Acta Cybernetica **17** (2005) 273–287.

An Approach based on Genetic Algorithms for Clustering Classes in Components

Dan Laurenţiu Jişa[*]

Abstract

The goal of this work is to create a model that allows identification of the software components (or subsystems according to the unified process terminology) based on the design models, or more exactly, based on the classes diagrams (for the static aspects) and on the interaction diagrams (for the dynamic aspects). The work also presents a genetic algorithm used for the clustering of classes into modules.

Keywords: object-oriented metrics, UML, unified process, genetic algorithms, clustering algorithms.

1 Introduction

The components based development (CBD) process tends to revolutionize the process of applications development, the usage of components opens the possibility of assembling applications from predefined building blocks. A component has to assure a better management of the complex applications, to lead to a decrease of the development costs and to an increase of the flexibility. These goals could be reached through a proper quality level of software components. One of the most important quality indicators for a software module is the reusability. The term "reuse" refers to the utilization of a software product, earlier developed, in a new project or software application. This reutilization can take many aspects:

- any physical component or program code;
- any product, result of a software development process: tools, documentation, models (requirements, analysis or design) etc.;
- any knowledge gains in the earlier projects.

The reutilisation, in a development process, of a software component that already exists, instead of the development of a new one, represents an activity that leads to an increase of product quality and, also, to a decrease of the development costs. These benefits are the consequence of two reasons:

[*]PhD student, Department of Business Informatics, Academy of Economic Studies, Calea Dorobanţilor 15-17, Sector 1, 71131-Bucharest, Romania, e-mail:`dan.jisa@estwest.ro`

- the development of new components (modules) is expensive;
- the reused components (modules) are considered as being very well tested, and their maintenance is not very expensive.

From the point of view of the software metrics, the reuse of software products provides many interesting attributes to estimate. For example, these attributes can be: the quantity of reused code in a software product, the reutilisation cost, the reusability of a certain product, module or class etc.

The measurement of the degree to which a software product can be reused, usually takes the name of reusability. Generally, the reusability is defined as the extent to which a software product can be used in other applications.

Often, the goal of reusability measurements is to identify components in large applications. In addition, the goal is often to automatically extract the components. For the domain experts it will be still necessary to assess the modules in order to identify the possible modifications. However, some results show that through the automation of the identification process, the quantity of code which has to be examined by the domain experts will be reduced.

Another important objective is to design and to write a reusable code. When the software development process is mature enough and the software metrics become part of the process carried on in an organisation, then the design metrics become more important for two reasons:

- the measurement process can be automated, so that to have a feedback for the designers;
- the measurement of the reusability, performed over the design model of a software product, allows the early identification of products with a low potential of reusability, when the modification and the refinement are still possible (and the cost is not too high).

The identification of reusable elements is well known as one of the most difficult tasks in software reuse. Although the traditional elements (cod segments, objects) are the most frequently reused, great benefits are obtained when big elements, as the business components, are reused.

According to the components based methodologies, the clustering process of classes into modules must begin as early as possible in the development lifecycle (in the analysis stage). So that, in the unified process ([BRJ99a]) the analysis model is decomposed into analysis packages in two ways: in a top-down manner, as well as a bottom-up manner. In the bottom-up approach, when the model becomes too large, it must be decomposed into many packages, in such a way that, after decomposition, it should have a high cohesion degree among the classes inside the packages and low values for coupling among packages (that is as few relationships as possible among classes that belong to different packages). The analysis packages will evolve in subsystems (this is the concept used by the unified process in order to specify a software component in the design stage), and the subsystems, in their turn, will evolve in physical components (used in components diagrams). Therefore, it can be asserted that, among the analysis packages, the subsystems of the design stage and the physical components (specified in UML through the component notation), the traceability is provided.

2 Mathematical model

It will be considered a design model consisting of n classes, where n > 1 (n is higher than one). It will be desired to cluster the classes into a certain number of components, k (where k < n), so there will be obtained values as high as possible for several quality indicators (indicators used to asses the clustering of classes). Till the present moment (for this stage of the work) it was used only one quality indicator, the reusability.

