An Approach for Optimizing CPU and Memory Performance by Selection and Deactivation of Optional Components

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

Master of Engineering

in

Computer Science Engineering

Submitted By

Jasneet Chawla

(851232003)

Under the supervision of:

Ms. Ashima Singh

Assistant Professor

CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "An Approach for Optimizing CPU and Memory Performance by Selection and Deactivation of Optional Components", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science 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 ***Ms. 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.

(Jasneet Chawla)

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

(Ms. Ashima Singh)
Assistant Professor
CSED, Thapar University
Patiala

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. S. Bhatiaa)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

First of all, I am thankful to God for his blessings and showing me the right direction. With His mercy, it has been made possible for me to reach so far. I wish to express my deep gratitude to Ms. Ashima Singh, Assistant Professor, Computer Science & Engineering Department for providing her immense help, guidance, simulating suggestions and encouragement all the time. She always provided a motivating and enthusiastic atmosphere to work with; it was a great pleasure to do this thesis under her supervision.

I am also thankful to Dr. Deepak Garg , Head, Computer Science and Engineering Department for his kind help and cooperation. I express my gratitude to all the staff members of Computer Science and Engineering Department for providing seminars and encouraging towards research work.

I want to express my appreciation to every person who contributed with either inspirational or actual work to this thesis. Last but not the least I am highly grateful to all my family members for their inspiration and ever encouraging moral support, which enables me to pursue my studies.

Jasneet Chawla

Abstract

Component-Based Development is an approach of developing software systems by using components. Component-based software system may contain external components as well as in-house built components. Component based software engineering has become a modern approach for software development. It is a multifaceted approach in complex scenarios that focuses on "develop once reuse multiple times" methodology. As per user requirements, the components from repository are selected and integrated to develop software. It help developers to deliver high quality softwares within a less amount of time and cost with less effort. Component based softwares are based on modular approach that provides the benefit of easy scalability and flexibility of the software. But along with this advantage comes the disadvantages too. Adding more components may result in performance degradation in terms of responsiveness, throughput, bandwidth as well as incompatibility problems in terms of resource requirement. This research work propose to identify the non participating components of a component based software like excessive graphics , unnecessary animations and other subcomponents and deactivating them for that time to increase compatibility and optimize the system for best performance. By using the case study of Windows XP as reference, we successfully demonstrated that a component based software that has higher system requirements can run smoothly on a lower configured system by identifying and disabling the non participating sub-components. We successfully achieve up to 27.43% increase in performance by this method.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations	Description
CBSE	Component Based Software Engineering
CBD	Component Based Development
COTS	Commercial off the shelf
UML	Unified Modeling Language
CBSD	Component Based Software Development
QML	Quality Modelling Language
SPE	Software Performance Engineering
ERP	Enterprise Resource Planning
OPM	Optimal Performance Model
SDLC	Software Development Life Cycle
CBS	Component Based Softwares
QN	Queueing Networks
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
ICBD	Improved Component Based Development
RG	Redundant Group
CB-SPE	Component Based-Software Performance Engineering

COMQUAD	Components with Quantitative Properties and Adaptivity
EJB	Enterprise Java Beans
CSAD	Component Selection and Deactivation

Chapter 1

Introduction

1.1Component Based Software Engineering

Introduction

Component-Based Software Engineering (CBSE) means the development of software by using pre-developed software components. It may contain external as well as in-house built components. Component-based development approach develops the software systems by choosing appropriate components and then integrating them by using a well-defined architecture. Component based software engineering (CBSE) has become a modern approach of software development that provides an optimal, efficient, economic and quick software development as per user requirements.

The Need of Component Based Software Engineering

Modern software systems have become more large scale and complex which results in higher cost of building the systems, low productivity, and degraded system-quality. Thus need of developing a good, economical software development strategy arose. CBSE helps to deal with complexity by adopting a divide and conquer approach, modularization of large software systems into smaller and reusable units called components. CBSE helps in decreasing the development cost of the systems, the time it takes to deliver to the customer, enhance the maintenance of the software and dependability on the system.

Component-Based Software Engineering Process

The Component-Based Software Engineering Process involves

a) Recognition of candidate components

b) Qualification of the interface of each component

c) Adjustment of components to eradicate architectural conflicts

d) Assembling the components into a defined architecture

e) Updating components when there is need of changing the system.

Fig. 1 shows the process model for CBSE, focusing on two major aspects that includes domain engineering and component based development (CBD).

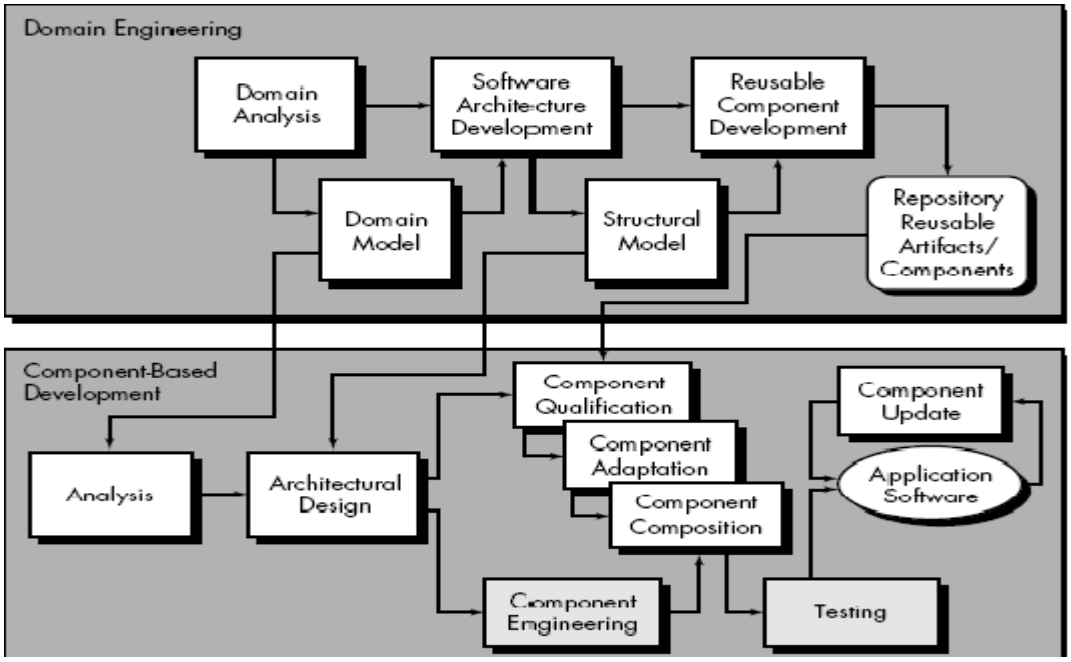

Fig.1: Component Based Software Engineering Process Model [1]

Domain Engineering

Domain engineering constructs a domain model of application which is used during CBD for analysis of user requirements, a structural model which is used as an input to architectural design, and provide reusable components to developers for component based development. Domain engineering focuses on recognizing, constructing and publicizing the components which can be used with the current and future softwares in a specific domain. It involves analysis of the application domain during which the domain analyst finds out the duplicate patterns in application inside a domain to design an application domain model, development of architecture for constructing structural model containing few number of structural elements showing unambiguous patterns of

interaction and development of components which can be reused and stored in a repository which can be used by developers during component based development.

Component-Based Development

Component-Based Development (CBD) is an approach of developing software systems by using components. The programmer uses already existing components to fulfill the desired function which is required in the application.

Fig. 2 describes that components are picked up and then integrated into the destined system.

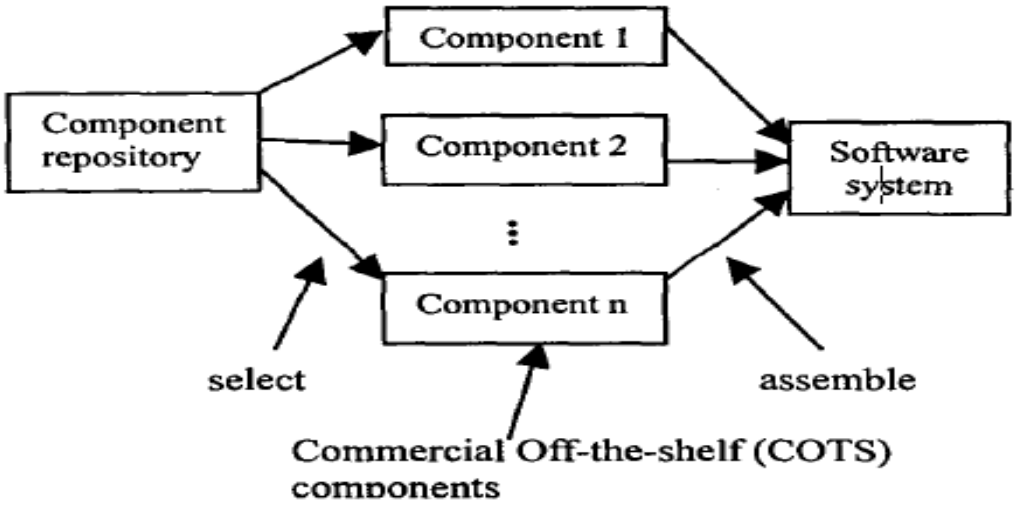

Fig.2: Component Based Development [2]

Component based development includes analysis of customer requirements and enhancement of architecture which is suitable according to analysis model being designed for the application. Then architecture is supplied with components which are either already existing or newly developed. If the components are available to be reused then the component qualification (i.e. ensuring that the selected component executes all the desired functions, appropriately adjusts into the specific architecture and reveals the quality attributes such as availability, security, portability which are essential for the application), component adaptation (i.e. ensuring that constant ways for managing resources are used for every component, for every component a same set of activities are there and interfaces are realized in a constant way) takes place, however if not, then components are engineered and then finally all the components are integrated and tested.

