Volume 2, No. 6, June 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

SOFTWARE COMPONENT COMPLEXITY MEASUREMENT THROUGH PROPOSED INTEGRATION METRICS

Ms. Latika[1*], Dr. Vijay Singh Rathore[2]

[1]Research Scholar,
NIMS University, Jaipur,India
latika.kharb@gmail.com
[2]Director
S.K. College, Jaipur,India

Abstract: Component based software paradigm has 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 is also changing the ways for software professionals to develop software applications. Till today, numerous attempts have been made by several organizations, software development teams, developers as well as researchers to improve software component systems through improved measurement tools and techniques i.e. through an effective metrics. Our paper is a simple attempt to work for the demand of an appropriate and relevant metrics and in this paper, we've proposed some integration metrics for the measurement of complexity of a software component, which could be used as one of the approaches for further guidance in component complexity measurement and problem reduction.

Keywords: Component complexity measurement, COSS, integration metrics.

SOFTWARE COMPONENT BASED SYSTEMS: AN INTRODUCTION

Software Component based systems has 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, so it's changing the ways for software professionals to develop software applications. However, a mismatch exists in the usability of metrics by academicians/researchers and industry men/developers. Even today, much of the metrics activity in industrial sector is based on metrics invented long ago in 70's. This mismatch exists between them because of the following reasons:

A) Researchers/academicians are mainly concerned with detailed and code-oriented metrics while the industrial sectors demand those metrics that could help them in their software process improvement. This difference between needs is the main cause behind the mismatch between usability criteria of various metrics.
B) Industrial sectors have to abide by some rules and regulations/standards of their company while academicians/researchers are not bound by any such rigid standards and can select/change metrics whenever needed according to their needs and requirements.
C) Researchers go for relatively small field works with small data (consisting of small programs) that could get them quick outputs. But the industry men have to go for large projects (to develop huge software). In academics, metrics may or may not be evaluated for correctness, quality and timeliness in hard values. They have to just provide data/values in form of theoretical validations. But the industry men are the one who deal with practical implementation of data and so they have to check each and every metric very minutely as even 1% error rate could be critical if it belongs to real life software development viz. aeronautical systems. A software component should be adequately packaged/ specified through its interfaces in order to facilitate proper usage. CBSD offers an effective approach to develop the components required to support various functions and processes for a particular area.

In short, in the last few years most of the research has been inclined towards methods and approaches that work towards development of software systems and in comparison, a very little work has been made for the development of measures/metrics that can be used to evaluate the complexity of components being developed, using component integration. The main issues in component metrics for capturing integration complexity and complex interfaces tend to complicate the testing process of the system [1].

INTRODUCTION TO SOFTWARE COMPONENT TECHNOLOGY

As the software development managers are increasingly changing their focus on component technology, it seems that in recent future Software Component based systems will become the most preferred industry approach towards development of improved software systems. In Software Component based systems, software components are assembled so that they interact with each other and satisfy predefined functions, so each component has to provide a pre-specified service with other components and thus interface is an important concern to be discussed before proposing metrics for measurement of integration complexity. Software Component based systems is a branch of the software engineering discipline which lays emphasis on decomposition of the engineered systems into functional

or logical components with well-defined interfaces used for communication across the components. A COSS is a software system that is modeled, designed and developed by integrating components through independent deployment. Software Component based systems are built by combining the well-defined, independently produced pieces with self-made components [2]. 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. A software component is made up of three essential parts: the *interface, the implementation, and the deployment* [3]. 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 collection of possible implementations; the fixed part expresses invariant characteristics of the component. Combining components to form a software system implies combining their fixed and variable parts. Combining the variable parts may easily lead to a combinatorial explosion of possible configurations. 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.

COMPONENT COMPLEXITY

Complexity is a measure of the resources expanded by a system while integrating with a piece of software to perform a given task [4]. Software complexity is that aspect of software which is used to predict external properties of the program like reliability, understandability & maintainability. Complexity measures the number of components and their interconnections and is typically based on internal product attributes, such as cohesion and coupling [5]. Complexity mainly results from the components' organization and interactions between these components. We can define overall system complexity as a function of the interactions between system components and individual complexities of components. Software Component based systems can be obtained as a result of the composition of some components with defined interfaces and then the component's functionality is implemented in its methods and is provided for other components through its well-defined interfaces [6].

Component Composition and Component Decomposition

A productive component market would deliver a wide range of components, designed for different integration mechanisms, programmed in different programming languages, and located at a diverse number of places. True collaborative software development demands that such diverse components can easily be composed, retrieved, and configured. However, in practice achieving such compositionality turns out to be rather complicated [7]. Components must be integrated through some well-defined infrastructure. This infrastructure provides the binding that forms a system from the disparate components [8].

Coss Metrics

The component paradigm focuses on developing large software systems by integrating prefabricated software components [9]. Component metrics are used to evaluate properties of something being measured, such as quality, complexity or effort, in an objective manner. A component's functionality is implemented in its methods, which is then provided to other components through its well-defined interfaces.

Component Integration

One of the main objectives of developing Software Component based systems is to enable efficient building of systems through the integration of components [10]. Integration can occur at different moments in time, each requiring a different integration mechanism. The component paradigm focuses on developing large software systems by integrating prefabricated software components [9] and also facilitates the process of software development to solve problems of adaptation and maintenance.