In the book "Designing Object Oriented C++ Applications Using the Booch Method" ([MAR95]), Robert Martin presents several metrics that can be applied for the *class categories*. The equivalent concept in the Universal Modelling Language is the *package*. Therefore, the R. Martin's metrics can be applied in a context of models developed with the unified process and UML, more exactly they can be applied for the subsystems, which represent the concept used to specify the components at a design level (according to the unified process terminology). A basic premise is the desire to build class structures such that a single change to a class will not propagate up and down the class hierarchy without restraint. The class categories are designed to have the property of *closure*. One way of creating subsystems that exhibit closure, is to ensure that the subsystems with the most dependencies on other subsystems also have the greatest resistance to change; those subsystems that are the most changeable, should also have the fewest relationships with other subsystems.

R. Martin's metrics, used in the presented model (and in the algorithm which will be presented in the next section), are the following:

- Relational cohesion – through this metric it is tried to assess the cohesion degree among the classes inside a package. Relational cohesion is defined as the number of relationships between the classes that belong to the package, divided by the number of classes belonging to the same package;

- Afferent coupling – represents the number of classes that depend on the classes belonging to a specified package; it is recommendable to have low values for this metric;

- Efferent coupling – represents the number of classes outside a specified package which depend on the classes belonging to the package; there is recommendable to have low values for this metric;

- Instability – is defined as the ratio between the efferent coupling and the total coupling (defined as the sum between the efferent and afferent coupling) computed for a package;

The elements of the mathematical model will be represented as follows:

- A matrix $M = (m_{ij})$, that represents the inheritance relationships among the classes of the model, where $i = \overline{1,n}$, $j = \overline{1,n}$, and

$$m_{ij} = \begin{cases} 1, & \text{if class } i \text{ is inherited by the class } j \\ 2, & \text{if class } i \text{ inherits class } j \\ 0, & \text{otherwise} \end{cases}$$

- A matrix $A = (a_{ij})$, that represents the association relationships among the classes of the model, where $i = \overline{1,n}$, $j = \overline{1,n}$ (n is the number of classes of the model), and a$_{ij}$represents the number of association relationships between classes i and j.

- A matrix $D = (d_{ij})$, that represents the dependency relationships among the classes of the model, where $i = \overline{1,n}$ (i takes values between one and the maximum number of classes - n), $j = \overline{1,n}$, and d$_{ij}$represents the number of dependency relationships from class i to class j.

- A matrix $Msg = (msg_{ij})$, that represents the messages exchanged among the objects (that is the instances) of the model classes, where $i = \overline{1,n}$, $j = \overline{1,n}$, and msg$_{ij}$represents the number of messages sent by the objects of class i to the objects of class j.

- A matrix $Met = (met_{ij})$, where $i = \overline{1,n}$ and $j = \overline{1,m}$, where m represents the number of internal metric considered.

$$met_{ij} = \text{the value of metric } j \text{ computed for class } i$$

The goal is to find k components (modules), where $1 \leq k \leq n - 1$. For each component it will be possible to impose constraints as regarding the number of classes included (for example, $2 \leq dim \leq n/2$).

A solution is represented by a vector $X = (x_i)$, with $i = \overline{1,n}$, where:

$$x_i = \begin{cases} \text{index of the component to which the class } i \text{ belongs} \\ 0, \quad \text{if the class does not belong to any component} \end{cases}$$

Further on there will be presented the formulas used for computing the R. Martin's metrics (relational cohesion, afferent and efferent coupling), based on the model's elements described above.

In order to compute the relational cohesion, the generalization/specialization, association and dependency relationships were considered, as well as the messages exchanged by classes' objects.

The coupling, also, was computed based on the relationships among classes: generalization/specialization, association and dependency. The interaction diagrams, in UML, are used in order to specify the dynamic aspects of the collaborations among classes: the objects of the classes involved in collaboration exchange messages along the links between them. But, according to UML, the concept of link represents an instance of an association; therefore, the exchange of messages will be done between objects of the classes related by association relationships. In the formulas used to compute the cohesion and the coupling, the messages exchanged by objects are considered in order to have a measurement of the coupling or cohesion intensity.

