Customization of Black Box Complexity Metric for Component Based Softwares

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

Master of Engineering

in

Computer Science and Engineering

Submitted By

Deeksha Rani

(Roll No. 851232002)

Under the supervision of:

Mr. Vinod Kumar Bhalla

(Assistant Professor in CSED)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

June 2015

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Customization of Black Box Complexity Metric for Component Based Softwares"**, 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 Asst Prof. Vinod Kumar Bhalla 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: Deeksha Rani

(Deeksha Rani)

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

Vinod K Bhalla

(Asst Prof. Vinod Kumar Bhalla)

(Assistant Prof. in CSE)

Countersigned by:

(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

Thapar University, Patiala

(Dr. S.S. Bhatia)

Dean (Academic Affairs)

Thapar University, Patiala

Acknowledgement

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

I must convey my gratitude to my supervisor Mr. Vinod Kumar Bhalla , Assistant Professor ,CSE department for giving me the constant source of inspiration and help in preparing the thesis, personally correcting my work and providing encouragement throughout the thesis.

I am especially thankful to Dr. Deepak Garg, 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

Deeksha Rani

Abstract

The generalized model of complexity calculation does not work properly for component based software as this calculates only theoretical complexities of software. In practical this hypothetical model does not works properly as the different end user's needs are different so the use the various components differently.

Although complexity calculations of component based software are machine independent but it clearly the influence on performance of software on various system components capabilities like CPU cycle, Architecture, available memory etc.

Secondly all components of software are not used uniformly by all the users. This again leads to user based customization of complexity rather than generalized complexity calculation.

This thesis analyzes and interprets the impact of different end users on component based software. This thesis also analyzes the impact of hardware capabilities on component based software. A significant reduction in complexity is found over the earlier methods.

We propose CCCM(BBCBS) Customization of complexity calculation method for Black Box Component Based Software. In this method cumulative complexity of the software is determined.

The thesis successfully demonstrates and reduced the complexity metric of component based software by eliminating complexities of unused or rarely used components. This reduce the complexity metric of software component significantly and computes more realistic complexity results as compared to earlier methods which includes all the components as participating for complexity calculations .

Keywords: UFDU; CBSD; CCCM(BBCBS), HCW, CC.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Component Based Software Engineering

Component-based software engineering (CBSE) manages deve1opmet of software system by integrating already build software components. It is a reuse-based approach. In this approach various components are gathered from different sources and then integrated by using well defined architecture [25]. An individua1 software component is group of software, Web administration and various free units. Reusability of software components is a chief trait[8]. Software components should be designed in such a manner that their reuse is high. Then again, it requires major effort to compose such a software component so as to be reused adequately. The developer should document the components completely and validate it by testing the component with suitable input so that component is free from errors.

Presently days, CBSE is confronting numerous difficulties, out of which some are concluded here. Component detail is the first challenge. Component specification is a chief matter as the crucial concepts of the component development depends on it. Despite the fact that the offered development model presents great technologies and they had numerous characteristics but they were exceptionally hard to utilize. Lifecycle of component based software are flattering more difficult as lots of stages were separated in unsynchronized tasks. Creation consistency is another challenge [11]. Regardless of the fact that the software engineer indicates all the applicable traits of components, but it is redundant that these properties are adequate for the framework for which component is utilized. As the component is developed by one person and used by some other person. So it is very difficult to judge whether component is trust worthy or not. One method of classifying components is certifying them. A component configuration is also an issue. The macro component contains many micro components. So within such type of systems structure configuration problem occurs. Software engineering gives practical solutions to problems and for successful CBSE performance existence of appropriate tool is essential. One of the examples of such a tool is Visual Basic. It has proved to be the extremely

successful. The purpose of this approach is to make the systems from different components proficiently. This is possible by using wide-ranging tools. Use of component based development in the critical fields, real-time applications etc is mainly challenging. Such systems demand thorough reliability. So the key hitch with component development is the partial prospect of guaranteeing quality.

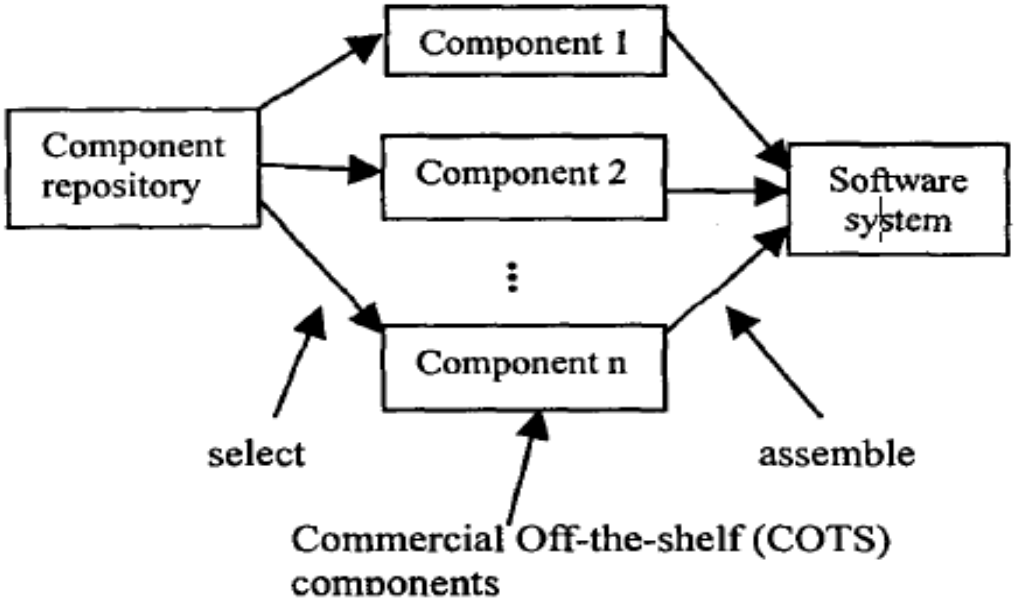
Component based software development(CBD)

Component based software development is an approach that guarantees to acquire the software engineering into another period [14]. It builds on the success of object oriented software development. Component based development plans to hand out software engineering into an industrial era. Here software is constructed by integrating various components, in the manner, the hardware frameworks are developed from various parts.

In CBD the programmer uses already existing components to fulfill the desired function which is required in the application. For example developer created a component named "Sign Up". Then other programmers can use this component for their applications. As shown in the figure below, there is repository of components where all the components are placed. The programmer can take the suitable component from this repository according to the application need and then develop the software system by assembling these components.

It manages commercial off-the-shelf components, a method used for CBD. That is how properties of a combination can be predicted from its constituents.

Fig.1: Component Based Development



1.2 Component Based Software Development Life Cycle

The component development life cycle includes all the activities and work products fundamental to build up component based software system.

1.2.1 Main Phases of Component Based Software Development Life Cycle

i) **Component Selection:** Selects appropriate component for reuse.
ii) **Component Qualification:** Qualifies that the component properly fit into the architecture of the system.
iii) **Component Adaption:** Adapts components if modifications are needed so that components can be properly integrated.
iv) Component Testing: Test each component after adaption[3]**.**
v) **Components Assembling:** Integrates or merge the components to structure subsystems and to build an entire application.
vi) **System Evolution:** Replace former versions with new versions of components.

1.2.2 Factors Affecting Component Based Software Development

i) Parameter Incompatibility

In CBD when components are integrated with other components, exchange of data occurs between the components. But sometimes this may cause some problems. As most of the software vendors provide b1ack box components which do not contain source code. So it may be hard for the component purchaser to predict the working of black box components. It is also very difficult to perform modification in b1ack box component. Thus it is not easy to use b1ack box component during software development, when value returned by one component's procedure is delivered to another component's procedure as a parameter. The problem of parameter mismatch occurs, as their data types are different. This is called parameter incompatibility dilemma. In such a scenario an error may occur and it might affect other component's working connected with that component.

ii) Interface Complication

It is extremely important to control and minimize complexity of software as it influences several different perspectives like software reusability, testability and maintainability etc. During CBD while selecting a component we should always consider its interface complexity [22]. It is stated by taking into account component communications with the other components. Selected component ought to have minimum number of incoming and outgoing connections with the other components. It is supposed to have low coupling between components. If interface complexity is less then it will help in reducing assimilation and the repairs efforts.

1.2.3 A Technique for Component Based Software Development

In CBSD, integration and maintenance effort can be reduced significantly by taking components having less no. of approaching and the outgoing communications. It will too make it easy to understand use of component for the component integrator.

1.2.4 Testing

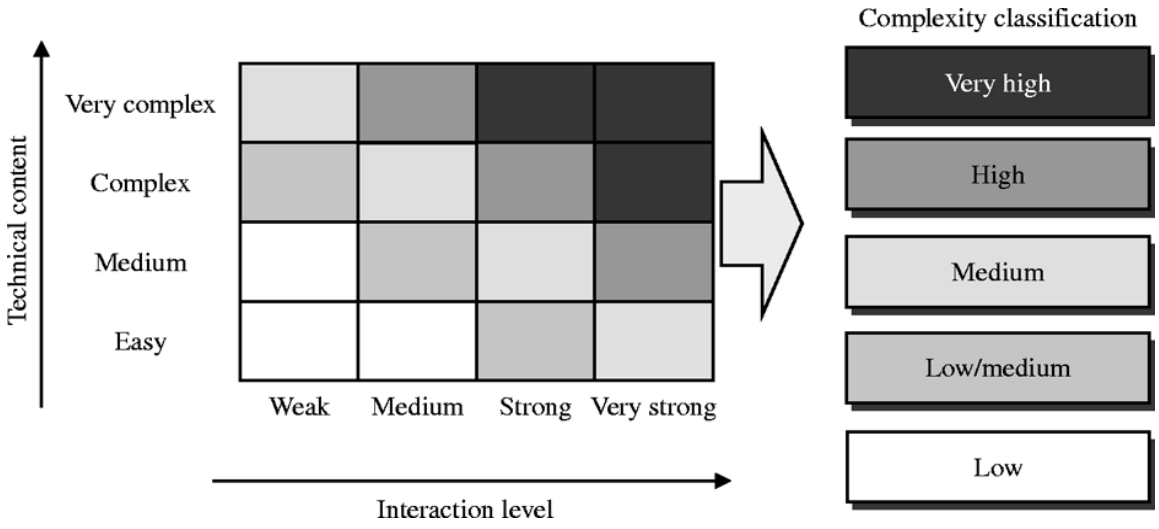
In every software development process testing plays an important role. For creating good quality software product's testing must be done thoroughly [4]. At this moment b1ack box components are being generated for the purpose of reuse and most of the component

vendors do not provide source code with the component. So component consumer can't trust functionality of black box component. So the component consumers have to test each component properly. Due to unavailability of source code it is very complex for the component purchaser to generate appropriate test. So it results in difficulty for testing the component behavior and also causes difficulty in testing the whole behavior of a software product. Even it will be harder to create the test cases, track errors and fix them. So this difficulty can be removed by using components having low coupling. Via this simple test cases can be generated and it will help in finding the errors.

