# Software Replaceability: An NFR Approach

Lei Zhang Lawrence Chung Jing Wang
Department of Computer Science
The University of Texas at Dallas
{lei74, chung, jwang}@ utdallas.edu

## Abstract

*Building software systems from components instead of from scratch is a trend in software industry world. Software replaceability gains growing interest recently since the replaceable, standard components in the marketplace is claimed as one of the goals and benefits of components. Treating software components as fully replaceable units will help CBS (Component Based System) development and evolvement dramatically. However, the market place for replaceable components is still not at sight due to many reasons. Unlike hardware system, software replaceability is a more complicate, flexible and broad concept besides standardization. In this paper, we will give a comprehensive definition of replaceability, use NFR (non-functional requirement) framework to analyze software replaceability, and illustrate the NFR approach using a BOM (Bill of Materials) management system.*

## I Introduction

Building systems from components presents both promise and challenges. CBS development can be treated as a process to assemble the independent components together to form a functioning system. NFRs are critical in using or selecting the components. Software replaceability is one of the NFRs that gains growing interest recently since the replaceable, standard components in the marketplace is claimed as one of the goals and benefits of components.

Treating software components as fully replaceable units will help CBS development and evolvement dramatically. Replaceable components are expected because software system always deals with change. To continue using existing software, system must manage the evolution to accommodate the change; the use of replaceable components is one of the possible ways to manage the system maintenance and evolution.

However, the marketplace for replaceable components is still not at sight due to many reasons. Unlike hardware system, besides standardization, we must conquer both technical and non- technical barriers to realize replaceable software components.

Replaceability is a more complicate, flexible and broad concept in software world besides the simplest scenarios "A substitute B". It is necessary to apply methodology to analyze software replaceability and guide building replaceable system at the beginning of requirement phase. In this paper, we will use NFR framework to analyze software replaceability. The rest of the paper is organized as follows. Section 2 is the related work. Section 3 gives a comprehensive definition of software replaceability and introduces the NFR approach. In section 4, we illustrate the NFR approach using a BOM (bill of materials) management system. Future work and conclusion are summarized in section 5.

## II Related Work

Replaceable unit has different meanings depending on the perspectives adopted. From the view of COTS, industrial standardization on a small number of component frameworks is demanded [Brown1998]. Popular component models such as Java Bean, EJB and COM define mechanisms to help integrate distributed components, but do little to support or encourage component as replaceable unit. Although some successful stories exist, for example, some Java applications that focus on special domains like JDBC, JNDC, and JAXP do encourage the creation of replaceable components [Seacord1998]. The marketplace for replaceable components is still not at sight.

Replaceable component is hard to achieve due to many reasons. Missing, insufficient, incompatible standards make replaceable unrealistic. Using software patents to support the business models of software component and suppress multiple standards caused by hypocritical marketing strategies may help compatible standardization, but this is a long way to go [Guntersdorfer2000]. Moreover, It's difficult to identify and quantify the exact economic cost and benefit derived from the development of replaceable components. The component providers may resist the emergence of replaceable component market since component substitutability means price competition, which is not the interest of software product vendors. [Wallnau1999]

Simple plug-in component is an ideal scenario. However, modification to the replaceable component or system is required more or less when the new component is integrated into system. Evaluating the similarity of relative components and the effort of modification is a necessary and nontrivial work. Component has a lot of characters. [Yacoub1999] Based on the component's three important internal characters (structure, behavior, and granularity), a metric (DRD---Directed Replaceability Distance) is designed to represent how different two components are in the case the system requirements are the same before and after replacement. And this me0tric is used in the component search engine RetrivalJ. [Washizaki2002]

## III Software Replaceability---an NFR Approach

### 3.1 Motivation

Building software system by combining existing pieces of software rather than build-from-scratch is claimed as a goal of component engineering and future breakthrough in software development. System must accommodate change and keep pace with future technology improvement after deployment. For example, components can become obsolescent, developers may choose components from different vendors, software system or component can have variable usage, and the system running environment, dependencies, requirements also change from time to time. Therefore, maintaining and evolving the system with component substitution is a necessary and important issue for CBS. The CBS should be designed in a way that the component can be isolated and are replaceable. To achieve this goal, we will use NFR framework to guide requirement analysis and architecture design.

