Evolutionary Approach for the Strategy-based Refactoring Selection

Camelia Chisăliţă-Creţu, *Member, IAENG*

***Abstract*—In order to improve the internal structure of object-oriented software, refactoring has proved to be a feasible technique. Scheduling a refactoring process for a complex software system is a difficult task to do. Refactorings may be organized and prioritized based on goals established by the project management leadership, that shapes a refactoring strategy.**

The paper presents a multi-objective approach to the Strategy-based Refactoring Set Selection Problem (SRSSP) by treating the cost constraint and the refactoring impact as objectives of a weighted-sum fitness function.

The first results of the proposed weighted objective genetic algorithm on a experimental didactic case study are presented and discussed.

***Index Terms*—genetic algorithm, multi-objective optimization, refactoring, object-oriented programming, software engineering.**

I. Introduction

SOFTWARE systems continually change as they evolve to reflect new requirements, but their internal structure tends to decay. Refactoring is a commonly accepted technique to improve the structure of object oriented software. Its aim is to reverse the decaying process of software quality by applying a series of small and behaviour-preserving transformations, each improving a certain aspect of the system [11].

Refactorings may be organized and prioritized based on goals established by the project management leadership. The SRSSP definition is based on the Refactoring Set Selection Problem (RSSP) [4], [6]. Therefore, the SRSSP is the refactoring set selection problem that combines multiple strategy criteria in order to to find the most appropriate set of refactorings.

The rest of the paper is organized as follows. The motivation for the addressed problem is highlighted in Section II. Section III presents close related work on refactoring selection for the SRSSP. Useful formal notations inherited from RSSP [4], [6], together with the formal definition for the SRSSP are presented in Section V. Section IV gives the definition of the Multi-Objective Optimization Problem (MOOP). The multi-objective optimization formulation for the SRSSP is stated in Section VI. A short description of the Local Area Network (LAN) Simulation source code used to study our approach is provided in Section VII. The proposed approach and several details related to the genetic operators of the genetic algorithm are described in Section VIII. The obtained results for the studied source code are presented and discussed in Section IX. The paper ends with conclusions and future work.

Manuscript received August 05, 2014; revised August 14, 2014.

C. Chisăliţă-Creţu is with the Computer Science Department, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania, 1, M. Kogalniceanu Street, RO-400084, Tel: 40-264-405.300/5240; email: cretu@cs.ubbcluj.ro.

II. Motivation

A refactoring management process for a complex software system has proved to be a difficult task to do [11]. Multiple refactoring aspects of different parts of a heavy working system need increased attention when planning the order to refactor. Moreover, within a development team, each programmer perceives the refactoring process in his own manner. A refactoring strategy allows to fit each transformation performed on the software system in a general refactoring plan, following a criteria set that unifies particular transformation requests into a homogenous single and desired development trend.

A tool [17] may be used to identify refactoring opportunities for each established bounded piece of the software system, i.e, class hierarchies, software components. Each software programmer involved in the development process may advance his set of refactorings that improves the internal structure of the software piece developed by him. Thereafter, a consistent number of refactorings is handed to the project management leadership. It has to decide the appropriate refactoring plan, based on the already known targets. The set of refactorings is used to select a subset of transformations suggested by the previously specified criteria.

The project management leadership faces several problems within the considered context. These problems emphasize different aspects of a complex refactoring process, as:

- a large number of refactorings advanced;
- different types of dependencies among the affected software entities, e.g., an inherited method from a base class is called within another method of a derived class;
- different types of dependencies among refactorings to be satisfied when combining the transformation sequences, i.e., applying a suggested refactoring may cancel the application of another refactorings that have been already selected by the developer;
- a specific refactoring priority for each software entity;
- a clear request to include a transformation within the final refactoring plan.

III. Related Work

A closely related previous work to refactoring selection problems is the Next Release Problem (NRP) studied by several authors [2], [12], where the goal was to find the most appropriate set of requirements that balance resource constraints to the customer requests, the problem being defined as a constrained optimization problem.

Other Feature Subset Selection (FSS) problems in previous work on Search-Based Software Engineering (SBSE)

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

include the problem of determining good quality predictors in software project cost estimation, studied by Kirsopp et al. [16], choosing components to include in different releases of a system, studied by Harman et al. [13] and Vescan et al. [20].

Previous work on search-based refactoring problems [14], [15], [1] in SBSE has been concerned with single objective formulations of the problem only. Much of the other existing work on SBSE has tended to consider software engineering problems as single objective optimization problems too. But recent trends show that multi-objective approach has been tackled too, which appears to be the natural extension of the initial work on SBSE.