Principles that govern the Component Based Software Engineering Design Process

1. Components are independent.
2. Implementation is hidden.
3. Communication is through well defined interfaces.
4. Component platforms are shared and reduce development cost.

1.2 Properties of a Software Component

Component

A component is a modular building block of a for a computer software. More formally OMG Unified Modelling Language Specification[3] define a component as " a modular, deployable and replaceable part of a system that encapsulates implementation and exposes a set of interfaces".

Properties of a Software Component in CBSE [4]

i. **Identity**: Every component should be uniquely recognizable in the environment in which it is developed and in the environment in which it is to be deployed.

ii. **Modularity and encapsulation**: Software systems are divided into modules called software components thus they possess modularity. Each component encapsulates related data elements and implements a logic to provide a functionality.

iii. **Independent delivery**: Components should be provided as independent parts to component users so that they can use them to develop a component-based system.

iv. **Reusability:** Every software component must have the property to be reused. The assets that are reusable include analysis of specification, design, source code and executables.

v. **Customizability and packaging**: This feature mean that software components should be customized and packaged to meet the functionality desired by the user. Such components are called customizable components.

vi. **Deployable:** A software component is said to be deployable if it is developed using a clearly stated strategy. It results in making an executable instance in the environment in which it is to be deployed.

vii. **Interoperability**: A component is said to have this property if it permit interactions and transfer of data with other components. It can be classified as local and remote interoperability. Local interoperability means a host centered environment for components and remote interoperability means components are on network.

viii. **Composition**: It allows components to have composition relationship which helps a component to produce and demolish other components. It is transitive in nature , which means if C is a part of A, and A is a part of E then C is also a part of E.

ix. **Model conformity:** Components should be built on a clearly stated model of architecture, interface style etc.

1.3 Component Based Software Development Life Cycle

The CBSD Life Cycle includes all the activities and work products essential to engineer a component based software system. Various Phases of Component Based Software Development Life Cycle are listed below:-

i) **Component Selection:** Appropriate components hat can be reused are selected.

ii) **Component Qualification:** The component that properly fits into the architecture of the system qualifies. Alternatively, generate a proprietary component that can be used in the system.

iii) **Component Adaptation:** Modifications are made in the components so that components can be properly integrated.

iv) **Component Testing:** Test each component after adaption**.**

v) **Components Assembling:** Integrates or merge all the components to form subsystems and to develop a complete application.

vi) **System Evolution:** Former versions are replaced with new versions of components.

1.4 Disadvantages of Component Based Software Development

a) Identifying appropriate components which fits the architectural design of the software to be developed may become difficult sometimes because a gap may exist among the requirement of the user and component's features.

b) CBD has not broadly accepted in the embedded system domain because of the incapability of the approach to adjust with various important concern areas of embedded systems.

c) Component Based Development possess a feature of Flexibility. This allows easy addition of new components in existing software system. This leads to some compatibility and performance issues.

1.5 Challenges for Component Based Development [5], [6]

a) Parameter Incompatibility

During the component based development exchange of data occur between components that are integrated. But sometimes this may cause some problem. Because the components may be provided by different vendors and most of the times the black box components are provided i.e. the source code is not available. So it becomes difficult for the component user to predict the functionality of the black box components. Also it is difficult to modify black box component. Thus it is difficult to use the black box component during the component based development, when value returned by one component's function is passed to another component's function as an argument to perform its operation but their data types are dissimilar so parameter mismatch occur. This is called parameter incompatibility problem. In this case an error may arise and it may affect other component's functionality. This may also results in the incorrect output and in some cases system performance may degrade. This problem can be reduced by selecting the independent components or the components having low coupling with the other components throughout the component based development.

b) Interface Complexity

Minimizing and Controlling complexity of the software is one of key concern of the software development paradigm as it affects various other aspects like software

reusability , testability , compatibility , maintainability etc. The Interface complexity is a key aspect to be considered while selecting a component during component based software development. The interface complexity can be defined by the considering its interactions with other components. So the component chosen should have less number of incoming and outgoing interactions with other components, in other words the component should have low coupling with the other components. Less interface complexity assist in minimizing integration and the maintenance efforts. Thus the components having less complex interfaces can be easily integrated with other components.

c) Performance Challenges

CBSE possess two major performance engineering challenges.

i. The component development is done by third party companies. For performance analysis, the distribution must be reflected in design of prediction models.

ii. Performance specifications must be parameterized according to the usage and deployment.

d) Incompatibility Problems

Faults may appear in components while integration even if they are thoroughly tested in isolation. This issue is termed as component incompatibility problem. Selection of an appropriate component that fits in a system is very essential. Incompatibility can further lead to performance degradation of the software.

1.6 Functional and Non Functional Requirements of Component Based Software

Functional Requirements describe high level statements of what the system should do. It describes the behavior of the system as it is related to the functionality of the system. It describes system services in detail.

The components fulfilling the functional requirements are called the Functional Components. Each component provides the specific functionality. In CBD , such components are integrated together to deliver user desired functionality.

Let's say, a component is created by a developer which allows a user to "log in" then other programmer can use it in other applications for that require same functionality.

Non Functional requirements put constraints on how the system should provide services.

Specifications of functional properties is easy to understand whereas analysis of non functional properties are under research.

Zschaler et. al Proposed a new specification language QML/CS to effectively model the non functional properties of components and component based softwares.[8]

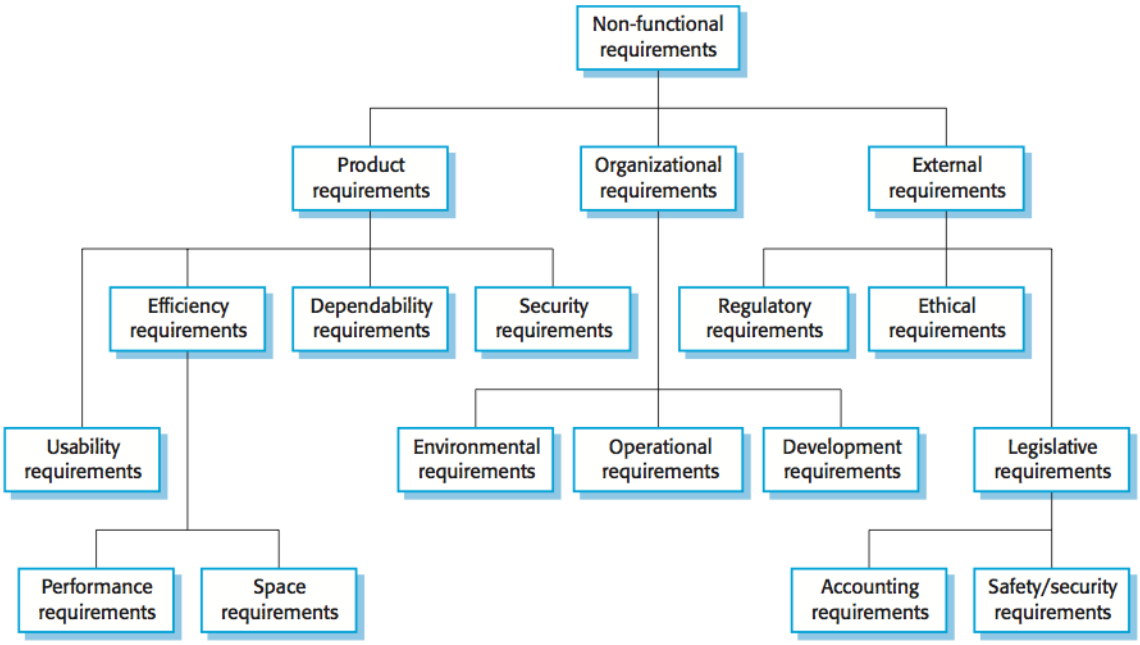

Fig.3 Non-Functional Requirement types[7]

1.7 Performance Testing[9]

In Software Engineering,This testing is usually performed to determine the system's performance in terms of stability and responsiveness under a particular environment . It can also be used to validate the various quality factors of the system like resource utilization , reliability etc.