### 3.2 Replaceable Software Component

The replaceability paradigm implies two aspects, the engineering of replaceable components, and the engineering of system by using the replaceable components.

Component is subject to composition by third parties. Composition means the component is developed by integrating and using service provided by previous components. Therefore, a component can encapsulate and describe other components that in turn can be embedded by other larger components. The composition hierarchy can be nested up to any arbitrary level. A component-based system is built on top of low-level components recursively and can be viewed as the root element of the composition hierarchical structure.

A replaceable component is a conceptual independent part that can be added, deleted, modified or substituted. It can be the entity in any hierarchical level. That is, a replaceable component can be a single interface, a single class, a multi-class modular or a whole system.

According to whether the system's functionality will be affected by replaceable components, we classify the replaceable components into two cases. Both cases can be found in the illustration system.

Case 1: Homogeneous replaceable components
Replaceable components are used to maintain the system's functionality while running environment change, vendor change, or technique advance. For example, a set of components for sorting.

Case 2: Heterogeneous replaceable components
Replaceable components are used to accommodate the functional requirements change. System will provide different or extensive functionality after component substitution. For example, the components used to present data file in different format.

### 3.3 Software System Replaceability

In this paper, we define replaceability as the ability of system to substitute some composite components to accommodate change with reasonable effort.

More specifically, the component (C) of a software system (S) can be substituted by a new component (C') and results in a new system (S'). The effort for the replacement must be reasonable (effort< MaxEffort, the MaxEffort is the maximum threshed value defined by project manager.)

express NFRs explicitly, deal with them systematically and use them to drive development process rationally. [Chung2000]

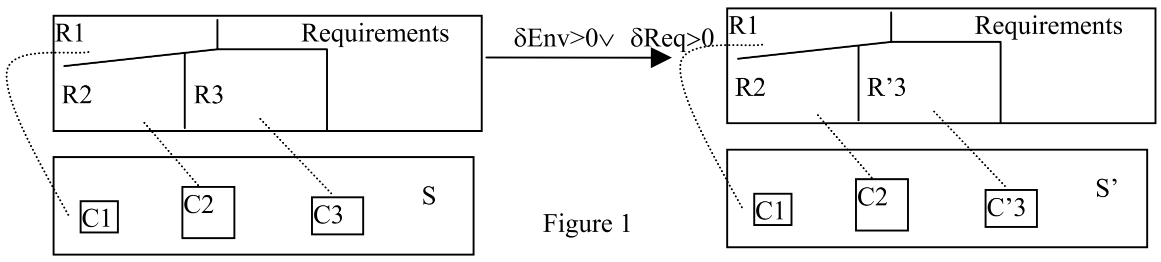


Figure 1

$$\exists C \in S\ \exists R \in Req\ \exists E \in Env\ Match(R, C, E) \wedge (\delta Env>0 \vee\ \delta Req>0)$$
$$\rightarrow \exists C' \in S'\ \exists R' \in Req'\ \exists E' \in Env'\ Match(R', C', E') \wedge (C \neq C') \wedge effort < MaxEffort$$

The replacement is caused by change (δEnv) from old environment (Env) to new environment (Env') or by change (δReq) from old requirements (Req) to new requirements (Req').

Each component should satisfy a small part of overall requirements under its dependencies (Match(R, C, E)). When change happens, new component is integrated into the system to satisfy the new requirements or adapt to the new environment (Match (R', C', E')) (See Figure 1).

To discuss system replaceability, we need consider following major tasks:

1. The ability to identify change. (δEnv>0∨ δReq>0)
2. The ability to trace the change to appropriate components.
3. The ability to search, identify and evaluate the candidate components for replacement that may accommodate the change. If C'∈S', can Match(R', C', E') hold? Is the modification effort needed for replacement acceptable?

