Simulator for evaluating Reliability of Reusable Components in a Domain Interconnection Network

P.K. Suri[1] and Neeraj Garg[2]

1 *Professor, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra (Haryana) India,*

2 *Asst. Professor & Head, Department of Masters in Computer Science & Applications , Maharaja Agresen Institute of Management & Technology , Jagadhri(Haryana), India*

Summary

This paper analysis the various domains of a Interconnection network systems and the different reusable components that are available for these Domains. The detailed sub-domain analysis is done and the Reliability is calculated for each sub-domain of a domain for each component A number of components are available for a particular domain for black-box reuse. A Simulator uses the parameters of the available components for black box reuse. In Phase-I and only that component qualifies which can meet the requirements of a particular domain. In the Phase-II the simulator checks the reliability of the qualified components by simulating the actual environment , system , user and other factors where the components will be used. The reliability of the system is calculated which incorporating the effects of every sub-domain of each qualified component. The component with the highest reliability qualifies to be reused in the system.

Key words:

Software Reusability-Domain Analysis –Sub-domains-Operational Profiles of component-Domain Interconnection Network – System Reliability of – Simulation

1. Introduction

1.1 Domain Oriented reusability

Domain-oriented reusability is a process of identifying the reusable abstractions in a problem. Domain analysis and modeling deal with identifying reusable abstractions and architectures for the development

We have taken a domain-oriented approach frequently reusable abstractions are identified for the application domain of a software. Specific domain roles are identified, reuse guidelines are used for the domain analysis, and a rule-based approach taken for the domain representation.

The domains are related by virtue of specifications from abstract domains being refinable to specifications (or programs) in domains of lower level of abstractions. This implicitly establishes a domain interconnection network (an example of which can be found in Figure1) with specific application domains at the most abstract level, generic application domains, computer science domains, and execution model domains at intermediate levels of decreasing abstraction, and target execution languages at the lowest level(s) of abstraction. [10]

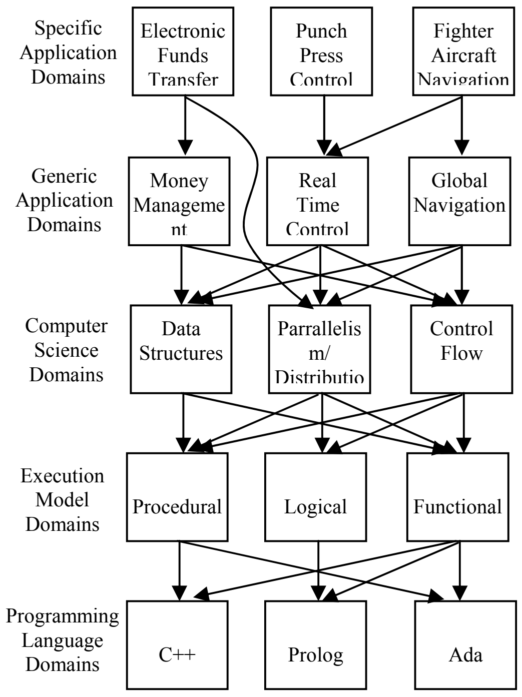

Fig.1 A Domain Interconnection Network[10]

In this research, we have tried to identify the reusable component who will give the maximum reliability to the system . In our approach to domain analysis, we have identified the following:

- Support for frequently reusable abstractions.
- A specific set of domain roles.

Manuscript received March 5, 2008
Manuscript revised March 20, 2008

- Practical and objective reuse guidelines to represent the application domain knowledge and language knowledge, and to provide reuse analysis and advice.

- A rule-based approach for the domain representation.

- Methods for assessing components for reuse which provide the maximum reliability to the system.

The domain knowledge in the network can be heavily reused in the construction process of many different software systems, and can consequently be well tested. Most of the errors that originally might have been compiled in the domain knowledge are eliminated over time. This contributes to higher reliability of newly constructed systems reusing these domains, as well as to systems already constructed with the possibly erroneous domain knowledge via maintaining them. Similar to hardware, software performance is significantly dependent on the environment in which it operates. With hardware, the environment physically changes a piece of equipment. this physical change is mainly responsible for faulty behavior. A software system doesn't change, but can still fail due to the inputs it receives from the external environment.

