4-2008

COTS-Based Software Product Line Development

Luiz Fernando Capretz
University of Western Ontario, lcapretz@uwo.ca

Faheem Ahmed
Thompson River University, fahmed@tru.ca

Shereef Al-Maati
American University of Kuwait, ShereefAlMaati@gmail.com

Zaher AlAghbari
University of Sharjah, zaher@sharjah.ac.ae

International Journal of Web Information Systems, Volume 4, Number 2, pp. 165-180, Emerald Group Publishing, 2008.

COTS-Based Software Product Line Development

Luiz Fernando Capretz [1]), Faheem Ahmed [2]),
Shereef Al-Maati [3]), Zaher Al Aghbari [4])

Structured Abstract:

Purpose of this paper	Software Product Line (SPL) is at the forefront among the techniques for reducing costs, decreasing schedule time, and ensuring commonality of features across a family of products - as components off-the-shelf (COTS) are reused in multiple products.
Design/methodology/approach	A disciplined process for software product line development is still needed. We propose the Y-model for COTS-based software product line development. The model put forward identifies and elaborates the essential phases and activities of software product line development from COTS-based repository.
Findings	The Y-model provides an efficient way of integrating the approaches of software product line and COTS-based development as a cohesive software development model.
Practical implications	The model has the potential to tremendously increase software engineers' productivity. Thus software architects, domain engineers and component designers should become aware of how to use these ideas to structure their models and designs.
What is original/value of paper	This research describes a systematic approach for COTS-based development that takes into account the cataloguing and retrieval of software assets permeating a process that encompasses all stages of software development from system product requirements engineering to system deployment.

Keywords: Software Product Lines, Component-Based Software Engineering, Component-off-the-Shelf, COTS, Software Reuse, Software Process Model

Research Paper

[1] Department of Electrical and Computer Engineering, University of Western Ontario, London, Ontario, Canada, N6A 5B9, lcapretz@uwo.ca. (on sabbatical leave – visiting associate professor in the Department of Computer Science at the University of Sharjah, P.O. Box 27272, Sharjah, United Arab Emirates)

[2] College of Information Technology, United Arab Emirates University, P.O. Box 17555, Al-Ain, Abu Dhabi, United Arab Emirates, f.ahmed@uaeu.ac.ae

[3] Computer Science and Information Systems, American University of Kuwait, P.O. Box 3323, Safat, Kuwait, 13034, smaati@auk.edu.kw

[4] Department of Computer Science, University of Sharjah, P.O. Box 27272, Sharjah, United Arab Emirates, zaher@sharjah.ac.ae

1. Introduction

Experience shows that a company can drastically improve its competitive advantage if it optimizes how it develops these product lines. Using product line engineering, some organizations have reduced the number of defects in their products and reduced costs. However, many companies do not use a product line engineering approach when developing their products. More often than not, they either start from a single system, branching off new variants as the need arises and ending up with completely independent code bases, or they start with the different variants as independent projects from scratch.

A software product line is a group of software-intensive systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way (Clement and Northrop, 2001). A software product lines usually starts with the analysis of the common and the variable features supporting a product-line development, and then defining a set of reusable elements that can be customized and combined into new products. The establishment of a software product line is expected to bring dramatic change in software engineers' primary roles and required skills for software development.

A COTS (Component-off-the-shelf) is a self-contained piece of software that provides clear functionality, has open interfaces, and offers plug-and-play services. A product line can be built around a set of COTS by analyzing the products to determine the common and variable features. The product structure and implementation strategy around a set of COTS prepares a platform for several products. A product line based on COTS has broad implications for how software engineers develop and maintain software systems, so this approach is here to stay.

Software product line deals with the assembly of products from existing core assets commonly known as components and there is continuous growth in the core assets as the production proceeds (Weiss and Lau, 1999). This idea has emerged as vital in terms of software development from component-based architecture (Griss, 2001). According to (Jazayeri et al., 2000), product family software architecture defines the concepts, structure, and texture necessary to achieve variation in features of variant products while achieving maximum sharing parts in the implementation. (Meekel et al., 1998) identified three axes of variability among products resulting from software product line: features variability, hardware platform variability and performance variability.

Although software product line is gaining popularity over time due to economical impacts, as asserted by (Ahmed and Capretz, 2007), (Linden, 2002), (Buckle et al. 2004), there has not been a great deal of research in establishing appropriate models for developing software product line from COTS. By having controlled variability and in satisfying the market demands, COTS-based software product line development model has broad implications on how software engineers develop and evolve multiple software products.

2. The Y-model for Component-Based Software Development

Independent work carried out in software reusability, object-orientation, and software architecture has reached a point at which many activities can be integrated to yield a new coherent approach to product-line integration. Traditional software life-cycle models do not encourage reusability within their phases.

Hence, a software life-cycle model that emphasizes the importance of incorporating COTSs during software development for the production of multiple products is in demand. The Y-model as taken in Figure 1 has been proposed as a feasible solution. Our approach focuses on a collection of components within a particular application domain, and encourages reuse of components from reusable libraries within that application domain (Capretz, 2005).

Component reuse involves both "development with reuse" and "development for reuse". Our approach addresses the mechanisms used when components are retrieved from and catalogued into a reusable library of core assets. Initially, the software engineer identifies potentially reusable components from existing reusable libraries of core assets. The components are then selected, adapted and reused through composition, generalization and specialization mechanisms. At the end of software development, there may be many new reusable components need to be validated, classified and stored as core assets.