We will show how to perform these tasks using NFR approach in the following section.

### 3.4 NFR Approach

In our research, NFR framework is the methodology that guides system to accommodate change with replaceable components. NFR framework is a goal-oriented and process-oriented quality approach guiding the NFRs modeling. Non-functional requirements such as security, accuracy, performance and cost are used to drive the overall design process and choosing design alternatives. It helps developers express NFRs explicitly, deal with them systematically and use them to drive development process rationally. [Chung2000]

In the NFR Framework, each NFR is called an NFR softgoal (depicted by a cloud), while each development technique to achieve the NFR is called an operationalizing softgoal or design softgoal (depicted by a dark cloud). Design rational is represented by a claim softgoal (depicted by a dash cloud). Each softgoal has a name following the convention Type [Topic1, Topic2, …], The goal refinement can take place along the Type or the Topic. These three kinds of softgoals are connected by links to form the softgoal interdependency graph (SIG) that records the design consideration and shows the interdependencies among softgoals.

In this paper, software replaceability is the NFR encapsulated in the whole design process. We will use following major steps to show how software system encapsulate software replaceability in the requirement and design phase and how to identify replaceable components to accommodate the change.

1. Develop the NFR softgoals and their decomposition based on the replaceability of the concrete system.
2. Develop design softgoal alternatives based on the knowledge of development techniques.
3. Analysis design tradeoffs and rational.
4. Develop critical goals; document the rational with claim goal.

   In this step, the potential change should be identified and associated with corresponding NFR softgoal. The NFR softgoal will be marked as critical. Sometimes, the SIG will be revised to reflect the change.

Certain functions of a software system are subject to frequent change on the basis of the business needs. Some system services such as database engineering will only be modified by vendors as COTS component upgrade. Other services are frequently changed by the ender users, such as an order processing system must respond to a customer order differently depending on the actual product request. The developer must understand which functionality would like to be changed, which service will actually be changed and changed by who during the lifecycle of the system.

5. Identify replaceable components, evaluate and select alternatives.

   The NFR framework provides clear traceability from requirement to design. By analyzing the impact of change on the system, the critical goal identified in step 4 can be traced to corresponding design softgoal. Moreover, Using SIG, the candidate design softgoals for replacement can be identified and isolated by searching the offspring node or sibling node. Then, the design softgoal will be mapped to components of solutions considered for the target system by searching, matching and selecting from repository. A goal-oriented process model that explicitly supports the selection and integration of COTS components with NFR framework is presented in [Chung 2003a], [Chung2003b].

   Other necessary activities include identify the significant dependencies of replaceable component among other components, analyzing the significant difference among suitable components, determining the effect that replaceable software component will have on the system's behavior, evaluating the effort request for the replacement.

The steps in the entire process are interleave and iterative. A SIG will be generated by carrying out these steps. In section 4, a BOM system will be analyzed by complying above steps.

## IV Illustration

### 4.1 BOM Management System

BOM (Bill of Materials) identifies and lists all components, assembles, sub-assemblies, parts and raw materials that contribute to the end product. It is one of the necessary parts of Material Requirement Planning (MRP). A BOM may be in a tree form (See Figure 2), or in a printed indent document (e.g. excel)

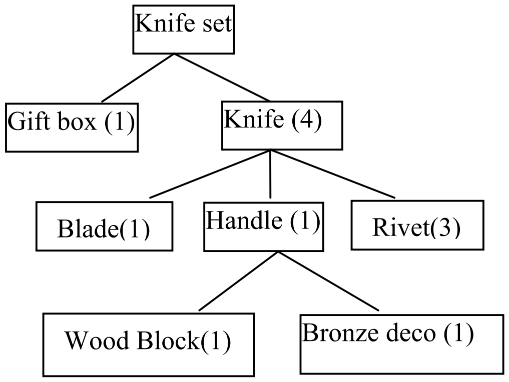


Figure 2: A kind of BOM format

The BOM encapsulates design knowledge and history of the product. It supports not only production but every supplier's activity like manufacture, purchasing, accounting, engineering, technical writing and marketing. The BOM management system must accomplish two important tasks.

