Research Journal of Information Technology

ISSN 1815-7432

Academic Journals Inc.

www.academicjournals.com

Research Journal of Information Technology 4 (4): 228-234, 2012
ISSN 1815-7432 / DOI: 10.3923/rjit.2012.228.234

Criticality Estimation for Software Components: An Empirical Approach

[1]Pooja Batra, [2]Arun Sharma and [1]Maitreyee Dutta
[1]Department of Computer Science, National Institute of Technical Teachers' Training and Research, Chandigarh, India
[2]Department of Computer Science and Engineering, Krishna Institute of Engineering and Technology, Ghaziabad, India

Corresponding Author: Dr. Arun Sharma, Professor of Head-Department of Computer Science and Engineering, Krishna Institute of Engineering and Tech., Ghaziabad, India

ABSTRACT

Components are developed by different programmers in different environment; interactions among components can be characterized by the use of component interface or through other component interactions. Interactions results dependencies, any modification to component can cause the change of component functionality, because the composite functionality is reflected in different components. Also day by day the complexity of software applications growing and there is more emphasis on reusability and maintenance, to maintain these aspects there is a need to identify the highly dependent component which is called as critical component. This paper introduced an approach to find critical components in a component based system. The proposed approach is also validated on existing metric and results have been found very positive.

Key words: Component, dependency, criticality, functionality, software applications

INTRODUCTION

Component-based software systems are built by assembling preexisting components, which provides high flexibility and reusability. The major work with Component-Based Development (CBD) is component integrating rather than writing code and developing everything from scratch. Fazal-e-Amin *et al*. (2011) advocates software reuse which yields benefits such as a reduction in the development time, cost and effort required and an increase in the productivity and quality. In conventional software development, the concept of complexity is related to the difficulty to analyze source code, modify and maintain its modules. However, this concept is different in CB systems because the maintenance and reconfiguration only involves replacing, adding and deleting components rather than source code changes. Alhazbi and Jantan (2007) illustrated that component is usually configured only once during the build-time. Thereafter, versions of component Based Systems are generated by adding a new component, deleting a component or replacing a component with a new version. Therefore, in Component Based systems, the complexity resides in the dependencies among components.

Dependency is a relationship involving two or more components, where a change of state in one or more component leads to a potential for a change of state in one or more other Components.

In the simplest case of dependency, a unidirectional dependency between two Entities, d (A, B), implies that A depends upon B. If A depends upon B, then a change in B implies a potential or possible change in A. A is referred as dependent and B as the Antecedent. In this study we have used this information to find Critical Component in a Component Based System. Critical components are those which should be properly taken care of at the time of maintenance or at the time of updation. While adding a new component, while deleting a component or while replacing a component with new version dependent components must be known. So, more dependent components are more critical. As criticality depends upon the dependencies and dependencies promote interactions so proper dependency analysis is required (Sharma *et al.*, 2007, 2009).

Graph based approach has been widely used by researchers for representing Dependency. Lisa and Delugach (2001) represented the dependencies in terms of conceptual graphs. Conceptual graphs are formal, logic based and semantic network language and are used in domain modeling and requirement modeling. A conceptual graph is made up of concepts, relations and a possible value. Relations are connected to concepts with directed arrows. Hierarchies represent the subtype/super type. The proposed methodology also implemented by taking three examples and compared it against other existing methodologies, including UML.

Yi and Nahrstedt (2001) categorized the dependency into functional dependency and resource dependency and directed graph based approach is used to represent these dependencies. Guo (2002) suggested a category theory based framework to model component dependencies. Category theory is a branch of Mathematics, designed to describe various structural concepts from different mathematical fields in a uniform way. So, the work defined several definitions and represented component dependencies by using these definitions. The proposed model and definitions are very much similar to object oriented data model to represent various elements like subtypes, attribute inheritance etc. the proposed framework, However, is not being implemented empirically.

