A MODEL AND ITS ALGORITHM FOR SOFTWARE REUSE OPTIMIZATION PROBLEM WITH SIMULTANEOUS RELIABILITY AND COST CONSIDERATION

ZHIQIAO WU\textsuperscript{1}, JIAFU TANG\textsuperscript{1}, CHUN KIT KWONG\textsuperscript{2} AND CHING YUEN CHAN\textsuperscript{2}

\textsuperscript{1}Department of Systems Engineering
Key Lab of Integrated Automation of Process Industry of MOE
Northeastern University

No. 11, Lane 3, Wenhua Road, Heping District, Shenyang 110819, P. R. China
wuzhiqiao@hotmail.com; jftang@mail.neu.edu.cn

\textsuperscript{2}Department of Industrial and Systems Engineering
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, P. R. China
{mfckkong; mfcychan}@polyu.edu.hk

Received January 2010; revised May 2010

ABSTRACT. In this paper, a model is proposed where the cost and reliability of activities of reuse-based development can be predicted or measured, and all possible reuse scenarios can be analyzed and compared to support the selection of alternative scenarios. A scheme of typical reuse mode is designed from the perspectives of application engineering and domain engineering activities. According to the scheme, six reuse modes are addressed from the sequences of activities. Alternative industry reuse scenarios can be derived from the modes. An optimization model is proposed that can assist decision-makers in selecting a reuse scenario for minimizing cost, maximizing reliability and satisfying system requirements. To solve efficiently the bi-objective 0-1 integer programming involved in the model, a new algorithm is presented that can find the entire set of efficient solutions.

Keywords: Software reuse, Reuse models, Optimization models, Bi-objective programming problems, Efficient solution

1. Introduction. In recent years, a significant number of efforts have been spent on techniques and methods to define the “grain” of software reuse components, such as the recent research on Software Product Line (SPL) \cite{1,2}. However, selecting an appropriate set of components to satisfy the functional and non-functional requirements of a system is difficult. The task is dependent on the experience, especially on the non-functional requirements such as cost and reliability. Non-functional requirements are important to develop a successful software application. As shown by Tomer \cite{3}, software reuse is not merely a technical issue. The organizational challenges of software reuse outweigh the technical questions. The functional properties of component assemblies are usually easier to model and validate compared with non-functional properties. The reuse activities in software design should be systematically measured or estimated, and alternative reuse scenarios should be evaluated and compared for effective support of the whole reuse process. However, only a few methodologies were developed to support the automation of an assembly process on the non-functional requirements. Many selection models of reuse strategy are aimed to address the issue of systematically increasing the efficiency of software reuse \cite{4-7}. Very few studies provide a method for precisely analyzing and evaluating alternative scenarios of various reuses. Therefore, a systematic framework of
software development process to support analyzing, comparing and selecting particular development scenarios and to guide the component implementation is required [8].

One of the major constraints of software development projects in the industry is the limited budget. The best solution for software development may not be feasible due to high costs. Some software systems are adapted to human life. Thus, a trifling fault of the components of large-scale reuse components would largely affect the functions of the entire software systems. Therefore, the reliability of software component and its reuse implementation is important. Thus, nearly all the optimal methods developed in previous research focus on weighing the system cost or reliability. The objective functions considered in the models can be categorized into two classes: maximizing reliability [9-11] and cost-minimization [12-14]. However, previous works did not provide methods that consider the two objective functions simultaneously. In fact, the relationship between system cost and reliability is complex and nonlinear. The current research attempts to deal with the selection problem of reuse scenario with consideration of cost and reliability in a reuse-based software development.

Although no previous studies directly investigated the aforementioned issue, related studies were conducted by Tomer. These studies showed that the major sticking point in software reuse is not the implementation technique but the outstanding organization in the entire development process [3]. Main discussions focused on reuse strategy decision for supporting the reuse activity and proposed a variety of formal approaches [4]. However, all the approaches of the previous studies only provide outlines in selecting reuse strategy based on qualitative analysis, and only a few support automation of the task in a quantitative way. As an extended work of the previous studies, a systematic definition of various reuse scenarios that can be used to weigh the different alternatives for scenario choice is developed in this study.