1.1.5 Less Ease of Modification and Replacement

To keep system up to date maintenance of software system is very important. It reduces chances of the software failure. While performing software maintenance, sometimes we have to replace the existing component. But if components are coupled with some other components, so this may lead to make modifications in the interacting components. This will also consume more effort, time and cost. So the component that is less replicable and modifiable should be avoided during component based software development. Thus by using the components having less coupling with the other components or by using independent components, this problem can be solved.

Fig: 2 Complexity Classification



1.2.6 Problems in Testing Software Components

There are various issues in testing software components. These are as follows.

i**) Informal test suites -** Nowadays component provider don't give test suite and the test information about the component to the customers. Even component vendor do not any give quality report and acknowledgement test about the component. So the customers have to spend their own effort to check the quality and to understand component's behaviors. After that customer create their own test drivers and test suite to perform acceptance testing [5]. For component testing engineers usually create their own test suites by using various technologies, repositories, tools and inconsistent test formats. Generally these are formed by different members in no. of projects.

ii) **In Adequate testing of components-** For adequate testing of components we first have to deal with two key issues. As the components are developed according to the specific project requirement, so most of fully experience components don't include appropriate test sets. So reusing the components in other projects/products may cause testing problems. This is the foremost issue of testing the component.

iii) **Deprived testability of reusable components-** Testabi1ity of the software components depends upon its controllability and observability [8]. Controllability of component how easily inputs, operations, behaviors and outputs of the component can be controlled. Observability of component shows how easy is to examine a program in term of its working behavior, output to the related input. There are various reasons for poor testability of the current software. These are:

i) There is no exterior tracking mechanisms for purchaser to check the outside behaviors.
ii) There are no integral, convenient interfaces in the software components to carry out tests.
iii) There are no integral test suites for the software components to carry black box tests at entity level.

iv) **Ad-hoc component testing processes and certification criteria**:- Nowadays, to check software quality lots of internal quality control processes have been built. The

traditional software construction process has been replaced by the component based software construction[6].Quality control processes are less in number then ad-hoc methods are used.

v) **Ad-hoc administration of component test suites and test drivers/stubs**:- For supporting the evolution of software products configuration management method and tools plays very important role. That is the reason for spending extra cost and effort while maintenance of test drivers and stubs.

1.3 Problems in Component Integration

In component based software deve1opment, projects are assembled using various distinctive sorts of programming components which incorporates outsider components, in-house components, and recently built components. To meet the prerequisite set [l0] the components are accumulated from distinctive sources. There are numerous issues while performing integration of components. These issues are as follows:

i) **Difficulty to structure integration test suites from existing component test suites:** Constructing test suites from the already built component test suite are a difficult process because of conflicting test information layout, varied access method along with dissimilar data store tools. The supplementary cause is deficient of test selection techniques to construct integration test suites according to component test suites.

Integration of components mainly center on

a) The communications between components.
b) The incorporated structure of components.
c) The incorporated capacities in view of tweaked Components.

ii) **High cost on constructing integration environments:** In component based engineering, the construction of integration environment is costly process because we have to construct test drivers and stubs according to our own need. This customization property sometimes may cause problem also.

iii) **Lack of component integration test models and test criteria**: To support some specific functions the customer creates components with customized features that are later on deployed in the integrated system. A component based construction is composed of several personalized components. These components work together to provide desired features. That's why it is very complicated to perform integration testing due to customization capability [7]. New integration models and criteria are needed to support component integration. The models ought to signify integration structure and functions at component rank. Integration criteria must concern data domain coverage, integrated functional coverage and structure coverage.

1.4 Problems occur during System Testing

Testing of whole system means to check system behaviors though various tests like execution test, recovery test etc. While developing component based software program, we encompass some problems[35].

i) It is complicated to realize and track behavior of the system while system testing by the tester.
ii) Engineers usually utilize specially appointed systems to track component behaviors of internal components which create trouble for testers because of inconsistent trace messages and formats.
iii) There are no integral tracking mechanisms to check external behaviors.
iv) There is no configuration method for tracking in components to allow customers to manage and organize the integral tracking mechanisms.
v) Lack of logical techniques and advances for controlling and monitoring the external working.
vi) It is very difficult to isolate, track and debug the errors.
vii) It is very costly to run performance test. As component vendor do not provide any performance information with the component, so it is the key challenge for the system testers. They have to spend lots of efforts in identifying and isolating the errors.
viii) It is very tricky for system testers to find the reason of the system assets problems.

Software complexity be very critical issue as it affects cost, testability and maintainability. So it is important to measure the complexity while doing software improvement. There are various metrics have been anticipated for measuring software complexity. But these conventional metrics are not sufficient. Measuring component complexity is very essential while developing component based software (CBS) system. The whole CBS system's complexity depends upon the complexity of the each integrated component. It also affects understandability, testability, maintainability etc. Presently component vendors provide black box component for the reuse purpose. So measuring complexity of software component is very difficult because source code is not given with b1ack box component.

1.4.1 BLACK-BOX TESTING

Black-Box Testing Strategy

Another name of Black box testing is behavioral testing. It mainly focuses on the useful prerequisites of the product.

In this testing technique source code is not given [14]. Another type of testing is white box testing which is a complementary approach that uncovers different classes of error than this testing technique. Black box testing endeavor to discover following categories of errors:

i) Incorrect or missing capacities
ii) Interface blunders
iii) Errors in external data base access
iv) Execution or behavior blunder
v) Initialization and termination mistake

White-box testing is performed during early part of testing as procedural details are given. Black-box testing is connected to later stages.

1.4.2 Graph-Based Testing Methods:-

Graph based testing strategy use black box technique. It contains relationships (links) among the program objects (nodes).This is in form of graph where test cases are designed to traverse it [6]. In this method we have to define objects and relationship between the objects. We have to design tests in such a way that it covers the whole graph. The graph consists of nodes (objects), node weights (property of node – specific data value), links (relationships) and link weights (characteristic of a link). There are various behavioral testing functions that utilize graphs:

i) Transaction flow testing (here hubs speak to ventures in the exchange and connections speak to legitimate associations between steps).
ii) Finite state displaying (here hubs speaks to conditions of the product and connections speak to moves between the states).
iii) Data flow demonstrating (here hubs are data objects and connections are changes starting with one data object then to the next)
iv) Timing demonstrating

1.4.3 Equivalence Partitioning

Black box technique isolates the data space into various classes. These classes we can derive test cases. A perfect test tries to uncover a class of errors. For observing general error we require arbitrary test cases. There are some guidelines for equivalence classes:

i) If the data condition determines a range, then one substantial and two invalid identicalness classes are characterized.
ii) If the data condition obliges a component worth, then one substantial and two invalid identicalness classes are characterized.
iii) If a data condition indicates an individual from a set, then one substantial and one invalid comparability class is characterized.
iv) If a data condition is Boolean, one substantial and afterward one invalid proportionality class is characterized

1.4.4 Boundary Value Analysis

In black box testing technique main focus is on the limits of the data area rather than its middle because it has been observed that a more noteworthy number of blunders liable to happen at the limits instead of in the "inside". That's why boundary value analysis (BVA) has been defined as testing strategy. In this we give more emphasis to the values near the boundary.

i) If data condition determines a scope of qualities between m and n, then experiments are designed with values m and n.
ii) Instead of range if data condition indicates various qualities, experiments are planned with least esteem, greatest esteem and numbers simply above and beneath them.
iii) Guidelines given in l and 2 are applied to yield conditions; experiments ought to be intended to create the base and most extreme yield reports.

1.4.5 Comparison Testing

Comparison testing is applied in situation where reliability is very critical like aircraft avionics, automobile braking system [12]. To minimize the possibility of error in such situations the redundant hardware and softwares are often used. For this purpose separate teams of developer are deployed. Different versions are developed for the application.In this we give same input data to all versions. All versions are executed simultaneously and comparison is done. So another name given to comparison testing is **back-to-back** testing.

1.4.6 Orthogonal Array Testing

Another black-box technique is orthogonal array testing. In this we develop little arrangement of experiments that give maximum test scope. There are numerous applications in which the information area is generally little. For example 3 variables a, b, c and each can take 3 values. In this case there are 27 different combinations and exhaustive testing may be possible but it will be time consuming. So we can apply a

technique called orthogonal array testing. With only 9 test cases dispersed evenly we can cover input domain more completely.

1.5 Organization of Thesis

i. Chapter 1 discusses the basic concepts related to CBSE and black box testing so as to have a basic knowledge about these concepts.

ii. Chapter 2 gives the overview and analysis of various complexity metrics of component based software.

iii. Chapter 3 states the problem statement and research gap analysis.

iv. Chapter 4 describes the proposed algorithm-"Cumulative Complexity Calculation for Components".

v. Chapter 5 validates the proposed technique using case study.

vi. Chapter 6 concludes the work done and also stated its future scope.

Chapter 2

Literature Review

A vast literature found on the topic of component based software and its complexity metric calculation. Gaoyan[l] exhibited an Automata-Theoretic methodology for model checking. He presented the Model-checking Black-box Testing Algorithms or project for Systems with Unspecified Components: Here he displayed both LTL (direct time temporal logic) and CTL (processing tree rationale) model-testing calculations for the frameworks with unspecified programming components. The LTL (resp. CTL) recipe about the framework that straightforwardly concludes the condition as far as the correspondence diagrams and after that different tests are performed on the unspecified components. These tests are generated automatically from the condition.