PROPOSED COMPONENT INTEGRATION METRICS

Software metrics can provide useful information to project managers and software developers by providing means of measuring the complexity of a software product. Software complexity means measurement of the resources expended in developing, testing, debugging, maintenance, user training, operation and correction of software products. Component complexity possesses two intrinsic complexities coming from methods inside the component, and extrinsic complexities resulting from interactions with other components i.e. incoming and outgoing interactions. In other words, according to our metric approach, a component oriented complexity metric is valid if it accurately measures the aspects of component-oriented system that influence its internal and external interactions. Incoming-interactions are any received interface that is required in a component, and/or any received event that comes to a component. Outgoing-interactions are any provided interface used and/or possible source of events consumed.

% Of Component Interactions ($Ci_{\%}$)

$CI_{\%}$ is defined, as a ratio of the available number of incoming interaction used to the available number of outgoing interaction i.e. the component interaction metric $CI_{\%}$ will provide a ratio of interactions in a system where, I_o is denoted for the number of outgoing interaction used, and I_i is denoted for the number of incoming interaction available. The equation is given by $CI_{\%}$.

$$CI_{\%} = \frac{I_i}{I_o} * 100\% \quad \ldots \text{equation (i)}$$

Interaction %Age Metrics For Component Integration (I%Mci)

We've proposed the Interaction %age metrics for component integration (I%MCI) in order to measure the interaction density among components in a software system. To measure I%MCI, we define I%MCI which is the ratio between the actual number of interactions to the $CI_{\%}$ which is the % of component Interactions metric (as in equation (i)).

$$I\%MCI = \frac{Ii + Io}{CI_{\%}} \quad \ldots \text{equation (ii)}$$

Actual Interactions (Ai)

We've proposed the metric actual interactions (AI), which is the ratio between the actual numbers of interactions ($I_i + I_o$) to the maximum number of performed interactions (I_{max}).

$$AI = \frac{I_i + I_o}{I_{max}} \quad \text{....} \quad \text{equation (iii)}$$

TOTAL INTERACTIONS PERFORMED (TIP)

It's based on measurement of actual interactions for component Integration (AI), as in equation (iii) to the total number of components (C). Here C includes C_1, C_2, C_3……..C_n.

$$TIP = \frac{AI}{C} \quad \text{…} \quad \text{equation (iv)}$$

Complete Interactions in A Cbs (Ci)

This component interaction metric would provide an estimate of actual interactions in a component. It's a sum of C_i and C_o which is the complexity of incoming and outgoing interactions respectively divided by the C, the total number of components.

$$CI = \frac{C_i + C_o}{C} \quad \text{…} \quad \text{equation (v)}$$

Such a metric could be helpful in improving systems quality because complex component integration complicates the testing and debugging processes, so it should account for effort/difficulty in integration process of components. Such a metric could be helpful in improving systems quality because complex component integration complicates the testing and debugging processes, so it should account for effort/difficulty in integration process of components.

DISCUSSION

Component metrics are used to evaluate properties of something being measured, such as quality, complexity or effort, in an objective manner. The component paradigm focuses on developing large software systems by integrating prefabricated software components and an attempt has been made to overcome the lack of some suitable metrics through proposed metrics. Such a metric could be helpful in improving systems quality because complex component integration complicates the testing and debugging processes, so it should account for effort/difficulty in integration process of components. For software developers, these values of $CI_{\%,}$ I%MCI, AI, TIP and CI will be useful to examine the density of interactions within the component. A component with high value of interactions indicates the need to employ high quality design and testing procedures to be followed. The values will also be valuable to assess the whole system interaction. The low value indicates a simple system, which also means lower effort to do the software risk analysis. As the complexity in Software Component based systems is related to the degree of difficulty of perception of the system, this complexity could be evaluated using three constituents of component integration:

A) Complexity of incoming interactions,
B) Complexity of outgoing interactions, &
C) Components of the whole system.

In our view of complexity, we concentrated on the complexity that is mainly dependent on structure. The metrics proposed in this paper could explain additional variance in measurement effort beyond that explained by other integration metrics. Our paper would provide insight into how application complexity evolves and how it can be managed through the use of metrics.

FUTURE WORKS AND CONCLUSION

An effort for contributing towards improved quality through reduced complexity by providing a new metrics for components integration has been made in this paper. In this paper, we've proposed some metrics to measure the complexity of software systems integrated through component based software paradigm and also discussed how a good metrics for the component complexity can be of great use for software developers and managers. The component metrics proposed above can further guide component complexity management in component based systems, by reducing problems encountered during software development.

REFERENCES

[1] Sedigh-Ali, S., Ghafoor, A., and Paul, R. A. 2001, Software Engineering Metrics for COTS-Based Systems. IEEE Computer, vol. 34, No. 6: 44—50
[2] Brown A. W., Large-scale Component-Based Development, Prentice-Hall, 2000.
[3] McInnis, Component-Based Development: The Concepts, Technology and Methodology, Castek Software Factory Inc., www.CBD~HQ.com
[4] 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.
[5] Goldberg A., Rubin K.S.: Succeeding with Objects, Decision Frameworks for Project Management, Addison-Wesley Publishing, 1995.
[6] C. Szyperski, Component Software: Beyond Object-Oriented Programming. Addison -Wesley, 1997.
[7] Merjin de Jonge, to reuse or to be reused: Techniques for Component Composition & Construction, 2003.
[8] Capt Gary Haines, AFMC SSSG, David Carney, SEI, John Foreman, Component-Based Software Development / COTS Integration, SEI: Copyright 2007 by Carnegie Mellon University, URL: http://www.sei.cmu.edu/str/descriptions/cbsd_body.html
[9] Dogru, A. H., and Tanik, M. 2003. A process Model for Component Oriented Software Engineering. IEEE Software, March/April: 34—41
[10] Ivica Crnkovic, Magnus Larsson, and Otto Preiss, Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes.

