ISSN:0975-9646

Strategies for Designing and Building Reusable Software Components

Sampath Korra[#1], Dr S.Viswanadha Raju[*2], Dr A.Vinaya Babu [#3]

[#1]*Dept. of CSE, JNTUK, AP, India*
[#2]*JNTUCEJ Jagtial, Karimnagar, AP, India*
[#3]*JNTUH Hyderabad, India*

Abstract:**This paper presents time proven methods and strategies for creating, managing, and accessing a library of reusable software components and also software engineering strategies for designing and building reusable components with proper planning and execution, these methodologies will bring significant cost saving. In addition, cost-benefit guidelines are developed to help an organization decide when the benefits involved in implementing reusable coding procedures outweigh the implementation overhead. Specific recommendations are made for code documentation practices, software design, and management procedures that encourage and result in successful code reuse practices. This paper will address how to deal with specific application data dependencies and software engineering components for portability across different data models.**

Keywords: reuse, component, quality, design requirements, repository.

1. INTRODUCTION:

A component is the fundamental user interface object in Java. Everything you see on the display in a Java application is a component. To be used, a component usually must be placed in a container. Container objects group components, arrange them for display using a layout manager, and associate them with a particular display device. The **Component Based Development (CBD)** approach brings high component reusability and easy maintainability, and reduces time-to-market. Therefore it improves productivity of software systems and lower development cost in the context of reusable software components. **Component-Based Development (CBD**) approach develops software systems by assembling preexisting components under well-defined architecture or framework [1]. The CBD approach brings high component reusability and **easy maintainability**, and reduces **time-to-market**. Therefore it improves productivity of software systems and lower development cost [2].

What is Software Component Reuse?

Software component reuse is the software engineering practice of creating new software applications from existing components, rather than designing and building them from scratch. Reusable components can be requirements specifications, design documents, source code, user interfaces, user documentation, or any other items associated with software. All products resulting from the software development life cycle have the potential for reuse[3].

Advantages of Software Component Reuse

Reusable components are easier to maintain (over time) and typically have a higher quality value (more robust and fewer errors). The business case is reduced application development time, reduced application cost, and improved application quality[4]. software component reuse is one or several of the following:

- To reduce time
- To reduce effort
- To save cost
- To improve quality

2 .OBSTACLES TO SOFTWARE REUSE

Frequent architecture and system changes.
Organizational and cultural issues, reuse requires a fairly drastic change.
Lack of automation tools to assist with specific reuse mechanisms.
Higher up-front investments.
Organization size and amount of application development performed will be proportional to payback. Smaller organizations will not benefit as much from code reuse.
Too large of a project or effort is often attempted initially.
Experience has shown that it requires three to five years to implement a formal reuse program across a large corporation.
Initially, the end-to-end software life cycle will be longer.
Lack of component indexing and searching mechanisms.

Characteristics Needed for Qualification as a Reusable Component

In order for a component (specification, design, code) to be reusable, it needs to have certain qualities that contribute to its reusability [5]. General with build-in adaptability/specialization

Widely applicable
Modular/self-contained
Complete and consistent
Machine independent
Implementation/application independent
Data model independent
Reliable
Robust (good error/exception handling built in)
Understandable/well documented
Adaptable/extensible
Standardized
Portable (across hardware and operating systems)
Certified/testable
Maintainable
Encapsulated (details are isolated and hidden from user)

Guidelines for Creating a Reusable Component

We also provided the software industry with techniques for building reusable components independent of whether or not an object-oriented language is available[6]. The techniques for creating components most applicable to reuse, are as follows:

1. Generalize
2. Standardize
3. Automate
4. Certify
5. Document

Generalization

Generalization is the practice of identifying and designing/building for common uses of a component and removal of special case processing or differences required by various uses of the component. When reused, the generalized component offers a common service to many uses. The reuser of the component is responsible for adding any custom services or specializations back into the component for the particular use[7]. Techniques identified to assist in generalization of a component include the following:

Use parameters or parameter lists for invoking the component. The specializations will provide unique parameter values to determine or cause different behaviours.

