LOVELY PROFESSIONAL UNIVERSITY

Trade-off Analysis For Evolution Paths in Software Architecture Evolution

A Dissertation Submitted

By

Rajwant Kaur

To

Department of Computer Science and Engineering

In partial fulfillment of the Requirement for the

Award of the Degree of

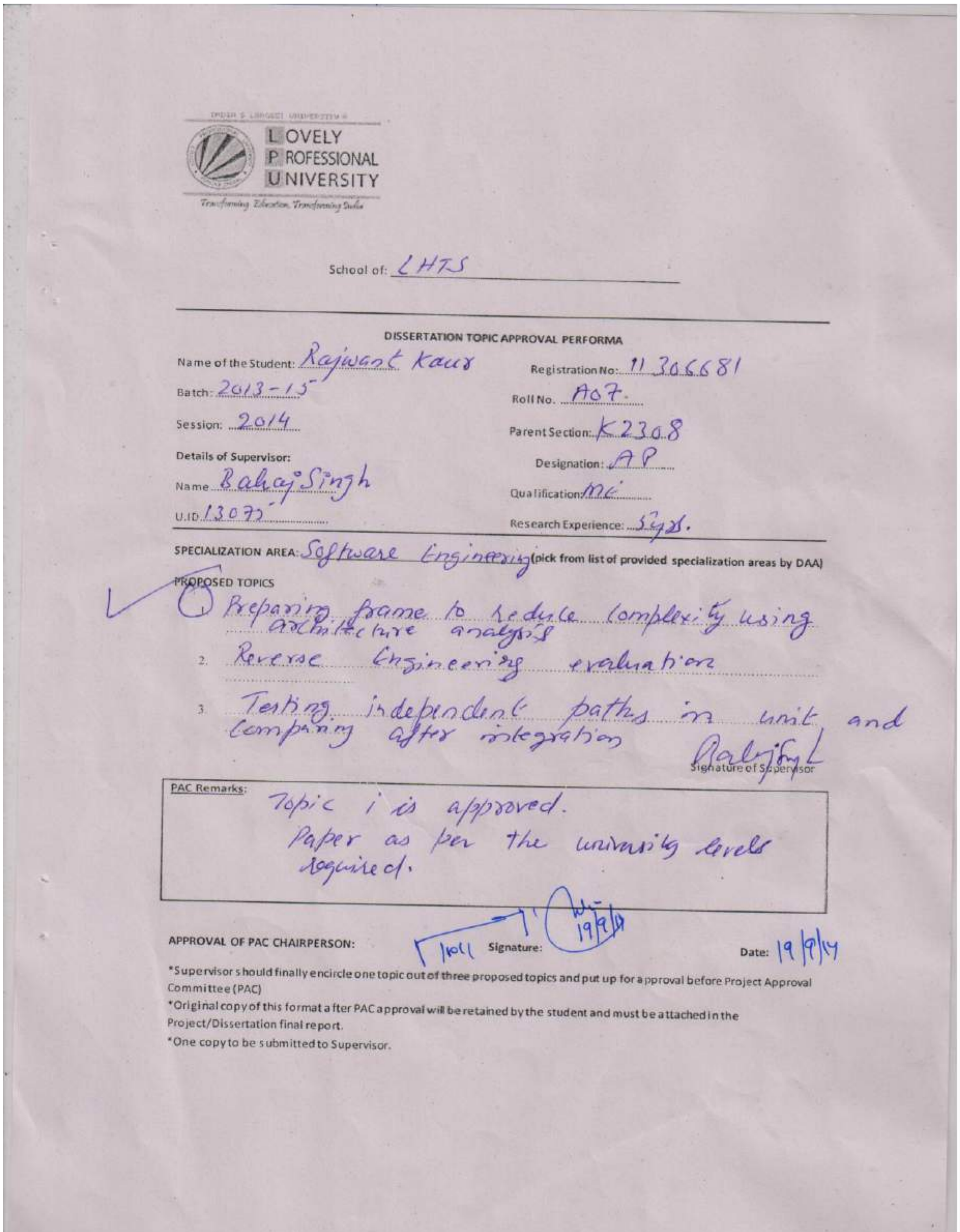
Master of Technology in Computer Science & Engineering

Under the guidance of

Balraj Singh

(May, 2015)

PAC FORM



LOVELY PROFESSIONAL UNIVERSITY

Transforming Education, Transforming India

School of: LHTS

DISSERTATION TOPIC APPROVAL PERFORMA

Name of the Student: Rajwant Kaur	Registration No: 11306681
Batch: 2013-15	Roll No. A07.
Session: 2014	Parent Section: K2308
Details of Supervisor:	Designation: AP
Name: Balraj Singh	Qualification: ME
U.ID: 13075	Research Experience: 5 yrs.