The relational cohesion, for a component k and a solution X, was computed based on the model elements presented above, as follows:

$$RCoeh(k,X) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} rc(k,i,j,X)}{\sum_{i=1}^{n} s(k,i,X)}, \tag{1}$$

where

$$rc(k,i,j,X) = \begin{cases} w_m \times m_{ij} \; + \; w_a \times a_{ij} \; + \; w_d \times d_{ij} & + w_{msg} \times msg_{ij}, \\ & \text{if } x_i = k \wedge x_j = k, \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

and

$$s(k,i,X) = \begin{cases} 1, & \text{if } x_i = k \\ 0, & \text{otherwise} \end{cases}$$

$w_m =$ the weight assigned to the inheritance relationships;
$w_a =$ the weight assigned to the association relationships;
$w_d =$ the weight assigned to the dependency relationships;
$w_{msg} =$ the weight assigned to the messages exchanged by classes' objects;

As it can be observed in the above formulas, for each relationship type implied in the metrics calculation there was allowed the assignment of weights. This is so because different types of relations among classes have a different meaning as regards the relations' strength (an inheritance relationship represents a stronger relation than a dependency). As a consequence, the software developers could want to assign weight to different types of relations, in accordance with the importance assigned in the clustering process.

The formulas used in order to compute the afferent and efferent coupling for a component k and a solution X, are the following:

$$AC(k,X) = \sum_{i=1}^{n}\sum_{j=1}^{n} ac(k,i,j,X), \tag{3}$$

where

$$ac(k,i,j,X) = \begin{cases} w_m \times m_{ij} \; + \; w_a \times a_{ij} \; + \; w_d \times d_{ij} & + w_{msg} \times msg_{ij}, \\ & \text{if } x_i \neq k \wedge x_j = k \\ 0, & \text{otherwise} \end{cases}$$

$$EC(k,X) = \sum_{i=1}^{n}\sum_{j=1}^{n} ec(k,i,j,X), \tag{4}$$

where

$$ec(k,i,j,X) = \begin{cases} w_m \times m_{ij} \ + \ w_a \times a_{ij} \ + \ w_d \times d_{ij} & + w_{msg} \times msg_{ij}, \\ & \text{if } x_i = k \wedge x_j \neq k \\ 0, & \text{otherwise} \end{cases}$$

The mathematical model, based on the previous elements, is:

$$maximize\, F(X) = \sum_k \left[\frac{RCoeh(k,X)}{(AC(k,X) + EC(k,X) + 1) + \dfrac{EC(k,X)}{AC(k,X) + EC(k,X)}} \right], \tag{5}$$

where

$X =$ represents the searched solution $X = (x_i)$, $i = \overline{1,n}$
It will be necessary to find out X so that $F(X)$ will be maximum.
$RCoeh(k,X) =$ represents the relational cohesion for component k (formula 1);
$AC(k,X) =$ represents the afferent coupling for component k (formula 3);
$EC(k,X) =$ represents the efferent coupling for component k (formula 4);

The constraints of the model are:

$$\begin{cases} G(k,X) \geq \dfrac{number\_of\_classes\_in\_module - 1}{number\_of\_classes\_in\_module} \\ AC(k,X) + EC(k,X) > 0 \\ k\min \leq \sum\limits_i s\,(k,i,X) \leq k\max \end{cases}$$

where

$$G\,(k,X) = \sum_i \sum_j g\,(k,i,j,X)\,,$$

and

$$g\,(k,i,j,X) = \begin{cases} 1, & \text{if } x_i = k \wedge x_j = k \wedge rc\,(k,i,j,X) > 0 \\ 0, & \text{otherwise} \end{cases}$$

The meaning of the constraints is:

- a subsystem must contain more than one class, and among them there have to be relationships (otherwise the value of the relational cohesion will be zero); it must not contain classes that are not related to the other classes within the subsystem;

- a subsystem must have relationships with the other subsystems (it cannot exist in isolation); this means that the total coupling (efferent or afferent) will be higher than zero;
- for a subsystem, the developer can impose conditions as regarding the number of classes (a minimum and a maximum number of classes).

3 The genetic approach

For the clustering of classes in subsystems (software components) the work proposes the utilization of a genetic algorithm. A genetic algorithm applies ideas taken from the natural selection theory, in order to navigate through a large solutions space.

