IN-OUT INTERACTION COMPLEXITY COMPUTATION FOR COMPONENT BASED SOFTWARE PRODUCTS: FUZZY APPLICATION

***P.K.Bharti, @Dr. Ajay Agarwal**

**Phd Scholar, @Phd Supervisor*

ABSTRACT

In the current scenario of software engineering, Component-based software Development (CBSD) is the most intuitive appeal for developing large and complex software products by integrating pre-engineered, context-based manipulated, deployable software components. These components provide big advantages for software development such as shorter development times, better quality of the developed products and better maintainable software products. And these facilities can be translated into cost savings. This not only encourages the productivity and overall quality but decreases in time to market and maintain the software product. Today, the main emphasis of industry and researchers is on developing some impressive and efficient metrics and measurement tools through which they can minimize the complexity and over burdens arise during the In-interactions and Out-interactions among the components. In this paper we are proposing some simple methods and metrics for computing the complexity of composable components. This paper include the computation of total interactions,

Keywords: CBSD, pre-engineered, context-based manipulated software components, In-Out interactions, and complexity.

1. INTRODUCTION

1.1 Software Component

Software Component may be defined as software element offering a defined service or event, and able to communicate with other components of the software. These components can be designed and developed as either conventional software modules or object-oriented classes or packages of classes. A component may be:

1.1.1 Qualified component: address to the requirements of the system or product to be built in terms of functionality, performance, and reliability.

1.1.2 Adaptable components: Able to modify unwanted and undesirable features.

1.1.3 Assembled components: integrated and interconnected into an architecture to allow the components to be coordinated and managed effectively.

1.1.4 Updated components: existing components must be replaceable as new versions become available.

2. COMPONENT COMPOSITION

A software product can be made up of fully independent components or may be coordinate and cooperate with existing components. If the component is composable with other components it may share data, logic, information and other valuable contents. To share these each component should have: a) Interface, b) Services, c) Deployment techniques.

2.1 Interface: How a component can interact to other components to provide its services. It specifies what the parameters are what results to expect. It is the way through which coupling is done. It may be data, content, accidental. Interface is an essential requirement to integrate (couple) two or more independent components. It work as a protocol among the components when they assembled to provide their intended goal.

2.2 Services: It includes what type of services it is going to provide to other components, i.e. what is the use of that component in that particular software. It includes the functionality of that particular component which is going to integrate with existing or other independent components. Services includes-

a) Purpose	b) Robust
c) Reliability	d) Efficiency
e) Performance	f) Adaptability

2.3 Deployment Techniques: Deployment techniques include the executable codes of that component which is used to deploy that component with the codes of other components. Interface provides the basis for this deployment so that all the codes of all the components can be integrated in one single executable software to perform its required aim and goal. Deployment includes-

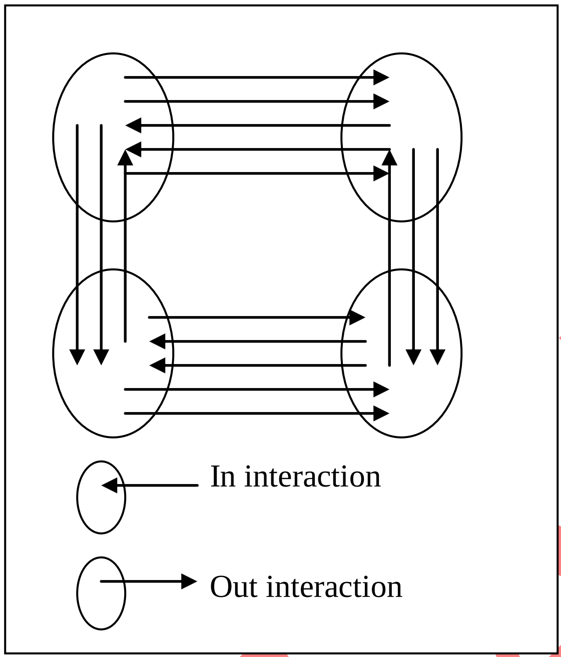
a) Physical code of new and existing components,
b) Changes in terms of coding in new and existing components wherever required,
c) Integration testing and
d) System testing.

