International Journal of Computer Applications (0975 – 8887)
Volume 2 – No.1, May 2010

Component Based Software Engineering

Arvinder Kaur
Student, GNDEC, LDH

Kulvinder Singh Mann
Assistant Professor , GNDEC, LDH

Abstract

Traditional software estimation models are directed towards large monolithic software development projects. Contemporary software development practices require a new approach to software cost estimation. Contemporary development practices characterize a software application as interacting, independent components. Component-based development offers many potential benefits such as a greater reuse.
Component based software development approach is based on the idea to develop software systems by selecting appropriate off-the shelf components and then to assemble them with a well-defined software architecture. Software community faces a major challenge that is raised by fast growing demand for rapid and cost-effective development and maintenance of large scale and complex software systems. To overcome the challenge, the new trend is to adopt component based software engineering (CBSE).The key difference between CBSE and traditional software engineering is that CBSE views a software system as a set of off-the-shelf components integrated within appropriate software architecture. CBSE promotes large-scale reuse, as it focuses on building software on building software systems by assembling off-the-shelf components rather than implementing the entire system from scratch. CBSE also emphasis on selection and creation of software architecture that allow systems to achieve their quality requirements .As a result, CBSE has introduced fundamental changes in software development and maintenance.

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

1. Introduction

Software is the heart of many industrial systems in today. Added value to products is to a large extent provided by the software. Furthermore, production cost reduction is imperative and is often achieved by introducing software that permits the use of less complex hardware. Domains in which the use of software is now essential include the automotive, medical systems, process control, and manufacturing industries. Industrial products are often systems consisting of software and hardware, the software part being referred to as a software system incorporating many software programs or applications that must cooperate without fail.Delivering a software product on time, within budget, and to an agreed level of quality is a critical concern for many software organizations. Underestimating software costs can have detrimental effects on the quality of the delivered software and thus on a company's business reputation and competitiveness. On the other hand, overestimation of software cost can result in missed opportunities to funds in other projects.

Component-based Software Engineering (CBSE) has emerged as a technology for rapid assembly of flexible software systems. CBSE combines elements of software architecture, modular software design, software verification, configuration and deployment. Component-based software engineering (CBSE) is an approach to software development that relies on software reuse. It emerged from the failure of object-oriented development to support effective reuse. Components are more abstract than object classes and can be considered to be stand-alone service providers. In CBD, software systems are built as an assembly of components already developed and prepared for integration. The main advantages of the CBD approach include effective management of complexity, increased productivity, a greater degree of consistency, and a wider range of usability and extendibility, reduced time to market.

2. Background

Definitions

Component:-A component is an existing piece of software written with reuse in mind that can be deployed with little or no modification. We assume that components are designed to be used in certain types of applications, which implies that there are constraints regarding the incorporation of a component into a system. Components can be
obtained in-house or of-the-shelf.
Architecture: - Software architecture deals with the definition of components, their external behavior, and how they interact .In this context architecture contains a description of component needs or roles. The process of architectural definition can be viewed as a number of architectural decisions that need to be made. We are chiefly concerned with the way in which these decisions affect the components needed by the system.
Architecture can be expressed informally; or using modeling is useful because it may uncover more subtle architectural assumptions; however, the method we present does not rely on any specific technique for specifying architecture.
Architectural Approach:-The process of defining architecture involves decisions about components and their interactions. An architectural approach captures instances of these decisions, thereby partially specifying the components needed by a system and restrictions on how they interact.
Implementation Approach: - The specification of an architectural approach provides us with a list of components needed. When an architectural approach is combined with a set of actual of-the-shelf components that meet these needs, the result is an implementation approach.

