Global Journal of Computer Science and Technology
Volume 12 Issue 3 Version 1.0 February 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Effort Distribution in COTS Components Integration: A Simulation Based Approach

By P K Suri & Sandeep Kumar
Galaxy Global Group of Institutions, Dinarpur, Ambala, Haryana

Abstract - Component Based Development lays emphasis upon composing software from pre-existing commercially off the shelf (COTS) components. Component repositories are searched for the existing components according to requirement specifications and then components are integrated in the system. Though all the components are important for the success of a Component Based Software, some of them may be more important than others. While distributing the cost, efforts, time and other resources, starting from component requirement specification to component integration, we need to differentiate between more and somewhat less important components and distribute the resources accordingly. In this paper we have developed a simulator for identifying the critical components in a component based system for optimum distribution of the resources while integrating the components in the system. This simulator can be used to plan the distribution of available resources in a better way. This will help to overcome the problems of cost and time overrun while integrating and deploying components in a Component Based System (CBS).

Keywords : *Component Based Software, COTS, Simulation, Component Integration and Deployment, Erlang*

GJCST Classification: *I.6 , I.6.8*

Strictly as per the compliance and regulations of:

Effort Distribution in COTS Components Integration: A Simulation Based Approach

P K Suri [α] & Sandeep Kumar [σ]

Abstract - Component Based Development lays emphasis upon composing software from pre-existing commercially off the shelf (COTS) components. Component repositories are searched for the existing components according to requirement specifications and then components are integrated in the system. Though all the components are important for the success of a Component Based Software, some of them may be more important than others. While distributing the cost, efforts, time and other resources, starting from component requirement specification to component integration, we need to differentiate between more and somewhat less important components and distribute the resources accordingly. In this paper we have developed a simulator for identifying the critical components in a component based system for optimum distribution of the resources while integrating the components in the system. This simulator can be used to plan the distribution of available resources in a better way. This will help to overcome the problems of cost and time overrun while integrating and deploying components in a Component Based System (CBS).

Keywords : Component Based Software, COTS, Simulation, Component Integration and Deployment, Erlang

I. INTRODUCTION

Although Component Based Software Engineering is new paradigm in software engineering, it may be likened to traditional engineering branches like civil or mechanical engineering, where emphasis is not laid on developing as such; rather it is on designing and composing. Almost similar approach is followed by the Component based software engineering where instead of developing the application from scratch, pre-developed and pre-tested black box components are integratedtogether to compose a new software. Due to this reasonthese black box components are also known as Commercially–Off-The-Shelf (COTS) components [1]. This is so because they are developed by someone else and used by someone else. Every component has some clearly defined interfaces.

Through these interfaces it takes services from other components and provides services to other components. This give and take of the service by components is called interaction [2]among components. But before the components can interact with each other, they must be integrated together. Process of integrating the components in a component based system is not as easy and straightforward as it seems to be. Existing components may be part of some other applications. From there they are collected into component repositories. Depending upon the requirements of the current application, component repositories are searched for the required components. Different components may be found in different repositories. Then these components are integrated together to compose a new application. This process is shown in fig.1.

So developing Component Based Software does not mean developing everything afresh. One thing that is very important here is that sometimes it may happen that no component satisfying the user requirements is found in any of the component repositories; in that case we may have to develop a new component. Although developing a new component follows the usual procedure of developing any software module, here wewill assume that we don't need to develop any new components, rather we have all the required components in one or the other repositories and we only need to identify and search them according to our requirements and then integrate them in the system. Finding a component that meets the user requirements in itself is a tedious task and involves many activities. Composing existing components to form a new application follow certain predefined procedure. This procedure consists of many steps [3], [4], [5], [6]. Although different researchers have given different models of composing component based software, there are certain things where almost all of them agree upon.

There are following six broad activities that are essential for component integration:

a) Component Requirement Specification
b) Component Requirement Review
c) Component Identification and Selection
d) Component Adaptation
e) Component Integration
f) Component Deployment

A brief description of these activities is given below.

a) Component Requirement Specification

This isconcerned with providing a formal

Author [α] : Dean, Research and Development; Chairman CSE/IT/MCA, HCTM Technical Campus, Kaithal, Haryana, India.-136034.
E-mail : pksuritf25@yahoo.com.
Author [σ] : Assistant Professor and Head, Faculty of Computer Applications, Galaxy Global Group of Institutions, Dinarpur, Ambala, Haryana-133207, India E-mail : sandeepnain77@gmail.com

documented consensus regarding the requirements, scope and boundaries of a software component [15]. It is important to document customer's vision of the component after analyzing the customer requirements. Depending upon the customer's requirements, an existing component may be reused or may be adapted deployment review the requirements thoroughly[14]. In this phase all the component related requirements are reviewed by stakeholders.

c) Component Identification and Selection

Success of a Component Based Software depends a lot upon our ability to select a suitable component according to user requirements [8], [9], [10], [11]. This selection and identification process involves four steps [4]: Search, Screen, Evaluate and for reuse. In the worst case a new component may have to be developed. This is a very important phase

