Developing Reusable .NET Software Components

Muthu Ramachandran
School of Computing, Creative
Technologies and Engineering
Leeds metropolitan University
Leeds, UK
Email: m.ramachandran@leedsmet.ac.uk

Gopal Singh Jamnal
School of Computing, Creative
Technologies and Engineering
Leeds metropolitan University
Leeds, UK
Email: gopal.jamnal@gmail.com

***Abstract* --- Software Development with reuse and for reuse is the foundation of CBSE (Component based software engineering) which allow faster development at lower cost and better usability. A reusable software component works as a plug and play device, which abstract the software complexity and increase performance. Software reuse guidelines have been addressing the issue of capturing best practices. Form a long while, Software industry has collected the enormous wealth of knowledge, experience, domain expertise, design principals & heuristics, hypothesis, algorithms, and experimental results but still there are no rock solid and mature software component development guidelines defined for the current technologies such as .NET.**

This paper presents a guidelines based framework (known as .NET Guider) for guidelines based component development for reuse in .NET family. We have demonstrated our approach by designing a binary component as part of development for reuse based on our own .NET component framework. This paper also provides a number reuse analysis and metrics and a prototype component guider tool which sits on top of the .NET architecture with built-in software development & reuse knowledge.

Keywords -- Software Reuse, Software Guidelines, Software Design Knowledge, CBSE, GSE

I. INTRODUCTION

In last two decade, Software Development has emerged as a global market for business. It is equally contributing in nation's economy as other industries i.e. Production, Mining etc. In current scenario, IT Companies have slowly shifted from traditional software development environment to component based environment. The reason for such change is to maximize reusability and increase profit. Correspondingly, CBSE (Component Based Software Engineering) is the new global market trend for software development industries.

According to (Council and George, 2001) software component is an independent software element which can be deployed and composed in further software development cycle without any modification as a handy component.

Software components abstract the complexity from end users and provide quick and easy implementation, subsequently help to reduce the cost by its reusability. All major software development companies adopted the component based development technique to survive in competitive market. According to Zirpins et al. [8] suggested that, Today's Modern ERP systems are made of several software components and it shows the real example of Software Component reuse on big scale.

In addition [5] stated that, Software components works as an autonomous hardware components, which abstracts the internal complexity of device and provide easy user interface to operate and similarly can be used as a building block in new product development. Among all IT giant, Microsoft was the first one, who understood the industry needs and succeed to cash this ready market. Microsoft offered the Development Kit known as "Microsoft Visual Studio" to develop reusable software component with many development options. Now days all Software Industries have understood that component can deliver great reusability, extensibility and maintainability for creating a large scale software system by breaking down into the small binary components. Ravichandran and Rothenberger [6] noticed that, now industry is emphasizing more on black box reuse over white box reuse. Component based programming emphasize on "Black Box Reuse" which means implementing client of such component, not need to worry about its internal functionality.

The significant contribution of this paper is to develop a generic framework for .NET component model, and to provide comprehensive guidelines for development team to adopt them. The analysis of a .NET based binary tree component compares with efficiency and reusability analysis of the existing example [4] owned component. This paper has also provided reusability analysis of our own binary component against our standard framework and guidelines.

II. COMPONENT BASED SOFTWARE ENGINEERING

In plain English, software component is an independent unit of binary code, which can be used as plug and play, like a hardware device. It designed and developed in modular architecture, which promote interoperability with other component and framework for reuse with reuse.

Although, a software component is an independent modular unit, which is loosely coupled and not bonded to one client and most important, it possesses official usage guidelines for further reuse. In general, a typical software component model is divided into three parts as by [2] are:

a) *Semantics*: Denotes, what component are mean to be?
b) *Syntax*: Defines, how it is constructed, developed and represented?
c) *Compositions:* Finally, how it is going to be composed and reassembled?

Semantics provide description of components and describe the components usage and functionality. Where syntax denotes the component's algorithms and development complexity, which give the physical structure to component. Finally composition provides overall wrapper mechanism to compose the various functionality of a component and present it for reuse to customers.