Successfully applying a genetic algorithm in order to solve a problem implies that:

a) a suitable representation of the solution must be found; a solution will be represented by a chromosome;

b) a fitness function must be established in order to evaluate each solution;

c) the genetic operators, which will be applied in the algorithm, must be established;

d) the values of the algorithm parameters must be established, that is: dimension of the population, the probabilities with which the operators will be applied etc.;

a) The ***representation of a solution*** must be:

- complete;
- valid.

A complete representation assumes that there is a possibility to encode all the solutions for the studied problem. A valid representation assumes that all the codifications are in the solution space. Invalid representations can be used, but the algorithm has to be adapted in order to avoid invalid solutions.

In this paper, for the discussed problem, a solution was encoded as a vector (the vector elements are positive integers), as follows:

- the index of each element represents a class of the model;
- the value of each element is a positive integer and represents the index of the component to which the class belongs.

Each element of a chromosome can take a value between one and the maximum number of components. Through this representation the completeness is assured. Therefore, it will be possible to represent any clustering of classes in modules, through a chromosome. The validity of a chromosome has to be checked out, in case there are imposed constraints regarding the number of classes within a module.

b) The ***fitness function*** quantifies each solution represented by a chromosome and is used as basis for chromosomes selection for mating. For the discussed problem the fitness function is represented by formula (5).

The objective function is built so that it will lead to a clustering in subsystems (components) with low values for coupling (afferent and efferent) and for instability, and high values for relational cohesion.

The function computed for each subsystem (component), is the ratio between the relational cohesion of the classes within the module, and the sum between the coupling (afferent and efferent) and the module instability.

The relational cohesion (formula 1) is computed as the ratio between the number of relationships among the classes within a module and the number of classes belonging to the module, so that it will not be possible to have high values for cohesion (and for the fitness function) if a subsystem contains a high number of classes, but with few relationships among them.

c) The ***operators*** used by the algorithm are the following:

Initialization operator – the initial population is randomly generated. The algorithm was tested using several initialization operators that generate individuals composed by groups of two, three, four and five classes. For each element of a chromosome is generated a positive integer between one and the maximum number of subsystems (in which the model can be divided).

The *selection operator* used by the algorithm is *roulette wheel.*

As regarding the *crossover* operator, the algorithm was tested using several crossover operators. The classical operators used, are the following:

- one-point-crossover;
- two-point-crossover;
- uniform-crossover;

In addition to the above presented operators, it was tested a new one (named ClusterCrossover) for which there have been obtained the good values for the fitness function. The results are presented in section 4.

Further on the new crossover operator will be presented.

The operator works as follows: having two chromosomes selected for crossover, $X_p = (x_{p1}, ..., x_{pn})$ and $X_q = (x_{q1}, ..., x_{qn})$. One point, in the first chromosome, is randomly selected: s_{pi}. The new chromosomes are created as follows:

- the first child is composed of all the positions from Xp equals with x_{pi} and the others positions from Xq;
- the second child takes the remaining values from the two parent chromosomes.

The mutation operators used by the algorithm were *swap-mutator* and a mutation operator presented in the paper "FGKA: A Fast Genetic K-means Clustering Algorithm" ([LLF04]), adapted for the discussed problem, which will be presented further on.

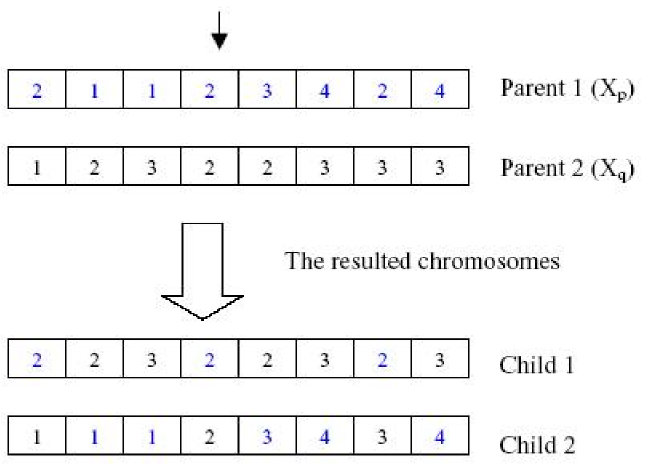

Figure 1: The crossover operation

If it is considered a chromosome $X_p = (x_{p1}, ..., x_{pn})$, the element x$_{pi}$ is replaced by the value $x_{pi'}$, for $i = \overline{1,n}$, where $x_{pi'}$ is a subsystem, randomly selected from 1 to k, with the probability:

$$pmut_t = \frac{1,5 * Dist_{pmax} - Disp_p(i, x_{pi}) + 0.5}{\sum_{j=1}^{k}(1,5 * Dist_{pmax} - Dist_p(i,j) + 0.5)},$$

where t between 1 and k (k is the maximum number of components).