An important breakthrough in this field is the introduction of optimization techniques [9,15], which quickly drew the attention of scholars [12]. Jung proposed a 0-1 knapsack model to select a set of software requirements to yield maximum value while minimizing costs. The concept of value can be interpreted as an implementation priority [15]. Considering the reliability growth as a function of cost, Berman and Cutler proposed an optimization model to maximize the reliability of an assembly of assignments under a budget constraint [9]. Thereafter, a number of studies considered the setback as a cost-minimization problem within a reliability constraint. Cortellessa developed a model that treats component selection as a cost-minimization problem under delivery time and quality constraints [13]. Cortellessa further introduced goal-oriented requirement engineering, which describes the satisfaction level of system requirements in the requirement analysis process [16]. For this reason, this research adopts the approach of Cortellessa to ensure the satisfaction of the system requirements under constraints.

In this paper, a model for identifying possible reuse activities and scenarios involved in reuse process is proposed. The scheme of a typical reuse mode is described from the perspectives of application engineering and domain engineering. According to the scheme, six reuse modes are identified. The formulation of cost and reliability model corresponding to these modes is presented. By defining the variables to represent alternative industrial reuse scenarios, a bi-objective 0-1 integer programming model can be developed to help decision-makers simultaneously select reuse scenarios for minimizing cost and maximizing reliability, and satisfy system requirements. To solve the model, a three-phase based entire efficient solution algorithm is developed. In the first two phases, all supported efficient solutions are obtained. In the third phase, the non-supported efficient solutions are determined.
The paper is organized as follows. Section 2 describes the typical reuse modes and their cost and reliability formulation. In Section 3, a model of Cost and Reliability Optimization under a goal Satisfaction constraint (CROS) is described. The model is expected to achieve the highest reliability and the lowest cost while keeping the satisfactory values of system requirements. Section 4 describes an entire efficient solution method to solve the problem. Finally, conclusions and future work directions are given in Section 5.

2. Typical Reuse Modes and Their Cost and Reliability Formulation. In this section, all possible reuse modes from the viewpoints of the product line [17] are formulated. Clements summarized the activities in a product line using two main circles: core asset development and product development [2]. As the issue of reuse strategies has been introduced [3], operations categorization is reinvestigated and a logic map is created for a common SPL process as shown in Figure 1.

The track in Figure 1 denotes a product line where artifact assets circulate. The gray range in the two circles are assets that have the ability to rebuild components, codes or other resources for any specific reuse (i.e., ②, ⑤). The two circles are not isolated and allow asset exchange using some operations (i.e., ③, ④) as well as build a new asset for reuse (i.e., ②, ⑥). Furthermore, an organization can obtain assets from the outside (i.e., ①). However, no matter in which mode an asset is gained, the terminal aim is to reuse them (i.e., ⑧, ⑨) for a specific target system. The categories are operations of asset acquisition and asset development from the perspectives of domain engineering and application engineering respectively [8]. To summarize the aforementioned operations, the authors contrive and identify six typical reuse modes: pure development (PD), opportunistic reuse (OR), controlled reuse (CR), systematic reuse-COTS (SRC), systematic reuse-adapted (SRA) and systematic reuse-new (SRN) (Figure 2).
3. Mathematical Modeling for Selecting Reuse Scenario. In this section, the problem formulation and modeling of goal satisfaction constraints as well as cost and reliability are described.