•The data content and structure of BOM undergoes constant change. It's important that the process of updating the BOM should be as straightforward and automatic as possible. The change of the BOM should be in time and consistent in the entire enterprise. (Data consistence) For example, a frame may add to the "Knife Set"; all departments from sales to accounting must update the information immediately and consistently.

•The portions of BOM typically will be shared by different departments across the entire enterprise, Different pieces of its content are critical to different departments in quite different ways. So the problem is how to expose, extract and present those various items of BOM knowledge in highly specialized way. (Data access) For example, "revision code" records the information about product quality; it may only viewed by manager and quality assurance department.

One possible implementation of the BOM system is to represent the BOM with well-formed XML file (See Figure 3, 4). The "Data Producer" gets raw product design data from product manager and generates XML file, then save the file. The "Data Consumer" can be any application (such as order processing, inventory control, accounting, and so on) that requests BOM information. The “data Processor” is the subsystem that really accesses the BOM file; it processes the XML data according to the applications’ requirements and provides the data to various applications. All these subsystems can reside in same or different domain. The XML solution doesn’t require radical data conversion, only a straightforward change in already structured data.

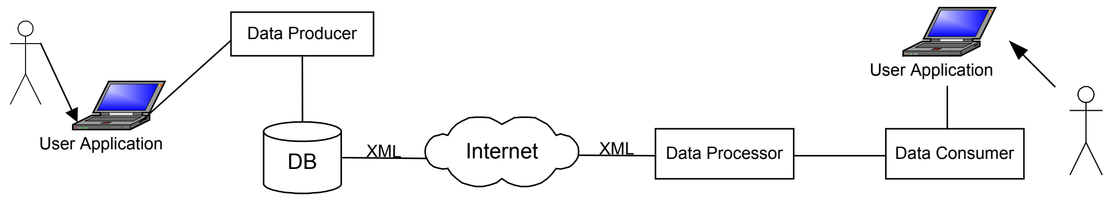


Figure 3: An overview of BOM system

```
<item>
        <itemno>000-000-0010</itemno>
        <desc>the gift box </desc>
        <itemlevel>1</itemlevel>
        <quan>1.00</quan>
        <rev>A1</rev>
        <leaf>yes</leaf>
        <parent>00</parent>
</item>
<item>
        <itemno>614-000-0100</itemno>
        <desc>a set has four knives</desc>
        <itemlevel>1</itemlevel>
        <quan>4.00</quan>
        <rev>B1</rev>
        <leaf>no</leaf>
        <parent>00</parent>
</item>
……
```

