Component Selection for Component based Software Engineering

Arvinder Kaur
Student, GNDEC, LDH

Kulvinder Singh Mann
Assistant Professor, GNDEC, LDH

Abstract
Component selection is not an easy task in Component Based Software Engineering .and it is very difficult to select component for CBSE. Component Based Software Engineering (CBSE) is a concerned with the assembly of pre-existing software components that leads to a software system that responds to client-specific requirements. This paper presents an approach for defining evaluation criteria for reusable software components. We introduce taxonomy of factors that influence selection, describe each of them, and present a hierarchical decomposition method for deriving reuse goals from factors and formulating the goals into an evaluation criteria hierarchy. It also presents a summary of the common problems in reusable off-the-shelf software selection, describes the method .It also indicates that the evaluated aspects of the method are feasible, improve the quality and efficiency of reusable software selection. In this paper the selection of component is done on the basis of the cost of the component which is calculated on the basis of the quality attributes of the component. The approach used for selecting the component is a part of OTSO method that has been developed for reusable component selection process.

Keywords software reuse, COTS, multiple criteria decision making, OTSO stands for Off-The-Shelf Option.

1. Introduction
Component-Based Software Engineering (CBSE) is concerned with composing, selecting and designing components. As the popularity of this approach and hence number of commercially available software components grows, selecting a set of components to satisfy a set of requirements while minimizing cost is becoming more difficult. In Component-based Software Engineering (CBSE), the construction of cost-optimal component systems is not a trivial task. It requires not only to optimally select components but also to take their interplay into account. In this paper, the problem of component selection is described. Informally, the problem is to select a set of components from available component set which can satisfy a given set of requirements. The dependencies between the components must be taken into account. To achieve this goal, we should assign each component a set of requirements it satisfies. Each component is assigned a cost which is the overall cost of acquisition and adaptation of that component. Many organizations have supported their reuse with component-based technologies. The increased commercial availability of embeddable software components, standardization of basic software environments (such as Microsoft Windows, UNIX), and the explosive popularity of the Internet has resulted in a new situation for reusable software consumers: there are many more accessible reuse candidates. Consequently, many organizations are spending much time in reusable component selection since the choice of the appropriate components has a major impact on the project and resulting product. The component selection process is not defined so each project finds its own approach to it. Here a method has been introduce which supports the search, evolution and selection of reusable software and provides specific techniques for defining the evolution criteria, comparing the cost and benefits of alternatives and consolidating the evolution and selection and benefits of alternatives and consolidating the evolution results in decision making.

2. Component Selection Problem
Informally, our problem is to select a subset of components (each of them satisfying a set of functionalities) and to connect them such that the target component system fulfills the requirements that need to be satisfied.

2.1 Simple Component Selection Problem (SCSP)
Simple Component Selection Problem (SCSP) is the problem of choosing a number of components from a set of components such that their composition satisfies a set of objectives. The notation used for formally defining SCSP, as laid out in with a few minor changes to improve appearance is described in the following.
Consider *SR* the set of the final system requirements (target requirements) as
SR = {r1, r2, ..., rn},
and SC the set of components available for selection as
SC = {c1, c2, ..., cm}.
Each component ci can satisfy a subset of the requirements from SR,
SRci = {ri1, ri2 , ..., rik}.
The goal is to find a set (subset) of components Sol in such a way that to every requirement rj from the set SR can be assigned a component ci from Sol where rj is in SRci .
Figure1 describes our component selection problem: from a set of existing available components *SC* we need to select a subset that satisfies a set of objectives *SR*. We have denoted by *ri* the i−th requirement or offered service. In our previous work we have considered in the composition only the provided services (as satisfied requirement in the final system) of a component. We haven't took under consideration the required services of a component, only those requirements as dependencies that were in the set of SR. For example, in the above Figure 1 the final system computed after selection consists of *C1*, *C3* and *C4* components. The *r2* requirement of the first and the forth

components is satisfied because it is included into the final solution, but the requirements r7 and r8 of the second and the fourth components are not satisfied. A more complex situation is going next to be considered: components required services are going to be used. As stated above, the requirements r7 and r8 is going to be considered in our next research work

