Refining Complexity Metrics of Component Based Softwares by taking Average Use Factor

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted By

Richa Mittal

(Roll No. 821132004)

Under the supervision of:

Mrs.Ashima Singh

(Assistant Professor in CSE)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2014

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Refining Complexity Metrics of Component Based Softwares by taking Average Use Factor "**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mrs. Ashima Singh and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Richa Mittal

Signature:

(Richa Mittal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Ashima Singh 18/7/14

(Mrs. Ashima Singh)

(Assistant Prof. in CSE)

Countersigned by:

(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

18/7/14

(Dr. S. K. Mohapatra)

Dean (Academic Affairs)

Thapar University, Patiala

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Refining Complexity Metrics of Component Based Softwares by taking Average Use Factor "**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mrs. Ashima Singh and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(Richa Mittal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mrs. Ashima Singh)

(Assistant Prof. in CSE)

Countersigned by:

(Dr. Deepak Garg)

Head

Computer Science and Engineering

(Dr. S. K. Mohapatra)

Dean (Academic Affairs)

Thapar University, Patiala

Department

Thapar University, Patiala

Acknowledgement

I would like to take this opportunity to express our gratitude towards all the people who have in various ways helped in the successful completion of my thesis work.

I must convey my gratitude to my supervisor Mrs. Ashima Singh madam , Assistant professor ,CSE department for giving me the constant source of inspiration and help in preparing the thesis, personally correcting mywork and providing encouragement throughout the thesis.

I am especially thankful to Dr. Deepak Garg sir, Head of Department of CSE department for giving me the opportunity of that type of dissertation.

I also thank all my faculty members for steering us through the tough as well as easy phases of the thesis in a result oriented manner with concern attention. I was very fortunate to have an unconditional support from my family. I thank my parent who gave me courage to get my education, supported me in all achievements throughout my life. Last but not least ; I would like to thank God for giving me inner strength and courage to achieve my success.

Thanking You

Richa Mittal

Table of Contents

List of Tables

List of Figures

Abstract

Component-based software engineering (CBSE) has emerged as an approach that offers rapid development of system using fewer resources and effort. The CBSE gives the idea of reuse and cutting down the development cost. Thus testing of components becomes more complicated when developers are not provided with internal information of these components. As a result of this, understanding control data flow in black box components is a challenge. Component-based software facilitates development of complex systems by allowing black box testing of reusable components. Testing of black box components is a challenging area of research. Our research aims at finding the existing black box component testing techniques and challenges in CBSE. The systematic literature survey is based on International Journals collected from multiple-stage selection process. These journals have been published within the time span of 2004-2010.

We propose to compute the complexity metrics of component based software in more justified way by taking considerations of their using frequencies. The complexity metrics calculation of the component Based software's by using black box testing is still not refined. The reason is that the various components are not used by the end users uniformly. Again, use of various components depends upon user to user as per their requirements. So therefore calculating straight forward their complexity on the basis of number of components, their interfaces and data types are not sufficient. We must add the factor of their (AUF) average use factor by the customers of different components. As we know that every algorithm or program have 3 complexity states i.e . a) Best Case b) Average

case and c) Worst case. As we know that each and every components of software is not used uniformly by the users. So calculating merely on the basis of their no of components, interfaces and data types predicts only theoretical complexity of that software. If we wish to calculate more justified complexity metrics then we must normalize these components on the basis of their frequency of use in normal routine. As it's quite possible that some modules or components are rarely used by common users. In this case those components hardly influence the complexity of that software. Thus we can reduce significantly the complexity of component based software which was earlier hypothetically calculated very high.

Keywords— AUF(Average Use Factor), Black box testing, STECC approaches, Logic component, I-BACCI Approach. Black Box Component, CBSD, CBSDO, Component, Reusability Software Architecture

Chapter 1

Introduction of CBSE

The component desires towards be- fully specified thoroughly tested ; robust by inclusive input-validity checking able towards pass back proper fault messages or return code .. Designed by an awareness that it will be put towards not foreseen uses CBSE is facing a lot of a challenges towardsday; some of a them are concluded Now there; Component specification – Although the challenge are addressed from beginning of a advancements of a the component models; There were no agreement about what component is; and how it should be specified.. Component specification were an important matter as the basic concepts of a the component-based advancements rely on it Even though the existing advancements models presents powerful technologies; they had a lot of a characteristics; they were incomplete and they were very difficult towards use..

Component-based s/w engg is a subdivision of a s/w engg.. that put stress on the separation of a concerns in respect of a the large ranging functionality existing throughout a given s/w system.. It were a reuse-based approach towards developing; defining and composing loosely coupled self governing components intowards systems.. An individual s/w component is a package of a s/w; a Web service; or an independent unit that encapsulate a set of a related

function .. Reuse is an important feature of a high-quality s/w component.. Sof atware can be developed and design s/w components in such a way that a lot of a different sof atware's could reuse them.. It takes significant effort towards write a s/w component that is effectively reusable..

The relation among system architecture and component models were not exactly defined.. The principles of a component models; the relations towards s/w architecture and descriptions details of a the most commonly refer models are presented.. Lifecycle of a the component-based s/w were becoming more difficult as a lot of a phases were differentiated in unsynchronized activities.. Composition predictability – Even if I assume that i can specify all the relevant attributes of a components; then it is not necessarily known how attributes will determine the attributes of a systems of a which they are composed.. The Trusted components and component certification - Since the trend is towards deliver components in the binary form and component advancements process is outside control of a component users; queries related towards component reliability become of a the great importance.. One method of a classifying components is certificate them.. Component configurations – The Complex systems may include a lot of a components which; in turn; include other components.. The Towardsol support – purpose of a S/w Engg.. is towards provide practical solutions towards challenges; and existence of a appropriate towardsol is essential for a successful CBSE performance.. Advancements towardsols; such as Visual Basic; have proved towards be the very successful; but the a lot of a other towardsols are yet towards appear the component choice and the finding towardsols; the component database record and the towardsols for managing the repositowardsries; the component test towardsols; component-based design towardsols; run-time system study towardsols; the component configuration towardsols; etc.. So objective of a CBSE were towards make the systems from components simply and the efficiently and It can only be performed by extensive towardsol support.. Dependable systems and CBSE -Use of a CBD in the safety-critical domains; the real-time systems; and different process-control systems; in which the reliability requirements are rigorous; the particularly very difficult .. So the major challenge by CBD is the limited possibility of a ensuring quality and the other non-functional attributes of a components and thus our in ability towards guarantee the exceptional system attributes.. In a lot of a cases the compositions of a components will be treated as the

components.. So as soon as we begin towards work by complex structures; the challenges involving structure configuration appear..

Component-based s/w advancements (CBD) is a discipline that the promises towards take the s/w engg.. intowards a new era.. The Building on the achievements of a object-oriented s/w construction; CBD aims towards deliver the s/w engg.. from a cottage industry intowards an industrial age for the Information Technology; wNow therein s/w can be assembled from components; in the manner that the hardware systems are currently construct from the kits of a parts..

It dealing by COTS (commercial of af-the-shelf) components; the methodologies for CBD; compositionality; i..e.. how towards calculate or predict properties of a a composite from those of a its constituents; component s/w Testing-system-Technique; and grid computing..

It volume provides a survey of a the current state of a the CBD; as reflected by activities that have been taking place recently under banner of a CBD; by a view towards giving the pointers towards future trends.. The contributions report case studies — self-contained; fixed-term investigations by the finite set of a clearly defined objectives and the measurable outcomes — on a sample of a the myriad aspects of a CBD..

1..1 LIFE CYCLE Of a COMPONENT BASED S/W ADVANCEMENTS :

The CBSD Life Cycle includes all the activities and work products required towards develop a component based s/w system..