Types of performance testing:

a) Load Testing

It is the simplest Performance testing conducted to understand the behaviour of the system under specific expected load.

b) Stress Testing

It is usually performed to understand the upper limit of capacity of the system. It helps to determine the behaviour of the system if the load goes above expected.

c) Soak Testing

It is also known as Endurance testing, done to determine if the system can bear continuous expected load. Significant load is applied for a prolonged period of time and the behaviour of the system is tested.

d) Spike Testing

It is performed to observe the behaviour of the system in case the load increases instantly.

e) Configuration testing

Testing of software is performed on systems having different hardware and software configuration and their behaviour is studied.

1.8 Software Compatibility

It is a characteristic of software components that can operate together on a same computer or on different computers in a network. It is possible that some computers are compatible in one computing environment and incompatible with others.[6]

Software Compatibility Testing

It is conducted to evaluate the compatibility of application with the computing environment that includes the computing capability of hardware, compatibility of peripherals, operating system and other system softwares.[6]

1.9 Organization of Thesis

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

ii. Chapter 2 gives the overview and analysis of various approaches being proposed for Optimizing Performance of component based software.

iii. Chapter 3 states the Problem formulation and research objectives.

iv. Chapter 4 describes the proposed approach for Performance Enhancement of Component Based Software.

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

vi. Chapter 6 concludes the work done and states future scope of work.

Chapter 2

Literature Review

This section describes an overview of various approaches being proposed for optimizing non-functional requirements of component based softwares. The aim is to investigate the techniques used, to know the contribution and to analyze the benefits and shortcomings of various approaches.

Attribute	**Component based Software Engineering**	**Traditional Software Engineering**
Cost	Allows the reuse of components, thus reduces the cost.	There is no reuse concept, so no cost reduction.
Development time	The reuse of prior developed components lead to reduction in development time.	Pre developed components are not used and so no significant reduction in development time takes place.
Quality	The Software components that are reused exhibit characteristics like performance , reliability and usability, thus improving quality.	No reused component is used , so improvement in quality.
Applicability	This approach of software development is only applicable to	There is no such restriction with traditional software

	softwares with pre built components.	engineering approach.

Table 1:Comparison of Component based Software Engineering Process with Traditional Software Engineering process[10]

Software component testing means testing the component in isolation or a group of interrelated components [11]. Software component testing also refer to group of activities to find out errors and to ensure the components fulfill quality requirements[4].

Software component testing has certain characteristics which are similar to that of software testing. Several are described as follows [12].

Software Component Testing Characteristics

i. **Dynamic:** Software testing can either be dynamic or static. In dynamic testing, the actual execution of system takes place with certain inputs to check whether expected the output is achieved or not , this is done by executing the test cases. While the static testing is performed during early stages of software development life cycle i.e. usually before the coding phase and this testing is done without actually running the system . It is preferred that system should be dynamically tested to make sure that component fulfills the user's demands and hence gives more clear results as compared to static testing.

ii. **Finite:** The number of test cases can be sometimes infinite because of the huge number of combinations of the conditions to be tested, which makes testing infeasible in terms of time and cost. Therefore the total test space should be finite i.e. the number of test cases to be executed must be finite so that it takes limited time and resources for testing.

iii. **Selection:**From a huge existing test suite , the test cases should be selected properly and efficiently in accordance with specific testing criteria for test selection and coverage to achieve cost-efficient and adequate fault revealing testing.

iv. **Expectation:** As a result of test execution a decision should be generated whether the test has successfully passed or failed in order to assess expected quality of the software. For this purpose test oracle is used which generate desired testing

results for certain input and compares the actual testing results with the desired expected results.

Software testability means ability to be tested or in other words we can say testability is the extent to which a software allows setting up of a test criteria and efficiency of test to check that the established criteria is met or not; or it also refers to the extent to which the requirements are described in a way that they allow setting up of a criterion for testing and efficiency of test to check if the established criteria is successfully met or not [11]. Software testability is one of the major factors to compute software reliability [13]. In order to make component reliability measurable or quantifiable, the following stated characteristics can be used for measuring software component testability efficiently and effectively. Fig. 4 shows characteristics that can be used to measure the testability of a software [14].

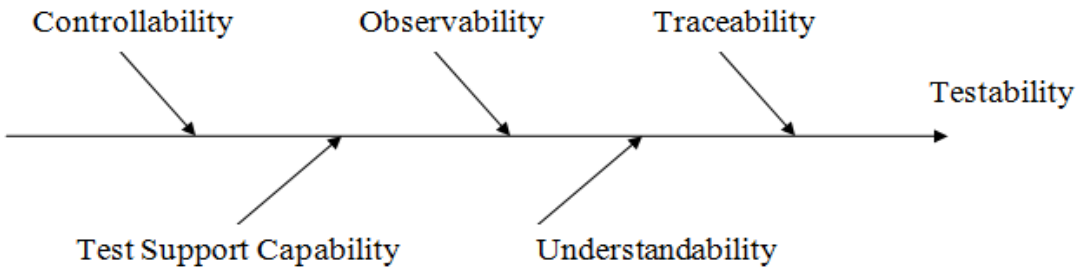

Fig.4: Software Component Testability Characteristics [14]

Freedman [15] considered two factors to describe testability of domain in his research that are observability and controllability. Binder [16] also uses these as features of testability and traceability for representation of testability and testing environment. Gao et al. [4] [17] expresses component testability by using following properties. Fig 5 shows software component traces.

i. **Component Traceability:** Component traceability is the degree to which a component is capable of tracking its functions, features and the way it behaves internally or externally.

A component that is traceable allows all the essential details for testing i.e. details which are required to describe the way it executes and to be recorded. The major traces which can add value to testing of component are:

a. Operation Traces – It keeps a track of all the communications within a component as well as between different components.

b. Performance Traces – It keeps a track of the data related with efficiency and the standards for every method in a component for a specific work environment. These traces are also used to recognize the problems related to performance testing.

c. State Traces – It keeps a track of all the states in a component.

d. Event Traces – It keeps a track of an event and sequence which has been followed in a component.

e. Error Traces – It keeps a track of the information regarding errors, and details of exceptions being produced by component.

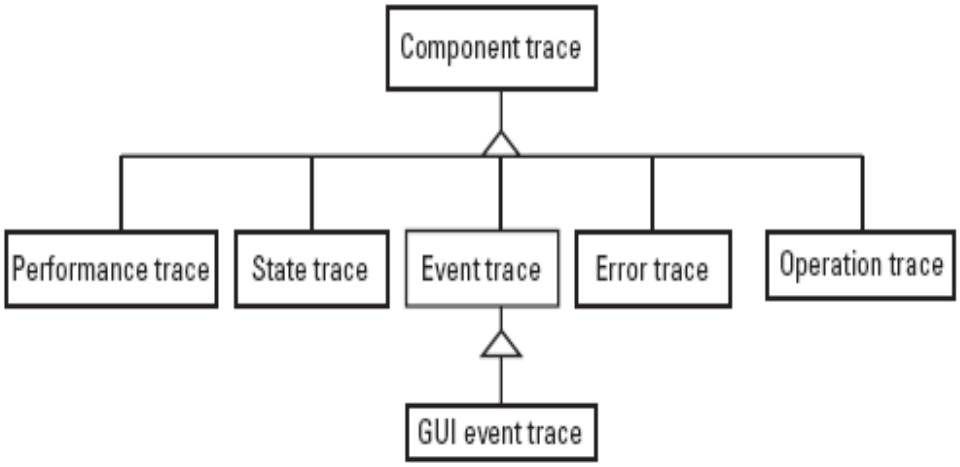

Fig.5: Software Component Traces [4]

ii. **Component Observability:** Component Observability means the extent to which it is simple to monitor the testing details on the basis of behavior of a component, the given input, resultant output when executed for a test case. A clearly designed interface can improve the capability of a component to be observed, which allows

user to identify a relation between input given and subsequent output produced for a test case during testing.

iii. **Component Controllability:** Component Controllability means the extent to which it is simple to control input given, resultant output, functions and behavior of component implementation while it is being tested. It is used for measuring the ease of executing tests and generating a particular output by giving a particular input, in order to control the predictability of some desired output being generated from corresponding input supplied to the component.

iv. **Component Understandability:** Component Understandability means the extent to which it is simple to understand the details of a component, allowing the test team to simply use the necessary details such as requirement and specification to perform testing and create efficient tests for software component testing. The key factors associated with this property are

- Availability i.e. whether all the necessary documents like SRS, readme document, design, code are available or not.
- Understandability i.e. whether the details are represented in a presentable way such that it is easy to read and understand it or not.

v. **Component Test Support Capability:** Component Test Support Capability means ability of a component to support its automatic testing and is concerned with ability of a component to generate tests, organize tests, evaluate and analyze test coverage and ability to execute tests and support testing.