1.2 Operational Profiles

A reliability of a software-based product depends on how the computer and the external elements will use it[1]. The operational profile , a quantitative characterization of how the software will be used , is therefore essential in any Software reliability Engineering application.
Since most designers try to reuse software as much as possible but the process of searching for the reusable software costs money, use the operational profile to select operations where looking for the opportunities for the reuse will be most cost effective [2]

We generally practice Domain Engineering for the development of components that will be used in a number of different systems . The process for the developing operational profile for such components is the same as that used for the other systems .A need for greater breadth of analysis to understand all the different ways that such components may be used is required. [3]

An operational profile is a complete set of operations with probabilities of occurrence . Probability of occurrence refers to probability among all invocations of all operations . [Musa,04] For example , a probability of occurrences of 0.15 for an operation 'x' means that 15 of every 100 invocations of operations will be 'x'
A profile is a set of independent possibilities called elements ,and their associates probability of occurrence . If operation P occurs 55 percent of time , Q occurs 40 percent , and R occurs 15 percent , the profile is (P,0.55…Q,0.4…R,0.15).

Operation profile of a software reflects how it will be used in practice . It consists of a specification of classes a of input and the probability of their occurrence . [12]

Developing an operational profile for the system involves ; finding the customer profile, establish the user profile, define the system mode profile and finally to determine the functional profile and operational profile itself. The process for developing the operational profiles is as shown in figure- 2

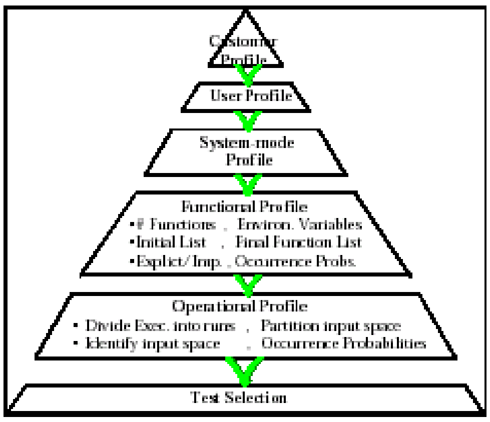
Customer Profile
User Profile
System-mode Profile
Functional Profile
•# Functions , Environ. Variables
•Initial List , Final Function List
•Explict/Imp. , Occurrence Probs.
Operational Profile
• Divide Exec. into runs , Partition input space
• Identify input space , Occurrence Probabilities
Test Selection

Figure 2. Operational profile Development [2]

Operational profiles can be developed by the following steps[3]

1. Identify initiators of Operations
2. Create Operation List
3. Review operation list
4. Determine occurrence rates
5. Determine occurrence probabilities

A customer profile consists of an array of independent customer types. A customer type is one or more customers in a group that intend to use the system in a relatively similar manner, and in a substantially different manner from other customer types. Each of these types of customers may be expected to utilize the software in a substantially different way. The customer profile is the list of customer types and associated probabilities. These probabilities are simply the proportion of time that each customer would be using the system

A system users may be different from the customers of a software product. A user profile is the set of user types and their associated probabilities of using the system.
A system mode is way that a system can operate. The system includes both hardware and software . Most systems have more than one mode of operation. For example , system testing may take place in batch mode or user-interactive mode. An airplane consists of takeoff and ascent mode, level flight mode and descent and land mode. System modes can be thought of as independent segments of a system operation or various different ways of using a system

Functions are essentially tasks that an external entity such as a user can perform with the system . For instance, the user , of an e-mail system would want the following functions : create message, look up address, send message ,open message , etc. A functional profile can be explicit or implicit , depending upon the key input variables which is an external parameter which affects the execution path of a software system. These key parameters variables consists of ranges called levels of variables that cause different operations to be performed . A profile is explicit if each element is designated by simultaneously specifying levels of all key input variables needed for its identification.

For example , there are two independent parameters X and Y , each taking on three discrete values . Nine operators can be defined based on combinations of variables as shown in table below

Table 1: Implicit Operational Profiles

Key Input Variable	Occurrence Probability	Key Input Variable	Occurrence Probability
X1	0.5	Y1	0.6
X2	0.4	Y2	0.3
X3	0.1	Y3	0.1

Table 2: Explicit Operational Profiles

Key Input Variables values	Occurrence Probabilities
X1Y1	0.30
X2Y1	0.24
X1Y2	0.15
X3Y1	0.06
X1Y3	0.05
X2Y2	0.12
X2Y3	0.04
X3Y2	0.03
X3Y3	0.01

For five variables with five levels, assuming complete independence, the implicit profile requires only 25 elements whereas the explicit profile would call for 5^5, or 3125 elements.[4]

Generating a functional profile .
Development of functional profile generally involves the following four steps :

1. Generate an initial function list.
2. Determine environment variables
3. Create final function list.
4. Assign occurrence probabilities

The initial function list should be comprised of features and capabilities needed by the users . The next step is to define the environmental inputs (this can be provided as random values by the Simulator) variables and their value ranges that segregate development.

