Breaking the Boundaries for Software Component Reuse Technology

B.Jalender
Asst.Professor
IT Department
VNRVJIET, Hyderabad
Andhra Pradesh, INDIA.

Dr A.Govardhan
Principal & Professor
JNTUH college of Engineering
JAGTIAL, Karimnagar
Andhra Pradesh, INDIA.

Dr P.Premchand
Professor
CSE Department
UCEOU, Osmania University
Hyderabad, INDIA.

ABSTRACT
Reusable software components are designed to apply the power and benefit of reusable, interchangeable parts from other industries to the field of software construction .Benefits of component reuse such as sharing common code, and components one place and making easier and quicker. The most substantial benefits derive from a product line approach, where a common set of reusable software assets act as a base for subsequent similar products in a given functional domain. Component is fundamental unit of large scale software construction. Every component has an interface and an Implementation. The interface of a component is anything that is visible externally to the component. Everything else belongs to its implementation. This paper addresses the primary boundaries for software component reuse technology.

Keywords
Software Reuse, component, boundaries, interface, product.

1. INTRODUCTION

1.1 Software Reuse
Software reuse is the process of creating software systems from existing software rather than building them from scratch [1]. Software reuse is still an emerging discipline. It appears in many different forms from horizontal reuse and vertical reuse to systematic reuse, and from white-box reuse to black-box reuse. Many different products for reuse range from ideas and algorithms to any documents that are created during the software life cycle [2].Source code is most commonly reused in software systems; thus many people misunderstands software reuse as the reuse of source code alone. Recently source code and design reuse have become popular with (object-oriented) class libraries, application frameworks, and design patterns. Software components provide a vehicle for planned and systematic reuse [2].

1.2 Need to Reuse Software
A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses[2]. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects and ultimately reducing the risk of new projects that are based on repository knowledge [3].

Reuse of Software components concept has been taken from manufacturing industry and civil engineering field. Manufacturing of vehicles from parts and construction of buildings from bricks are the examples [4]. Spare parts of a product should be available in markets to make it successful. The best example in this case is the manufacturers of Honda, Toyota and Suzuki cars would have not been so successful if these companies have not provided spare parts of their cars? Software companies have used the same concept to develop software in parts [4]. These companies have provided plug and play parts with their softwares to market themselves successful. Software parts are shipped with the libraries available with SW. These SW parts are called components. Different people have defined component in different ways. A binary code that can be reused is called a component. A component is an independent part of the system having complete functionalities [4]. In short, software engineering is maturing as an engineering discipline.

Four levels of reuse are proposed:

1. code level components (modules, procedures, subroutines, libraries, etc.)

2. Entire applications

3. Analysis level products

4. Design level products

Code level component reuse occurs most frequently. Standard libraries and popular language extensions are the most obvious examples [5]. However, the level of abstraction is low for these components and that places a limit on the amount of reuse that can be expected. Reusing entire applications, with little to no modification, is great when it can happen, but is just not feasible for many real world problem domains (e.g., real-time software environments) [6].

Perhaps the less represented areas are design products, which allow reuse of similar system implementation strategies, and analysis products, which allow reuse of knowledge about real world domains. Because analysis products allow description and manipulation of real world domains, there are probably the most powerful of reusable products [6].

From the last two decades there have been waves of methods introduced to break both developments cycle time and development labor/cost. Just a few of those include:

- Structured Programming

- Object-oriented programming
- Top-down design methodologies
- Requirements Management
- Improved project management methods
- Unified Modeling Language (UML)

1.3 Types of Software Reuse

1.3.1 Systematic software reuse

Independent of what a component is, Systematic Software Reuse influences almost whole software engineering process. For providing guidance in the creation of high quality software systems at low-cost the software process models were developed. The original models were based on the (mis)conception that systems are built from scratch according to stable requirements.Software process models have been adapted since based on experience and several changes and improvements have been suggested since the classic waterfall model. With increasing reuse of software, new models for software engineering are emerging. New models are based on systematic reuse of well-defined components that have been developed in various projects [4].These trends are particularly evident in markets, such as electronic commerce and data networking, where reducing development cycle time is crucial to business success.

1.3.2 Horizontal reuse

Horizontal reuse refers to software components used across a wide variety of applications. In terms of code assets, this includes the typically envisioned library of components, such as a linked list class, string manipulation routines, or graphical user interface (GUI) functions. Horizontal reuse can also refer to the use of a commercial off-the-shelf (COTS) or third-party application within a larger system, such as an email package or a word processing program. A variety of software libraries and repositories containing this type of code and documentation exist today at various locations on the Internet [7].