Based on available experience, the use of the Y-model appears to cover the likely phases of large software development and enforces software reuse. This model supports “development with reuse” through component assembly, as well as “development for reuse” through component cataloguing, so that in the future, such components can be reused in other systems in the same software family.

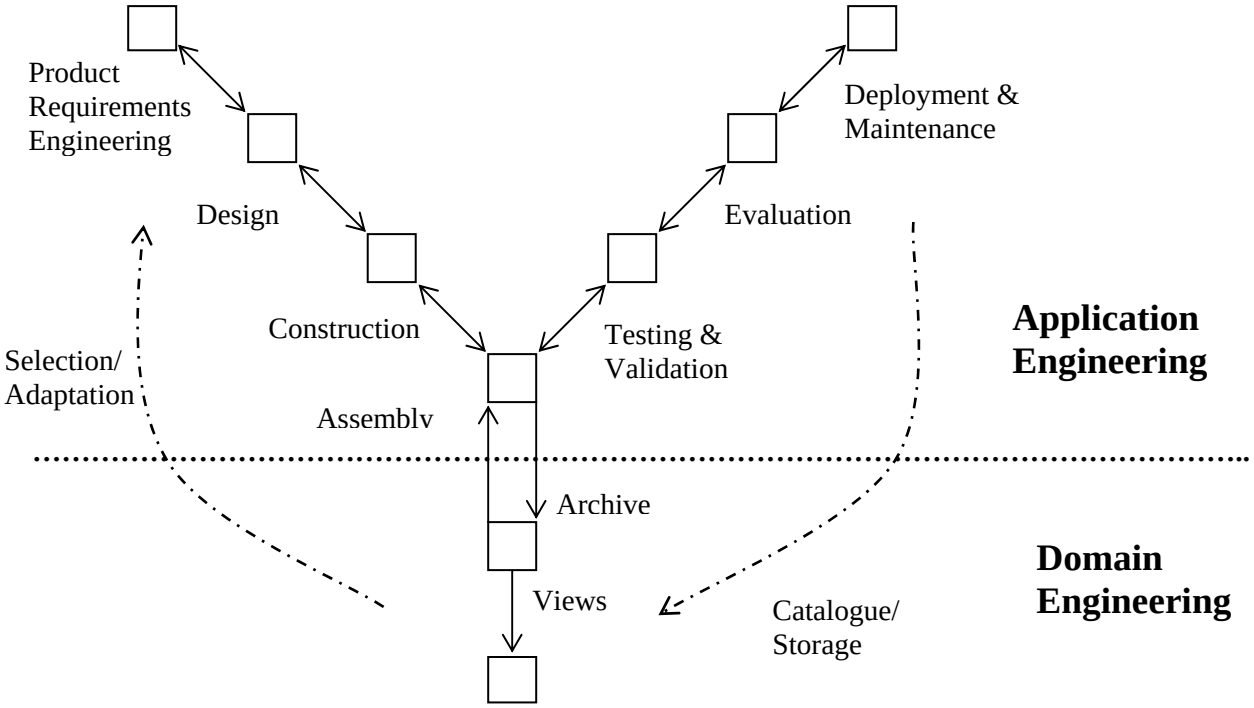

Figure 1: Overview of the Y-model for COTS-Based Software Development

2.1 Domain Engineering Phase of Y-model

The *Domain Engineering* phase of Y-model taken in Figure 2 establishes an infrastructure for software product line and constructs a COTS repository for product development. During the *Domain Engineering* phase, we initiate *Product Line Infrastructure View* and *COTS Archive View*. The iterations in the activities of *Product Line Infrastructure View* and *COTS Archive View* provide feedback to one another. The aim is to generate a COTS repository, which fulfills the product line requirements and meets the production constraints.