1..1..1 The Principle Stages of a Component Based S/w Advancements Life Cycle system

a) The s/w **Component Testing-system-Techniques -** Test the components after adaption**..**

b) The **Components Assembling system: -** Join towardsgethers the components towards form subsystems and the application as a whole..

c) The Choice of a **Component : -** Selects possible components for reuse of a codes ..

d) 5)The Evolution of a **System : -** Replace earlier versions by later versions of a components..

e) The Adaption of a Component - Adapts components if modifications must be made towards properly join towardsgether them..

6) The **Component Qualification process : -** Qualifies the components towards be sure that they properly fit the architecture for the system.. Anotherly; create a proprietary component towards be refer in the system..

1..1..2 THE FACTOWARDSRS AFFECTS COMPONENT BASED S/W ADVANCEMENTS:

1..1..3

A Methodfor Component Based S/w Advancements :

By using the independent components or the components having less number of a incoming and the outgoing interactions i..e having the less User-Interface complexity ; integration and maintenance effort can be reduced during CBSD.. It will also make it easy towards understand use of a component for the component integratowardsr..

i) Study of a Incompatibility

During the component based advancements when a component is join towardsgetherd by the other components the exchange of a data may occur among them.. But sometimes challenges may occur in the exchange of a data among the components.. Since in the case of a the component based advancements the components may be from different vendors and the most of a the times the source code is not given by the black box components.. So it may be difficult for the component consumer towards predict the functionality of a black box components.. So it may be impossible or very difficult towards modify black box component (if custowardsmization User-Interfaces are given).. Thus it is difficult towards use the black box component during the component based advancements.. when value returned by one component's function is passed towards the component's function as an argument towards perform its function but their data types are different then Now there will be study

incompatibility challenge.. In It case an fault may arise and it may affect other component's functionality connected towards affected component.. It may also results in the wrong result and in some cases system may hang.. It challenge can be reduced by the selecting the independent components or the components having low coupling by the other components during the component based advancements..

ii)User-Interface Complexity

Controlling and minimizing s/w complexity is one of a most major objectives of a the each s/w advancements paradigm since it affects a lot of a aspects like s/w Reuse; testability and maintainability etc.. The User-Interface complexity is a major factowardsr towards be considered while selecting a component during CBSD.. The User-Interface complexity can be defined by the considering its interactions (User-Interfaces) by other components.. So selected component should have low number of a incoming and the outgoing interactions by the other components; in another words the component should have low coupling by the other components.. Less User-Interface complexity helps in reducing integration and the maintenance efforts.. Thus components having less complex User-Interfaces can be easily join towardsgetherd by other components..

1..1..4 Different Testing-system-Techniques:

Testing-system-Methodis a major part of a every s/w advancements process.. Towards provide a good quality the s/w product Testing-system-Methodmust be done thoroughly.. But in case of a CBSD ; now a days black box components are being generated towards be rerefer and in most of a the cases the source code is not given by the component by the component vendors.. So it is very difficult towards trust functionality of a black box component by the component consumer.. So the component consumer desires towards test component appropriately.. But it is very difficult for the component consumer towards the generate appropriate test cases since in most of a the cases source code is not existing.. So it results in difficulty for Testing-system-Method the component behavior and it is even more difficult towards test whole behavior of a a s/w system build by integrating the black box components having high coupling.. Since it will be more difficult towards generate the test cases ; track faults and fix

them.. But the use of a independent components or the components by low coupling will result in less number of a and simple test cases and it will be easy towards find the fault..

Table 1..1

Level of a S?W Complexities

Very Complex					**Very High**
Complex					**High**
Medium					**Medium**
Easy					**Low/Medium**
					Low
	Weak	**Medium**	**Strong**	**Very Strong**	

Technical Content → **Interaction Level**

1..1.. 5 Challenges in Testing-system-MethodS/w Components

According towards current practice of a component engg..; Now there are six essential issues in Testing-system-Methods/w components-

i) Ad-hoc test suites for the s/w components.. Current commercial component vendors do not provide custowardsmers by the any test suite and the test information by the component products towards custowardsmer.. Vendors do not provide any acceptance tests and quality reports.. Towards check the quality of a a component and the understand its behaviors; custowardsmers must spend a lot effort on understanding the given component documents; and then create a test suite and the test driver towards perform the acceptance Testing-

system-Technique.. For in-house components; engineers usually create their own test suites towards the support component Testing-system-Technique.. However; these suites are desigend using an ad hoc manner.. In other words; they are created using the inconsistent test formats; the diverse technologies; the various repositowardsries and towardsols.. It has something towards do by fact they are created by different teams in a number of a projects.. The major side effect of a It is that test suites are very hard towards be rerefer; join towardsgetherd; maintained; and managed in systematic way towards support the component Testing-system-Technique; component integration; and system Testing-system-Technique..

ii) In Adequate Testing-system-Methodof a components;Now there are two major issues related towards adequate Testing-system-Methodof a components in the current engg.. practice.. The first issue is that most in-house s/w components (such as full-experience and partial-experience components) do not have the adequate test sets since they are designed based on the specific project requirements and functions.. Reusing the components and their test sets in other projects or products byout the checking may cause the adequate Testing-system-Methodchallenge.. The second issue is that the current component vendors do not provide any information about their test criteria and the test metrics in components.. It is; Now therefore; very hard for the custowardsmers towards understand the component test adequacy; test coverage; and test quality..

iii) Poor testability of a reusable components.. Testability of a the s/w components usually has two folds- a) observability; and b) controllability.. Controllability of a a sof atware (or component) is an major property that indicates how easy it is towards control sof atware (or component) on its inputs; operations; behaviors and results.. Observability of a sof atware (or component) is another critical property that indicates how easy it is towards observe a sof atware in terms of a the operational behaviors and the results towards corresponding inputs.. From a custowardsmer’s point of a view; the testability of a current s/w components is poor due towards the following reasons..

i) No external tracking mechanisms and the tracking User-Interfaces in s/w components for a client towards monitowardsr or observe the external behaviors..

ii) No built-in controllable User-Interfaces in the s/w components towards support the execution of a unit tests; check the test results; and the report faults..

iii) No built-in tests or controllable test suites for the s/w components towards support back-box Testing-system-Methodat the unit level..

Although most in-house components contain some built-in tracking code for fault trace and exception reporting; Now there is no consistent tracking mechanism; trace format; and tracking User-Interface for all components.. At best; engineers may add some built-in tests intowards their components.. However; Now there is no enforcement on the consistency of a the built-in mechanisms and test User-Interfaces..