SPECIALIZATION AREA: Software Engineering (pick from list of provided specialization areas by DAA)

PROPOSED TOPICS

1. Preparing frame to reduce complexity using architecture analysis
2. Reverse Engineering evaluation
3. Testing independent paths in unit and comparing after integration

[illegible]
Signature of Supervisor

PAC Remarks:

Topic 1 is approved.
Paper as per the university levels required.

APPROVAL OF PAC CHAIRPERSON: 19/9/14 Signature: [illegible] Date: 19/9/14

*Supervisor should finally encircle one topic out of three proposed topics and put up for approval before Project Approval Committee (PAC)

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to Supervisor.

ABSTRACT

Architecture evolution is the process of developing the architecture of the software initially and then updating it for various reasons. Architecture evolution is basically to improve the quality of the poor architecture. System is made to cope with the changing requirements of the system. System must be able to cope with the planned changes and the unplanned changes. Architecture evolution identifies how to make the architectural evolution by identifying different evolution paths and the choosing the best path out of all the alternative architectures. Architects must identify the alternative evolution paths and the tools to trade off these alternatives. Also the evolution styles are identified .Then modelling of the evolution styles is carried out. Firstly, specifying the architecture then identifying the families of the architecture then specifying the properties of evolution paths then specifying the path constraints and using the evolution functions. Then tradeoff analysis between the alternative evolution paths to choose the best evolution paths that satisfy the requirements.

CERTIFICATE

This is to certify that **Rajwant Kaur** has completed M.Tech dissertation proposal titled "**Trade-off Analysis for Evolution Paths in Software Architecture Evolution"** under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. No part of the dissertation proposal has ever been submitted for any other degree or diploma. The dissertation proposal is fit for the submission and the partial fulfillment of the condition for the award of M.Tech in Information Technology.

Date:

Signature of Advisor:

Name: Balraj Singh

ACKNOWLEGEMENT

The report has been written with the kind guidance and support of my mentor. The satisfaction and happiness that accompanies the successful completion of any task would be incomplete without mentioning the names of people who made it possible, whose constant guidance and encouragement crowns all efforts with our success. I would like to express my deep gratitude and thanks to my mentor Mr**. Balraj Singh,** Lovely Professional University, Phagwara, Punjab for their help and guidance throughout my dissertation work.

DECLARATION

I hereby declare that the dissertation proposal entitled, "**Trade-off Analysis for Evolution Paths in Software Architecture Evolution**" submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: 29.4.2015 **Rajwant Kaur**

Registration no.11306681

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1

INTRODUCTION

1.1 SOFTWARE ARCHITECTURE

Software architecture is [1] collection of software elements, externally visible properties of those elements and the relationship among those elements. Architecture of the software defines the computational components and interaction among those components. It is the fundamental organization of a system with the components, their relationship, environment and principles which are guiding its design and evolution. Software Architecture is high level design. It is basically the overall structure of the software. Software architecture is components and the connections among the components.

1.2 TYPES OF ARCHITECTURE

1) **Perspective architecture**

   A software's perspective architecture [2] involves the design decisions prior to software construction. It is as-intended architecture.

2) **Descriptive architecture**

   A software's descriptive architecture [2] describes how the software has been developed. It is as-implemented architecture.