3. Methodologies

Component-based software systems are developed by selecting various components and assembling them together rather than programming an overall system from scratch, thus the life cycle of component-based software systems is different from that of the traditional software systems. The life cycle of component-based software systems can be summarized as follows
a) Requirements analysis
b)Software architecture selection, construction, analysis, and evaluation
c)Component identification and customization
d) System integration
e) System testing
f) Software maintenance

a) Requirements analysis:- Component requirement analysis is the process of discovering, understanding, documenting, validating and managing the requirements for components.

b) Software Architecture Selection:-The objective of this phase is to select the architecture of the component according to the user requirements .In this we will construct the component and that component can be Component off the shell (COTS) component.

c) Component Identification and Customization:- Identification of the component can be done by selecting the right components in accordance to the requirement for both functionality and reliability. Component Customization is the process that involves: - 1) Modifying the component for specific requirement. 2) Doing necessary changes to run the component on special platform. 3) Upgrading the specific component to get a better performance and higher quality.

d) System Integration: It is the process of assembling components selected into a whole system under the designed system architecture. The objective of system integration is the final system Composed of several components**.**

Interactions As The Origin Of The Integration Problem

Integration problems arise when a component depends on certain assumptions concerning its interactions with its environment, but is to be placed into a system that is based on different assumptions. The result is interaction protocol mismatches. We define four types of interactions:
i)Component-platform interactions. A component must be executed somewhere. It can be either a real processor or an operating system for binary executables, or a virtual one. If an executable program was compiled for one type of CPU, it will need an emulator **or** a code converter in order to run it on another CPU.

II) Component-hardware interactions. A component can interact directly with hardware writing-reading from ports. If the port's numbers are different from what is expected by the component, the component must undergo some modification.

III) Component-user interactions. A component's user interface requirements may also change. For example, a component can have its messages in one language, when the system requires another language.

IV) Component-software interactions. A component almost always interacts with other software components, and there can be mismatches between the components. A set of possible mismatches between components representation, communication, packaging, synchronization, semantics, control etc. Although all four types of interactions can cause problems for a component reuse and must be overcome, the main concern of this
study is component-software interactions.

e) System Testing: - System testing is the process of evaluating a system to

I) confirm that the system satisfies the specified requirements.
ii) Identify and correct the defects in system in system implementation.
The objective of system testing is the final system integrated by components selected in accordance to the system requirements.

f) System Maintenance:-It is the process of providing service and maintenance activities needed to use the software effectively after it has been delivered.

4) Strategy for Selecting Component

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. A method that addresses the selection process of packaged, reusable software, or OTS as we refer to it in this paper. The method, called OTSO, supports the search, evaluation and selection of reusable software, and provides specific techniques for defining the evaluation criteria, comparing the costs and benefits of alternatives, and consolidating the evaluation results for decision making .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 (*deployed*), 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. While Figure 1 presents the overall OTSO process

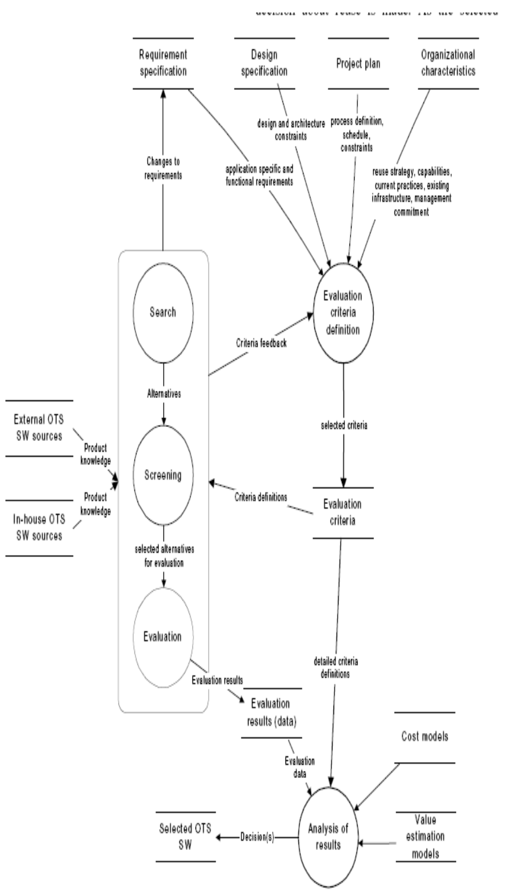
decision about reuse is made. As the selected
Requirement specification
Design specification
Project plan
Organizational characteristics
Changes to requirements
application specific and functional requirements
design and architecture constraints
process definition, schedule, constraints
reuse strategy, capabilities, current practices, existing infrastructure, management commitment
Search
Evaluation criteria definition
Criteria feedback
Alternatives
selected criteria
External OTS SW sources
Product knowledge
Screening
Product knowledge
In-house OTS SW sources
Criteria definitions
Evaluation criteria
selected alternatives for evaluation
Evaluation
Evaluation results
Evaluation results (data)
detailed criteria definitions
Cost models
Evaluation data
Selected OTS SW
Decision(s)
Analysis of results
Value estimation models