Existing SBSE work that does consider multi-objective formulations of software engineering problems, uses the weighted approach to combine fitness functions for each objective into a single objective function using weighting coefficients to denote the relative importance of each individual fitness function. In the search based refactoring field, Seng et al. [19] and O'Keeffe and O'Cinneide [15] apply a weighted multi-objective search, in which several metrics that assess the quality of refactorings are combined into a single objective function.

More recent work on search based refactoring problems [3], [4], [5] in SBSE have defined the General Refactoring Selection Problem (GRSP), used to refine the Multi-Objective Refactoring Set Selection Problem (MORSSP) [4] and the Multi-Objective Refactoring Sequence Selection Problem (MORSqSP) [5].

Our approach is similar to those presented in [19], [15]. The research has addressed the heterogeneous objective functions approach, where multiple objectives are combined together into a single weighted fitness function. Thus, we gather up different objectives as the refactoring cost and refactoring application impact in a single fitness function.

IV. MOOP Model

MOOP is defined in [21] as the problem of finding a decision vector $\vec{x} = (x_1, \ldots, x_n)$, which optimizes a vector of M objective functions $f_i(\vec{x})$ where $1 \leq i \leq M$, that are subject to inequality constraints $g_j(\vec{x}) \geq 0$, $1 \leq j \leq J$ and equality constraints $h_k(\vec{x}) = 0$, $1 \leq k \leq K$. A MOOP may be defined as:

$$maximize\{F(\vec{x})\} = maximize\{f_1(\vec{x}), \ldots, f_M(\vec{x})\},$$

with $g_j(\vec{x}) \geq 0, 1 \leq j \leq J$ and $h_k(\vec{x}) = 0, 1 \leq k \leq K$ where $\vec{x}$ is the vector of decision variables and $f_i(\vec{x})$ is the i-th objective function, $g(\vec{x})$ and $h(\vec{x})$ are constraint vectors.

There are several ways to deal with a multi-objective optimization problem. In this paper the weighted sum method [18] is used.

Let f_1, f_2,..., f_M be the addressed objective functions. This method takes each objective function and multiplies it by a fraction of one, the "weighting coefficient" which is represented by $w_i, 1 \leq i \leq M$. The modified functions are then added together to obtain a single fitness function, which can easily be solved using any method which can be applied for single objective optimization. Mathematically, the new mapping may be written as:

$$F(\vec{x}) = \sum_{i=1}^{M} w_i \cdot f_i(\vec{x}),\ 0 \leq w_i \leq 1,\ \sum_{i=1}^{M} w_i = 1.$$

For those cases where the conflicting objectives exist, they must be converted to meet the optimization problem requirements. Therefore, for an objective $f_i,\ 0 \leq i \leq M$, with MAX the highest value from the objective space of the corresponding objective mapping f_i that needs to be converted to a minimized objective, there are two ways to switch to the optimal objective:

- $MAX - f_i(\vec{x})$, when MAX can be computed;
- $-f_i(\vec{x})$, when MAX cannot be computed.

V. Strategy-based Refactoring Set Selection Problem

The Strategy-based Refactoring Set Selection Problem (SRSSP) is mainly based on the Refactoring Set Selection Problem (RSSP) fully formalized in [3]. SRSSP is a special case of RSSP where the refactoring selection is enhanced by certain criteria, e.g., refactoring application priority, refactoring application type: optional or mandatory.

The SRSSP formal definition requires several input data notations from the RSSP. Subsequently, additional terms and notations are introduced to completely state the SRSSP.

Input Data

Let $SE = \{e_1, \ldots, e_m\}$ be a set of software entities as it was defined in [3].

The software entity set SE together with different types of dependencies among its items form a software system named SS. The set of software entity dependency types SED and the dependency mapping ed are described in [3].

A set of relevant chosen refactorings that may be applied to the software entities of SE is gathered up through $SR = \{r_1, \ldots, r_t\}$. The ra mapping sets the applicability for each refactoring from the chosen set of refactorings SR on the set of software entities SE as it was defined in [3].

The set of refactoring dependencies $SRD = \{\mathbf{B}efore,\ \mathbf{A}fter,\ \mathbf{A}lways\mathbf{B}efore,\ \mathbf{A}lways\mathbf{A}fter,\ \mathbf{N}ever,\ \mathbf{W}henever\}$, together with the mapping rd that highlights the dependencies among different refactorings when applied to the same software entity are stated in [3].

