International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 - 5161

Available at http://inpressco.com/category/ijcet

Research Article

An Approach to generate Reusable design from legacy components and Reuse Levels of different environments

N Md Jubair Basha[Á*], Humera Sabhia[B] and Md Abdul Rawoof Sayeed[Ć]

[Á]Information Technology Department, MJCET, Hyderabad, India.
[B]Computer Science Department, Ummul-Qura, Makkah,Saudi Arabia.
[Ć]Mathematics Department, MJCET, Hyderabad, India

Accepted 20 Nov 2014, Available online 24 Dec2014, **Vol.4, No.6 (Dec 2014)**

Abstract

Software Components plays a vital role in the software development process in software industry. There are different artifacts executed throughout the life cycle of the process, each will have it's own prominence in the work products achievement. The reuse of each work product helps in the reduction of cost, reduces maintenance, reduces testing efforts as the benefits. When different components are evolved in different environments in the repository, there is a dire need of designing reusable components is still an issue. There are different approaches discussed literature which has it's own consequences. Many of them were proposed and only considered the code as component in a isolated environment. The issue of incompatability of legacy components motivates to propose the approach for the generation of reusable design components from legacy components and reuse levels in different environments. This approach will helps the software developers for faster software development.

Keywords: *Software components, reuse levels, legacy components, component based software development*

1. Introduction

Software engineering has been more focused on original development but it is now recognized that to achieve better software, more quickly and at lower cost, it is necessary to adopt a design process that is based on systematic software reuse.

Component-based software development is a new trend in software development. The main idea is to reuse already completed components instead of developing everything from the very beginning each time. Use of component-based development brings many advantages: faster development, lower costs of the development, better usability, etc. Component-based development is however still not mature process and there still exist many problems. For example, when you buy a component you do not know exactly its behavior; you do not have control over its maintenance, and so on. To be able to successfully develop component-based products, the organizations must introduce new development methods (Crnkovic *et al*, 2001), (Clemens Szyperski *et al*, 2002), (Basha, N.M.J *et al*, 2012), (Fahmi, S.A *et al*, 2008).

Although source code search systems are well known as being helpful to reuse source code, they have an issue that they often suggest larger code than what users actually need. This is because they suggest code based on the structure of programming languages such as files or classes (Tomoya *et al*, 2013). The main concern of the identified work was development of components on the specific programming languages. The discussions were presented and the drawbacks were brought out with an extensive literature (Ibraheem *et al*, 2014).

This motivates to consider the proposed project solves the identified issues in the literature. An approach has been considered for the generation of reusable design from legacy components in different environments has been achieved. The reuse level of the design component and legacy component will be compared for the verification of reusability in the reusable design components.

This approach is applicable to the software industry in the effort reduction and budget minimization. The paper is organized as follows. Section I provides the detailed introduction of the paper. Section II includes the Literature Survey focuses detailed theory on Component based software engineering and it's life cycle development. This section also presents the comparison of CBSD and traditional software development. It also discusses about the software reuse and different approaches in software reuse. The section-III provides the motivation of the work and the methodology to generate Reusable design from legacy components work. Section IV provides the related tools required for implementation of this work. Section- V concludes the paper.

2. Literature Survey

Lifecycle processes include all activities of a product or a system during its entire life, from the business idea for its

*Corresponding author: **N Md Jubair Basha**
DOI: http://Dx.Doi.Org/10.14741/Ijcet/22774106/4.6.2014.85

development, through its usage and its completion of use. Different models have been proposed and exploited in software engineering, and different models have exhibited their inabilities to efficiently govern all activities required for a successful development and use of products. One among them which supports for the successful development is Component Based Software Development and Software Reuse.

2.1 Component Based Software Engineering

CBSE implies building systems from components as reusable units and keeping component development separate from system development. This separation was significant implications for business goals (for example building a market of components), technologies (for example on the-fly deployment of new functionality) and legal & social issues(for example trust, responsibility and maintenance). Although there is no IEEE/ISO standard definition that we know of, one of the leading exponents in this area, (Clemens Szyperski *et al*, 2002), defines a software component as follows: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".

To achieve its primary goals of increased development efficiency, quality and decreased time to market, CBSE is built on the following four principles.

