Developing a Multi-objective Decision Approach to Select Source-Code Improving Transformations [*]

Ladan Tahvildari and Kostas Kontogiannis
Dept. of Electrical and Computer Eng.
University of Waterloo
Waterloo, Ontario
Canada, N2L 3G1
{ltahvild,kostas}@swen.uwaterloo.ca

Abstract

Our previous work on improving the quality of object-oriented legacy systems through re-engineering proposed a software transformation framework based on soft-goal interdependency graphs [11]. We considered a class of transformations where a program is transformed into another program in the same language (source-to-source transformations) and that the two programs may differ in specific qualities such as performance and maintainability. This paper defines a decision making process that determines a list of source-code improving transformations among several applicable transformations. The decision-making process is developed on a multi-objective decision analysis technique. This type of technique is necessary as there are a number of different, and sometimes conflicting, criterion among non-functional requirements. For the migrant system, the proposed approach uses heuristic estimates to guide the discovery process.

1 Introduction

As it is known, most non-functional requirements involve multiple and conflicting objectives which relate to various alternative design decisions [4]. The task of adequately modeling and analyzing such problems has been the subject of the multi-attribute utility theory [5]. However, accurately and confidently describing preference in the context of multiple objectives using a scalar criterion can be difficult. In order to avoid this potential difficulty, we can search for the set so-called *non-dominated* solutions [9] which represent the multiple-objective approach to problem solving.

A transformation T is said to be non-dominated among a set of transformations if there is no other transformations in the set that is at least as "good" as T with respect to all the non-functional objectives and is strictly "better" than T with respect to at least one of the non-functional objectives, where "good" and "better" are defined in terms of an impact-valued criterion associated with each of the multiple non-functional objectives under consideration. Thus, instead of seeking a single optimal solution, one seeks the set of non-dominated solutions. Determination of the most preferred alternative from the set of non-dominated alternatives may depends on the nature of the target systems.

This paper is organized as follows. Section 2 summarizes the concept of Soft-Goal Interdependency Graphs and presents a model for a SIG representation. While Section 3 discusses a soft-goal evaluation algorithm based on a set of heuristic functions, Section 4 discusses a proposed multi-objective decision approach whose objective is to identify the set of all non-dominated solution graphs, using sets of vector impact-valued heuristics estimates. them on some case studies. Finally, Section 5 provides the conclusion and outlines directions for further research.

2 Soft-Goal Interdependency Graphs

The re-engineering of object-oriented legacy systems requires a comprehensive framework to relate software transformations with specific requirements for the new target migrant system. We refer to this approach as "*Quality-Driven Object-Oriented Re-engineering*" [10, 12] which exploits the synergy between *non-functional requirements*, *software architecture*, and *reverse engineering*, and adopts the NFR framework proposed in [4]. The NFR Framework introduces the concept of *soft-goals* whose level of success is evaluated by the success of other soft sub-goals. The *soft-goal interdependency graphs* (SIGs) have been proposed for supporting the systematic, goal oriented process of architectural design [4]. The leafs of the soft-goal interdependency graph represent transformations which fulfill or contribute posi-

[*]This work was funded by the IBM Canada Ltd. Laboratory, Center for Advanced Studies in Toronto.

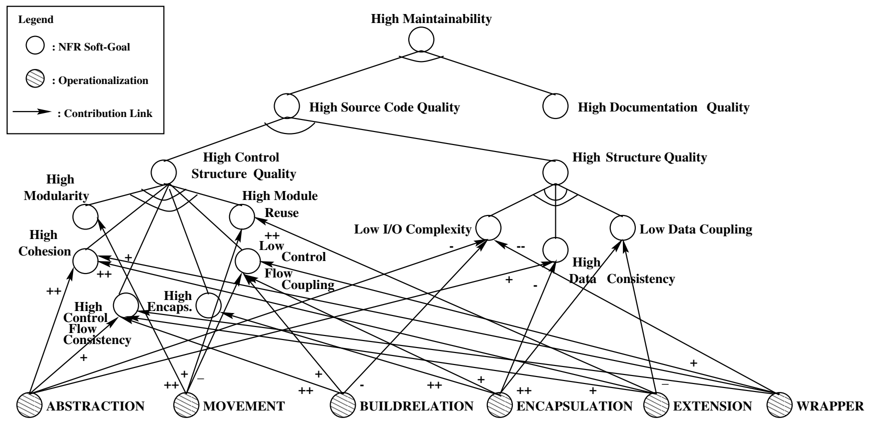

Figure 1. Relating Source-Code Improving Transformations to Maintainability Soft-Goal Graph.

tively/negatively to soft-goals above them. Figure 1 shows portions of the soft-goal interdependency graph for the maintainability non-functional requirement as presented in [11].