The next step is to define the environmental input variables and their value ranges that segregate development. Environmental variables characterize the conditions that influence the paths traversed by a program , but do not correspond directly to features. Examples of environmental variables include hardware configuration and traffic load.

The final numbers of functions in the list are calculated as the product of the number of functions in the initial list ad the number of environmental variable levels, minus the combinations of initial functions and environmental variables values that do not occur. The final function list consists of the functions and environmental variables for each function.

The final step in functional profile development is assigned of occurrence probabilities . These measurement may be obtained from the system logs and data storage devices . Occurrence probabilities computed with the historical data should- be updated to account for new functions , users , or environments

2. Occurrence Probabilities of Domains and Sub-Domains

Figure no . -- shows the elements involved in determining operational profiles from the functions. After each input variable is portioned into ranges , probabilities associated with each domain and sub-domain must be identified. A interaction matrix is created where input variables are plotted against other key input variables. The matrix reveals the combinations of variables that do not occur or contain dependencies. The remainder of matrix contain independent combinations where the estimates of occurrence probabilities are the product of individual input variable probabilities.

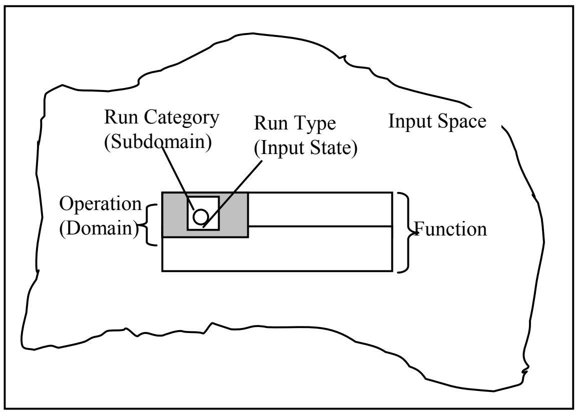

Figure3 Operational profile Development[2]

2.1 Operational Profile Mapping

A component developer provides a data sheet giving a set of sub-domains, and two mappings of an input profile. A profile is assumed to be expressed as a weighting over the sub-domains. Let the sub-domains Si partition the component input domain

$$D = S1 \cup S2 \cup ::: \cup Sn.$$

A profile W is then a vector of weights to be assigned to each Si:

$$W =< h1; h2; :::; hn > :$$

2.2 Reliability mapping

Carries a profile vector to a real value $R \in [0; 1]$, the probability that the component will not fail on an input drawn according to this profile. Failure rates **fi** are measured within each sub-domain using random testing. Then reliability R of a domain is given by the equation

$$R = \sum_{I=1}^{n}(1- hifi)$$

3. Profile-transformation mapping

Carries an input profile vector to an output profile, the latter expressed as a weighting vector over an arbitrary set of sub-domains U1; U2; :::; Um (unrelated to the sub-domains on the data sheet). The weightings of the output profile

$$Q =< k1; k2; :::; km >$$

on these sub-domains is the sum of the contributions from each input sub-domain,

$$kj = \sum_{i=1}^{n} hi \,|\{z \in Si \mid c(z) \in Ui \}| / |Si|$$

where c is the function computed by the component. The information on the data sheet and the ability to execute the component to calculate c, are sufficient to estimate the ki given Ui, by making a random selection from each Si. [5] The profile-transformation describes how a profile is altered across a component, and the reliability mapping gives a component's independent contribution to the system reliability.

3.1 System Design and Reliability

The example of a system design and reliability calculation using two Domains A and B in sequence will illustrate our ideas.

Components available for black box reuse are for domain A . The available component are Xa, Ya, Za , La, Ma.

