Importance of Software Component Characterization For Better Software Reusability

Nasib S. Gill
Department of Computer Science & Applications,
Maharshi Dayanand University, Rohtak – 124001
Haryana (India)
Email: nsgill_2k4@yahoo.com

Abstract

Component-based software development (CBSD) is the process of assembling existing software components in an application such that they interact to satisfy a predefined functionality. This approach can potentially be used to reduce software development costs, assemble systems rapidly, and reduce the maintenance overhead. One of the key challenges faced by software developers is to make component-based development (CBD) an efficient and effective approach. Since components are to be reused across various products and product-families, components must be characterized and tested properly. The present paper is a survey paper and firstly, it discusses CBD and related issues that help improving software reuse. Testing of third party components is a very difficult task in the absence a properly characterized software component. Besides improving software reusability, component characterization also provides better understanding of architecture, better retrieval, better usage and better cataloguing. This paper mainly discusses the essence of proper component characterization that ultimately helps the developers in software reuse, which is highly desirable in component-based software development. Further, paper also discusses other benefits of component characterization that are most essential in component-based development.

Keywords: Software components, Component-based Development, Software Reusability, Component Characterization.

1. Introduction

Component-based software development (CBSD) is both a subset as well as an extension of current software engineering practices. The prospect of increased product reliability and stability with shorter development time and reduced cost continues to fuel the on-going interest in CBD. This approach advocates the acquisition, adaptation, and integration of reusable software components, including commercial-off-the-shelf (COTS) products, to rapidly develop and deploy complex software systems with minimum engineering effort and resource cost. Although component-based development offers many potential benefits, such as greater reuse and a commodity-oriented perspective of software, it also raises several issues that developers need to consider Component-based software engineers intend to define and describe the processes required to assure timely completion of high quality, complex software systems that are composed of a variety of pre-produced software components.

2. Software Reusability

The software development community is gradually drifting toward the promise of widespread software reuse, in which any new software system can be derived virtually from the existing code. As a result, an increasing number of organizations are using software not just as all-inclusive applications, as in the past, but also as component parts of larger applications. In this new role, acquired software must integrate with other software functionality.

Software reusability is an attribute that refers to the expected reuse potential of a software component. Software reuse not only improves productivity but also has a positive impact on the quality and maintainability of software products. The move toward reuse is becoming so widespread that it has even changed software industry's vocabulary. For example, software acquired externally is described with such terms as commercial-off-the-shelf (COTS) or third-party software, commercially available software (CAS), and non-developmental item (NDI). When used as parts of systems, they are called components, componentware, and so on.

There are good reasons why the industry is moving toward large-scale reuse, including savings in time and money. The cost to develop a new system from scratch is significant. This has made custom software development very expensive. It is generally assumed that the reuse of existing software will enhance the reliability of a new software application. This concept is almost universally accepted because of the obvious fact that a product will work properly if it has already worked before. Some general reusability guidelines include ease of understanding, functional completeness, reliability, good error and exception handling, information hiding, high cohesion and low coupling, portability and modularity. CBD lacks appropriate reusability guidelines that could further benefit component-based development from cost-savings, time-savings, quality and productivity improvements, reliability improvements, etc.

3. Need of Component Characterization

A component can be considered an independent replaceable part of the application that provides a clear distinct function. A component can be a coherent package of software that can be independently developed and delivered as a unit, and that offers interfaces by which it can be connected, unchanged, with other components to compose a larger system.

Yacoub and others [4] characterized about a component under three main categories:

- Informal Description,
- Externals, and
- Internals.

Components can be considered as mentally driven building blocks of applications. Informal description encompass the following characteristics:

- *Age:* Age is a characteristic of a component that reflects its stability and maturity. When a component is first introduced, there is a high risk associated with its usage.
- *Source:* Components can be characterized by the source supplying the component, for example, Commercial Off the-Shelf (COTS), Military Off the-Shelf (MOTS), or Government Off the-Shelf (GOTS)
- *Level of Reuse:* A component can be reusable at different phases of the development life cycle. The nature of the components plays an important role in determining where in the development phase is the component reusable.
- *Context:* Target domains and applications should be considered when characterizing the context for a component.
- *Intent:* This includes the problem that the component solves.
- *Related Component:* This feature defines other well-known components that solve the same (or similar) problems.

Component external define its interactions with other application artifacts and with the platform on which the component resides. It encompass the following characteristics:

- *Interoperability:* Interoperability characterizes how the component interacts with other components in the same system. A component should be interoperable because it lives with other components in an application environment.
- *Portability:* It characterizes how the components interact with the underlying platform. It is the property of a component that specifies which platform the component runs on and how it can be ported to other platforms.
- *Role:* A component is characterized by its role (active or passive) that identifies what it provides to other application artifacts.
- *Integration Phase:* This characterizes when a component is integrated to the application. A component can be integrated at the development phase or at run-time.
- *Integration Frameworks:* A component should be characterized by identifying the underlying framework on which it runs.
- *Technology:* This characterizes the technology on which the component is dependent and the restrictions on the technology for developing the component and for reusing it in other applications.
- *Non-Functional Features:* Sometime it is essential to denote the non-functional aspects of a component specifically for mission-critical domains.

