Simulator for Identifying Critical Components for Testing in a Component Based Software System.

P K Suri[1], Sandeep Kumar[2],

[1] Dean Science Faculty and Professor, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra (Haryana), India
[2] Assistant Professor & Head, Department of Computer Applications, Dronacharya Institute of Management and Technology, Kurukshetra (Haryana), India.

Summary
Component Based Development is the buzz word in software industry. Lot of efforts are being put in by researchers, academicians and industry professionals in making the CBSE as the default technology of software development.. But very little work has been done in the field of Testing Components and component based systems. Any component based system is composed of many components. While integrating these components it is not possible to test each and every component. So there is a need to identify the key components for testing. A simulator has been designed here to identify the key components (that are most important for the working of a component based system) so that most of the efforts and resources can be put in to test these critical components. A Component Execution Graph (CEG) is the basis for developing such a simulator.

Keywords
Component, Component Based software, Testing, Component Execution Graph (CEG), Simulation, Erlang

1. Introduction

Composability is the main aspect of the Component Based Software Engineering. Instead of developing the system from scratch software components (COTS) are purchased from the market and then composed after adapting them according to requirements. Besides this there are components that are developed in house for some other projects and can be reused in the current project as it is and sometimes they may be upgraded for reuse in current project. If the components are not available Off the Shelf, neither have they been developed in house for some other projects, then, we may need to develop them. They are developed in such a way that their reusability aspect is not compromised so that they can later be put into libraries of reusable component. As the Software Component Technology is evolving, it is becoming clearer that the quality of a component based product depends upon the quality of software components and the effectiveness of the process that is used to test the component based software [18]. Challenges related with component testability in the form of component traceability& observability, component controllability and component understandability have been listed by Gao [6]. There is always a strong possibility that the components, that are composed together to make a new system, have been developed using different languages on different platforms. Among these, some would have been purchased off the shelf and others developed in house. Main advantage of such type of paradigm is the rapid development and savings in the form of resources, efforts, cost, time etc. But it also gives rise to many problems. Any product developed using Component based technology consists of hundreds of components. If any one of these components is of poor quality, that may effect the quality of the overall system. Problems may be more severe if key components are not tested properly. But identifying these key or critical components of a system is a very challenging task. Testing of these critical components should not be compromised at any cost. This paper deals with this challenge. Basis for identifying the most critical components in the system is a graph called Component Execution Graph (CEG). CEG is a network representation of a Component Based System. Each node of the CEG represents a Component in the system and an edge from node i to node j represents the transfer of execution control from component i to component j. Each execution starts at first component of the CEG and finishes at the last component of the CEG. Before we can proceed further and discuss the simulation model, it is very important to discuss some of the problems associated with existing testing techniques as far as their relevance to the component based technology is concerned. Wu [1] has listed following issues related with testing of component based software.

Heterogeneity: As components are taken from heterogeneous environments, it becomes very difficult to integrate them and achieve desired results. Each programming language has its own syntax, data processing procedure, and way of using the data structures. Component may have been developed on a particular machine with a particular architecture and having a particular operating system environment. Although components may have been tested in their respective

Manuscript received June 5, 2010
Manuscript revised June 20, 2010

environments for quality, it becomes very necessary to test them in new integrated environment.

Non Availability of the source code: Different COTS components are developed by different vendors and generally source code of the COTS components is not provided with it. It makes the task of testers more difficult while integrating the components for a new application.

Evolvability: Components keep evolving with time according to customer needs and evolving industry standards. Each time some change is introduced it also results in introduction of new types of errors and bugs.

Besides these problems Gao [18] has also mentioned many problems associated with testing of Component Based Software.

- Problems in testing Software Components
- Problems in component integration
- Problems in system testing Component Based Software.

Since a long time, researchers and software testing tool players have been developing many white box as well as black box test methods for the traditional paradigms [19], but from these problems associated with the Components, it is clear that testing the components based software is not similar to that of testing other types of software, developed using other traditional methods. Many references could be found in literature to prove this fact.

2. Related Work

Jerry Gao[17,18,19], in a series of articles on Testing component based software, proposed a model to measure the maturity levels of a component testing process. Issues related to software components and Component based software testing have been identified and classified. It discusses component testability in terms of controllability, traceability, test suit, presentation, test support, and configuration management.