3.1. Problem formulation and notations. Assuming an SPL to be developed has n modules. Each module can be built using an optional alternative reuse scenario belonging to one of the six typical modes from PD to SRC. Referring to the goal-oriented requirement engineering approach [18], the functional or non-functional requirements are referred to as goals. The goal-oriented requirement engineering approach matches the goals of system and the features of reuse scenarios through modules as shown in Figure 3. With this approach coupled with the hierarchical scheme, the problem to be addressed by software developers will be how to select a reuse scenario for building each module by minimizing total developing cost and maximizing system reliability under the constraints of goal satisfaction.
To formulate the reuse scenario selecting problem, the following notations are used throughout this paper:

- \( n \) Number of modules within a target SPL system
- \( m_j \) Number of reuse scenarios available for module \( j \)
- \( c_{ij} \) Cost of reuse scenarios \( i \) for module \( j \)
- \( n_{ij} \) Failure rate of reuse scenarios \( i \) for module \( j \)
- \( s_{ij} \) The average of invocations on module \( j \) with reuse scenarios \( i \)
- \( \text{Sat}_{ij}(k) \) Satisfaction level of goal \( k \) when module \( j \) is implemented by using scenario \( i \)
- \( x_{ij} \) Decision variable, which is equal to 1 when module \( j \) is implemented by using scenario \( i \) and 0, otherwise

### 3.2. Modeling of goal satisfaction constraints.

Many methods of requirement analysis were proposed to evaluate functional and non-functional attributes [18-22]. In this paper, the goal-oriented requirements engineering, widely used by researchers and practitioners [20-22], is applied to reflect the goals of a target system.

Assuming that satisfaction level \( \text{Sat}_{ij}(k) \) is obtained by matching scenario preference up to a certain pattern, and its value corresponds to integer number within the range \([0, 1]\), the overall performance of module \( j \) can be obtained as

\[
\text{Sat}_{ij} = \sum_{k=1}^{K} \tau_k \times \text{Sat}_{ij}(k)
\]  

(1)

where \( \tau_k \) is the weight of goal \( k \). The targeted software system should at least meet an overall satisfaction level \( R \). The formulation can be shown as

\[
\sum_{j=1}^{n} \omega_j \sum_{i=1}^{m_j} \text{Sat}_{ij} \times x_{ij} \geq R
\]  

(2)

where \( \omega_j \) is the weight of module \( j \). The analytic hierarchy process (AHP) is used to determine the \( \omega_j \) of the module based on the access frequencies of the modules [23].
3.3. Modeling of cost and reliability. The objective functions of the reuse scenario selection problem are to minimize overall cost of a software system and to maximize system reliability.

Let \( c_{ij} \) be the cost of reuse scenarios \( i \) for module \( j \). The overall cost of a software system can then be given as follows:

\[
\min \sum_{j=1}^{n} \sum_{i=1}^{m_j} c_{ij} x_{ij},
\]

The parameter \( c_{ij} \) is depended on its reuse mode as discussed in Section 2. Similarly, for a module \( j \) with the reuse scenario \( i \), reliability of a module can be formulated in terms of failure rate \( \eta_{ij} \), which is also depended on its reuse mode. The formulation of the reliability of an entire SPL is described as follows.

Let \( p_j \) define the probability of at least one failure occurrence when the module \( j \) is invoked in system execution. Referring to the work of Berman [9], the effect of failures from a module can be represented in an exponential function. One can obtain the probability that no failure occurs during an execution of module \( j \) as follows:

\[
1 - p_j = \exp \left[ - \sum_{i=1}^{m_j} s_{ij} \eta_{ij} x_{ij} \right], \quad \forall j.
\]

In view of the Poisson probability principle, which states that the probability of a failure occurring is proportional to the length of sojourn time, and the assumption that an operation of each module is independent, the probability of the occurrence of no failure in a software system may be formulated as

\[
\prod_{j=1}^{n} (1 - p_j) = \exp \left[ - \sum_{j=1}^{n} \sum_{i=1}^{m_j} s_{ij} \eta_{ij} x_{ij} \right].
\]

Accordingly, to maximize the reliability of a software system, the probability of at least one failure occurrence in a system during the execution of a call run should be minimized. It can be expressed as follows:

\[
\min \left( 1 - \prod_{j=1}^{n} (1 - p_j) \right) = \min \left( 1 - \exp \left[ - \sum_{j=1}^{n} \sum_{i=1}^{m_j} s_{ij} \eta_{ij} x_{ij} \right] \right).
\]