As shown in Figure 1, *AND* and *OR* contribution operators relate a group of offsprings to a parent. To keep track of the information regarding these two contribution operators, we will build an adjacency matrix, namely *Soft-Goal Adjacency Matrix*, which will be elaborated in Section 2.1. *MAKE* and *BREAK* provide sufficient support, *MAKE* being positive and *BREAK* being negative. *MAKE* is defined as: if the offspring is satisficed, then the parent is satisficeable. *BREAK* is defined as: if the offspring is satisficed, then the parent is deniable. *HELP* and *HURT* provide partial support, *HELP* being positive, and *HURT* being negative. *HURT* is defined as: if the offspring is denied, then the parent is satisficeable. *HELP* is defined as: if the offspring is denied then the parent is deniable. To keep track of the information regarding these contribution operators, we will build an incidence matrix, namely *Transformation Impact Matrix*, which will be elaborated in Section 2.2.

2.1 Soft-Goal Adjacency Matrix (SAM)

Let n_1 and n_2 be vertices of a SIG. If n_1 and n_2 are joined by an arc e with any of AND or OR contribution operator, then n_1 and n_2 are said to be *adjacent* with the defined rule. If the arc e is directed from n_1 to n_2, then the arc e is said to be *incident from* n_1 and *incident to* n_2.

Let D to be a SIG in digraph form with n vertices or soft-goal nodes, $N = \{d_1, d_2, ..., d_n\}$. The simplest graph representation scheme uses an $n \times n$ matrix SAM (Soft-Goal Adjacency Matrix) of &'s, |'s, and 0's given by:

$$SAM_{i,j} = \begin{cases} \&, & \text{when } AND(d_i, \{d_j\}) \\ |, & \text{when } OR(d_i, \{d_j\}) \\ 0, & \text{otherwise} \end{cases}$$

that is, the (i, j)th element of the matrix is not equal to 0 only if $d_i \longrightarrow d_j$ is an edge in D with a contribution operator. Clearly, the number of entries which is not equal to zero in the SAM is equal to the number of edges in a soft-goal interdependency graph. One advantage of using an adjacency matrix is that it is easy to determine the sets of edges with their contribution operators emanating from a given vertex. Each non-zero entry in the ith row corresponds to an edge with its contribution operator that emanates from vertex d_i. Conversely, each non-zero entry in the ith column corresponds to an edge incident on vertex d_i. We have developed an algorithm to generate an adjacency matrix for n soft-goals. We call it *Generate Adjacency Matrix* (GAM).

2.2 Transformation Impact Matrix (TIM)

As shown in Figure 1, there are two types of nodes in a SIG, namely *NFR soft-goal* nodes and *transformation operationalization* nodes. In practice, a transformation node is considered for the purpose of implementing a subset of the NFR soft-goals. Thus, each NFR soft-goal is associated with a set of transformations that have been associated to it according to what features of the source code a transformation affects, and according to the associating of these features with the specific allocating a soft-goal node is representing. The NFR soft-goal allocation is not necessarily one-to-one – that is, a single soft-goal may be associated to more than one transformation.