While integrating components from heterogeneous environments, it is not possible to test each and every component. One solution is to test potentially risky component. Now the question arises how to select such type of component. No AdHoc arrangements can be made for such selection. According to McGregor [21] we should select a component for testing when penalty for component not working is greater than the effort required to test it. Author uses application of a risk analysis technique to the task of identifying which components to be tested more intensely than rest. Author conducted an analysis on the requirements to determine the potential business and technical risks for the development process. Using this analysis risks identified at the requirements level were mapped onto the various components. All components were classified according to three risk categories (Low, Medium and High) and components falling in one category were tested at the same coverage level. But exact quantification of the risks associated with each component is not possible using this technique and it fails to give an account of number of most critical components that need to be tested.

According to Wu [1] lot of work has been done in the field of component based development; still there are very few techniques available for the testing of component based software. Author has also presented a test model and suggested some key test elements for the component based software. The base of the research work is the interaction and dependence among components. Artifact is a test adequacy criterion that results in optimization of budget, schedule and quality requirements. Although much work has been done for testing of object orients systems [9,12,14,15,16], very few researchers have extending this work to cover the testing of component based software although, this can be an interesting research area [2] . Rosemblum [22] has extended the techniques of object oriented software and proposed a model for adequate testing of the component based software. Harrold et al. [13] have proposed a testing technique that is based on analysis of component based systems. But this technique uses the source code of the components provided by the component vendors. But there are very few vendors who will provide the source code of the components.

C. Mao and Y. Lu [3] have again described non availability of the details about components as a major bottleneck in testing component based software. They have analyzed the shortcomings of some existing regression testing techniques for component based systems [8,10] proposed a regression testing method for systems composed of modified components. But this method requires the constant interaction among component developers and component testers.

In [4] Byoun et al. used the state transition model for generation of test cases for interoperability test of Component Based Systems. Main emphasis is on checking the interoperability among various components of a system.

Some of the problems related with testing component based systems may be scaled down by selecting proper components among many candidates. But that in itself is a challenging task. According to [5] there is no existing effective technique available that can quickly check the various alternatives and focus on a subset of likely compatible components. Further authors have given a technique based on regression testing that uses a behavioral model to represent interaction among components and automatically generates and prioritize test suits that test the compatibility among various components of a system. Another author to make use of regression testing for component based systems is Mao [11]. This paper uses a built in test design. Test interfaces

are constructed after analyzing the effected methods in the new component version by the component developers and then components users pick-out the subset of test cases for regression testing with these testing interfaces. This method again requires a continuous interaction among component developers and component users. Y Wu [1] has suggested use of static and dynamic analysis to guide test case generation. Integration among components is used to determine what needs to be tested. This is done using static analysis. During the process, interfaces that are invoked and events that are triggered during each execution are kept track by dynamic analysis. This information is then used to determine the test adequacy.

3. CEG (Component Execution Graph): A representation of Component Based System

As is clear from above discussion testing component based systems is a challenging task. It is challenging in the sense that pre-tested components are composed together to make a new application but they may have to be tested again when they become part of a different environment. It is not possible to test all the components of a system if the system is composed of hundreds of components. The main challenge is to identify the components that are critical for the overall working of the system. Then critical components can be tested more rigorously and thoroughly as compared to other components. For identifying these critical components, we make use of Component Execution Graph. Each component based system (CBS) can be represented with the help of a Component Execution Graph. It is a network representation of the CBS. CEG consists of edges and nodes. One node represents one independent component and an edge from node i to node j represents an execution link from component i to component j. Through each execution link, execution control is transferred from component "i" to component "j". To achieve one meaningful output or result, at least one path, starting at the first node of the CFG and terminating in the last node of CEG, must be executed. In between it may take any courses of execution, depending upon the result desired. So it is clear from this discussion that all nodes of the CEG are not covered during each execution. Many components in sequence make an execution path. Figure 1 shows a Component Based System in the form of a CEG. This system contains 9 components and 11 edges. Each component is assigned a weight which is a composite value composed of four independent and sequential parameters 1). I –Interfaces value, 2). E – Exceptions value, 3). C – Complexity Value and 4). R – Reusability value, in that sequence. All these four parameters are quantifiable and stochastic in nature. They are exponentially distributed. According to Erlang distribution, if there are k independent random variables v_i(i= 1 to k), which in this case happens to be 4, having the same exponential distribution given by the function:

$f(v_i) = \mu k e^{\mu k v_i}$ where $v_i>0$, $\mu>0$, k a positive integer then (1)

$$V = \sum_{i=1}^{k} v_i \text{ has the Erlang distribution} \quad (2)$$

We can obtain a random variate from Erlang distribution by obtaining k random variates from the exponential distribution and then summing them up. So if

$$v_i = -\frac{1}{\mu k} \ln r_i \quad \text{for } i = 1 \text{ to } k \quad (3)$$

Then we can have

$$v = \sum_{i=1}^{k} v_i = \sum_{i=1}^{k} -\frac{1}{\mu k} \ln r_i = -\frac{1}{\mu k} \ln \prod_{i=1}^{k} r_i \quad (4)$$

So here I,E,C and R are composed together and the composite weight is assigned to the corresponding component and to all the execution links terminating into that component in turn. This composite weight is Erlang-4 distributed because it is a composition of four independent parameters.

4. Assumptions

Because we are trying to solve the problem with the help of simulation, we will definitely make some assumptions. These assumptions are given as under:

1. Each node of the graph represents one independently developed/purchase off the shelf/modified component.
2. One component is one unit of execution.
3. Control is transferred from one component to another along an execution link depending upon the result desired.
4. Each execution link is represented with the help of an edge or arrow from source to destination component
5. All the execution links are assigned number in topological order according to Fulkersons's 'i-j' rule [20].
6. One execution path of the CEG is a combination of many execution links starting from first component (node) of the graph and terminating in the last component. In between there may lie

many link combinations forming many paths. But each execution path starts from first node and terminates in the last node.

7. Each execution link is assigned a weight "w". This weight is actually the weight of the destination component. This weight is a composite parameter composed of four independent sequential parameters I,E, C and R as already discussed. I, E, C and R follow exponential distribution and their composition follows Erlang-4 distribution pattern.
8. Weight 'W' of an execution path is the sum of all 'w[i]'s of execution links along that path.
9. Execution path having the maximum weight is called the "Critical Execution Path" and execution links falling along that path are all critical execution links and all the components falling on this path are the critical components.

4.1 Terms and Notations used

Following are the terms and notations used to represent various parameters in the algorithm:

- F Starting Component
- L Last Component
- SC[i]: Starting Component of execution link i.
- TC[i] Terminating Component of execution link i.
- Min_Start[i] Minimum starting cumulative weight of Execution Link i.
- Min_Term[i] Minimum terminating cumulative weight of execution link i.
- Max_Start[i] Maximum starting cumulative weight of Execution Link i.
- Max_Term[i] Maximum terminating cumulative weight of execution Link i.
- Min_C[j] Minimum weight of component j
- Max_C[j] Maximum weight of component j.
- SIMURUNS No of simulation runs
- M No of Components in the system
- N No of Execution links in the system
- w[i] Weight of execution link i.
- W Weight of complete execution path.
- E Error
- CritIndex_E[i] Criticality index of ith execution link.
- CritIndex_C[j] Criticality index of jth component

5. Algorithm

1. Input
   a. SIMURUNS (Number of Simulation Runs)
   b. N (Number of Execution Links)
   c. M (Number of Components)
   d. SC[i] (Starting Component for each execution link from 1 to N)
   e. TC[i] (Finishing Component for each execution link from 1 to N)
   f. E (Error)
2. Initialize
   a. *CritIndex_E[i] = 0 for i = 1 to N*
   (Set Criticality Index of each execution link to 0)
   b. *CritIndex_C[j] = 0 for j = 1 to M*
   (Set Criticality Index of each component to 0)
3. Repeat steps 4 through 8 SIMURNS times
4. Generate N random variants from Erlang-4 distribution and store them in vector *w i.e. w[i] = n[i] for i = 1 to N.*
5. Start Forward Traversal of the CEG
   a. *Min_Term[i] = Min_Start[i] + w[i]*
   b. (Each Component node may have many execution links terminating into it. Same process is applied on each execution link. Once all the execution links terminating into a component node have been covered, minimum weight that can be assigned to a component is computed).
   Min_C[j] = max{Min_Term(all execution links terminating into component node j)}
   c. (Once minimum weight of component j has been computed, next step is to compute the minimum cumulative starting weight for the ith execution link starting from this component as follows).
   Min_Start[i] = Min_C[SC(i)]
   Repeating this process for each combination of execution links and component nodes, ultimately end of the CEG is reached
   d. (Compute the minimum possible weight for this last component).
   Min_C[M] = W