Smith[18] defines software performance engineering(SPE) as a systematic and quantitative approach to develop software systems that fulfills the performance objectives. SPE is a software-oriented approach which focusses on the architecture , design as well as implementation choices considered while software development. It uses model predictions to figure out the trade-offs in the size of the hardware, the functionality of the software, the requirement of the resources and the expected quality. It prescribes the principles and the performance patterns for creation of response-oriented softwares, performance anti-patterns for recognising and correcting common problems, the data required for evaluation, the procedures for obtaining performance specifications and the guidelines for types of evaluation to be carried at every software development stage.

The following fig 6 shows the Performance engineering process.

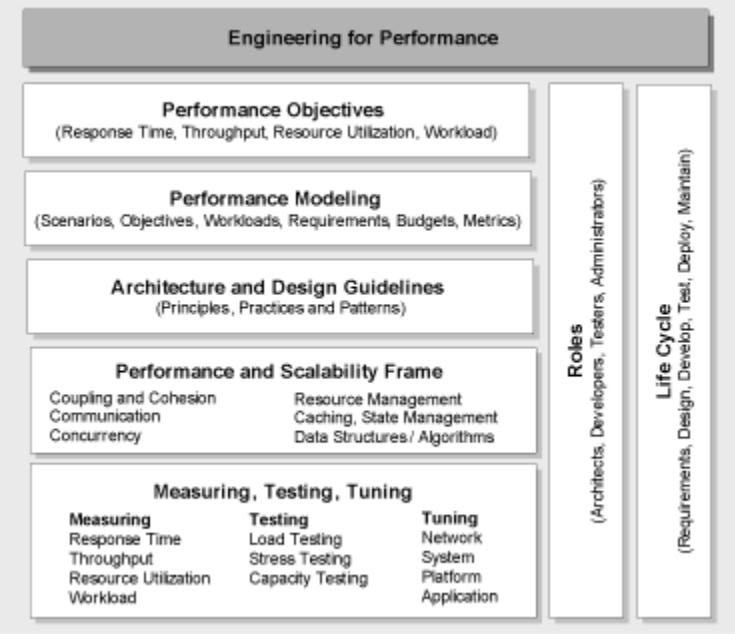

Fig.6: Engineering for Performance [19]

Performance Engineering Objectives[20]

a) Eradicte avoidable system rework because of performance related issues.

b) Eradicate the system deployment because of performance related issues.

c) Prevent unnecessary hardware acquisition cost.

d) Deacrease increased software maintenance cost due to performance problems.

e) Decrease additional operational overhead for handling system issues due to performance problems.

Osama et. al[21] identified various problems related to CBD like inadequate inclusive tools, less efficient methods to manage and collect the information required for selection of COTS for a specific application. He proposed an Optimal Performance Model (OPM) that made the selection of COTS for ERP systems in a more effective and efficient manner. OPM is based on several Standards of Quality. This information helps in attaining more useful and quality based ERP solutions(whether for implementing a new ERP or upgrading the existing one) that meet the business needs in a better way. The fig.7 demonstrates the optimal performance model.

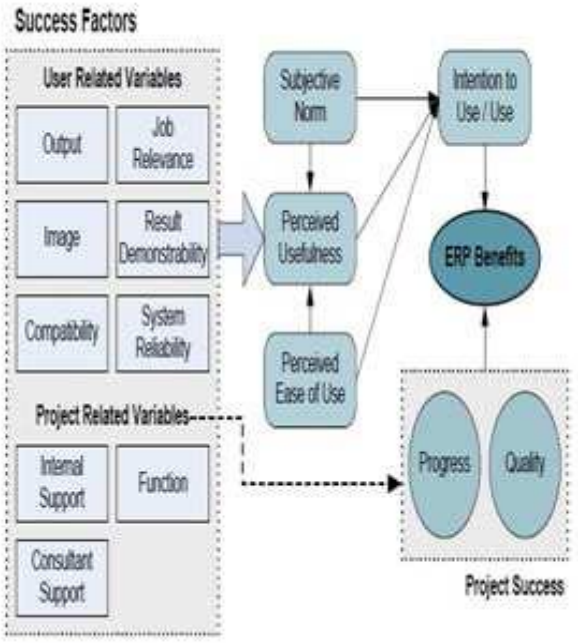

Fig.7: Optimal Performance Model[21]

Balsamo, Marzolla and Mirandola[22] presented a paper in which the issue of performance evaluation at early stages of SDLC was addressed. An approach that was based on Queueing network analysis evaluation of CBS was proposed. By using software specifications that are annotated in terms of UML use case, acivity and deployment diagramsare used to analyse the performance bound. This is based on multi-class and multi-chain QN model. The approach performed succesful performance evaluation at architectural level.

Anupama Kaur[23] addressed the need to recognise reusable components from a software and their reusability was determined using neural networks. The approach works in two steps. In First step, the code is parsed to calculate metric values: Cyclometric Complexity Using Mc Cabe's Measure, Halstead Software Science Indicator, Regularity Metric, Reuse-Frequency Metric and Coupling Metric. The generated metric values are supplied as input dataset for different neural networks to evaluate reusability. In second step, The neural network is designed and is used for evaluation. Firstly,the neural networks are trained using the training dataset. After training, the neural network is evaluated against the testing data and comparison is made on the basis of MAE(Mean

Absolute Error), RMSE(Root Mean Squared Error) and Accuracy values of neural networks.

Asif Irshad Khan[24] Proposed an Improved Component Based Development Model that uses Expert Opinion Technique to overcome some of the problems associated with Traditional Component Based Development Models. Fig 8 shows an overview of various phases of ICBD Model and Fig 9 show a detailed view of ICBD Model.

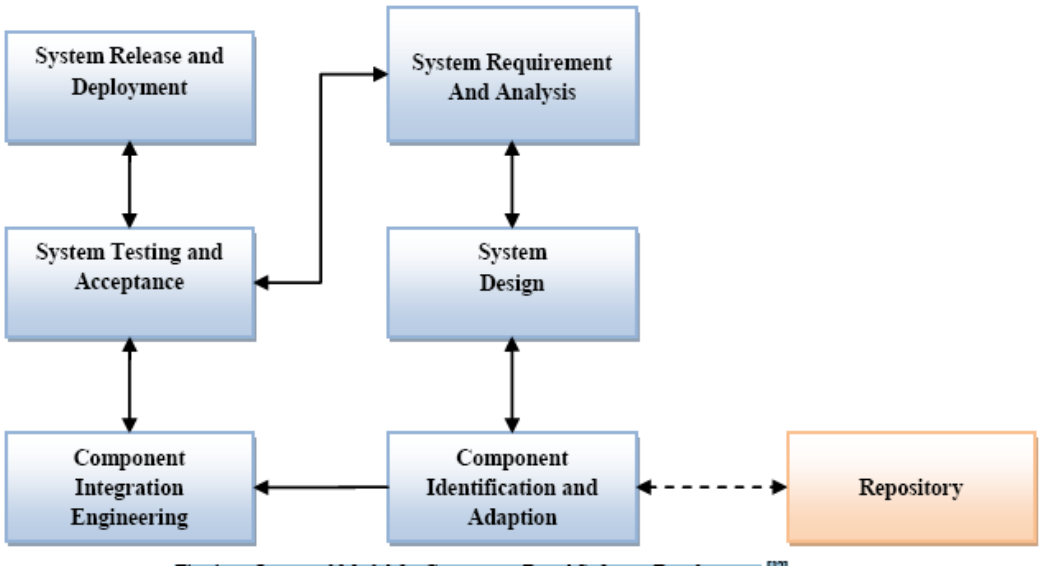

Fig.8: An Overview of various Phases of ICBD Model[24]