Based on the above equation, it can be concluded that an optimization model for reuse scenario selection problem with minimized cost and maximized reliability under constraints of goal satisfaction (CROS) can be formulated as follows:

\[
\min \sum_{j=1}^{n} \sum_{i=1}^{m_j} c_{ij} x_{ij},
\]

\[
\min \left( 1 - \exp \left[ - \sum_{j=1}^{n} \sum_{i=1}^{m_j} s_{ij} \eta_{ij} x_{ij} \right] \right)
\]

s.t. \(
\sum_{j=1}^{n} \omega_j \sum_{i=1}^{m_j} Sat_{ij} x_{ij} \geq R
\)

\[
\sum_{i=1}^{m_j} x_{ij} = 1, \quad \forall j
\]

\[
x_{ij} = 0/1, \quad \forall i, j.
\]

4.1. Algorithm concept. Assuming that the failure that occurred in a module is equivalent to a failure in the SPL, then the objective function (8) of minimizing the probability is equally interpreted as minimizing the failure rate of SPL. Subsequently, the optimal solution to the objective function can be determined by the following equation:

$$\min \sum_{j} \sum_{i} s_{ij} \eta_{ij} x_{ij}. \quad (12)$$

In contrast to the single-objective optimization, solution to the multi-objective problem is not in the form of a simple unique solution. Often, the solution should be represented by a set of points that satisfies the predetermined conditions. The predominant concept in defining an optimal point is the Pareto optimality, where the corresponding solution called non-dominated solution is defined below.

**Definition 4.1.** A feasible solution \( x \) is said to be efficient if and only if there is no feasible solution \( y \), such that \( z_{k}(y) \leq z_{k}(x) \), \( k = \text{COS}, \text{REL} \), and \( z_{k}(y) < z_{k}(x) \) for at least one \( k \). The image of an efficient solution in the criterion space is called a non-dominated solution.

Given a preference value \( \lambda \) in \([0, 1]\), let

$$z(x, \lambda) = \lambda z_{\text{COS}}(x) + (1 - \lambda) z_{\text{REL}}(x) \quad (13)$$

be the weighted-sum of the two original objective functions, where preferences refer to the opinions of the decision-maker concerning points in the criterion space.

In this paper, a method to obtain the entire efficient solutions (Pareto optimal set) and to provide the trade-off information on solutions is presented. This is done by minimizing the weighted-sum of the objective function (13) subject to the defined constraints of Equations (9), (10) and (11); this is called the Supported Efficient Solutions (Supported-ES). However, the non-supported efficient solutions (Non-supported-ES) cannot be obtained this way because they are located in the dual gaps of consecutive supported non-dominated solution [24].

Dantzig [25] showed that an optimal solution for the continuous 0-1 knapsack problem could not be obtained by sorting the items according to their non-increasing profit-to-weight ratios (also called efficiencies) and by adding them to the knapsack until the capacity is reached. Clearly, CROS is similar to the bi-objective 0-1 integer knapsack problem. Thus, this paper proposes a three-phase based entire efficient solution method for the bi-objective 0-1 integer programming. For the purpose of effectively presenting the proposed methodology, some basic concepts and notions are given as follows.

**Definition 4.2.** Cost-satisfaction ratio: \( \alpha_{ij} = \frac{c_{ij}}{\omega_{ij} S_{ij}} \). It represents the payment for enhancing a unit satisfaction of requirements when module \( j \) is accomplished by reusing in scenario \( i \).

**Definition 4.3.** Reliability-satisfaction ratio: \( \beta_{ij} = \frac{s_{ij} \eta_{ij}}{\omega_{ij} S_{ij}} \). It represents the decreased reliability (failure rate increased) for increasing one unit satisfaction of requirements when module \( j \) is accomplished by reusing in scenario \( i \).

For a module \( j \) using the reuse scenario \( i \), aggregated efficiency \( e_{ij}(\lambda) \) is defined as a function of \( \lambda \):

$$e_{ij}(\lambda) = \alpha_{ij} \lambda + \beta_{ij}(1 - \lambda) = \beta_{ij} + (\alpha_{ij} - \beta_{ij})\lambda. \quad (14)$$
It becomes the jointed effort on cost and reliability to enhance one unit of satisfaction of requirements in module $j$ when used in scenario $i$.