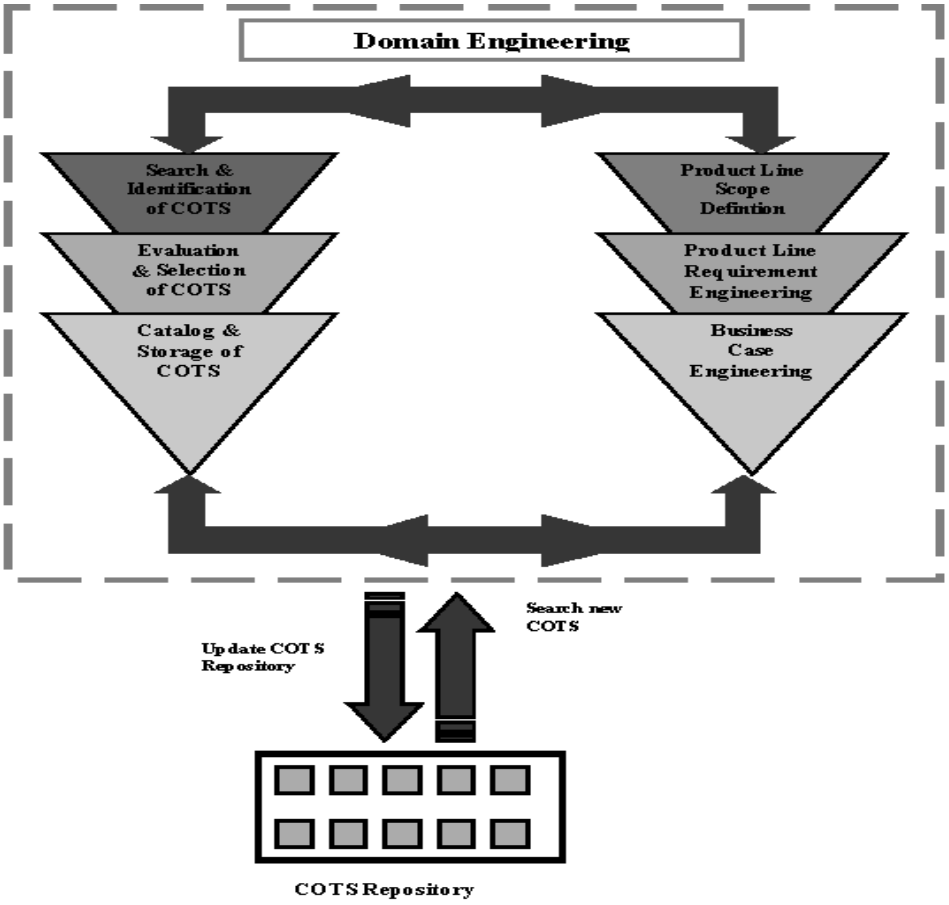

Figure 2: Domain Engineering Phase of the Y-model

2.1.1 Product Line Infrastructure View

Product Line Infrastructure View involves the activities related to conceptualization and initiation of software product line within an organization. This view performs activities that establish an infrastructure for software product line. The *Product Line Infrastructure View* constantly provides feedback to *COTS Archive View* for effective search, identification, evaluation, selection and catalogue/storage. The various activities performed during this view are as following:

1. **Product Line Scope Definition:** Software product line scope identifies the characteristics of the product line and the products that comprise the product line. Software product line scope

definition activity provides iterative feedback to COTS search and identification activity in *COTS Archive View*. This way it ensures that all the searched COTSs are consistent with the scope of product line. The aim is to develop a COTS repository within the scope of the product line, one that can be utilized to develop products.

2. **Product Line Requirement Engineering:** A well-established requirement management activity for the software product line assists in understanding the scope and boundaries of the products to be developed. Product line requirements deal with features or functionalities common to all the products belonging to that family. The requirement engineering for product line gives feedback to COTS selection activity in the *COTS Archive View* to generate a candidate list of COTSs that meets the product line requirements.

3. **Business Case Engineering:** The goals of the software product line are explained by the business cases identified, and they promote the product line. Each product released from the software product line is a valid business case for the organization, and this helps an organization to achieve its financial goal along with the justification of the product line. The identification of business cases helps in evaluating identified COTSs in *COTS Archive View* in order to meet the production criteria and product requirements.

2.1.2 COTS Archive View

COTS Archive View is responsible for building up a COTS repository for the COTS-based software product line. It communicates with *Product Line Infrastructure View* to generate a COTS repository, which fulfills the product line requirements and meets the production constraints. Initially the *COTS Archive Engineer* identifies potential COTSs from existing reusable libraries and open markets based on the software product line requirements and scope. The components are evaluated and selected. The selected COTS is catalogued and stored so that they are readily available for assembly of products to capture market segments. The various activities performed during *COTS Archive View* are as following:

1. **Search & Identification of COTS:** The process of searching and identifying potential COTSs for software product line development starts when we conceptualize the product line by defining the product line scope. The selection of COTSs for software product line involves four steps:

    - Index the COTS by the information that uniquely identifies them. A multi-dimensional index structure, such as R-tree, could be used for this purpose. The R-tree index will narrow the search space and thus reduce the search to $O(\log_m n)$, where n is the total number of COTSs and m is the minimum number of items stored in an R-tree node.
    - Search the index structure for the required (target) COTS, which are within the scope of the software product line.
    - Understand the functionalities provided by the searched COTS.
    - Evaluate COTS adaptation trade-off (specialization, generalization, composition or adjustment).

2. **Evaluation & Selection of COTS:** The search and identification process yields a number of potential COTSs that can be used in the development of various products in a software product line. Those COTSs need to be evaluated at the individual component level as well as at the product line level before they are selected for use in a software product line development:

    - ***COTS Level Evaluation:*** involves evaluating possible quality attributes of a COTS such as reliability, portability, efficiency, etc.
    - ***Product Line Level Evaluation:*** involves evaluating a COTS with respect to integration into the common architecture of the resultant products, its interoperability and its standard compliance to product line and product requirements.

3. **Catalogue / Storage of COTS:** The selected COTS is catalogued and stored in the COTS repository with enough information so that they can be easily traced and retrieved as and when required for assembly. One way to express the association between COTSs involves organizing them through a set of pre-defined categories. Such categories allow COTSs to be classified and correlated with each other in order to be reused. Categories are used to represent information about COTS, and this information can help in solving the problem of discordance of terminology among professionals. When the archive of COTSs is huge, manual categorization of COTSs will be tedious and time consuming; thus, an automatic categorization can be used. A machine learning-technique, such as neural networks, may be used. First, we prepare a training dataset of COTSs whose categories are known *a priori* and train the neural network using this training dataset. After training the neural network, the archive of COTSs is fed into the neural network, which will automatically categorize them.