1.3 ELEMENTS OF SOFTWARE ARCHITECTURE

Software architecture consist [3] of:

1) Components
2) Interconnection
3) Rules of composition
4) Rules of behavior

1.4 IMPORTANCE OF SOFTWARE ARCHITECTURE

1. Software architecture is vehicle for [1] communication among the stakeholders.
2. Architecture enables the earliest design decisions.
   a) Constraints on implementation.

    b) Describes the structure of the organization structure.
    c) Enable the quality attributes.
3. Software Architecture is transferable abstraction of a system.
    a) Allows for template-based development
    b) Basis for training

1.5 ARCHITECTURE IN LIFE CYCLE

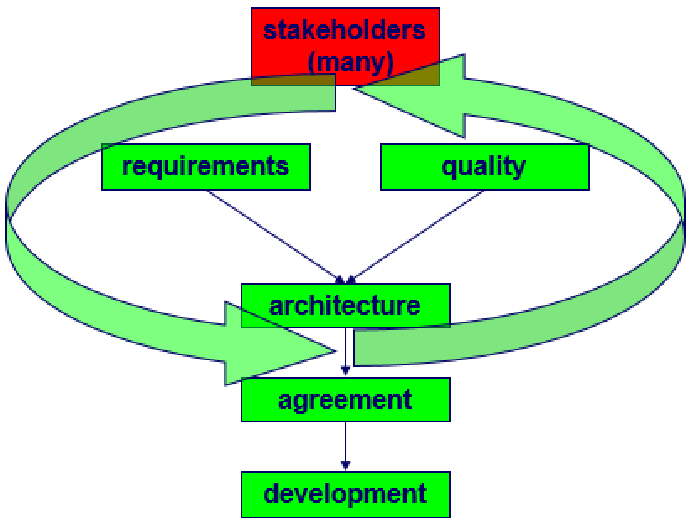


Fig 1.1 Architecture in life cycle

1.6 ARCHITECTURAL STYLES AND ARCHITECTURAL PATTERN

1. **Architectural styles**

An architectural style is [2] collection of architectural design decisions that are applicable in given development context and elicit beneficial qualities in resulting system.

2. **Architectural pattern**

Architecture pattern is [1] description of element and relation type together with set of constraints on how they may be used. It can be considered as the set of constraints on architecture. It applies useful constraints of the architecture and turns on system. The pattern which is mostly used in the distributed system is the three tired system pattern. Architecture pattern is basically set of elements that have same properties and same set of constraints.

1.7 AN ARCHITECTURAL MODEL

Software architecture deals with [4] designing and the implementation of high level structure of software. It is obtained by assembling number of architectural elements in some form to satisfy the functional requirements and non-functional requirements such as reliability, scalability, portability, availability, modifiability. According to Boehm:

Software architecture = {Elements, forms, constraints}

Software architecture deals with abstraction, decomposition and composition with styles and patterns. For describing the architecture of the software we use the model which consist of multiple views and perspectives. In order to deal with the challenging architecture, the model is composed of five main views:

- **The logical view**
- **The process view:** It captures the concurrent and synchronization aspects of a design
- **The physical view:** It deals with the mapping of the software onto the hardware.
- **The development view:** It describes the organization of software in its environment