The approach suggested by Egon et al. [2] He says that the without checking components and connection it was about difficult to assemble strong frameworks. The Testing of such frameworks obliged blend of different unit and coordination tests. It must manage the checking gets that empowered connection of components. The creators exhibited Crash, a test structure for component based testing named as Crash. This test system confirms the correspondence in the middle of customer and the supplier component. These checkers had the capacity check the condition of every component at each time. The technique was taking into account mix of black-box and FST (limited satiate testing).

Fevzi Belli and Christof J. Budnik [3] states that it was widely accepted that traditional test methods were not necessarily appropriate for testing of component based software (CBS) system. According to him traditional test instruments cause comparative issues for the test automation of CBS framework, on the grounds that for any level of client centered testing, area learning and information about the execution of the CBS is fundamental to run tests. The Component merchants were normally not willing to convey code to ensure his, or her, business premium. For tackling this contention, the creator presented a system for automation of client situated Component testing. This system

fundamentally decreased the test expenses. The idea was fundamentally in light of the discovery testing procedures.

Sami Beeydeda and friedhofstr [4] demonstrated that it is very beneficial to use of CBSE based softwares for the gigantic software systems as it surely have benefits for cost cutting as well as faster delivery. However, CBSE complexity remains an issue in software engineering. The customer of a component frequently faced with a trouble that information that is required is not available in general as it is black box and having different vendor. Due to absence of adequate information, it distinguishes testing of components from other software entities that is called as non-component-based development.

In another development Chengying Mao [5] suggested various features of component, like high resolvability, limited access support and implementation transparent, bring an awesome stand up to for testing the frameworks manufactured by remotely given programming Components. This is pertinent exceptionally for relapse testing. The Built-in test configuration was a very compelling approach to enhance component's trying. He exhibited an inherent relapse testing procedures to legitimize the change and its effect on component based programming. It needs common collaboration between the Component clients & Component engineer's group. Through utilizing preparatory trials on some medium scale frameworks, their relapse testing technique in view of inherent test outline has been ended up being achievable and down to earth. In spite of the fact that their technique demonstrated the same accuracy as Orso et al's, but was more cost-effective as it needs less exchanged information and test scripts.

Miao et al. [6] suggested the belief of conviction of component communication. He built up the thought of Logic Component [LC].According to his concept a Web application was isolated into LCs which was mapped into the genuine physical Components. They accepted that each Component and LC was a discovery that is all around tried. The Web applications may be viewed as a gathering of commonly intuitive software components.. To model each component, automation was used. The organizations of automata were utilized to show the Component based connection. For every component test arrangement, another machine can be utilized by piece of automata. Hypothetical experiments can be made from the more current machine. By mapping activities to the

real operations and adding the measurements of information to test space, the Component experiments were made.

Zheng et al. [7] demonstrated that software yield was designed with commercial-off-the-shelf (COTS) components. The attainability contextual investigation was directed at ABB on the CBSE programming items. It was composed in some C/C++ dialect to show the efficiencies of the I-BACCI. As a consequence of the contextual analysis indicate this procedure could cut the essential number of relapse tests by as much as l00 rates. They recommend that product items were regularly arranged with business off-the-rack (COTS) components. Source code was every now and again not given, when new arrivals of these components were made accessible for fuse and testing.

Navneet et al. [8] introduced that rapid and quick development of softwares can be made possible by means of CBSE. In CBSE, the software item was constructed by combining diverse techniques of on hand software from diverse suppliers or vendors. By method for this strategy, cost and time of the product bundle diminished essentially. However in the testing stage there are numerous difficulties for a product analyzer, because of constrained access to the source code of the reusable Component of the product item. To encourage CBSE programming testing, the Component meta-information could be utilized to join additional data with the Component. The Black box testing was utilized as a component of this technique and the code of the Component was not accessible. For the most component, a Component has a covered interface and analyzers are not ready to put information values in it until its interface was not wrapped up. The difficulties in Component based testing by utilization of metadata technique for discovery testing would be utilized when component's interface not exists. Here they showed the strategies that how the metadata could be utilized as a component of discovery testing.

Jiang et al. [9] again demonstrated that the satisfactory testing of black box software components is the key before they reuse it in the idea of component based software development. As source code of black-box components is not available so test data production and ensuring test sufficiency is difficult. They stretched out component interface detail model to bolster the Component understanding, testing and reuse. At that point the capacity of various types of specification components in testing was characterized. The proposed test-information era system could deliver test suite meeting a

certain change score, based on the syntactic and semantic details. It was seen as a sort of compelling test ampleness foundation. Last however not the minimum, numerous examinations were done and their outcomes have demonstrated that the various sorts of necessity could bolster the testing of discovery components.

McIlroy [1968] first suggested techniques Software reuse, at NATO Software Engineering Conference, he anticipated that large scale production of components of softwares leads to software crisis [36]. His goal was apparent: to craft something only one time and reuse it many time by industry for low cost.

Frakes and Terry [1996] – was first person to suggest metric and models on software reusability for low cost and rapid delivery. He recommended that models based on cost benefits, evaluating the maturity level, reclaim library metrics [33].

Kim [2005] takes – on the problem of component based software reusability for low cost production. It discusses the problems in understanding the component based software reuse and to discuss the pre – conditions required to meet before practicing softwares re-claim on a wide scale in a formal manner [35].

Sharma et. al [2007] debates about managing component – based systems with recyclable components. It converses the reusability thoughts for components based systems and to discover the reusability metrics to calculate reusability direct or indirect ways [38].

Anas al - badareen [2011] proposed a framework that covers the extraction, acceptance and storing of reusable software components [32].

Gupta and Kumar [2013] showed in a study about reusable software component recovery system. The study debates the techniques for storage and recovery of software components which can be reused by the industry [33].

In case of component based software engineering approach (CBSEA), the stress is on components while designing software. A component can be considered or assumed as a sovereign system that completes a specific assignment. A component also can be composed of several other components. More specifically software components are prebuilt substances that act as a building block in a software to perform specific functions. These components can connect with each other using standard interface. After

the solidification of component based engineering practices, notion of software recycle has progressed more significantly[37].

Reusability focuses on programming software to build software system [38]. There lies several benefits of component reuse like compact development time and cost, the quality and efficiency will also be increased as the pre verified components are being used. But the developers could not harvest much benefit from it. There may be several reasons for that but the lack of proper definition for the components which is to be reused can be the one reason. The non-availability of a recognised meaning for a component can also create misperception and misconstruction among the developers. It can be explained with an example: If one purchase a stereo music system and bring it at house. We can fit various components similar to speakers, earphones etc. into it. Each of these components has been intended to fit a definite architectural style, the influences between components are homogenous, and communication protocols have been pre-established. Assembly of such components is cool than to build a system of thousands of discrete parts.

Traditional software complexity metrics have been designed and applied for measuring the software complexity of organized framework since 1976.

One of the customary metric is Lines of Code (LOC). This metric provides assess of physical lines, statements or remarks. High estimation of metric shows more complexity[6,10].

Another metric is McCabe's Cyclomatic Complexity Metric. It considers the program graph and is defined as[10] :V(G) = e - n + 2p In this e represents the quantity of edges, n represents no. of nodes in the graph and p represents no. of linked nodes. This metric gives assess of autonomous algorithmic test paths. Additional testing effort are required if independent paths are more [22].

Halstead's Complexity Metric attempts to estimate the programming effort [10]. It measures complexity of the program by summarizing the no. of operators and operands in the program [23].

Henry and Kafura additionally proposed the complexity metric by taking into account the quantity of neighborhood data streams entering (fan-in) and leaving (fan-out) in every method. This metric is given as:

Complexity = (Proc. length) * (fan-in * fan-out) 2

Here length can be measured by any method like lines of Code, McCabe's Cyclomatic complexity etc. All these four metrics can be applied for measuring the component complexity [24].

Weighted Methods Per Class (WMC) metric is projected to count merged complexity of local methods in the given class. It is characterized as total of complexity of class's local methods[26].

Depth of Inheritance metric is used for classes. The depth of a class is defined as extreme number of ventures from the class hub to the base of the tree. It is calculated by the no. of antecedent classes [27]. High value of DIT shows high design complexity. But it represents more prominent potential for reuse of acquired strategies.

Gill and Grover projected an unpredictability metric methodology called Component Interface Complexity Metric (CICM) [7]. This approach assumes interface marks, requirements on the interfaces and bundling for distinctive connection of utilization, for deciding the multifaceted nature. A definition is additionally proposed for each of these angles. On the other hand, work still lack of any observational assessment and approval of the proposed metric.

Ardimento et al. given the study of impact of characteristics of individual components on the quality of the component based systems. Few attributes of the components are: Adequateness of the Component regarding the objective framework. It further depends upon the functional coverage, compliance of component and costs key for preparing individuals while utilizing the Component, nature of the working group with the component, level of bolster given by the component's seller and so forth. But the metrics proposed for these component characteristics are subjective and some of them have not been empirically validated which leaves the work incomplete.

Chapter 3

Problem Statement

3.1 Research Gap Analysis

The former technique ignores the usability factor of the different software components which are the most key accept and varies user to user. We propose toward finding the complexity metrics of the component based software in more justified way by taking considerations of their using frequencies. The complexity metrics calculation of the component based softwares by using black box testing is still not refined. The reason is that the various components are not used by the end user 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 are familiar with that every algorithm or programs have 3 complexity states i.e. Best, Average, Worst case. As we be familiar with that each and every components of software is not used uniformly by the user. So calculating just on the basis of their no of components, interfaces and data types predicts only theoretical complexity of that software. If we desire to calculate more justified complexity metrics then we must normalize these components on the basis of their frequency of use in normal routine. It is quite possible that some modules or components are rarely used by common user. In this case those components hardly influence the complexity of that software. Thus we can reduce significantly the complexity of component based software which is former hypothetically calculated very high.