2.2 Application Engineering Phase of Y-model

In the *Application Engineering* phase of the Y-model taken in Figure 3, actual products are developed from COTSs present in the COTS repository. In this phase, activities of the *Product Line Application View* interact with the activities of the *COTS Utilization View* to produce required products.

2.2.1 Product Line Application View

Product Line Application View interacts with *Product Line Infrastructure View* to identify potential business cases to capture market segment. The *Product Line Application View* generates the product requirements of the potential business case and provides feedback to *COTS Utilization View* to find out which candidate COTS is to be used in product development. The various activities performed during *Product Line Application View* are as following:

1. **Product Requirement Engineering:** Product requirements are composed of a constant and a variable part. The constant part comes from product line requirements in the *Product Line Infrastructure View* and deals with features common to all the products belonging to the family. The variable part represents those functionalities that can be changed to differentiate one product from another. This activity defines the variable part of the product requirement.

2. **Assembly:** The assembly activity involves the development of product. The product requirements guide the assembly process to get feedback from the query activity of *COTS Utilization View* to find out those potential COTSs suitable to be assembled in order to produce the product. If it is required then assembly activity performs specialization, generalization, or adjustment of the COTS.

3. **Product Testing & Validation:** In product testing and validation, products developed from software product line are tested to analyze whether they meet the product line testing and evaluation criteria or not. Specific testing and validation about integration of a COTS ensures that adaptability has no consequences.

4. **Business Case Evaluation:** Business case evaluation identifies the success and failure story of the products developed and deployed. It compares the proposed business case strategy with the outcome of the development and deployment process of products. It studies the market and analyzes the impact of the product in terms of cost to benefit ratio. The study re-establishes the business case identification, keeping in view the market demands and product evaluation.

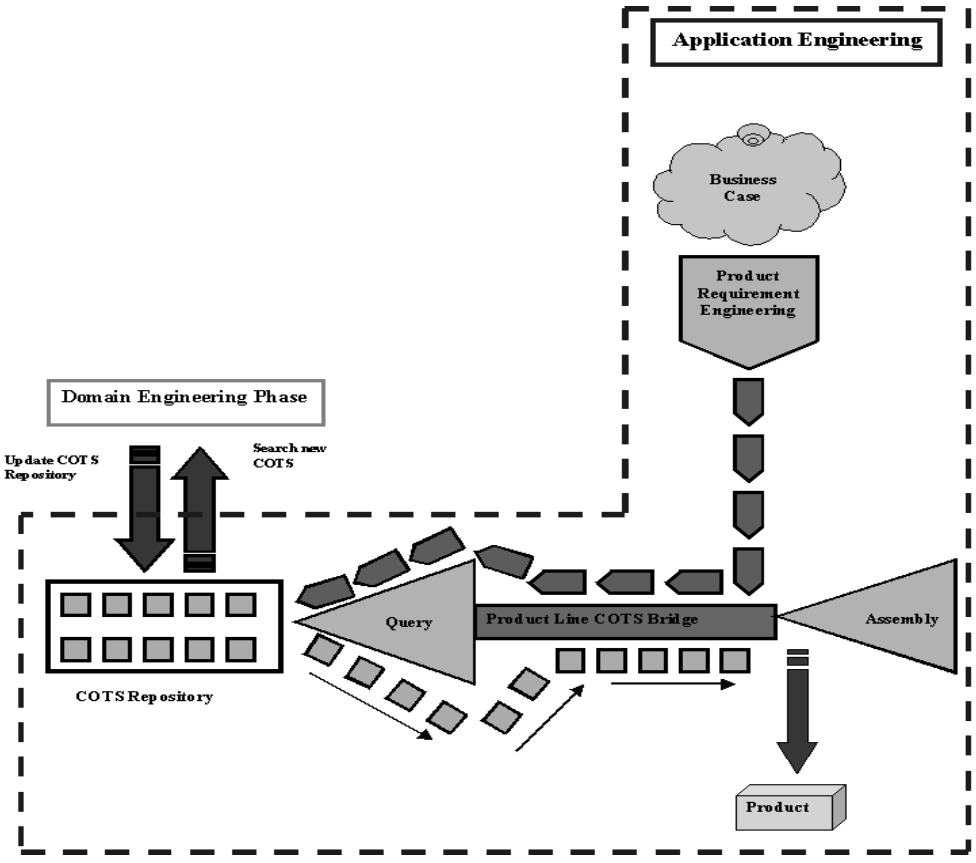

Figure 3: Application Engineering Phase of the Y-model

2.2.2 COTS Utilization View

COTS Utilization View is responsible for providing required COTSs from COTS repository to develop products. *COTS Utilization View* interacts with *Product Line Application View* to receive product requirements and then communicates with *COTS Archive View* to search and retrieve the required COTSs from COTS repository, as developed and maintained by *COTS Archive View*. The various activities performed during *COTS Utilization View* are as following:

1. **Query**: In the query activity of the *COTS Utilization View*, COTSs are searched from the COTS repository in order to develop the product. A well-cataloged COTS repository established by *COTS Archive View* reduces the efforts to trace the suitable COTS for assembly. The product requirements serve as parameter for query activity, and continuously traversing the COTS repository yields the required COTS, either exactly matched, partially matched, or not matched:

