Quality Enhancement in Reusable Issues in Component – Based Development

Lovepreet Kaur (M.Tech)
Department of Computer Science & Applications
Patiala Institute of Engineering & Technology for Women
Patiala (Punjab)-147001

Abstract - Component –based development (CBD) advocates the acquisition, adaptation, and integration of reusable software components to rapidly develop and deploy complex software system with minimum engineering effort and resource cost .Software reusability is an attribute that refers to the expected reuse potential of a software component. Software reuse not only improves productivity but also has a positive impact on the quality and maintainability of software products. The paper first discuss CBD and its associated challenges, and later outlines the issues concerning component reusability and its benefits in terms of cost time-savings. Guidelines are presented to further assist software engineers in the development of reusable software products and to extract reusable components from existing software. Quality and productivity improvement activities within organisations adopting CED can also benefit from the adoption of these guidelines.

Keywords: Software components, component-based development (CBD), software reuse.

I. INTRODUCTION

Component-based development (CBD) has received considerable attention among software developers, vendors and IT organisations. A marketplace for software components is emerging [6].Component –based development has evolved from previous design and programming paradigms.CBD is both a subset as well as an extension of current software engineering practices. The prospect of increased product reliability and stability with shorter development time and reduced cost continues to fuel the on-going interest in CBD. This approach advocates the acquisition, adaptation, and integration of reusable software components, including commercial-off-the-shelf (COTS) products, to rapidly develop and deploy complex software systems with minimum engineering effort and resource cost [12]. Although component-based development offers many potential benefits, such as greater reuse and a commode it oriented perspective of software, it also raises several issues that Developers need to consider [3].

Reuse of software has been cited as the most effective means for improvement of productivity in software development projects [2, 9]. Reuse of software in a development project is generally assumed to increase productivity, improve product reliability, and Lower overall costs. In fact, several software development projects have reported productivity increase up to 50% with high levels of software reuse.

Component-based software engineers intend to define and describe the processes required to assure timely completion of high quality, complex software systems that are composed of a
Variety of pre-produced software components. Evidence of the wide spread interest in CBD within industry and academies include [11]:

- The continuing adoption of COTS software solutions to on-going complex software development projects across all application domains.
- The continuing push for standardisation of protocols, frameworks, and semantics among product vendors to support better interoperability and integration between COTS products.
- The continuing dominance of CBD-related issues being presented, published, and addressed at prestigious software engineering conferences, symposia, and workshops.
- The continuing increase in effort and improvement for the development of better component-level design, implementation, testing, packaging, and documentation techniques.

Potential CBD quality attributes include: reusability, maintainability, accuracy, clarity, replace ability, interoperability, scalability, performance, flexibility, adaptability, and reliability[14].Enabling technologies have the potential to accelerate time-tomarket,
Integrate disparate applications, and ensure consistency and connectivity across the supply chain. Although the enabling technologies are still maturing and there are complex market Forces at work, the state of the practice in many organisations falls well short of realizing the potential of these technologies [1].The adoption of component-based development brings with it many changes. The following are a few significant lessons learnt through past experiences of component-based development [11].

- CBD can be based on a component reference model.
- CBD facilitates parallel development.

- CBD requires pros and cons of reuse to be analyzed.
- CBD offers immutability of components.
- CBD adoption offers prototypes, which are quit advantageous.
- CBD encourages a two-tier error message structure.
- Testing strategies are altered to support changes that
- CBD makes to the project life cycle.

In particular, developers of large-scale and mission-oriented applications require many additional capabilities including [4]:

- Re-engineer legacy applications to harvest existing
- Components reusable in other applications, or replaceable by newer technologies.
- Find suitable components both locally and externally.
- Integrate components implemented in a variety of different technologies.
- Validate a component's behaviour before using it.
- Manage multiple implementations of the same
- Component in different technologies and as it evolves over time.

This paper is structured into three remaining sections. Section 2 provides an overview of software reusability and discusses its pros and cons. Section 3 proposes some reusability guidelines for CBD. Conclusions are presented in section 4.