Internal aspects of a component encompass the following characteristics

- *Nature:* The nature of a component determines where the component can be used in the development process. A component can abstract a function, data, package, cluster and system abstraction or a system structure.
    - *Design Components*: A component can be a design principle or idea.
    - *Specification Component*: A specification can be considered a reusable component.
    - *Executables*: Executable components can be static libraries, dynamic link libraries (DLLs or VBXs), or executable pieces of an application.
- *Code*: Components can also be code that has been tested and executed in other projects.

4. Importance of Component Characterization

Component characterization is gaining substantial interest among researchers. Component characterization discussed above is a great help in the process of software reuse.

Most commercially available software components are delivered in binary form. We have to rely on the components' interface description to understand their *exact* capability. Even with the components' development documentation available, people would certainly prefer or can only afford to explore their interface descriptions rather than digesting their development details.

Following are some benefits offered by component characterization:

- *Improved Cataloguing*: Understanding the characteristics of a component play a major role in documenting, cataloging, and classifying the wide literature of available components.
- *Improved Usage*: Understanding the characteristics and properties of a specific component means better usage and ensures usage of the correct component in particular application development.
- *Improved Retrieval* from component libraries: Characterizing components could facilitate selection, acquiring, and acquaintance of components.
- *Improved Understanding* of architecture problems: Identifying characteristics of components helps in solving architecture mismatch problems such as the nature of the component and its control mode

Conclusions

Component-based software development (CBSD) can potentially be used to reduce software development costs, assemble systems rapidly, and reduce the maintenance overhead. Since components are intended to be reused across various products and product-families, possibly in different environment, components must be characterized and tested properly. The present paper firstly discussed CBD and all such related issues that finally aims in attaining higher level of software reuse. This paper has briefly explored some of the issues in context of component-based software development (CBSD). Important aspects of CBSD including software component characterization, software reusability and impact of characterization of software components have also been discussed.

References

1. Gill N.S. (2003). ***"Reusability Issues in Component-based Development",*** ACM SIGSOFT SEN, Volume 28, Number 4, 2003, ACM Press, New York, NY, USA, Pages 4-7.
2. Brown, W., and K. Wallnau (1998). "The Current State of CBSE", *IEEE Software*, October 1998, pp37-46
3. David S. Rosenblum (1998).*"Challenges in Exploiting Architectural Models for Software Testing"*, Proc. International Workshop on The Role of Software Architecture in Testing and Analysis, 1998.
4. Sherif Yacoub, Hany Ammar, and Ali Mili, *"Characterizing a Software Component"*. Available at www.sei.cmu.edu
5. Gill N.S. and Grover P.S. (2003). *"Component-Based Measurement: Few Useful Guidelines",* ACM SIGSOFT SEN, Volume 28, Number 6, 2003, ACM Press, New York, NY, USA, Pages 30-34.
6. Mila Keren (1999). *"Test Adequacy Criteria for Component-Based Visual Software"*, IBM Haifa Research Lab, 1999.
7. D'Souza, D. F., and Alan C. Wills (1998). "Objects, Components, and Frameworks with UML : The Catalysis Approach" , ISBN 0-201-31012-0 Addison-Wesley, 1998
8. Brown, A., and K. Wallnau (1996). "Engineering of Component-Bases Systems" in *Component Based Software Engineering*, Alan W. Brown (edt.) Software Engineering Institute, IEEE Computer Society, 1996
9. Gill N.S. and Grover P.S.(2004). "*Few Important Considerations For Deriving Interface Complexity Metric For Component-Based Systems,* ACM SIGSOFT SEN, Volume 29, Number 2, 2004, ACM Press, New York, NY, USA, Pages 30-34.
10. Meyer, B. (1999). "On To Components", *IEEE Computer*, Jan 1999, pp139-140
11. Iglesias, A., and J. Justo (1998). "Building System Requirements with Specification Components", *Proc. of Joint conference of Information and Computer Science*, JICS'98 , Vol III, pp499-502, Oct. 1998
12. Kroeker, K. (1998). "Component Technology", *IEEE Computer*, Vol 31,132-133, Jan 1998.
13. Szyperski, C. (1998). "Component Software: Beyond Object Oriented Programming", Addison Wesley Longman, 1998
14. Han, J. (1998). "Characterization of Components", *First Int'l Workshop on Component-Based Software Engineering*, in conjunction with ICSE'98, Kyoto, 1998
15. Elaine J. Weyuker (1998).*"Testing Component-Based Software: A Cautionary Tale"*. IEEE Software, September/October 1998, PP 54-59.
16. Alan W. Brown, Kurt C. Wallnau (1996). *"Engineering of Component-Based Systems"*, IEEE Computer Society Press, Los Alamitos, CA., 1996, PP 7-15.
17. Component-Based Software Development/ COTS Integration. Available at www.sei.cmu.edu/str/descriptions/cbsd.html.
18. Boris Beizer (1990). *"Software Testing Techniques"*, International Thomson Computer Press, 2nd edition (1990).
19. Jerry Gao (2000). *"Component Testability and Component Testing Challenges"*, San Jose University, CA, 2000.
20. Jerry Gao (2000). *"Tracking Component-Based Software Systems"*, San Jose University, CA, 2000.