2.1.1 Reusability

The CBSE approach is effectively realized only if the components developed once, have the possibility for reuse numerous times in different applications. Industry has recognized quite a few reusability types as best practices: COTS (commercial off-the-shelf) components, product-line components, and open source components. CBSE is also useful in building architectural components for a particular system, without intention to reuse the components in other systems.

2.1.2 Substitutability

With substitutability, systems preserve correctness even when one component replaces another. This requirement boils down to the Liskov substitution principle.

Let q(x) be a property provable about objects x of type T. Then q(y) should be true for objects y of type S where S is a subtype of T.

This principle is realistic for functional properties, but it isn't obvious for extra-functional properties because it depends on other factors, such as a system context. For example, a faster component can cause a deadlock and break timing requirements in a system using a non-preemptive scheduling mechanism.

2.1.3 Extensibility

In CBSE, extensibility aims to support evolution by adding new components or evolving existing ones to extend the system's functionality. A typical solution to support component evolvability is to provide components multiple interfaces.

2.1.4 Composability

Composability is a fundamental CBSE principle. Every component-based technology supports the composition of functional properties (component binding). More rarely, there's support for composition of extra-functional properties, for example composition of components' reliability, or execution time, or memory usage. Composition of extra-functional properties remains one of the most important challenges of CBSE research.

The major advantages in using the software components are (Basha, N.M.J *et al*, 2012), as follows

- Reduced Cost & Schedule: As the component is reused the cost and the time needed to develop the component is saved. If needed the component can be modified.
- Reduced Testing effort: As testing requires more than 60% of software development effort. Using domain specific component approach testing effort is reduced.
- Enhanced Quality: All successfully developed components goes through the certification process. Hence the components available in the repository are usually of good quality.

Many organizations have developed their own domain specific components which act as an asset, so that they could be reused later. Even if the component is not reused as a mirrored component, it can be modified. The effort required to alter a component is less compared to that developed from the scratch. However there is a need for an effective approach to identify and develop the components.

The difference between the traditional software development and component based software development has explained below.

Table 1: Comparison of CBSD and Traditional Software Development (Fahmi S.A *et al*, 2008)

Component Based Software Development	Traditional Software Development
Housing System from already available components.	Housing system from bottom-line.
Components and Systems integrated from those components are developed through interfaces.	Software System is developed by following all the phases of life cycle.
Component selection, identification and evaluation are special life cycle phases.	There is no provision for selection, identification and evaluation in this life cycle.
Effort is required for component selection, testing & verification only once.	Much effort is required for throughout the software development cycle.
Reuse of components guide to development of component in a faster manner	Reuse property is applicable in a less manner.
Cost and time management is required less.	Cost and time management is applicable for every project.
Depends on the requirements, component management can be done applicable to the project.	Software development activity has to be carried out for every project.

2.2 Component Based Software Development Life Cycle

Spiral Models characteristics are involved in the Component Based Development (CBD) Model. The CBD Model consists of the applications from the grouped software components (called Classes). The component development starts with the identification of the potential components. This is achieved by identifying the data to modify by the application and the relevant algorithms that will be applied. For this the data and algorithms are encapsulated into the classes.

The components created in the software projects are stored in repository. Once potential components are identified, the repository is mined to check whether the desired components are present in the repository. If available, they are retrieved and are reused. If the component does not exist in the repository, it is engineered using the object-oriented methodology. The first iteration of the application to be build is composed of components retrieved from the repository and new components engineered to meet the novel needs of the particular application.

Process Flow is reversed back to the spiral model and will ultimately continue the component assembly looping during subsequent passes through the component life cycle as shown in Figure 1. Software Reusability can be achieved by CBD model which is highly useful to the software engineers. The usage of software reusability by QSM Associates Inc., reports component assembly moves to a reduction in development life cycle, 84% reduction in project cost, and a productivity index of 26.2, compared to an industry norm of 16.9.With these results, the robustness of the component repository and CBD model provides many advantages to the software engineers.

(Sommerville *et al*, 2004) has provided a sequential process for CBSD as in Figure 1. It includes six steps which are as follows

- The user requirements are developed in outline rather than in detail as specific requirements limit the number of components might be used.
- A complete outlined place of requirements are used to identify as many components as possible for reuse.
- Requirements are sophisticated and tailored so that they can comply with components.
- Development of Architectural design is possible with the above steps.
- With this system architecture can be designed. Steps 2 and 3 can be repeated.
- Lastly, the components can be integrated which makes the complete system.