II. SOFTWARE REUSABILITY OVERVIEW

Software reusability is an attribute that refers to the expected reuse potential of a software component. The software development community is gradually drifting toward the promise of widespread software reuse, in which any new software system can be derived virtually from the existing code. As a result, an increasing number of organisations are using software not just as all-inclusive applications, as in the past, but also as component parts of larger applications. In this new role, acquired software must integrate with other software functionality. The move toward reuse is becoming so widespread that it has even changed software industry's vocabulary. For example, software acquired externally is described with such terms as commercial-off-the-shelf (COTS) or third-party software, commercial available software (CAS), and non-developmental item (NDI). When used as parts of systems, they are called components, component ware, and so on. The systems themselves are known as component-based software or systems of systems [13].There are good reasons why the industry is moving toward large scale reuse, including savings in time and money. The cost to Develop a new system from scratch is significant. This has made custom software development very expensive. Embedding large grained software components into multiple applications also lets one spread component development costs across each application. This allows for a wider variety of software systems to enter the market at lower costs.

Reuse of software components is justifiable if the cost of reuse is less than the cost of developing new components. Hence one empirical attribute for reusability is the effort or cost required to reuse a certain software component. The amount of work needed to reuse a component in another system of the same domain may be determined on the basis of the scale provided in [7,8]. Similar considerations are taken into account by Caldera and Basil [5]
Who propose a model defining three reusability factors:

- Cost of reuse
- Usefulness of reusable components
- Quality of the reusable components.

III. REUSABILITY GUIDELINES FOR CBD

Productivity and low quality are still the biggest problems in software engineering. The fundamental cause of "software bottleneck" is that new software systems are usually developed from scratch [7]. However, all of the already designed, implemented, documented, and tested software contains very much knowledge and experience. Remarkable benefits could be gained if useful information from the enormous mass of existing software could be extracted somehow. For a long time, software developers have realised that software reuse is the way to utilise the existing knowledge and work already done when building new applications. Software reuse not only improves productivity; it also has a positive impact on the quality and maintainability of software products [7, 10].

It is generally assumed that the reuse of existing software will enhance the reliability of a new software application. This concept is almost universally accepted because of the obvious fact that a product will work properly if it has already worked before. Some general reusability guidelines, which are quite often similar to general software quality guidelines, include [10]:

- Ease of understanding
- Functional completeness
- Reliability
- Good error and exception handling
- Information hiding
- High cohesion and low coupling
- Portability
- Modularity

CBD lacks appropriate reusability guidelines that could further benefit component-based development from cost-savings, time savings, quality and productivity improvements, reliability improvements, etc.

In order to augment the level of software reusability in CBD, the following high-level guidelines are suggested:

- Conducting software reuse assessment
- Performing cost-benefit analysis for reuse

- Adoption of standards for components
- Selecting pilot project(s) for wide deployment of reuse.
- Identifying reuse metrics

IV. CONDUCTING SOFTWARE REUSE ASSESSMENT

Software reuse assessment is the first step to be initiated by a software development organisation. Software reuse assessment is performed to measure the potential for practicing reuse in an organisation, to determine if the organisation is ready to embark on a reuse programme and to define where to focus its reuse efforts in order to gain the maximum benefit from practicing reuse. The result of reuse assessment can be used as the basis for defining the organisation's reuse goals, reuse adoption strategies, the domains in which to practice reuse and the reuse programme implementation plan [15].

Reuse Assessment is performed to help successfully introduce reuse into software development organisations. The purposes of the reuse assessment are to:

- Evaluate the organisation's current reuse strategy and the implementation of that strategy in current software projects and various systems groups.
- Use the results of the assessment to determine an organisation's reuse goals, elements of reuse program to achieve those goals, and domains in which to focus reuse efforts.
- Recommend actions to take to implement its reuse strategy.

Instituting the practice of reuse across an organisation is a large and complex task, and its success requires careful planning, cooperation and good management practices. To ensure success an organisation needs to determine how ready, willing and able it is to practice a reuse-driven development approach and what actions it needs to take to prepare itself to accomplish its reuse objectives and goals.

The assessment should investigate technical and management/organizational reuse issues. On the technical side, some important issues include:

- Identifying and defining core business objects and other kinds of reusable components
- Defining guidelines and standards for reuse
- Defining the organizational structure and classification scheme for the reuse

On the management/organizational side, issues include:

- Defining personnel support for core business objects/reusable component
- Establishing reuse training programs
- Establishing the reuse infrastructure (i.e. reuse metrics and measurements, corporate reuse
- Policy, reuse incentives).

V. PERFORMING COST-BENEFIT ANALYSIS FOR REUSE

Cost-benefit analysis of software reuse provides valuable information that helps organizations decide whether or not reuse is a worthwhile investment. Such an analysis can be conducted using well-established economic techniques. Cost-benefit analysis alone should not serve as the sole criterion in deciding whether or not to pursue reuse.