The component could be built in two different languages. The use, function signature, behaviour, etc., may be exactly the same for both implementations, but the internal implementation of the component may be entirely different depending on the language used [8].

Separate the behaviour of the component from the application-specific use, business rules, logic, and procedures. Represent the component at a higher level of abstraction. Develop a clear description of the component without using application-specific details. Practice information hiding and encapsulation [9].

Describe components by requirements that are the same regardless of the use or application and describe them by requirements that will differ from application to application [10].

Design two parts to each component: (1) the fixed part, and (2) the variable part. A reuser of the component should only need to change the variable part. The more implementation that can be placed in the fixed part, the more reusable the component becomes.

Standardization

Standardization is the practice of developing and following a uniform approach to defining and building each component. GUI interfaces, help systems, coding styles, uniform structured programming techniques, naming conventions for variables and functions, avoiding use of global variables, decomposition into modules that are completely independent, information hiding concepts, and module cohesion need to be defined and enforced. The higher the level of standardization, the easier it will be for component developers and users to find and assemble applicable components for system solutions. A typical standardization checklist is presented below that has been derived from several of the referenced authors, which may be utilized in a formal technical review of an application development assignment.[11]

Avoid literal constants, excessive levels of inheritance, and excessive use of complex logical constructs. Maximize cohesion; all operations in a component are closely related.

Minimize coupling; reduce connectivity and dependencies on other components (e.g., avoid use of global variables). Proper use of information hiding - If developed correctly, another developer should not need to know what happens inside the component. For example, inputs, outputs, the behaviour of the component, and results are all the next software engineer should ever need to know about the component in a reuse scenario [12].

Automation of Components

Automation refers to having a CASE tool generate some piece of an application, such as a design document or actual code. If the developer can plug in some unique characteristics of a particular requirement, such as a data model, and have another tool automatically generate a component, there are large opportunities for reuse savings.

The software component is the better one of the design decisions in component engineering is how much functionality will be exactly the same in all use scenarios. If a large chunk of functionality can be placed in a single component, then reuse cost savings are maximized. The tendency when designing for reuse is to break the system down to very small, very simplified tasks and assign components to each. If it can be discovered that for every implementation, a large group of tasks can be combined into a single component that will service each implementation equally without specialization, then this represents a tremendous reuse advantage [13].

Certification of Quality

Reuse requires some blind faith on the part of the reuser that the component being used will be as suitable and reliable as documented [14]. It would not take more than a couple of bad experiences in using other components for an entire software team to lose faith in the reusability of a library of components and reuse in general. In order for a reuse initiative to succeed, it is critically important for an independent quality certification process. The following checklist of certification items are recommended for each module inserted into a library

- Testing
- Inspection
- Complexity Measurement
- Automated syntax check
- Standards compliance
- Performs according to design and documentation
- Component has no external data dependencies
- Platform portability (e.g., test an AML component on both NT and UNIX platforms)
- Is the component actually used in multiple systems
- Reuse statistics
- Internal code review/walk-through
- Unit, as well as integration, testing has been performed

In the simplest applications, each component corresponds to an HTML page, and no two applications share components [15]. However, one of the strengths of the WebObjects architecture is its support of reusable components: components that, once defined, can be used within multiple applications, multiple pages of the same application, or even multiple sections of the same page[16].

This section describes reusable components and shows you how to take advantage of them in your applications. It begins by illustrating the benefits of reusable components. It then describes how to design components for reuse, how reusable components can communicate with the parent component, and how state is synchronized between parent and child components. Finally, it provides some design tips for you to consider when designing your own reusable components [17].