    - ***Exact Match:*** An exact match between an available COTS in repository and product requirements is reached. In this scenario the COTS is retrieved from the repository and placed for product assembly.
    - ***Partial Match:*** Some closely matching COTSs are available and requires adaptations in order to match the product requirement. In this scenario, necessary specialization, generalization, or adjustment of the COTS to the requirements is performed before the COTS is ready for product assembly.
    - ***No Match:*** COTS repository is searched thoroughly, but neither an exact match nor a partial match COTS is available to satisfy the product requirement. In this scenario a request for a search of new a COTS is passed to the *Domain Engineering* activity, and this stimulates *Archive View* and *Product Line Infrastructure View* to find the potential COTSs from the open markets and update the COTS repository respectively.

2. **Version Management of COTS:** The COTS, after adaptation, generates versions, which are documented in this activity. A comprehensive version management and dependency link strategy for components and products in software product line provides us with vital information about components and products having a relationship of composition and utilization.

3. **Update COTS Repository:** The software product line develops an initial COTS repository in the *Domain Engineering* phase. As we maintain the software, products resulting from the product line tend to develop new versions of COTS, which must be added to the COTS repository so that they can be reused in later products. The COTS repository is dynamic and continues increasing its size with the addition of a new COTS when required.

3. Component Cataloguing

So far, most of the work that has been done in the reusability arena involves storing and recovering components from reusable libraries, but there are still many problems related to reusing such components. For instance, as a software system becomes mature, the reusable libraries may grow as domain-specific libraries and reusable components can be added over time. It does not take long for such libraries to expand to enormous proportions and often with multiple versions of a component,

which makes it difficult for software engineers to look for components that might meet their needs. Reusable libraries are usually large and their organisation makes it problematic to find potentially reusable components.

Additionally, one of the great difficulties in identifying a reusable component lies in the fact that there is discordance in terminology among professionals, such that a component someone is looking for might be described in unfamiliar or unexpected terminology. Google code search engine (Google, 2007) and Koders (Koders, 2006) have been tackling this problem and have given us some hope. They are both primarily used by software engineers in searching of related samples from the available open source project on the web.

Ideally, the potential re-user of software components must be able to find a connection between what is needed and what is available. One way to express relationships between components of a reusable library involves organizing them through a set of pre-defined relations. Such relations allow components to be classified, and correlated to others that could also be reused. In addition, relations can be used to express a link between different components, facilitating the understanding of the components. Relations used to represent information between two reusable components can help solve the problem of discordance of terminology among professionals because the relations can establish some fixed semantic concepts between components. Four different relations to link components and to express relationships among components have been proposed:

1. *Compose* (<component-1>, <list-of-components>): This relation represents <component-1> as a composition of components in a <list-of-components> (*has-a* relationship). Complex software system behaviour can be achieved with compositions that combine the simple behaviour of several types of components.

2. *Inherit* (<component-1>, <component-2>): This relation indicates that <component-1> is a generalization of <component-2> or the other way round that <component-2> is a specialization of <component-1> (*is-a* relationship).

3. *Use* (<component-1>, <list-of-components>): This relation indicates that <component-1> interacts with components in a <list-of-components> (*uses-a* relationship). It means that any operation of <component-1> uses any service provided by any component in a <list-of-components>.

4. *Context* (<component-1>, <context-1>): This relation associates a <component-1> with a <context-1> defined by the software engineer (*is-part-of* relationship). The <context-1> can be a framework or application domain.

These relationships are vital for data mining - the automated extraction of hidden predictive information from large data sets, such as reusable library of software assets. In this way re-users will have a functionality which may read something like: "Software engineers who used these components also used…" to help the software engineers while browsing the assets. This is just like the Amazon feature that lists other books that were related to the same purchase, and reads "Customers who bought this book also bought…".

4. Component Assembly

The decisions involving the reuse of a component are very important in that the software engineer must select the component that requires the least effort to adapt, with an exact match between what is needed and what is available being the goal. Basically, the selection of a component from a reusable library involves four steps:

1) identifying the required (target) component;
2) selecting potentially reusable components;
3) understanding the components;
4) adapting (specializing, generalizing, composing or adjusting) the components to satisfy the needs of the developing software system.

The search for a component in a reusable library can lead to one of the following possible results:

- An identical match between the target and an available component is reached.
- Some closely matching components are collected, then adaptations are necessary.
- The requirements are changed in order to fit available components.
- No reusable component can be found, so the target should be created from scratch.

Following a procedure which helps select potentially reusable components is vital to the reuse process. The procedure described in Figure 4 illustrates a typical attempt to reuse a component from a reusable library. The procedure describes only the selection and adaptation of reusable components.

While searching for components it is also necessary to address the similarity between the required (target) component and any near matching components. The best component selected for reuse may also require specialization, generalization or adjustment to the requirements of the new software system in which it will be reused. Sometimes, it is preferable to change the requirements in order to reuse the available components. The adaptability of the components depends on the difference between the requirements and the features offered by the existing components, as well as the skill and experience of the software engineer. The process of adapting components is the least likely to become automated in the software reuse process.