In some areas, life cycle of component is much similarly as window or web software development lifecycle but modeling, packaging and implementation are totally different from general software development process. An in depth domain analysis and method of packaging and deployment of compiled binary code in such a loose couple way, makes component's lifecycle special.

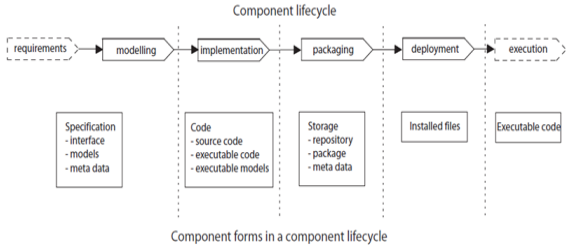

Figure 1. Represent the phases of a software component development in software Development team.

The demands and requirements for software component keep fluctuating according to new system requirements in market, so while developing a new software component for reuse, developers need to follow some guidelines for component design and development, however still software industry don't have rock solid guidelines for building reusable components but seasoned author like Lowy [3] and [5] proposed their own guidelines for component architecture, table1 represent their component guidelines classification.

TABLE 1. SOFTWARE COMPONENT SPECIFICATION

(Lowy, 2010)	*(Ramachandran, 2008)*
1. Separation of interface and implementation.	*1. Language specific.*
2. Binary Compatibility.	*2. Design specific.*
3. Language Independence.	*3. Domain specific.*
4. Location Transparency.	*4. Product specific.*
5. Version Control.	*5.Architecture Specific*
6. Component Based Security	*6. Organizational& managerial*

We can see from the above comparison table, where [3] is more specific about lower level of programming concepts, while [5] is more focused on bigger picture about component design and reuse potential for industries.

Furthermore, software component lies, in two categories as, "for reuse" and "with reuse". Similarly [1], claimed that usage of software component can be divided into two categories:

a) Consumer Reuse: which mean, development of new software systems, using existing component, called as "development with reuse".

b) Producer Reuse: which means, developing, building new portable components, for further reuse, called "Development for Reuse".

The figure 2, presents the differentiation between producer reuse and consumer reuse.

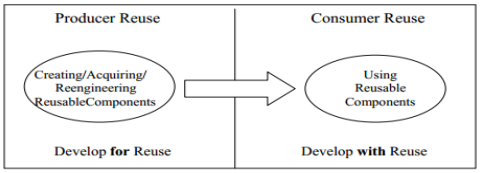

Figure 2. Difference between Producer Reuse and Consumer Reuse.

III. .NET FRAMEWORK REUSABILITY

In 1993, COM was designed and developed to create platform independent, distributed and object oriented based, reusable binary component, along with MTS (Microsoft Transaction server) for distributed transactions. During 2000, at the time of second phase of release, Microsoft combined COM and MTS and introduced it as COM+. This time it is integrated with subscriber/publisher event, Queued Service and late binding features (Microsoft, 2013).

Now a day, .NET framework is the latest invention from Microsoft, which provides latest component technology as WCF for web and assemblies for windows. The .NET framework is made of CLR (Common language runtime) and many class libraries. Also .NET framework provides compatibility with COM object, with the help of RCW (Runtime Callable wrapper). (Ibit)

IV. PROPOSED GUIDELINES FOR DEVELOPING REUSABLE .NET COMPONENTS

The good program design guidelines and systematic approach for computing, has been defined since early 70s and 80s by seasoned authors as Parnas and Dijkstra. However still a big gap exists in current software industry to adopt that silver software engineering principal to handle todays software development needs .The reasons behind are, market changes, customer expectation changes in over a period and it became very difficult to survive in competitive software development world to fulfill growing customers need in timely manner. So as a solution , terminology "Guidelines based Software Engineering"(GSE) is grounded, on the basis of collecting experiences and knowledge from past several years of software development experiences and wealth of knowledge & artifacts and use them as rationale for developing a new customized guidelines to face todays software development challenges [5].