Let n_1 be a soft-goal node and n_2 a transformation or operationalization node. If n_1 and n_2 are joined by an arc e with any of $MAKE$, $HELP$, $HURT$, or $BREAK$ contribution operator, then n_1 and n_2 are said to be *adjacent* with the defined rule. The arc e which is directed from n_2 to n_1 is said to be *incident from* n_2 and *incident to* n_1. Let S be a set nodes, $S = \{s_1, s_2, ..., s_n\}$, representing soft-goals and T is a of set operationalization nodes, $T = \{t_1, t_2, ..., t_m\}$, representing transformations. The simplest impact representation scheme uses an $m \times n$ matrix TIM (Transformation Impact Matrix) of ++'s, +'s, −−'s, −'s, and 0's given by:

$$TIM_{i,j} = \begin{cases} ++, & \text{when } (t_i, s_j, MAKE) \\ +, & \text{when } (t_i, s_j, HELP) \\ -, & \text{when } (t_i, s_j, HURT) \\ --, & \text{when } (t_i, s_j, BREAK) \\ 0, & \text{otherwise} \end{cases}$$

that is, the (i, j)th element of the matrix is not equal to 0 only if $t_i \longrightarrow s_j$ is an edge in D with a contribution operator. Now, it is needed to define another algorithm, namely *Generate Impact Matrix*, to incorporate soft-goals, transformations, and their relationships based on our definition of transformation impact matrix (TIM).

3 Soft-Goal Evaluation Procedure

In our re-engineering framework [12], we need to find a list of potential transformations among several possible ones [11] that can achieve the desired non-functional requirements. Given a SIG, one can determine whether each soft-goal or interdependency is satisficed. This is done through the assignment of a *label*. Using the notion of *satisficeable* and *deniable*, a *catalogue of label values* for a soft-goal or interdependency in the graph is presented for depicting the desired *objectives*: i) *satisficed* ($\surd$) if it is satisficeable and not deniable, ii) *weakly satisficed* ($\mathcal{W}^+$) if it is representing inclusive positive support for a parent, iii) *denied* ($\times$) if it is deniable but not satisficeable, iv) *weakly denied* ($\mathcal{W}^-$) if it is representing inclusive negative support for a parent, v) *conflicting* ($\natural$) if it is both satisficeable and deniable, vi) *undetermined* ($\mathcal{U}$) if it is neither satisficeable nor deniable.

Then, the Soft-Goal Evaluation (SGE) algorithm illustrated in Figure 2 consists of two basic steps. For each soft-goal or interdependency in a SIG, the procedure first computes the *individual impact* of each source-code improving transformation which we call it *Generate Individual Label (GIL)*. Secondly, the individual impacts of all interdependencies are combined into a single label which we call it *Combine Generated Labels (CGL)*. Details of the each step of the procedure will be elaborated in the following sections.

3.1 Impact Analysis: Generate Individual Label (GIL)

In the first step, we determine the "individual impact" of an offspring's contribution towards its parent. For AND and OR contributions, we treat all of the offsprings as one group, with a single "individual impact", defines as follows:

- If $offspring_1$ AND ... AND $offsping_n$ SATISFICE parent then
  $$label(parent) = \min_i(label(offpring_i))$$

```
Algorithm $SGE(\mathcal{AC}, \mathcal{DSG}, \mathcal{TIM}, \mathcal{SAM}, \mathcal{VIC}) =$
Input:
    $\mathcal{AC}$: A set of classes in an object-oriented legacy system.
    $\mathcal{DSG}$: A List of Desired Soft-Goals.
    $\mathcal{TIM}$: Transformation Impact Matrix.
    $\mathcal{SAM}$: Soft-Goal Adjacency Matrix.
Output:
    $\mathcal{VIC}$: A set of Vector-Impact Cost for $\mathcal{AC}$.
Variables:
    $\mathcal{COM}_i$: Complexity Metrics Vector for class $i$.
    $\mathcal{COH}_i$: Cohesion Metrics Vector for class $i$.
    $\mathcal{COU}_i$: Coupling Metrics Vector for class $i$.
Method:
 1. for each class $\mathcal{C}$ in $\mathcal{AC}$ do begin
 2.   if $\mathcal{C}$ is a deteriorated class based on
      $d_\mathcal{C}(\mathcal{COU}_\mathcal{C}, \mathcal{COM}_\mathcal{C})$ and $\mathcal{COU}_\mathcal{C}$ then begin
 3.     for each applicable transformation $\mathcal{T}_i$ do begin
 4.       find the satisfaction level for each $\mathcal{DSG}$;
 5.       $\mathcal{GIL}$ $(\mathcal{SAM}, \mathcal{TIM}, \mathcal{T}_i)$;
 6.       $\mathcal{CGL}$ $(\mathcal{SAM}, \mathcal{TIM}, \mathcal{T}_i)$;
 7.       combine all satisfaction level as $\mathcal{VIC}$;
 8.     end-for
 9.   end-if
10. end-for
```

Figure 2. Soft-Goal Evaluation Algorithm.