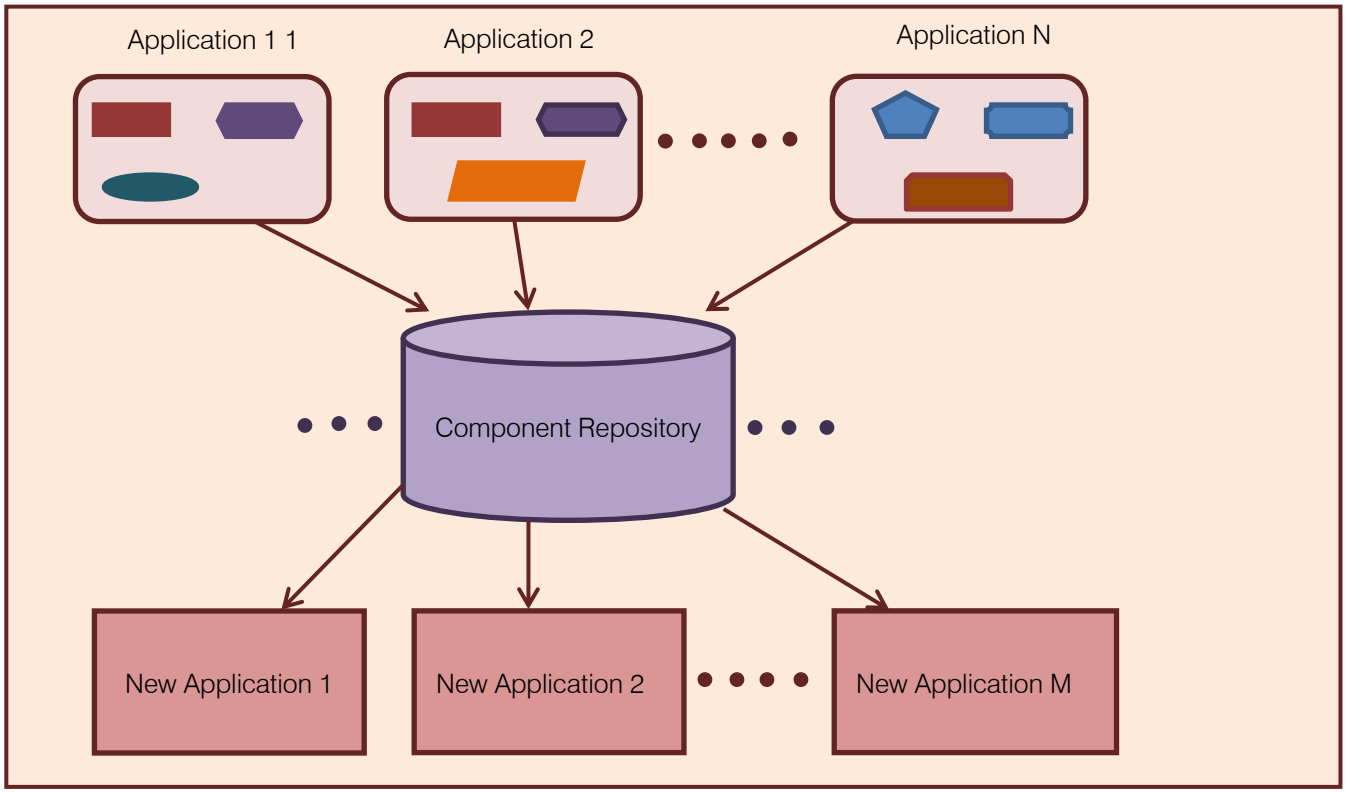


Fig. 1. Storage and Searching of COTS Components

because quality of the component selected and ability of the component to perform a specified task depends upon how well the requirements have been understood and documented.

b) Component Requirement Review

Every software component has a clearly specified functionality. Conceptual Design of a component and its functional requirements are specified in component requirement specification document. It is very important that requirements specified in this document are complete, adequate, without any ambiguity, feasible to implement and consistent with the intended component. To avoid any problems related with these attributes, it is important that all the parties' involved in component selection, integration and Analyse. The search process may give a list of many candidate components and it becomes very important to select best suited component. One such technique of component selection is given by Suri et al in [12].

d) Component Adaptation

All the components that have been selected using the Component integration and selection process may have been developed using different platforms, by different teams of people for different applications. So it is very important that component services are provided through a standard published interface so that components are able to interoperate. Practically it is quite difficult to find a software component that can fulfill the hundred per-cent user requirements. So it becomes very important to adapt the component for the present application. Rine et al [13]have proposed the use of adapters to adapt the components according to present application.

e) Component Integration

Next job is to integrate the components so that they can interact with each other. Vigder [16]identified three important components of the component integration. These are: Wrappers, Glue and Component

Tailoring. Through wrappers, underlying components are isolated from other components in the system. Glue provides the functionality to combine the components and through Component Tailoring, functional ability of a component is enhanced. Testing and validation are also part of the component integration process. Components may have been tested earlier but it is very important to test them for the present application too. Generally internal structure of a component is not known so most appropriate for testing the components is Black Box Testing[1],[17], [18]. Also it is very difficult to test all the components when number of components in the system is very large. This search space can be reduced using a technique proposed by Suri and Kumar[19].

f) Component Deployment

According toNing[20], deployment involves packaging components so that they can connect , disconnect and reconnect at runtime. A tool called packager is proposed for the purpose.