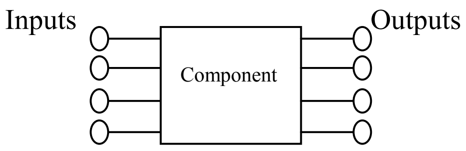

Fig.4 A typical Component

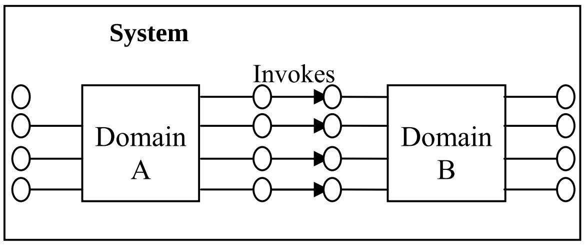

Fig.5 System and its sub-domains

4. Assumptions

1. The components must be functional, that is, must not communicate through global state.
2. A is given the system input, and invokes B, passing its output as B's input; B's output is the system output. A must *invoke* rather than *use* B, in the sense of Parnas [6. The system developer

has a profile vector for the system, and the data sheets for A and B.

3. The domain analyzer and improver must know the attributes to be generalized within that domain so that it enhances reusability of that abstraction. For example, if a component of a dynamic abstract data structure is to be generalized then the system should check for generic abstraction.
4. It is necessary to provide advice on structural information on existing abstractions and on newly required abstractions. For example, it is not always clear how to select the most suitable abstract data structure for a specific application, and how to hide representation details.
5. It is always difficult for the reuser to understand how a particular abstraction can be reused and what are the possible applications. For example, it is not always clear to component reusers what are the possible applications of a selected abstraction. Therefore, the domain analyzer must know to advise on how to reuse and what are the possible applications of an abstraction.
6. The domain system should analyze and assess components against reuse guidelines that are represented and should report the percentage of matching guidelines, so that the designer is aware of his component's potential for reuse. Also it should provide suggestions on how that abstraction can be improved for reuse automatically. Assessment reports can be produced based on the grading system introduced earlier, a component is weakly/ limitedly/ strongly/ immediately reusable.

5. System Reliability

Software Reliability is the probability of failure-free operation of a program for a specified time under a specified set of operating conditions[flakes – Foundational Issues in software Reuse and Reliability]
Software often fails when the when it is given input and used in operating environments not foreseen by the software developers . Domain Engineering and reuse design address this problem by trying to predict the various users to which the software will be put . This is sometimes k known as the oracle hypothesis.
A key difference between software and hardware Reliability is that software does not deteriorate or physically change in any other way . The typical hardware notion of components "wearing out " does not apply to software .
The software reliability modeling is based on some other source of randomness in the failure times. The most fruitful understanding of the sources of randomness in software seems to be that some inputs will produce correct outputs and others will produce incorrect Outputs (failures), and the unpredictability of the next input means that a model based on random input sequences may be best one can do
Using the system profile (which is input to A), A's reliability mapping can be used to calculate RA, the reliability of A alone. (The sub-domains of the system are projected onto the sub-domains of A's data sheet.) A's profile-transformation mapping can be used with the sub-domains from B's data sheet as the Ui, to calculate the profile B will see. Then B's reliability mapping can be used to calculate reliability RB for B alone. The system reliability is finally calculated as RARB.

6. Simulator

The guidelines discussed in this paper have been partially or completely automated in our system for which a prototype has been developed as shown in Figure 3 & 4. For most of these guidelines, Simulation depends on some user interactions and domain knowledge**.**

6.1 Simulator -Phase I

In Phase-I Simulator analysis the components on the basis of inputs from Environment , Hardware System on which the component is going to run and the other factors which vary from time to time; in comparison with the parameters of the component . The output is the component which qualifies for Reuse .

Qualification of a component for Reuse assessment

Two measures of software quality that qualify for a data sheet are
(1) that the component has been proved correct, or
(2) that it has been randomly tested using an accurate user profile. Both of these require a formal specification of what the component is supposed to do (the part of the data sheet that we do not consider).