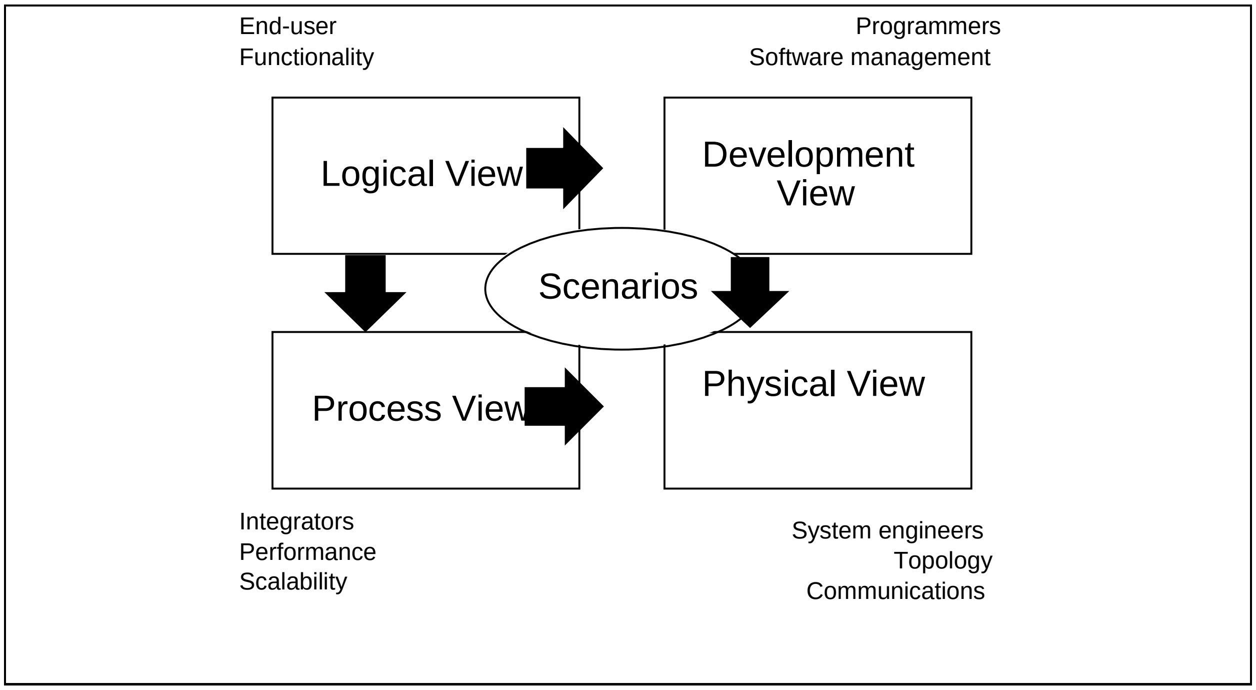


Fig 1.2 “4+1” view model

1.8 SOFTWARE ARCHITECTURE EVOLUTION:

Architecture evolution [5] is process of developing software and repeatedly updating it for various reasons. To survive in the competitive market, the existing system must be able to accommodate changing requirements which can be accomplished by restructuring of the software. Broadly, we can say it must follow evolution process. In most of the cases, these changes takes longer than expected time so the architecture must be developed in such a way with the support of evolution plan so that change in the architecture can be achieved in a given time frame. The basic constraints necessitate the development of the evolution architecture that must address issue of achieving business goals through the evolution within time frame and quality requirements. There is need to keep in mind principled trade-offs between time and development effort. Systems are developed to provide the functions over long time but planning productivity over long time is difficult since market, social, technology forces are not constant. Systems may able to cope with these forces by making changes. These changes are made to increase the utility of system. Engineering decisions [6] are taken by keeping in mind economic considerations.

- Enhancement of quality of architecture
- System is made to cope with changing requirements of business
- Investment in architecture
- Different alternative architectures that provide support to cope with changing requirements
- System is made dealing with planned change
- System is made to dealing with unplanned change
- Independent changes are described.
- Alternative architecture evolution paths
- Making principle tradeoffs between evolution paths
- Make surety that evolution path is correct and makes sense
- Tools used for architecture evolution

Architecture evolution identifies how to make the architectural evolution by identifying different evolution paths and the choosing the best path out of all the alternative architectures. Architects must identify the alternative evolution paths and the tools to trade off these alternatives. We can consider the [5] architecture evolution as choosing an evolution path in state machine:

- Nodes represent individual architecture
- Link between the two nodes represent the communication
- Initial state is the architecture of current system
- Final state is the architecture of target system
- Evolution path is path from initial state to final state.

1.9 LEGACY SYSTEM STRUCTURE AND DISTRIBUTION

For ideal distribution,there [7] is clear seperation between user interface,system services and system data management.