II. Problem Statement

Each component based system may consist of hundreds to thousands of components. These components are developed using different technologies on different platforms. Depending upon the user requirements, component libraries are searched for the suitable components. Once suitable components are found, they are integrated or composed to create the system. While integrating these components in a Component Based System (CBS), we need to put in efforts and resources in each step or activity that leads to component integration. All components of a CBS are not equally important. So that the available resources, time, efforts can be utilized optimally, it is important that a plan of the schedule of component integration be prepared. This plan must contain the details of the efforts, may be in person-hours, to be put in various component integration activities. There are certain components that are heavily loaded as compared to other components because they need to be accessed more frequently for getting some job done. For example the initial component, that provides an interface to the user for the input, is accessed every time user has to work on the system. Similarly, the component that get output to the user is also accessed each time user needs an output. There may be other components in the systems that are not accessed so frequently. It is very important to identify these critical components so that they can be allotted more time and efforts. Identifying such components is the main theme of this paper.

III. Proposed Simulation Model

Here a Component Based System is represented with the help of an activity network. The network consists of nodes and edges. Nodes in the network represent the individual components of the system. Edges represent the flow of execution between various components. An edge from component C_i to component C_j represents an interface link ('provide' or 'gets' interface) from C_i to C_j. Many components in sequence starting from the first component and terminating into the last component make an interfaces path. To achieve an artifact result, execution starts from the first component that provides a user interface. The control keeps transferring from one component to another, through interface links. When a component is integrated in the CBS, six activities: *Component Requirement Specification, Component Requirement Review, Component identification and selection, Component Adaptation, Component Integration and Component Deployment* are to be performed, in that order. These activities are stochastic in nature because time taken by each of these activities and efforts required in fulfilling these activities are distributed exponentially. If we assume that on average, each of these six activities take constant efforts (say β), then effort required for integrating and deploying a component is governed by Erlang-6 distribution [24]. Because if there are k independent stochastic random variables v_1, v_2, v_3,.....v_k, having same exponential distribution.

$f(v_i) = \mu k e^{-\mu k v}$ given that $v_i > 0$; $\mu > 0$ and k is a positive integer, then $V = \sum v_i$ is governed by Erlang distribution.

So all six stochastic variables so obtained are composed using above formula and then weight so obtained is assigned to the corresponding component. Weight assigned to a component is distributed equally among all interface links terminating into that component node. This weight computed is governed by Erlang-6 distribution. Because efforts required for integrating each component are stochastic and not deterministic in nature, it will be erroneous to assume single effort estimate for the each component's integration. Due to stochastic nature of the component integration efforts, we take three types of effort estimates as

- Minimum Effort Estimate (E_{min}): The minimum possible efforts required for integrating the component. We will require minimum efforts if everything goes well, there are no employee switchover, and no new requirements, no conflicting requirements and we are in an ideal situation.
- Maximum Effort Estimate (E_{max}): The maximum possible efforts required for integrating the component. This is the effort requirement when everything goes bad.
- Normal Effort Estimate (E_{nor}): This is the effort required if the component is integrated under normal circumstances. All practical problems that may arise have been considered.

Taking these three types of efforts into

consideration following may hold true for Mean Efforts Required (μ) and Standard Deviation (σ)

$$\text{Mean of Efforts } (\mu) = (E_{min} + 4*E_{nor} + E_{max})/6$$
$$\text{Standard Deviation } (\sigma) = (E_{max} - E_{min})/6$$
$$\text{Variance} = (\sigma)^2$$

IV. Why Simulation For This Problem

Each activity of component integration process is stochastic in nature. As we have assumed that efforts required in performing each activity follow beta distribution, it is possible to identify the critical components along one interface path. But sometimes results given by this process may be wrong. In many cases we can assume that efforts required are available in the form of a discrete or continuous frequency distribution. In some cases in addition to the components along critical interface path there may be other components that are near critical and are also important and need a fair share of the time and efforts. In this case total efforts required along near critical path may be slightly less than the critical path, but quite possible that may have happened because the variance along the near critical path is slightly more than the critical path. So it is important that near critical path is also tested because if we run the simulator for many number of times, then near critical components may also sometimes become critical. This is why simulation as a tool has been used for identifying the critical components.

V. Assumptions

Following assumptions have been made for the proposed simulation model :

- Each node of the network represents a component.
- Directed edge from component C_i to component C_j means that a 'provides' and/or 'gets' relationship exist between these two components.
- Integrating each component involve six phases (component requirement specification, component requirement review, component identification and selection, Component Adaptation, Component Integration, Component Deployment).
- Effort required in carrying out each of these phases is constant and exponentially distributed.
- Effort required in integrating and deploying each component is governed by Erlang-6 distribution.
- All the components (nodes) are assigned numbers in topological order according to Fulkerson's i-j rule[23].
- Any execution in the CBS starts with the first component that provides an interface to the user and terminates with nth interface that provides the final output interface. In between many components are accessed along different interface paths.
- Each component is assigned a weight which is effort required in integrating corresponding component.
- All the interface links terminating into a component node are assigned the equal weight. This weight is equal to the weight assigned to the component towards which these links are directed/ number of components.
- Total weight W is the sum of all link weights along a path and represents the total efforts required in integrating all the components along that path.
- Path with maximum total weight is a critical path and all the components that fall along that path are critical components. All the interface links along this path are also critical interface links.