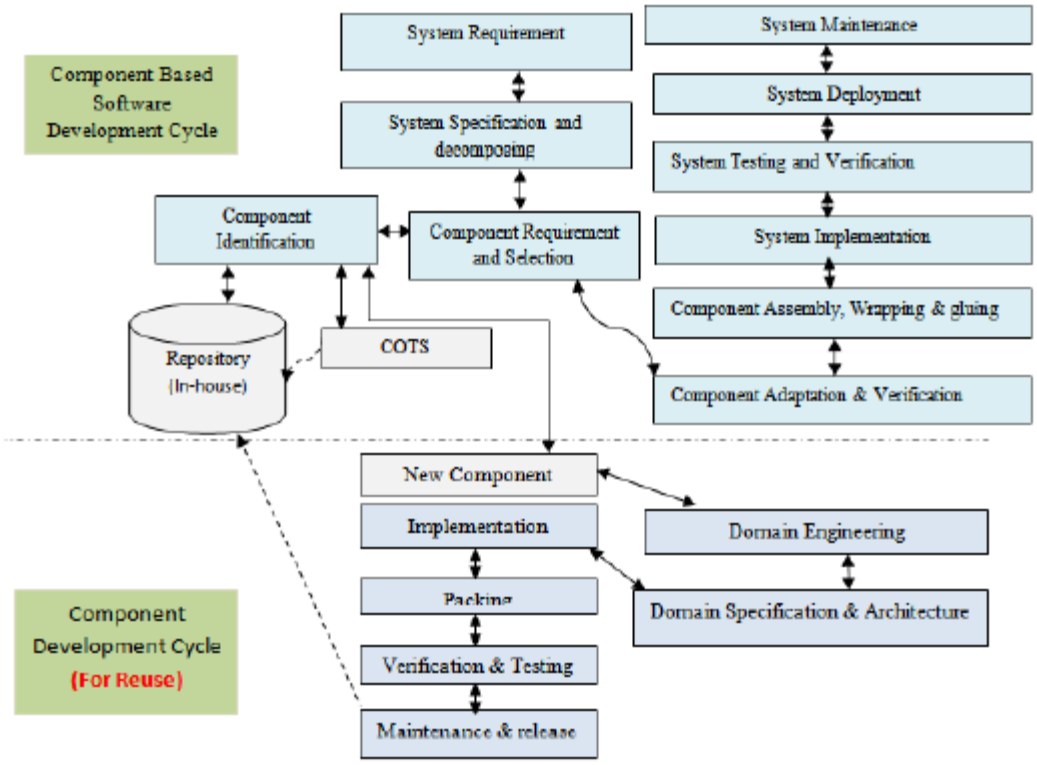

Fig.9 A Detailed View of ICBD Model[24]

Experts who were software engineers and those who have been working with component based software in many renowned organisations were sent the questionnaires and their responses were analysed.Likert Scale was used by the experts. It was analysed from the survey that the rating for ICBD Model was between nominal to high.

Kahkipuro[25] presented a performance modelling framework to produce predictive performance models that can help in generating information related to performance at all stages of SDLC for development and maintenance of component based distributed systems. The framework describes a UML based notation for describing performance model and set of special techniques for the modelling of component based distributed systems.

Diaconescu and Murphy[26] devised an approach AQuA for automating management of component based enterprise systems in which multiple component variants serving same functionality are categorised as a redundant group. At run time based on the execution environment, the selection of component from RG is done to optimize the system for best performance.

Bertolino and Mirandola[27] devised an easy to use technique for prediction and analysis of performance of a component based system. For this purpose, a CB-SPE framework composing a methodology for software performance engineering and a supporting tool was proposed. The approach is divided into component layer and application layer. At component layer, developers model the schedulable resources demand of individual performance service in dependence to environment parameters. Parameteric performance evaluation of components is done in isolation. At Application layer, software architecture pre-selects the performance models and then compose them into architectural models. They model the flow of control using sequence diagrams. CB-SPE technique also includes the free available modelling tools, transformation tools and performance solver tools.

Zschaler[28] devised the COMQUAD "Components with Quantitative properties and Adativity" Model enabled the run time support for non functional properties that are described orthogonal to the application structure using CQML+ descriptors. This has been achieved by extending the concepts of existing component based systems Enterprise Java beans (EJB) and COBRA components(CCN).A component container acting as contract manager is developed that evaluates the performance requirements at run time. The contract manager evaluates the performance requirements of the client against the performance specifications to select the components.

Chapter 3

Problem Statement

3.1 Research Gap Analysis

While reviewing the wide literature it was observed that researchers have proposed many approaches that predicts the performance at architectural level , at design level and other optimizing techniques but the reviewed approaches didnot optimize the perrformance after the development of the softwares.The proposed approach do appropriate selection of the non required components and then deactivates the identified components for performance enhancement.

3.2 Problem Formulation

Component-Based Development is an approach of developing software systems by using components. The components are stored in a repository, which are usable by other programmers. Component-based software system may contain external components as well as in-house built components.Modular approach of Component based software engineering has advantages of adding and removing and updating system components as per user requirements. It makes system more efficient towards problem solving. But adding more and more components have several drawbacks too. It degrades the system performance measured in terms of system response time, throughput and also degrades the system compatibility measured in terms of system resources. This overall degrades gradually the components based softwares performance for system with constant configuration.

As software versions are upgrading very rapidly with changing business requirements whereas the hardware components of a system almost remains constant that becomes incompatible in due course of time which is a major problem.

We propose to indentify and detect the non participating components viz.

a) Excessive Graphics
b) Unnecessary animations

c) Other non required functional subcomponents

And deactivate them for that particular time. This will optimize the system for best performance. Thus the software will be able to run even on lower configured systems efficiently.

3.3 Objectives

i. To study and examine the existing component based software development approaches for their performance and compatibilities.

ii. To propose an approach to enhance the performance and compatability of component based system.

iii. To validate the proposed approach using a case study.

Chapter 4

Proposed Approach: Component Selection and Deactivation (CSAD)

4.1 Overview

Softwares that are engineered using component based development comprise of components that provide different functionalities. While using a software for a specific functionality or purpose, we require only the related component whereas other components are not needed at same time.

Example Consider a library management information system, it comprise of various components providing different functionalities like login component, books detail component, book issue component, book return component etc. Suppose a User logs in and want to use book issue component so at this time the other non required functional components can be deactivated for that specific period of time to enhance the performance of the system.We propose an approach "Component Selection and Deativaion "(CSAD) that identifies the non participating functional components and non required functionalities and deactivate them for enhancement of performance and for better compatibility.

4.2 Detailed Description of Proposed Approach

Component Selection and Deactivation (CSAD) : CSAD is an approach that primarily implemented in two phases. In the first phase, the Identification of all the non participating functional and non required components is done. The non participating functional and non required components will be named as optional components in further discussion. In second phase, the selection is made and the optional components are deactivated.

Phase-1 Analysis and Identification of Components

All the component are analysed on the basis of user requirement for that particular time and the components are categorised as essential and optional components.

All the identified components are ranked on priority basis.

Step 1: On the basis of user requirement , the components are categorised into essential and optional components.

Step 2: After the analyses of components, the components are ranked on the basis of their priority for user.

Phase-2 Selection and Deactivation of Identified Optional Components

On the basis of priority , the components that are low on priority are deactivated in order to achieve increased performance and better compatibility.

Step 1: From Priority list, the components having low priority are selected.

Step 2: Deactivate the identified optional components.

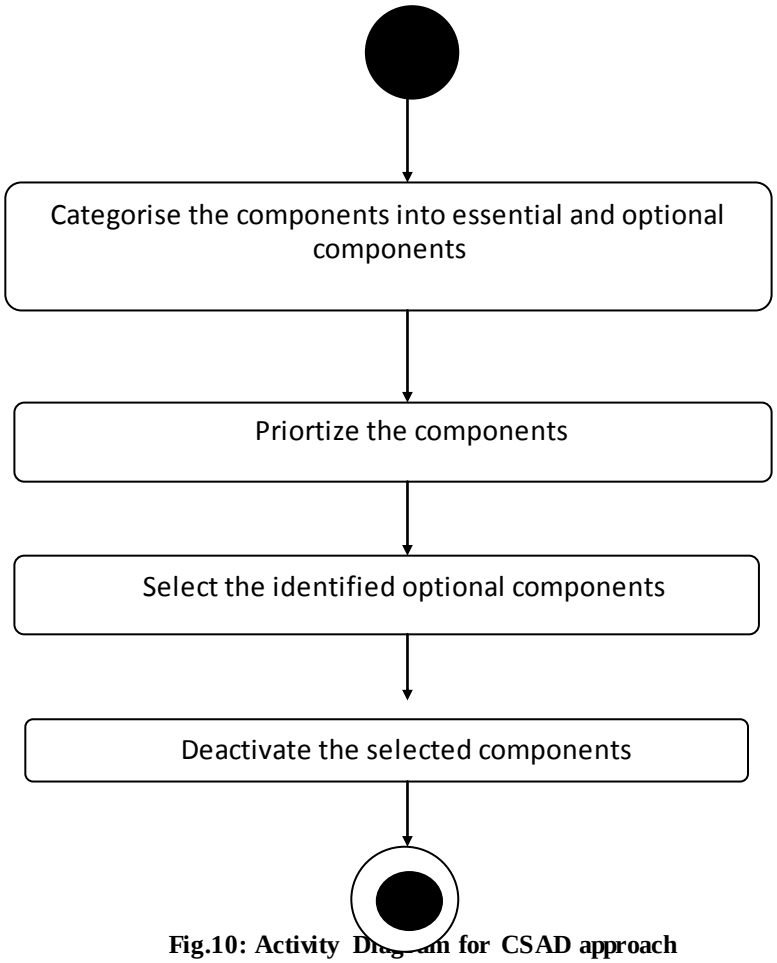

Fig.10: Activity Diagram for CSAD approach

4.3 Calculation for Performance Enhancement

The CSAD approach helps to improve the performance of the system, To calculate the increase use following steps:-

Step 1:Calculate difference in memory consumptions by the components.