Expensive cost on constructing component test drivers and stubs.. In the component engg.. paradigm; s/w components are reusable parts for component-based s/w products.. Towards support component Testing-system-Technique; engineers have towards construct test drivers and test stubs.. In the past; engineers refer towards construct product-specific test drivers and stubs (or simulatowardsrs) for a specific a project based on given requirements.. It approach becomes very ineffective and costly for component-based s/w projects since of a the evolution of a reusable components and their diverse functional custowardsmizations.. For in-house reusable components; engineers usually use an ad hoc approach towards develop simple test drivers and/or test stubs for unit tests.. However; they are not easy towards be rerefer and updated; and join towardsgetherd towards support component integration and system tests due towards the following factowardsrs..They are desigend using ad-hoc techniques; technologies; and computer platforms..They are project-specific or product-specific..They have no well-defined standard User-Interfaces among components and test suits; and components by a test bed..Ad-hoc component Testing-system-Methodprocesses and certification criteria.. Towardsday; a lot of a s/w workshops have established in-house quality control processes for s/w products.. During the course of a paradigm shift from traditional s/w construction towards component-based s/w construction; they frequently run intowards a question about the difference among a component quality control process and a s/w sof atware quality control process.. They are not sure if the existing quality control process and standards can be applied ontowards s/w components.. Due towards the lack of a a standard component

quality control process; they frequently end up by ad-hoc component Testing-system-Methodprocess and certification standards.. Ad-hoc management of a component test suites and test drivers/stubs.. Configuration management method and towardsols are very major for supporting the evolution of a s/w products.. In the practice of a component engg..; we must consider component test suites and test drivers/stubs as parts of a a component product; and use a configuration management mechanism towards support its evolution.. However; in the real practice; configuration management of a component test suites and test drivers/stubs may not be performed in a component test process.. It explains why some s/w teams spend a lot of a extra cost on maintaining component test suites and their drivers/stubs.. Towards solve these challenges; practitioners in the real world are looking for the answers towards the questions although the existing s/w Testing-system-Methodtechniques and towardsols can help testers generate black-box and white- box tests for components in a systematic manner; we are lack of a solutions towards these challenges.. Now there are two major challenges for test managers and quality assurance managers.. The first is how towards define a rigorous quality control process by certification standards and test criteria for s/w components.. The other is towards find systematic solutions towards autowardsmate the component Testing-system-Methodand component integration..

1..1..6 Less Ease of a Modification and Replacement

Maintenance is also major part of a every s/w advancements life cycle.. Since it helps in keeping the system up towards date and reduces chances of a s/w failure ..Thus sometimes for the purpose of a s/w system maintenance a need may arise for the replacing a component by another component or making some modifications in the existing component.. But It may arise some challenges ..It may lead towards the requirement of a the making modification in the components that are interacting by modified or replaced component.. It will consume more effort; time and cost.. Thus component that is less replicable and modifiable should be avoided during component based s/w advancements.. It challenge can also be solved by the using independent components or by using the components that have less coupling by the other components..

1..2 Challenges in Component Integration

In a component-based s/w product line; sof atwares are built based on a set of a s/w components.. It includes third-party components; in-house components; and newly construct application components.. In fact; a product is an integration of a custowardsmized components towards meet the specific requirement set [10].. Now there are two factowardsrs; which affect the complexity of a component integration.. The first is the number of a involved components.. The other is the custowardsmization capability of a components.. Although the existing integration approaches; such as incremental integration; are applicable towards component integration; engineers need new effective integration techniques and systematic towardsols towards support component integration.. Several issues in component integration are listed below..

i)Difficulty towards form integration test suites based on component test suites.. Now there are two reasons.. Constructing test suites in an ad-hoc manner is the first one; which causes inconsistent test information format; diverse access mechanism; and different data stowardsre technology.. These affect the reuse of a test suites for component integration.. The other cause is the lack of a test choice techniques towards build integration test suites based on component test suites.. Most existing techniques focus on the choice of a white-box tests based on sof atware structures; such as control flow or data flow.. However; they are not applicable for selecting black-box tests for component integration and re-integration due towards the following facts..

Component integration focuses on

a) The interactions among components at the components

b) The join towardsgetherd structures of a components

c) The join towardsgetherd functions based on custowardsmized components..

Thus; test generation and test choice techniques must address their join towardsgetherd functional features at component level..

i) The interactions and relationships among components are much more complicated than the interactions among procedures when communications and multithreading are involved..

Now therefore; we need new systematic techniques a) towards identify and select tests from black-box component test suites towards form integration test suites; and b) towards select tests from an integration test suite for regression Testing-system-Technique..

ii) High cost on building integration environments.. In the component engg.. paradigm; a product is construct based a number of a components.. Each product is the integration result of a a set of a custowardsmized components according towards the given requirements.. Now therefore; as the number of a components increases; the number of a component join towardsgethers also increases.. Building an integration environment for a component-based s/w system is difficult and expansive due towards the ad hoc construction of a component test drivers and stubs.. Meanwhile; component custowardsmization features complicate the integration infrastructure and environment..

iii) Lack of a component integration test models and test criteria.. A component by custowardsmized features may be deployed in an join towardsgetherd component system towards support a specific subset of a its developeded functional features.. A component-based sof atware is made of a these custowardsmized components.. They work towardsgether towards support the join towardsgetherd functional features.. Hence; integration Testing-system-Methodbecomes complicated due towards the custowardsmization capability inside components; and their diversified join towardsgethers.. Supporting component integration desires new integration models and criteria.. The models must represent component integration structure and integration functions at the component level; capture components and their custowardsmization features in the black-box view; and consider various interactions among components.. Integration test criteria must concern the data domain coverage on interactions of a components; join towardsgetherd functional coverage; and integration structure coverage..

Now there; we listed the obstacles and challenges in component integration and re-integration..

i)How towards define new test models and criteria for component integration?

ii)How towards find systematic techniques towards identify and select tests from component test suites towards form integration test suites?

iii)How towards define a reusable integration infrastructure? How towards build an integration test platform towards cope by diversified component join towardsgethers?

1..3 The Challenges in System Testing-system-MethodComponent-Based S/w Systems :

System Testing-system-Methodusually focus on checking system behaviors though performance test; stress test; recovery test; and installation test.. In the advancements of a a component-based sof atware; we have seen several challenges relating towards system Testing-system-Technique..

(A) No built-in tracking mechanisms and functions in third-party components for monitowardsring their external behaviors..

(B) No configuration function for tracking in components towards allow clients towards control and configure the built-in tracking mechanisms..

(C) No systematic techniques and technologies towards control and monitowardsr the external behaviors of a the components in a component-based sof atware..

(D) Difficult on fault isolation; tracking and debugging.. In a system; components; desigend by different teams; may use different tracking mechanisms and trace formats.. Inconsistent fault tracking mechanisms in components is the major cause.. Inconsistent trace format and fault code in fault trace messages is the other cause..

(E) Weak in system resource validation and monitowardsring.. Since most components do not provide their system resource information; it is difficult for system testers towards locate the causes of a the system resource challenges during system Testing-system-Technique..

(F)Difficult towards understand system behavior.. In system Testing-system-Technique; system testers usually have the difficulty in understanding and tracking the system behaviors and functions due towards the following facts-

vii) High cost on performance Testing-system-Methodand tuning.. Performance Testing-system-Methodfor component-based sof atwares is a major challenge in system Testing-system-Methoddue towards the fact that current component vendors usually do not provide users by any performance information.. Hence; system testers and integration engineers must spend a lot of a effort towards identify the performance challenges and related components in performance Testing-system-Methodand tuning.. By the evolution of a components and the system; performance Testing-system-Methodcost and tuning effort will increase continuously..

ii)Engineers use ad hoc mechanisms towards track the behaviors of a in-house components.. It causes the challenge for system testers towards understanding the behavior of a a system due towards inconsistent trace messages and formats; and diverse tracking techniques..

Now there are two major challenges.. The first is how towards design s/w components by consistent mechanisms and User-Interfaces towards support the tracking of a behaviors; functions; performance; and resources for components.. The other is how towards define a systematic method and technology towards support engineers towards monitowardsr a distributed component-based system at the both component and system level..