VI. Notations

ORIGIN[ ]:		Array containing originating component number of all the execution links.
TERMINAL[ ]:		Array containing terminating component number of all the execution links.
S	:	Starting or First Component of the CBS.
F	:	Finishing or Last component of the CBS
ORIGIN[i]:		Originating (tail end) Component of link i.
TERMINAL[i]	:	Terminating (head end) Component of link i.
WT[i]	:	Array containing weights assigned to all the interface links.
LSW[i]	:	Least cumulative starting weight of link i.
LTW[i]	:	Least cumulative terminating weight of link i
MSW[i]	:	Most cumulative starting weight of link i.
MTW[i]	:	Most cumulative terminating weight of link i.
MinCW[j]	:	Minimum weight that can be assigned to component j.
MaxCW[j]	:	Maximum weight that can be assigned to component j.
M	:	Number of components in the CBS.
N	:	Number of interface links in the system.
Crit_M[j]:		Criticality index of the jth component
Crit_N[i] :		Criticality index of the ithinterface link.

VII. Algorithm Description

One of the most important question that arises during the effort and resource allocation is that which are the components and interfaces in the system that are most important for the overall working of the system, so that they can be allocated efforts and resources in bulk. These are the components and interfaces that fall along the path that has got the maximum weight of all possible paths. This path is called the critical path and components and interfaces along that path are called critical components and critical links respectively. Pseudo code below in this section describes the process to find the critical components and interfaces. In this process first we move in the forward direction in the network of components. While moving in the forward direction we compute the least cumulative terminating weights of all the links by adding link weights to their least cumulative starting weights. At the end of the forward process we compute the minimum weight that can be assigned to the Mth component (output interface component). Next we traverse the network in backward direction finding most cumulative starting weight of each link by using most cumulative terminating weight and link weights. The procedure is describes as follows:

1. Initially assign weights to all the interfaces or execution links. These weights are generated using a random number generator. Samples so generated follow Erlang-6 distribution. Theseweights are stored in the array WT[i], for i = I to N.
2. Traverse the component network in forward direction.

a. Set the minimum component weight MinCW[j] for all components (j = 1 to M) to zero.
b. Each component node may have many execution links terminating into it. Once all the execution links enter into the present component, compute the minimum component weight. This is equal to the maximum of the weights of all the execution links entering into that component node. Call it MinCW[j] for the jth component. By definition MinCW[1] is zero.

This process is repeated for all combinations of links and components and finally minimum component weight of last component i.e. Mth component is computed. Call it MinCW[M]. Assign this to W (this is sum of weights of all components along that path).

3. Traverse the component network in backward direction.

a. Assign the minimum component weight computed in step 2 to the last component of the network, call it MaxCW[M] = W. This weight is also assigned as maximum terminating cumulative weight of all links terminating into Mth component i.e. MTW[all execution links terminating into M] = MaxCW[M]. Moving further maximum starting weight of link N is computed as MSW[N] = MTW[N] – WT[M].
b. Moving further backwards, maximum starting cumulative weight of each execution link is computed as MSW[i] = MTW[i] – WT[i].
c. Next, maximum weight that can be assigned to a component 'j' is computed. Call it MaxCW[j]
d. Compute maximum terminating weight MTW of all the links starting from component j. They are assigned the value MaxCW[j].

All the components for which MaxCW = MinCW, are critical components and form a critical path. Similarly all the interface links that are part of this critical path are critical interface links.This algorithm is repeated many times, each time for a different set of random weights assigned to the execution links. This is done to accommodate the error element, E. Due to this error sometimes, components that may have remained near critical in some previous run, may become critical in some other simulation runs. This way we can find out how many times a particular component and interface link becomes critical. More number of times a component becomes critical, more efforts we need to put in while integrating this component in the system to safeguard our system from failure. Same is true for interface links too. Formally the algorithm is described in fig.2.

VIII. Case Studies

a) Case Study 1

We experimented with the simulator developed in C Language over Window 7.0 Platform using DosBox0.73. Two case studies for experiment were conducted. In the first case study, a CBS consisting of 8 components was considered. The execution flow through interface links of the CBS is shown in figure 3. There are a total of 11 interface links and 8 components in this system. Each component in the system is integrated using six steps described above. Each interface link connects two components. At the tail end of the interface is the originating component and at the head end is terminating component. The details about originating components, terminating components and criticality indices of each interface link are given in table 1. Table 2 contains the information about criticality indices of the components. This information is also depicted in graphs in fig. 4 and fig. 5 respectively.

b) Case Study 2

For the second case study, we have taken a CBS with eight component nodes and 13 interface links. The graphic representation of the system is shown in fig 5. Table 3 contains information about the originating and terminating component of each interface link and also the criticality indices of the interface links after the simulation run are performed 1000 times. Table 4 contains the criticality indices of different components

for 1000 simulation runs. Criticality indices of interface links and components are also depicted in fig. 6 and fig.7 respectively.

1. Input the values for RUNS, N, M, E and Populate arrays ORIGIN[] and TERMINAL[] accordingly.
2. for i = 1 to N (Set Crit_E[i] = 0)
3. for j = 1 to M (Set Crit_E[j] = 0)
4. for x =1 to RUNS repeat steps 5 to 9 in step of 1.
5. fori = 1 to N (Generate random variants and store them in array WT[i]).
6. Start forward pass
    a. Set MinCW[1] = 0
    b. Compute MinCW for all component nodes as follows
        i. LSW[i] = MinCW of the node at tail end
        ii. LTW[i] = LSW[i] + WT[i]
        iii. MinCW[j] = max{LTW(all interface links terminating into component node j)}
7. Start Backward Pass
    a. Assume MaxCW[M] = MinCW[M] (MinCW[M] was computed in forward pass).
    b. fori = I to N (MTW [i] = MaxCW of the node at head end).
    c. MSW[i] = MTW[i] – WT[i]
    d. Compute MaxCW for all component nodes (except last node, M) as follows

        MaxCW[j] = min {MSW(all links originating from component j)}
8. Update Criticality index of interface links

    If (MSW[i] – LSW[i] <= E (Increment Cri_E[i] by 1)
9. Update Criticality indes of Component node

    If (MaxCW[j] – MinCW[j] <= E (Increment Cri_C[j]] by 1)
10. for I = I to N (Print Crit_E[i]).
11. for j = I to M (Print Crit_C[j].
12. Stop.