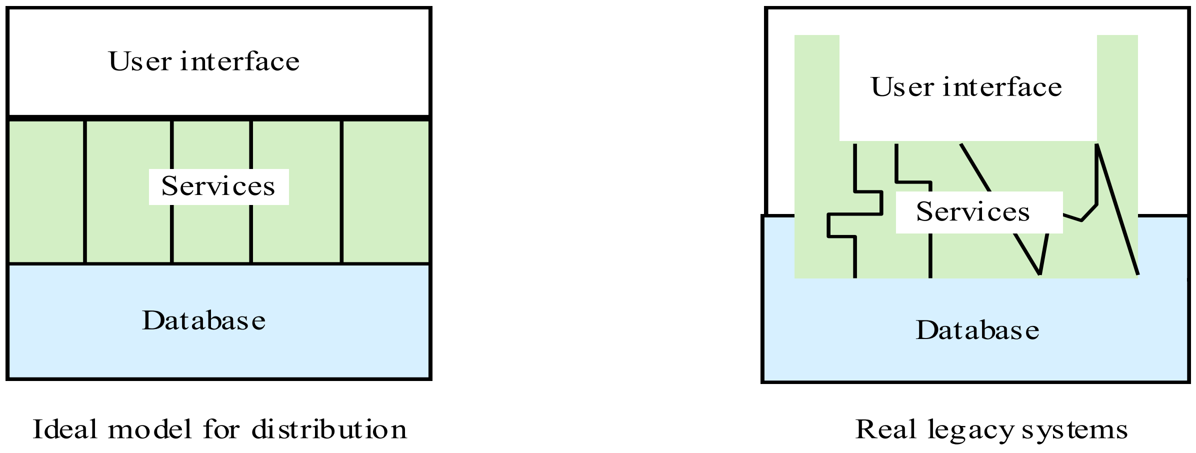


Fig 1.3 Legacy system structure

Legacy System Distribution

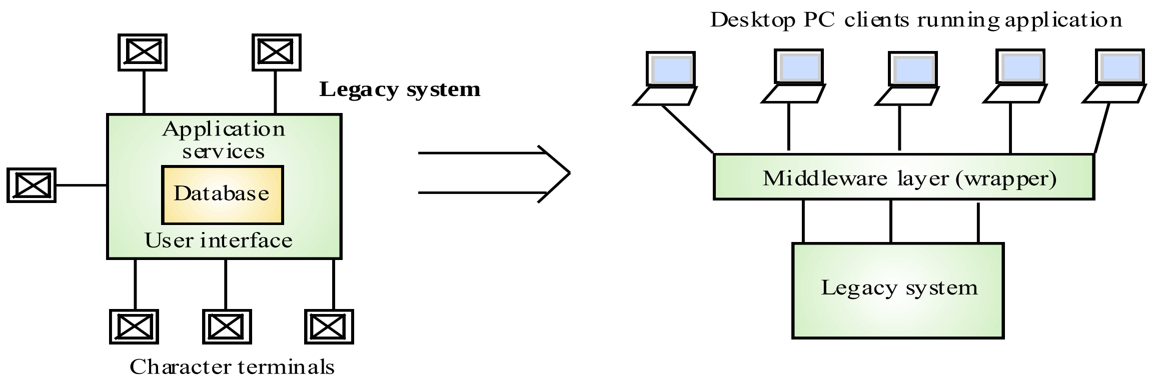


Fig 1.4 Legacy system distribution

1.10 TYPES OF SOFTWARE ARCHITECTURE EVOLUTION

There are two types of software evolution:

1) **Internal Evolution**

   Internal evolution includes [8] the change in the models the changes in the topology of the components and interactions as they are created or destroyed during execution. It captures the dynamics of the system.

2) **External Evolution**

   External evolution includes the change in specification of a components and the interactions which are needed to deal with the new requirements of a software. It includes the use of software architecture.

1.11 SOFTWARE EVOLVABILITY

Software Evolvability is the [8] ability of the software to easily accommodate the changes. There are two software architecture evolvability analysis:

a) **Qualitative Evolvability analysis:**

   This method focuses on the improvement of the capability of being able to understand and analyze systematically the impact of the change stimuli on the software architecture evolution.

b) **Quantitative Evolvability analysis:**

   This method focuses on the evolvability concerns of the stakeholders and the impact of the architectural solutions on evolvability

There are ways to define ways to analyze the ability to evolve software:

1) Identification of the characteristics which are important for evolvability of software system.
2) Identification of the challenges of the software architecture evolution and the future research studies of software architecture.
3) Systematic assessment of the software evolvability.
4) Describe how evolvability is addressed in open source software evolution.

.1.12 TRADE-OFF ANALYSIS BETWEEN EVOLUTION PATHS

To analyze the [5] trade-off between alternative evolution paths, following are the constraints.

- Benefits: Services provided, Quality attributes
- Cost: time, effort
- Then we will calculate the utility of paths
- The problem: select the path from initial state to final state that maximizes the utility

Trade-off analysis of alternative evolution paths to choose best evolution path.

1.13 SOFTWARE ARCHITECTURE EVOLUTION MODEL

In this model, we consider software architecture as levels of abstraction .This model provides the software architecture at all of these abstraction levels. In this model, components and interfaces are first class entities.