5. The COTS Life Cycle

As COTS-based software is produced essentially out of interrelated collections of independently developed components, it is important to understand the stages that such components go through. The stages reflect the activities involving the design, implementation, verification, classification, storage, selection, retrieval, and adaptation of the component. Figure 5 depicts the lifetime of a reusable component.

Reusability not only involves reusing existing components in a new software system but also producing components that are meant for reuse. When a software system has been developed, the software engineer may realise that some components can be generalized for potential reuse in future projects. A component must be easily adaptable for different uses, either in original or in modified form.

Therefore, developing reusable components is considerable more difficult and involves much greater expense then producing ordinary components, although it may still be worth the investment over the longer term.

If a newly implemented component does not exist in the reusable library, then a decision has to be made as to whether the new component should be classified as a reusable component, and to be frozen and validated, then put in a reusable library. The validation is applied only to that component, not to the whole software system and should include treatment of exceptional conditions.

```
begin
  //The component reuse process.
  //Given a keyword for a target component: Search libraries
  //   for potentially reusable components and their relations.
 if (identical match between the target and an available component
     exists)
 then
   //reuse by composition
   retrieve it and reuse the component as it is
 else
   collect fitting components
   for each collected component
      assess the degree of matching
   endfor
   rank the components
   select the best component
   if  (the target shares commonalities with the best component)
   then
     adjust the best component to the requirements or
     adjust the requirements to the best component
   else
     //reusability is not possible
     create the target component from scratch
   endif
 endif
end
```

Figure 4: A Procedure for Component Reuse

Classification of components depends on the experience of the software engineer, and storage issues are straightforward. By properly storing a component using the relations proposed previously, the chances of finding potentially reusable components are increased. The effort required to get a suitable component is reduced because the classification scheme based on relations guides the software engineer through the various relations quickly and efficiently.

Storing a component also involves classifying it, taking it from the software product line, relating it to other components and putting it into the reusable assets library. Selection involves browsing to find a component, retrieving it, and transferring it from the reusable assets library to the software product line.

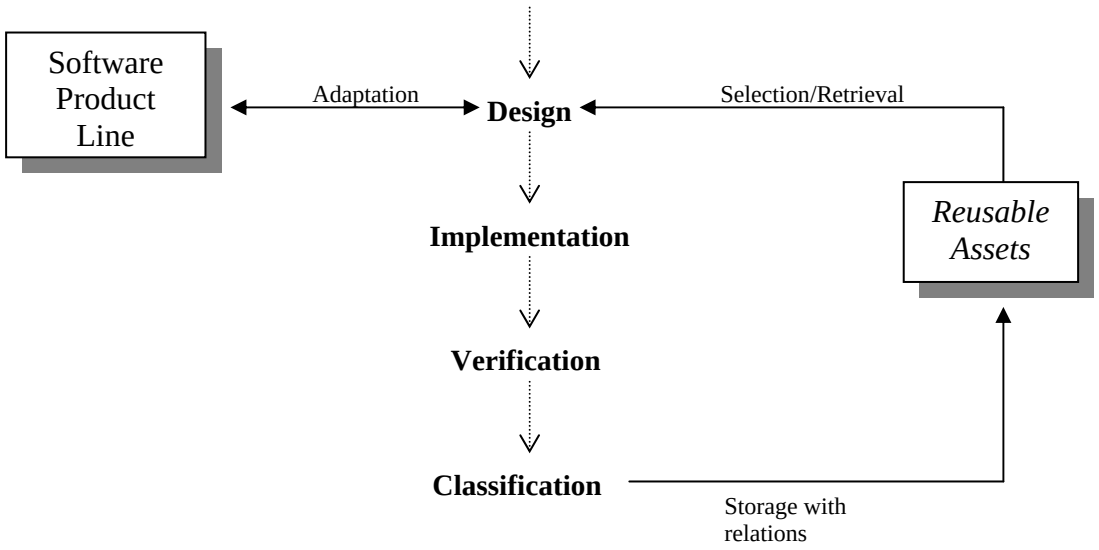

Figure 5: Lifetime of a Reusable Component

A software system is not merely produced out of reusable components. On the contrary, usually, components selected and derived from reusable libraries are combined with newly-written components, and all of them have to be bound together in the final software. It is natural that with some of the components, the software designer will face the decision of whether to reuse them straightforwardly, adapt them then reuse, or write them from scratch. The break-even point of reusing versus redoing is where the cost of search plus adaptation exceeds the cost of producing the respective component.

6. Following the Y-model: A Case Study

One day while Luiz, Faheem, Sheeref, and Zaher were busy in discussions about how they could evaluate the Y-model, somebody knocked at the office door. The man at the door was Luiz's friend Lucas who works for a software development company at the managerial level. He seemed to be upset. When Luiz asked what the problem was, he said:

"*My boss wants to establish a product line from COTS and I don't have any ideas about what to do. He gave me a very short time frame and I think I'm in trouble. You guys have to help me either to establish a product line or to find a new job*."

Luiz glanced at Sheeref and Faheem and all came to the same conclusion that here was the test bed for the Y-model.

Sheeref said, *"Don't worry Lucas, we have developed a model for COTS-based Software Product Line Development that might help you to meet your target."*

Lucas said "*I have no other choice; I came here with very high hopes so tell me what I have to do."*

Luiz asked Faheem to open the Y-model diagram.