Fig.2. Algorithm for finding Critical components and Interface Links

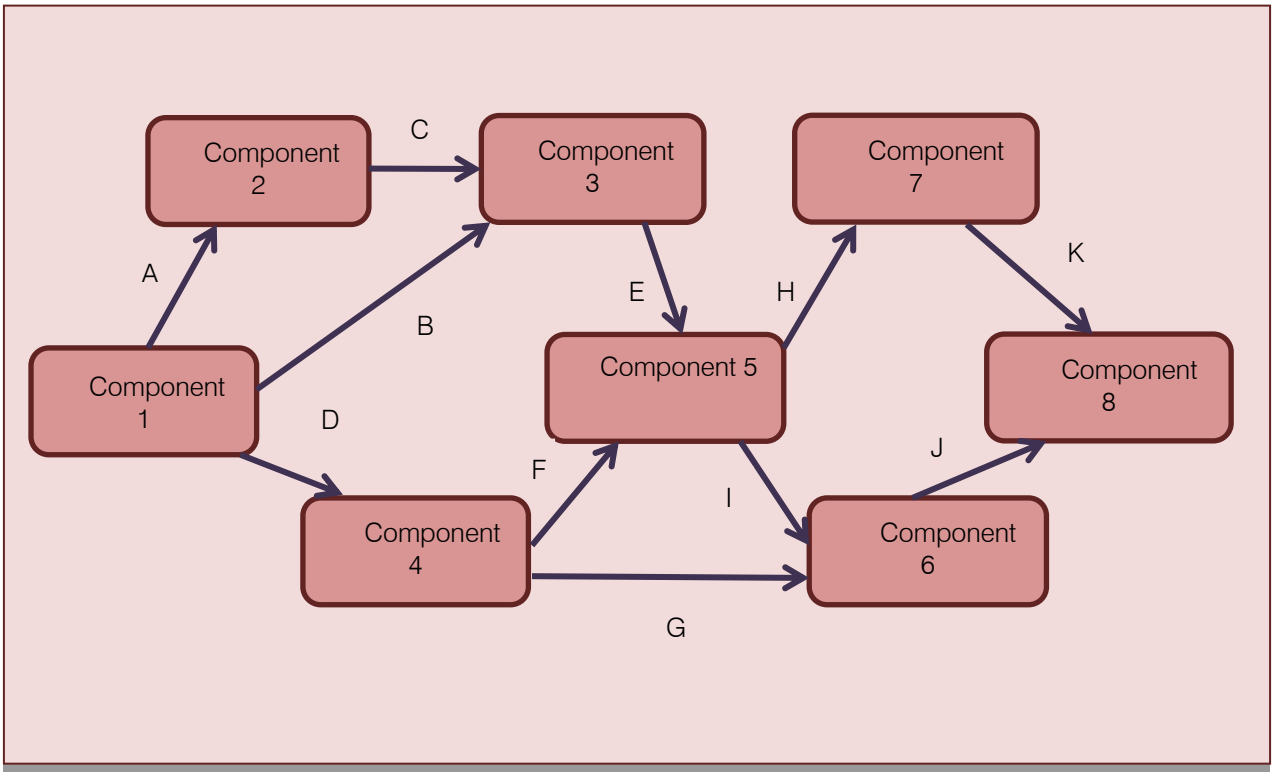


Fig.3. CBS Component Network 1

Interface Link	Originating Component	Terminating Component	Mean Weigh (Efforts)	Standard Deviation	Criticality Index
A	1	2	$\mu 1$	σ 1	.912
B	1	3	$\mu 2$	σ 2	.260
C	2	3	$\mu 3$	σ 3	.610
D	1	4	$\mu 4$	σ 4	.912
E	3	5	$\mu 5$	σ 5	.958
F	4	5	$\mu 6$	σ 6	.610
G	4	6	$\mu 7$	σ 7	.000
H	5	7	$\mu 8$	σ 8	.502
I	5	6	$\mu 9$	σ 9	.499
J	6	8	$\mu 10$	σ 10	.500
K	7	8	$\mu 11$	σ 11	.502

Table1. Originating Component, Terminating Component and Criticality Indices of Interface Links.