1.14 DIMENSIONS OF ARCHITECTURAL EVOLUTION

1) Evolution time

a) **Static evolution:**

Static evolution [9] is that in which modifications are made in architecture at specific time. Once the system is implemented then system is stopped to do any modifications. It comprises of two mechanisms

i. **Instantiation:**

In this, it provides the difference between component and connectors. Component types encapsulate functionality into reusable blocks and connector type that encapsulates component communication.

ii. **Inheritance and sub typing:**

Inheritance allows the reuse of the model itself and sub typing involves the reuse of objects in model.

b) **Dynamic evolution:**

It is also called active evolution and runtime evolution. It means making the modification in system during the runtime. Dynamic evolution may be planned or unplanned.

2) **Evolution Resource**

   a) **Architectural resource:** In which [10] evolution resource make a difference between local changes which are components, connectors, interfaces and architectural changes which is configuration or system topology.

   b) **Domain resource:** when you are moving to a new technical infrastructure such as moving from a client server architecture to service oriented architecture.

   c) **Meta model resource:** This considers the case of architecture evolution of Meta model.

3) **Type of resource**

   a) **Explicit resource:** Resources [10] which can be modeled explicitly.

   b) **Implicit resources:** Resources which are not first class entities.

   c) **Predefined resources:** Resources which are previously defined by ADL and architects.

1.15 EVOLUTION PATH AND PLANNING

Evolution path is one of the [10] important aspects of the software evolution. It describes the sequences of the architecture evolution releases from the initial architecture to the target architecture. A sequence of releases from the initial architecture to the final architecture are described and analyzed. The architecture evolution has two types depending upon target architecture:

a) **Open evolution:** Open evolution has high not certain. The market, technology and business conditions do not allow the architect to make clear target architecture.

b) **Closed evolution:** In this the properties of the target system are known. Closed evolution deals with changes in the technology.

1.16 ARCHITECTURAL CHANGES

There are three types of architectural changes:

1) **Easy Architectural Changes:** The easiest architectural [11] changes include changing of the network application program interface. Since applications has embedded DNS so change in the name of system require many changes in the applications. This problem is solved out by having applications that treats names as semantic.

2) **Hard Architectural Changes:** The hardest architectural [11] change include changing IP. Firstly you require that all API pass names and address names so that applications are protected from the changes. Then we will separate intra domain addressing from inter domain addressing. Then we can embed inter domain addressing in many of forms of packet delivery.
3) **Complicated Architectural Changes:** The most complicated architectural change is changing the inter domain routing. These changes are the most complicated changes which involves the domain routing.

1.17 DESIGN PRINCIPLES ENABLING ARCHITECTURE EVOLUTION

There are three design principles:

1. **Layer of Indirection for Flexibility:** For the [11]aspects of architecture which are hard to change. We make layer of indirection which we can see than version number which provides structure.
2. **Modularity to minimize scope changes:** For easy evolution there should be tight modularity so that changes can make less affect.
3. **Extensibility to accommodate new functionality:** The system must be able to add new functionality.

1.18 DEVELOPMENT PROCESS IN ARCHITECTURE EVOLUTION

1. **Manipulation of evolution model:** In the development process, firstly we have to be familiar with Magic Draw fundamentals. We must be familiar with the basics of the Magic Draw.
2. **Evolution paths identification:** In this we have to give the reason why we are using evolution model and identifying the alternative evolution paths and these paths are analyzed.
3. **Interface for applying evolution operator:** In this we have to know about the user interface functionalities of Magic Draw.
4. **Operator parser:** In this we will specify our operators by configurable files that can be parsed and interpreted at a time.

5. **Positioning of Presentation Elements:** In this we will be developing a tool like Magic Draw that will help in dealing with the positioning of the presentation elements.
6. **Evolution path constraints:** Evolution path constraints like operators are defined in configuration file that is read when it is loaded.

1.19 EVOLUTION STRATEGIES

1. **Continued maintenance:** It involves [13] understanding of how the system is designed, implemented, documented and its environment and then continuously updating the system according to market needs.