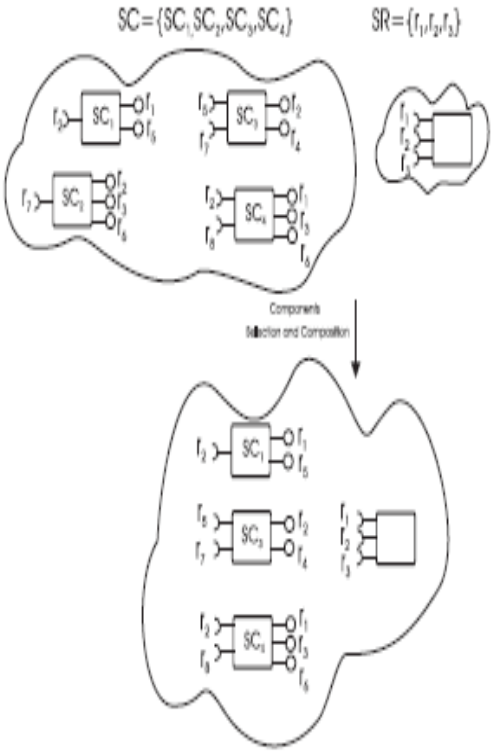

Figure 1. Component Selection Problem.

2.2 Criteria-based Component Selection Problem (CCSP)

Criteria Component Selection Problem (CCSP) is the problem of choosing a number of components from a set of components such that their composition satisfies a set of objectives and using various criteria. Another variation of Component Selection Problem is that stated in. In addition to the above description a cost of a component *ci* is considered cost (*ci*). The goal is to find a set of components Sol in such a way that to every requirement *rj* from the set *SR* can be assigned a component *ci* from Sol where *rj* is in *SRci*, while minimizing the number of components in the solution Sol and/or while minimizing $ci \in Sol\ cost(ci)$. Another criteria introduced in a previous paper concerns the dependencies between the involved components. To specify the component dependencies we have introduced in a dependency matrix. We are only interested in the provided functionalities (that are in the set of requirements *SR* of the final system) of the components.

3. Evolutionary approaches for the component selection problem

Evolutionary algorithms are a part of evolutionary computing which is a rapidly growing area of artificial intelligence.They are well known suitable approaches for optimization problems.There are several ways to deal with a multiobjective optimization problem. We have used both the weighted sum method and the Pareto dominance principle.The weighted sum method**.** Let us consider we have the objective functions *f*1, *f*2,. . . , *fn.* This method takes each objective function and multiplies it by a fraction of one, the "weighting coefficient" which is represented by *wi*. The modified functions are then added together to obtain a single cost function, which can easily be solved using any method which can be applied for single objective optimization.

The Pareto dominance principle. Consider a maximization problem. Let x, y be two decision vectors (solutions) from the definition domain. Solution x dominate y (also written as x _ y) if and only if the following conditions are fulfilled:

1. $fi(x) = fi(y),\ \forall i = 1, n;$
2. $\exists j \in \{1, 2, , n\} : fj(x) > fj(y).$

That is, a feasible vector x is Pareto optimal if no feasible vector y can increase some criterion without causing a simultaneous decrease in at least one other criterion.In what follows, we present two evolutionary approaches which use different representations for the component selection problem.

4. OTSO Method

The OTSO method was developed to facilitate a systematic, repeatable and requirements driven COTS selection process. The main principles of the OTSO method are the following:

a) Explicit definitions of tasks in the selection process, including entry and exit criteria
b) Incremental, hierarchical and detailed definition of evaluation criteria
c) A model for comparing the costs and value associated with each alternative, making them comparable with each other
d) Use of appropriate decision making methods to analyze and summarize evaluation results .

The main characteristics of the OTSO method are as follows:

a) A defined, systematic process that covers the whole reusable component selection process.
b) A method for estimating the relative effort or cost benefits of different alternatives.
c) A method for comparing the "non-financial" aspects of alternatives, including situations involving multiple criteria.