Vieira and Richardson (2002) represented the dependency relationships by using pomsets description. Pomsets define a set of labeled events in a sequence and is able to express what can take place after a particular component access point is called by another component. In other words, it is a sequence of one or more actions in the form of concurrent regular expressions. Pomsets are considered to be compact and low computational overhead and therefore can be used to describe the component dependency effectively.

Stafford *et al.* (2003) represented dependency relationship between two or more components by a graph. This directed graph is further used to form an adjacency matrix AM [i, j]. If there is a dependency between two components Ci and Cj, then AM [i, j] is 1, else it is 0. This representation is used to compute the total dependencies of a component and of the system. Only the presence of the dependency is stored, not the type of the dependency or the event/interface through which these components are dependent.

Li (2003) described the dependency in terms of adjacency matrix and component dependency graph. He categorized dependency into eight categories, namely, data, control, interface, time, state, cause and effect, input/output and context dependencies. Author considered these eight dependencies to measure the final dependency by using Boolean operators. By using this approach, several dependency relationships can be deduced. Like, if adjacency matrix obtained by all these dependencies is upper triangle matrix, it means that all the dependencies are unidirectional. Similarly, if adjacency matrix is a diagonal matrix, it means that there is no relationship and components are isolated. However, except these two information, it is failed to extract other important details like number of interaction parameters, types of these parameters, interaction complexity and others.

Alhazbi and Jantan (2007) illustrated why dependency analysis is required while adding, deleting and replacing a component in component based system, also author proposed that beside direct dependencies indirect dependencies can also be stored using matrix representation.

Wu and Offutt (2003) performed static analysis to identify the interface events and the dependence relationship by using UML notation. The work provided a UML based framework to evaluate the similarities among old and new components.

Sharma *et al.* (2009) proposed a link-list based approach to represent the dependency relationship in CBS. Every component is represented by a node, consisting of all its required and provided interfaces. The provided interfaces of a component are accepted by other components as required interfaces. Component with required interfaces is called as dependent on component with provided interfaces. By using link-list based representation, along with the dependency, other information like name, number and type of interfaces, which are responsible for interaction can also be extracted. This information can further be used to measure the complexity of interaction, incoming and outgoing interaction density, most critical components and isolated components. All this information can be helpful in understanding, testing, debugging and maintaining the system.

PROPOSED APPROACH

Criticality depends upon dependencies or interactions; we can measure interactions among components by weight assignment, which is based upon type and number of interactions between components. Steps followed in approach are:

Step 1: We follow linked list based approach proposed by Sharma *et al.* (2009) to represent dependency. In this approach along with dependency, we can store information about dependency also

Step 2: We assign different weight values to the edges of graph, based on number and type of interactions, used in between components. We classify data types in three categories, namely, Simple, Medium, Complex. Structured Data types like integer, float, character can be taken as simple. Structured data types like date, string, array list and vector can be taken as medium and data types like class type, built-in and user-defined components, pointers/reference and others can be taken as complex. Depending upon number and type of interactions we can assign weight to the edges, which are given in Table 1.

Step 3: After assigning the weights according to their number and types, these are added together to get total criticality of a component

Implementation: To implement the proposed methodology, we took a case study of Security System which has total nine modules as shown and described in the Fig. 1 and in the following section:

- Login component depends upon Hierarchy Management and both interact each with four interface methods with arguments

Table 1: Interaction weights

Data type number	Simple	Medium	Complex
1-2	0.1	0.25	0.5
3-5	0.2	0.5	0.75
>5	0.3	0.75	1

Table 2: Criticality with proposed approach

Name of component	Criticality
Server	2.50
Time management	2.05
Alarm management	2.05
Access management	2.00
Calculation script	1.45
Door management system	1.20
Devices	0.85
Login	0.75