The effort involved by each transformation is converted to cost, described by rc mapping [3]. Changes made to each software entity $e_i, i = \overline{1,m}$, by applying the refactoring $r_l, 1 \leq l \leq t$, are stated by the $effect$ mapping defined in [3]. The overall impact of applying a refactoring $r_l, 1 \leq l \leq t$, to each software entity $e_i, i = \overline{1,m}$, is defined as $res : SR \rightarrow Z$,

$$res(r_l) = \sum_{i=1}^{m} w_i \cdot effect(r_l, e_i),$$

where $1 \leq l \leq t$ and w_i is the weight of the corresponding software entity e_i from SE.

SR_e represents the *subset of refactorings that may be applied to a software entity* $e,\ e \in SE$ [6]. Therefore, $SR = \bigcup_{e_i \in SE} SR_{e_i},\ i = \overline{1,m}$.

SE_r represents the *subset of software entities to whom a refactoring* r *may be applied*, $r \in SR$ [6]. Therefore, $SE = \bigcup_{r_l \in SR} SE_{r_l},\ l = \overline{1,t}$.

In [8], the refactoring-entity pair notion was introduced, as it was required for the refactoring sequence selection problem definition. Therefore, a *refactoring-entity pair* was defined as a tuple $\widehat{r_l\ e_i} = (r_l, e_i)$ consisting of a refactoring $r_l,\ 1 \leq$

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$l \leq t$, applied to a software entity $e_i,\ 1 \leq i \leq m$, where $ra(r_l, e_i) = T$.

Let $REPSet = \{\widehat{r_1\ e_1},\ \widehat{r_2\ e_2}, \ldots, \widehat{r_p\ e_p}\},\ p \in \mathcal{N}$ be the *set of all refactoring-entity pairs* build over SR and SE, where $ra(r_s, e_s) = T,\ 1 \leq s \leq p$.

Refactoring Strategy

The refactoring strategy may be formally described by one or more functions $sf_i,\ i = \overline{1, NC}$, where NC is the total number of criteria integrated with the strategy. In the following, a sample strategy consisting of two criteria, i.e., mappings, is introduced.

The development team may consider relevant that in a specific context some refactoring applications to be mandatory, optional or selected from a subset. Let $RType = \{\mathbf{M}andatory,\ \mathbf{O}ptional,\ \mathbf{S}elected\}$ be the set of possible refactoring types. The mapping $rtype$ associates a type to each refactoring from SR as follows:
$rtype : SR \rightarrow RType$,

$$rtype(r) = \begin{cases} \text{M}, & \text{if } r \text{ application } \textit{mandatory} \\ \text{O}, & \text{if } r \text{ application } \textit{optional} \\ \text{S}, & \text{if } r \in \{r_1, \ldots,\ r_q\},\ 0 \leq q \leq t \end{cases}$$

A second criterion considered by the development team may refer the level of the affected entity when refactoring. Let $RLevel = \{\mathbf{A}ttribute,\ \mathbf{M}ethod,\ \mathbf{C}lass\}$ be the set of refactoring levels involved in the transformation process. Therefore, the function $rlevel$ maps each refactoring to the entity level that it mainly changes, as: $rlevel : SR \rightarrow RLevel$,

$$rlevel(r) = \begin{cases} \text{a}, & \text{if } r \text{ is applied to } \textit{a}\text{ttributes} \\ \text{m}, & \text{if } r \text{ is applied to } \textit{m}\text{ethods} \\ \text{c}, & \text{if } r \text{ is applied to } \textit{c}\text{lasses} \end{cases}$$

Output Data

The strategy-based refactoring set selection means to choose a appropriate refactoring subset such that the stated criteria on refactorings is met, e.g., refactoring application level and type.

Other specific conditions to be satisfied refers to the refactoring cost and the refactoring final impact on entities. Therefore, a multi-objective strategy-based refactoring set selection problem (MOSRSSP) may be defined.

Multi-objective optimization often means optimizing conflicting goals. For the MOSRSSP formulation it is possible to blend different types of objectives, i.e., some of them to be maximized and some of them to be minimized.

VI. MOSRSSP Formulation

Multi-objective optimization often means compromising conflicting goals. For our MOSRSSP formulation there are two objectives taken into consideration in order minimize required cost for the applied refactorings and to maximize refactorings impact upon software entities. Current research treats cost as an objective instead of a constraint. Therefore, the first objective function minimizes the total cost for the applied refactorings, as:

$$minimize\left\{f_1(\vec{r})\right\} = minimize\left\{\sum_{l=1}^{t}\sum_{i=1}^{m} rc(r_l, e_i)\right\},$$

where $\vec{r} = (r_1, \ldots, r_t)$.