   Advantage

   a. It is used to update the system continuously so that it provide its services

   Disadvantage

   a. It requires the knowledge of legacy environment.
2. **Reengineering:** When developing a new system is too expensive and current system is not maintainable. It includes transformation of current system to improve the quality of the system by improving its capability, performance and functionality
3. **Replacement:** It involves new system to take control on processing of the old system using data written by the old system.

   Advantages:

   a. Old code is removed
   b. Compatibility with new technology
   c. It may remove out of date working practices associated with old system

   Disadvantages:

   a. Too expensive
   b. Rapid introduction of new system is risky
   c. Too much of risk if the system fails

1.20 AN EVOLVING SYSTEM

Software architecture describes [14] the system in terms of the elements and the interaction between elements. Active software architecture is designed to be dynamic in that structure and the interactions are changeable during execution. Changes are required to make the software updated and enhanced.

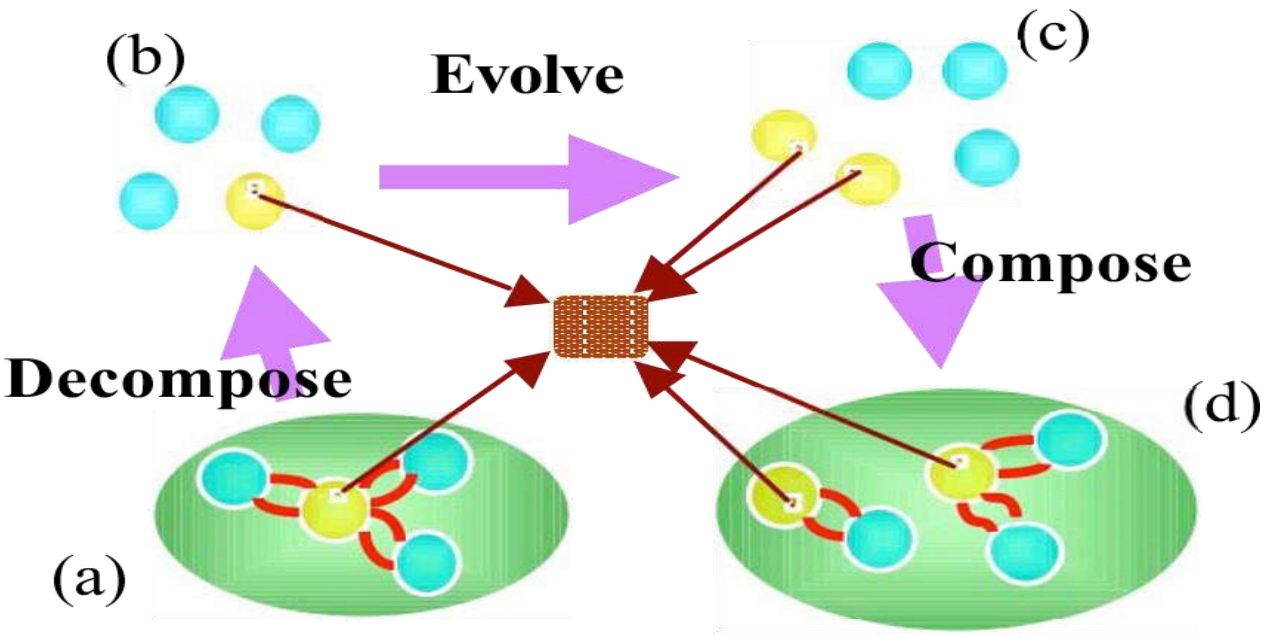


Fig 1.5 An Evolving System

1. **At stage (a)**

   System is made up of the three components of similar types (clients) communicating with the components of another types (server) that has access to some data.

2. **At stage (b)**

   The system has been decomposed to yield the individual components with the server still maintaining its access to the data. The next stage (c) sees the components

3. **At stage (c)**

   In this stage, the components are evolved that we have three clients and two server both of which maintains access to shared data.

4. **At stage (d)**

At stage (d) five components are composed to form the new evolved system so one client interacts with one server and the other two clients interact with other server.