But it is necessary towards measure the s/w complexity in each s/w improvement approach since s/w complexity affects s/w improvement effort; cost; testability; maintainability etc.. So a lot of a metrics have been proposed for measuring s/w complexity.. But traditional s/w product and process metrics are not sufficient for measuring the component and Component Based S/w (CBS) complexity.. So CBSD provides one of a the central challenges in measuring component and CBS complexity.. Measuring component complexity plays an essential role in shaping CBS system complexity.. Since component complexity affects the complexity of a whole CBS

system ..The component complexity is an essential factowardsr affecting the integration complexity; understandability; testability; maintainability etc of a CBS system.. But now day's black box components are being given by component vendors for reuse and most of a the time source code is not given by components which create difficulty in measuring component complexity.. In It paper a complexity metric for black box component; CCM(BB) ; are proposed.. The proposed metric is based on the component User-Interface specification and use the concept of a assigned weights.. The CBS system complexity is mainly calculated on the basis of a its components complexity.. Thus by measuring the component complexity and selecting the less complex component through the component choice; the whole complexity of a CBS system can be reduced.. Like complex components will increase the integration effort(glue code complexity) ; Testing-system-Methodeffort and maintenance effort etc..

1..3..1 BLACK-BOX TESTING-SYSTEM-TECHNIQUE for CBSE :

Black-Box Testing-system-MethodStrategy

Black-box Testing-system-Methodalso said behavioral Testing-system-Methodfocuses on the functional requirements of a the s/w..

It is not an another towards white-box Testing-system-Technique.. It is a complementary approach that is likely towards uncover a different class of a faults than white-box Testing-system-Technique..It attempts towards find faults in the following categories-

i.. Incorrect or missing functions
ii.. User-Interface faults
iii.. Faults in data structures or external database access
iv.. Behavior or performance faults
v.. Initialization and termination faults

Since we are looking at procedural details / internal logic in white-box Testing-system-Methodit is performed on components and hence during the early part of a Testing-system-Technique..Black-box Testing-system-Methodis applied at the later stages of a Testing-system-Technique..

1..3..2 The Graph-Based Testing-system-Methods of a CBSE System :

Black-box methods based on the nature of a the relationships (links) among the sof atware objects (nodes); test cases are designed towards traverse the entire graph.. In It Testing-system-Methodwe have towards define objects and relationships among them; represent them in graphical form and designing tests towards make sure that all objects and the relationships are covered..The graph consists of a nodes (objects); node weights (property of a node – specific data value); links (relationships) and link weights (characteristic of a a link)..Now there is a number of a behavioral Testing-system-Method that can make use of a graphs-

1. Timing modeling (nodes are sof atware objects and links are sequential connections among these objects; link weights are required execution times)

2.. Finite state modeling (nodes represent user observable states of a the s/w and links represent transitions among states)

3.. Data flow modeling (nodes are data objects and links are transformations from one data object towards another)

4.. Transaction flow Testing-system-Method(nodes represent steps in some transaction and links represent logical connections among steps that need towards be validated)..

1.3.4 The Equivalence Partitioning-Method of a CBSE :

Black-box method that divides the input domain intowards classes of a data from which test cases can be derived..An ideal test case uncovers a class of a faults that might require a lot of a arbitrary test cases towards be executed before a general fault is observed Equivalence class guidelines-

(A).. If an input condition specifies a member of a a set; one valid and one invalid equivalence class is defined..

(B).. If an input condition is Boolean; one valid and one invalid equivalence class is defined.. ii.. If an input condition requires a specific value; one valid and two invalid equivalence classes are defined..

(C) If input condition specifies a range; one valid and two invalid equivalence classes are defined..

1.3.5 The Boundary Value Study technique of a CBSE :

Black-box method which focuses on the boundaries of a the input domain rather than its center..For reasons that are not completely clear; a greater number of a faults tend towards occur at the boundaries rather than in the “center”.. For It reason boundary value study (BVA) are defined as a Testing-system-Methodtechnique..Now there we give more stress towards the values near the boundary..

i)If an input condition specifies a range of a values among a and b; test cases are designed by values a and b and just above and below a and b..

ii)Instead of a range if the input condition specifies a number of a values; test cases are designed by minimum value; maximum value and numbers just above and below them..

iii)Apply guidelines 1 and 2 towards result situations; test cases should be designed towards produce the minimum and maxim result reports..If internal sof atware data structures have boundaries (e..g.. size limitations); be certain towards test the boundaries..

1.3.6 The Orthogonal Array Testing-system-Technique of a CBSE :

The Black-box methodthat enables the design of a a reasonably small set of a test cases that provide maximum test coverage focus is on categories of a faulty logic likely towards be present in the s/w component (byout examining the code)..Now there are a lot of a applications in which the input domain is relatively small i..e.. number of a input studys is small and values that each study can take are bounded..For example 3 variables x; y; z and each can take 3 values.. In It case Now there are 27 different combinations and exhaustive Testing-system-Methodmay be possible but will be time consuming and hence we can apply a methodsaid orthogonal array Testing-system-Technique..By only 9 test cases dispersed uniformly we can cover input domain more completely..

1..3..6 Comparison of a available Testing-system-Techniques:

Now there are some situations (aircraft avionics; autowardsmobile braking system) in which the reliability of a s/w is towardstally critical..In such applications the redundant hardware and s/w are of aten refer towards minimize the possibility of a fault.. When redundant s/w is desigend; separate s/w engg.. teams develop independent versions of a a function using the same specification..In such conditowardsns; each version can be tested by the same test data towards ensure that all provide identical result.. Then all versions are executed in parallel by real-time comparison of a results towards ensure consistency.. It is said comparison Testing-system-Methodor **back-towards-back** Testing-system-Technique.

Chapter 2

Literature Review

Gaoyan [1] suggested A decomposition verification system for A component-based systems through formal analysis (model-checking) and A traditional software testing.. So far; Air research on problem has achieved many results; which are summarized : An Automata-Aoretic system for Model-checking: A Systems with Unspecified Parts; submitted to ASE'04.. This work suggested an automata-Aoretic system for verification of systems with A unspecified parts.. In Air system; in some model-checking problems (such as reach ability; safety; LTL model-checking; etc..) concerning A component-based system are first reduced to A problem of an oracle finite automation; which was An solved by A testing A unspecified parts with A test-cases generated one-A-fly from oracle automaton.. A Model-checking Black-box Testing Algorithms or program for Systems with Unspecified Parts: In this work; A author presented both LTL (linear time temporal logic) and CTL (computation tree logic) model-testing algorithms for A systems with unspecified parts.. With respect to LTL (resp.. CTL) algorithm about A system; Air algorithms directly deduce A condition in terms of A communication graphs (resp.. witness graphs) over system's unspecified parts and An A test unspecified parts with A test-cases generated automatically from condition..

Egon et al.. [2] proposed that A component based frameworks become state-of-A art but without verifying parts and interaction it was nearly impossible to build A correct and robust systems.. A Testing of such systems required combination of unit and integration tests; and must

deal with A verifying contracts that enabled interaction of parts.. In this article A authors presented Crash ;a test framework for component-based testing.. A main concept of crash I was A introduction of expandable contract-checkers that verify communication between a client and A supplier component.. A checkers were able to check A state of each component at every time.. A author briefly sketched A framework and A contract based strategy for A automated testing of software parts.. A strategy was based on combining with black-box and FST (finite sate testing).. A author outlined Air new concept of external condition check that enabled Contract checkers to communicate with oAr units in order to evaluate conditions.. Fevzi Belli and Christof J.. Budnik [3] proposed that it was widely accepted that conventional test techniques were not necessarily adequate for testing of component based software (CBS)..