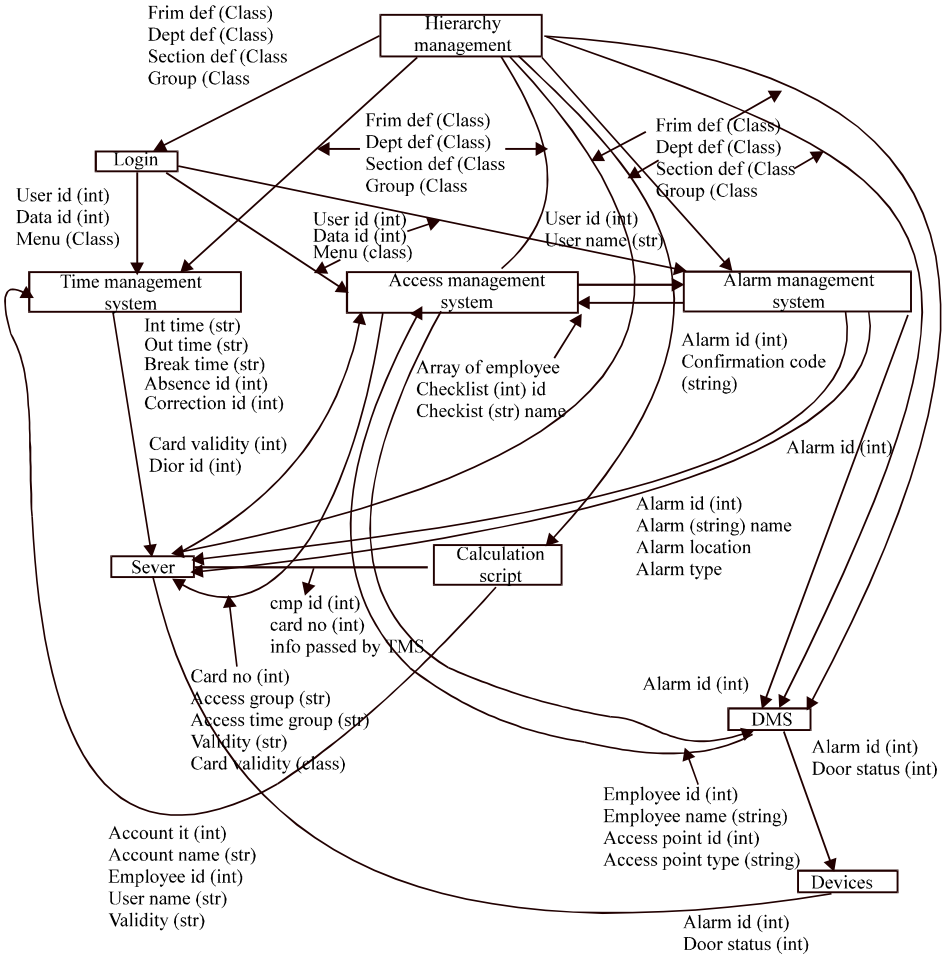

Fig. 1: Component diagram of security system

- Firm def (class)
- Dept def (class)
- Section def (class)
- Group (class)

And class is a complex type argument so from Table 2 criticality of login components is 0.75:

- Time Management component depends upon three components login, calculation script and hierarchy management.

For login component, there are two Interface methods with simple arguments:

- Userid(int)
- Dataid(int)

So, contribution of these two arguments towards criticality is 0.1 and these components share one interface method with complex argument menu (class).

Criticality of Time management component with login component is 0.1+0.5 = 0.6.

For Calculation Script component there are interactions with four Interface methods with simple arguments and three interface methods with medium type arguments.

Therefore, criticality of Time management component with this component is 0.2+0.5 = 0.7.

For Hierarchy Management interactions are with interface method with four complex type variables:

- Firm def (class)
- Dept def (class)
- Section def (class)
- Group (class)

Therefore, criticality of Time management component with this component is 0.75.

Thus, total criticality is 0.6+0.7+0.75 = 2.05.

Similarly, criticality of other components can be obtained.

RESULTS

Criticality of all components from proposed approach is calculated by assigning weights to the interactions depending upon their number and type. After assigning weights to all interactions of other components to which a component is dependent these weights are added to get final criticality which is given in Table 2.

From the above Table 2, we find that in terms of number for interactions and type of arguments Server component is most dependent in the above Component Based System. Obviously most dependent is most critical component. If we can find it in a component based system this can be helpful while maintaining and reconfiguring a system.