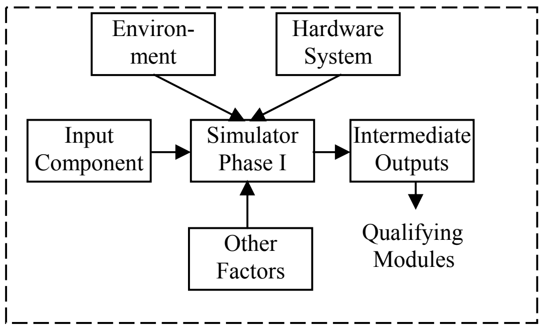

Fig :6 Simulator – Phase I – Qualifying Components

A statement that the component has been proved correct is a guarantee that it will perform according to its specification. A statement that (say) its reliability is better than 1 _ 10_4 per execution with an upper confidence bound of 99%, is a statistical assertion that there is no more
than 1 chance in 100 that it will not perform according to the specification in 10,000 trials. Correctness proofs (1) can be thought of as the special case of a profile-independent reliability
of 1.0, with 100% confidence. The reader of a data sheet might doubt that the developer has actually established such precise technical claims. But this is a question that can be answered scientifically, and if the developer has lied there are legal remedies. When the quality information on a component's data sheet is statistical, it must be obtained by random testing. The fundamental problem of assessing component quality is that any standard profile from which test inputs are drawn will not match the profile that the component will experience when placed in a system. We solve the profile problem with data-sheet mappings based on a sub-domain decomposition of the component input domain. A system designer can use these maps to predict the system reliability before implementation.

6.2 Simulator -Phase II

In Phase-II , the Simulator does the detailed reliability testing of the qualified components . The reliability of each domain and sub-domains are calculated for every component and then the system reliability is calculated. The component which gives the highest reliability to the system in a Domain Interconnection Network is selected for reuse in that system.

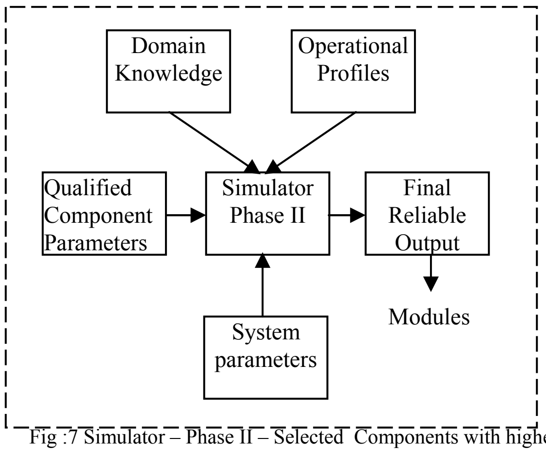

Fig :7 Simulator – Phase II – Selected Components with highest System Reliability

6.3 Algorithms

Phase I

Read the parameters of the available components

IF the parameters meet the domain requirements up to 95% THEN

Accept the component (qualifies) for the Phase II

ELSE Reject the component .

Phase II

For Independent Domain :

IF abstract structure is reliable AND

all reliability R of the sub-domain Analysis is greatest THEN

Component should be implemented as a generic package with the element type as a generic parameter; End if;

For Domain Interconnection Network

Domain A ------→ Domain B

IF abstract structure is reliable AND

all reliability R of the sub-domain Analysis is greatest AND

IF in the Interconnection Network the output Profile of component is Greater than the output profile of other components THEN

Component should be implemented as a generic package with the element type as a generic parameter; End IF.

Calculations and Observations From the Simulator

Case: The system has two domains A and B.A invokes B . Xa , Ya, Za, La, Ma are reusable components available for black box reuse . The respective failure rates in each sub-domain for each component is available.

Table 3: Specifications of Domain A

Domain A	
Sub domain	Operational Profile hi
A1	0.3
A2	0.1
A3	0.6
Total	1

Simulator Phase-I

Table 4: Results of Phase I of Simulator

Component	Percentage Applicability	Decision
Xa	98%	Qualifies
Ya	99%	Qualifies
Za	97%	Qualifies
La	93%	Not Qualified
Ma	94%	Not Qualified

Simulator Phase II

Table 5: Reliability of component X

Component Xa				
Sub domain	hi	fi	1-fi	hi(1-fi)
Xa1	0.3	0.01	0.99	0.297
Xa2	0.1	0.009	0.991	0.0991
Xa3	0.6	0.001	0.999	0.5994
	1			0.9955
Reliability of component Xa=0.9964				

Table 6: Reliability of component Ya

Component Ya				
Subdomain	hi	fi	1-fi	hi(1-fi)
Ya1	0.3	0.01	0.99	0.297
Ya2	0.1	0.02	0.98	0.098
Ya3	0.6	0.003	0.997	0.5982
	1			0.9932
Reliability of component Ya=0.9932				