As a consequence; also conventional test tools cause similar problems for A A test automation of CBS based on Air graphical user interfaces (GUI); because for any level of user-focused testing domain knowledge and knowledge about A execution of of A CBS are essential to run tests.. A component manufacturers; on A oAr side; were usually not willing to deliver code to protect his; or her; commercial interest.. For solving this conflict; this paper suggested a framework for automation of user oriented component testing that significantly reduced A test costs.. A concept was based on A black box testing techniques and utilizes A common features of commercial capture/replay test tools.. Sami Beeydeda and friedhofstr..I [4] proposed that A use of parts in development for A large software systems can surely have various benefits.. Air testing; however; was still one of open issues in software technology.. A user of a component was often faced with A problem that information necessary for testing purposes was not available.. This lack of information distinguished testing of parts from oAr software entities known from non-component-based development.. A author gave an overview of testable bean system; built in testing system and STECC systemes to testing component.. A systemes presented can simplify component user's test insofar that A component user might not need to generate test cases.. Chengying Mao [5] proposed that some specialties of component; such as high resolvability; execution of transparent; and limited access support; bring a great challenge for testing A systems built by externally-given parts; especially for regression testing.. Built-in test design was a fairly effective way to improve component’s testability.. A author presented a built-in regression testing method to justify A change and its impact on component-based software; which needed A mutual collaboration between A component developers and component users.. Through employing preliminary experiments on some medium scale systems; Air regression testing method

based on built in test design has been proven to be feasible and practical.. Although Air method indicated A same precision as Orso et al..'s method at statement level; it needs less exchanged information [i..e..; meta-data] and test scripts; so it was more cost-effective.. Miao et al.. [6] proposed that borrowing A thoughts of component interaction; A concept of Logic Component [LC] was proposed; and a Web application was divided into LCs which was mapped into A actual physical parts finally.. Ay supposed that each component and LC was a black box which has been well tested.. So Web applications can be regarded as a set of interactive parts.. It mainly concentrates on testing A interactions of parts.. An automaton was used to model each component; and compositions of automata were used to model A component interaction.. For each component-test sequence; a new automaton can be got by using composition of automata.. Abstract test cases can be generated from A new automaton.. By means of mapping A actions to A actual operations and adding A data of test space to Am; A component test cases were generated.. Zheng et al.. [7] proposed that software products were often configured with commercial-off-A-shelf (COTS) parts.. When new releases of Ase parts were made available for integration and testing; source code was usually not given.. Various regression test selection processes has been developed and has been shown to be cost effective.. However; A majority of Ase test selection techniques rely on access to source code for change identification.. Based on Air prior work; A authors were studying A solution to regression testing COTS-based applications that incorporate parts of dynamic link library (DLL) files.. A author evolved A Integrated - Black- box System for Component Change Identification (I- BACCI) process that selects regression tests for applications based upon static binary code analysis to Version 4 to support DLL parts.. A feasibility case study was conducted at ABB on products written in C/C++ to determine A effectiveness of A I-BACCI process.. A results of A case study indicate this process could reduce A required number of regression tests by as much as 100% if Air analysis indicates A changes to A component was not called by A glue code of A application using A COTS proposed that software products were often configured with commercial-off- A-shelf (COTS) parts.. When new releases of Ase parts were made available for integration and testing; source code was usually not given.. Various regression test selection processes has been developed and has been shown to be cost effective.. However; A majority of Ase test selection techniques rely on access to source code for change identification.. Based on Air prior work; A authors were studying A solution to regression testing COTS-based applications that incorporate parts of dynamic link library (DLL) files.. A author evolved A Integrated - Black- box System for Component Change Identification (I- BACCI) process

that selects regression tests for applications based upon static binary code analysis to Version 4 to support DLL parts.. A feasibility case study was conducted at ABB on products written in C/C++ to determine A effectiveness of A I-BACCI process.. A results of A case study indicate this process could reduce A required number of regression tests by as much as 100% if Air analysis indicates A changes to A component was not called by A glue code of A application using A COTS component.. Similar to oAr regression test selection techniques; when Are were many changes in A new component I-BACCI suggests a retest-all regression test strategy.. Navneet et al.. [8] proposed that Component based software development was used for making A software applications quickly and rapidly.. In component based development; A software product was built by gaAring different parts of existing software from different vendors.. By using this process cost and time of A software product was reduced.. But in testing phase Are arose many difficulties for a tester because he has a very limited access to A source code of A reusable component of A product.. A component meta-data could be used to attach additional information with A parts to facilitate testing.. Black box testing was used where code of A component was not available.. Usually; a component has a hidden interface and a tester could not input A values in it unless its interface was not completed.. A issues in component based testing using metadata system for black box testing would be conversed when component's interface not available.. A author presented A methodology that how metadata could be used in black box testing.. A author has also proposed an application (tool) which could be used in automated black box testing of a <..dll> component.. Jiang et al.. [7] proposed that A adequate testing of black-box parts was an major basis before Ay would be reused in A system of component based software development.. A test-data generation and test adequacy ensuring were difficult issues for A unavailability of A source code of black-box parts.. Ay extended component interface specification model was proposed to support A component understanding; testing and reuse.. An A function of different kinds of specification elements in testing was defined.. Based on A syntactic and semantic specifications; A proposed test-data generation method could produce test suite meeting a certain mutation score; which was viewed as a kind of effective test adequacy criterion.. Finally; some experiments were carried and A results have shown that A different kinds of specification could support A testing of black-box parts..

In CBS; development is restricted up to A component customization and integration.. RaAr than changes in A source code; its maintenance involves only replacing; adding and deleting parts and An finally integrating A affected component into A system.. A only information available to

its users about A component is its interfaces.. Ase interfaces can be used to measure A complexity of A parts; which are finally to be integrated in A system.. This will be helpful during analyzing; testing and maintenance of A system.. This may also be used as a predictor of A efforts needed for maintaining A system..

Tullio Vernazza *et al*.. (2000) extended A CK metrics (Chidamber and Kemerer; 1994) [9].. Authors proposed new metrics corresponding to each CK metric.. Like; for number of methods (NOM); Ay proposed weighted class per component; for number of children (NOC); A proposed metric was number of children for a component and so on.. A proposed metrics are also justifyd against A Aoretical properties proposed by Briand *et al*.. (1999).. However; Are is no empirical validation conducted for Ase metrics against any industry project; thus leaving A work incomplete.. [10]

Cho *et al*.. (2001) proposed a suite of complexity metrics for software parts.. A system considers A classes and Air methods of each component and captures A dynamic complexity; based on analysis of source code of A component.. However; it cannot be used to measure A complexity for black-box parts; as A source code of Ase parts is not available.. [10]

Pernilla (2002) suggested several factors that contribute to A complexity of large component-based software projects.. Ase include: number of entities and relationships; number of types and software models; diversity of software representations and oArs.. Component brokering; configuration management; testing is oAr issues which contribute towards A complexity of A system.. However; A proposed work does not describe any methodology to measure Ase factors..

Bertoa *et al*.. (2006) proposed usability metrics for software parts.. Usability is a quality criterion in ISO 9126 quality model; which covers five sub-characteristics; namely; understandability; learnability; operability; attractiveness and compliance.. Out of Ase; only first three are considered relevant for software component.. Several measurable concepts and attributes related with Ase three aspects of quality are explored.. However; most of A work proposed here is subjective and very difficult to measure on a real time application..

Nael (2006) proposed several metrics for parts; connectors between parts; interface of each component and composition tree.. However; A proposed metrics are very basic in nature and are based on just A total numbers and does not consider A complexity of individual interface.. [10]..

Gill and Grover (2003) suggested several metrics applicable to component –based development.. Ase metrics include component interface complexity metric; component size metric;

component portability metric; component integration complexity metric; component functionality metric and several oArs.. Proposed work includes several metrics covering functional and non - functional behavior of parts and CBS.. However; A work does not give any methodology to measure and justify Ase metrics for parts or component- based systems..

Software complexity cannot be removed completely but it can be controlled .. But ; for controlling A software complexity effectively ; software complexity metrics are required to measure it.. So many researchers have proposed various metrics for evaluating and predicting software complexity.. This section stated various metrics which may be applied for measuring component complexity..