6. Start Backward Traversal of CEG
   a. *Max_C[M] = W*
      (Last component node is assigned a weight W computed during the forward pass
      Max_Term [All executing links terminating in component M] = Max_C[M]
      Max_Start [N] = Max_Term[N] – w[N]
   b. (Starting from the last component node, compute the maximum starting weight of each execution link
      Max_Start[i] = Max_Term[i] – w[i]
   c. (Once Max_Start for all the execution links starting from a component node 'j' have been established, next step is to compute the maximum weight that can be assigned to component node 'j')
      Max_C = min{Max_Start (All exection links originiating from component node j)}
   d. (Compute Maximum Terminating weight, Max_Term, of all components of CEG
      Max_Term (all execution links starting from component j) = Max_C[j]
7. (Update Criticality Indecies)
   a. *If (Max_Start[i] – Min_Start[i] <= E*
      CritIndex_E[i] = CritIndex[i] + 1
   b. *If (Max_C[j] – Min_C[j] <= E*
      CritIndex_C[j] = CritIndex_C[j] + 1
8. Increment SIMURUNS
   SIMURUNS = SIMURUNS + 1
9. Print Criticality Indices
10. Stop

6. Case Studies

6.1 Case Study 1

This simulator was developed in C language on windows XP. For the first case study, we have taken a Component Execution Graph with 9 components and 11 execution links as shown in figure 1. Each execution link was assigned a random weight. This random weight is actually a composition of four independent sequential parameters I (Interface Value), E (Exceptions), C (Complexity) and R (Reusability value) that are respective values assigned to destination component of execution link i, and hence it follows Erlang-4 distribution. For 1000 simulation runs, results obtained are shown in table 2 and 3.. Table 2 shows the criticality indices of various Execution Links (No of times an execution link becomes critical) and its graphical presentation in given in Graph1. Table 3 shows the criticality indices of various components (No of times a component becomes critical). This data has been plotted in graph 2.

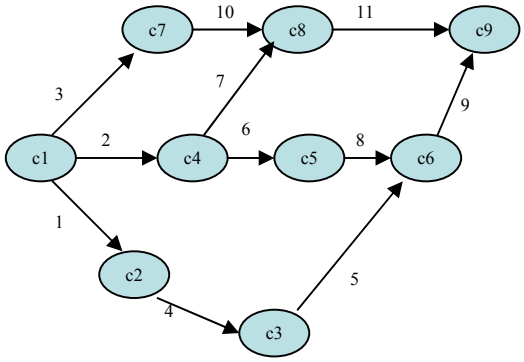


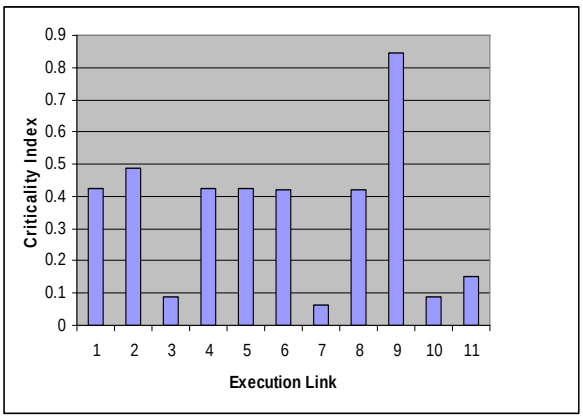
Figure 1: Component Execution Graph (CEG)

Table 1: Simulator input Data (Starting and Terminating Component for each execution link)

