DESIGNING CODE LEVEL REUSABLE SOFTWARE COMPONENTS

B.JALENDER [1], Dr A.GOVARDHAN [2], Dr P.PREMCHAND [3]

[1] Asst Professor, Department of IT, VNRVJIET, Hyderabad, India-500090
[2] Professor in CSE & Director of Evolution, JNTUH, Hyderabad,.
[3] Professor ,CSE Department,UCEOU, Osmania University, Hyderabad.

ABSTRACT:

The basic idea behind building Reusable software components is to design interchangeable parts from other industries to the software field of construction. A reuse library or component reuse repository organizes stores and manages reusable components. The biggest advantage of the building reusable software components is that it reduces the time and energy in developing any software. Frameworks provides a standard working system through which user 's main focus is on developing desired modules instead of developing lower level details. By using this facility the software developers can spend more time in developing the requirement of software, rather than preparing the tools of application development. Framework is set of reusable software program that forms the basis for an application. Frameworks help the programmers to build the application quickly .At its best code reuse is accomplished through the sharing of common classes and/or collections of functions, frameworks and procedures. This paper describes how to build the code level reusable components and how to design code level components. Finally providing coding guidelines, standards and best practices used for creating reusable code level components and guidelines and best practices for making configurable and easy to use.

KEYWORDS:

Reuse, code, component, barries, software, framework.

1. INTRODUCTION

1.1 What Is A "Reusable Software Component?"

Building Reusable software components is latest trend in the field of software construction. Reusable electronic components are found on the bread/circuit boards. All the typical part in your car can be replaced by a component made from one of many different competing manufacturers. Example your maruthi car type can be a J.K.tyre can be replaced by Bridge Stone company tyres. The idea is that standard interfaces allow for interchangeable, reusable components [2]. This definition of reuse does not meet our definition because it is not concerned with reusable software components incorporated into client programs.

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 . every thing visible in Java interface are reusable components .Software reuse is the use of engineering knowledge or artifacts from existing software components to build a new system [1][2]. There are many work products that can be reused, for example source code, designs, specifications, architectures and documentation [2].

1.2 Why Software Reuse?

In software engineering reuse is an important area where we can improve the productivity and quality of software [3].Software reuse is the use of existing software or software knowledge to construct new software [4]. A component is a object in the graphical representation of application and that can interact with user.The user must access these components accurately and quickly, and be able to modify them if necessary. Reusable software components are designed to apply the power and benefit of reusable, interchangeable parts from other industries to the field of software construction [5].Software reuse provides a basis for dramatic improvements in increased quality and reliability and in long-term decreased costs for software development and maintenance.

Reuse of SW 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. 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 [6][7]. 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 [7]. A component is an independent part of the system having complete functionalities.

1.3 Why a component can be reused.

There are so many ways to reuse the component in software. But the three major ways to reuse software, the first way is you can use the component in its original from in multiple systems, the second ways is you can extend component functionality as needed for individual systems, and the last way is you can restrict component functionality as needed for independent systems. In brief the first way involves the technique for writing reusable software components and identifying those components, the second way involves the covering of the steps required for extending reusable software components; finally the last way addresses testing and deploying your extensions and wrappers for reusable software components[8][9].

When using a component for reuse that must be meet requirements. We need to remember that to inheritance means to derive a new component from original component to extend the required functionality [10]. All most all the reusable software components comes in the one of the above three forms. In the first way code samples are copied and pasted among systems. In the second way you have a string parsing routine that your coworkers find useful. You email that code to them and they perhaps embed or modify it to a new method. Recipes are an extension of code samples by which a way to reproduce some behavior is described in terms of consuming an existing component [11]. In the last way you can reuse the binaries distributed on local or remote systems without distributing them with each product [12].

1.4 Approaches Supporting Software Reuse

- Application Frameworks
- Application product lines

- Aspect-oriented software development
- Component-based Development
- Configurable vertical applications
- COTS (Commercial-O-The-Shelf) integration
- Design Patterns
- Legacy system wrapping
- Program generators
- Program libraries
- Service-oriented systems

- **Application Frameworks**: Collections of concrete and abstract classes that can be adapted and extended to create application systems. It is used to implement the standard structure of an for a specific development environment. A framework is a incomplete implementation plus conceptually complete design. Application frameworks became popular with the rise of, since these tended to promote a standard structure for applications.

- **Application Product Lines**: Application product lines, or Application development, refers to methods, tools and techniques for creating a collection of similar product line systems from a shared set of software assets using a common . An application type is generalized around a common architecture so that it can be adapted in different ways for different customers. A type of application system reuse. Adaptation may involve component and system configuration; selecting from a library of existing components ;adding new components to the system; or modifying components to meet new requirements [14].