1.3.3 Vertical reuse

Vertical reuse, significantly untapped by the software community at large, but potentially very useful, has far reaching implications for current and future software development efforts. The basic idea is the reuse of system functional areas, or domains that can be used by a family of systems with similar functionality [7]. The study and application of this idea has spawned another engineering discipline, called domain engineering. Domain engineering is "a comprehensive, iterative, life-cycle process that an organization uses to pursue strategic business objectives. It increases the productivity of application engineering projects through the standardization of a product family and an associated production process "[7]. Which brings us to application engineering, the domain engineering counterpart: "Application engineering is the means by which a project creates a product to meet a customer's requirements. The form and structure of the application engineering activity are crafted by domain engineering so that each project working in a business area can leverage common knowledge and assets to deliver a high-quality product, tailored to the needs of its customer, with reduced cost and risk" [7]. Domain engineering focuses on the creation and maintenance of reuse repositories of functional areas, while application engineering makes use of those repositories to implement new products.

1.3.4 Horizontal and Vertical Software Assets:

Many systematic software reuse initiatives in organizations fail to take off or have a slow death. There are many factors for this but one key reason is the pursuit of generic technical assets. That is what I refer to as horizontal reuse. Why? Because the focus and intent is to find software assets that are reusable across most or all your applications. This is not only limits the potential for systematic reuse but also makes your reuse initiative extremely risky. Finding assets that are universally reusable is not only difficult but also will make your design overly complex. Overly generic components might also end up creating assets that are:

- hard to test and debug
- difficult to comprehend and maintain
- complex to integrate and configure

2. SOFTWARE COMPONENT

2.1. Describing a Software Component

Component is fundamental unit of large scale software construction. Every component has an interface and an Implementation [8].The following are the unique aspects of a Software Component that must not only be thoroughly described, but also searchable:

- Development
- Scalability
- Testing specification, and performance data
- Known deficiencies
- Version data
- Supportability data
- Evaluations
- Functional specification
- Interface specifications
- Use cases, use scenarios
- OS/Platform compatibility

2.2. Not all software is a component … nor can all software use components

There is a spectrum of software that is already developed, or currently in development that range from "Monolithic" in nature to Plug'n'play [9].

The general assumption is that Monolithic software cannot be adapted to CBD. This is not entirely true as there is a whole movement in software development called "providing a wrapper" around legacy software and then using the component interface to adapt to the framework. Within the spectrum shown above, there is reason to believe that the middle range, which implemented some form of standards-based development, is more likely to have higher value for component mining and retrofit to CBD technology. Clearly, though, the "modern"

technologies like Java have clear advantages in the CBD proposition [10].

2.2.1 The technologies are most suitable for CBD are:

- **COM / DCOM**
- **Java**
- **EJB**
- **CORBA**
- **Active-X**

Developing software with reuse requires planning for reuse, developing for reuse and with reuse, and providing documentation for reuse. The priority of documentation in software projects has traditionally been low [11]. However, proper documentation is a necessity for the systematic reuse of components. If we continue to neglect documentation we will not be able to increase productivity through the reuse of components [12]. Detailed information about components is indispensable. Although the track record for systematic software reuse has been rather spotty historically; several key trends bode well for software reuse in the future:

• Component and framework-based middleware technologies, such as CORBA, J2EE, and .NET, have become main stream.

• An increasing number of developers of projects over the past decade have successfully adopted OO design techniques, such as UML and patterns, and OO programming languages, such as C++, Java, and C#.

2.3 Reusable Software Component

A component is an independent piece of software that interacts with other components in a well-defined manner to accomplish a specific task. Components can be used to build both enterprise critical software and shrink-wrap software [13].

Reusable software refers to software components that can be incorporated into a variety of programs without modification (except possibly parameterization). Reusable software components are designed to apply the power and benefit of reusable, interchangeable parts from other industries to the field of software construction. Other industries have long profited from reusable components [14]. Reusable electronic components are found on circuit boards. A typical part in your car can be replaced by a component made from one of many different competing manufacturers.