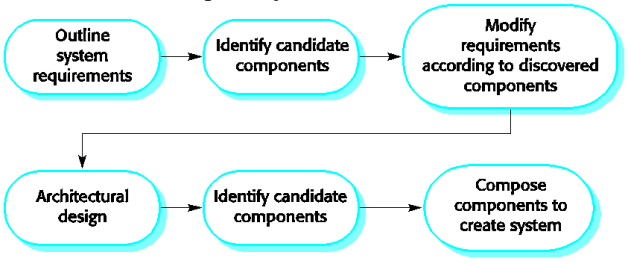

Figure 1: Component Based Software Development Process

2.3 Software Reuse

Software Reuse is the use of available software or to build new software from software knowledge. Reusable assets can be either reusable software or software knowledge. Reusability is a property of a software asset that indicates it's probability of reuse (William B Frakes *et al*, 2005). Software Reuse means the process that use "designed software for reuse" again and again (Xichen Wang *et al*, 2011). By reusing software, we can manage complexity of software development, increase product quality and makes faster production in the organization.

The component system includes the selecting, classifying and managing the components included in the repository and also the development of new components. The component repository should be spread throughout the development organization and that the components are accessible. The component repository should preferably be shared between several different products. It means that the component system should serve several projects. Whenever the new projects to be taken up, then the relevant components shall be needed for the development process. The project proposals should be reviewed by a group consisting of experienced designers and also someone from component department forming a software component committee. They should judge whether the proposed components are needs to be developed or not. If it is decided to construct the component, it is forwarded to component construction with a deadline. When ready, it is added to the component repository which then takes a new version state as showed in the Figure 2. As component is being used, the software component group should analyze it's value. Which component is used most? Which are not used at all? How much you gain from the components? This analysis helps to develop the component system.

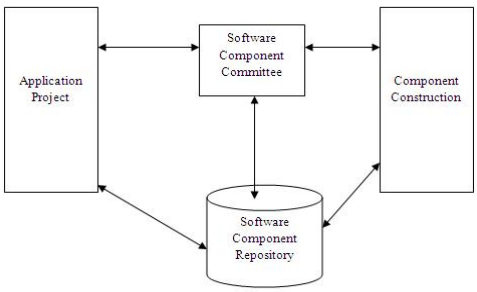

Figure: 2 An Organization for Component Management

2.4 Approaches for Software Component Reuse

The software component reuse benefits the development with many ways as discussed earlier. There are different approaches for software component reuse. (James Neigbors *et al*, 1984) has proposed a mechanism called Draco which aids in the construction of software systems. In particular, it is more concerned with the reuse analysis and design information in addition to the programming language code. But still there is an issue this mechanism will supports only one environment. (Krueger *et al*, 1992) has identified about reuse in design and code scavenging, but the proposed work was limited to the only code

scavenging but not up to the reusable design. This issue was taken up by many researchers to solve it.

(Johnson, Ralph E *et al*, 1997) has identified the strength of frameworks is the fact that the framework is expressed in code, it might be better to improve object-oriented languages so that they can express patterns of collaboration more clearly. Again the issue generation of reusable design of components is still highlighted.

(Maggo *et al*, 2014) has proposed a approach that provides a support for reusability evaluation at functional level rather than at structural level. But still there is a lagging in structural level reusability. (Ibraheem *et al*, 2014) surveyed on software reusability and identified an issue of major incompatability of legacy components. This motivates to solve the solve the problem of generating reusable design components in different environments.

The issue addressed in (Ibraheem *et al*, 2014) about the incompatibility of legacy components motivates to focus on the solution. (Tomoya *et al*, 2013) has proposed a new search technique that considers the past reuse. This technique detects reused code by using a fine grained code clone detection. But this is limited to only legacy code which is not focusing to the generation of reusable design components.

3. Proposed Work

The literature provides a brief introduction to the software components reuse approaches and the unsolved issues. This motivates to consider the problems identified in (Ibraheem *et al*, 2014), (Tomoya *et al*, 2013) to develop a strategy to identify the reusable components.