Sheeref said, "*First we had to work on the Domain Engineering Phase."*

Lucas asked, *"What is this phase and why do we need to do this?"*

Luiz explained to Lucas that first we have to develop an infrastructure for the product line and an initial COTS repository. The *Domain Engineering phase* will carry out certain activities in the *Product Line Infrastructure View* and *COTS Archive View* to do that job.

Sheeref said, *"Let's get started."*

Luiz asked Lucas if he could explain, *"What was the scope of his product line."*

Lucas asked, "*What do you mean by scope of the product line?*"

Sheeref replied, "*Okay forget about the terminology. Tell me what kind of products you want to produce from this product line*."

Lucas said, "*My boss wants to establish a production facility to create products to carry out commercial activities like selling and purchasing electronically, particularly via open networks like the Internet.*"

Luiz said, "*Here is your scope, now we can move ahead*."

Faheem looked at the diagram of the Y-model and asked Lucas, "*Is there anyone in your office who is good at searching and browsing the Internet for COTS, because, if there is, that person starts searching and identifying the COTS."*

Lucas replied, *"Yes, Zaher always does this for us whenever we want to have COTS*."

Sheeref called Zaher, and explained to him what kind of COTS he had to search and asked him to prepare a list of components.

Luiz was more curious about the product line requirements because he thought that if we could prepare them by the time Zaher had searched the COTS we would be able to figure out what exactly we needed.

Luiz asked Lucas, *"Okay tell me your product line requirements?"*

Again Lucas complained, "*Don't use this terminology. It confuses me. Explain to me what you want to know."*

Faheem elaborated, "*What kind of functions should all the products be able to perform?"*

Lucas quickly answered, "*The resulting E-Commerce products must provide traditional commercial and specific online activities, for example it must provide product information, conduct online retail in virtual malls, and publish digital information."*

Sheeref asked Lucas, *"Could you categorize the activities?"*

Lucas said, *"Okay, well there should be a Business Website comprised of a Payment Processing System, a Store Management System and a Shopping Cart."*

The phone rang. It was Zaher and he was anxious to tell us what he had found.

Luiz asked him if he had found things like *Business Website Templates, a Payment Processing System, a Store Management System and a Shopping Cart* in his list.

Zaher was delighted to tell us he had a huge list of these things.

Sheeref was not eager to move ahead with the Y-model until all the COTS that Zaher had searched were evaluated.

Faheem pointed out that before we could evaluate the COTS we must do the business case engineering, because it would help in evaluating the COTS.

Lucas said, "*What is this, I'm not a businessman.*"

Luiz explained, "*It is very important to know what the business cases are for which you are establishing this product line."*

Lucas said, "*There may be products like an Online Pharmacy, E-Book Shop, Online Auto Part Shop, Online Car Rental and Online Air Line Ticket Sale etc."*

Luiz said to Faheem, "*Faheem, I'm going for lunch with Lucas and Sheeref. In the meantime, please coordinate with Zaher and evaluate the COTS to ensure that they are fulfilling our product line requirements and make different categories of them."*

Faheem said, *"So, you want me to finish up with the Product Line infrastructure View and COTS Archive View."*

Lucas asked Faheem meant about the *"Views"* and what the outcome was of the activities we have done so far?"

Sheeref said, *"Don't worry Lucas, when we come back from lunch Faheem will have established a COTS repository and an infrastructure for your product line."*

When Luiz, Sheeref and Lucas came back from lunch, Faheem was eager to show them Table I that illustrates the initial COTS repository, which he and Zaher prepared.

Luiz said, *"Okay Lucas, now you have the COTS repository and an infrastructure of the product line. Which product do you want to develop first?"*

Faheem pointed out that now we are moving into the *Application Engineering Phase* of the Y-model.

Lucas asked, *"What is this Application Engineering Phase?"*

COTS Category	COTS Name	Vendor	COTS Assigned Code	Description of Selected COTS
Business Website	ASP-Template	HyperTemplates	BW-1	ASP business website template
	HTML-Template	Boxed Art	BW-2	HTML business website template
	PHP-Template	Webmasters Plaza	BW-3	PHP business website template
Payment Processing	PayPal	eBay	PP-1	Payment processing COTS for online secure transactions
	WorldPay	WorldPay	PP-2	Payment processing COTS for online secure transactions
Store Management	UStore	Superfreaker Studios	SM-1	ASP based Microsoft Access content management system
	ASPVendor	CJW Soft	SM-2	ASP based Microsoft SQL Server product management system
Shopping Cart	CactuShop	CactuSoft	SC-1	ASP based shopping cart tool to support shopping and order processing
	SquirrelCart	Lighthouse Development	SC-2	PHP based shopping cart tool to support shopping and order processing

Table I: Selected COTS for E-Commerce Software Product Line

Sheeref explained to Lucas, *"The Application Engineering phase of the Y-model deals with the development of products from COTS present in the COTS repository. In this phase, we will perform activities of the Product Line Application View and COTS Utilization View to produce the required products."*

Lucas asked, *"Please explain what these views will do?"*

Faheem said, *"The Product Line Application View generates the product requirements of the potential business case and provides feedback to the COTS Utilization View, so the user can find out which COTS is to be used in the product assembly."*

Lucas said, *"Okay, my boss was asking for an online pharmacy."*

Sheeref was interested in doing the *Product Requirement Engineering*; he asked Lucas, *"Is there any specific requirement for that?*

Lucas said, *"Yes, he wants an ASP-based website, an online transaction system from Pay Pal, and a MS Access based database."*

Sheeref said to Faheem, *"Be careful now. We are going to start assembly and query activities."*

Faheem queried Zaher, "*Could you please find COTS from the COTS repository that matches these requirements?"*

He replied, "Yes, Here you go!" and gave us BW-1, PP-1, SC-1 and SM-1.

