithm

RAJALAKSHMI.M

POST-GRADUATE SCHOLATUDENT, DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, KARUNYA UNIVERSITY, INDIA.

ABSTRACT

Software is naturally subject to modifications aiming at fixing bugs or introducing new features. Very often, such activities are conducted within a very limited time frame, and with a limited availability of software design documentation. Change activities tend to erode the original design of the system. Such design erosion mirrors a reduction of the cohesiveness of a module, the increment of the coupling between various modules and, therefore, makes the systems harder to be maintained or possibly more fault-prone. For this reason, various automatic approaches, aimed at supporting source code re-modularization, have been proposed. The Interactive Genetic Algorithms (IGAs) is used to integrate developer's knowledge in a re-modularization process of software in the form of feedback. The obtained results indicate that IGA is able to produce solutions that, from a developer's perspective, are more meaningful than those generated using the full-automated GA. While keeping feedback into account, the approach does not sacrifice the modularization quality.

KEYWORDS Software Re-Modularization, Genetic Algorithm, Interactive Genetic Algorithm.

1. INTRODUCTION

Software in general is subject to change activities aiming at fixing bugs or introducing new features. Very often, such activities are conducted within a very limited time frame, and with a limited availability of software design documentation. Change activities tend to \erode" the original design of the system. Such design erosion mirrors a reduction of the cohesiveness of a module, the increment of the coupling between various modules and, therefore, makes the system harder to be maintained or, possibly, more fault-prone. To improve maintainability, software systems are usually organized into subsystems using the constructs of packages or modules. However, during software evolution the structure of the system undergoes continuous modifications, drifting away from its original design, often reducing its quality. As valuable software gets older, the maintenance becomes more and more an issue. Software engineers face systems, were a minor change in some part can have unexpected results in an apparently unrelated part. The reverse engineering community aims at providing solutions to help software engineers understand, restructure or migrate old software towards more modern architecture and/or language.

A key method to do so is clustering. It is used to gather software components into modules significant to the software engineers. But clustering is a research domain in itself and a very sophisticated one. There are many different methods to answer the different needs: Size of the data, type of data, a priori knowledge of the result expected, etc. Reverse engineering is yet a young research domain, with unclear goals and more ad-hoc methods than well establish methodologies. In this context, clustering has often been used with no deep understanding of all the issues involved.

1.2 PROBLEM STATEMENT

While automatic re-modularization approaches proved to be very effective to increase cohesiveness and reduce coupling of software modules, they do not take into account developers' knowledge when deciding to group together (or not) certain components. This is because the developer is aware that, in future releases, such a function will strongly interact with the rest of the module. Similarly, a developer may decide that two functions must be placed in two different modules even if they communicate. This is because the two functions have different responsibilities and are used to manage semantically different parts of the system.

1.3 OBJECTIVE

To integrate developer's knowledge in the software re-modularization process using Interactive Genetic Algorithms.

1.4 PROPOSED METHOD

The basic idea proposed in this paper is that initially the package with classes and methods are given as input to the system for re-modularization process. Then Coupling (i.e.) the inter-module links are calculated based upon the relationships that exist between the classes. Then Cohesion (i.e.) the intra-module links are identified based on the relationships that exist within the classes. Modularization Quality (MQ) metric is calculated by taking an average of both inter-module and intra-module links of the classes. Depending on the MQ values a fitness function is used the range of different clusters in which the classified classes will be grouped in later.

2. PROPOSED METHOD

While automatic re-modularization approaches proved to be very effective to increase cohesiveness and reduce coupling of software modules, they do not take into account developers' knowledge when deciding to group together (or not) certain components. In the past, some authors proposed approaches to account for developers' knowledge in software re-modularization. However, such approaches assume the availability of a whole set of constraints before the re-modularization starts. This is often difficult to be achieved, especially for very large systems.

This paper proposes the use of Interactive Genetic Algorithms (IGAs) to integrate, into a re-modularization approach, a mechanism allowing developers to feed-back automatically produced re-modularizations. IGAs are a variant of Genetic Algorithms (GAs) in which the fitness function is partially or entirely evaluated by a human while the GA evolves.

2.1 BLOCK DIAGRAM

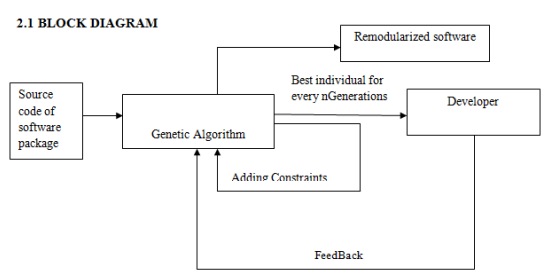

Fig 2.1. Architecture diagram for clustering and providing feedback

2.2 THE PROPOSED INTERACTIVE GENETIC ALGORITHMS

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.

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.

After enacting the developer's indications, a new GA population is created from such a best solution, and then the GA evolves for further nGens generations, keeping into account the provided constraints. One crucial point is choosing how to guide the developer to provide feedback. In principle, one could ask developers any possible kind of feedback. For this reason, we propose to guide developers in providing feedback, by means of two different kinds of IGAs. The first one referred to as R-IGA and described by Algorithm 1 takes the best solution produced by the GA, randomly selects two components (from the same cluster or from different clusters), and then asks the developer whether, in the new solutions to be generated, such components must be placed in the same cluster (i.e., stay together) or whether they should be kept separated.