For any module $j$ with given $\lambda'$, an aggregated efficiency can be ordered as

$$e_{t_{1j}}(\lambda') \leq \ldots \leq e_{t_{ij}}(\lambda') \leq \ldots \leq e_{t_{m_j}}(\lambda')$$

where $\langle t_1, \ldots, t_{m_j} \rangle = \langle 1, \ldots, m_j \rangle$.

To determine the efficient solution, the efficiency ratios have to be sorted using Equation (15). Certain efficient/non-dominated solutions can be obtained by minimizing the weighted-sum of the objective functions (i.e., Supporting-ES). The corresponding sorting is called Efficient Order (EO).

4.2. The first phase: Determination of module efficient orders (EOs). The EO of the subintervals for $\lambda$ of each module will be determined. As $\lambda$ changes within $[0, 1]$, the order defined in Equation (15) is not stable [24,26]. Hence, dividing $\lambda$ into subintervals $([0, u_{tj}], \ldots, [u_{tj}, u_{tj+1}], \ldots, [u_{-j}, 1])$ for every module $j$ such that the order of Equation (15) has to be maintained, the procedure is as follows:

**Procedure-1: Determine EOs and subintervals for module $j$**

Begin
Step 1: Initiation: $t \leftarrow 0$, $\lambda' \leftarrow 0$, $u_{tj} \leftarrow 0$
Step 2: Sort items according to relation in (15) with $\lambda'$
Step 3: Solve the upper bounds problem

\[
\begin{align*}
\text{Max} & \quad \lambda^\text{max} \\
\text{s.t.} & \quad e_{t_{ij}}(\lambda) \leq e_{t_{i+1j}}(\lambda), \ i = 1, \ldots, m_j - 1, \ 0 \leq \lambda \leq 1 \\
& \quad \lambda^\text{max} \text{ be the optimal solution.} \\
& \quad \text{If } \lambda^\text{max} \geq 1 \text{, then } t \leftarrow t + 1, \ u_{tj} \leftarrow 1. \\
& \quad \text{If (all intervals have been determined), then stop,} \\
& \quad \text{else go to Step 4.} \\
\end{align*}
\]
Step 4: $t \leftarrow t + 1$, $u_{tj} \leftarrow \lambda^\text{max}$, $\lambda' \leftarrow \lambda^\text{max} + \varepsilon$ (perturbation factor, $\varepsilon > 0$).
Go to Step 2.

End

Using Procedure-1, we can obtain all EOs ($t_1, t_2, \ldots, t_m$) and their corresponding subintervals for $\lambda$ for every module ($j = 1, \ldots, n$). The intervals $([0, v_1], \ldots, [v_g, v_{g+1}], \ldots, [v_G, 1])$ can then be determined by combring the subintervals of every module while keeping the aggregated efficiency of all the modules unchanged. Due to the parallel nature of Procedure-1, the intervals of each module regarding $\lambda$ can be executed on separated processors [27].

4.3. The second phase: Obtaining supported efficient solution set. Once the EOs of the module have been constructed, the next step is to obtain the Supported-ES set. This is done by using a greedy procedure traversing on the intervals of $\lambda$ determined in the first phase. The detailed operation is shown below.