Table 7: Reliability of component Za

Component Za				
Subdomain	Hi	fi	1-fi	hi(1-fi)
Za1	0.3	0.07	0.93	0.279
Za2	0.1	0.008	0.992	0.0992
Za3	0.6	0.005	0.995	0.597
	1			0.9752
Reliability of component Za=0.9964				

Calculated Reliability of B as it sees from Xa

Table 8: Invocations of Domain B from Xa

Subdomain	From Xa1	From Xa2	FromXa3	Fi
B1	0	0	0	0.1
B2	0.003	1	0.002	0
B3	0.147	0	0.162	0.0
B4	0.85	0	0.836	0.02

Table 9 :Calculation of K from Xa

Xa							
	h1	From Xa	h2	From Xa2	H3	From Xa3	K
K1	0.3	0	0.1	0	0.6	0	0
K2	0.3	0.003	0.1	1	0.6	0.002	0.1021
K3	0.3	0.147	0.1	0	0.6	0.162	0.1413
K4	0.3	0.85	0.1	0	0.6	0.836	0.7566

K1= 0.3(0)+0.1(0)+0.6(0)= 0
K2= 0.3(0.003)+0.1(1.0)+0.6(0.002)= 0.102
K3= 0.3(0.147)+0.1(0)+0.6(0.162)= 0.141
K4= 0.3(0.850)+0.1(0)+0.6(0.836)= 0.757
So the Profile B sees from A is <0,0.102,0.141,0.757> and B's Reliability is

Table 10 : Reliability of B from Xa

Subdomain	K	K-Value	Fi of B	1-FiB	K1*FiB
B1	K1	0	0.1	0.9	0
B2	K2	0.22	0	1	0.22
B3	K3	0.156	0	1	0.156
B4	K4	0.624	0.02	0.98	0.61152
				Rb	0.98752

Calculated Reliability of B as it sees from Ya

Table 11: Invocations of Domain B from Ya

Subdomain	From Xa1	From Xa2	FromXa3	fi
B1	0	0	0	0.1
B2	0.01	1	0.009	0
B3	0.156	0	0.19	0.0
B4	0.834	0	0.801	0.02

Table 12 : Calculation of K from Ya

Ya							
	h1	From Ya1	h2	From Ya2	h3	From Ya3	K
K1	0.3	0	0.1	0	0.6	0	0
K2	0.3	0.01	0.1	1	0.6	0.009	0.1084
K3	0.3	0.156	0.1	0	0.6	0.19	0.1608
K4	0.3	0.834	0.1	0	0.6	0.801	0.7308

Table 13: Reliability of B from Ya

Subdomain	K	K-Value	Fi of B	1-FiB	K1*FiB
B1	K1	0	0.1	0.9	0
B2	K2	0.1084	0	1	0.1084
B3	K3	0.1608	0	1	0.1608
B4	K4	0.7308	0.02	0.98	0.716184
					0.985384

Calculated Reliability of B as it sees from Za

Table 14: Invocations of Domain B from Za

Subdomain	From Za1	From Za2	FromZa3	fi
B1	0	0	0	0.1
B2	0.2	1	0.1	0
B3	0.12	0	0.2	0.0
B4	0.68	0	0.7	0.02

Table15 : Calculations of K from Za

Za							
	h1	From Za1	h2	From Za2	H3	From Za3	K
K1	0.3	0	0.1	0	0.6	0	0
K2	0.3	0.2	0.1	1	0.6	0.1	0.22
K3	0.3	0.12	0.1	0	0.6	0.2	0.156
K4	0.3	0.68	0.1	0	0.6	0.7	0.624

Table17 : Reliability of B as it sees from Za

Subdomain	K	K-Value	Fi of B	1-FiB	K1*FiB
B1	K1	0	0.1	0.9	0
B2	K2	0.22	0	1	0.22
B3	K3	0.156	0	1	0.156
B4	K4	0.624	0.02	0.98	0.61152
					0.98752

7. System Reliability

System Reliability	
Reliability using Xa= RY+Xa* RBXa=	0.9804281
Reliability using Ya= RYa* RBYa=	0.9786834
Reliability using Xa= Rxa* RBZa=	0.9630295

Result : Reusable component **Xa** will give maximum system reliability

8. Discussions and conclusions

In the example the system reliability was calculated by simulating the inputs from three available components Xa, Ya and Za. The maximum reliability was observes when Xa was used.