The second objective function maximizes the total effect of applying refactorings upon software entities, considering the weight of the software entities in the overall system, like:

$$maximize\left\{f_2(\vec{r})\right\} = maximize\left\{\sum_{l=1}^{t} res(r_l)\right\},$$

where $\vec{r} = (r_1, \ldots, r_t)$.

The goal is to identify those solutions that compromise the refactorings costs and the overall impact on transformed entities. In order to convert the first objective function to a maximization problem for the MOSRSSP, the total cost is subtracted from MAX, the biggest possible total cost, as it is shown below:

$$maximize\left\{f_1(\vec{r})\right\} = maximize\left\{MAX - \sum_{l=1}^{t}\sum_{i=1}^{m} rc(r_l, e_i)\right\},$$

where $\vec{r} = (r_1, \ldots, r_t)$. The final fitness function for MOSRSSP is defined by aggregating the two objectives and may be written as:

$$F(\vec{r}) = \alpha \cdot f_1(\vec{r}) + (1 - \alpha) \cdot f_2(\vec{r}) \quad (1)$$

where $0 \leq \alpha \leq 1$.

Let $DS = REPSet$ be the *decision domain* for the MOSRSSP and $\vec{x} = \{\widehat{r_1, e_1}, \widehat{r_2, e_2}, \ldots, \widehat{r_s, e_s}\}$, where $e_u \in SE,\ r_u \in SR,\ 1 \leq u \leq s,\ s \in \mathcal{N},\ \vec{x} \subseteq DS$, a decision variable.

The MOSRSSP is the problem of finding a decision vector $\vec{x} = \{\widehat{r_1, e_1},\ \widehat{r_2, e_2}, \ldots, \widehat{r_s, e_s}\}$ such that:

- the following objectives are optimized:
  - the overall refactoring cost is minimized (rc) [3];
  - the overall refactoring impact on software entities is maximized (res) [3].
- the following constraints are satisfied:
  - software entity dependencies (ed) [3];
  - refactoring dependencies (rd) [3].
- the addressed strategy-based criteria are met:
  - $RMandatory = \{r_1, \ldots,\ r_{rm}\}$ is the set of mandatory refactorings, where $r_1, \ldots,\ r_{rm} \in SR$, $0 \leq rm \leq t$;
  - $ROptional = \{r_1, \ldots,\ r_{ro}\}$ is the set of mandatory refactorings, where $r_1, \ldots,\ r_{ro} \in SR$, $0 \leq ro \leq t$;
  - $RSelect = \{r_1, \ldots,\ r_{rs}\}$ is the set of single selected refactorings, where $r_1, \ldots,\ r_{rs} \in SR$, $0 \leq rs \leq t$;
  - $1 \leq rm + ro + rs \leq t$, $RMandatory \bigcap ROptional \bigcap RSelect = \phi$;
  - conditions on the number of applied refactorings on attribute, method, and class levels are met.

VII. Case Study: LAN Simulation

The algorithm proposed was applied on a simplified version of the Local Area Network (LAN) simulation source code that was presented in [10]. Figure 1 shows the class diagram of the studied source code. It contains 5 classes with 5 attributes and 13 methods, constructors included.

The current version of the source code lacks of hiding information for attributes since they are directly accessed

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

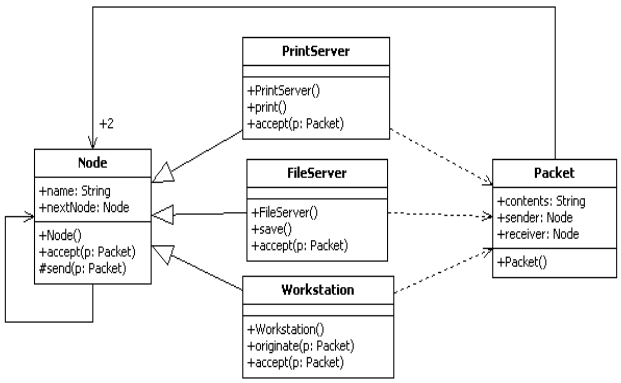


Fig. 1. Class diagram for LAN Simulation

by clients. The abstraction level and clarity may be increased by creating a new superclass for `PrintServer` and `FileServer` classes, and populate it by moving up methods in the class hierarchy.