- **Aspect-Oriented Software Development**: Aspect-oriented software development (AOSD) is an emerging software development technology that seeks new modularizations of software systems in order to isolate secondary or supporting functions from the main program's business logic. AOSD allows multiple concerns to be expressed separately and automatically unified into working systems [15].

- **Component-Based Development**: Systems are developed by integrating components (collections of classes) that conform to component-model standards. By adopting a component based development approach you will have the option of buying off-the-shelf components from third parties rather than developing the same functionality inhouse[16].

- **Configurable Vertical Applications**: Configurable vertical application is a generic system that is designed so that it can be configured to the needs of specific system customers[17]. An example of a vertical application is software that helps doctors manage patient records, insurance billing, etc

- **COTS Integration:** By integrating existing application systems System is developed. A type of application system reuse. A commercial off–the-shelf (COTS) item is one that is sold, leased, or licensed to the general public.[18]

- **Design Patterns**: A design pattern is a recurring solution for repeatable problem in software design. Design Pattern is a template for how to solve a problem that can be used in many different situations [19].

- **Legacy System Wrapping**: By wrapping a set of defining interfaces by legacy systems provides access to interfaces. By rewriting a legacy system from scratch can create a equivalent functionality information system based on modern software techniques and hardware [20].

- **Program Generators**: Program Generator is a program that enables an individual to easily create a program of their own with less effort and programming knowledge. With a program generator a user may only be required to specify the steps or rules required for his or her program and not need to write any code or very little code A generator system embeds knowledge of a particular type of application and can generate systems or system fragments in that domain. Program Generators Involves the reuse of standard patterns and algorithms [20].

- **Program Libraries**: Function and class libraries implementing commonly used abstractions are available for reuse. Libraries contain data and code that provides necessary services to independent programs. This idea encourages the exchanging and sharing of data and code .

- **Service-Oriented Systems**: SOA is a set of methodologies and principles for developing and designing software in the form of component. These components are developed by linking shared services that may be externally provided. An enterprise system often has applications and a stack of infrastructure including databases, operating systems, and networks [21].

1.5. Levels of Reuse

Reuse is divided into following four levels

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

2. Entire applications

3. Analysis level products

4. Design level products

The most frequently used component reuse is code level. Examples for code level component reuse are standard libraries and popular language extensions are the most obvious examples. In this case the level of abstraction is low for these components and the expected amount of reuse is low. For many real world problem domain reusing entire applications with little or no modification will give a high reuse when compared to code level component reuse. Using entire application means using commercial-off-the-shelf packages (COTS) or minimal adaptation of a specialized product applied to a new customer (i.e, Ford Motor Co. using NASTRAN, a NASA developed product). This paper describes about how to design code level reusable components [22].

1.6 Quantitative benefits achieved through software reuse

- **Nippon Electric Company** Achieved 6.7 times higher productivity and 2.8 times better quality through 17% reuse. Nippon Electric company improved software quality 5-10 times over a seven year period through the use of unmodified reuse components and achieved a better quality in the domain of basic system software development and in the domain of communication switching systems.

- **GTE Corporation** with reuse levels of 14% GTE Corporation Saved $14 million in costs of software development. GTE Data Services benefited from $1.5M in cost savings in 1988 for 20-50% reuse.

- **Toshiba** saw a 20-30% reduction in defects per line of code with reuse levels of 60%

- **DEC reported** cycle times that were reduced by a factor of 3-5 through reuse levels of 50-80% and an increase of 25% in productivity through software reuse

- **Hewlett-Packard (HP)** cited quality improvement on two projects of 76% and 24% defect reduction, 50% and 40% increases in productivity, and a 43% reduction in time to market with reuse levels up to 70%. ROI ranged from215% for one development to 410% for the other

- **Raytheon** achieved a 50% productivity increase in the MIS domain from 60% reuse using COBOL

- **A study of nine companies** showed reuse led to 84% lower project costs, cycle time reduction of 70%, and reduced defects

- **312 projects in aerospace industry**
  Average 20% increase in productivity; 20% reduction in customer complaints; 25% reduction in time to repair; 25% reduction in time to produce the system

- **Japanese industry study** 15-50% increase in productivity; 20-35% reduction in customer complaints; 20% reduction in training costs;10-50% reduction in time to produce the system

- **Simulator system developed for the US Navy**
  Increase of approximately 200% in number of SLOC produced per hour

- **NASA Report**
  Reduction of 75% in overall development effort and cost

- **AT&T** reported a 50% decrease in time-to-market for 40-90% reuse

- **Raytheon Missile Systems** experienced a 1.5 times increase in productivity from 40-60% reuse
- **SofTech** had a 10-to-20 times increase in productivity for reuse greater than 75%

2. CODE LEVEL COMPONENT REUSE

There are several technical issues that currently keep reusable software from becoming a reality. One of the techniques is designing code level reusable components. In this approach the technical issue is the lack of formal specifications for components. A programmer cannot be expected to reuse an existing part unless its functionality is crystal-clear. A component will only be reused if its behavior is completely and unambiguously specified in a form understandable by potential programmers. These specifications should be mathematically rigorous. Specifically, informal natural language descriptions are not sufficient. Summary of code level component reuse is shown in the following table 1.