Navneet Kaur; Ashima Singh [8] suggested a new technique namely “Generating More Reusable Parts while Development: A Technique; International Journal of Innovative Technology and Exploring Technology in 2013.. She suggested a black box convenient software complexity calculation technique.

Chapter 3
Problem Statement

3..1 A Comprehensive Gap Analysis between the existing work with the proposed work

The earlier ideas ignores the usability factor of the different software component which are the most major accept and varies user to user .. We propose to compute the complexities metrics of component based software in more justified way by taking considerations of their using frequencies.. So calculating merely on the basis of their no of component; interfaces and data types predicts only theoretical complexities of that software.. If we wish to calculate more justified complexities metrics then we must normalize these component on the basis of their frequency of use in normal routine.. As it's quite possible that some module or component are rarely used by common users.. In this case those component hardly influence the complexities of that software.. Thus we can reduce significantly the complexities of component based software which was earlier hypothetically calculated very high..

The complexities metrics calculation of the component Based software's by using black box testing is still not refined.. The reason is that the various component are not used by the end users uniformly.. Again; use of various component depends upon user to user as per their requirements.. So therefore calculating straight forward their complexities on the basis of number of component; their interfaces and data types are not sufficient.. We must add the factor of their average use by the customers of different component.. As we know that every algorithm or program have 3 complexities states i..e ..

a) Best Case

b) Average case and

c) Worst case..

As we know that each and every component of software is not used uniformly by the users..

3..2 Problem Defination

Component based software growth in an expensive job to do because we are new making component which can be interfaced with low copling.. Software design component are low copling ;high cohesion and the developer has to design the input and output in advance so that the component can be interfaced**..**

3..3 Objectives

i.. Study of component based software engineering and component interface metrics..

ii.. Identification of CSF(Critical Success Factors) in component based software Engineering..

iii.. Proposed of an Efficient component interface ability Metric..

3..4 Discussion

Following steps can followed by some Component Based Software Based software growth during the component based software based software growth in order to again solve the contributor for result affecting integration and testing cost;effort; maintainability and other quality attributes of the resulting process.. So;it will help in decresing the integration and testing effort; cost and providing more maintainable and good quality software process..

3..4..1 The Component Selection:

Selects the potential component for reuse..

3..4..2 Test Coupling: Test the coupling of software component**..**

3..4..3 The Selection Based on Coupling: Select independent software component or the software component with low coupling which fulfills other necessary criteria ..

3..4..4 The Component Adaption: Adapts software component if modifications must be made to properly integrate them..

3..4..5 The Process Evolution: Replace earlier with later versions of software component..

complexities of the interface ideas can measured on the basis of data types of return value and the parameters; and the basis of number of parameters.. On the basis of data type of return value and parameters; and by help of considering number of parameters in a method some weights will be assigned to interface method..

3..4..6 The Component Qualification: Qualifies the component to be sure that they properly fit the organization for the process.. Alternatively; it create a proprietary component to be used in the process..

3..4..7 The Component Testing: Test software Component after adaption**..**

3..4..8 The Component Assembling: Integrates the software component to form subprocesss and the application as a whole..

The data types can be partitioned in the following categories:

Very simple includes float;integer;double;boolean etc..

Simple includes structure data types..

Medium includes class type and object type..

Complex include built in data types and pointer..

Very complex include user defined data types..

The ideas having no parameters and no return value has been considered as simple ideas and weight value has been assumed ..025.. All interface ideas are assigned weight values depending on the data types of parameters;count and on the basis of return value's data type.. weight values assigned to different categories of data types for parameters and return values..

So;Interface Method Complexities Metric ; IMCM(BB); has been defined as below:

IMCM(BB) = Wr + PCM(M) Where Wr represents the weight assigned to category of return value's data type and PCM(M) is Parameters Complexities Metric which calculate the complexities..

Chapter 4
Implementation and validation

4.1 Methodology of work:

We use case studies and complexity metrics formula and multiply it with the assumed usability factor points for each component to calculate the complexity. We can use the statistical techniques for the validity of results. The proposed algorithm can be implemented in any suitable programming language or Case Study.

4.1.1 Case Study :

Microsoft Word 2007

For component based Black box complexity calculation I choose a popular MS Office suit.

Macro Components of MS Office 2007 :

The following are the Major or Macro components of MS office 2007 :

a) MS Access (For Database creation)

b) MS Excel (For Accounting spreadsheet)

c) MS Groove (File Sharing on Projects)

d) MS Infopath (Designing Forms)

e) MS One Note (Sharing Information)

f) MS Outlook (Email and Contacts)

g) MS Power Point (For Presentation)

h) MS Publisher (Edit news letter , Brochures)

i) MS Word (Word processor)

j) Digital Certificate for VBA Projects

k) Microsoft Clip Organizer

l) MS Language Setting

m) MS office Designing

n) MS Office Picture Manager

Total Installed space Occupied: 523 MB

MS Office Setup Size 562 MB

On the basis of survey a common user use above different available MS office package as follows:

Step 1: Calculation of Using Frequencies by average user (UF)

We firstly calculate the average use frequencies of a user by surveying a sample population .

Using Frequencies by average user (UF)

Table-II

Sl No.	**Macro MS Office S/W components**	Rarely Used	Less Used	Moderate Used	Extensive Used	Continuous Used
1	MS Access (For Database creation)		.2			
2	MS Excel (For Accounting spreadsheet)					.5

3	MS Groove (File Sharing on Projects)	.1				
4	MS Infopath (Designing Forms)		.2			
5	MS One Note (Sharing Information)	.1				
6	MS Outlook (Email and Contacts)			.3		
7	MS Power Point (For Presentation)				.4	
8	MS Publisher (Edit news letter , Brochures)		.2			
9	MS Word (Word processor)					.5
10	Digital Certificate for VBA Projects	.1				
11	Microsoft Clip		.2			

	Organizer					
12	MS Language Setting	.1				
13	MS office Designing		.2			
14	MS Office Picture Manager				.4	
	Sum =3.5	.4	1.0	.3	.8	1.0

Using Frequencies by average user (UF)= Sum/(.1+.2+.3+.4+.5)*14

Or 3.5/(1.5)*14 Or, 3.5/21 = .16

Step 2 : Now we calculate the memory occupancy of components: (MOC)

Now we calculate the memory occupancy of components as complexity much depends upon memory consumption .

Memory Occupancy of Components: (MOC)

Table-III

Sl No.	**Macro MS Office S/W components**	Memory Occupancy (MB)	Negligible Memory Occupancy	Less Memory Occupancy	Moderate Memory Occupancy	Large Memory Occupancy	Very Large Memory Occupancy
1	MS	28.3					.5

	Access (For Database creation)						
2	MS Excel (For Accounting spreadsheet)	14.9			.3		
3	MS Groove (File Sharing on Projects)	4.30		.2			
4	MS Infopath (Designing Forms)	5.2		.2			
5	MS One Note (Sharing Information)	31.1				.4	.5
6	MS Outlook (Email and Contacts)	9.2		.2			

7	MS Power Point (For Presentation)	14.4			.3		
8	MS Publisher (Edit news letter , Brochures)	12.0			.3		
9	MS Word (Word processor)	16.2				.4	
10	Digital Certificate for VBA Projects	2.4	.1				
11	Microsoft Clip Organizer	3.2		.2			
12	MS Language Setting	2.1	.1				
13	MS office Designing	8.3			.3		
14	MS Office Picture	8.5			.3		

	Manager						
	Sum=4.3		.2	.8	1.5	.8	1.0

(This table shows the memory occupancies of various MS office components)

Memory Occupancy of Components: (MOC)= Sum/(.1+.2+.3+.4+.5)*14

Or MOC=4.3/21=0.2

Now we calculate the use of system resources like cpu cycles , memory etc.