Execution Link Number	Starting Component	Terminating Component
1	C1	C2
2	C1	C4
3	C1	C7
4	C2	C3
5	C3	C6
6	C4	C5
7	C4	C8
8	C5	C6
9	C6	C9
10	C7	C8
11	C8	C9

Table 2: Simulation Output (Criticality Indices of Execution Links)

Execution Link	Criticality Index
1	.426
2	.487
3	.087
4	.426
5	.426
6	.422
7	.065
8	.422
9	.847
10	.087
11	.152



Graph 1: Criticality Indices of Execution Links

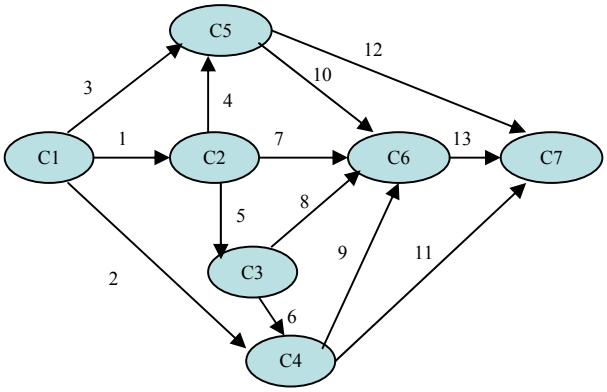
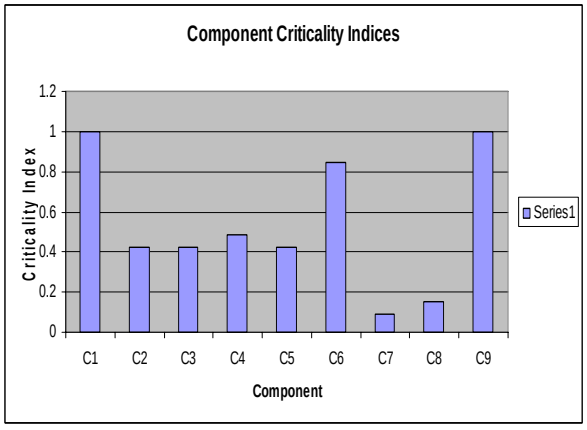


Figure 2: Component Execution Graph

Table 3: Simulation Output Data (Criticality Indices of Components)

Component	Criticality Index
C1	1.0
C2	.426
C3	.426
C4	.487
C5	.422
C6	.847
C7	.087
C8	.152
C9	1.0



Graph 2: Criticality Indices of Components

6.2 Case Study 2

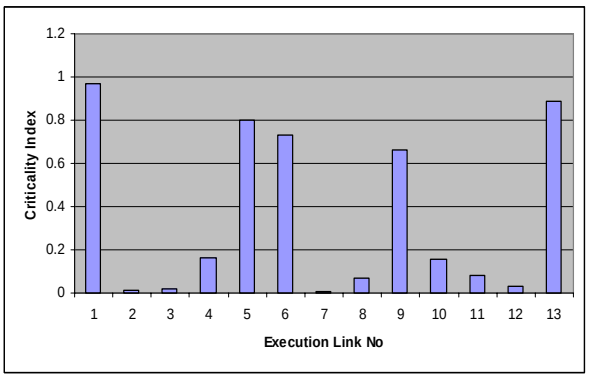
For the second case study we have taken a system with seven components and 13 execution links as shown in figure 2. Results obtained for the criticality indices have been shown in Table 5 and Table 6 and Graph 3 shows the critical indices of the execution links and Graph 4 shows the Criticality Indices of the Components.

Table 4: Simulator input Data (Starting and Terminating Component for each execution link

Execution Link Number	Starting Component	Terminating Component
1	C1	C2
2	C1	C4
3	C1	C5
4	C2	C5
5	C2	C3
6	C3	C4
7	C2	C6
8	C3	C6
9	C4	C6
10	C5	C6
11	C4	C7
12	C5	C7
13	C6	C7

Table 5: Simulation Output (Criticality Indices of Execution Links)

