BABEŞ-BOLYAI University of Cluj-Napoca
Faculty of Mathematics and Computer Science
Proceedings of the National Symposium ZAC2014
(Zilele Academice Clujene, 2014), p. 32-41

Scenario-based Refactoring Selection

by
Maria-Camelia Chisăliţă-Creţu[1]

Abstract. *In order to improve the internal structure of object-oriented software, refactoring has proved to be a feasible technique. Refactorings may be organized and goal-based prioritized shaping thus a refactoring strategy. The paper presents the formal definition of the Multi-objective Scenario-based Refactoring Set Selection Problem (MOSRSSP) by treating the cost constraint and the refactoring impact as objectives of a weighted-sum fitness function.*

Keywords: software engineering, refactoring, genetic algorithm, weighted-sum method

1. 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 behavior-preserving transformations, each improving a certain aspect of the system [6].

Refactorings may be organized and prioritized based on goals established by the project management leadership. The MOSRSSP definition is based on the Refactoring Set Selection Problem (RSSP) [3]. Therefore, the SRSSP is the refactoring set selection problem that combines

[1] Babeş-Bolyai University, Cluj-Napoca, *cretu@cs.ubbcluj.ro*

multiple strategy criteria in order to find the most appropriate set of refactorings.

The rest of the paper is organized as follows. Section 2 presents some close related work and the motivation for the MOSRSSP. The formal definition for the MOSRSSP is presented in Section 3. The Local Area Network (LAN) Simulation source code used by approach is discussed in Section 4. The proposed approach, several details on the genetic operators of the applied algorithm and the first results are presented and discussed in Section 5. The paper ends with conclusions and future work.

2. Related work

A closely related previous work to refactoring selection problems is the Next Release Problem (NRP) studied by several authors [1, 7], 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.

More recent work on search based refactoring problems [2, 3] in SBSE have defined the General Refactoring Selection Problem (GRSP), used to refine the Multi-Objective Refactoring Set Selection Problem (MORSSP) [3] and the Multi-Objective Refactoring Sequence Selection Problem (MORSqSP) [4].

Our approach is similar to those presented in [8]. 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.

A refactoring scenario 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. There are several problems faced, emphasing diferent aspects of a complex refactoring process, as:a large number of refactorings advanced; diferent types of dependencies among the affected software entities and applied refactorings, e.g., an inherited method from a base class is called within another method of a derived class;

- a specific refactoring priority for each software entity.

3. Scenario-based Refactoring Set Selection Problem

The Scenario-based Refactoring Set Selection Problem (SRSSP) is mainly based on the Refactoring Set Selection Problem (RSSP) fully formalized in [2]. 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.

Input Data

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 similar to the ones described in [2]. A set of relevant chosen refactorings that may be applied to the software entities of SE is gathered up through *SR*. The ra mapping sets the applicability for each refactoring from the chosen set of refactorings SR on the set of software entities SE [2].

The set of refactoring dependencies *SRD*, together with the mapping rd that highlights the dependencies among different refactorings when applied to the same software entity are stated in [2].

The effort involved by each transformation is converted to cost, described by rc mapping [2]. Changes made to each software entity $e_i, i=\overline{1,m}$, by applying the refactoring $r_l, 1 \le l \le t$, are stated by the $effect$ mapping defined in [3]. The overall impact of applying a refactoring $r_l, 1 \le l \le t$, to each software entity $e_i, i=\overline{1,m}$, is defined as: $res : SR \to Z$ in [2].

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* $r \in SR$ [2]. 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 $\overline{r_l\ e_i} = (r_l\ e_i)$

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

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

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 $RType = \{Mandatory, Optional, Selected\}$ be the set of possible refactoring types. The mapping $rtype$ associates a type to each refactoring from SR as follows: $rtype : SR \to RType$,

$$rtype(r) = \begin{cases} M, \text{ if } r \text{ is applied mandatory} \\ O, \text{ if } r \text{ is applied optional} \\ S, \text{ if } r \in \{r_1, \ldots, r_q\}, \ 0 \le q \le t \end{cases}.$$

A second criterion considered by the development team may refer the level of the affected entity when refactoring. Let $RLevel = \{Attribute, Method, Class\}$ 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: $rlevel : SR \to RLevel$,

$$rlevel(r) = \begin{cases} a, \text{ if } r \text{ is applied to attributes} \\ m, \text{ if } r \text{ is applied to methods} \\ c, \text{ if } r \text{ is applied to classes} \end{cases}.$$

Output Data

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.

The first objective function to for the MOSRSSP is 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 $\vec{r} = (r_1, ..., r_t)$.

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

The final fitness function for MOSRSSP is defined by aggregating the two objectives and may be written as:

(1) $F(\vec{r}) = \alpha \cdot f_1(\vec{r}) + (1-\alpha) \cdot f_2(\vec{r})$, where $0 \le \alpha \le 1$.