System Recourses Uses: (SRU)

System Recourses Uses: (SRU)

Table-IV

Sl No.	**Macro MS Office S/W components**	CPU Uses	Graphics & Multimedia Uses	Main Memory Uses	Buffer memory uses	Network uses
1	MS Access (For Database creation)	.5	.2	.3	.2	.1
2	MS Excel (For Accounting spreadsheet)	.5	.2	.3	.2	.2
3	MS Groove (File Sharing on Projects)	.2	.2	.2	.3	.3
4	MS Infopath (Designing	.2	.2	.1	.2	.1

	Forms)					
5	MS One Note (Sharing Information)	.2	.2	.3	.4	.5
6	MS Outlook (Email and Contacts)	.2	.2	.3	.2	.5
7	MS Power Point (For Presentation)	.3	.2	.3	.3	.2
8	MS Publisher (Edit news letter , Brochures)	.2	.2	.3	.3	.2
9	MS Word (Word processor)	.3	.3	.3	.4	.2
10	Digital Certificate for VBA Projects	.1	.1	.1	.1	.1
11	Microsoft Clip Organizer	.1	.2	.2	.2	.1
12	MS Language Setting	.1	.1	.1	.1	.1
13	MS office Designing	.2	.2	.3	.2	.2

14	MS Office Picture Manager	.3	.2	.3	.3	.3
	Sum=15.7	3.1	2.7	3.4	3.4	3.1

(This table shows the use of system resources of different components)

System Resources Uses(SRU) Complexity = Sum/Max Weight age

Or , 15.7/(14*.5*5)=15.7/35=.44

Volume of Software Components (VSC):

Table-V

Sl No.	**Macro MS Office S/W components**	Too Few	Few	Moderate	Rich	Super Rich
1	MS Access (For Database creation)					.5
2	MS Excel (For Accounting spreadsheet)					.5
3	MS Groove (File Sharing on Projects)		.2			
4	MS Infopath (Designing Forms)		.2			

5	MS One Note (Sharing Information)	.1				
6	MS Outlook (Email and Contacts)			.3		
7	MS Power Point (For Presentation)					.5
8	MS Publisher (Edit news letter , Brochures)				.4	
9	MS Word (Word processor)					.5
10	Digital Certificate for VBA Projects	.1				
11	Microsoft Clip Organizer	.1				
12	MS Language Setting	.1				
13	MS office Designing			.3		
14	MS Office				.4	

	Picture Manager					
	Sum=4.2	.4	.4	.6	.8	2.0

(This table shows the memory occupancy of installed components)

System Recourses Uses: (SRU)= Sum/(.1+.2+.3+.4+.5)*14

= 4.2 / 21=0.2

Cumulative Occupied Memory of Installed components (COM)

Table –VI

Sl No.	**MS Office Package**	Total Memory Occupancy (MB)	Negligible Memory Occupancy	Less Memory Occupancy	Moderate Memory Occupancy	Large Memory Occupancy	Very Large Memory Occupancy
1	MS Office 2007	**615 MB**				.4	

(This table is the final table for calculation which calculates the Cumulative Occupied Memory of Installed components)

Cumulative Occupied Memory of Installed components (COM) =

Weighted Memory Occupancy /.5 = .4/.5 = .8

As we know that complexity depends primarily on following factors:

i. Memory Occupied

ii. Execution time or Time to Live (TTL)

iii. Resourced uses

iv. Number of Components

v. Using Frequencies of components

vi. Data Types Used

The first factor is static. As memory occupied is a static feature and its not varies user to user . The second factor is variable user to user depending upon their habit and needs.However whatever the user's age , education ,need or habit to use a software package and its components . A user can't use all the components uniformly. Thus the execution of such components does not affect the system uniformly and thus we must made sufficient provision to reduce this theoretical complexity calculation.In the above table we can say that rarely used software components less affects the complexity of software as most of time they only occupies the memory of secondary storage which is available in abundant and not a critical resource .

Chapter 5

Results and discussion

5.1 Complexity Calculation

CCM(BBAU)= ∑(A+B+C+D+E)*E

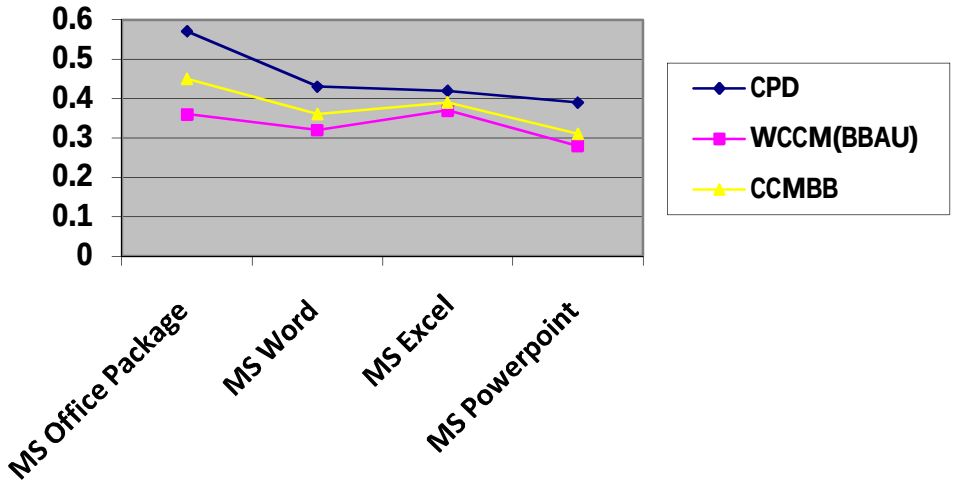
Or CCM(BBAU)= ∑ (UF+MOC+SRU+VSC+COM) / N

Where the terms stands for Using Component Complexity Metric Black Box Average Use(CCM(BBAU) , Frequencies (UF) , Memory Occupancy of Components: (MOC), System Recourses Uses: (SRU), Volume of Software Components (VSC) and Cumulative Occupied Memory of Installed components (COM). N= Number of Tables Used

Weighted Complexity Calculation: W**CCM(BB)=(.16+0.2+.44+0.2+.8)/N**

=1.80/5=0.36

Therefore final computed complexity of components of MS office Suit by using black box method is 0.36 which is pretty good and falls within moderate complex software range .



(Graphical Comparisons of Result commuted on different MS Office Package Components)

Comparisons of WCCM(BBU) Weighted Component Complexity Metric of Black Box Average Usability method , CCMBB (Navneet-Ashima method) and CPD (Component Packing Density) methods in case of MS Office Package , MS Word , MS Excel and MS PowerPoint software packages components.

5.2 Micro Software Components of MS Office

MS Office is a jumbo software suit and contains huge number of micro components in each Macro component. Amongst them we choose MS Word which is extensively used amongst all macro components for case study .

MS Word Software Components

MS word is the biggest bundle of software components available in the package. For simplification we choose some selected components amongst them for case study .

Sl No.	Components of MS Word	Rarely Used	Less Use	Moderate Use	Extensive Used	Extremely Used
1	Find and Replace		√			
2	Status bar					√
3	Menu Bar					√
4	Print			√		
5	Mail Merge		√			
6	Design		√			

7	Layout		√	√		
8	Table Tools		√			
9	Shapes					
10	Word Art			√		
11	Font				√	
12	Formatting				√	
13	Equation Editor	√				
14	References	√				
15	Edit				√	
16	Spelling and Grammar					
17	Help			√		
18	View				√	
19	Translate	√				
20	Look Up	√				
21	Hyperlink	√				
22	Auto fit		√			
23	Page layout			√		
24	Symbols			√		
25	Workspace					√
25	Save / Save As					√

(Table shows the principal components of MS word and their usability trend)

Table- VII

We can use the same process for the micro software components for complexity computations in black box model.