3.2 Problem Definition

Component development in an expensive job because we are developing new components which can be interfaced with low coupled components. Software design components has low coupling, high cohesion and the developer has to design the input and output in advance so that the components can be interfaced**.** The generalized model

of complexity calculation not works properly for component based softwares as this calculates only theoretical complexities of softwares. In practical this hypothetical model does not works properly as the different end user needs are different so the use the various components differently. Although complexity calculations of component based softwares are machine independent but it is seen clearly the influence on performance of a software on various system components capabilities like CPU cycle, Architecture , available memory etc. Secondly all components of software are not used uniformly by the entire user. This again leads to user based customization of complexity rather than generalized complexity calculation. This thesis analyzes and interprets the impact of different end user on component based softwares. This thesis also analyzes the impact of hardware capabilities on component based softwares. A significant reduction in complexity is found over the former methods.

3.3 Objectives

i. To study existing approaches being proposed for black box complexity calculation of component based softwares.
ii. To propose an approach which leads to user based customization of complexity rather than generalized complexity calculation.
iii. To validate the approach using the case studies of Apache Open Office, Mozilla Firefox, VLC media player for UFDU and analyze the results.

Chapter 4

Proposed Algorithm

The proposed algorithm for cumulative complexity calculation is as follows:

Step 1: Assign weights to each system type.

Step 2: Find hardware components weightage (HCW) by calculating mean of assigned weights for each type of system.

Step 3: Calculate components using frequency by different end users(UFDU).

Step 4: Calculate component complexity weightage(CCW).

Step 5: Then calculate cumulative compressive complexity by taking mean of HCW, UFDU and CCW.

Chapter 5

Implementation and Validation

5.l Methodology of work:

In this thesis three case studies have been discussed. In this approach weights are assigned to each component depending upon the type of user. These weights are assigned based on the data collected from the survey of different types of people. Then, sum of all weight is calculated for each user type. Then Using Frequencies weightage of different user (UFDU) is calculated.

Testing tools used: **SISA (Simple Interactive statistical Analysis)**

About SISA :

SISA is a simple interactive statistical analysis tool which is analogous to SPSS. It is automated online tests can be performed by using this tool. We can test various parameters automatically even online by use of this tool.

Some of its features are: T-test, ANOVA, regression testing, standard deviation, correlation analysis. one tailed test , 2 tailed tests etc.

5.2 Case Studies :

Step 1: Assign weights to each type of system

Step 2: Calculate Hardware Component Weightage (HCW)

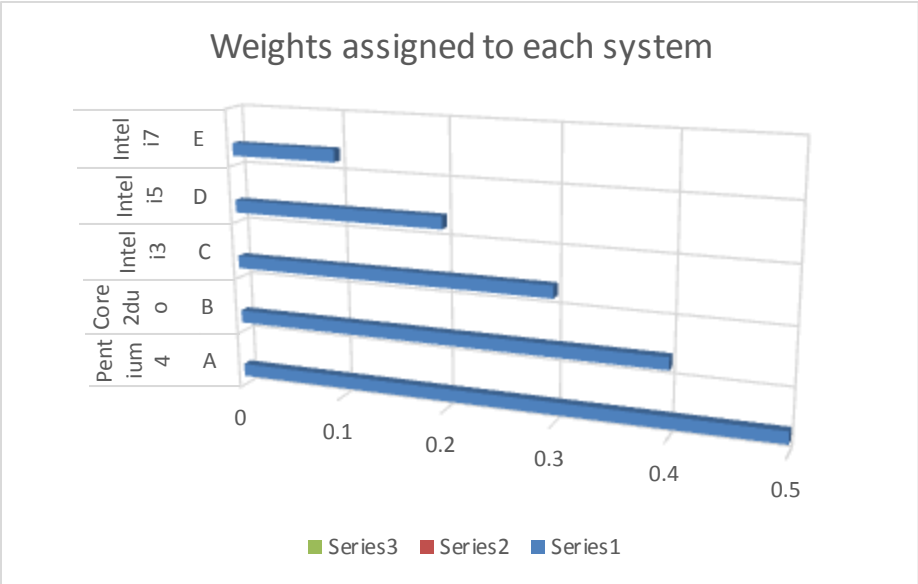
= (A+B+C+D+E)/5

= (.5+.4+.3+.2+.1)/5

=1.5/5

Hardware Components weightage (HCW) = 0.3

Fig.3: System hardware competencies Comparisons



Above figure shows system hardware competencies comparison. Figure 3 represents the weights assigned to each type of system. As shown in the figure, 0.5 weight is assigned to Pentium 4 **,** 0.4 is assigned to Core 2 duo and so on.

Table 1: Hardware Components weightage (HCW)

	System Configuration	Pentium 4	Core2duo	Intel i3	Intel i5	Intel i7
	System Level	**A**	**B**	**C**	**D**	**E**
	Assigned Weights	**.5**	**.4**	**.3**	**.2**	**.1**
1	**CPU**	1.5 GHz	2 GHz	2.13 GHz	2.6 GHz	2.6 GHz
2	**RAM**	512MB	1GB	2GB	4GB	8GB
3	**Cache Memory**	1MB	2MB	3MB	8 MB	8MB
	Sum of All Weights	**HCW =(A+B+C+D+E)/5 = 1.5/5= 0.3**				

Table 1 shows hardware component weightage. HCW is calculated by taking mean of all the weights.

Fig.4: Assigned weights to each type of system

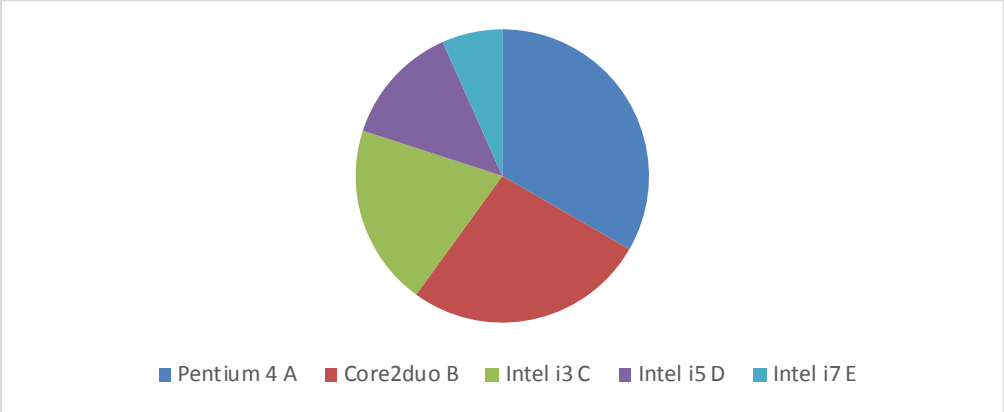


Figure 4 shows the weights assigned to each type of system i.e. Pentium 4, Core2duo, Intel i3, Intel i5, Intel i7 in form of pie chart.

Fig.5: Assigned Weights in form of bubble chart

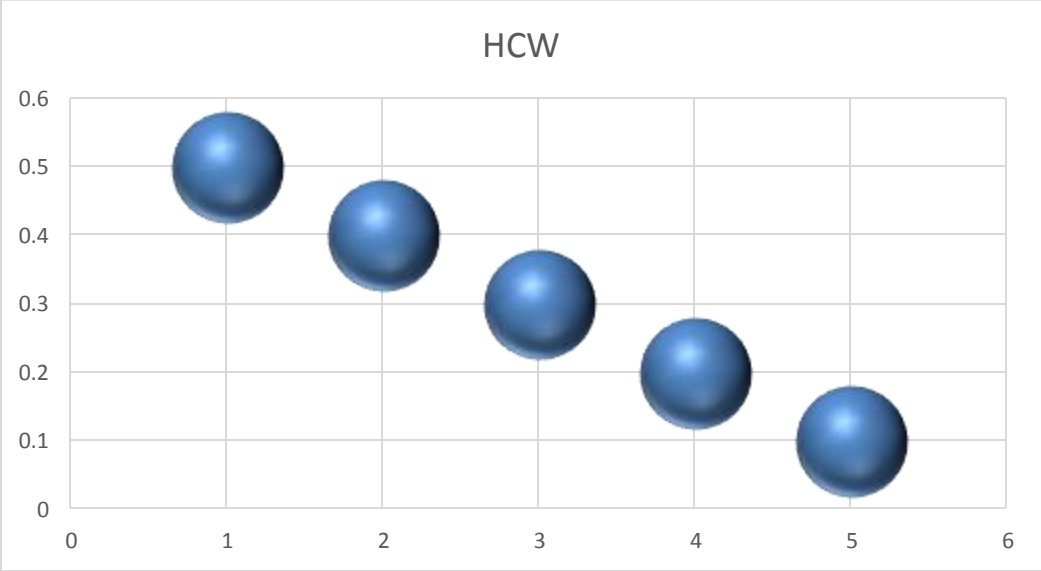


Figure 5 shows the weights assigned to each type of system in form of bubble chart.

Step 3: Calculation of Component Using Frequencies by different end user (UFDU)

We firstly calculate the average use frequencies of different user by surveying a sample population of 100 peoples having different levels and different machines.

5.2.1 Case Study I: Apache OpenOffice

About ApacheOpenOffice :

Apache OpenOffice is an open source venture. Both have been in vicinity since 13 October, 2000. OpenOffice.org 1.0 was realeased on 30 April, 2002. It offers access to APIs and a XML record position.

Fig.6: Apache OpenOffice Logo

Apache OpenOffice contains various parts.

Writer (word processor)

It is a tool used for making letters, books, reports, pamphlets, leaflets as well as different records. We can add representation along with items from diverse parts into its records. It can admission records into HTML, XML, a couple of word document versions etc. It moreover unites with our email client.

Calc (spreadsheet)

Calc contains greater part of the propelled investigation, plotting along with decision making components. It incorporates more than 300 capacities for monetary, factual, and numerical operations, among others. The Scenario Manager gives "suppose it is possible that" examinations. Calc produces 2-D and 3-D graphs that able to be joined with open office archives.