Table 3: CpIM values

Name of component	CpIM	Correlation coefficient with proposed approach
Server	3.44	0.89
Time management	2	
Alarm management	2.11	
Access management	2.77	
Calculation script	1.88	
Door management system	1.44	
Devices	1	
Login	1.44	

Validation of proposed approach: Correlation analysis is performed for validation of proposed approach with interaction metrices available. The available metrics are Component Interaction Metrics, Actual Interaction metrics and Complete Interaction Metrices. Amongst these, complete interaction metrices will estimate actual interaction of a component and our approach is also giving weights to interaction which is based upon number and their type. So we will use complete interaction metric for validation proposed by Chen *et al.* (2009):

$$CpIM = \frac{Ic + Oc}{C} \quad (1)$$

where:
Ic = Sum of Complexity of incoming interaction
Oc = Sum of complexity of outgoing interactions
C = Total No. of components

Table 3 shows CpIM values for given case study. Correlation coefficient calculated is 0.89 which is a very high positive correlation value. It shows that there is strong association between interaction calculated by metric and interaction calculated by our proposed weight assignment method. These values will be helpful during testing maintenance and reconfiguration of system. These correlation coefficients and their interpretation validate the proposed complexity metric for components complete interaction metric for each component.

CONCLUSION AND FUTURE WORK

To experiment the proposed approach, a small case study is taken with nine components, but a component based application may consist of hundreds of components. In future a big application may be taken to experiment the approach. If the criticality of component is high then its maintainability is high, reusability is low and testing efforts are high. The proposed approach can be used to identify the nature of critical components in eBS which in turn may be helpful in maintaining system on later stages.

REFERENCES

Alhazbi, S. and A. Jantan, 2007. Dependency management in dynamically updateable component based system. J. Comput. Sci., 7: 499-508.

Chen, J., W.K. Yeap and S.D. Bruda, 2009. A review of component coupling metrics for component-based development. Proceedings of WRI World Congress on Software Engineering, May 19-21, 2009, Xiamen, pp: 65-69.

Fazal-e-Amin, A.K. Mahmood and A. Oxley, 2011. A review of software component reusability assessment approaches. Res. J. Inform. Technol., 3: 1-11.

Guo, J., 2002. Using category theory to model software component dependencies. Proc. IEEE Int. Conf. Workshop Eng. Comput. based Syst., 9: 185-192.

Li, B., 2003. Managing dependencies in component-based systems based on matrix model. Proceedings of Net Object Days (NOD), AgeS Workshop, September 22-25, 2003, Erfurt, Germany, pp: 22-25.

Lisa, C. and H.S. Delugach, 2001. Dependency analysis using conceptual graphs. Proc. Int. Conf. Concept. Struct., 9: 117-130.

Sharma, A., P.S. Grover and R. Kumar, 2009. Dependency analysis for component-based software systems. ACM SIGSOFT Software Eng. Notes, 34: 1-6.

Sharma, A., R. Kumar and P.S. Grover, 2007. Managing component-based systems with reusable components. Int. J. Comput. Sci. Sec., 1: 60-65.

Stafford, J.A., A.L. Wolf and M. Caporuscio, 2003. The application of dependence analysis to software architecture descriptions. Lect. Notes Comput. Sci., 2804: 52-62.

Vieira, M. and D. Richardson, 2002. Analyzing dependencies in large component-based systems. Proceedings of the 17th IEEE International Conference on Automated Software Engineering, September 22-27, 2002, Edinburgh, UK., pp: 241-244

Wu, Y. and J. Offutt, 2003. Maintaining evolving component-based software with UML. Proceedings of 7th European Conference on Software Maintenance and Reengineering, March 26-28, 2003, Benevento, Italy, pp: 133-142.

Yi, C. and K. Nahrstedt, 2001. QoS-aware dependency management for component-based systems. Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing, August 07- 09, 2001, San Francisco, USA, pp: 127-138.