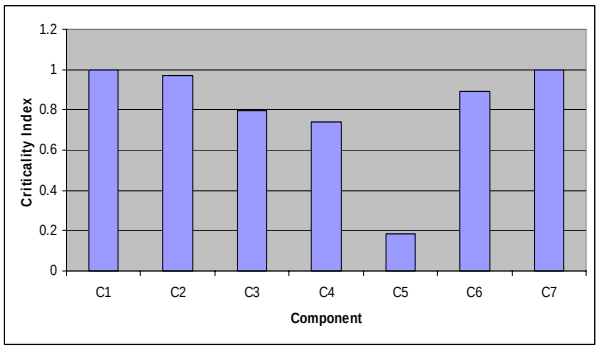
Execution Link	Criticality Index
1	.971
2	.010
3	.018
4	.165
5	.797
6	.732
7	.009
8	.067
9	.662
10	.154
11	.080
12	.029
13	.890



Graph 3: Criticality Indices of Execution Links

Table 6: Simulation Output Data (Criticality Indices of Components)

Component	Criticality Index
C1	1.0
C2	.971
C3	.799
C4	.742
C5	.183
C6	..890
C7	1.0



Graph 4: Criticality Indices of Components

7. Discussion and Conclusions

From the output received for given inputs many decisions can be made. It is clear from Table 2 and Graph 1 that Execution Links numbered 1,2,4,5,6,8,9 are the ones that are key links in this component based applications. Hence we need to concentrate more on these execution links and spend more resources and time to make these execution links error free. As is clear from graph 1, among these critical execution links, link number 9 is the most critical one and this link is part of almost all the execution paths, so this execution link needs to be tested most rigorously. Execution links numbered 3, 7, 10 and 11 are not those much critical and hence keeping this thing in mind the project team can make its decision on spending resources for various execution links. Also a decision can be made looking at graph 2 which components need to be tested more rigorously. As far as this data set is concerned, component C1, C6 and C9 are the key components in this application.

As far second case study is concerned, results of table 5 and graph 3 shows that Execution Links numbered 1,5,6,9 and 13 are the key execution links for this application and more emphasis should be given on these links while testing. It is clear from Table 6 and Graph 4 that for this application components C1, C2, C6 and C7 are most critical ones and there testing should not be skipped at any cost.

Here for the simplicity sake, we have considered an application that contains only 9 components and 11 execution links in first case study and 7 components and 13 execution links in the second case study respectively. In practical, however, a component based application may be composed of hundreds or thousands of components. In such a situation, it becomes very difficult for the project team to identify the components and execution links that are more error prone and need more testing time and efforts and to decide how to distribute the human as well as financial resources in testing the components and their interactions. This simulator can be a handy tool in such situations. Besides this, it can also be decided, which component and execution link needs to be tested up to what level.

References:

[1] Wu, Y., et al, "Techniques for Testing Component Based Software," In the proceedings of 7th IEEE International Conference on Engineering of Complex Computer Systems, Skovde, June 2001, pp. 222-232.

[2] Chattergee, R. and Ryder, B., "Data Flow Based Testing of Object Oriented Libraries," Technical Report, DCS-TR-382, Rutgers University, !999.

[3] Mao, C., and Lu, Y., "Regression Testing for Component Based Software by Enhancing Change Information," In the proceedings of 12th IEEE Asia Pacific Software Engineering Conference,2005, pp. 611-618.

[4] Byoun, W., et al, "Test Case Generation Techniques for Interoperability Test of Component Based Software from State Transition Model," IJCSNS Internation Journal of Computer Science and Network Security, Vol. 7, No. 5, May 2007, pp. 151-157.

[5] Mariani, L., et al, "Compatibility and Regression Testing of COTS- Component Based Software," In the proceedings of 29th IEEE conference on Software Engineering, 2007, pp. 85-95.

[6] Gao, J., "Component Testability and Component Testing Challenges," In the proceedings of 3rd International Workshop on Component Based Software Engineering: Reflects and Practice," 2000.

[7] Chattergee, R. and Ryder B., "Data Flow Based Testing of Object Oriented Libraries," Technical Report DCS-TR-382, Rutgers University, 1999.

