# Risk Identification while Component Integration

Dr. Latika Kharb
Associate Professor (IT),
JIMS, Sector-05, Rohini, New Delhi. INDIA.
latika.kharb@jimsindia.org

*Abstract*— **Till today, numerous attempts have been made by several organizations, software development teams, developers and researchers to improve CBS systems through improved tools and techniques [1]. Component-based development has emerged as a system engineering approach that promises rapid software development with fewer resources. In this paper, we have presented a risk identification approach for component-based development i.e. we have discussed a number of risks in various component-based development stages and these risks arise due to the widespread belief that it is a low risk development strategy. In the paper, a research probe on problems of component integration is delivered. In the first section, we have given a general introduction about CBS, followed by their characteristic features and then risks during component integration and the ways to manage these risks are discussed.**

*Index Terms*— **Risk identification, Component-based development, risk development, component integration.**

## I. Introduction

In the last years, software developers have been facing numerous challenges in software development leading to unbearable reduction in time-to-market as they have to develop software that retains not only maintainability, reusability, testability etc but also assure high dependability; because of increasing complexity and high competitiveness. Modern day systems could be called as the distributed systems of interacting components that work together to maintain the system as a whole [1].

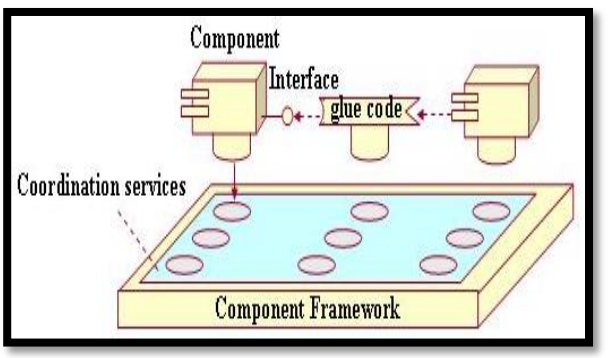


***Figure 1: A Software Component Framework***

Today most of the research has been inclined towards methods and approaches that work towards development of software systems and comparatively a very little work has been made using components for the development so that it can be used to evaluate the complexity of components being developed, using component integration. [2].

A very promising answer to these problems relies on the potential to obtain complex systems by the composition of prefabricated pieces of software known as components. A software component is a unit of composition that can be deployed independently by third parties and contains only the contractually specified interfaces and the explicit context dependencies [3]. Component based software system is modeled, designed and developed by integrating components through independent deployment i.e. built by combining the well-defined, independently produced pieces with self made components [4]. In software component based systems, software components are assembled and when they interact with each other: they satisfy predefined functions/ provide a pre-specified service in integration with other components. As a component is typically developed in a system environment, which is different from the environment of the final system, so it is difficult to predict the component behavior in the new system. We can define overall system complexity as a function of the interactions between system components and individual complexities of components. Complexity is a measure of the resources expanded by a system while integrating with a piece of software to perform a given task [5]. Complexity mainly results from the components' organization and interactions between these components. A software component is made up of three essential parts: the interface, the implementation, and the deployment [6]. A component interface consists of a variable part and a fixed part. The variable part corresponds to possible variants in the component's implementation and maps to the possible implementations; the fixed part expresses invariant characteristics of the component. Component integration involves combining components to form a software system i.e. combining their fixed and variable parts.

## II. Characteristics of Software Components

Software component based systems have changed the technology for software professionals to develop software applications and thus become one of the preferred streams for developing large and complex systems by integrating prefabricated software components which not only facilitates the process of software development but also solves many adaptation and maintenance problems. Some characteristic features of software components is discussed below namely, commercial off-the-shelf (COTS) components, source code unavailability, service granularity and plug and play.

### *A. Commercial Off-The-Shelf (COTS) Components:*

In component based development (CBD) approach, we use the option of buying off-the-shelf components (COTS) from third parties rather than developing the same functionality in-house. The Software Engineering Institute (SEI) defines a COTS product as one that is to be sold/ leased/ or licensed by the vendor in multiple and identical copies to the general public without any internal modification. The advantages of using COTS include: shortened development time, greater savings and enhanced functionality.

### *B. Source Code Unavailability:*

As software components are normally developed by a provider organization and used by multiple users that do not necessarily belong to the same organization. The implementation of components is generally not exposed to users, but only textual abstractions are attached as interface specifications. The signature of each interface is mentioned with an explanation of the component functionality, without any implementation details. Thus, the component source code is hidden, reducing development complexity for application developers at the component user end. The user gains the benefit of component services without having to be concerned with the implementation details, while the provider holds the component ownership rights even after component deployment.

### *C. Plug And Play:*

A software component is an autonomous entity with an inherent plug-and-play nature. It can be deployed in the system to provide services, brought offline, modified and again deployed in the same system providing modified functionality.

### *D. Service Granularity:*

Component interface defines the access point to a service provided by the component itself. The service that a component provides should be of use to someone, otherwise there is no reason to develop it.

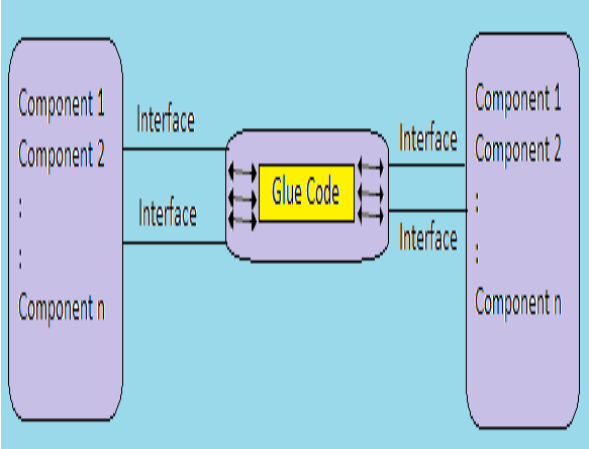


***Figure 2: Component Granularity***

The component should provide services sufficiently complex to justify its existence as a component. Simple components, in fact, can be more easily developed in-house. While a complex service may reduce the market for the component [7], and therefore the economic viability of developing it. Component granularity mainly affects the development environment [8] and, in particular, the production time and effort of the component developer.