Component	Criticality Index
1	1.0
2	.912
3	.938
4	.061
5	1.0
6	.500
7	.629
8	1.0

Table 2. Component Criticality Indices

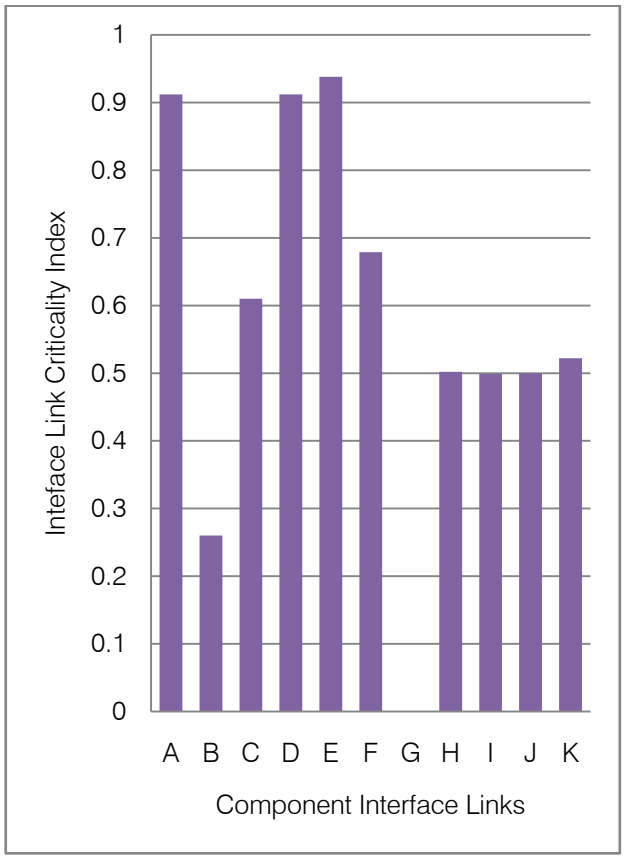


Fig.4. Graph of Criticality Indices of Interface Links

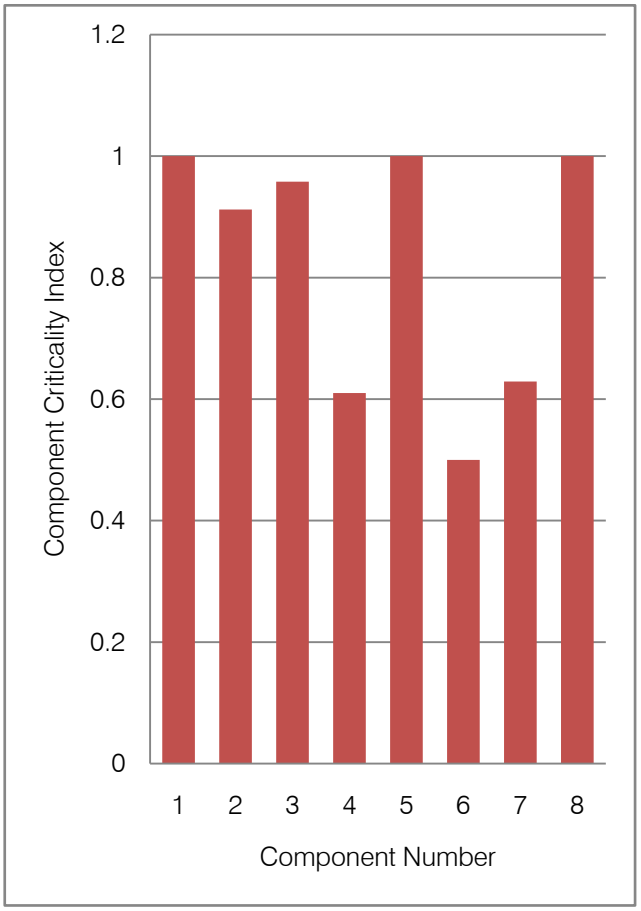


Fig.5. Graph of Criticality Indices of Components

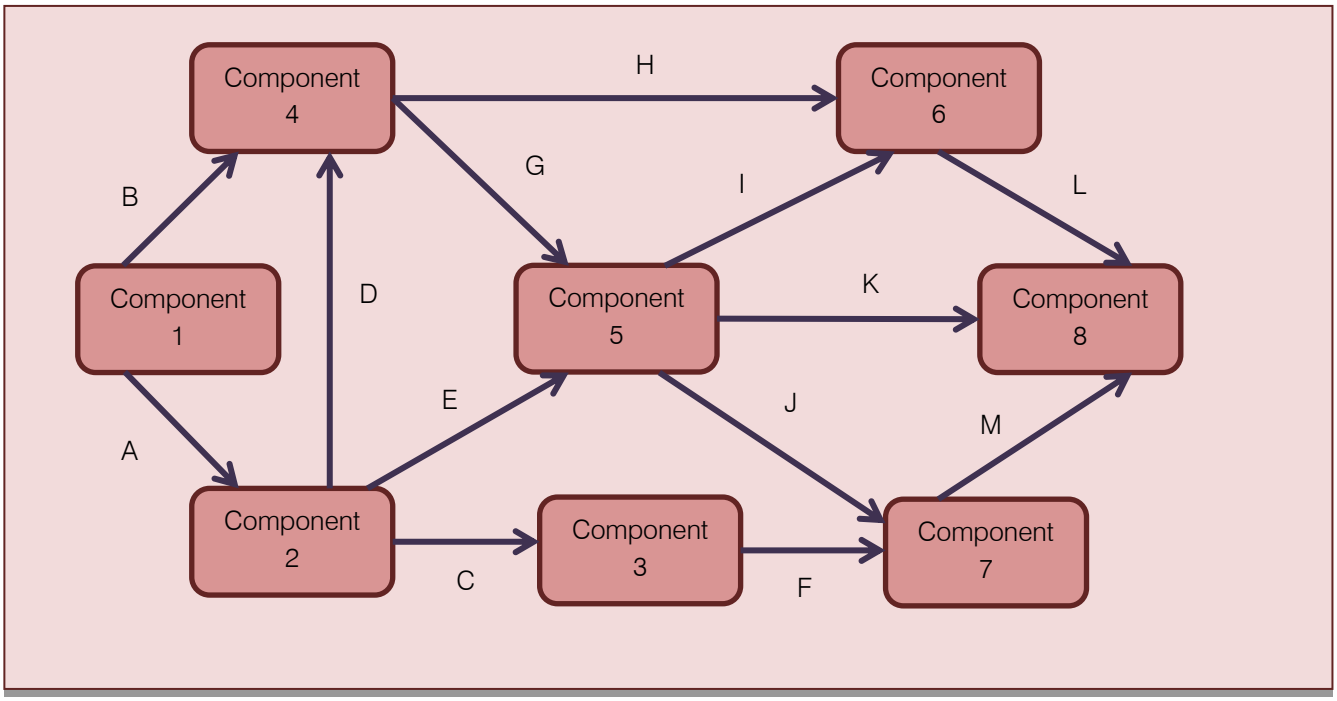


Fig.6. CBS Component Network 2

Interface Link	Originating Component	Terminating Component	Mean Weigh (Efforts)	Standard Deviation	Criticality Index
A	1	2	$\mu 1$	σ 1	.970
B	1	4	$\mu 2$	σ 2	.026