Dist$_{pmax}$ represents the maximum square of the Euclidian distance between class i and one of the subsystems, and Dist$_p$(i) represents the square of the Euclidian distance between class i and the subsystem x$_{pi}$.

The square of the Euclidian distance is computed as the square of the difference between the centroid of the subsystem x$_{pi}$(Centr(x$_{pi}$,X$_p$))and the weight of the class i (W(i,X$_p$)):

$$Dist_p(i) = [Centr(x_{pi}, X_p - W(i, X_p))]^2$$

The weight W of the class i is computed with the following formula (the ratio between all the relationships between class i and the other classes belonging to the same subsystem, divided to the number of classes within the subsystem, and the relationships between class i and the classes outside the subsystem):

$$W(i, X_p) = \frac{\sum_{j,i \neq j}[cupl\_in(i,j,X_p)/s(i,X_p)]}{\sum_{j,i \neq j} cupl\_out(i,j,X_p)}, \tag{6}$$

where

$$cupl\_in(i,j,X_p) = \begin{cases} w_m \times m_{ij} + w_a \times a_{ij} + w_d \times d_{ij} + w_{msg} \times msg_{ij}, & \text{if } x_{pi} = x_{pj} \\ 0, & \text{if } x_{pi} \neq x_{pj} \end{cases}$$

$$cupl\_out(i,j,X_p) = \begin{cases} w_m \times m_{ij} + w_a \times a_{ij} + w_d \times d_{ij} + w_{msg} \times msg_{ij}, \\ \qquad \text{if } x_{pi} \neq x_{pj} \\ 0, \quad \text{if } x_{pi} = x_{pj} \end{cases}$$

The formula for the centroid of the subsystem x_{pi} is:

$$Centr(i,X_p) = \frac{\sum_i class\_weight(i,t,X_p)}{\text{number of classes within module } t}, \tag{7}$$

where

$$class\_weight(i,t,X_p) = \begin{cases} W(i,X_p), & t = x_{pi} \\ 0, & t \neq x_{pi} \end{cases}$$

d) The ***algorithm parameters*** were found as follows:

- the initial dimension of the population: 20 x n (where n is the number of classes within the model);
- the crossover probability: 0.9;
- the mutation probability: 0.01;
- the overlapping degree of the populations: 20%;
- number of generations: 50 x n;

The algorithm involves the following steps:

1. The initial population is generated. There are created fixed length strings (the length of a string is equal to the number of classes within the model). The strings are randomly initialized using one of the initialisation operators described above.
2. There are selected the individuals for mating using roulette wheel operator.
3. One of the crossover operators described above is applied.
4. One of the mutation operators presented above is applied.
5. The old population is replaced by the new one: the worst individuals are removed from the temporary population, in order to return the population to its original size.
6. If the required number of iterations is not reached, then the algorithm will continue with step 2.

4 Case study and results

The results that will be presented further on, were obtained as a result of running the algorithm on a model composed of 17 classes. This model, showed in figure 2, is extracted from a larger project, in which a B2B application is developed: an intermediary site that takes the requests (the orders) from dealers and sends them to manufacturers.

At present, the information about the model is extracted manually and introduced into a database in order to be used by the algorithm. Further on, an application will be developed that will be able to extract the information about the UML model from an XMI (XML Metadata Interchange) file.

The following image presents the cluster of classes proposed by the algorithm (in the best case).