Figure 4: Part of BOM XML file for the product knife set

## 4.2 Apply NFR Approach

### 4.2.1 Develop the NFR Softgoals and Their Decomposition

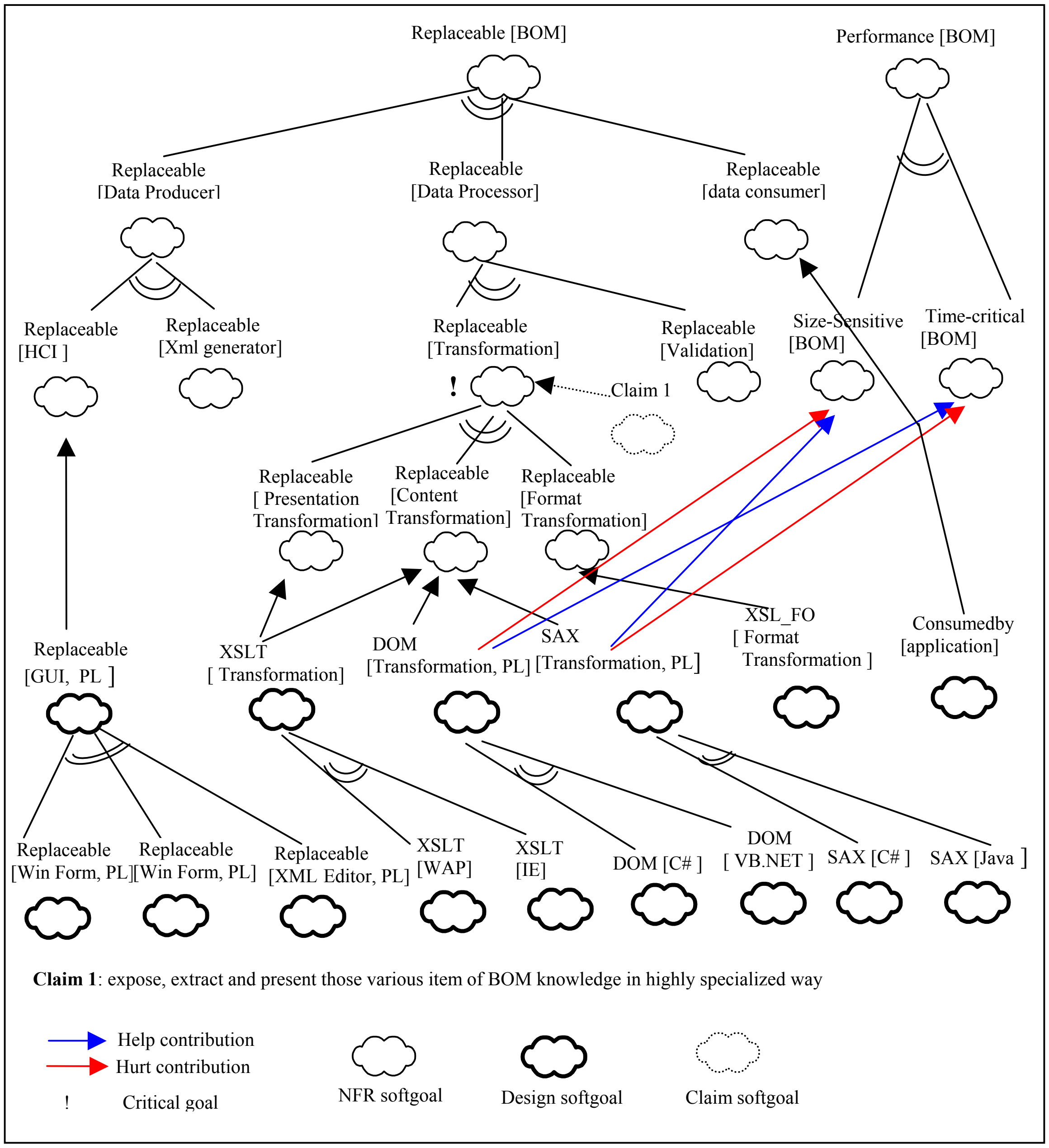
The NFR softgoal decomposition shown in top part of Figure 5 is refined by following way. The BOM management system consists three major modules, we “OR” decompose the top goal (Replaceable [BOM]) into three subgoals via subsystem. (Replaceable [Data Producer], Replaceable [Data Processor], Replaceable [Data Consumer]). In data producer modular, the product designer use human-computer interface (HCI) to input raw data of BOM file, then a XML generator should transfer the raw data into XML file and save the file into database. So the subgoal Replaceable [Data Producer] can be further refined into two subgoals. (Replaceable [HCI], Replaceable [XML generator].

The data processor module can perform two different tasks: data transformation and data validation, so the subgoal Replaceable [Data Processor] can “OR” refine to two subgoals (Replaceable [Data Transformation], Replaceable [Data Validation]). Data transformation can have different types, this is why the subgoal Replaceable [Data Transformation] further “OR” refines to Replaceable [Presentation Transformation], Replaceable [Content Transformation] and Replaceable [Format Transformation]. The data consumer modular relates to various business application of BOM. The decomposition requires detail business domain knowledge. We didn’t go further in this part right now.