Impress (presentations)

It gives the entire normal interactive media presentation apparatuses, for example, embellishments, movement etc. It is incorporated by propelled design capacities of open office Draw and Math portions. We can also update slideshows by using font enhancements content and moreover stable and feature cuts. It works good with Microsoft's PowerPoint record design.

Draw (vector graphics)

Draw is basically a drawing device that may be used for creation of graphs, flowcharts, 3-D images etc. It has elegant Connectors that permit us to characterize our own association. We can use Draw for creation of drawings in open office's parts. Even we can also create clipart that can be added into the Gallery. It can introduce representation from numerous regular arrangements and spare them into different formats.

Base (database)

Base offers gadgets to regular database job inside a clear interface. It can make, alter structures, reports, questions, perspectives, and associations etc therefore dealing with an associated database is very easy. This macro component gives numerous new components, for example, it helps to alter connections from a graph view. It consolidates HSQLDB like social database motor. Base likewise utilizes Microsoft Access, MySQL, Oracle etc databases.

Math (formula editor)

Math is basically a formula editor. We can employ it to make difficult mathematical statements to facilitate incorporate images not accessible in regular textual styles.

Although it is most generally used to make equations within different reports, such as Writer, Impress records, Math be able to likewise function as an individual device. We be able to spare equations into standard Mathematical Markup Language position used for consideration inside site pages with different reports not made via open office.

Apache OpenOffice Components User's Trends

Calculation of Component Using Frequencies by different end user (UFDU)

Table-2 (UFDU-1)

Sr No	Apache OpenOffice S/W Package Components	New User	Home User	Mediocre User	Office User	Professional User
	User Level	**A**	**B**	**C**	**D**	**E**
1	**Base (database)**	0	.2	.3	.1	.5
2	**Calc (spreadsheet)**	.1	.3	.4	.5	.5
3	**Math (formula editor)**	.1	.1	.3	.4	.5
4	**Draw (vector graphics)**	.2	.2	.3	.4	.5
5	**Writer (word processor)**	.3	.3	.4	.5	.5
	Sum	**0.7**	**1.1**	**1.7**	**1.9**	**2.5**
	Mean =Sum/5	**0.14**	**0.22**	**0.34**	**0.38**	**.5**
	Sum of Mean values	**(A+B+C+D+E) = 1.58**				
	Mean (ABCDE)	**1.58/5 = 0.316**				

Using Frequencies weightage of different user (UFDU) Case I = 0.316

Table 2 shows using frequency by different end user. In this table weights are assigned to each component for each type of user based on the data collected from the survey. Then mean for each type of user is calculated by adding the weights of each component and then dividing it with the number of components. Then mean of all types of user is calculated.

Table- 3 : Calculation of Component Complexity Weightage for case-1 (CCW-1)

Weighta	Apache OpenOffice S/W Package Components	
	Component complexity weights	**Out of (0.5)**
1	**Base (database)**	.5
2	**Calc (spreadsheet)**	.4
3	**Math (formula editor)**	.4
4	**Draw (vector graphics)**	.5
5	**Writer (word processor)**	.4
	Sum	**2.2**
	Mean =Sum/5	**0.44**

Table 3 shows component complexity weightage. In this weights are assigned to each component depending upon the memory consumed by each component.

Cumulative Complexity (case 1) = μ= (HCW + UFDU 1 + CCW1) / 3

CC(1)= (0.3 + 0.316 + 0.44) /3 = 1.056 /3

CC(1)= 0.352

5.2.2 Case study: II Mozilla Firefox

About Mozilla Firefox :

Mozilla Firefox (referred to just as Firefox) is a free and open-source[18] web program created for Windows, OS X, and Linux, with versatile renditions for Android and Firefox OS, by the Mozilla Foundation and its auxiliary, the Mozilla Corporation. Firefox utilizes the Gecko format motor to render site pages, which executes present and foreseen web standards. Firefox was made in 2002 under the name "Phoenix" by the Mozilla group individuals who needed a standalone program as opposed to the Mozilla Application Suite pack. Notwithstanding amid its beta stage, Firefox turned out to be mainstream by its analyzers and was lauded for its velocity, security and additional items contrasted and Microsoft's then-predominant Internet Explorer 6. Firefox was discharged in November 2004,[20] and was exceedingly fruitful with 60 million downloads inside of nine months, which was the first occasion when that Internet Explorer's strength was challenged. Firefox is viewed as the otherworldly successor of Netscape Navigator, as the Mozilla Foundation was made by Netscape in 1998 preceding their procurement by AOL.

Fig.7: Mozila Firefox Logo

As of February 2015, Firefox has somewhere around 12% and 20% of overall utilization as a "desktop" program, making it, per diverse sources, the third most well known web browser. Still, the program is most prominent in a few nations, as a desktop browser[28] including Indonesia, Germany, and Iran, at 50% 44% and 37%,of the piece of the overall

industry, individually. As indicated by Mozilla, as of December 2014, there are a large portion of a billion Firefox clients around the world.

Mozilla Fire Fox : Components Use trends

Calculation of Component Using Frequencies by different end user (UFDU)

Table : 4 (UFDU-2)

Sr. No	Mozilla Firefox	New User	Home User	Mediocre User	Office User	Professional User
	User Level	**A**	**B**	**C**	**D**	**E**
1	**Web IDE**	.4	.2	.4	.4	.5
2	**Search Tool**	.4	.4	.4	.4	.5
3	**Downloads**	.3	.4	.4	.4	.5
4	**History**	.1	.2	.2	.3	.5
5	**Developer**	0	.1	.2	.2	.3
6	**Sync**	0	.1	.2	.2	.3
7	**Security**	.1	.2	.3	.4	.5
8	**inspector**	.1	.1	.1	.2	.4
9	**Web Console**	.1	0	0	0	.5
10	**Debugger**	.1	.2	.2	.3	.4
11	**Style Editor**	.3	.3	.3	.3	.4
12	**Performance Analyst**	.1	.1	.2	.2	.4
13	**Network**	.1	.2	.2	.3	.4
14	**Page source**	0	.1	.1	.2	.3
	Sum	**2.1**	**2.6**	**3.2**	**3.8**	**5.9**
	Mean	**0.15**	**0.18**	**0.23**	**0.27**	**0.42**
	Sum of Mean	**(A+B+C+D+E) = 1.25**				
	Mean	**1.25/5 = 0.25**				

Using Frequencies weightage of different user (UFDU) Case II = 0.25

Table 4 shows using frequency by different end user. In this table weights are assigned to each component for each type of user based on the data collected from the survey. Then we calculate UFDU.

Table- 5 : Calculation of Component Complexity Weightage for case-2

(CCW-2)

Sr. No	Mozilla Firefox Components	Weights
	Component complexity weights	**Out of (0.5)**
1	**Web IDE**	0.4
2	**Search Tool**	0.4
3	**Downloads**	0.5
4	**History**	0.2
5	**Developer**	0.5
6	**Sync**	0.5
7	**Security**	0.5
8	**inspector**	0.4
9	**Web Console**	0.3
10	**Debugger**	0.4
11	**Style Editor**	0.2
12	**Performance Analyst**	0.2

13	**Network**	0.3
14	**Page source**	0.2
	Sum	**5**
	Mean =Sum/14	**0.357**
CCW2 = 0.357		

Table 5 shows component complexity weightage. In this weights are assigned to each component depending upon the memory consumed by each component.

Cumulative Complexity (case 2) = μ= (HCW + UFDU 2 + CCW 2) / 3

CC(2) = (0.3 + 0.25 + 0.357) / 3 = 0.907 /3

CC (2) = 0.3023

5.2.3 Case Study III : VLC media player , An Open Source Media Player

About VLC Media Player:

VLC media player (usually known as VLC) be a convenient, free and open-source, cross-stage media player and streaming media server composed by the Video LAN venture. VLC media player underpins numerous sound and feature pressure systems and record organizations, including DVD-Video, feature CD and spilling conventions. It has the capacity stream media over PC systems and to transpose interactive media files.

Fig.8: VLC media player logo

The default appropriation of VLC incorporates an expansive number of free interpreting and encoding libraries, maintaining a strategic distance from the requirement for discovering restrictive plug-in. The libavcodec library from the FFmpeg task gives a hefty portion of VLC's codecs, however the player for the most part uses its own particular muxers, and demuxers. It additionally has its own particular convention executions. It additionally picked up refinement as the first player to bolster playback of encoded DVDs on Linux and OS X by utilizing the libdvdcss DVD decoding library.

VLC Media Player: Components Use trends

Calculation of Component Using Frequencies by different end user (UFDU)

Table: 6 (UFDU-3)

Sr. No.	VLC Media Player Components	New User	Home User	Mediocre User	Office User	Professional User
1	**Video Track**	.1	.1	.3	.4	.5
2	**Aspect ratio**	.1	.2	.4	.5	.5
3	**Crop**	.1	.1	.3	.4	.5
4	**Audio track**	.1	.1	.1	.2	.2
5	**Stereo Mode**	.1	.2	.3	.4	.4
6	**Volume**	.1	.1	.1	.1	.1
7	**Play Back**	.1	.1	.1	.1	.1
8	**Media Streaming**	.2	.3	.3	.3	.5
9	**Effects and Filters**	.3	.3	.5	.5	.5
10	**Track Synchronization**	.1	.2	.4	.4	.5
11	**Update**	.1	.2	.3	.5	.5
12	**Debug Logging**	.3	.3	.4	.4	.5
13	**Video Formats**	.2	.3	.3	.5	.5
14	**Network Streaming**	.3	.4	.3	.5	.5
	Sum = 20.2	**2.2**	**2.9**	**4.1**	**5.2**	**5.8**

	Mean =Sum/14	0.157	0.207	0.292	0.371	0.414
	Mean Sum	**1.441**				
	Mean (ABCDE)/7	**0.2882**				

Using Frequencies weightage of different user (UFDU) Case II = 0.2882

Table 6 shows using frequency by different end user. In this table weights are assigned to each component for each type of user based on the data collected from the survey. Then we calculate UFDU.