[8] Orso, A., et al, "Using Component Metacontent to Support the Regression Testing of Component Based Software," In the proceedings of IEEE International Conference on Software Maintenance, IEEE Press, 2001, pp. 716-725.
[9] Chenn, M. and Kao, M., "Effect of class testing on the Reliability of Object-Oriented programs,"In Proceedings of the Eighth International Symposium on Software R liability Engineering, May 1997.
[10] Sanjeev, A. and Wibovo, B., "Regression Test Selection Based on Version Changes of Components," Proceedings of 10th Asia Pacific Software Engineering Conference, IEEE Press-2003, pp. 78-85.
[11] Mao, C., et al, "Regression Testing for Component Based Software via Built in Test Design," In the proceedings of ACM Symposium on Applied Computing, 2007, pp. 1416-1421.
[12] Chen, M. and Kao, M., "Testing Object Oriented Program-An Integrated Approach," 10th International Symposium on Software Reliability Engineering, November 1999.
[13] Harold, M. Liang, D. and Sinha, S., "An Approach to Analysing and Testing Component Based System" 1st International ICSE workshop on Testing Distributed Component Based Systems, L.A., USA, May 1999.
[14] Perry, D., and Kaiser G., "Adequate Testing And Object Oriented Programming," Journal of Object Oriented Programming, Vol. 2, Issue 5, 1990, pp. 13-19.
[15] Weyuker, E, "The Ealuation of Program Based Software Test Data Adequacy, Communication of ACM, June 1988, pp. 668-675.
[16] Weyuker, E., "Testing Component Based Software: A Cautionary Tale," IEEE Software, Sep/Oct 1998, pp. 54-59.
[17] Gao, J., "Monitoring Software Componenets and Component Based Software," In the proceedings of 24th Annual International Computer Software and Application Conference, Taipei, Taiwan, Oct 2000.
[18] Gao, J., et al "Testing and Quality Assurance for Component Based Software," Artech House Inc, 2003.
[19] Gao, J., "Testing Coverage Analysis for Software Component Validation," In the proceedings of 29th Annual International Computer Software and Applications Conference, Edinburgh, Scotland, July 26-28, 2005.
[20] Wills, R., "A Note on the Generation of Project Network Diagram," Operation Research Society, Vol. 32, 1981, pp. 235-238.
[21] McGregor, J.D., "Component Testing," Journal of Object Oriented Programming, Vol. 10, No. 1, 1997. pp. 6-9.
[22] Rosenblum, D., "Adequate Testing of Component-Based Software," Univ. California, Irvine, T.R. UCI-ICS-97-34, 1997.

Dr. P.K. Suri received his Ph.D. degree from Faculty of Engineering, Kurukshetra University**,** Kurukshetra, India and master's degree from Indian Institute of Technology, Roorkee (formerly known as Roorkee University), India. He is working as Dean, Science Faculty and Professor in the Department of Computer Science and Applications, Kurukshetra University, Kurukshetra – 136119 (Haryana), India. He has earlier worked as Reader, Computer Sc. & Applications, at Bhopal University, Bhopal from 1985-90. He has supervised twelve Ph.D.'s in Computer Science and thirteen students are working under his supervision. He has around 125 publications in International/National Journals and Conferences. He is recipient of 'THE GEORGE OOMAN MEMORIAL PRIZE' for the year 1201-92 and a RESEARCH AWARD –"The Certificate of Merit – 2000"for the paper entitled ESMD – An Expert System for Medical Diagnosis from INSTITUTION OF ENGINEERS, INDIA. His teaching and research activities include Simulation and Modeling, Software Risk Management, Software Reliability, Software testing & Software Engineering processes, Temporal Databases, Ad hoc Networks, Grid Computing, and Biomechanics.

Sandeep Kumar received his Masters Degree in Computer Science from Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana, India in 2001. He is a Ph.D. scholar under the guidance of Dr. P.K. Suri at Department of Computer Science and Applications, Kurukshetra University, Kurukshetra. He has more than seven years of teaching experience at institutions of repute. Presently he is working as Assistant Professor and Head, Department of Computer Applications, Dronacharya Institute of Management and Technology (DIMT), Kurukshetra since July 2007. Prior to this he worked as a lecturer at DIMT, Kurukshetra and Asia Pacific Institute of Information Technology SD India (APIIT SD India), Panipat, Haryana, India. He was the editor of Proceedings of an International Conference (CNFE' 05) at APIIT SD India. His research interests include Component Based Software Engineering, Simulation, and Operating Systems.