Creating Reuse Documentation

The importance of documentation in reuse is critical. A potential reuser needs accurate information about a component in order to align the component with a requirement [18]. A quick checklist of what should be documented is presented below:

- Name of the component
- Classification/categorization of the component
- Interface requirements
- Description of what the component does
- Description of the components properties
- Reuse specific information (e.g., history of reuse, limitations and conditions for use, how it should be used, systems it is implemented in)
- Specification of the component
- Quality/certification
- Author and creation data
- Tests developed for the component (e.g., test plan, use cases, data, results)
- Who maintains the component
- Recommendations for improvements
- Links to requirement specifications
- Version and language implementation data
- Relationships to other components

Strategies for Designing and Building Reusable Components

Listed below are several practical guidelines and advice to assist developers in the creation of new reusable components:

- Collaborate with multiple software engineers throughout the design and definition phases.
- Install a reuse analysis stage into all detailed design work, so that the opportunities for reuse can be effectively assessed.
- Strive for a clearly defined, single purpose per component.
- Document all component interface requirements, also known as parameter lists or function signatures, in the design phase. Do not attempt to start at the coding phase and "design as you go." Remember that component reuse requires much more up-front planning. More design time and less construction time should be expected and planned for by project managers.
- With the inputs, results, or outputs and a description of a component, another software developer should be able to use that component without ever knowing what the code does inside the component. The component should pass this "self test."
- Strive for loosely coupled and highly cohesive components.
- Develop components with overall future use in mind, not just a single project.
- Always put extra effort into error handling and making components robust.
- Develop a certification program and communicate the certification criteria to all component developers.
- Use consistent design styles.

Component Repositories

A reuse library or component repository organizes, stores, and manages reusable components, often supporting multiple versions of the components .The primary requirement is assigning and dedicating a staff resource to tend to and manage the repository[19,20]. A secondary requirement is allowance for developer access to the library in a manner that makes searching for and finding components an easy task. McClure recommends that several tasks be performed in the creation of a repository, including most of the following [21]:

- Define the types of components for storage in the repository.
- Define the organization structure for the reuse repository.

Select a good configuration management tool [e.g., Clear Case (excellent commercial system) or Concurrent Versioning System (excellent shareware)]. Implement as much automation as the market will provide, such as cataloging tools and repository browsers. (This is one area of software engineering tools that is somewhat limited now, but will experience explosive growth this year)[22].

Define a classification scheme for indexing each component. The scheme should allow for change and growth over time. Examples of indexing fields and a list of contents or values that might be used for each are as follows [23]:

Classification, cataloging, and certification needs to be performed for every component placed into the repository. A defined growth path and plan should be adopted for needed components. As more and more application development is performed on a given architecture, the projects should be burdened with populating the repository where and when needed. A corporate repository should be treated as any other valuable asset in the corporation. It will need to be managed, budgeted for, and upgraded, and its status and contents need to be communicated often to users of the asset. Implement a repository browser and search engine. Development and implementation of Web-based tools may be the best approach for browsing and searching [24].

3. STRATEGIES FOR IMPLEMENTING SOFTWARE REUSE PRACTICES IN AN ORGANIZATION

Consider size of organization.
Consider investment required.
Acquire tools required to support the transition.
Plan, implement, and analyse incremental efforts to adopt reuse.
Implement a reuse incentive system. Incentives may take the form of monetary rewards or recognition and reward programs. The incentive should draw positive attention to reuse and promote its implementation.
Develop reuse metrics to determine where you are in implementing your plan and to give yourself a way to determine when changes are required to your strategy.
Treat technology transition as a project: manage it, resource it, schedule it, and measure progress.
Select and conduct a small pilot reuse project.
Implementation of change.
Set expectations, conduct training, and keep everyone up to date on what goes into a reuse repository.
Communicate the results of your initiatives to your internal development and management teams, as well as to your clients or customers, letting them know how you are improving the value of your products.

What are the Costs Associated with Software Reuse Practices?

Implementing and following reuse practices does not come without a price tag. Sometimes substantially more effort is required to build for reuse[25]. For example, to create a generic reusable form of a program module can take twice as long as creating it for one time use...and to create a general object class may require 10 times the effort to create the class for use in one application. Additional design and planning efforts will be required initially [26].

What are the Most Common Organizational Failures when Transitioning to Reuse?