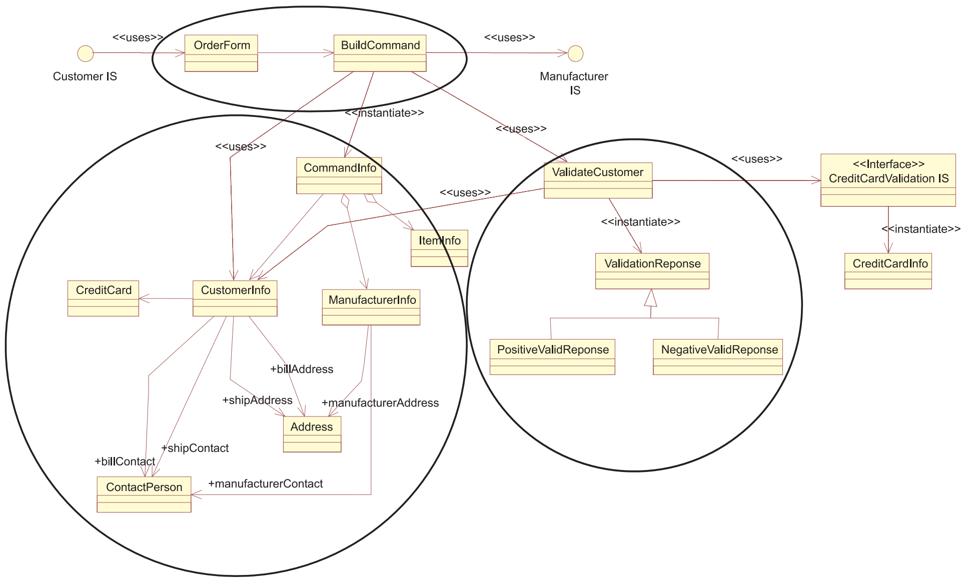

Figure 2: The clusters of classes proposed by the algorithm

The following figure represents the results obtained as a consequence of applying the swap-mutator operator together with the uniform, two-points and cluster-crossover (the new operator tested), for initial population which contains individuals formed by clusters of three classes.

As it can be observed from the above picture, the best results were obtained for the ClusterCrossover operator.

Figure 4 presents the results obtained as a consequence of applying the same crossover and mutation operators as in the previous figure, but with an initial population which contains individuals formed by clusters of four classes (the initialization operator is different).

Again, like in the first case, the best fitness function value is obtained in a smaller number of iterations for the ClusterCrossover operator, than for uniform and two-point crossover.

In the following picture the same operators are applied as in the previous case, but with the initial population which contains individuals formed by cluster of five classes.

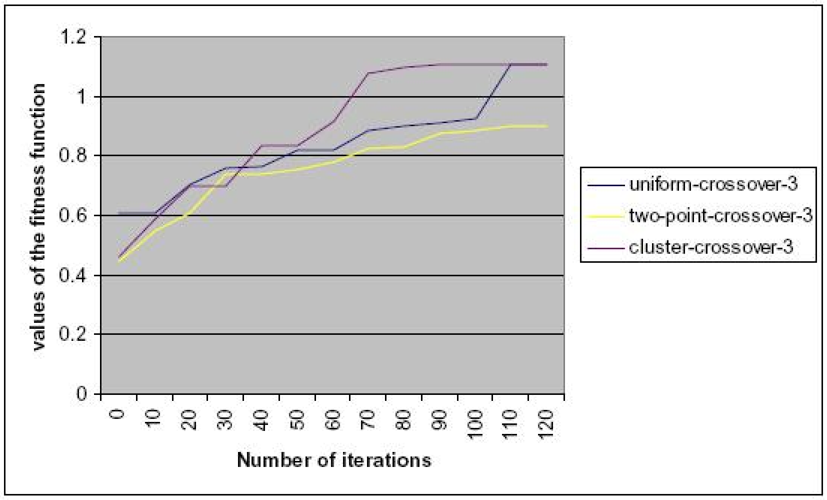

Figure 3: Results for an initial population composed by individuals formed by clusters of three classes