Table-7 : Calculation of Component Complexity Weightage for case-3 (CCW-3)

Sr. No.	VLC Media Player Components	Assigned Component Complexity weights
1	**Video Track**	.3
2	**Aspect ratio**	.3
3	**Crop**	.4
4	**Audio track**	.3
5	**Stereo Mode**	.4
6	**Volume**	.2
7	**Play Back**	.2
8	**Media Streaming**	.5
9	**Effects and Filters**	.5

10	**Track Synchronization**	.5
11	**Update**	.5
12	**Debug Logging**	.3
13	**Video Formats**	.4
14	**Network Streaming**	.5
	Sum	**5.3**
	Mean(μ)=Sum/14	**5.3/14 = 0.378**

Table 7 shows component complexity weightage. In this weights are assigned to each component depending upon the memory consumed by each component.

Cumulative Complexity (case 3) = μ= (HCW + UFDU 3 + CCW 3) / 3

CC(3) = (0.3 + 0.2882+ 0.378) /3 = 0.9662 /3

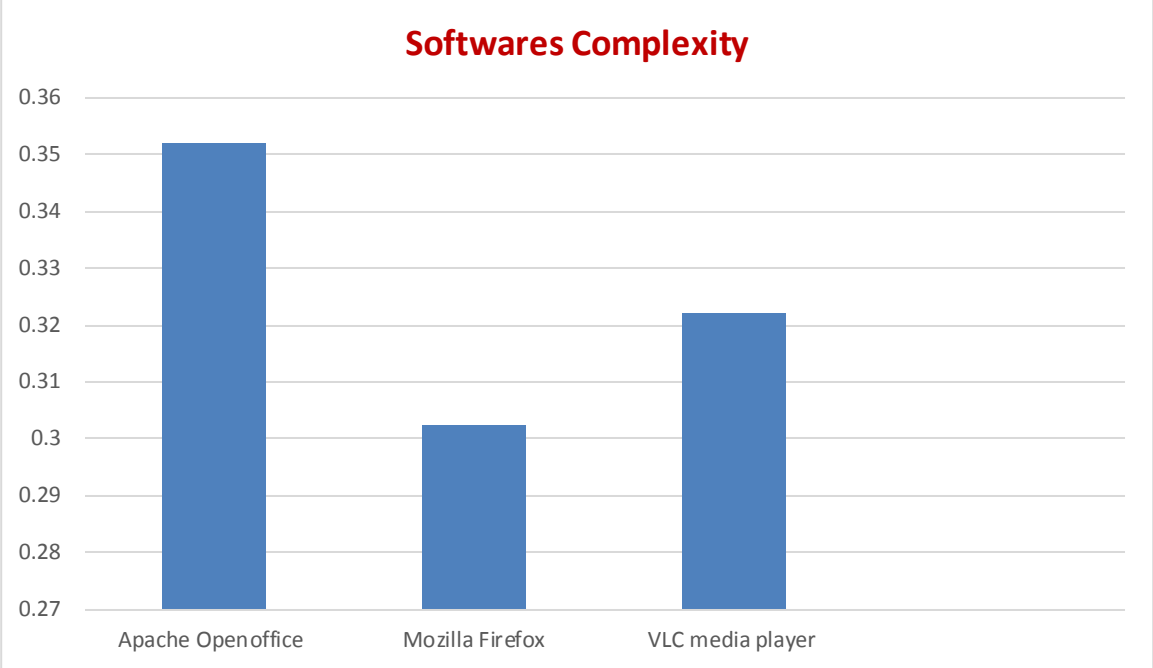
CC (3) = 0.322

Chapter 6

Result and Discussion

6.1 Result: Complexity Comparisons

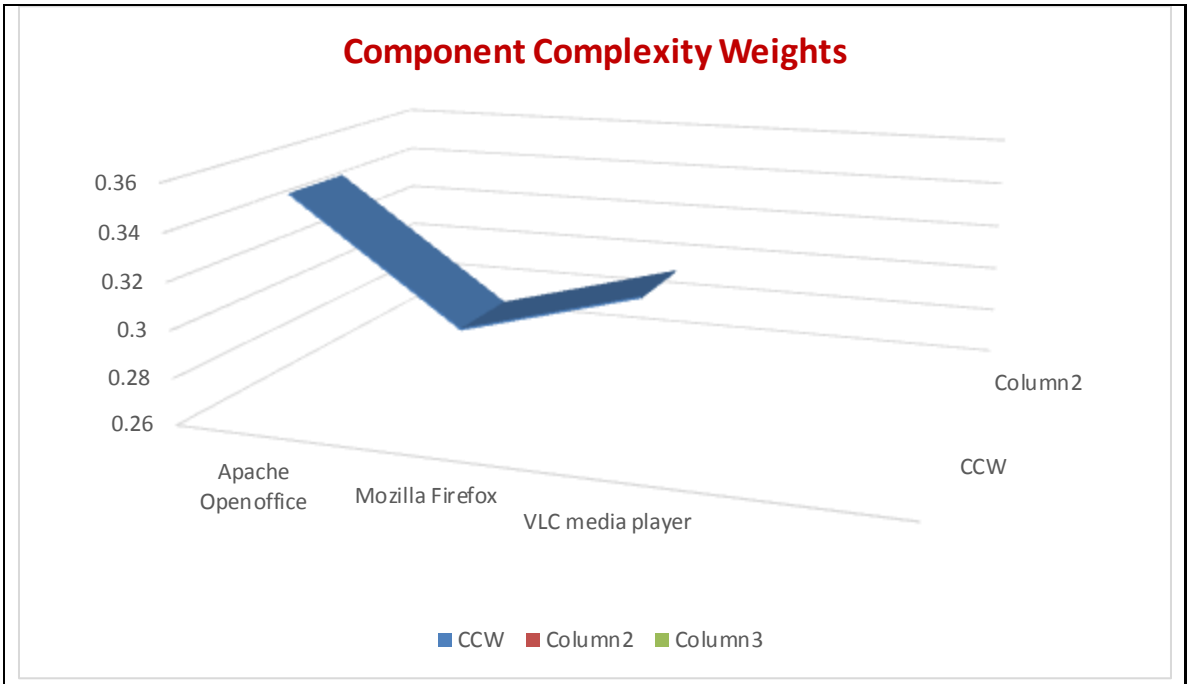
Fig.9: Complexities Comparison



		CC
1	**Apache Openoffice**	**0.352**
2	**Mozilla Firefox**	**0.3023**
3	**VLC media player**	**0.322**
Sum of CC(123) =		**0.9763**
μ of (CC 123)=		**0.3254**

Figure 9 shows the comparison of cumulative complexities of each type of software i.e. Apache OpenOffice, Mozilla Fire fox and VLC media player in form of graph.

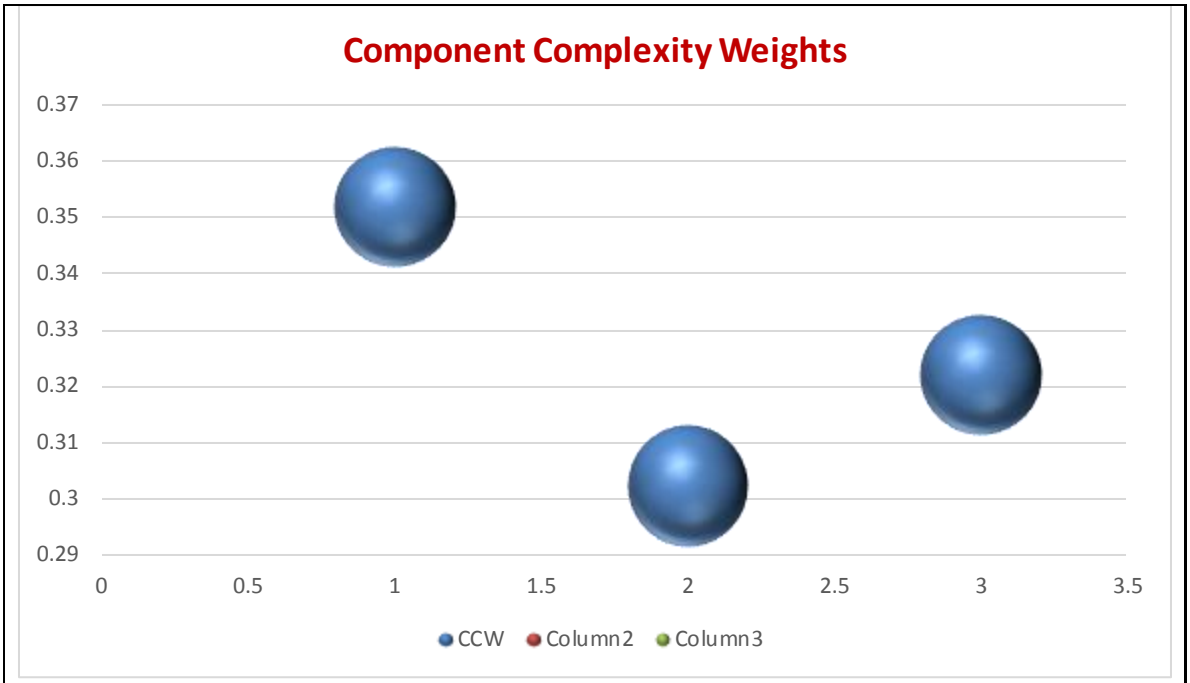
Fig.10: Complexities Comparison in form of Line Chart



		CC Weightage
1	**Apache Openoffice**	**0.352**
2	**Mozilla Firefox**	**0.3023**
3	**VLC media player**	**0.322**
Sum of CC(123) =		**0.9763**
μ of (CC 123)=		**0.3254**

Figure 10 shows the comparison of cumulative complexities of each type of software i.e. Apache OpenOffice, Mozilla Fire fox and VLC media player in form of graph.

Fig.11: Complexities Comparison in form of bubble chart



		CC Weightage
1	**Apache Openoffice**	**0.352**
2	**Mozilla Firefox**	**0.3023**
3	**VLC media player**	**0.322**
Sum of CC(123) =		**0.9763**
μ of (CC 123)=		**0.3254**

Figure 11 shows the comparison of cumulative complexities of each type of software i.e. Apache OpenOffice, Mozilla Fire fox and VLC media player in form of bubble chart.