The independent reliability may be different for each component if used separately but the effect on the system may vary depending upon th e operational profiles and the reliability of a particular sub-domain. The decision for the selection of a Component is thus based on the combination of both Domains A and B whose sub-domains are also referred to.

One reason most software engineers think reused software should be more reliable is that more frequent use should reveal any faults in the product. Instead , we may take it as an axiomatic that more frequent use of a software product increases the likelihood that any failure modes that may exist in the product will be uncovered and, sometimes ,be repaired . This is a standard reliability
the guiding principles of the open source community. To apply this argument sensibly to reused software , however , requires that we divide the reused software 's failure modes into two classes : those pertaining to functionality of software and those connected with the incidental work that has to go on for the software to be reused.[15]
There have been attempts to improve software reliability by using OR configurations of components that are identical in function but developed by different organizations. (Black Box Reuse)

Reference:

[1] Musa ,John D; Iannino, A.; Okumoto, K; "Software Reliability Measurement ", Prediction , Application , McGraw Hill,1987
[2] Musa,J.D., "Operational Profiles in Software Reliability Engineering ," IEEE Software Magazine, March 1993.
[3] Musa ,John D;" Software Reliability Engineering', Tata McGraw-Hill , 2005.
[4] Operational Profiles, www.cs.colostste.edu
[5] Hamlet Dick; Mason Dave; Woit Denise ; " Theory of Software Reliability Based on Component".
[6] David Pranas, on a 'buzzword': hierarchical structure, Proc. IFIP Congress '74, North Holland 1974.
[7] Denise Woit and Dave Mason , Software Component Independence , Proc. 3rd IEEE High- Assurance Systems Engineering Symposium 1998.
[8]Hamlet Dick; " Software Component dependability , a Subdomain based Theory , Technical Report RSTR-96-999-01, Reliable Software Technologies , Sterling, VA, 1996
[9] Denise Woit and Dave Mason , Software system reliability from component Reliability, Proc. Of 1998 Workshop on software Reliability Engineering (SRE'98) , July 1998.

[10] Pidgeon W. Christopher, 'Organizing and Enabling Domain Engineering to Facilitate Software Maintenance'
[11] Jacobson Ivar, Griss Martin , Jonsson Patrik ; 'Software Reuse – Architecture , Process and Organization for Business Success', Pearson Education ,2004
[12] Somerville Ian, 'Software Engineering ', 6th edition , Pearson Education ,2001
[13] Suri PK, Aggarwal KK , Software Reliablity of Programs with Network Structure. Microelectron reliab Vol 21 ,1981.
[14] Pressman S. Roger ,' Software Engineering – A practitioner's Approach', McGraw-Hill International Edition, Computer Science Series, 2001.
[15] Ramchandran M., Sommerville I. 'Software Reuse Guidelines.

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 Professor in the Department of Computer Science & Applications, Kurukshetra University, Kurukshetra - 136119 (Haryana), India since Oct. 1993. He has earlier worked as Reader, Computer Sc. & Applications, at Bhopal University, Bhopal from 1985-90. He has supervised five Ph.D.'s in Computer Science and thirteen students are working under his supervision. He has more than 100 publications in International / National Journals and Conferences. He is receipient of ''THE GEORGE OOMAN MEMORIAL PRIZE' for the year 1991-92 and a RESEARCH AWAWD – "The Certificate of Merit-2000" for the paper entitles ESMD- An Expert System for Medical Diagnosis from the Institution of Engineers, India. His Teaching and Research include Simulation and Modeling, SQA, Software Reliability , Software Testing & Software Engineering Process, Temporal Databases , Ad Hoc Networks , Grid Computing , and Biomechanics.

Neeraj Garg received his B.E. Degree and Masters in Computer Science and Applications (MCA) Panajb University , Chandigarh and Kurukshetra University, Kurukshetra in the year 1992 and 2001 respectively. Presently he is Head of the Department of MCA department at Maharaja Agresen Institute of Management and Technology , Jagadhri, Haryana, India . He had also worked with various organizations including C-DOT where he had carried out research work in SS#7 Protocol of Telephone Networks. He is co- editor of MAIMT- Journal of IT and Management. His research areas include Simulation and Modeling, Software engineering , System Programming and Networks.