C	2	3	$\mu 3$	σ 3	.031
D	2	4	$\mu 4$	σ 4	.918
E	2	5	$\mu 5$	σ 5	.021
F	3	7	$\mu 6$	σ 6	.031
G	4	5	$\mu 7$	σ 7	.940
H	4	6	$\mu 8$	σ 8	.004
I	5	6	$\mu 9$	σ 9	.469
J	5	7	$\mu 10$	σ 10	.491
K	5	8	$\mu 11$	σ 11	.004
L	6	8	$\mu 12$	σ 12	.474
M	7	8	$\mu 13$	σ 13	.523

Table 3. Originating Component, Terminating Component and Criticality Indices of Interface Links.

Component	Criticality Index
1	1.0
2	.970
3	.031
4	.944
5	.964
6	.474
7	.523
8	1.0

Table 4. Component Criticality Indices

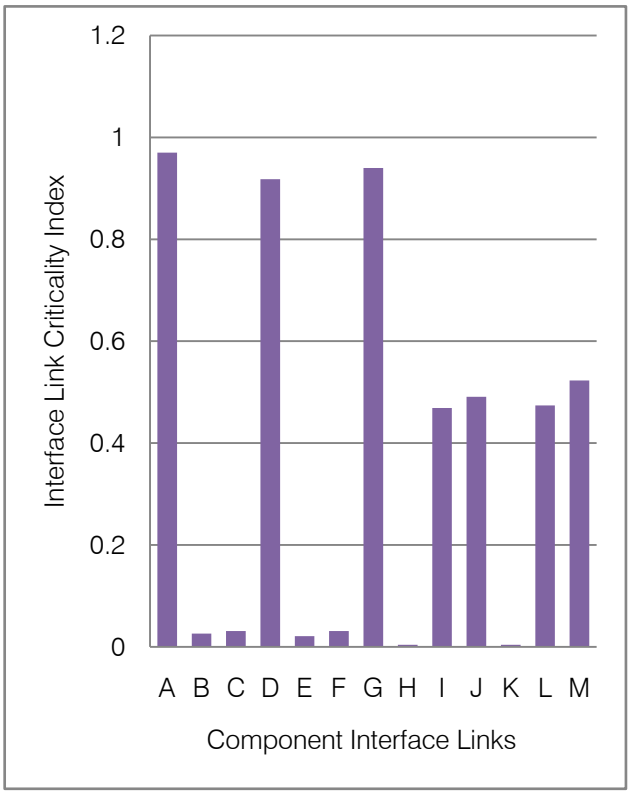
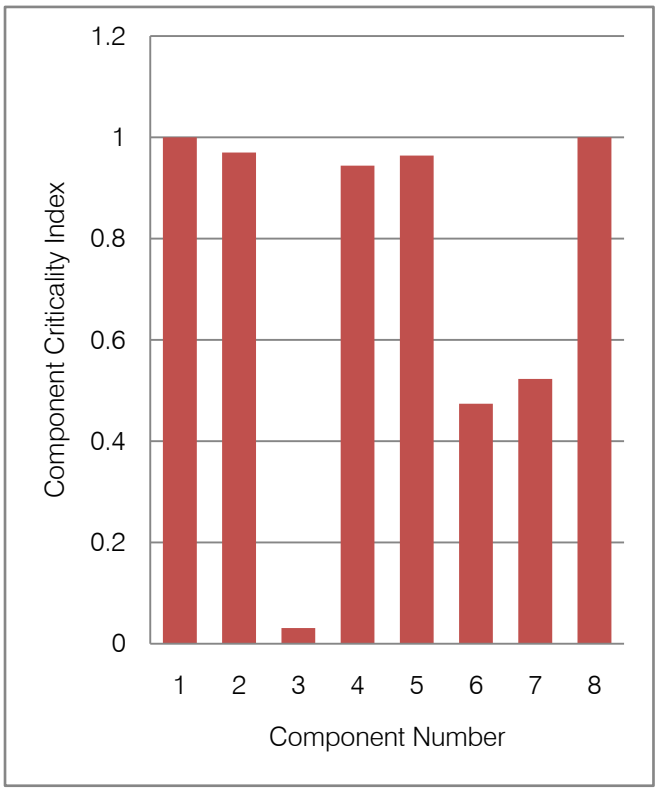