Figure2 shows the main activities in the OTSO reusable component selection process using a dataflow diagram notation. Each activity in presented as a process symbol – a circle – and artifacts produced or used are presented as data storage symbols in Figure 1. In the search phase, the goal is to identify potential candidates for further study.The screening phase selects the most promising candidates for detailed evaluation In the analysis phase, the results of product evaluations are consolidated .and a decision about reuse is made. As the selected alternative is used,

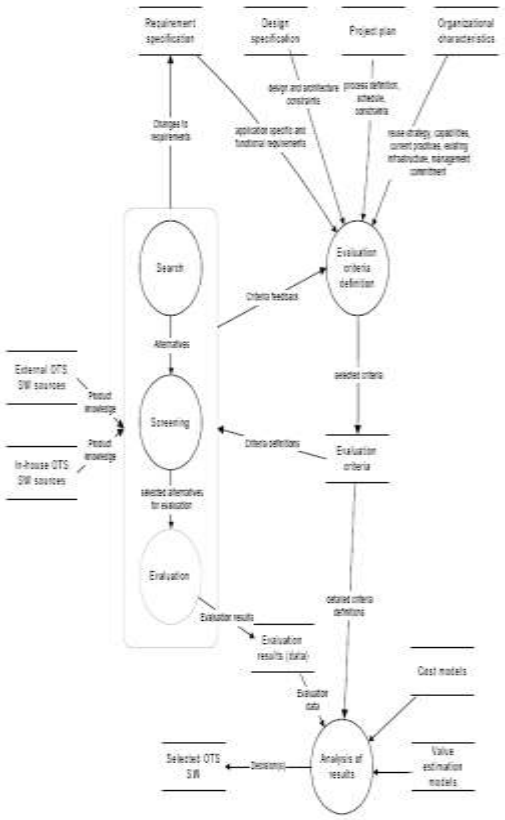

Figure 2.The main phases in the OTSO process

the effectiveness of the reuse decision, eventually, can be assessed. Reuse candidates are evaluated in different ways in all phases. The OTSO method is based on incremental, evolutionary definition and use of the evaluation criteria so that the criteria set can be gradually refined to support each phase.

5. Factors in Reusable Software Selection

The overall relationships among influencing factors, reuse goals and evaluation criteria are presented in Figure The first main task in reusable software evaluation is to define the reuse goals. an organization's reuse infrastructure and reuse maturity should also be considered when defining reuse goals. Reuse maturity is particularly important for the in-house production of reusable components. Also, if an organization has no experience in OTS software reuse, it may have a limited ability to integrate OTS software and to estimate the effort required for OTS software integration. Application and domain architecture introduce additional elements that need to be evaluated. The architecture, in this context, provides a set of constraints deriving from how particular applications are built: this includes, for instance, components and design patterns used or assumed, communication and interface standards, platform characteristics. The application domain may also have some specific characteristics that are not addressed by OTS software developed for other domains (for example, real-time applications vs. batch processing). Application requirements are likely to be the most important factor in evaluating reusable software. Such requirements can include functional requirements (such as the ability to manage and display graphical geographical data) and non-functional requirements (such as available memory or speed of operations). Requirement specification typically does not define how the system should be implemented or what components could be implemented through OTS software. Second, the requirement specification may not be detailed enough for evaluating OTS software alternatives. Project objectives and constraints may influence the library selection through the schedule or the budget of the project. The availability of features in software reuse candidates also affects the evaluation criteria definition. This works in two ways. On one hand, it is important to check that the evaluation criteria are based on realistic expectations. That is, the criteria set should not assume characteristics that are not provided by any OTS software alternative. On the other hand, it may be useful to know about the possibilities that OTS software alternatives offer but that may not have been included in the requirement

Reuse objectives can be divided into development process goals, maintenance process goals and product characteristics goals. Development process goals relate to the cost, effort and schedule of the development project. The maintenance process goals deal with issues such as the ease or cost of maintenance and who will be responsible for maintenance. Product characteristics goals refer to product functionality and product quality.