Let $DS = REPSet$ be the decision domain for the MOSRSSP and $\vec{x} = (\overline{r_1\ e_1}, \overline{r_2\ e_2}, \ldots, \overline{r_s\ e_s})$, where $e_u \in SE$, $r_u \in SR$, $1 \le u \le s$, $s \in N$, $\vec{x} \subseteq DS$ a decision variable. The MOSRSSP is the problem of finding a decision vector $\vec{x} = (\overline{r_1\ e_1}, \overline{r_2\ e_2}, \ldots, \overline{r_s\ e_s})$, such that:

- the following objectives are optimized:

– the overall refactoring cost is minimized (rc) [2] and the overall refactoring impact on software entities is maximized (res) [2].

- the following constraints are satisfied:

– software entity dependencies (ed) [2] and refactoring dependencies (rd) [2].

- the addressed strategy-based criteria are met:

– $RMandatory = \{r_1, ..., r_{rm}\}$ is the set of mandatory refactorings, where $r_1, ..., r_{rm} \in SR$, $0 \le rm \le t$;

– $ROptional = \{r_1, ..., r_{ro}\}$ is the set of optional refactorings, where $r_1, ..., r_{ro} \in SR$, $0 \le ro \le t$;

– $RSelect = \{r_1, ..., r_{rs}\}$ is the set of single selected refactorings, where $r_1, ..., r_{rs} \in SR$, $0 \le rs \le t$;

$-1 \le rm + ro + rs \le t$, $RMandatory \cap ROptional \cap RSelect = \phi$;

– conditions on the number of applied refactorings on attribute, method, and class levels are met.

4. 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 [2]. Figure 1 shows the class diagram of the studied source code. It contains 5 classes with 5 attributes and 13 methods, constructors included.

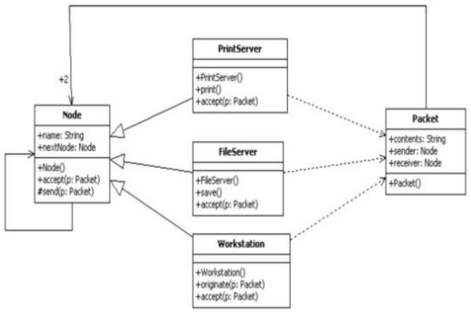


Figure 1. Class diagram for LAN Simulation

The current version of the source code lacks of hiding information for attributes since they are directly accessed 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 $SR = \{r_1,...,r_6\}$in the following.

The values of the res function for each refactoring are: 0.4, 0.49, 0.63, 0.56, 0.8, and 0.2. The full input data table is included in [3]. 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 the entity $m_i, i = \overline{1,13}$, r_4 will not be selected to be applied to the same entity;

• $1 \leq |RMandatory| + |ROptional| + |RSelect| \leq 6$, $RMandatory \cap ROptional \cap RSelect = \phi$;

• 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.

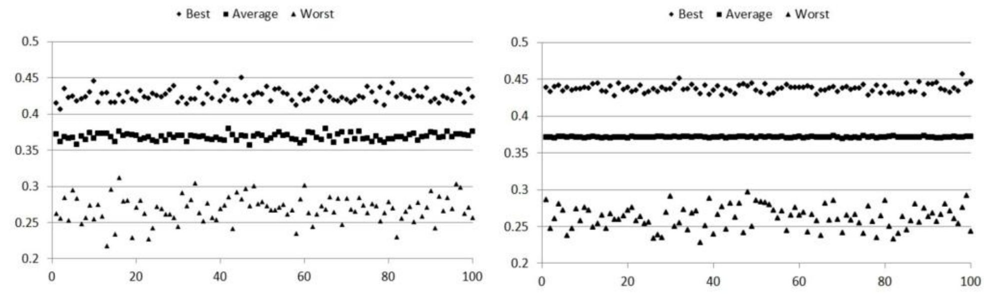
5. First Results of the Evolutionary Approach

An adapted genetic algorithm to the context of the investigated problem, with weighted sum fitness function, similar to the one in [3, 4], 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*. Crossover and mutation operators are used by the genetic algorithm as well, being fully described in [2].

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 numbers of generations and of individuals were used: number of generations 10, 50, 500, and 1000 and number of individuals 20, 50, 100, and 200. Equal weights (i.e., $\alpha = 0.5$) on the refactoring cost application and the transformation impact was investigated.

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.



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

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

Figure 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 α = 0.5

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

• after 10 generations, the best fitness value = 0.4499:

• best chromosome = [[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]];

• after 1000 generations, the best fitness value= 0.457:

• best chromosome = [[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]].

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

6. Conclusions

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

Further work may be done by investigating the results where refactoring impact or the refactoring cost has a greater weight on the fitness function. 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.

References

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

[2] M.C. Chisalita-Cretu, A. Vescan. *The Multi-objective Refactoring Selection Problem*, in ”Studia Universitatis Babeş-Bolyai”, Series Informatica, Special Issue KEPT-2009: Knowledge Engineering: Principles and Techniques, July 2-4, 2009, pp. 249 - 253.

[3] M.C. Chisalita-Cretu. *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.

[4] M.C. Chisalita-Cretu. *An evolutionary approach for the entity refactoring set selection problem*. Journal of Information Technology Review, ISSN: 0976-2922, pp.107 - 118, 2010.

[5] 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), pp. 123 - 134, 2005.

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

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

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