- Rapidly changing underlying technology
- Expectation of an early high payoff
- Not selecting a narrow domain as a starting point
- Focusing on developing components for deliverables of a single project, instead of multiple projects

What are the Most Common Technical Failures when Transitioning to Reuse?

Inadequate configuration and version management control mechanisms

- Inadequate searching/browsing/look-up mechanisms
- Too little control over what is put in a library
- Undocumented interfaces and/or components
- No facility for exceptions; all or nothing reuse
- All requirements cannot be satisfied all of the time
- Neither top-down nor bottom-up design is adequate to capture the benefits of reuse

4. CONCLUSION AND FUTURE WORK

As above mentioned, CBSE can be a fundamental technology for software development so that it requires to re-think various aspects of software development. Besides technical issues, non technical issues such as commerce of components and management issues are also important [27]. As the software component vendors have been growing,
a software component market is emerging. Since software can be distributed over the Internet, web-based software component brokers have emerged.

1) CBSE requires a new process that is component acquisition
2) The workload for testing is drastically reduced.

Besides these cases, a number of component-based software developments have been conducted. Current component technologies have been used to implement different software systems, such as object oriented distributed component software and Web based enterprise application [28].

REFERENCES

[1]Coleman, Derek, Arnold, Patrick, Bodoff, Stephanie, Dollin, Chris, Gilchrist, Helena, Hayes, Fiona, and Jeremaed, Paul, *Object-Oriented Development: The Fusion Method*, Prentice Hall, Englewood Cliffs, NJ, 1994

[2]Coulange, Bernard, *Software Reuse*, Springer -Verlag, London Limited 1998

[3]Jacobson, Ivar, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, Reading, MA, 1992

[4]McClure, Carma, *Software Reuse Techniques, Adding Reuse to the Systems Development Process*, Prentice Hall, Upper Saddle River, NJ, 1997.

[5]Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[6]Reifer, Donald J., *Practical Software Reuse*, John Wiley & Sons Inc., New York, New York, 1997

[7]Sametinger, Dr. Johannes, *Software Engineering with Reusable Components*, Springer - Verlag, Heidelberg, Berlin, 1997

Center for Computer Systems Engineering Information Clearinghouse (CFCSE-IC)

[8] 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).

[9] 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,*.

[10] B.Jalender, Dr A.Govardhan and Dr P.Premchand. Article: Breaking the Boundaries for Software Component Reuse Technology. International Journal of Computer Applications 13(6):37–41, January 2011. Published by Foundation of Computer Science.

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

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

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

[14] B.Jalender, Dr A.Govardhan, Dr P.Premchand "A Pragmatic Approach To Software Reuse", 3 vol 14 No 2 Journal of Theoretical and Applied Information Technology (JATIT) JUNE 2010 pp 87-96.

[15] Article "Considerations to Take When Writing Reusable Software Components"

[16]. R.G. Lanergan and C.A. Grasso, "Software Engineering with Reusable Designs and Code," IEEE Transactions on Software Engineering, vol. SE-10, no. 5, September 1984, pp. 498-501

[17] T.J. Biggerstaff and A.J. Perlis, eds.," Software Reusability: Concepts and Models" ACM Press, New York, vol. 1, 1989.

[18] B.Jalender, Dr A.Govardhan, Dr P.Premchand, Dr C.Kiranmai, G.Suresh Reddy" 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.

[19] 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.

[20] W.A. Hegazy, The Requirements of Testing a Class of Reusable Software Modules, Ph.D. dissertation,Department of Computer and Information Science, The Ohio State University, Columbus, OH, June 1989.

[21] 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.

[22] B.H. Liskov and S.N. Zilles, "Specification Techniques for Data Abstractions," IEEE Transactions on Software Engineering, vol. SE-1, no. 1, March 1975, pp. 7-19.

[23] Douglas Eugene Harms "The Influence of Software Reuse on Programming Language Design" The Ohio State University 1990.

[24] Article "assess reuse risks and costs "www.goldpractice.thedacs.com/practices/arrc/".

[25] 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.

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

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

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