M(CSAD)= M(f)-M(r)

Here M : Memory , CSAD : Component selection and Deactivation ,

M(f) ; Memory consumptions in full component mode ,

M(r): Memory consumption in reduced component mode

Step 2:Calculate the CPU utilization :

C(CSAD)= C(f)-C(r)

Here C(CSAD) is the CPU performance indicator using the proposed approach ,

C(f)= CPU usage with all components ,

C(r) = CPU usage with reduced component .

Step 3:Calculate CSAD :

The final CSAD = $\sum$(M(CSAD)+C(CSAD)) / 2

Chapter 5

Validation of CSAD using Case Study

5.1 Case Study: Windows XP

The case study of Windows XP has been used to validate the proposed approach (CSAD).

Step 1: Place the cursor on start button and choose control panel.

Step 2: Then choose system.

Step 3: Then choose advanced system settings

Step 4: From system properties dialog box choose Avanced tab.

Step 5: Under Advanced tab, choose performance settings option.

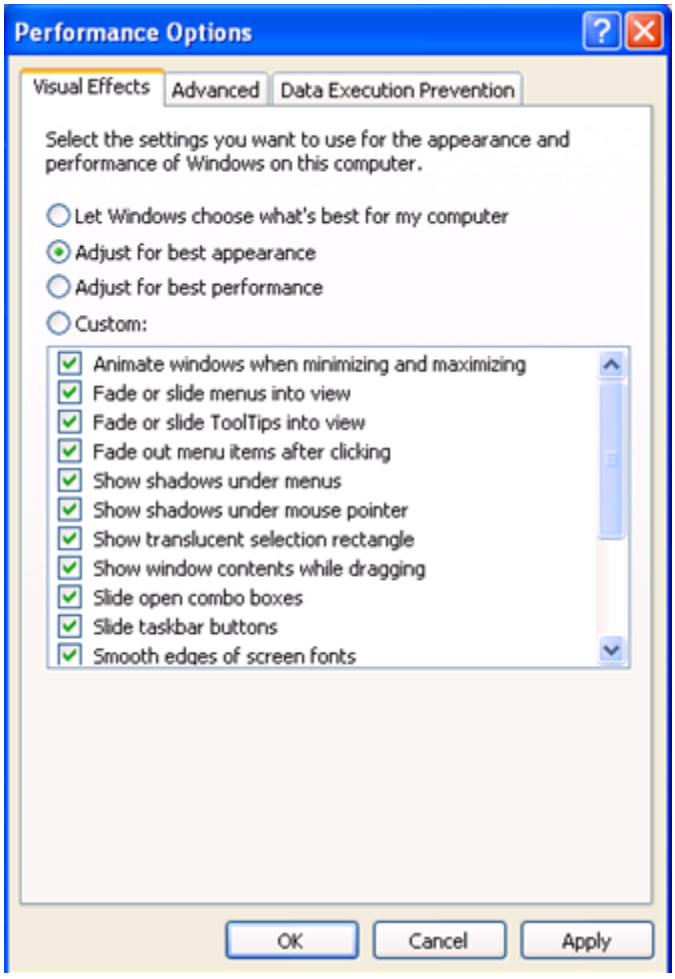

Fig.11 : In case of adjusting the system for best appearance (with non deactivation of components)

In the above figure we can see that three cases. In first case, the computer system automatically optimizes for best performance. In this as per the resource availability the system for that particular time switches to the mode in which it can utilize the available resources in the bes possible way. In second case, system is adjust for best appearance. In this case animator and graphic components are acivated.This uses all resources to attain best appearance for the system. In third case, the system is adjusted for best performance and in this case all the unneccesary components like excessive graphics and animations are disabled to optimize system for best performance.

In case of adjusting the system for best appearance, if we want to monitor the resource utilization like CPU and memory usage, following steps are performed.

Step 1: Open Windows task manager.

Step 2: Under performance tab, monitor CPU usage history and physical memory usage history.

The following graph appears when system is adjusted for best appearance.

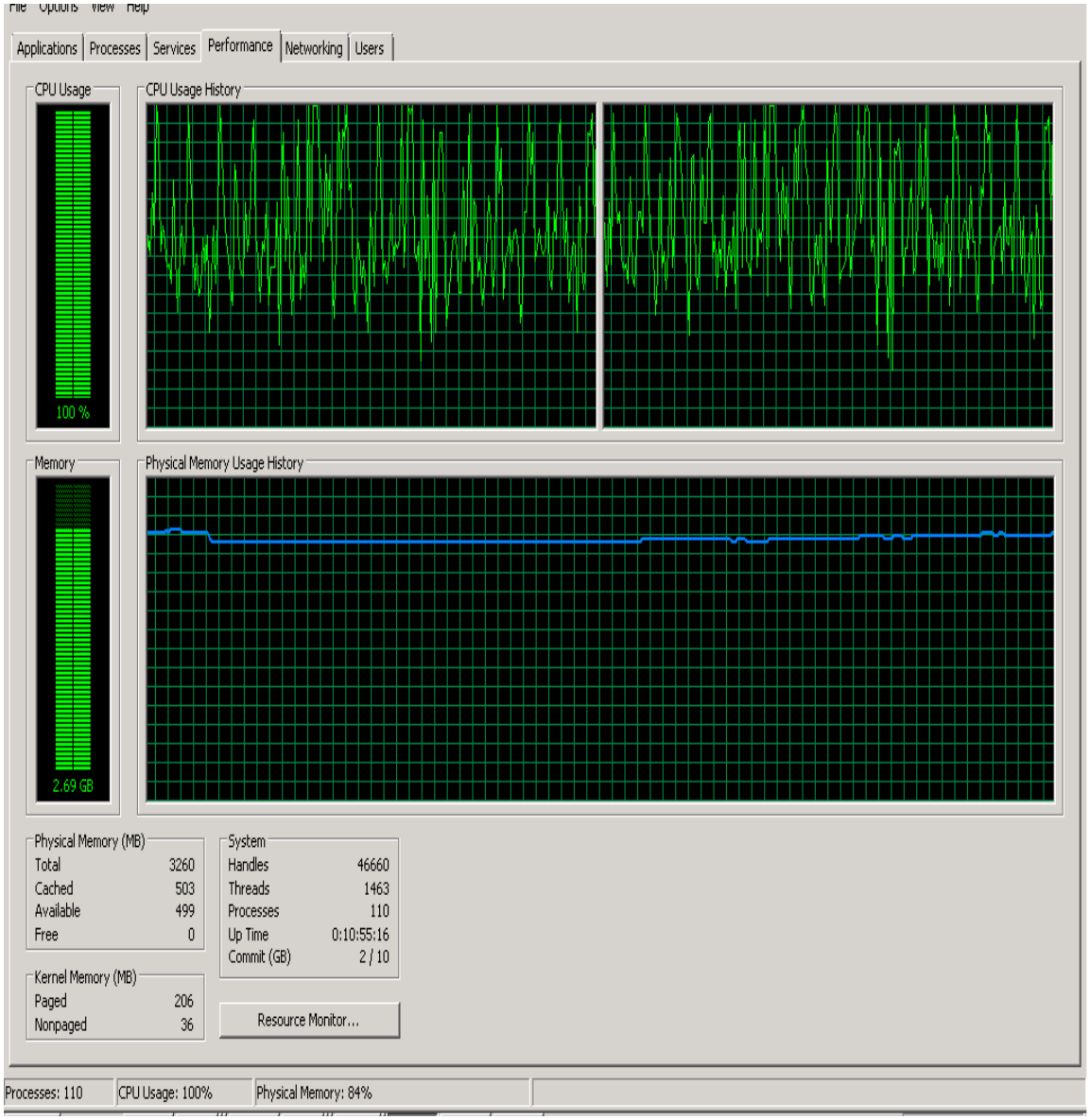

Fig.12 : Performance (In case of adjusting the system for best appearance-with non deactivation of components)

From the figure above , we can see that the CPU usage is 100% and memory usage is 2.69 GB in case of adjusting system for best appearance.

Now consider the third case in which the system is adjusted for best performance. If we want to optimize system for best performance all the non required components like excessive graphics and other animator components are deactivated.

Consider the following figure that lists the deactivated components.