Lucas asked, *"Can we assemble the product right away?"*

Luiz said, *"If the COTS are exactly match then yes, otherwise, no, because you need to perform adaptation so that you can fulfill the individual product requirements."*

Sheeref highlighted the changes to be performed in the COTS in order to assemble the *Online Pharmacy*, the one Lucas's boss wants.

Faheem emphasized to Zaher the need to update the information of new versions of COTS in the COTS repository.

Zaher said, "*I know that BW-1 has version BW-11 whereas PP-1 has version PP-11."*

Luiz asked Faheem to look at the Y-model diagram and to tell them what else should be done now.

Faheem said, *"We need to test this Online Pharmacy that we have developed. That's what the Y-model says. And we need to evaluate the business case."*

Lucas said, *"Don't worry. I will ask my boss to look at it and evaluate this business case."*

Lucas inquired, *"What happens when we don't have any matched or partially matched COTS in our repository?"*

Faheem told him *"You have to go through the Domain Engineering phase to search again for new COTS in order to update your COTS repository."*

Luiz explained to Lucas, *"This is the way you establish a COTS-base software product line. We did a quick exercise to explain to you the various steps of the Y-model. Now you have to go through carefully all these steps in your office and understand the various views of the Y-model. You will need to establish a team and let them understand the collaboration protocols. We have shown you that the Y-model works to establish a COTS-based Software Product Line."*

Lucas left with high hopes and after few days conveyed to us that he has established a COTS-based software product line in his office and has produced four *E-Commerce Applications* so far and is in the process of producing more.

7. Final Remarks

This paper has provided an overview of a pragmatic approach to COTS-based development. Starting from the set of common and variable features needed to support a software product line, we can systematically develop and assemble the reusable elements needed to produce the customized components and component infrastructures needed to implement the family members of the product line.
There is still a need for tools to support the management of software families, which are specific to particular application domains. A software family comprises a group of software systems that expresses a general solution for a family of related applications in that application domain. Therefore, a component will not be as generally useful outside the application domain because it contains domain-dependent components. However, it is sometimes beneficial to adapt the developing software so that it fits into a software family, resulting in a tremendous gain in productivity.

The Y-model for COTS-based software product line development highlights various activities of software product line and COTS-based development. The model integrates the concept of software product line with COTS to come up with a prescribed way of establishing COTS-based software product line capable of producing multiple products within an application domain. The interdependence of various activities of software product line and COTS shows a strong relationship within a common framework of product development. In order to validate the model, we conducted a case study based on E-Commerce application, which revealed that productivity, in terms of cost, time and quality, increased when we followed the Y-model for COTS-based software product line.

Finally, the use of software product lines as a platform for larger systems is becoming increasingly commonplace. The shift from custom development to software family is occurring in both new development and maintenance activities. Shrinking budgets, accelerating rates of COTS enhancements, and expanding systems requirements are all increasing the need for software product lines. If done properly, this shift can help establish a sustainable practice. We believe that software product lines and COTS-based development encompass the best practices of software engineering and hold the promise of improving the quality of software as well as the productivity of software engineers.

References

Ahmed, F. and Capretz, L. F. (2007), "Managing the business of software product line: an empirical investigation of key business factors", *Information and Software Technology*, Issue 49, pp. 194-208.

Buckle, G., Clements, P., McGregor, J. D., Muthig, D. and Schmid, K. "Calculating ROI for software product lines", *IEEE Software*, Vol 21, No 3, pp. 23-31.

Capretz L. F. (2005), "Y: a new component-based software life cycle model", *Journal of Computer Science*, Vol 1, No1, pp. 76-82.

Clement, P. and Northrop, L (2001), *Software Product Lines: Practices and Pattern,* Addison Wesley, New York.

Google (2007), Google Code Search Engine, http://www.google.com/codesearch.

Griss M. L. (2001), "Product-line architectures", in Heineman, G. T. and Councill, W. L. (Eds.), *Component-Based Software Engineering*, Addison-Wesley, New York, pp. 405-419.

Jazayeri, M., Ran, A., van der Linden, F. (2000) *Software Architecture for Product Families: Principles and Practice*, Addison-Wesley, New York.

Koders (2006) Koders Source Code Search Engine, http://www.koders.com.

Linden, F. (2002), "Software product families in Europe: the Esaps & Café projects", *IEEE Software*, Vol 19, No 4, pp. 41-49.

Meekel, J., Horton, T., and Mellone, C. (1998), "Architecting for domain variability", *2nd International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, Lecture Notes in Computer Science, Vol 1429, pp. 205-213.

Weiss, D. M. and Lai C. T. (1999), *Software Product-Line Engineering: A Family-Based Software Development Approach*, Addison-Wesley, New York.