3.1 A Methodology to generate reusable design from legacy components

The motivation clearly shows the unsolved issue in the literature. The proposed methodology will solve the problem identified as follows

1) Consider the components of different environments and arrange them in the repository.
2) Construct a Reachability Matrix (RM_{LC})for every component.
3) Classify the components based upon the environment in which they were developed.
4) Make a note of SLOC of each component which was in the repository.
5) Perform Reverse Engineering process on the classified components.
6) Repeat step 4 for different environment components.
7) Step 5 generates a design artifact.
8) Generated design artifact in step 5 must match with the Reachability Matrix (RM_{DC}) with it's attributes.
9) If RM_{DC}=RM_{LC} then level of reuse can be identified.
10) Step 9 can be achieved by the interaction of the components (Coupling of components).
11) Step 9 fails, then the level of cannot be checked.
12) Repeat step 2 to step 9, for different environments.

Reachability Matrix (RM_{LC}): This can be generated by considering the cyclomatic complexity of the code.

Reachability Matrix (RM_{DC}): This can be generated from the state transition diagram of the structural representation of the component.

4. Tools

The proposed project needs different set of tools to perform various activites. NetBeans GUI is used for the development of legacy components.Net Beans profiler is used to identify the interaction between the components. StarUML is a tool to develop artifacts of the system for achievement of work product. R tool is used to generate the statistical process for the different environment legacy components. MATLAB is used to calculate the level of reusability and to generate the graphs for the proposed hypothesis.

Conclusion

Software industry demands reuse for budget minimization and effort reduction. This can be attractively achieved by reusable components. Many approaches were presented and limited only upto the mark of reusing code. But the issue of incompatability of legacy components motivates for designing reusable components from legacy components in different environments was discussed.. In this scenario, an approach for the generation of reusable design from legacy components and identify the reusability level has been proposed. The future work leads for realization of the proposed methodology that can be tested on different environments with the respective reuse levels.

References

Crnkovic Ivica, and Magnus Larsson (2001) Component based software engineering-new paradigm of software development, *Invited talk and Report, MIPRO* pp 523-524.

Clemens Szyperski, (2002) Component Software: Beyond Object-Oriented Programming, 2nd Edition, Addison Wesley Publications.

Basha, N.M.J.; Moiz, S.A. (2012), Component based software development: A state of art , *Advances in Engineering, Science and Management (ICAESM),International Conference on* , pp 599-604

Fahmi, S.A; Ho-Jin Choi (2008), Life Cycles for Component-Based Software Development, CIT Workshops. *IEEE 8th International Conference on Computer and Information Technology Workshops*, pp 637-642.

Sommerville I (2004), Software Engineering, 7th Edition, Pearson Education.

William B. Frakes, Kyo Kang (2005), Software Reus and Research: Status and Future, *IEEE Transactions on Software Engineering*", Vol. 31(7), pp 529-536.

Xichen Wang, Luzi Wang (2011) : Software Reuse and Distributed Object Technology, *International Joint Conference in Computational Sciences and Optimization(CSO)*, pp 804-807.

Neighbors, James M (1984), The Draco approach to constructing software from reusable components, *IEEE Transactions on Software Engineering*, Vol 5, pp 564 574.

Krueger, Charles W 2 (1992). Software reuse, *ACM Computing Surveys (CSUR)*, Vol (24), pp 131-183.

Johnson, Ralph E (1997), Frameworks=(components+ patterns), *Communications of the ACM* Vol(40), pp 39-42.

Maggo, Surbhi, and Chetna Gupta(2014), MLP based Reusability Assessment Automation Model for Java based Software Systems, *International Journal of Modern Education and Computer Science*, Vol.8, pp 45-58.

Ibraheem Y.Y. (2014), Taxanomy, Definition, Approaches, Benefits, Reusability levels, Factors, and Adaptaion of Software Reusability: A Review of the Research Literature, *Journal of Applied Sciences* Vol 14(20) pp 2396-2421.

Tomoya, Keisuke Hotta, Yoshiki Higo, Shinji Kusumoto (2013), Reusing reused code, *Reverse Engineering (WCRE) 20th Working Conference on*, pp 457-461.