### 4.2.2 Develop the Design Alternatives

The development techniques are represented as design softgoals (the dark cloud) in the bottom part of Figure 5. Human-computer interface for the data producer will be provided by the GUI technique. However, different front-end exists to realize the GUI, so the design softgoal Replaceable [GUI, PL] can be refined into three subgoals---Replaceable [Win Form, PL], Replaceable [Web Form, PL], Replaceable[XML editor, PL]. Moreover, the program language (PL) used to implement the GUI can always replaceable according to the enterprise platform and available resource, the look and feel of the GUI are also replaceable according to the user preference. Therefore, further refinement to these subgoals are possible and useful based on requirements.(Further refinement is not shown in this graph.) The functionality of GUI is to get input from users; the functionality of XML generator is to transfer raw data into XML file.

Their alternatives are the examples of homogeneous replaceable components.



**Claim 1**: expose, extract and present those various item of BOM knowledge in highly specialized way

Figure 5: SIG for BOM system

As for the data processor, current techniques to manipulate XML data have XSLT, XSL_FO, XPath, DOM, and SAX. (The XML technique is still evolving). The replaceability depends on the concrete system requirements and environments. However, DOM and SAX are standard interfaces that can always be implemented with multiple ways and multiple programming languages. The uniform interface supports replaceable components by giving the system freedom to choose different implementation techniques based on distinguish resource constraints.

### 4.2.3 Analysis Design Tradeoffs and Rational

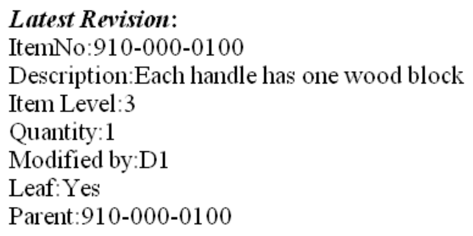
Some XML techniques have crosscutting functionality but will affect system's non-functional aspects or more suitable in certain environment. This difference provides a rational for choosing design alternatives. In Figure 5, we use "performance" as a rational to compare SAX and DOM.

### 4.2.4 Develop Critical Goal

The task of this step is to identify the important potential change, associate the change with softgoals and mark them as critical. According to the requirements of BOM system mentioned before, "The BOM system must expose, extract and present those various item of BOM file to different users in highly specialized way." the softgoal Replaceable [Data Transformation] is marked as a critical goal and the rational is documented with the claim goal (a dash cloud).

### 4.2.5 Identify Replaceable Components

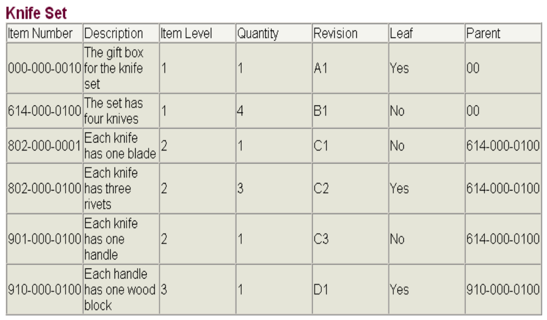
The NFR softgoal Replaceable [Data Transformation] can be traced to design softgoal XSLT [Transformation]. When change of presentation is required, for example, same BOM data should be displayed in different ways based on platform-dependency, we can search the offspring nodes for the alternative development techniques. If the server detects the end user is WAP-compatible device, it will send following page card (Figure 6) to the device using XSLT [WAP]. If the server detects the end user is IE 5.0 or IE 6.0, it will present the table (Figure 7) by replacing the XSLT [WAP] with XSLT [IE]. These components for data transformation produce different output with same input (BOM file). They are the examples of heterogeneous replaceable components.



***Latest Revision*:**
ItemNo:910-000-0100
Description:Each handle has one wood block
Item Level:3
Quantity:1
Modified by:D1
Leaf:Yes
Parent:910-000-0100

Figure 6: One card for WAP compatible device

Another example is about "Data Content Transformation". Both DOM and SAX can be used to retrieve part of the XML data. They are siblings in the SIG. SAX is efficient for retrieving small amounts of information. While DOM is used, the whole XML files should be loaded into the memory, when the size of the XML file is large, it has high memory space request. So the system can adopt the corresponding technique according to the environment and resources constraints (such as high performance vs. memory conservative parser). Since DOM and SAX are standard interfaces, multiple implementations are always possible and replaceable. We can search the offspring for the design softgoal with preferred program language as part of its topic and map the design softgoal to concrete XML parser.