A simple example of a reusable software part is Reusable software components can be simple like familiar push buttons, text fields list boxes, scrollbars, dialogs . Software reuse is the use of engineering knowledge or artifacts from existing software components to build a new system. There are many work products that can be reused, for example source code, designs, specifications, architectures and documentation. [3][15]

3. THE BARRIERS FOR COMPONENT BASED DEVELOPMENT

1. Lack of Components - certain discrete areas may enjoy a high level of available components, but these are often the focus of large, well funded projects in specialized areas. A good example is the Common Ada Missile Packages (CAMP) program sponsored by the Department of Defense [16].

2. Poor Cataloging, Distribution and Access Methods - reuse repositories appear to have holdings in the range of under thousand. Even if standard classification schemes could be agreed upon, the burden of component discovery is largely the responsibility of the user. In this manner, the cataloging and distribution of reusable software lags behind other areas of information transfer. For example, there is a well known resource for obtaining technical aerospace literature [17], but no such resource for obtaining software.

3. Data Rights and Copyright - Intellectual property is a difficult concept in the computer and information community. Who own works derived from reused components? How does one protect intellectual property when others reuse it? Copyright and patent law is a complex and dynamic area. The newspaper is the best source for current developments, but a good discussion of the background can be found in [18].

4. Liability - Somewhat related to Data Rights and Copyrights, reuse is slowed by producers not wishing to be liable for their components' performance in some other system, and by the consumers of reusable components not wishing to risk their system on potentially unknown software.

5. Component Qualification - A "rating" system for code reuse would make people more comfortable reusing software. However, this would be a large task and only defers the suspicion and questions to another level: who assigned this rating? Who authorized this rating system? Such systems are generally only encountered within the limited scope of an individual company [14] [19].

6. Incentive and Management Support - Adopting a strong reuse program will certainly help organizations long term competitiveness; the initial investment required can hurt its short term competitiveness. In an attempt to find a "shortcut", there is a temptation to make reuse a process to be applied after development is completed instead of infused throughout the entire development lifecycle [6] [13].

7. Education and "Ego" - Software reuse is not taught in academic training. In fact, Computer Science departments make it quite clear to students that "reuse" and "cheating" are the same thing. Yet, upon arrival at the job, the same people are expected to undergo a radical transformation and understand how to program in teams. This initial educational bias often manifests itself later in programmers' careers as the suspicion of "if I reuse it, others will think I'm not smart enough to write it myself."[6].

3.1 How to break the Barriers for component based development??

The following are the unique approaches for breaking the barriers for Software Component Reuse Technology.

(1) Technology to overcome other potential barriers to component-based software

- commercial security
- location and use of relevant components
- component usability
- assurance of component reliability

(2) Initial benefits of reduced costs, increased reliability and interoperation, or promises of horizontal markets may be inadequate to overcome 'first user' risk.

(3) The greatest benefits require a sustained marketplace in specialized components supporting a change to predictable costs and schedules from component assembly.

(4) Value in legacy software systems will not be easily abandoned.

(5) Pricing and revenue collection schemes may be inadequate.

Component-based software is a new software technique that decreases the production costs of developing new software products and other products that use software as an input [20].

4. RELATIVE ECONOMIC ADVANTAGE OF CBD

Let's start by outlining the potential benefits and risks of Component Based Development from the business perspective.

4.1 Potential business benefits of CBD

- Shorter development cycles
- More maintainable product
- Smaller development staff
- Reduced development costs per project
- Reduced lifecycle costs
- More-easily assessable trade-off evaluations
- More flexibility overall

4.2 Potential business risks of CBD

- Risks of wrong component selection
- Worry about ongoing support
- loss of development control
- Tradeoffs required adapting certain components
- Worries about component quality
- Uncertain internal costing to compare costs
- Availability of quality, high-value components

With those benefits and risks stated, there are still several other challenges to creating the compelling business proposition for CBD which fall into two groups: Economic measure and risk assessment. These are distinctly different facets of risk and must be assessed separately. In the interest of fairness, it is worth noting that there are some associated business costs related to CBD that cannot be termed as risks persespective [6][21]. Those include the time and expense to acquire new organizational skills including training and instilling practice related to software component architectures, component integration skills, and tolerance oriented software design to improve the reusability of software components[9][21].

5. CONCLUSION AND FUTURE WORK

There is little doubt that CBD has a long way to go to move from early adopter to the mainstream. It may be the most unique challenge ever faced by the software industry because of its need to traverse literally every form of software technology in order to be successful. There has never been an attempt to achieve such a wide scope of change in the software industry, let alone to do it world-wide. We believe that CBD will ultimately succeed because it is so desperately needed. However, there must be some very strong pushes from some very important corners of the industry. That push will be needed for CBD to succeed in a way that contributes to the better common purpose of improving the software development business proposition overall. Perhaps CBD's ultimate legacy will be that it managed to transcend the technology camps and entrenched interests within our industry a tall order even for such a very compelling technology.