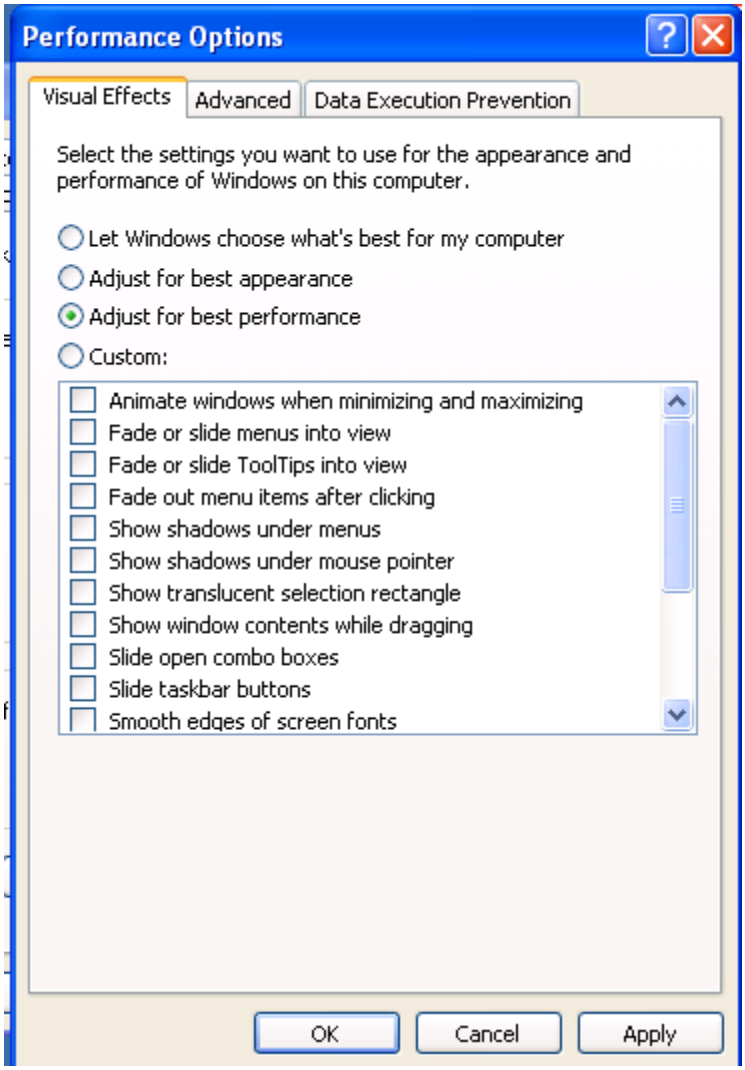

Fig.13 : In case of adjusting the system for best performance(with deactivation of components)

To monitor the CPU and memory usage again choose performance from Windows task manager's dialog box.

To evaluate the performance difference analyse the following figure.

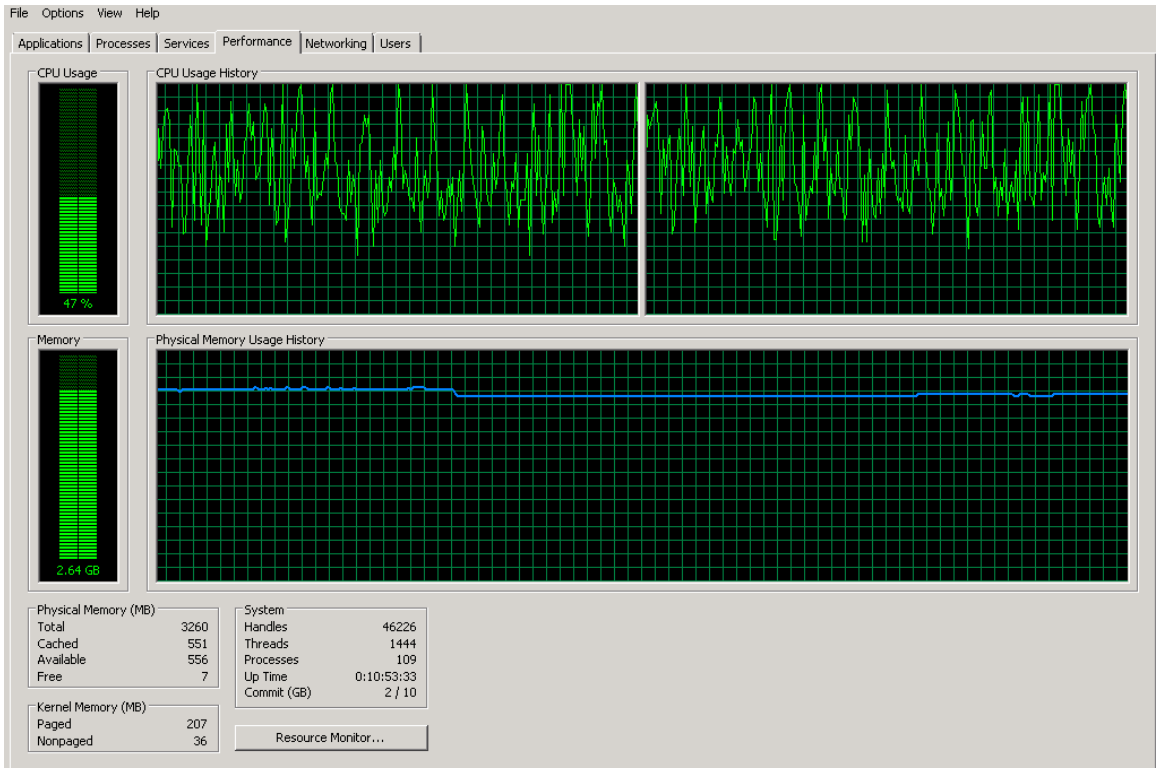

Fig.14 : Performance (In case of adjusting the system for best performance-with deactivation of components)

The figure above shows that after deactivation of less required components the CPU usage is 47% and memory utilization is 2.64 GB.

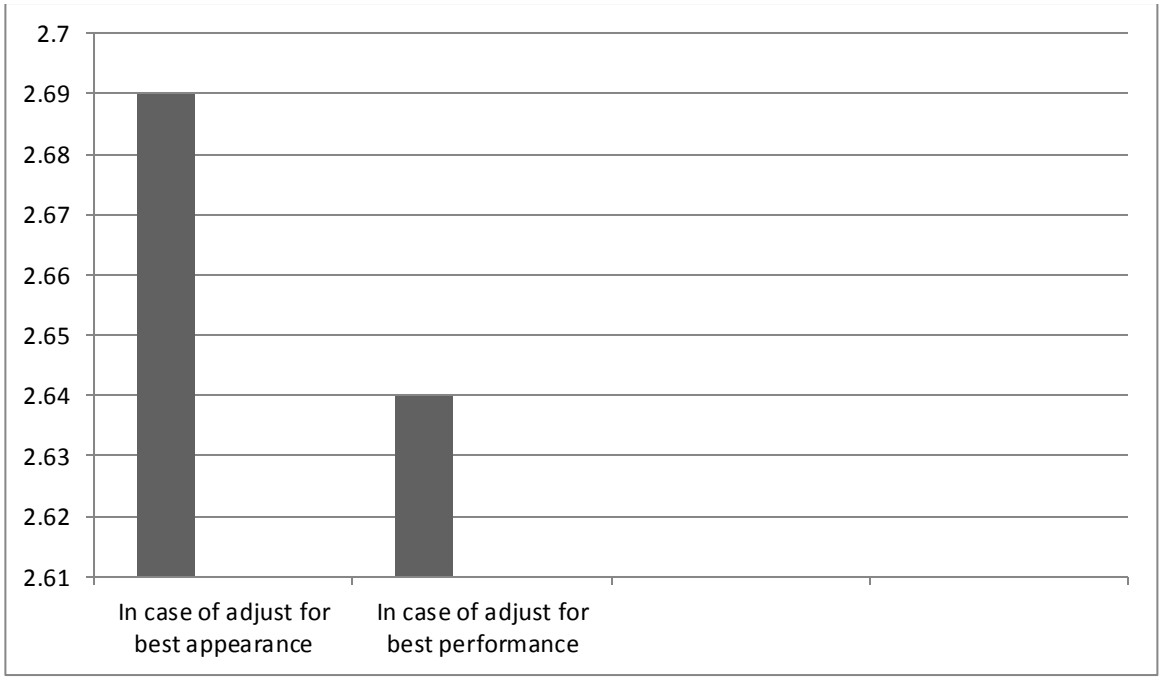

Fig. 15 Graph showing the memory consumption of a system (with and without component deactivation)

The Fig.15 shows the comparitive analysis of memory consuption of a system with and without component deactivation using Graph.

Table 2 Memory consumption

In case of adjust for best appearance (without deactivating components)	2.69 GB
In case of adjust for best performance(with component deactivation)	2.64 GB
Difference	0.05 GB
Percentage Increase (A)	= (0.05/2.69)*100 = 1.85 %

The table depicts that the difference in both the cases is 0.05 GB. This leads to performance increase of 1.85%.