Thus, for the studied problem the software entity set is defined as: $SE = \{c_1,\ ...,\ c_5,\ a_1,\ ...,\ a_5,\ m_1,\ ...,\ m_{13}\}$. The chosen refactorings that may be applied are: *renameMethod*, *extractSuperClass*, *pullUpMethod*, *moveMethod*, *encapsulateField*, *addParameter*, denoted by the set $SR = \{r_1, \ldots, r_6\}$ in the following. The dependency relationship between refactorings is defined as follows: $\{(r_1, r_3) = B,\ (r_1, r_6) = AA,\ (r_2, r_3) = B,\ (r_3, r_1) = A,\ (r_6, r_1) = AB,\ (r_3, r_2) = A,\ (r_1, r_1) = N,\ (r_2, r_2) = N,\ (r_3, r_3) = N,\ (r_4, r_4) = N,\ (r_5, r_5) = N,\ (r_6, r_6) = N\}$.

The values of the final effect were computed for each refactoring, by using the weight for each existing and possible affected software entity, as it was defined in Section V. Therefore, the values of the res function for each refactoring are: 0.4, 0.49, 0.63, 0.56, 0.8, 0.2. The full input data table is included in [4].

Here, the cost mapping rc is computed as the number of the needed transformations, so related entities may have different costs for the same refactoring. Each software entity has a weight within the entire system, but $\sum_{i=1}^{23} w_i = 1$. For the $effect$ mapping, values were considered to be numerical data, denoting the estimated impact of applying a refactoring. Due to the space limitation, intermediate data for these mappings was not included.

The refactoring strategy consists of the following refactoring criteria:

- $RMandatory = \{r_2,\ r_5\}$;
- $ROptional = \{r_1,\ r_6\}$;
- $RSelect = \{r_3,\ r_4\}$, where if r_3 is applied to entity $m_i,\ i = \overline{1,13}$, r_4 will not be selected to by applied to the same entity;
- $1 \leq |RMandatory| + |ROptional| + |RSelect| \leq 6$, $RMandatory \bigcap ROptional \bigcap RSelect = \phi$;
- refactorings of all levels have to be selected (attribute, method, and class).

An acceptable solution denotes lower costs and higher impact on transformed entities, both objectives being satisfied. The entities dependencies and refactoring dependencies need to be met as well, while the strategy selection criteria constraints have to be fulfilled.

VIII. Proposed Approach Description

The MOSRSSP is approached here by exploring the possible application strategy for the addressed refactorings. As its multi-objective formulation states it (see Section VI), two conflicting objectives are studied, i.e., minimizing the refactoring cost and maximizing the refactoring impact, together with the constraints to be kept and the selection strategy criteria to be followed.

There are several ways to handle a multi-objective optimization problem. The *weighted sum method* [18] was adopted to solve the MOSRSSP. The overall objective function to be maximized $F(\vec{r})$, defined by the formula 1, is shaped to the weighted sum principle with two objectives to optimize.

Therefore,

$$maximize\left\{F(\vec{r})\right\} = maximize\left\{f_1(\vec{r}), f_2(\vec{r})\right\}$$

is mathematically rewritten to:

$$maximize\left\{F(\vec{r})\right\} = \alpha \cdot f_1(\vec{r}) + (1-\alpha) \cdot f_2(\vec{r}),$$

where $0 \leq \alpha \leq 1$ and $\vec{r}$ is the decision variable, within a decision space.

An adapted genetic algorithm to the context of the investigated problem, with weighted sum fitness function, similar to the one in [5], [7], is proposed here.

In a steady-state evolutionary algorithm a single individual from the population is changed at a time. The best chromosome (or a few best chromosomes) is copied to the population in the next generation. Elitism can very rapidly increase performance of genetic algorithm, because it prevents to lose the best found solution to date.

The genetic algorithm approach uses a *refactoring-based* solution representation for the strategy-based refactoring set selection problem, being denoted by *SRSSGARef*.

The decision vector $\vec{S} = (S_1, \ldots, S_t)$, where $S_l \in \mathcal{P}(SE), 1 \leq l \leq t$, determines the entities that may be transformed using the proposed refactoring set SR. The item S_l on the l-th position of the solution vector represents a set of entities that may be refactored by applying the l-th refactoring from SR, where any $e_{l_u} \in SE_{r_l}, e_{l_u} \in S_l$, $S_l \in \mathcal{P}(SE), 1 \leq u \leq q, 1 \leq q \leq m, 1 \leq l \leq t$. This means it is possible to apply more than once the same refactoring to different software entities, i.e., distinct gene values from the chromosome may contain the same software entity.

A. Genetic Operators

Crossover and mutation operators are used by this approach, being described in the following.