The net cost savings for reuse can be estimated as [17] below in equation (1):

$$\mathbf{Csave = Cs - Cr - Cd} \qquad \mathbf{(1)}$$

Where Cs is the cost of the project developed from scratch, Cr is the overhead costs associated with reuse, and Cd is the actual cost of the software as delivered.

Cs can be determined by applying one of the many cost estimation models. The overhead costs associated with Cr include:

- Domain analysis
- Increased documentation to facilitate reuse
- Maintenance and enhancement of reuse artefacts (documents and components)
- Royalties and licenses for externally acquired components
- Creation (or acquisition) and operation of a reuse repository
- Training of personnel in design for reuse and design with reuse.

The actual cost of the software, Cd , will include project-related reuse costs, such as the adaptation and integration of reuse artefacts.

A cost-benefit analysis can also be viewed from two perspectives [17]:

- That of the producer, a creator of reusable components, and
- That of the consumer, a user of these components in the creation of other software.

It is possible for a component to be economically feasible for the consumer but not for the producer. Producer costs include those incurred from creating and maintaining a reuse program and reusable components. In the case where the producer does not Explicitly charge for its components or services, its benefits are simply those that are enjoyed by its consumers, namely, reduced costs, avoided costs, and in some cases, increased profits.

The Net Present Value (NPV) technique is a well-established and popular method for conducting a cost–benefit analysis [17]. The NPV method determines the present value of a stream of cash flows that result from creating a component or establishing a reuse program. The technique can be used to determine the attractiveness of reuse as an investment compared with other software development strategies.

Here, costs refer to total life cycle costs, or those incurred from investigating, designing, coding, testing, debugging and maintaining the components. Similarly, benefits include total life cycle costs saved. Benefits also include additional profits resulting from earlier completion of the product.

In many organizations, a software developer may have both consumer and producer roles. This distinction between producers and consumers is important because the costs and benefits of each may differ. Thus, it is possible that reuse is economically feasible for the consumer but not for the producer.

Consumers incur the costs of locating, understanding, adapting and integrating reusable components into their software. Benefits can be derived in two ways:

- From the reduction and avoidance of costs that accrue from not having to create and maintain all the equivalent functionality provided by the components and
- From increased profit in completing the product and delivering it to the market earlier.

For producers, the costs consist of start-up and on-going expenses. Start-up costs include the expense of creating a reusable components or making an existing component reusable, standardisation costs, and the cost of setting up a library.

For reuse to be economically feasible, total benefits should exceed total costs:

Producer Costs < Consumer Benefits
In monetary terms,
Cash outflow < Cash inflow

Cash outflows will typically include one-time start-up costs (e.g. library, guideline component creation) and ongoing costs (e.g. library and component maintenance). Cash inflows result from reduced costs, avoided costs and increased profits. Reduced costs result when the cost to develop and maintain software, using reuse is less than that using the current method. Reuse may lead to a shortened time to market and thus, additional revenues and profits that the organisation might not have received otherwise.

VI. ADOPTION OF STANDARDS FOR COMPONENTS

Reusability requires a set of software standards so as to facilitate a better and fast understanding of a component, and faster integration into a system. Different levels of standards for components can be identified.

The cost of standardizing a component must be taken into account when performing a cost-benefit analysis of reuse. To determine whether or not a component is a standard, both interface and functionality should be taken into account. A component without an interface will cost adaptation and integration of the components.

Software standards mainly include standardizing the interface of software components. While the interface for higher-level components is simply a protocol for concepts with a low degree of formalization, it becomes more important as the level of abstraction of the component decreases and the component itself is used mostly.

Software managers hesitate to use software reuse being a cost intensive investment. Therefore, it is necessary to understand the cost impact of software reuse. Other factors that can affect the success of reuse are the design and realization of the components likely to be reused, and particularly their adequate standardization.