Fig.1 Overview of the OTSO method process

Evaluation criteria definition

The evaluation criteria definition process essentially decomposes the requirements for the COTS into a hierarchical criteria set. Each branch in this hierarchy ends in an evaluation attribute: a well-defined measurement or a piece of information that will be determined during evaluation. This hierarchical decomposition principle is analogous to the GQM method. The evaluation attributes should have clear operational definitions so that consistency can be maintained during evaluation.
The criteria set is specific to each COTS selection case but most of the criteria can be categorized into four groups: functional requirements for the COTS; required quality characteristics, such as reliability, maintainability and portability ,business concerns, such as cost, reliability of the vendor, and future development prospects; and relevant software architecture, such as constraints presented by operating system, division of functionality in the system or specific communication mechanisms between modules. It is possible to identify three different sub processes in the definition of evaluation criteria. Figure 2 presents these processes graphically using the modified dataflow diagram (DFD) notation. First, when the available alternatives are searched and surveyed it is necessary to define the main search criteria and the information that needs to be collected for each alternative. The search criteria are typically based on the required main functionality (e.g., "visualization of earth's surface" or "hypertext browser") and some key constraints (e.g., "must run on Unix and MS-Windows" or "cost must be less than $X'). An effective way to communicate such requirements is to use an existing product or COTS as a reference point, i.e., defining the functionality search criteria as "hmk for COTS that are similar to our prototype". The search of alternatives should try to cover breath more than depth. It is enough to define the survey criteria broadly so that the search is not unnecessarily limited by too many constraints. The search phase uses the criteria and determines the "qualifying thresholds", which are in deciding which alternatives are selected for closer The search of alternatives should try to cover breath more than depth. It is enough to define the survey criteria broadly so that the search is not unnecessarily limited by too many constraints. The search phase uses the criteria and determines the "qualifying thresholds", which are in deciding which alternatives are selected for closer evaluation.