## III. Research Probe : Risk Identification in Component Integration

More and more projects involve more integration of custom or commercial off the shelf packages, rather than in-house development and/or enhancement of software. While some project managers might think such outsourcing of development would reduce the overall risk; but each integrated component brings with it significantly increased risks in terms of system quality. By treating software security risk explicitly throughout the software lifecycle, the consequences of security failure could be identified early and could be used to implement security onto a system from the start up [9]. We have enumerated numerous factors that could increase risk in terms of system quality like:

*A. Coupling:* Might prove to be a strong interaction with the system/or a hazard to the system: when any component fails.

*B. Irreplaceability*: Due to shortage of components availability, we have to use a component that creates quality problems.

C. *Essentiality*: Some key feature/s of the system will be unavailable if the component does not work properly. These features might be the essential ones.

D. *Vendor quality problems*: If there is a high chance of the vendor providing you a bad component, the level of risk to the quality of the entire system is higher.

E. *Inflexibility*: Component unavailability leads to no changeovers, even if it's not flexible with new system.

F. *Poor component evaluation schemes*: No particular efficient test strategies are available for component evaluation in terms of system quality.

To summarize, component-based development poses significant risks to organizations intending to adopt the technology. Component developers, application integrators, as well as customers must know about component based systems and their advantages and/or disadvantages before working with the component-based applications.

## IV. A PROBE TO MANAGE RISKS

We have learnt how numerous types of risks in terms of system quality are prevalent in component based software systems.

| RISKS DURING INTEGRATION |
|---|
| Lack of Interoperability standards in different components of COTS |
| Lack of inflexibility while component evaluation |
| Incompleteness in components |
| Need of Add on creates difficulty in integration |
| Glue code criticality |
| Effort of integration increases beyond expectations |
| Compatibility Problem in integration of many components |
| Platform independence not achieved |
| Vendor support is not lifetime |
| Format variations of integrated components |
| Lack of control in terms of functionality and performance. |
| High cost on building integration environments |

***Table 1: Risks during Integration***

Risk of a project is reduced when the most important features are included at the beginning because the complexity of a project increases with its size, which means there is more opportunity for mistakes as development progresses [10].We have tried to give a probe into the matter and provide solutions to it; namely:

- We can make use of open standard technologies which provide basis for consistency among different COTS components.
- We can make use of compositional wrappers that can provide a comprehensive treatment of component integration, including improved possibilities for automation of component in future.
- We have to try our best in choosing exact match of COTS components with respect to system requirements instead of a supposed to be appropriate match of COTS components.

Trust is important while assembling the components, as users want to know that a component will function as required and "advertised" [11]. By managing the above said risks, developers can maintain this trust.

## CONCLUSION

As the software development managers are increasingly changing their focus towards component technology, it seems that in recent future; it will become one of the most preferred industry approaches towards development of improved software systems. Beyond these potential benefits, one critical aspect of component-based software has yet to be adequately addressed: the need for systems that will predictably exhibit the quality attributes required of them. In this paper, we have presented a risk identification approach for component-based development i.e. we have discussed a number of risks in component-based development integration that arise due to the widespread belief that CBD is a lower risk development strategy. In our paper, we have concluded a number of risks associated with component-based development and this identification will help to carry out activities that can minimize their ill-effects. However, we acknowledge that a lot more needs to be done in the area of CBD and future work will involve establishing a set of measurement strategies for the risks identified during various component-based development activities.

## REFERENCES

[1] Kharb L., Autonomic Computing Systems, ICFAI Journal of System Management, Vol. 5, Issue 4, November 2007.

[2] Kharb L., Software Component Complexity Measurement Through Proposed Integration Metrics,

Journal of Global Research in Computer Science, Volume 2, No. 6, June 2011.

[3]http://www.sei.cmu.edu/productlines/frame_report/comp_dev.htm

[4] Brown A. W., Large-scale Component-Based Development, Prentice-Hall, 2000.

[5] V. Basili, "Quantitative Software Complexity Models: A Panel Summary," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost, IEEE publication, October 1979.

[6] McInnis, Component-Based Development: The Concepts, Technology and Methodology, Castek Software Factory Inc., www.CBD~HQ.com

[7] C. Szyperski, Component Software: Beyond Object-Oriented Programming. Addison -Wesley, 1997.

[8] Brereton P, Budgen D. Component-based systems: A classification of issues. *IEEE Computer* 2000; 33(11):54–62.

[9] Kharb L., Software Security Improvement with Aspect Oriented Software Development (AOSD), ICFAI Journal of Systems Management, Vol 6, Issue 1, February, 2008.

[10] Kharb L., Proposed C.E.M (Cost Estimation Metrics): Estimation of Cost of Quality in Software Testing, International Journal of Computer Science and Telecommunications, IC Value:7.14, Vol 6, Issue 2, Feb 2015.

[11] Kharb L., CCTF: Component Certification & Trust Framework, International Journal of Scientific Research in Computer Science and Engineering, Vol-1, Issue-6. December 2013.