VII. SELECTING PILOT PROJECT(S) FOR WIDE DEPLOYMENT OF REUSE

Pilot project(s) are necessary to implement and demonstrate the viability of reuse for its wider deployment [16]. The choice of pilot project(s) is important because it:

- Serves as a test site for proposed reuse practices
- Upon completion, may serve as a showcase for wide deployment of reuse throughout the organisation
- May determine the scope and extent of allocated resources for reuse

The following considerations may be helpful in selecting pilot project(s):

- Identify the success factors for reuse.
- Identify the inhibitors to reuse and the way these can be overcome.
- Determine what makes reuse an appropriate strategy for an organization

VIII. DENTIFYING REUSE METRICS

Appropriate reuse metrics should be developed and identified. Several software metrics have been developed for measuring code reuse and the benefits of reuse. The benefit associated with reuse within a system S can be expressed as a ratio [17] below in equation (2):

$$\mathbf{Rb(S) = [Cno\_reuse - C\ reuse]/Cno\_reuse} \qquad (2)$$

Where
Cno_reuse is the cost of developing S with no reuse,
And Creuse is the cost of developing S with reuse.

It follows that Rb(S) can be expressed as a non-dimensional value in the range

$0 <= Rb(S) <= 1.$

It has been suggested [18] that

1. Rb(S) is affected by the design of a system, and

2. Since Rb(S) is affected by design, Rb(S) may be used to assess design alternatives.

A general measure of reuse in object-oriented systems, termed reuse leverage [19] is defined as:

Rlev = OBJreused/OBJbuilt

Where

OBJ reused is the number of objects reused in a system
OBJbuilt is the number of objects built for a system

It follows (hopefully) that as Rlev increases, Rb also increases. As for code reuse metrics, several attributes can be selected, they are reuse level (RL), reuse frequency (RF), and reuse density (RD) and each of them is defined as follows [17]:

- RL, reuse level, in a repository is the percentage of different items coming from a given source.
- RF, reuse frequency, in a repository is the percentage of reference to items coming from a given source.
- RD, reuse density, in a repository is the normalized number of items coming from a given source.

IX. CONCLUSIONS

Software reuse not only improves productivity but also has a positive impact on the quality and maintainability of software products. This paper represents an attempt to highlight the relevant issues related to software reusability for component based development. Challenges related to reusability issues in CBD have been outlined. Considering the important issues related to software reusability, some high-level reusability guidelines have been suggested, which will further help in enhancing quality and productivity activities within organisations adopting CBD.

X. REFERENCES

[1] Allen Paul (2001): The State of the Practice. In Component Development Strategies Journal, March 2001, Vol. XI, No. 3. pp. 1-16.
[2] Boehm B.W., Pendo M., Pyster A., Stuckle E.D., and William R.D. (1984): An Environment for Improving Software Productivity. In IEEE Computer, June 1984.
[3] Brereton, B. and Budgen, D. (2000): Component-Based Systems: A Classification of Issues. In IEEE Computer, November 2000, pp. 54-62.
[4] Brown Alan (1998): From Component Infrastructure to Component-Based Development. In http://www.sei.cmu.edu/cbs/icse98/ papers/p21.html.
[5] Caldiera G. and Basili V.R. (1991): Identifying and Qualifying Reusable Software Components. In IEEE Computer, February 1991, pp. 61-70.
[6] Councill Bill and Heineman George T. (2000): Component-Based Software Engineering and the Issue of Trust. In Proceedings of International Conference on Software Engineering (ICSE 2000), pp. 661-664.
[7] Itkonen Juha: Measuring Object-Oriented Software Reusability. In http://www.soberit.hut.fi/~tony/seminaari/reports/juho_anttila.doc.
[8] Mao Y., Sahraui H.A., and Lounis H. (1998): ReusabilityHypothesis Verification Using Machine Learning Techniques: ACase Study. In Proceedings of the 13th IEEE International Conference on Automated Software Engineering, 13-16 October, 1998, pp. 84-93.
[9] Paul R.A (1995): Metric-Guided Reuse. In proceedings of 7th International Conference on tools with artificial Intelligence (TAI'95), 5-8 November, 1995, pp. 120-127.
[10] Poulin Jeffrey S. (1994): Measuring Software Reusability. In proceedings of 3rd International Conference on Software Reuse, Brazil, 1-4 November 1994, pp. 126-138.
[11] Sparling Michael (2000): Lessons Learned – Through Six Years of Component-Based Development", Communications of the ACM Journal, Vol. 43 No. 10, October 2000, pp. 47-53.
[12] Tran Vu N. and Liu Dar-Biau. Application of CBSE to Projects with Evolving Requirements – A Lesson-learned. http://www.computer.org/proceedings/apsec/0509/0509toc.htm.
[13] Voas Jeffrey M. (1998): The Challenges of Using COTS Software in Component-Based Development. In IEEE Computer.