6. Evaluation Criteria

Classes of evaluation criteria

The factors and goals described in Figure above, determine the reuse goals for the system. The content and priorities of these goals determine which characteristics must be considered in the of the OTS software selection process. The evaluation criteria themselves can be categorized into four main areas:

(a) Functional requirements
(b) Product quality characteristics
(c) Strategic concerns, and
(d) Domain and architecture compatibility.

Functional requirements: These refer to identifiable, functional features or characteristics that are specific to the particular situation. These criteria are derived from the requirement or design specification and are expressed in the form of requirements.

Product quality characteristics are common to a broader set of reuse situations. Typically the structure and relationships of these characteristics remain the same but their

acceptable values may vary from case to case. Three examples are:

· Defect rate
· Compliance to the project user interface guidelines
· Clarity of documentation.

Strategic concerns: These are the short-term and the long-term effects of the reuse candidate, the cost-benefit issues and the organizational issues beyond the scope of the project in question. These help to consolidate information for decision making. Three examples are:

· Acquisition costs
· Effort saved
· Vendor's future plans.

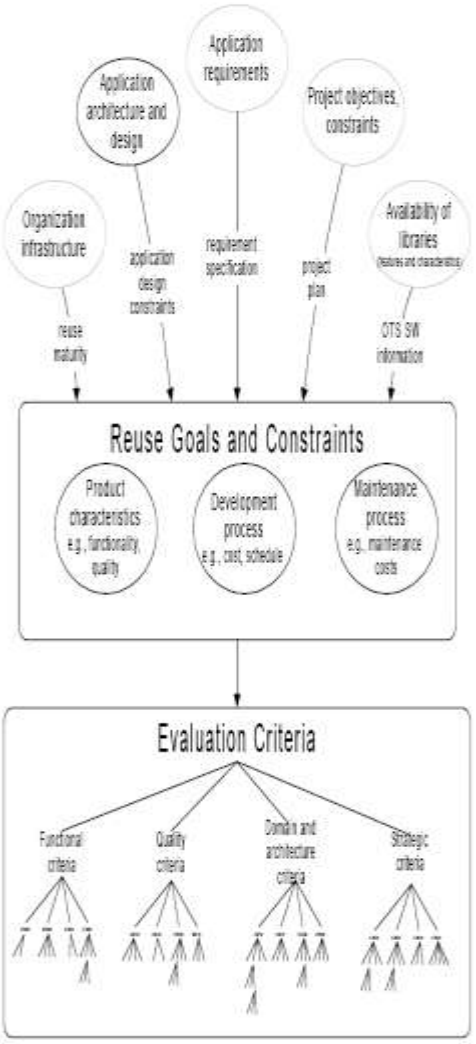

Figure 3: Factors influencing the selection of reusable off-the-shelf software

Domain and architecture compatibility: Domain compatibility refers to how well the reuse candidate and its features map into the domain terminology and concepts. Architecture compatibility refers to software or hardware architecture requirements that are common to the application area.

7. Case Studies

We carried out two case studies using the OTSO method. The results of these case studies are reported separately .The both the case studies it is assessed the overall feasibility of the method. The first case study took place in the NASA's Earth Orbiting System (EOS) program with Hughes Information Technology Corporation and were dealing with real software development projects facing a COTS selection problem. First case study dealt with the selection of a library that would be used to develop an interactive, graphical user interface for entering location information on Earth's surface areas. This case study used the OTSO method's hierarchical and detailed criteria definition approach. Part of the criteria hierarchy is presented in Figure 4.The main conclusion was that the OTSO method was a feasible approach in COTS selection and its overhead costs were marginal. The first case study also showed that OTS package features can change the application requirements: one of the OTS alternatives was able to display ocean bathymetry data graphically. Although this was not initially specified as a requirement, the application designers considered it a valuable feature and proposed it to be included in the requirements specification. This important feedback loop is characterized by the arrow from the search/screening/evaluation contour in Figure 2.