**Knife Set**

| Item Number | Description | Item Level | Quantity | Revision | Leaf | Parent |
|---|---|---|---|---|---|---|
| 000-000-0010 | The gift box for the knife set | 1 | 1 | A1 | Yes | 00 |
| 614-000-0100 | The set has four knives | 1 | 4 | B1 | No | 00 |
| 802-000-0001 | Each knife has one blade | 2 | 1 | C1 | No | 614-000-0100 |
| 802-000-0100 | Each knife has three rivets | 2 | 3 | C2 | Yes | 614-000-0100 |
| 901-000-0100 | Each knife has one handle | 2 | 1 | C3 | No | 614-000-0100 |
| 910-000-0100 | Each handle has one wood block | 3 | 1 | D1 | Yes | 910-000-0100 |

Figure 7: A table format for IE

## V Conclusion and Future Work

Replaceable component is an effective way to modify the system to adapt to change. During CBS development and evolvement, the possibility of current composite components to be replaced by new ones without much modification to the system or the new components is an important issue that need be addressed. With NFR framework, the replaceability is treated as a NFR that can guide whole software engineering process, including requirement, component selection, architecture design, implementation, maintenance and evolution.

In this paper, we first gave a comprehensive definition of replaceability, then introduced the NFR approach for replaceability analysis. In the illustration section, we constructed a SIG for the BOM management system from the view of replaceability, showed how to identify the critical softgoal through SIG and trace it to design alternatives. Finally, we explored the replaceable design techniques and make the trade-off analysis.

Replaceability is a NFR relating to the whole system, although our research focused on replaceable components in that phase, we should not limit the scope only to components. The concept of replaceable will be extended to all constituents of the system architecture in the future (component, connection, constraint, style and pattern.)

Also in our future work, we will analysis the attributes contribute or damage software replaceability. Software replaceability will also be used as a rational to guide the design and development of software architecture supporting replaceable constituents. The correlation of replaceability with other NFRs such as performance, extensibility, modifiability, and adaptability will also be a direction of future work.

## Reference

[Brown1998] Brown, W., and Wallnau, K., “the current state of CBSE”, *IEEE Software*, October 1998, pp37-46.

[Chung2000] Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J., *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston, 2000.

[Chung2003a] Chung, L., Cooper, K., "Defining Goals in a COTS Aware Requirements Engineering Approach", submitted to *Systems Engineering*

[Chung2003b] Chung, L., Cooper, K., "Architecting Adaptable Software Architecture Using COTS: An NFR Approach," *Proc., Int. Conf. on Software Engineering Practice and Research (SERP'03)*, June, 2003, pp. 155-161.

[Guntersdorfer2000] Guntersdorfer, Michael et.al , “ Using Software Patents to Support the business Model of Software Components. *ICSE workshop*, 2000.

[Lamsweerde2000] Axel van Lamsweerde, “Goal-Oriented Requirements Engineering: A Guided Tour”, *Proc., Fifth IEEE International Symposium on Requirements Engineering,* Aug. 2001.

[Seacord1998] Seacord, Robert C., Wrage, L., *Replaceable Components and the Services Provider Interface*, Technical Note, CMU/SEI-2002-TN-009.

[Szyperski1998] Szyperski, C. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley and ACM Press, 1998.

[Wallnau1999] Kurt C. Wallnau, “On Software Components and Commercial ("COTS") Software”, *ICSE,* 1999.

[Washizaki2002] Washizaki, H., Fukazawa, Y., “Retrieving Software Components Using Directed Replaceability Distance”, *OOIS* 2002, pp298-310.

[Yacoub1999] Yacoub, S. “Characterizing a Software Component”, *International Workshop on Component based Software Engineering*, 1999.