Fig. 12 : Average CPU / Memory uses graph for an Expert user

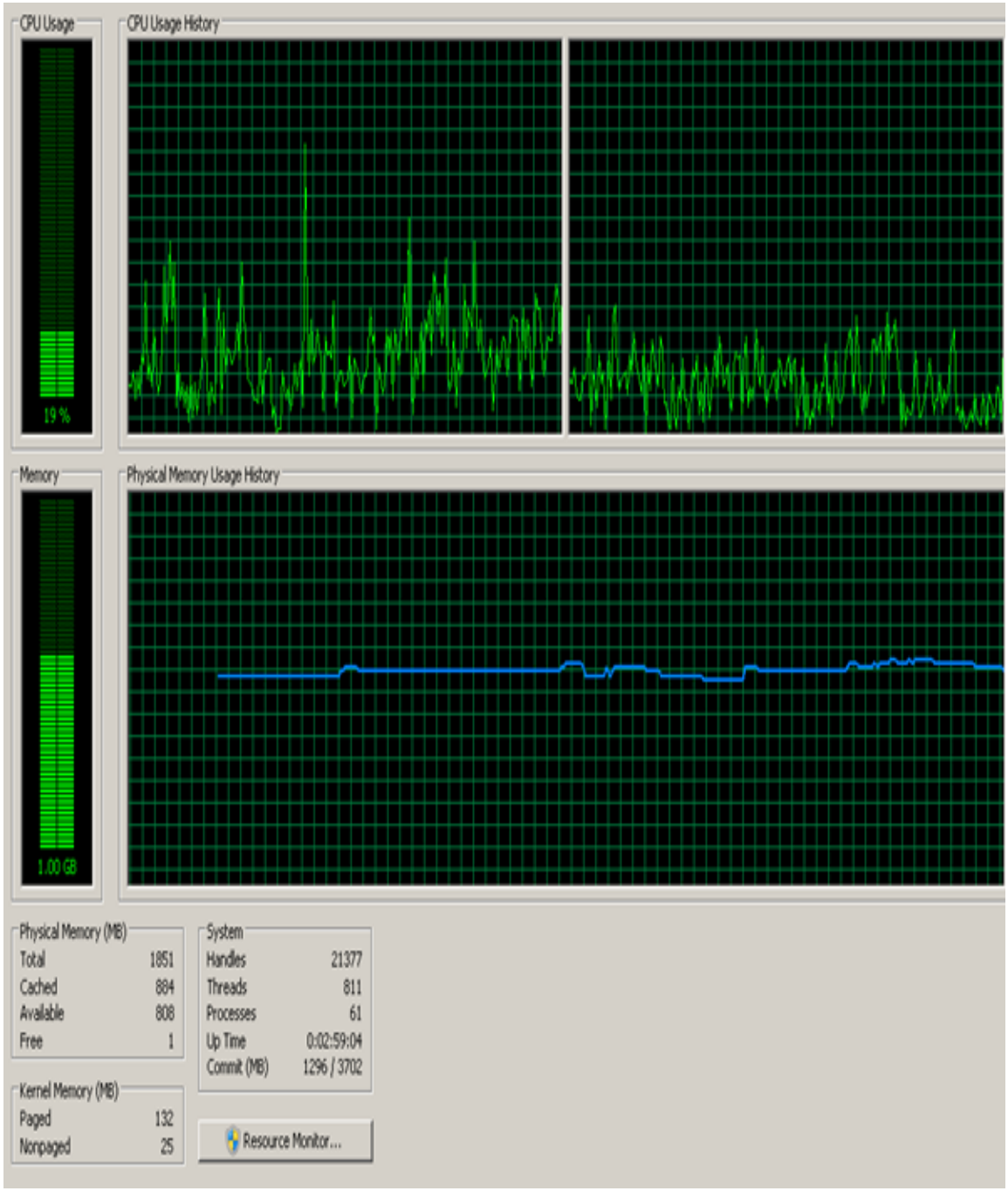


Figure 12 shows Average CPU / Memory uses graph for an Expert user. As expert user use more components of the software, so CPU and memory usage will be high.

Fig .13 : Average CPU / Memory uses graph for a Mediocre User

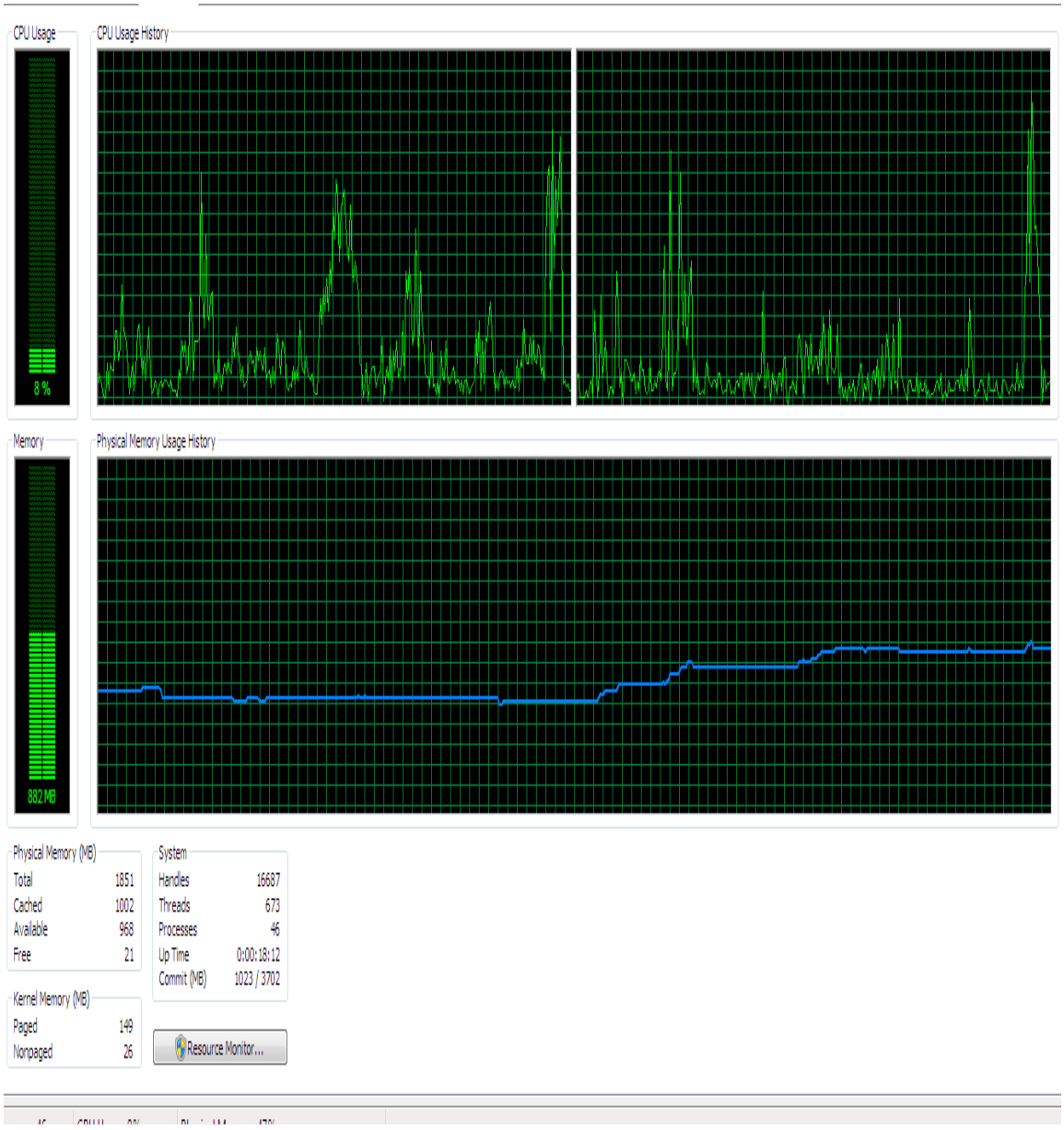


Figure 13 shows Average CPU / Memory uses graph for a Mediocre User. As mediocre user use less number of components of the software as compare to expert user so CPU and memory usage will be less as compare to expert user.