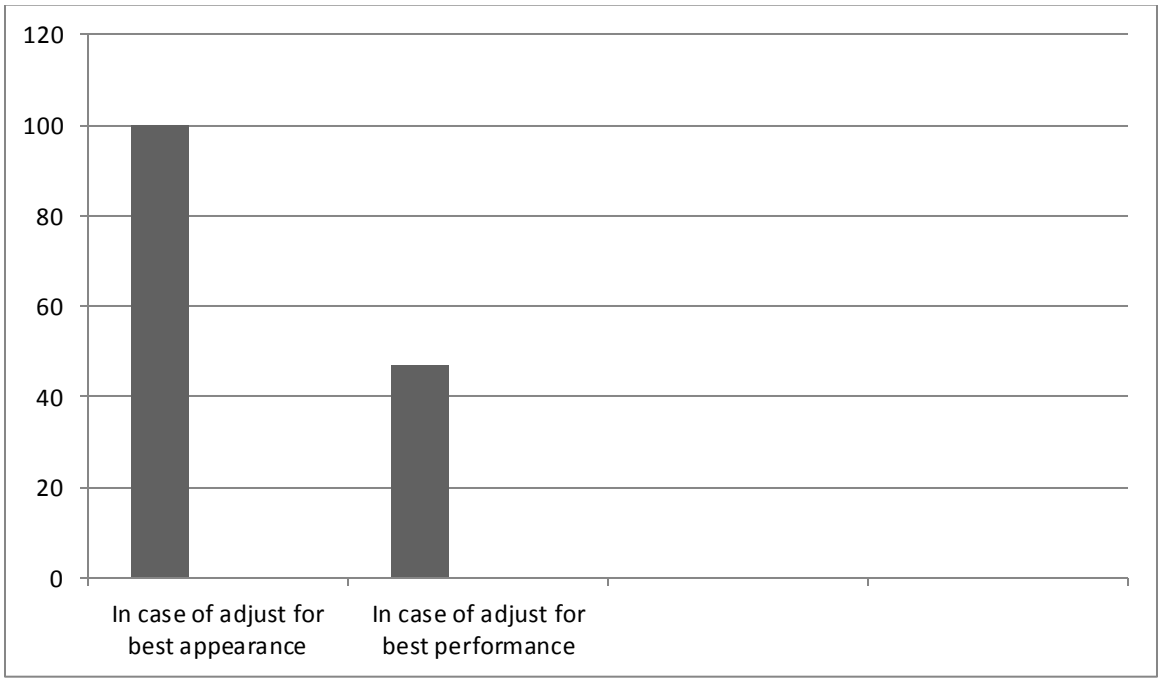

Fig.16: Graph showing the CPU utilization of a system (with and without component deactivation)

The graph shows the CPU usage of a system with and without component deactivation.

Table 3 CPU utilization

In case of adjust for best appearance (without deactivating components)	100%
In case of adjust for best performance(with component deactivation)	47%
Difference (B)	100-47=53% increase

The table above depicts that the increase in percentage attained is 53%.

5.2 Results

Calculation of Overall Performance Index

= ∑ Sum of all the Differences in the Metrics / Number of metrics used
= ∑ A+B/ 2
= ∑ (1.85+53)/2
= 54.85/2
= **27.43 %**

Thus we achieved 27.43% increase in performance by deactivating the optional components. This enhances the performance of the system as well as increase the compatibility for a low configured system.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Performance has always been a major concern while development of a software. Many researchers have proposed multiple approaches for achieving performance enhancement. The proposed approach is successfully validated using a case study. It is validated that the appropriate selection and deactivation of the optional components can help in increasing the performance of the system. This approach also helps the component based softwares with high resource (i.e. hardware, memory etc) or configuration requirements to become compatible and execute efficiently with low configured systems. This is achieved due to the modular approach of CBSE.

6.2 Contribution

The new approach "Component Selection And Deactivation" is based on the identification, selection and disabling of non participating and non required functional components is proposed. The approach also aims at improving the compatibility among system components and allows systems with high configuration demand run smoothly on low configured systems. At the same time it also provides the disabling of non participating and non required functional components to improve performance of the system. A case study of windows XP is considered to validate the proposed approach. After the analysis, it can be inferred that an increase of 27.43 % is attained in the performance of the system.

6.3 Future Scope

The future work aims to develop an application or an independent component that detects and identifies the optional i.e. non participating and non required functional

components automatically and deactivates them for a period of time to enhance the performance of the system and to increase compatibility among components.

References

[1] Roger S. Pressman,"*Software Engineering: A Practitioner's Approach*," 5th ed. New York: McGraw-Hill, 2001.

[2] X.Cai, M.R. Lyu, K.F. Wong and R.Ko,"Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes," In *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pp. 372-379. IEEE, 2000.

[3] Object Management Group, *OMG Unified Modeling Language Specification*, version 1.5, March 2003, available at www.rational.com/uml/resources/documentation/

[4] J.Gao, H.-S. Tsao and Y.Wu,"*Testing and Quality Assurance for Component Based Software*," London: Artech House, 2003.

[5] Khan, U. A., Al-Bidewi, I. A., & Gupta, K. (2011). Challenges in Component Based Software Engineering as the Technology of the Modern Era. *International Journal of Internet Computing (IJIC), 1*.

[6] Yadav, D., & Kaur, J. Component Based Software's: Issues Related to Test the Compatibility of the Components.

[7] Ian Sommerville,"*Software Engineering:7th edition"*.India:Pearson Education , 2004.

[8] Zschaler, S. (2004). Formal specification of non-functional properties of component-based software. In *Workshop on Models for Non-functional Aspects of Component-Based Software (NfC'04) at UML conference* (Vol. 2004).

[9] Software Performance testing. [online] https://en.wikipedia.org/wiki/Software_performance_testing

[10] Verma, I. Study of Component Based Software Engineering.

[11] IEEE Std 610.12–1990, IEEE Standard Glossary of Software Engineering Terminology IEEE Standards Board, 28 Sept 1990.

[12] "SWEBOK: Guide to the Software Engineering Body of Knowledge," 2004 Edition, IEEE. [online] http://www.swebok.org, Accessed Thu 28 Jun 2007.

[13] J.M. Voas and K.W. Miller, "Software Testability: The New Verification," *IEEE Software*, vol. 12, no. 3, pp. 17–28, May 1995.

[14] W.Zheng, "Applying Test by Contract to Improve Software Component Testability," Technical Report CIIPS_ISERG_TR–2007–02, Centre for Intelligent Information Processing Systems, School of Electrical, Electronic and Computer Engineering, University of Western Australia, WA, Australia, 2007.

[15] R.S. Freedman, "Testability of Software Components," *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 553–564, June 1991.

[16] R. V. Binder, "Design for testability in object-oriented systems," *Communication of ACM*, vol. 37, no. 9, pp. 87–101, Sep 1994, ACM Press.

[17] J.Gao and M. Shih, "A Component Testability Model for Verification and Measurement," *Proc. 29th Annual Intl on Computer Software and Applications Conf (COMPSAC 2005)*, Edinburgh, Scotland, 26–28 July 2005, IEEE Computer Society Press, 2005, pp. 211–218.

[18] Smith, Connie U., and Lloyd G. Williams. "Best practices for software performance engineering." In *Int. CMG Conference*, pp. 83-92. 2003.

[19] Chapter 1 Fundamentals of Engineering for performance . [Online] https://msdn.microsoft.com/en-us/library/ff647781.aspx

[20] Performance engineering . [Online] https://en.wikipedia.org/wiki/Performance_engineering

[21] Muhammad Osama Khan, Ahmed Mateen, Ahsan Raza Sattar. Optimal Performance Model Investigation in Component-Based Software Engineering (CBSE). *American Journal of Software Engineering and Applications.* Vol. 2, No. 6, 2013, pp. 141-149.0

[22] Balsamo, S., Marzolla, M., & Mirandola, R. (2006, August). Efficient performance models in component-based software engineering. In *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on* (pp. 64-71). IEEE.

[23] Kaur, A., Monga, H., & Kaur, M. (2012). Performance Evaluation of Reusable Software Components. *International Journal of Emerging Technology and Advanced Engineering*, *2*(4).

[24] Khan, A. I., Alam, M. M., & Khan, U. A. (2013). Empirical Study of an Improved Component Based Software Development Model using Expert Opinion Technique. *International Journal of Information Technology and Computer Science (IJITCS)*, *5*(8)

[25] Kahkipuro, P. (2001, January). UML-based performance modeling framework for component-based distributed systems. In *Performance Engineering, State of the Art and Current Trends* (pp. 167-184). Springer-Verlag.

[26] Diaconescu, A., & Murphy, J. (2005, November). Automating the performance management of component-based enterprise systems through the use of redundancy. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (pp. 44-53). ACM.

[27] Bertolino, A., & Mirandola, R. (2004). CB-SPE Tool: Putting component-based performance engineering into practice. In *Component-based software engineering* (pp. 233-248). Springer Berlin Heidelberg.

[28] Göbel, S., Pohl, C., Röttger, S., & Zschaler, S. (2004, March). The COMQUAD component model: enabling dynamic selection of implementations by weaving non-functional aspects. In *Proceedings of the 3rd international conference on Aspect-oriented software development* (pp. 74-82). ACM.

List of Publications

i. Jasneet Chawla and Ashima Singh, "Enhancement of Compatibility and Performance of Component Based Softwares by Deactivating Non-Participating / Non-Functional Components", *in European Journal of engineering and technology.* **[Accepted]**

ii. Jasneet Chawla and Ashima Singh, "A Survey on Regression Test Techniques of Component-Based Softwares", in *International Journal of Advanced Research in Computer Science and Software Engineering* .