The success of software component is utterly based on Knowledge of domain. Identifying and classifying the generic component form a specific software package or domain area is a human intensive task, which only can be done by experts after several years of experience. Figure 3 presents the relationship between domain and software engineering.

.

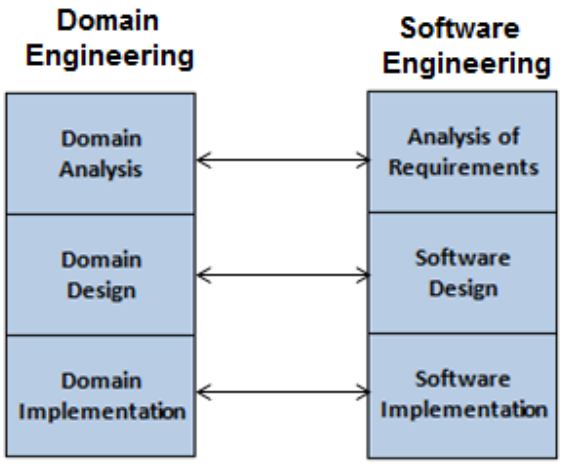

Figure 3. Process and Similarity in Domain Engineering and Software Engineering.

However, some self-assessment questions can be helpful for architects to component design [5] :

- Identifying the common functions in domain to avoid duplications of tasks.
- Dependency on other components and hardware devices.
- Optimized designed for further technology up gradation.
- Easy use and implementation with some minor changes.
- How valid is component decomposition for reuse?

- From business point of view, a good Return on Investment (ROI) of a software component ensures the component's longer life and usage in application domains.

The design for reuse, provides a set of clear implementation steps, which should be followed by architects and developers. This set of guidelines can be classified into a number of categories:

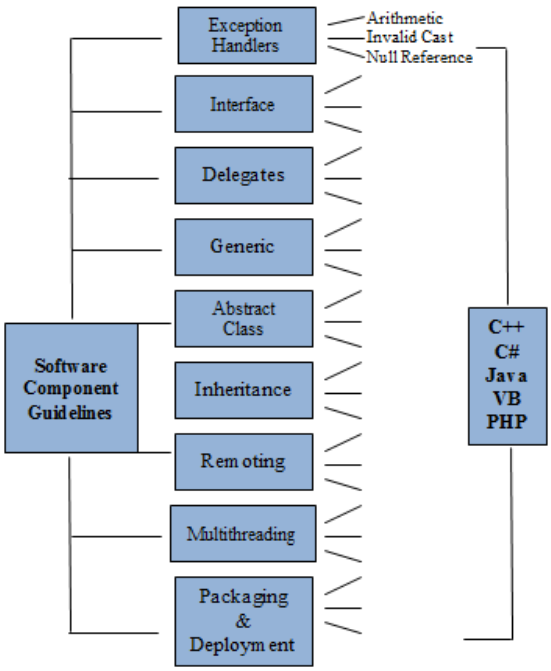

Figure 4 .NET Guider represents the Guidelines for New Component Development.

- Supporting components reuse by providing exception handlers.
- Interface based programming.
- Using Delegate to provide flexibility using strong type function pointers.
- Use of Generic<T>, make classes and functions more reusable with any data type.
- Inheritance (Is a relationship) ensure the relationship between classes and their derived objects; provide foundation of Object Oriented Programming concepts.
- Design and develop components with Interface based Programming, which abstracts the code complexity for user.
- Using Abstract Classes, which provide flexibility over interface, to alter the implementing class's functionality.
- Remoting, Object Marshaling
- Multithreading with Thread safety.
- Packaging and deployment.

A. Exception handlers

Exception kills the program execution if any error found. In C#, exceptions are type safe and cover system level and application level errors. System Exception provides the base class for all exceptions(System.DivideByZeroException, System.ArthmeticException,System.NullReferenceException) are examples of system level exceptions.

In Java, java.lang.Exception class, Provides the base for exception handling and all exceptions are divided in two categories as IO Exceptions or Runtime Exceptions.

TABLE 2 SHOW THE EXAMPLE OF EXCEPTION HANDLING IN C# AND JAVA.