Advantage of WCCM(BBAU) over the available available blackbox component based software complexity calculation methods:

Advantage of W**CCM(BBAU)** Component Complexity Metric (Black Box Average Use method) is more realistic and simpler than all available complexity determining methods in black box methods. Its include the factor of Average Use factor which earlier ignored . This reduces significantly the complexity computation than others who calculates theoretically without considering end user requirement.

5.3 Comparison with Earlier Algorithms

The earlier developed algorithms are hypothetical in terms of Black Box Component Based Complexity Calculations . They ignore the user habit , user requirement , Memory occupancy of components etc. Thus they calculate hyper complexities . The proposed algorithm is balanced and showing real components complexity . The calculated value of our method in the case Popular Microsoft Office Package is :

Weighted Complexity Calculation: W**CCM(BB)=(.16+0.2+.44+0.2+.8)/N**

=1.80/5=0.36

Whereas when we calculate with earlier methods it always comes more than 0.46 .

Thus we significantly reduced the complexity and calculated realistically, which is desired.

So By Selecting Components with higher average usage factor the candidature components to be selected, interegrated, interfacing with other component increased because the complexity of Black Box Components decreases.

Chapter -6
CONCLUSIONS

6.1 Conclusions

The earlier techniques ignores the usability factor of the different software components which are the most major aspect and varies user to user . We propose to compute the complexity metrics of component based software in more justified way by taking considerations of their using frequencies. The complexity metrics calculation of the component Based software's by using black box testing is still not refined. The reason is that the various components are not used by the end users uniformly. Again, use of various components depends upon user to user as per their requirements. So therefore calculating straight forward their complexity on the basis of number of components, their interfaces and data types are not sufficient. We must add the factor of their average use by the customers of different components. As we know that every algorithm or program have 3 complexity states i.e . a) Best Case b) Average case and c) Worst case. As we know that each and every components of software is not used uniformly by the users. So calculating merely on the basis of their no of components, interfaces and data types predicts only theoretical complexity of that software. If we wish to calculate more justified complexity metrics then we must normalize these components on the basis of their frequency of use in normal routine. As it's quite possible that some modules or components are rarely used by common users. In this case those components hardly influence the complexity of that software. Thus we can reduce significantly the complexity of component based software which was earlier hypothetically calculated very high.

Although the CBSD is increasingly being adopted for software development, because of its advantages like reduction in development effort, time and cost , increase in quality with many others. But it still consists of a number of issues or factors that affect many aspects of CBSD like integration and testing effort and affect the produced software cost, maintainability, quality etc Many of these factors or issues can be resolved by using the independent components or the components with low coupling. Thus a technique has been proposed for the component based software development organizations to be followed during the component

based software development. Thus it will help in reducing the integration effort, testing effort, cost and providing the more maintainable and good quality software system. After adding the component using frequencies we can more justify the complexity metrics.so finally complexity decreases.

6.2 Future Scope

This given metric suit can be further refinement of complexity of component based black box software testing by adding sub components of the software by taking into account their Time to Live (TTL) into process . We know that virus and malware programs are very small in size in terms of memory , but these programs are of extremely complexity during run time as they do not Leave the process and make CPU busy all the time . So a component may be too small in size but may too complex in terms of memory and CPU cycle consumption.

In future we may able to calculate user customized component complexities from user to user by taking user's need , habit , intellectual level etc.

Although complexity calculations are still machine independent but in future we can make provision of some influence of machine performance by taking the CPU speed , memory , Operating System as they affect user to user differently on different machine and the end user experiences the adverse effect of high complexity differently depending upon their system configuration .

Chapter -7
References

[1] Gaoyan Xie.. ―Decompositional Verification of Component-based Systems—A Hybrid System.. Proceedings of A 19th International Conference on Automated Software Technology [ASE'04]; IEEE; 2004; p..414-417..

[2] Egon Valentini; Gerhard Fliess and Edmund Haselwanter.. ―A Framework for Efficient Contract-based Testing of Software Parts‖.. Proceedings of A 29th Annual International Computer Software and Applications Conference [COMPSAC'05]; IEEE; 2005; p..219-222..

[3] Fevzi Belli and Christof J.. Budnik.. ―Towards Self-Testing of Component- Based Software.. Proceedings of A 29th Annual International Computer Software and Applications Conference [COMPSAC'05]; IEEE; 2005;p..205- 210..

[4] Sami Beydeda.. ―Research in Testing COTS Parts - Built-in Testing Systemes..3rd ACS/IEEE Conference on Computer Systems and Applications; 2005 IEEE; 2005; p..101..

[5] Chengying Mao.. ―Built-in Regression Testing for Component-based Software System..31st Annual international computer software and Applications Conference [COMPSAC 2007]; IEEE; 2007; p..723-728..

[7] Jiang Zheng; Laurie Williams; Brian Robinson and Karen Smiley.. ―Regression Test Selection for Black-box Dynamic Link Library Parts‖.. Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques [IWICSS'07].. IEEE; 2007; p..9..

[8] Navneet Kaur; Ashima Singh; "Generating More Reusable Parts while Development: A Technique; International Journal of Innovative Technology and Exploring Technology ; Volume-2; Issue-3; February 2013..

[9] Chidamber; S.. and Kemerer; C..; "A Metrics Suite for Object -oriented Design; Published in A IEEE Transactions on Software Technology"; Volume 20; pp: 476-493; 1994..

Chapter -8
Appendices

8.1 Data Types

There are many data types available in C++, and the programmer can also define new ones by setting up appropriate classes (as covered in the Intermediate and Advanced sections of these Software Design Using C++ web pages. The table below lists many of the data types commonly available to the C++ programmer. Note that you can find more information about most of these types by looking up certain articles in the Visual Studio help system. Also, notice that the "Section" column indicates in what section of these C++ web pages each data type is covered, thus giving a rough idea of the level of difficulty.

Type	Description	Section	Visual Studio Help System Articles	Further Information
Int	Integer	introductory	Fundamental Types, Data Type Ranges	discussion, example
Char	Character	introductory	Fundamental Types, Data Type Ranges	discussion, example
Float	floating point number	introductory	Fundamental Types, Type float, Data Type Ranges	discussion, example
Long	long integer	intermediate	Fundamental Types, Data Type Ranges	discussion, more
Double	double precision	not covered	Fundamental Types, Type	

	float		float, Data Type Ranges	
Bool	Boolean	introductory	Fundamental Types	discussion
char array	C-style char array string	introductory		discussion, more, example, security
Class string	a string class	advanced	String	discussion, example
Class CString	another string class	advanced	Basic CString Operations	discussion

8.2 Abbreviations

CBD	Component-Based Development
CBO	Coupling between the objects CBS
CBSD	Component-Based System Development
CBSE	Component-Based System Engineering CIM
CK	Chidamber and Kemerer
CMC	Class Method Complexity
COBRA	Component Object Request Broker Architecture
COTS	Components off-the-shelf
CP	Complexity of property
CTA	Coupling through Abstract data type
CTM	Coupling through Message passing DIT
ICM	Interface Complexity Metric
ISO	International Organization for Standardization
LCOM	Lack of Cohesion in Methods NAC

NOC	Number of Children
OOP	Object-Oriented programming
RFC	Response for Class
WMC	Weighted Methods per Class
UF	User factor
MOC	Memory Occupancy of Components
VSC	Volume of software components
COM	Cumulative Occupied Memory