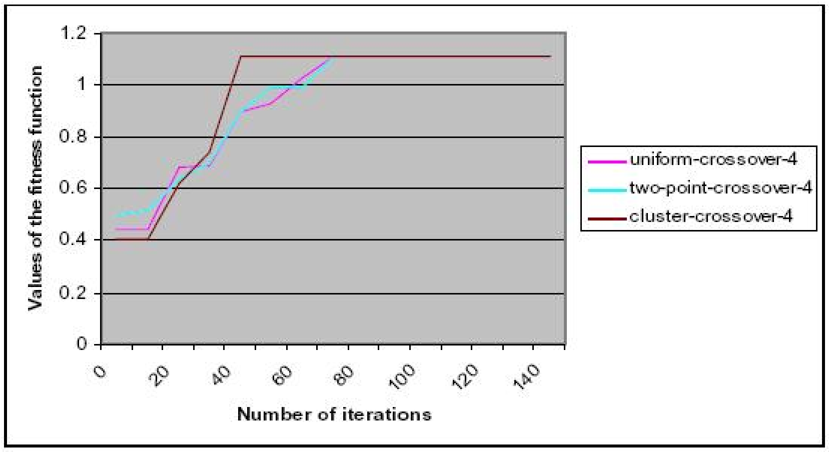

Figure 4: Results for an initial population composed by individuals formed by clusters of four classes

In figure 6 there are presented the results obtained as a consequence of applying the FGKA-mutator operator together with the uniform, two-points and cluster-crossover (the new operator tested), for the initial population which contains individuals formed by clusters of two classes. It can be asserted that if the algorithm starts with a large number of subsystems that must be reduced during the evolution of the algorithm, then the FGKA-mutator is more suitable than the swap-mutator operator.

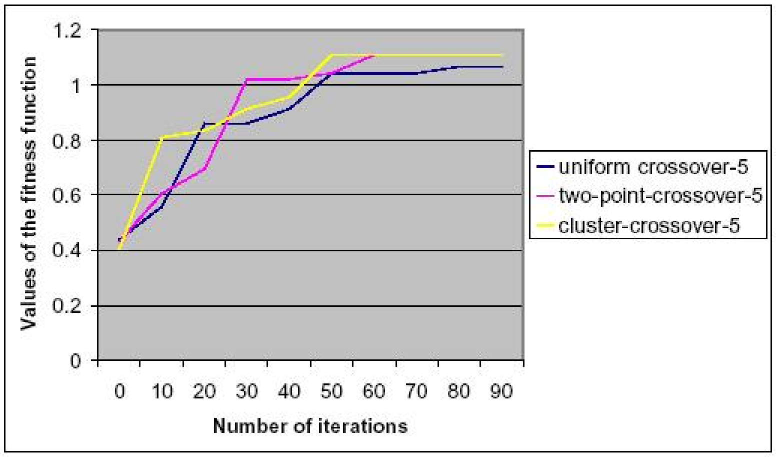

Figure 5: Results for an initial population composed by individuals formed by clusters of five classes

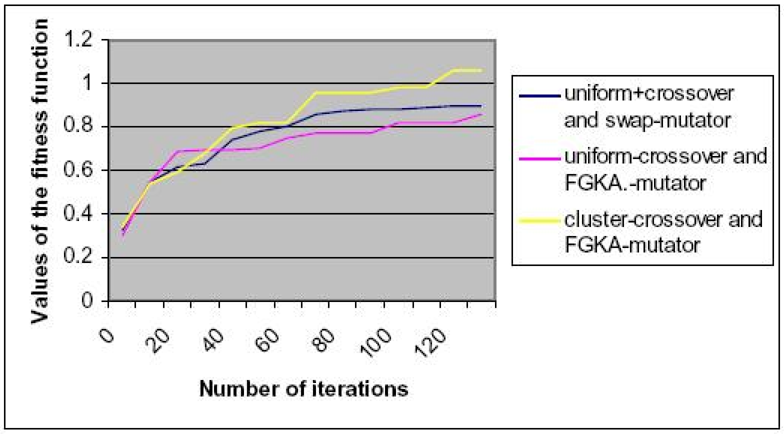

Figure 6: Results of applying FGKA-mutator together with the uniform, two-points and cluster-crossover

5 Conclusions and future work.

The structure of the software systems can be very complex. A design model for a medium size application can include tens of classes (even more than one hundred), the objects (the instances) of a class could be included in more than one interaction diagram. Therefore, it is difficult enough for a developer to find the optimum distribution of classes among the modules that compose the application, so that the reusability should be as easy as possible and should involve a minimum number of modifications (the ideal case is to have not any modification) within the module.