Crossover Operator

A simple one point crossover scheme is used. A crossover point is randomly chosen. All data beyond that point in either parent string is swapped between the two parents.

For instance, if the two parents are:

$parent_1 = [ga[1,7],\ gb[3,5,10],\ gc[8],\ gd[2,3,6,9,12],\ ge[11],\ gf[13,4]]$ and

$parent_2 = [g1[4,9,10,12],\ g2[7],\ g3[5,8,11],\ g4[10,11],\ g5[2,3,12],\ g6[5,9]]$, for the cutting point 3, the two resulting offsprings are:

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$offspring_1 = [ga[1,7],\ gb[3,5,10],\ gc[8],\ g4[10,11],\ g5[2,3,12],\ g6[5,9]]$ and
$offspring_2 = [g1[4,9,10,12],\ g2[7],\ g3[5,8,11],\ gd[2,3,6,9,12],\ ge[11],\ gf[13,4]]$.

Mutation Operator

The mutation operator used here exchanges the value of a gene with another value from the allowed set. Namely, mutation of the i-th gene consists of adding or removing a software entity from the set that denotes the i-th gene.

For example, if the individual to be mutated is
$parent = [ga[1,7],\ gb[3,5,10],\ gc[8],\ gd[2,6,9,12],\ ge[12],\ gf[13,4]]$ and if the 5-th gene is to be mutated, the obtained offspring is
$parent = [ga[1,7],\ gb[3,5,10],\ gc[8],\ gd[2,6,9,12],\ ge[\mathbf{10},12]\ gf[13,4]]$ by adding the 10-th software entity to the 5-th gene.

In order to compare data having different domain values the normalization is applied firstly. We have used two methods to normalize the data: decimal scaling for the refactorings *cost* and min-max normalization for the value of the res function.

IX. First Practical Experiments for the *SRSSGARef* Algorithm

The algorithm was run 100 times and the best, worse, and average fitness values were recorded. The parameters used by the evolutionary approach were as follows: mutation probability 0.7 and crossover probability 0.7. Different number of generations and of individuals were used: number of generations 10, 50, 500, and 1000 and number of individuals 20, 50, 100, and 200.

A first experiment run for the *LAN Simulation Problem* source code proposes equal weights (i.e., $\alpha = 0.5$) the refactoring cost application and the transformation impact within the aggregated fitness function.

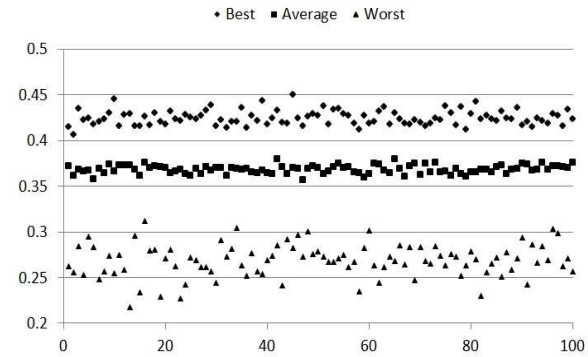
Figure 2 presents the 10 and 1000 generations runs of the fitness function (best, average, and worse) for 100 chromosomes populations, with 11 mutated genes, for *SRSSGARef* Algorithm.

There is a strong competition among chromosomes in order to breed the better individuals. In the 100 individuals populations the competition results in different quality of the best individuals for various runs, from very weak to very good solutions.

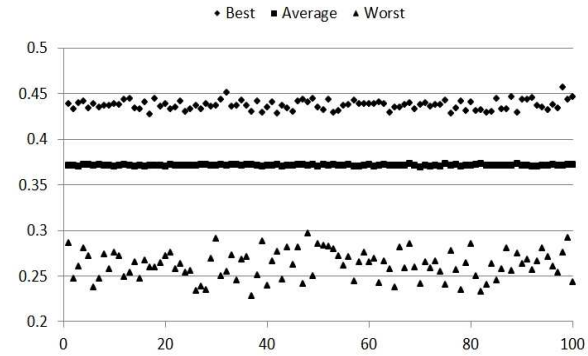
For the refactoring-based solution representation, the runs with 10 evolutions have few very weak solutions, better than 0.3, but they are scattered over $[0.2,\ 0.3]$. The very weak solutions for the runs with 1000 evolutions are grouped in the upper part of $[0.2,\ 0.3]$, but no weak solution has the fitness value better than 0.3. The same behavior was perceived among best and average solutions for the 100 chromosomes populations.