A component is evaluated across a number of topic levels, each of the level which provides guidance about what one can expect at each reuse level. The topic levels currently defined are:

Level 1: Documentation

Level 2: Extensibility

Level 3: Intellectual Property Issues

Level 4: Modularity

Level 5: Packaging

Level 6: Portability

Level 7: Standards compliance

Level 8: Support

Level 9: Verification and Testing

Level	Summary
Level 1	Limited reusability; the software is not recommended for reuse.
Level 2	Initial reusability; software reuse is not practical.
Level 3	Basic reusability; the software might be reusable by skilled users at substantial effort, cost and risk.
Level 4	Reuse is possible; the software might be reused by most users with some effort, cost, and risk.
Level 5	Reuse is practical; the software could be reused by most users with reasonable cost and risk.
Level 6	Software is reusable; the software can be reused by most users, although there may be some cost and risk.
Level 7	Software is highly reusable; the software can be reused by most users with minimum cost and risk.
Level 8	Demonstrated local reusability; the software has been reused by multiple users.

Level 9	Proven extensive reusability; the software is being reused by many classes of users over a wide range of systems.

Table 1.Code Reuse Level Summaries

3. HOW TO BUILD CODE LEVEL REUSABLE COMPONENTS

A code level reusable software component is self-contained and has clearly defined boundaries with respect to what it does and does not do. At these boundaries it will present an equally and clearly defined set of interface points that will allow easy integration with the other components. For most of the users, the interface will be sufficient to allow reuse the code level components; that is, the implementation will be hidden through encapsulation .For those users who need to modify the internals/functionality of the component in some way, for example to add a feature, or fix a previously undiscovered defect, a clear, unambiguous, and understandable specification for the component will be required. The component will then conform to the specification and user-reproducible tests will validate this conformance. This allows users to modify implementation details, assuming source code is available and to build code level reusable components [22].

We need to provide clear documentation when distributing a code level software component. That will provide the information about how to reuse it along with example applications and installation guides.. Finally, it is critical that the component is correctly licensed and full details are made available to the end user [23]

The following ways to build code level reusable components

- **Class libraries**
- **Function libraries**
- **Design patterns**
- **Framework Classes**

Class libraries

Class libraries are the object-oriented version of function libraries. Classes provide better abstraction mechanisms, better ability and adaptability than functions do. Reusability has greatly from concepts like inheritance, polymorphism and dynamic binding. In many class libraries there are classes devoted to generic data structures like lists, trees and queues. The major problem with class libraries is that they consist of families of related components. Thus members of families have incompatible interfaces. Often several families implement the same basic abstraction but have interfaces. This makes libraries hard to use and makes interchanging components. Also, most class libraries are not scalable [24].

Function libraries

Functions are the most common form of reusable components. For many programming languages, standard libraries have been, for example, for input/output or mathematical functions. A few decades ago languages had much functionality in the language itself, e.g., PL/I. Later on, the trend was towards lean languages with standard libraries for various functionalities, e.g., Modula-2. There are many example of function libraries, from collections of standard routines (e.g., the C standard libraries) to domain libraries (e.g., for statistics or numerical purposes) [25].

Design patterns

The purpose of design patterns is to capture software design know-how and make it reusable. To save time and effort, it would be ideal if there was a repository which captured such common problem domains and proven solutions. In the simplest term, such a common solution is a design pattern. Design patterns can improve the structure of software, simplify maintenance, and help avoid architectural drift. Design patterns also improve communication among software developers and empower less experienced personnel to produce high-quality designs. you design and build different applications, you continually come across the same or very similar problem domains. This leads you to find a new solution for the similar problem each time. They standardize piecework to larger units. For example, many times there exists a special arrangement of classes and/or objects in order to avoid reuse errors. A subsystem is a set of classes with high cohesion among themselves and low coupling to classes outside the subsystem [26].