The goal of the algorithm proposed in this work is to help software developers in order to reach a decision as close as possible to the best one (to find the optimal distribution of classes among subsystems).

There are, already, in the speciality literature, approaches that try to identify software components (to group classes in components), but they are applicable during the implementation phase (on the source code). Therefore, in [ETZ97] is presented an algorithm for the automated identification of reusable software components, which collects the metrics from a C/C++ source code. The algorithm is based on metrics like LCOM, average number of comment lines per method etc., that it is applicable only for the implementation phase.

Another approach, presented in [JCI01], achieves a clustering of classes (using a clustering algorithm) based on coupling intensity between them. In a second phase there are applied heuristics in order to find the best values for several quality indicators. In contrast with this approach, the algorithm presented in this work achieves the clustering of classes into subsystems, having as goal a high level of reusability for them. The clustering is based on several architectural metrics (R. Martin's metrics), that can be applied against the subsystems (packages). In addition to the well known metrics, like cohesion and coupling, there is also considered the module stability. According to R. Martin, "the dependencies between components in a design should be in the direction of stability" and "a component should only depend upon components that are more stable than it".

The approach presented in this work is addressed to the design phase because it was considered that the impact of modifications, involved by the cluster of classes proposed by the algorithm, is less than during the implementation phase. The future work will consist of:

- New metrics will be included in the objective function, metrics which can be computed at design model level, like the following: depth of inheritance tree (DIT), number of children (NOC) etc.

- Modalities (indicators) will be defined for the assessment of the impact over the final product, as a result of applying the algorithm in the design phase (how evolved the identified subsystems in physical components, implemented using the new technologies).

- The modification of the mathematical model and of the algorithm, in order to be possible to assign a class to several packages. In practice it is possible to have situations in which, in order to reduce the dependency between modules, it should be necessary to introduce new classes, even if the information managed by these is redundant. Therefore, a solution will be represented as a matrix (n x n, where n is the number of classes) in which a matrix element, $n_{ij,}$ will contain the index of the subsystem to which both classes (i and j) belong, or 0 if the classes don't belong to the same subsystem.

References

[BRJ99a] Booch G., Raumbaugh J., Jacobson I., *The Unified Software Development Process*, Addison/Wesley 1999.

[BRJ99b] Booch G., Raumbaugh J., Jacobson I., *The Unified Modelling Language. User Guide*, Addison/Wesley 1999.

[CHK94] Chidamber S., Kemerer C., *Metrics Suite for Object Oriented Design*, IEEE Transaction on Software Engineering, vol.20, No.6, 1994.

[ETZ97] Etzkorn, L. H., *A Metrics-Based Approach to the Automated Identification of Object-Oriented Reusable Software Components*, http://www.cs.uah.edu/∼letzkorn/disserta.pdf, 1997.

[JCI01] Hemant Jain, Naresh Chalimeda, Navin Ivaturi, Balarama Reddy, *Business Component Identification – A Formal Approach*, Fifth IEEE International, Enterprise Distributed Object Computing Conference, 2001.

[LLF04] Lu, Y., Lu S., Fotouhi F., *A Fast Genetic K-means Clustering Algorithm*, http:// www.cs.wayne.edu/∼shiyong/papers/sac04.pdf, 2004

[KHA01] Khaled El Emam, *Object Oriented Metrics: A Review of Theory and Practice*, http://citeseer.nj.nec.com/479219.html, 2001.

[MAR02] Marinescu, R., *Principles of Object-Oriented Design*,

http://labs.cs.utt.ro/labs/ip2/html/2002/lectures/3/lecture3.pdf

[MAR95] Martin, R., *Designing Object-Oriented C++ Applications Using the Booch Method.*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[MIC96] Michalewicz Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Heidelberg, 1996.

[MRC98] Marchesi, M., *OOA Metrics for the Unified Modelling Language*, Proceedings of the 2end Euromicro Conference on Software Maintenance and Engineering, 1998, pp. 67-73.

[XHC99] Tao Xie, Huang Huang, Xiangkui Chen, *Object Oriented Software Metrics Technology*, http://www.cs.washington.edu/homes/taoxie/ RicohmiddleReport.pdf , 1999.