6. REFERENCES

[1] B.Jalender, Dr A.Govardhan, Dr P.Premchand "A Pragmatic Approach To Software Reuse", Journal of Theoretical and Applied Information Technology (JATIT) Vol 14 No 2 pp.87-96. JUNE 2010.

[2] Sametinger, "Software Engineering with Reusable Components", Springer-Verlag, ISBN 3-540-62695-6, 1997

[3] B.Jalender, N.Gowtham, K.Praveenkumar, K.Murahari, K.sampath "Technical Impediments to Software Reuse" International Journal of Engineering Science and Technology (IJEST) , Vol. 2(11),p. 6136-6139.Nov 2010.

[4] M. R. J. Qureshi, S. A. Hussain, A reusable software component-based development process model, Advances in Engineering Software, v.39 n.2, p.88-94, February, 2008

[5] P.Shireesha, S.S.V.N.Sharma,"Building Reusable Software Component For Optimization Check in ABAP Coding" International Journal of Software Engineering & Applications (IJSEA) Vol.1, No.3, July 2010

[6] 'Barriers to Software Reuse and the Projected Impact of World Wide Web on Software Reuse' (1996), Michael L. Nelson.

[7] Department of the Navy. DON "Software Reuse Guide, NAVSO P-5234-2, 1995.

[8] B.Jalender, Reddy, P.N. "Design of Reusable Components using Drag and Drop Mechanism" IEEE Transactions on Information Reuse and Integration. IEEE International Conference IRI Sept. 2006 Pages: 345 – 350.

[9] "Breaking Down the Barriers to Software Component Technology" by Chris Lamela IntellectMarket, Inc

[10] D'Alessandro, M. Iachini, P.L. Martelli, "A The generic reusable component: an approach to reuse hierarchicalOO designs" appears in: software reusability,1993

[11] Hafedh Mili, Fatma Mili, and Ali Mili "Reusing Software: Issues and Research Directions" IEEE Transactions on software engineering, VOL 21, NO. 6, JUNE 1995

[12] Charles W. Krueger Software Reuse "ACM Computing Surveys (CSUR) Volume 24, Issue 2 (June 1992) Pages: 131 - 183.

[13] M. Pat Schuler, "Increasing productivity through Total Reuse Management (TRM)," Proceedings of Technology2001: The Second National Technology Transfer Conference and Exposition, Volume 2, Washington DC, December 1991, pp. 294-300.

[14] Brian W. Holmgren, "Software reusability: A study of why software reuse has not developed into a viable practice in the Department of Defense," Masters Thesis, Air Force Institute of Technology, AFIT/GSM/LSY/90S-16, September 1990.

[15].M. Aoyoma, "New Age of Software Development: How Component-Based Software Engineering Changes the Way of Software Development?," in Proceedings of the 1998 International Workshop on CBSE,.

[16] Constance Palmer, "A CAMP update," AIAA-89-3144, Proceedings of Computers in Aerospace 7, MontereyCA, Oct. 3-5, 1989.

[17] Michael L. Nelson, Gretchen L. Gottlich, David J. Bianco, Sharon S. Paulson, Robert L. Binkley, Yvonne D.Kellogg, Chris J. Beaumont, Robert B. Schmunk, Michael J. Kurtz, Alberto Accomazzi, and Omar Syed, "The NASA Technical Report Server", Internet Research: Electronic Network Applications and Policy, vol. 5, no. 2, September 1995 , pp. 25-36.

[18] Pamela Samuelson, "Is copyright law steering the right course?," IEEE Software, September 1988, pp. 78-86.

[19] Cai, M.R. Lyu, K. Wong, "Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes," in Proceedings of the 7th APSEC, 2000

[20] Jihyun Lee, Jinsam Kim, and Gyu-Sang Shin "Facilitating Reuse of Software Components using Repository Technology" Proceedings of the Tenth Asia-Pacific Software Engineering Conference (APSEC'03).

[21] B.Jalender, Dr A.Govardhan, Dr P.Premchand" Drag and Drop: Influences on the Design of Reusable Software Components" International Journal on Computer Science and Engineering Vol. 02, No. 07, pp. 2386-2393 July 2010.

.