In the context of equal weights for the established objectives, the obtained solutions by the applied algorithm, for 100 individual populations, when $\alpha = 0.5$ are:

- after 10 generations:
  - $bestFitness = 0.4499$:
    - $bestChrom = [[16,\ 11,\ 23,\ 22,\ 21],\ [5],\ [12,\ 16,\ 19,\ 23,\ 11,\ 14,\ 20],\ [11,\ 20,\ 18,\ 23,\ 14],\ [6],\ [20,\ 16,\ 14,\ 15,\ 11,\ 23]]$;



(a) The *SRSSGARef Algorithm*: Experiment with 10 generations and 100 individuals



(b) The *SRSSGARef Algorithm*: Experiment with 1000 generations and 100 individuals

Fig. 2. The fitness function (best, average, and worse) for 100 individuals populations with 10 and 1000 generations runs, with 11 mutated genes, for the *SRSSGARef* Algorithm, for $\alpha = 0.5$

- after 1000 generations:
  - $bestFitness = 0.457$:
    - $bestChrom = [[12,\ 23,\ 15,\ 18,\ 11,\ 20,\ 14],\ [2,\ 1,\ 3,\ 4],\ [13,\ 16,\ 18,\ 23,\ 14,\ 15,\ 11],\ [20,\ 16,\ 19,\ 23],\ [10],\ [12,\ 19,\ 20,\ 11,\ 23,\ 22]]$.

The various runs as number of generations, i.e., 10, 50, 500, and 1000 generations, show the improvement of the best chromosome. For the recorded experiments, the best individual obtained for the *SRSSGARef Algorithm* after 1000 generations of evolution with a 100 chromosomes population, has the fitness value of 0.457. This means in small populations (with fewer individuals) the reduced diversity among chromosomes may induce a harsher struggle compared to large populations (with many chromosomes) where the diversity breeds near quality individuals.

As the Figure 3 shows it, after several generations greater populations produce better individuals (as number and quality) than smaller ones, due to the large population diversity itself.

A. SRSSGARef Algorithm: Impact on the LAN Simulation source code

The best individual obtained when the refactoring cost and impact on software entities have the same relevance allows improving the structure of the class hierarchy. The analysis of the best chromosome partially satisfies the initial strategy (see Section VII).

The current version of the *SRSSGARef Algorithm* lessens criteria constraints of the addressed strategy. Therefore, it

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

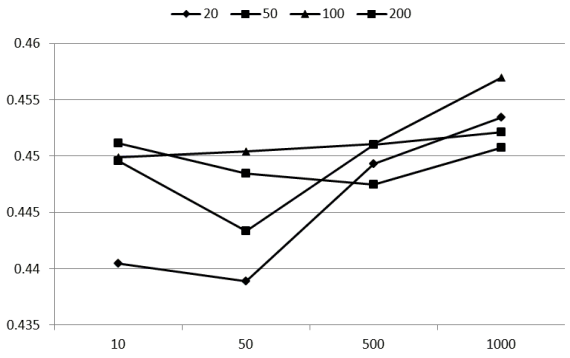


Fig. 3. The fitness value for the best chromosomes within populations with 20, 50, 100, and 200 chromosomes and 10, 50, 500, and 1000 generations evolution, for the *SRSSGARef* Algorithm, with $\alpha = 0.5$

admits as a valid solution chromosomes where the number of applications for the mandatory refactoring *encapsulateField* is at least 1. For the single selected refactorings from the set $RSelected$, the current version of the algorithm accepts the solutions that have at least an additional application of the addressed refactoring, i.e., *pullUpMethod* and *moveMethod*.

X. Conclusions and Future Work

Refactoring may be used in complex software management development processes to achieve several enforced targets. Multiple refactoring aspects of different parts of a heavy working system need increased attention when planning the refactoring order. Refactorings may be organized and prioritized based on goals established by the project management leadership.

The appropriate refactoring selection for various sized software is a stimulating research problem. Software entity dependencies and refactoring dependencies are the basic intriguing elements that drive the research within this domain. This paper addresses the strategy-based refactoring set selection problem.

This paper has advanced the evolutionary-based solution approach for the MOSRSSP. An adapted genetic algorithm has been proposed in order to cope with a weighted-sum objective function for the required solution. Two conflicting objectives have been addressed, as to minimize the *refactoring cost* and to maximize the *refactoring impact* on the affected software entities, following a refactoring application strategy. The run experiments used a balanced weighted fitness function between the cost and the impact on the entities. Further work may be done by investigating the results where refactoring impact or the refactoring cost has a greater weight on the fitness function.