3. IMPLEMENTATION

3.1 MODULE DESCRIPTION

There are mainly four modules:

- Cohesion and Coupling
- Modularization Quality (MQ)
- Crossover and Mutation
- Feedback

3.1.1 COHESION AND COUPLING

In this module the source package is selected the relationships between the classes are identified using the calling function. The Intra-Connectivity (A_i) is the measure of the density of connections between the nodes of a single cluster Doval D et al.(1999). This can be calculated using the following formula,

$$A_i = \mu_i / N_i^2 \quad \textbf{(3.1)}$$

where μ_i is the intra-edge dependencies and N is the number of components. Inter-Connectivity ($E_{i,j}$) is the measure of the connectivity between distinct clusters Doval D et al. (1999). This is can be calculated using the formula,

$$E_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \frac{\varepsilon_{ij}}{2N_i N_j} & \text{if } i \neq j \end{cases} \quad \textbf{(3.2)}$$

where $\varepsilon_{i,j}$ is the inter dependencies between the two clusters. A class and the methods used in that class are grouped under the class name and is stored in the database for later usage.

3.1.2 MODULARIZATION QUALITY (MQ)

In this paper, genetic algorithm for clustering is applied based on a single objective function. In this module the modularization quality (MQ) is taken as the single objective function Bavota et al. (2012). The modularization quality (MQ) is calculated by using the following formula,

$$MQ = \begin{cases} \left(\frac{1}{k}\sum_{i=1}^{k} A_i\right) - \frac{1}{\frac{k(k-1)}{2}} \sum_{i,j=1}^{k} E_{i,j}\ ) & if\ k > 1 \\ A_1 & if\ k=1 \end{cases} \quad \textbf{(3.3)}$$

The MQ and the probability function for each chromosome is calculated and the chromosome with the highest MQ value is taken for the next generation.

3.1.3 CROSSOVER AND MUTATION

Upon the population of chromosomes apply the crossover and mutation operations in order to get the population for next generation. Crossover can be implemented by using two parent chromosomes. There are many types of crossover operations. The single-point crossover is used in this paper. A single datapoint is fixed based on the length of the chromosome and the parent chromosomes are swapped after the datapoint, thus yielding two new offsprings or child chromosomes. The fitness for the newly created chromosomes are calculated and the best fitted chromosome will be taken for the next generation. Mutation is implemented in order to maintain the genetic diversity of one generation of population of chromosomes to the next generation.

Then the best chromosomes in the newly generated population are selected and again a new generation of population of chromosomes is generated by performing crossover and mutation operations. This process will come to an end only when the population of chromosomes are converged to the best solution of chromosomes.

3.1.4 FEEDBACK GENERATION

Two different kinds of IGAs have been proposed to guide developers in providing feedback. The first one referred to as R-IGA takes the best solution produced by the GA, randomly selects two components (from the same cluster or from different clusters), and then asks the developer whether, in the new solutions to be generated, such components must be placed in the same cluster (i.e., stay together) or whether they should be kept separated. As the algorithm indicates, every nGens generations the developer is asked to provide feedback about a number nFeedback of component pairs from the best solution (in terms of MQ) contained in the current population. The feedback can either be

- ci and cj shall stay together or
- ci and cj shall be kept separate

After feedback is provided, the solution is repaired by enforcing the constraints, e.g., by randomly moving one of ci and cj away if the constraint tells that they shall be kept separate. After all nFeedback have been provided, a new population is created by randomly mutating such a repaired solution. Then, the GA starts again. When creating the new population and when evolving it, the GA shall ensure that the new produced solutions meet the feedback collected so far.

The second IGA we propose called IC-IGA which stands for Isolated Cluster –Interactive Genetic Algorithm, focuses on specific parts of the re-modularization produced by the GA. Among others, very small clusters should be subject to manual changes by the developer. In fact, automatic re-modularization approaches often tend to create a large number of many small clusters that seldom reflect the actual or desired system decomposition. For this reason, the second variant of our IGA asks feed-back on the nClusters smallest clusters in the best solution (in terms of MQ). Then, for each of these clusters, if it is an isolated cluster (i.e., composed of one component only), the developer is asked to specify a different cluster where the isolated component must be placed while for not isolated clusters the developer is asked to specify for each pair of com-

ponents whether they must stay together or not.

CONCLUSION

The underlying idea of approaches that uses IGA is to group together in module highly cohesive source code components, where the cohesiveness is measured in terms of intra-module links and to reduce the coupling between modules, where the coupling is measured in terms of inter-module dependencies. Such approaches use various techniques, such as clustering, formal concept analysis or search-based optimization techniques to and (near) optimal solutions for such objectives, e.g., cohesion and coupling. While automatic re-modularization approaches proved to be very effective to increase cohesiveness and reduce coupling of software modules. 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, IGAs is applicable with a limited number of feedbacks, and thus, require a relatively limited effort to the developer.

REFERENCES

Doval, D., Mancoridis, S. & Mitchell, B.S : Automatic clustering of software systems using a | genetic algorithm. In: STEP. pp. 73-82. (1999). | Gabriele Bavota, Filomena Carnevale, Andrea De Lucia, Massimiliano Di Penta & Rocco | Oliveto : Putting the developer in the loop: an interactive GA for software re- | modularization, In proceedings of 4th International Symposium, SSBSE pp.75-89. (2012). | Mathew Hall, Neil Walkinshaw & Phil McMinn : Supervised Software Modularisation, | IWPC. pp. 194-208. (2012). | Mitchell, B.S & Mancoridis S : On the automatic modularization of software systems | using the bunch tool. IEEE TSE 32(3), 193-208 (2006) | Praditwong, K., Harman, M. & Yao, X : Software module clustering as a multi-objective | search problem. IEEE TSE 37(2), 264-282 (2011). |