Second case study dealt with the component selection based on the cost and benefit of the component. The costs and benefits involved in addition of a component are calculated by evaluating the quality parameters such as cost avoidance, reliability, productivity, understandability etc. of the components. The desirability of the component is to be found the component having higher desirability is to be taken because it is having higher benefit and having the minimum cost. The quality parameters can be calculated by following defined metrics.

Total Cost= Cost Of Identifications +Cost Of Evaluation + Cost Of Integration+ Cost Of Capital

Total cost avoidnce = Development_Cost_Avoidance + Planing_Cost_Avoidnce + Downtime_Loss_Avoidance

Final cost avoidnce = totalCostAvoidance+developmane Cost;

DevelopmaneCost = tempDevelopmaneCost1+tempDevelopmaneCost2;

tempDevelopmaneCost1 = total_count_lines*.8 *Cost_of_per_lone_code

tempDevelopmaneCost2 = ("total_code_lines")* errorRate* Cost_per_error

Comment Density = totalComments/totalLines;

Understandability =Comment Density **of Code**

Producitivity Of Code == (effort*24*30)/timeToReUseCode

Reliability = basicCompDetail+graphing+ dataBaseAccess+security+transaction

TotalBenefit= totalCostAvoidance+
toatlUnderstandability+totalProductivity
+totalReliability

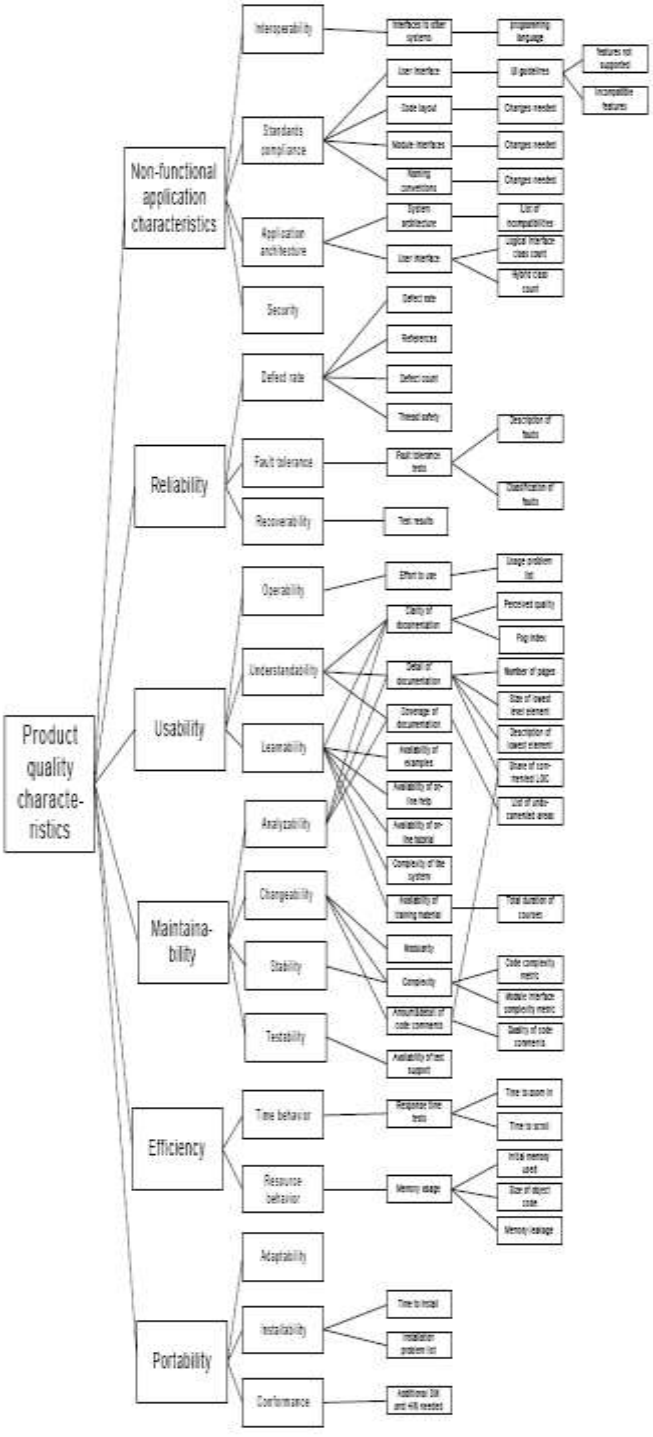

Figure 4: Example of a product evaluation criteria hierarchy

Relation between Various Quality Attributes which is to be defined in the second case study