**Procedure-2: Determination of Supported-ES set by greedy procedure**

Begin
Step 1: Initiation $h \leftarrow 0$, $g \leftarrow 0$, $\lambda^h \leftarrow 0$, $\text{EfficientSet} \leftarrow \{\phi\}$, $\text{ExploreSet} \leftarrow \{\phi\}$.
Step 2: Find $x_{t_{ij}}$ where $\min_{t_i} e_{t_{ij}}$ and $\lambda^h \subseteq \{u_{tj}, u_{tj+1}\}$, according to the corresponding EO. Update $\text{ExploreSet} \cup \{x_{t_{ij}}\}$, $\text{EfficientSet} \cup \{x_{t_{ij}}\}$ for all $j$.
Step 3: Loop up to accepted level of satisfaction $R$ (requirements):
Find $x_{t_{i+1j}}$ and $x_{t_{ij}}$ where $\min_{j}(e_{t_{i+1j}} - e_{t_{ij}})$ and unlocked, according to the corresponding EO.
If $x_{i+1,j} = \text{non}$, then set $x_{i,j}$ locked
else update $\text{ExploreSet} \cup \{x_{i+1,j}\}$, $\text{EfficientSet} \leftarrow \{x_{i+1,j}\}$.

Step 4: If Dantzig rule [25] is not satisfied
Save the Supported-ES,
Set $\text{EfficientSet} \leftarrow \{\phi\}$,
$\text{ExploreSet} \leftarrow \{\phi\}$.

Step 5: $\lambda^h = v_{g+1} + \varepsilon$ (perturbation factor, $\varepsilon > 0$).
If $\lambda^h \geq 1$, then Stop (all the intervals have been determined),
else go to Step 2.

End

Using Procedure-2 for every module, the entire set of Supported-ES can be obtained.
Each solution can associate with the trade-off information of $\lambda$.

4.4. The third phase: Determination of non-supported efficient solutions (non-supported ESs).
Based on the experimental results, the Supported-ES set is very close to all efficient solutions [26].
However, dropped Non-supported-ESs are located in the triangular sectors generated in the objective space by two successive Supported-ES [24].
These are represented in Figure 4.

![Figure 4. Method of obtaining efficient solutions](image)

Therefore, an exact procedure starts with the Supported-ES set, which can be employed to determine the Non-Support ESs.
The additional step is to guarantee that the decision space is explored completely.

Procedure: Determining Non-Supported ESs
Begin
Step 1: Obtain the Supported-ES set constructed.
Step 2: Limit the search space in the objective space – The lower bound of the objective space is the stepped line that connects the solutions from the Supported-ES set, and the upper bound is the line connecting the extreme non-dominated solutions from the linear relaxation (region $Z_{non}$ in Figure 4).
Step 3: Select the branch-and-bound [24] or dynamic programming method [28] to explore the reduced objective space; the reduced problems can be solved by using general approaches (or their adaptations) for solving multi-objective 0-1 integer programming problems.

End
In addition, the aggregated functions can be used to define bounds to cover the entire reduced objective space. The upper and lower bounds for these functions (the limits for the rooms) can be derived to provide “optimal” value for some variables to obtain non-dominated solutions within the rooms (Figure 4). When all the rooms have been completely explored, the result is in the ES set. Using the three phases cited above, the ESs found are the “optimal” solution to the CROS model.

5. Conclusions and Future Work Directions. This paper presents a method to examine the cost and reliability of reusing modules in the creation of software products by employing the product line approach. The CROS model based on binary non-linear programming was developed to assist the decision-maker to select alternative reuse scenarios towards simultaneous cost minimization and reliability maximization while satisfying system requirements to a certain degree. A three-phased algorithm for finding all efficient solutions was discussed, and preference-based decision and budget constrained decision method were provided to support the decision-maker in choosing a good solution from the entire range of efficient solutions.

This paper focuses on the optimization of cost and reliability, which are generally the important factors in product line development. However, other factors, such as time-to-market and component supplier selection, are also expected to be included in the reusing software assets. We are currently working on extending our model to include these factors.

Acknowledgment. This work was financially supported by the National Natural Science Foundation of China (NSFC) (70721001, 70625001), the National 973 Basic Research Project (2009CB320601) and the Fundamental Research Funds for the Central Universities (N090604004). This work was also supported by a grant from The Hong Kong Polytechnic University, Hong Kong, China (Project No. G-YJ09). The authors would like to thank the reviewers for their valuable comments and suggestions that helped us improve this paper.

REFERENCES

A MODEL AND ITS ALGORITHM FOR SOFTWARE REUSE OPTIMIZATION PROBLEM