A refactoring-based solution representation was used by the algorithm implementation. The first recorded experiments have lessened the constraints criteria of the refactoring strategy.

Strengthening the refactoring strategy criteria is another task that will be approached in the future. The results achieved here will be compared to the experiments results obtained from the entity-based solution representation for the same algorithm.

The study of the weighted-sum fitness function will be further investigated, by including the strategy-based criteria in the fitness function.

References

[1] M. Bowman, L.C. Briand, and Y. Labiche. Multi-objective genetic algorithm to support class responsibility assignment. In *In Proceedings of the IEEE International Conference on Software Maintenance (ICSM1007)*, pages 124–133, 2007.

[2] A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, 2001.

[3] M.C. Chisăliţă-Creţu, A. Vescan. The Multi-objective Refactoring Selection Problem, in *"Studia Universitatis Babes-Bolyai", Series Informatica, Special Issue KEPT-2009: Knowledge Engineering: Principles and Techniques, July 2-4*, 2009, pp. 249–253.

[4] M.C. Chisăliţă-Creţu. A multi-objective approach for entity refactoring set selection problem, in *"Proceedings of the 2nd International Conference on the Applications of Digital Information and Web Technologies", August 4-6, London, UK*, 2009, pp. 790–795.

[5] M.C. Chisăliţă-Creţu. First results of an evolutionary approach for the entity refactoring set selection problem, in *"Proceedings of the 4th International Conference Interdisciplinarity in Engineering, Nov. 12-13, Târgu Mureş, Romania*, 2009, pp. 303–308.

[6] M.C. Chisăliţă-Creţu. The Refactoring Plan Configuration. A Formal Model, in *"Proceedings of the 5th International Conference on Virtual Learning, Oct. 29-31, Târgu Mureş, Romania*, 2010, pp. 418–424.

[7] M.C. Chisăliţă-Creţu. An evolutionary approach for the entity refactoring set selection problem. *Journal of Information Technology Review, ISSN: 0976-2922, accepted paper*, 2010.

[8] M.C. Chisăliţă-Creţu. The refactoring plan configuration. a formal model. In *The 5th International Conference on virtual Learning (ICVL 2010), October 29-31, 2010, Târgu Mureş, România*, pages 418–424, 2010.

[9] M.C. Chisăliţă-Creţu. *Refactoring in Object-Oriented Modeling*, Todesco, Cluj-Napoca, Romania, 2011.

[10] S. Demeyer, F. Van Rysselberghe, T. Grba, J. Ratzinger, Marinescu R., T. Mens, B. Du Bois, D. Janssens, S. Ducasse, M. Lanza, M. Rieger, H. Gall, and M. El-ramly. The lan simulation: A refactoring teaching example. In *8th Int. Workshop on Principles of Software Evolution (IWPSE'05)*, pages 123–134, 2005.

[11] M. Fowler. *Refactoring Improving the Design of Existing Code*, Addison-Wesley, 1999.

[12] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, 2004.

[13] M. Harman, S. Swift, and K. Mahdavi. An empirical study of the robustness of two module clustering fitness functions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005), Washington DC, USA*, pages 1029–1036, 2005.

[14] M. Harman and L. Tratt. Pareto optimal search based refactoring at the design level. In *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2007)*, pages 1106–1113. ACM Press, 2007.

[15] M. O'Keefe and M. O'Cinneide. Search-based software maintenance. In *In Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR 2006)*, pages 249–260, 2006.

[16] C. Kirsopp, M. Shepperd, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002, W. B. Langdon, E. Cant-Paz, eds., Morgan Kaufmann Publishers, San Francisco, CA, USA*, pages 1367–1374, 2002.

[17] T. Mens, G. Taentzer, O., Runge. Analysing refactoring dependencies using graph transformation, in *Software and System Modeling*, vol. 6 (2007), no. 3, pp. 269–285.

[18] Y. Kim and O.L. deWeck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and Multidisciplinary Optimization*, 29(2):149–158, 2005.

[19] O. Seng, J. Stammel, and D. Burkhart. Search-based determination of refactorings for improving the class structure of object-oriented systems. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 1909–1916. ACM Press, Seattle, Washington, USA, 2006.

[20] A. Vescan and H.F. Pop. The component selection problem as a constraint optimization problem. In *Proceedings of the Work In Progress Session of the 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques (Software Engineering Techniques in Progress), Wroclaw University of Technology, Wroclaw, Poland,*, pages 203–211, 2008.

[21] E. Zitzler, M. Laumanss, and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *Computer Engineering and Networks Laboratory, Technical Report*, 103:5–30, 2001.

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)