- If $offspring_1$ OR ... OR $offsping_n$ SATISFICE parent then
  $$label(parent) = \max_i(label(offpring_i))$$

where the labels are ordered in increasing order as follows:

$$\times \leq \mathcal{U} \approx \natural \leq \surd$$

Now, we need to consider the individual impact for some of the other contribution types. There are some rules for propagation of labels from an offspring to its parent which are depicted in Table 1. As shown in table, looking left-to-right at the individual impacts for satisficed offspring (labeled $\surd$), there is a left-to-right progression through the label catalogue. For individual impacts of denied offspring (labeled $\times$), there is a right-to-left progression through the label catalogue. Also, it may be noted that Table 1 does not have entries for other offspring, such as $\mathcal{W}^+$ and $\mathcal{W}^-$. In a nutshell, this step generates labels for a parent given the labels of its offsprings. As we discuss below, the second step, namely the Combine Generated Labels (CGL), amalgamates such values in a single label. We will also discuss a possible extension to retain such values as outputs of the CGL step, and inputs to the GIL step.

	Label of Contribution Successors				
Label of an Offspring	BREAK	HURT	?	HELP	MAKE
$\times$	$\mathcal{W}^+$	$\mathcal{W}^+$	$\mathcal{U}$	$\mathcal{W}^-$	$\times$
$\natural$	$\natural$	$\natural$	$\mathcal{U}$	$\natural$	$\natural$
$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$
$\surd$	$\times$	$\mathcal{W}^-$	$\mathcal{U}$	$\mathcal{W}^+$	$\surd$

Table 1. Individual Impact of an Offspring.

3.2 Objective Analysis: Combine Generated Labels (CGL)

Once all contributing labels have been generated for a given parent, the second step of the evaluation procedure combines them into a single label. The labels that contribute to a parent are placed in a "bag" (a collection which allows duplicate entries, unlike set). The possible label values in the bag are $\times$, $\mathcal{W}^-$, $\natural$, $\mathcal{U}$, $\mathcal{W}^+$, and $\surd$. A bag is used because duplicate labels are useful as for the instance, where several positive supporting contributions indicated by several $\mathcal{W}^+$ labels may be combined into a $\surd$ label.

The $\mathcal{W}^+$ and $\mathcal{W}^-$ labels in the bag are first combined into one or more $\surd$, $\times$, $\natural$. Typically, $\mathcal{W}^+$ values alone in a bag would result in $\surd$ or $\mathcal{U}$, and $\mathcal{W}^-$ values alone would result in $\times$ or $\mathcal{U}$. Moreover, $\mathcal{U}$ labels depend on the nature of the target system. A mixture of $\mathcal{W}^+$ and $\mathcal{W}^-$ values would typically resulting $\surd$, $\times$ or $\natural$. The resulting set of labels is then combined into a single one, by choosing the minimal label of the bag, with a label ordering:

$$\natural \leq \mathcal{U} \leq \times \approx \surd$$

In this way, we could consider modifying the evaluation procedure so that values such as $\mathcal{W}^+$ and $\mathcal{W}^-$ could appear as outputs of this step (CGL) instead of being eliminated. If this is the case, this step (CGL) would use the following ordering:

$$\natural \leq \mathcal{U} \leq \mathcal{W}^- \approx \mathcal{W}^+ \leq \times \approx \surd$$

4 Multi-Objective Decision Approach

After applying the evaluation procedure, we have a list of potential transformations with their impact cost on each desired soft-goal. In this step, we need to select a sequence of transformations among several possible ones [11], that may yield the desired properties for the migrant system. Such a problem can be formulated as a search in the space of alternative transformations. Most of the efforts in this research directions concentrated on the definition of transformations and their implementation [1, 2, 3, 7, 8]. To the best of our knowledge, there is not much effort on the automatic detection of the situation where a sequence of these source-code improving transformations can apply in an object-oriented legacy system.