Fig .14 : Average CPU / Memory uses graph for a Learner

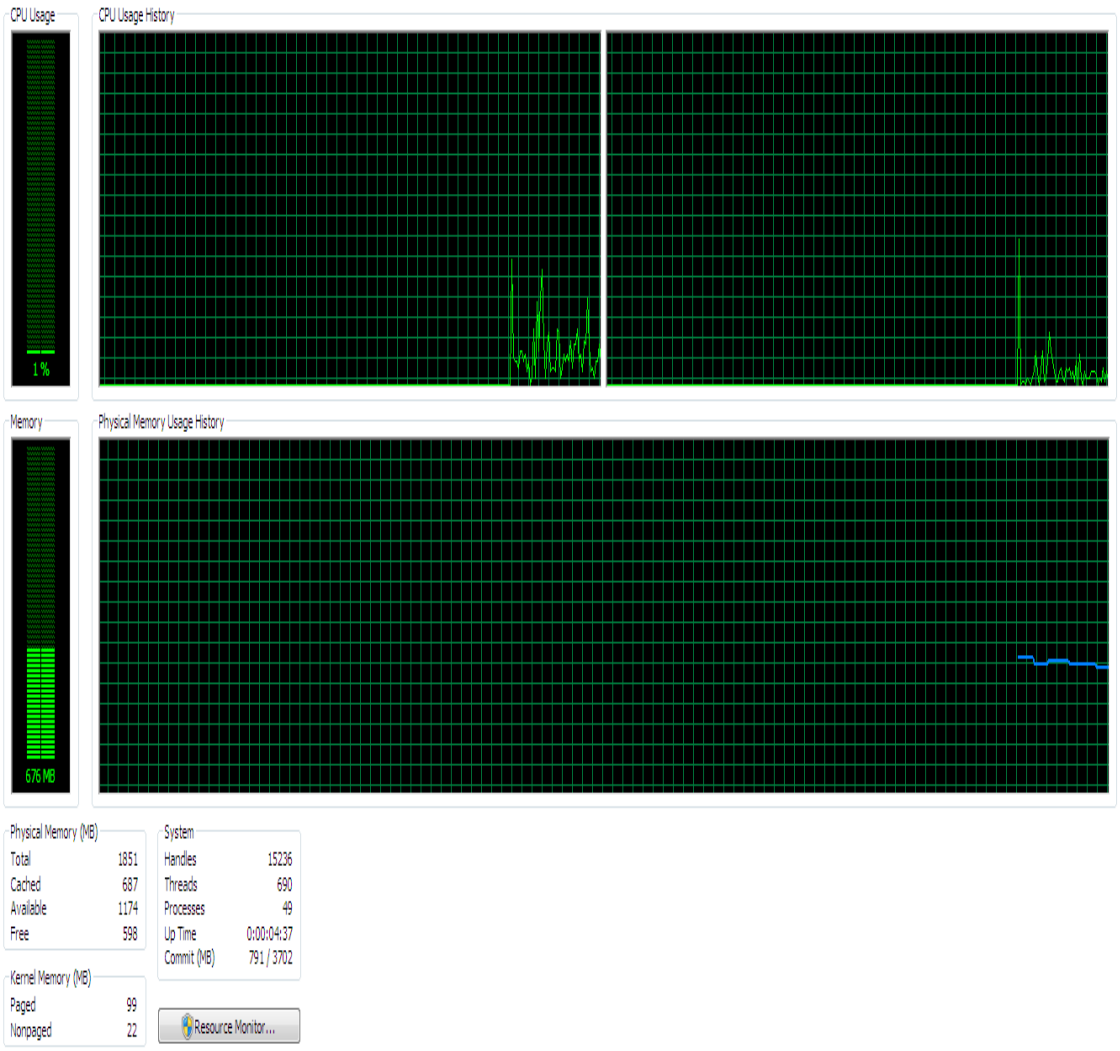


Figure 14 shows Average CPU / Memory uses graph for Learner. As learner user use few numbers of components of the software as compare to other user so CPU and memory usage will be very less.

6.2 Discussion

As shown in the above case studies that this method of complexity calculation gives the customized value of complexities of black box component. It shows that the using frequency has a greater effect on the overall complexity of the software. If using frequency is high than overall complexity will increase significantly. Despite this other factors like hardware complexity; components individual complexity also affects the overall complexity of the software. So in this method cumulative complexity is calculated by considering all the factors i.e hardware complexity, components individual complexity and using frequency of the component by different end users. So this method provides a realistic way for calculation of the complexity.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

The objective set in the Chapter 3 is fulfilled successfully. This method of complexity calculation give customized value component based software. This is more realistic and easy to calculate. The performance of hardware system affects greatly on complexities as well as the user's level of use of components also affects greatly. This method gives cumulative average complexity of components by considering system's hardware competencies, component's individual complexity and component using frequencies by different end users. After that we have taken mean of all three which gives us cumulative average complexity. Hence it is more efficient method than the earlier methods that ignore the usability factor of the component. This method computes the complexity metrics of component based software in more justified way.

7.2 Future Scope

In future we can expand this complexity calculation based on internet speed as in these days the concept of cloud computing and extensive use of internet is increasing greatly. The response time of component based softwares served through client-server model or Cloud computing is highly dependent upon the bandwidth of internet, network signal availability. So therefore the end user's experience differently about the complexity of software, as the slow bandwidth user experiences greater complexity than high bandwidth user.

Chapter -8

References

[l] Gaoyan Xie. —Decompositional Verification of Component-based Systems—A Hybrid Approach. *Proceedings of the l9th International Conference on Automated Software Engineering [ASE'04],IEEE,* **2004,** pp. 4l4-4l7.

[2] Egon Valentini, Gerhard Fliess and Edmund Haselwanter. -A Framework for Efficient Contract-based Testing of Software Components. *Proceedings of the 29th Annual International Computer Software and Applications Conference [COMPSAC'05], IEEE,* **2005,** p.2l9-222.

[3] Fevzi Belli and Christof J. Budnik. - Towards Self-Testing of Component- Based Software. *Proceedings of the 29th Annual International Computer Software and Applications Conference [COMPSAC'05], IEEE,* **2005**, pp.205- 2l0.

[4] Sami Beydeda. —Research in Testing COTS Components - Built-in Testing Approaches.3rd ACS/IEEE Conference on Computer Systems and Applications, 2005 IEEE, 2005, p.l0l.

[5] Chengying Mao. —Built-in Regression Testing for Component-based Software System.3lst Annual international computer software and Applications Conference [COMPSAC 2007], IEEE, 2007, p.723-728.

[6] Huaikou Miao, Shengbo Chen, Huanzhou liu and Zhongsheng Qian. An Approach to Generating Test Cases for Testing Component-based Web Applications.Workshop on Intelligent Information Technology Application. IEEE, 2007,p.264-269.

[7] Jiang Zheng, laurie Williams, Brian Robinson and Karen Smiley. —Regression Test Selection for Black-box Dynamic link library Components. Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques [IWICSS'07]. IEEE, 2007, pp.9.

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

[9] W. B. Frakes and K. C. Kang, "Software reuse research: status and future," IEEE Transactions on Software Engineering, vol. 31, pp. 529-536, 2005.

[10] M. Morisio , "Success and Failure Factors in Software Reuse," IEEE Transactions on Software Engineering, vol. 28, pp. 340-357, 2002.

[11] Dan laurenţiu Jişa , "Component Based Development Techniques – comparison", International Conference on Computer Systems and Technologies – CompSysTech,2004.

[12] Gill and Grover, "Reusable software components," Advances in computers, vol. 33, pp. 1-65, 1991.

[13] Sandeep Khimta, Parvinder S. Sandhu and Amanpreet Singh Brar, "A Complexity Measure for Java Bean based Software Components", World Academy of Science, Engineering and Technology, Volume 42, 2008.

[14] Richa Mittal , Ashima Singh "Refining Complexity Metrics of Component Based Softwares by taking Average Use Factor in Black Box Testing" Vol 10 no. 2 , pp 1881, 2014.

[15] M. Morisio , "Success and Failure Factors in Software Reuse," IEEE Transactions on Software Engineering, vol.28, pp. 340-357, 20l2.

[16] Jerry Gao, "Testing Component-Based Software," available at http://www.engr.sjsu.edu/gaojerry/report/star-test.htm

[17] B. Weide , "Reusable software components," Advances in computers, vol. 33, pp. l-65, l99l A Technique for Component Based Software Development l33

[18] Ivica Crnkovic, Stig larsson and Judith Stafford, "Component-Based Software Engineering: Building systems from Components," 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems, May 20l2.

[19] Gui Gui and Paul. D. Scott , "Measuring Software Component Reusability by Coupling and Cohesion Metrics", JOURNAL OF COMPUTERS, Vol. 4, NO. 9, SEPTEMBER 2009.

[20] Nasib S. Gill , "Reusability Issues in Component-Based Development", Department of Computer Science & Applications, M.D. University, Rohtak , Haryana

[21] Dogru, A.H. and Tanik, M.M. , "A process model for component-oriented software engineering ," IEEE Software, Vol. 20(2), 34-4l,2003.

[22] SONG Cui-Ye, DU Cheng-lie, "Formal Interface-Component Based Software Analysis and Design", School of Computer Science and Technology, Northwestern Polytechnic University,China,20l0.

[23] Usha Kumari and Shuchita Upadhyaya, "An Interface Complexity Measure for Component-based Software Systems", *International Journal of Computer Applications* , Volume 36– No.l, December 20ll.

[24] Arun Sharma, "Design and Analysis of Metrics for Component- Based Software Systems", PhD Thesis, 2009.

[25]Arun Sharma, Rajesh Kumar, and P.S.Grover, "Empirical Evaluation and Critical Review of Complexity Metrics for Software Components", published in the proceedings of the 6th WSEAS *International Conference on Software Engineering, Parallel and Distributed Systems*, Corfu Island, Greece, pp: 24-29, 2007.

[26] Cay S. Horstmann and Gary Cornell, "Core Java: Advanced features", Publisher: Prentice Hall, Volume 2, 8thEdition, 2008.

[27] Chidamber, S and Kemerer, C. "A Metrics Suite for Object-oriented Design, Published in the *IEEE Transactions on Software Engineering*", Volume 20, pp: 476-493,l994.

[28] DeTran-Cao, Ghislainlevesque, and Jean-Guy Meunier, "Software Functional Complexity Measurement with the Task Complexity Approach" pp: l, 2005

[29] Faisal Siddiqui, "Component Based Software Engineering: A look at reusable software components", available at http://www.smb.uklinux.net/reusability.

[30] George T. Heineman and William T. Councill "Component Based Software Engineering" Publisher: Addison Wesley longman, 200l.

[31] Ian Sommerville "Software engineering",7th edition, 2004, available http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/Presentations/PDF/Ch19.pdf

[32]Al – Badareen A., Selamat M.H., Jabar M.A., "Reusable Software Component lifecycle", *International Journal of Computers*, 5(2), pp. l9l – l99, 20ll.

[33] Gupta S., Kumar A., "Reusable Software Component Retrieval System", International Journal of Application or Innovation in Engineering and Management, 2[l], pp. l87 – l94, 20l3.

[34] Imeri F., Antovski l., "An Analytical View on the Software Reuse", ICT Innovations 20l2, Web Proceedings of the 4th ICT – ACT Conference, Ohrid - Macedonia, ISSN: l857 – 7288, pp. 2l3 – 222, 20l2.

[35] Kim W., "On Issues with Component – Based Software Reuse", Journal of Object Technology, 4[7], pp. 45 – 50, 2005.

[36] McIlroy M.D., "Mass Produced Software Components", Software Engineering: Report on a Conference by the NATO Science Committee, Brussels, pp. l38 – l55, l968.

[37] Pressman R.S., *Software Engineering – A Practitioner's Approach*, 5th ed. New York: McGraw-Hill Inc., ISBN: 0073655783, 200l.

[38]Sharma A., Kumar R., Grover P., "Managing Component – Based Systems with Reusable Components", *International Journal of Computer Science and Security*, l[2], pp. 60 – 65, 2007.