C#.net	Java

using System;
class Program
{
static void Main()
{
Try // try method
{
int getvalue = 1 / int.Parse("0")
Console.WriteLine(getvalue);
}
// catch error
catch (Exception ex)
{
Console.WriteLine(ex);
} }}
	import java.io.*;
	public class ExceptionTest
	{
	public static void main(String args[]){
	try{ // try method
	int Id[] = new int[2];
	System.out.println("third element :" + Id[3]);
	}
	//catch error
	catch(ArrayIndexOutOfBoundsException ex)
	{
	System.out.println("Exception Found :" + ex);
	}
	System.out.println("Result Out");
	}}

As we mentioned the guidelines for new component development, so process of a component development can be understood by figure 9. This will start from .NET Guider's component skeleton creation and finish at registering component in global repository.

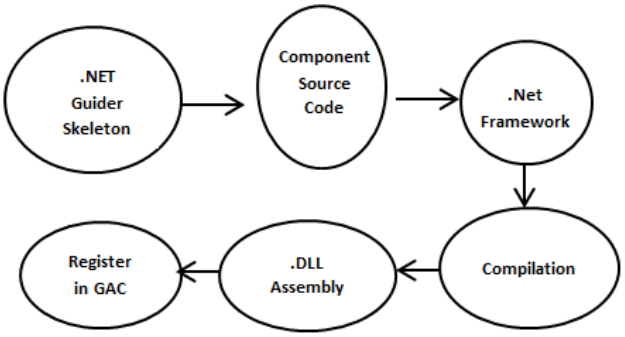

Figure 5: Component development process with .NET Guider.

V. REUSE ANALYSIS OF SOFWARE COMPONENT

The effectiveness of a component can be analysis by its reusability and portability. A component based software development saves, a lot of money and human efforts. But if a component is too complex to implement in further development, than it is not be called a Worthy Component. Specifically component complexity means, its required interfaces and provider interface's implementations. Further, Poulin [11] claimed that, during 1990 U.S Department of defense defined, that an effectiveness of a software component is determined by , number of lines of code required to obtain its functionality in new software development.

Hence, linked to above mention guidelines for reusable component design, helps to assess the component reusability factors and give an analysis about its effectiveness in development for reuse and with resue.so that component can be further redeveloped or modified with more enhanced attributes and functionality for further reuse.

In addition, for development of a .NET component [9] claimed that, C# is a better programming language as compare to JAVA, and mentioned the programming logic difference between them as below table:

TABLE 3 REPRESENT THE KEY DIFFERENCE FEATURES BETWEEN C# AND JAVA

	C#	Java
1	C# uses .Net runtime.	It uses Java runtime.
2	It has more primitive data types.	Less Primitive data types.
3	It use "const " to declare class constant.	It use Static final.
4	It support Struct type.	It does not.
5	It provide operator overloading.	It does not.
6	Have to use "Virtual" and "Override" keyword to override base class.	By default all members are virtual.
7	C# provide better versioing.	Java have limited options.
8	C# use delegates and events.	Java use interfaces and inner classes.
9	Parameter passed by using "Ref" keyword.	Parameter passes by value.
10	Have "Internal" as speical access modifier.	Java Does not have.
11	C# include native support for properties.	Java Does not.
12	C# directly supports enumerations.	Java Does not.
13	C# uses Indexes.	Java Does not have.
14	In C#, switch satement support integer and string expressions.	Java only support for integer expression.
15	C# supports "foreach loop" for quick iteration.	Java Does not.
16	C# checks overflows using "checked" statements.	Java Does not.

Besides this, we have conducted a pilot project to observe the efficiency of .NET Guider and evaluate its success rate in new component development. Although the size of project is small to critically evaluate the efficiency of defined guidelines in .NET Guider but we successfully able to reveal some facts related to defensive programing for component design and development. Figure 20 and table 2 represents the effective ness result of .NET Guider.

1. The workload for component design and development is divided efficiently.
2. A in depth domain analysis achieved.
3. A clear set of programming concepts, helped to avoid development complexity and ambiguity.