Let $G = (N(G), A(G))$ to be a directed graph, $N(G)$ to be the set of nodes in G that are transformations, and $A(G)$ be the set of $arcs$ in G. Thus, $A(G) \subseteq N(G) \times N(G)$. For each arc $\alpha = (n_i, n_j)$, node n_j is referred to as the *head* of α and n_i as the *tail* of α. Furthermore, each node n_i in the graph is labeled with a vector of impact values:

$$K(n_i) = (impact_1(n_i), ..., impact_q(n_i)) \qquad (1)$$

where $impact_u(n_i) \quad , u = 1, 2, ..., q$ relates to the impact of associated transformations on the corresponding non-functional requirement u. $impact_u(n)$ is defined as $\frac{Cost}{Benefit}$ which needs to be minimized by maximizing $Benefit$ and minimizing $Cost$ as will be elaborated further.

Any transformation in the QDR Framework [12] performs operations such as: i) comprehending/understanding source code for the application of transformation that can be done by using *investigation functions*, ii) adding and/or deleting source code entities, and iii) modifying source code entities. By these considerations, the $Cost$ for applying a transformation t_i in QDR Framework [12] can be measured as follows:

$$\begin{aligned} Cost(t_i) &= \# \, of \; points \; of \; investigation \; + \\ &\quad \# \, of \; source \; code \; features \; added \; + \\ &\quad \# \, of \; source \; code \; features \; deleted \; + \\ &\quad \# \, of \; source \; code \; features \; modified \qquad (2) \end{aligned}$$

In order to measure the $Benefit$ for a potential transformation to be applied, a summary of proposed metrics needs to be evaluated both before and after applying the transformations. Let $M_b = \langle m_1, m_2, ..., m_n \rangle$, $(m_i \neq 0)$, be a vector of metrics showing different features of the source code before applying a candidate transformation and $M_a = \langle m'_1, m'_2, ..., m'_n \rangle$, $(m'_i \neq 0)$, be a vector of metrics showing the same features of the source code after applying the transformation, then for maximizing $Benefit$ we need to find the difference which is calculated as follows:

$$(M_a - M_b) > 0, \quad \forall \, t_i, \; (m'_i - m_i) > 0$$

For the metrics that their decreasing values provide more benefit, we need to consider $\frac{1}{m_i}$ instead of m_i. Then $Benefit$ can be defined as:

$$Benefit(n) = \left(\frac{m'_1 - m_1}{min(m'_1, m_1)} + ... + \frac{m'_n - m_n}{min(m'_n, m_n)}\right)/n$$

Then, the search problem can be represented by a unique node in G called start node $s \in N(G)$ and $\Gamma \leq N(G)$ a set of goal nodes. We consider this search problem to be an instance of the class of multi-objective search problems [5]. We can therefore formulate the selection of source-code improving transformations as a multi-objective graph search algorithm. Such a multi-objective graph search approach identifies the set of all non-dominated solution graphs in SIGs for selecting the source-code improving transformations. This algorithm can be considered as an adaptation of A^* algorithm [6]. The description of the proposed algorithm is depicted in Figures 3.

5 Conclusion

This paper presents a multi-objective decision approach to identify the set of all non-dominated solution graphs in SIGs for selecting the source-code improving transformations. Three types of heuristics information are used in the approach: 1) a node evaluation heuristic (GIL), 2) a node selection heuristic (STP), and 3) a solution based graph selection heuristics (CGL). This multi-objective decision approach can be used to prevent loss of maintainability of a legacy system or restore it through re-engineering.

A possible direction for future research concerns the qualification of trade-offs weights to direct the multi-objective search. This results in a parametric multi-objective formulation that should be much more tractable than strictly multi-objective form described in this paper.

References

[1] P. Antonini, G. Canfora, and A. Cimitile. Re-engineering legacy systems to meet quality requirements: An experience report. In *Proceedings of the IEEE ICSM*, pages 146–153, Sept. 1994.

[2] I. Baxter and C. Pidgeon. Software change through design maintenance. In *Proceedings of the IEEE ICSM*, pages 250–259, Oct. 1997.

[3] G. Canfora, A. R. Fasolino, and M. Tortorella. Towards reengineering in reuse reengineering processes. In *Proceedings of the ICSM*, pages 147–156, Nov. 1998.