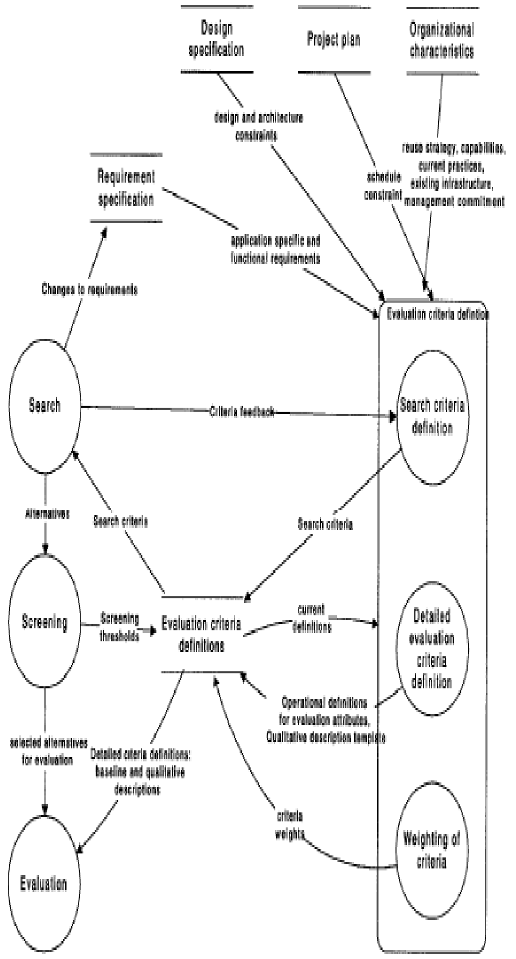
Design specification
Project plan
Organizational characteristics
design and architecture constraints
Requirement specification
schedule constraint
reuse strategy, capabilities, current practices, existing infrastructure, management commitment
application specific and functional requirements
Changes to requirements
Evaluation criteria definition
Search
Criteria feedback
Search criteria definition
Alternatives
Search criteria
Search criteria
Screening
Screening thresholds
Evaluation criteria definitions
current definitions
Detailed evaluation criteria definition
Operational definitions for evaluation attributes, Qualitative description template
selected alternatives for evaluation
Detailed criteria definitions: baseline and qualitative descriptions
criteria weights
Weighting of criteria
Evaluation

Fig2. Evaluation Criteria Definition Process

5. Case Study

The case study dealt with the selection of a hypertext browser for the EOS information service. This case study included a comparison between two analysis methods, the AHP method and a weighted scoring method. A total of over 48 tools were found during the search for possible tools. Based on the screening criteria, four of them were selected for hands on evaluation. The evaluation criteria were derived from existing, broad requirements. However, as in the first case study, the requirements had to be elaborated and detailed substantially during this process. This case study further supported our conclusion of the low overhead of the OTSO method. Furthermore, this case study involved several evaluators, and our criteria definition approach improved the efficiency and consistency of the evaluation.

6. Conclusions

It is concluded that:-1) CBSE is a reuse-based approach to defining and implementing loosely coupled components into systems.
2) A component is a software unit whose functionality and dependencies are completely defined by its interfaces.
3) The life cycle of the component model defines a set of standards that component providers and composers should follow.
The OTSO method was developed to consolidate some of the best practices we have been able to identify for COTS selection. The detailed evaluation criteria also contribute to the refinement of application requirements. The requirements driven, detailed evaluation criteria definition seemed to have a positive impact on the evaluation process.
The experiences from case studies indicate that our method is feasible in an operational context it improves the efficiency and consistency of evaluations, it has low overhead costs, and it makes the COTS selection decision rationale explicit in the organization.

7. References

1) J. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection,"1996. Proceedings of the 18th International Conference on Software Engineering.
2) J. Kontio and S. Chen, "Hypertext Document Viewing Tool Trade Study: Summary of Evaluation Results," 441-TP-002-001, 1995. ECS project Technical Paper. Hughes Corporation, ECS project.
3) 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.
4) T. Davis, "Toward a reuse maturity model,"eds. M. L. Griss and L. Latour. pp. Davis_t-1-7, 1992. Proceedings of the 5th Annual Workshop on Software Reuse. University of Maine.
5) C. Syzperski. Component Software: Beyond Object-Oriented Programming. Addison- Wesley, 1998.
6) G. T. Heineman and W. T. Council. Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, 2001.Component-Based Software Engineering:
7)Technologies, Development Frameworks, and Quality Assurance Schemes Xia Cai, Michael R. Lyu, Kam-Fai Wong The Chinese University of Hong Kong Hong Kong Productivity Council {xcai@cse, lyu@cse, kfwong@se}.cuhk.edu.hk
8) Introduction to component based software engineering Ivica Crnkovic Mälardalen University, DepartmentofComputerEngineering,Sweden,ivica.crnkovic@mdhse