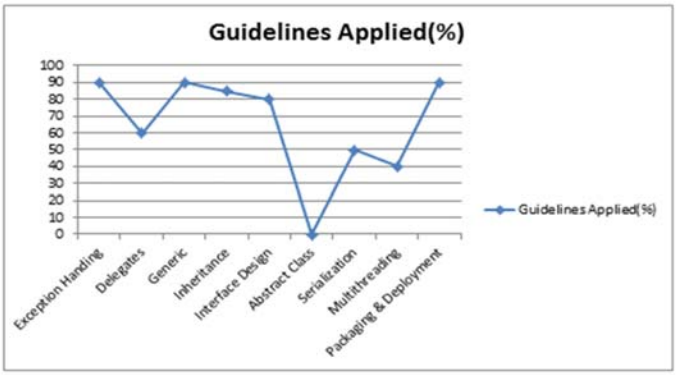

Figure 6. Represent the Component Design Guidelines in Percentage.

TABLE 4 REPRESENT THE COMPONENT DESIGN GUIDELINES IN PERCENTAGE.

	Guidelines	Guidelines Applied(%)
2	Exception Handing	90
3	Delegates	60
4	Generic	90
5	Inheritance	85
6	Interface Design	80
7	Abstract Class	0
8	Serialization	50
9	Multithreading	40
10	Packaging & Deploy	90

VI. REUSE METRICS OF SOFTWARE COMPONENT

Metrics provide the reuse and complexity analysis of component sources code based on component's variable methods, classes and interfaces. Such metrics only work on the basis of component source codes but in the case of Black Box component, all internal source code properties are abstracted from end users, so such metrics analysis is not appropriate for black box component.

During our research we observed, component Reuse values (RV) can be justified and measured , based on number of fulfilled proposed guidelines in component development. Such reuse values is highly appropriate for black box components. So we identified and proposed a new reuse metrics for software component as (Reuse value%):

$$RV\% = \frac{\text{Number of Applied Guidelines}}{\text{Number of Proposed Guidelines}} * 100$$

The result of RV, will be able to express the reuse value in percentage (%), so it will be more easier to analysis the component reusability. For our reusable binary package component, RV is nearly 75%.

VII. Architecture of Proposed .NET Guider

A .NET guider will provide a platform for handpicked set of guidelines for component development. The selection of guidelines will be correlated with component's category, this would work as a category manager. The .NET guider will work with .NET framework libraries and provide a robust and efficient development environment. Figure 7 shows the proposed architecture of out .NET guider tool.

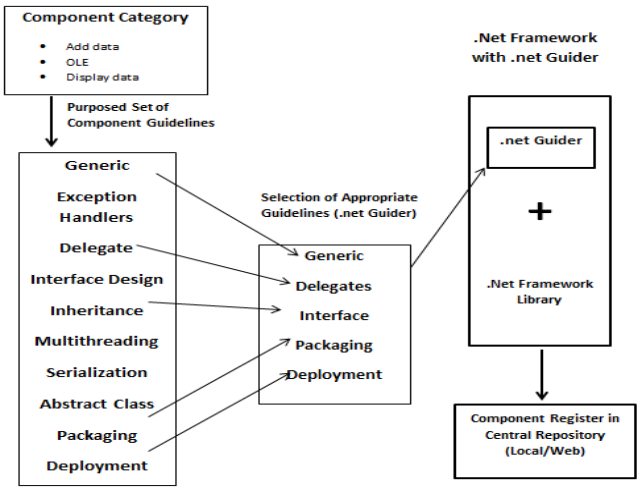

Figure 7. Proposed architecture of .NET Guider as new technique for creating reusable component.

Based on our proposed .NET Guider, we created a GUI interface for .NET Guider, it is subjected to futuristic development and creation of a GUI based Component Builder tool , which will provide an user friendly platform to developers where they can select the appropriate guidelines for component . Our .NET Guider will create a skeleton for developer based on their guidelines selections.