[4] L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.

[5] R. L. Keeney and H. Raiffa. *Decision with Multiple Objectives*. John Wiley and Sons, 1976.

[6] N. J. Nilsson. *Principles of Artificial Intelligence*. Tiago Publishing Company, 1980.

[7] C. W. Pidgeon. *Analysing Decision Making in Software Design*. PhD thesis, University of Californian at Irvine, 1990.

[8] H. M. Sneed. Transforming procedural program structures to object-oriented class structures for the purpose of populating a common software repository. In *Proceedings of the IEEE International Conference on Software Maintenance*, page 286, Oct. 2002.

[9] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application*. John Wiley and Sons, 1986.

[10] L. Tahvildari. *Quality-Driven Object-Oriented Re-engineering Framework*. PhD thesis, Department of E&CE, University of Waterloo, Canada, 2003.

[11] L. Tahvildari and K. Kontogiannis. A software transformation framework for quality-driven object-oriented re-engineering. In *Proceedings of the IEEE ICSM*, pages 596–605, Oct. 2002.

[12] L. Tahvildari, K. Kontogiannis, and J. Mylopoulos. Quality-driven software re-engineering. *Journal of Systems and Software, Special Issue on: Software Architecture - Engineering Quality Attributes*, 66(3):225–239, June 2003.

```
Algorithm $STP(\mathcal{PSP}, \mathcal{GN}, \mathcal{NDS}, \mathcal{AC}, \mathcal{DSG}, \mathcal{TIM}, \mathcal{SAM}) =$
Input:
    $\mathcal{PSP}$: A finite directed graph with transformations as nodes.
    $\mathcal{GN}$: A finite set of goal paths in the graph.
    $\mathcal{AC}$: A set of classes in an object-oriented legacy system.
    $\mathcal{DSG}$: A List of Desired Soft-Goals.
    $\mathcal{TIM}$: Transformation Impact Matrix.
    $\mathcal{SAM}$: Soft-Goal Adjacency Matrix.
Output:
    $\mathcal{NDS}$: A set of all non-dominated solution graphs.
Method:
 1. Initialize $OPEN$ to a set contains start node from $\mathcal{PSP}$;
 2. $SGE(\mathcal{AC}, \mathcal{DSG}, \mathcal{TIM}, \mathcal{SAM}, \mathcal{VIC})$;
 3. while ($\mathcal{VIC} \notin SOLUTION\_IMPACTS$ & $GN \in OPEN$) do begin
 4. if $ND = \emptyset$ then
 5. set $SOLUTION\_GOALS$ to $s$; append solution path impacts to $SOLUTION\_IMPACTS$ ;append $\mathcal{VIC}$ to $LABEL$ sets; append solution paths to $SOLUTION$;
 6. else Choose a node $n$ from $ND$ based on $\mathcal{CGL}$; remove $n$ from $OPEN$; append $n$ to $CLOSED$;
 7. Update accrued impacts $LABEL$ based on $n$;
 8. if $n \in GN$, then
 9. Append $n$ to $SOLUTION\_GOALS$; append $n_{impact}$ to $SOLUTION\_IMPACTS$; remove any dominated members of $SOLUTION\_IMPACTS$; Go to Step 19.
10. Generate the successors of $n$;
11. if $n$ has no successors then Go to Step 19;
12. else for all successors $n' \in n$, do begin
13. if $n'$ is a newly generated node, then
14. Establish a back-pointer from $n'$ to $n$; set $G(n') = LABEL(n', n)$; compute node selection values, $F(n')$, using $G(n')$; compute heuristic function values at $n'$, $H(n')$; append $n'$ to $OPEN$;
15. else while $n' \neq \emptyset$ do begin
16. Append ${n_{impact}}'$ to $LABEL(n', n)$; update $NDS$ based on $G(n')$; if $n' \notin G(n')$, then begin delete associated path with $n'$ from $LABEL(n', n)$; if $n' \in CLOSED$ then add $n'$ to $OPEN$;
17. end-while
18. end-for-all
19. Increment $Iteration$ counter by 1;
20. end-while
```

Figure 3. Selection Transformation Path Algorithm.