Following relations between various quality attributes have been determined by the analysis:

a)Performance α $\frac{1}{\text{Communication between components}}$

b)Performance α Cohesion α $\frac{1}{\text{Coupling}}$

c) Ease of Modifiability α $\frac{1}{\text{Cohesion}}$ α $\frac{1}{\text{Coupling}}$

d) Security α $\frac{1}{\text{Performance}}$ α Reliability α Cost

e) Fault Tolerance α mean time of failure

f) Fault Tolerance α $\frac{1}{\text{Programming}}$ α Cost α morcritical Performance Elemnets

g) Time α $\frac{1}{\text{COTS products used (in number)}}$

h) COTS product used α $\frac{1}{\text{Cost}}$

8. Conclusions

The OTSO method was developed to consolidate some of the best practices we have been able to identify for OTS software selection. our first case study was performed in the application domain, we have not encountered any domain specific characteristics that would limit the applicability of the method in other domains. Also, while the case study them was relatively small, the evaluation processes, and the resulting criteria, were quite extensive. This leads us to suggest that the method may be able to scale up to larger situations as well. However, further validation is necessary to determine this with more confidence. From the second case the results which we get after the analysis of quality attributes of a component i.e. reliability, cost avoidance, productivity, understandability help us in making decisions whether any component has to be added into the system or not. The costs involved in adding a component is evaluated by taking into consideration cost involved in selection, identification, evaluation of a component. The results depend on the quality assurance score which the stake holders have to decide, basis of which any component is selected. It is noticed that if the contribution of the component is more towards that particular software quality attribute and the quality assurance score of that attribute is more then there is sharp increase in the desirability of that component.

9. References

1)M. R. Fox, D. C. Brogan, and J. Paul F. Reynolds. Approximating component selection. In WSC ’04: Proceedings of the 36th conference on Winter simulation, pages 429–434.Winter Simulation Conference, 2004.

2) N. Haghpanah, S. Moaven, J. Habibi, M. Kargar, and S. H.Yeganeh. Approximation algorithms for software component selection problem. In APSEC, pages 159–166. IEEE Computer Society, 2007.

3) A. Vescan. Dependencies in component selection problem.In 5th International Workshop on Formal Aspects of Component Software (submitted). ENTCS, 2008.
4)Y. Kim and O. deWeck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. Structural and Multidisciplinary Optimization, 29(2):149–158,2005.
5)A. Abraham, L. Jain, and R. Goldberg. Evolutionary Multiobjective Optimization: Theoretical Advances and Applications.Springer Verlag, London, 2005.
6) J. Kontio, "OTSO: A Systematic Process for Reusable Software Component Selection,"CS-TR-3478, 1995. University of Maryland Technical Reports. University of Maryland. College Park, MD.
7) J. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection,"1996. Proceedings of the 18th International Conference on Software Engineering
8) J. Kontio, S. Chen, K. Limperos, R.Tesoriero, G. Caldiera, and M. S. Deutsch,"A COTS Selection Method and Experiences of Its Use," 1995. Proceedings of the 20th Annual Software Engineering Workshop. NASA. Greenbelt, Maryland.
9) R. Prieto-Díaz, "Implementing faceted classification for software reuse,"Communications of the ACM, vol. 34,5.1991.
10) C. V. Ramamoorthy, V. Garg, and A.Prakash, "Support for Reusability in Genesis," pp. 299-305, 1986. Proceedings of Compsac 86. Chicago.
11) J. W. Hooper and R. O. Chester. "Software Reuse: Guidelines and Methods," R.A.Demillo (Ed). New York: Plenum Press,1991.