Fig.7. Graph of Criticality Indices of Interface Links

Fig.8. Graph of Criticality Indices of Components



IX. Discussion And Conclusion

From above discussed case studies 1 and 2 important conclusions were drawn. In the first case study it was found that interface links A, D and E are the most critical interface links. These links provide interfaces between components 1-2, 1-4 and 3-5 respectively. Links C (2-3) and F (4-5) are also important but not as important as A,D and E. Interface Link G (4-5) is the least critical link. Out of 8 Components of the CBS Components 1,2,3,5 and 8 are most critical and we should allocate most of the resources and efforts while integrating and deploying these components in the system. Similarly in case study 2, we found that interface links A (1-2), D (2-4) and G (4-5) are the most critical ones and H (4-6) and K (5-8) are the least critical. As far as components are concerned, components 1,2,4,5,8 are the most important for the overall success of the system and a good number of efforts must be put in while integrating them.

So as we saw this simulator can be a handy tool for the project team that has to decide on how much efforts, time and cost should be put in while specification and review of requirements and identification, selection, adaptation, integration and deployment of the different components in a component bases system.

References Références Referencias

1. Korel, B. (1999) Black Box Understanding of COTS Components. *IEEE Xplore Digital Library*. 92-99.
2. Councill, B.&Heinman, G. (2001) *Component Based Software Engineering: Putting the pieces together*. NY: Addison Wesley. 5-99.
3. Cai, X., Wong, K., Ko, R. &Lyu, M. (2000) Component Based Software Engineering: Technologies, Development Frameworks and Quality AssuranceSchemes. *IEEE Xplore Digital Library*. 372-379.
4. Mahmood, S. & La, R. (2007) Survey of Component Based Software Development. *IET Soft*. 1 (2) 57-66.
5. Crnkovic, .I (2003) Component Based Software Engineering: New Challenges in Software Development..*IEEE Xplore, 25th Int. Conf. on Information Technology Interfaces*. 9-18.
6. Crnkovic, I., Chaudron, M. & Larson, S. (2006) Component Based Development Process and Component Lifecycle. *Proc. of IEEE Int. Conf. on Software Engineering Advances*.
7. Maiden, N.&Ncube, C. (1998) Acquiring COTS Software Selection Requirements. *IEEE Soft.* 15 (2) 46-56.
8. Leung, K.& Leung, H. (2002) On the Efficiency of Domain Based COTS Product Selection Method. *Info. Soft. Tech.*44 (12) 703-715
9. Alves, C.& Finkelstein, A. (2003) Investigating Conflicts in COTS Decision Making. *International Journal of Software Engineering Knowledge Engineering*. 13. 1-21.
10. Chung, L.&Kooper, K. (2002) Knowledge based COTS aware Requirements Engineering Approach. *Proceedingsof 14th Int. Conf. on Software Eng. (ACM Press)*. 175-182.
11. Sommervile, I (2005) Integrated Requirement Engineering: A Tutorial. *IEEE Soft.* 22 (1) 16-23.
12. Suri, P., Kumar, S. & Singh, G. (2011) Precision in Rapid Application Development and Reusability of Software Components for Greater Performance using Ranking Mechanism. *International Journal of Computer Applications, NY*. 35 (8) 21-27
13. Rine, D.& Nada, N. (1999) Using Adapters to Reduce Interaction Complexity in Reusable Component Based Software Development. *Proceedings of Symp. on Software Reusability (ACM Press)*. 37-43
14. Mazzza, C (1995) Guide to the Software Requirements Definition Phase. *ESA PSS*. 1 (1).
15. Dixit, A &Saxena, P (2011) Umbrella: A new Component Based Software Development Model. *International Conference on Computer Engineering and Applications, Singapore. (IASCIT Press)*.
16. Vidger, M.& Dean, J. (1997) AnArchitectural Approach to Building Systems from COTS Software Components. *Proc. of conference of Advanced Studies of Collaborative Research (IBM Press)*.
17. Gao , J.& Gupta, K. (2002) On Building Testable Software Components. *Proc. of 1st Int. Conf. on COTS Based Software Systems, Springer-Verlag*. 108-121.
18. Muller, C &Korel, B (2001) Automated Black Box Evaluation of COTS Components with Multiple Interfaces. *Proceedings of 2nd Int. workshop on Automated Programming, Analysis, Testing and Verification, ICSE*.
19. Suri, P.& Kumar, S. (2010) Simulator for Identifying Critical Components for Testing in a Component Based Software System..*International Journal of Computer Science and Network Security, Korea.*10 (6) 250-257.
20. Ning, J. (1997) Component Based Software Engineering (CBSE). *Proc. of 5th International Symposium on Assessment of Software Tools and Technologies,* 34-43.
21. Meling, R.& Wong, E. (2000) Storing and Retrieving Software Components: A Component Description Manager.*Proc. of Australian Software Eng. Conference (IEEE Press).*107-117.
22. Smith, R. (1998) *Effort Estimation in Component Based Software Development: Identifying Parameters.* SIGCSE Doctoral Consortium, Available: http://www.cs.ulexas.edu.
23. Wills, R. (1981) A Note on the Generation of Project Network Diagram. *Operation Research Society*. 32. 235-238
24. Deo, N (2009). *System Simulation with Digital Computer*. New Delhi: PHI Learning Pvt. Ltd.

This page is intentionally left blank