We believe .NET Guider will help to maintain development uniformity and complexity of component and ensure the cost and time saving as compare to traditional component development. Figure 8 and 9 represent the blue print of proposed .NET Guider.

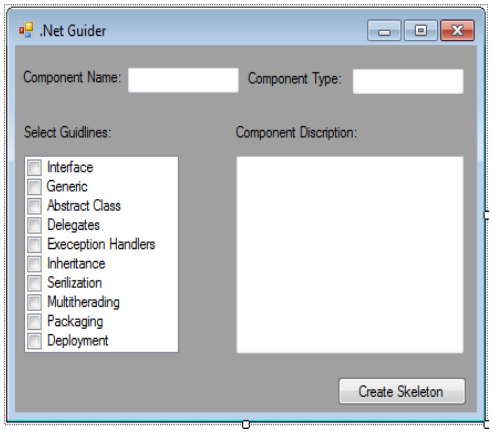

Figure 8. Provides the selection of suggested component development guidelines with component description.

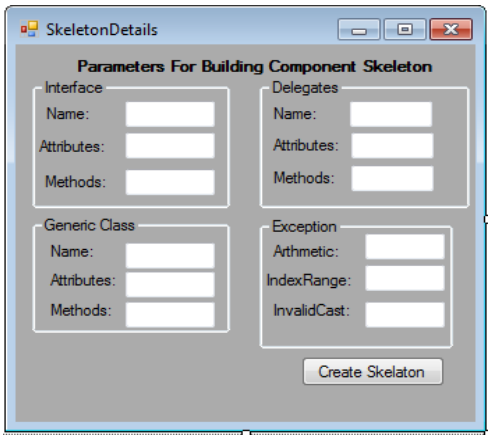

Figure 9. Creation of component code skeleton based on selected guidelines.

VIII. CONCLUSION

Software component guidelines offer the best practice for component development. software industry have enormous experience from previous development project's success and failures, so guidelines for new component development should be flexible to add in more artefacts and principals to manipulate them according to current businesses requirements, Which would help to save cost and labor and provide flexible development environment. At large scale of component development, various programming language can be used, so some universally applicable guidelines should be established.

Also from non-technical point of views, issues such as management, commercialization should have also some unified guidelines to survive in competitive market and maintain its profitability ratio in software industry.

REFERENCES

[1] Emn (2013) A guide to detect reusable components. [Online]. EMN.FR. Available from <http://www.emn.fr/z-info/emoose/alumni/thesis/ltorres.pdf> [Accessed December 1 2013].

[2] Lau, K. K. and Wang, Z. (2007) Software Component Models. IEEE Transaction Software Engineering, Vol. 33(10) p. 709-724.

[3] Lowy, J. (2003) Programming .NET Component. Cambridge, O'Reilly.

[4] MSDN (2013)An Introduction to c# Generics. [online]. MSDN. Available from <http://msdn.microsoft.com/en-us/library/ms379564(v=vs.80).aspx> [Accessed November 24 2013].

[5] Ramachandran M. (2008) Software Component: Guidelines and Applications. New York: Nova Science publisher.

[6] Ravichandran, T. and Rothenberger, A. (2003) Software reuse strategies and component markets. Communication of ACM. Vol. 6(8) p. 109-110.

[7] Thai,T. and Lam Q.(2003) Net Framework Essential. 3ed ed. Sebastopal, Calif, O'Reilly.

[8] Zirpins, C., Ortiz, G., Lamersdorf, W. and Emmerich, W. (2013) Proceeding of the first international workshop on engineering service compositions [Online]. IBM. Available form: http://domino.research.ibm.com[Accessed October 11 2013].

[9] Balagurusamy, E. (2010) Programming in C#.3 reed. New Delhi, Tata McGraw-Hill.

[10] Ramachandran, M. (2012) Guidelines based software engineering for developing software components, Leeds Metropolitan University, Vol.5, pp.-6

[11] Pouline, J. S (1994) Measuring Software Reusability, Proceedings of the Third International Conference on Software Reuse, Rio de Janeiro, Brazil, 1-4 November 1994