3. PROPOSED INTERACTION COMPLEXITY COMPUTATION

3.1 Integration Complexity

When two or more components are integrated then they must share some interaction between themselves. We can define two interaction techniques which a component is required during integration-

a) **In Interaction:** All the incoming interactions to a component which are requested by other components to interact with that particular component. These are represented by incoming edges (arrows).
b) **Out Interaction:** All the outgoing interactions from a component which are requested by him to interact with other components. These are represented by outgoing edges (arrows).



3.2 Total Interactions of a component (TI)

Total Interactions of a component may be computed as sum of In-Interactions **(I_{in})** to and Out-Interactions (**I_{out}**) from that component.

Total Interactions of a component

$$\mathbf{TI = I_{in} + I_{out}}$$

3.3 Interaction Ratio of a Component (IR)

Interaction ratio of a component can be computed as total number of In-Interactions **(I_{in})** divided by total number of Out-Interactions (**I_{out}**).

Interaction Ratio of component

$$\mathbf{IR = I_{in} / I_{out}}$$

3.4 Average Interaction among Components (AI)

Average Interaction among components can be calculated as the ratio of sum of In-Interactions (**I_{in}**) and Out-Interactions (**I_{out}**) to the number of components participating in the integration **C_n.**

Average Interaction among components

$$AI = \frac{(I_n + I_{out})}{C_n}$$

3.5 Interaction Percentage between components (IP%)

Percentage of interaction between components can be given as the ratio of sum of In-Interactions (**I_{in}**) and Out-Interactions (**I_{out}**)to the Maximum possible interactions (**I_{max}**) among the components

Interaction Percentage between components

$$IP\% = \frac{(Iin + Iout) *100}{I_{max}}$$

4. DISCUSSION ON PROPOSED COMPUTATION

The primary objectives for component metrics computation include the following:

1. to better understand the interaction of components;
2. to assess the effectiveness of component integration;
3. to improve the quality of integration of components and final products.

A component should be of multiple uses, non-context specific, composable with other components, encapsulated, and a unit of independent deployment and versioning [Clements Szyperski and David Messerschmitt].

5. CONCLUSION

Since software components are developed using a compiler-oriented or common compiler based programming languages that has a limited vocabulary, an explicit grammar and well defined rules of syntax and semantics. Software components, like the idea of hardware components, should be interchangeable and reliable. Today modern reusable components encapsulate both data structures and the algorithms that are applied to the data structures.

Nowadays, the biggest challenge is to find ways of cutting the ties that inherently bind programs to specific computers and to other programs. Developers and Researchers are investigating several promising approaches, including a common language that could be used to describe software parts, programs that reshape components to match any environment, and components that have lots of optional features.

Software reuse is told to provide big advantages for software development such as shorter development times, better quality of the developed products and better maintainable software products. And these should translate into cost savings.

6. REFERENCES

1. [Ran01] Nadeesha Ranasinghe, History of Component Based Development, http://infoeng.ee.ic.ac.uk/~malikz/surprise2001/nr99e/article1

2. Pierre N. Robollard, Philippe Kruchten, with Patrick d'Astous "Software Process with UPEDU", ISBN 81-297-0309-2

3. Stephen R. Schach "Software Engineering", seventh edition, TMH.

4. "Software Engineering A practitioners Approach", Roger S. Pressman, sixth edition, TMH International edition.

5. A. Salesh, "Measuring the complexity of component based system architecture", ieeeeexplore.com, IEEE(2004).

6. Lalit Kharb and Rejende Singh, "Complexity Metrics for Component oriented software system", ACM SIGSOFT software engineering Notes, march 2008, Volume 33, Number2.

7. Nasib S. Gill and Balkishan, "Dependency and interaction oriented complexity metrics of component based systems ", ACM SIGSOFT Software engineering notes, January 2008, volume 33 number 2.