A software design pattern describes a family of solutions to a software design problem.By using available methods, functions, threads we can build the code level reusable components. Example design patterns are Model/View/Controller (MVC), Blackboard, Client/Server, and Process Control. Design patterns can correspond to subsystems, but often they have level of granularity. Design patterns have been to avoid dependence on classes when creating objects, on particular operations, representation or implementation, on particular algorithms, and on inheritance as the extension mechanism [27]. MVC is enforces the separation between the input, processing, and output of an application. To this end, an application is divided into three core components: the model, the view, and the controller. Each of these components handles a different set of tasks.. The architecture of MVC shown in below figure 1.

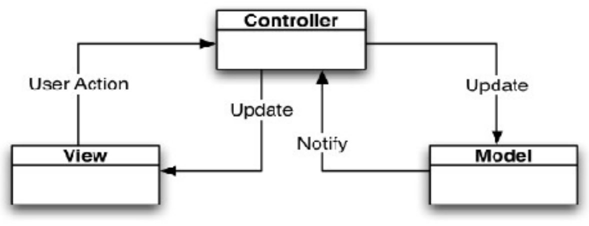

Figure 1.MVC architecture

Framework Classes

For large-scale reuse, isolated classes are small-scale primitives that are to boost productivity; systems have to be built out of largescale composites. Thus we have to focus on sets of classes that collaborate to carry out a common set of responsibilities, rather than on individual classes. Frameworks are flexible collections of abstract and concrete classes designed to be extended and for reuse. Components of class libraries can serve as discrete, stand-alone, context-independent parts of a solution to a large range of applications, e.g., collection classes. Components of frameworks are not intended to work alone; their correct operation requires the presence of and collaboration with other members of the framework components. Reusers of framework classes inherit the overall design of an application made by experienced software engineers and can concentrate on the application's functionality [28].

The major advantage of framework classes over library classes is that frameworks are concerned with conventions of communication between the components [29] .Today the combination of components from class libraries is the exception rather than the rule. This is because there is some implicit understanding of how components work together. High cohesion and low coupling increase the reusability of components. But unless the component does have extensive functionality, it is required to cooperate and communicate with many others. In a framework this interaction is built in and eases interaction of its components [30].

MacApp is a framework for Macintosh applications. An abstract MacApp application consists of one or more windows, one or more documents, and an application object. A window contains a set of views, each of which displays part of the state of a document. MacApp also contains commands, which automate the undo/redo mechanism, and printer handlers, which provide device independent printing. Most document classes do little besides define their window and how to read and write documents to disk. An average programmer rarely makes new window classes, but usually has to define a view class that renders an image of a document. MacApp makes it much easier to write interactive programs [31].

Framework is set of reusable software program that forms the basis for an application. Building Reusable Frameworks help the developers to build the application quickly. These are useful when reusing more than just code level component[32] .Frameworks are having well written class libraries. By reusing these class libraries we will build the code level reusable software components[33].

4. CONCLUSION AND FUTURE WORK

Software reuse can save time, save money, and increase the reliability of resulting products. However, an attempt to reuse software that is not easily reusable can have the reverse effect. The biggest advantage of the software framework is that it reduces the time and energy in developing any software. Building code level reusable components will increase the quality and reduces time to design.

There are a number of methods for designing software components for reuse, but these methods tend to focus on. One of the best method is to develop code level reusable componets.These will give the best code reuse and improves the quality of the product. At its best code reuse is accomplished through the sharing of common classes and/or collections of functions, frameworks and procedures. At its worst code reuse is accomplished by copying and then modifying existing code causing a maintenance nightmare. This paper gives the concept of Reuse Code Levels and explores their applicability to reuse the software components. Using frameworks, the developers can devote more time in developing the software requirement, not in preparing the environment and tools of application development.The code level reuse can save the time and money and increase the productivity and quality in the product.

REFERENCES

[1] 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.

[2] 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.

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

[4]. 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

[5] J.M. Boyle and M.N. Muralidharan, "Program Reusability through Program Transformation," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, September 1984, pp. 574-588.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12]Sullivan,K.J.;Knight,J.C.;"Experience assessing an architectural approach to large-scale, systematic reuse," in *Proc. 18th Int'l Conf. Software Engineering*, Berlin, Mar. 1996, pp. 220–229

[13] Schmidt, D. C., *Why Software Reuse has Failed and How to Make it Work for You* [Online], Available: http://www.flashline.com/content/ DCSchmidt/lesson_1.jsp, [Accessed: 18 August 2002].

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

[15] http://www.esdswg.com/softwarereuse/Resources/rrls/RRLs_v1.0.pdf.

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

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

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

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

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

[21] 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.

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

[23] 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.

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

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

[26] 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).

[27] Ralph E. Johnson & Brian Foote *"Designing Reusable Classes" (IEEE Computer Society Press Tutorial)* IEEE Computer Society Press, Los Alamitos, CA May 1991, 299 p.

[28] http://diwt.wordpress.com/tag/struts2/

[29] 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