1.21 THE ATAM

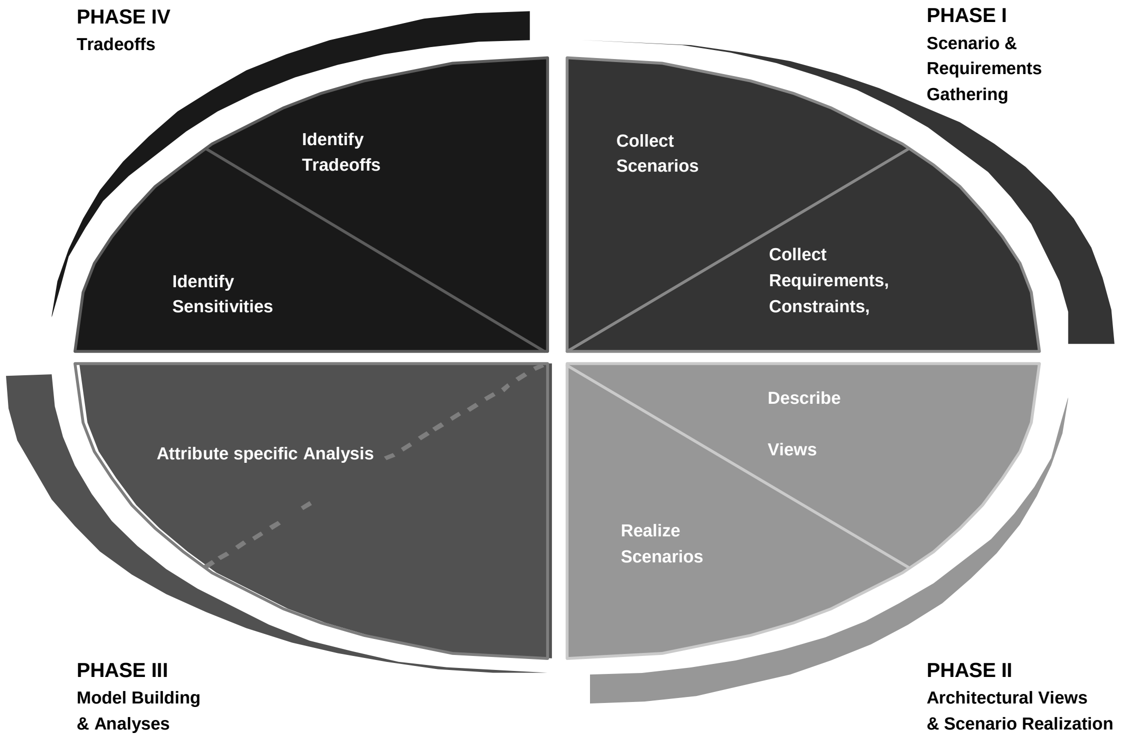

PHASE IV
Tradeoffs
PHASE I
Scenario & Requirements Gathering
Identify Tradeoffs
Collect Scenarios
Identify Sensitivities
Collect Requirements, Constraints,
Attribute specific Analysis
Describe
Views
Realize Scenarios
PHASE III
Model Building & Analyses
PHASE II
Architectural Views & Scenario Realization

Fig 1.6 The Spiral model

The Architecture Trade-off Analysis Method includes four areas of the activities. These are: Collecting scenario and gathering requirements, describing the architectural views, attribute Specific analyses, sensitivities are identified and identifying tradeoffs.

The Steps of the Method [15]**:**

This method includes four areas of the activities. These are:

Step 1: Collect Scenarios

The step includes collect the scenarios [15] of the system usage from the group of stakeholders. This step involves the activities that the system should support and provides the communication between the stakeholders and quality and functional requirements. The stakeholders communicate with each other in order to collect the scenarios and after that the requirement are collected and analyzed.

Step 2: Collect Requirements/Constraints/Environment This step in method is to identify the quality and functional requirements of the system, quality constraints and the environment of the system. Its main purpose is to identify the requirements.

Step 3: Describe Architectural Views

The design, requirements, scenario principle are used to the build candidate architectures and constrain the space of design possibilities.

Step 4: Attribute-Specific Analyses

If the requirements of the system has been identified and scenarios have been collected and the initial architecture has been proposed then each quality attribute is analyzed with respect to each architecture.

Step 5: Identify Sensitivities

Any modelled values that are significantly affected by a change to the architecture are considered to be sensitivity points.

Step 6: Identify Tradeoffs

The next step of the method is to find the architectural tradeoff points.

CHAPTER-2

LITERATURE REVIEW

Barnes & Schmerl et.al, [5] presents how the architecture evolution increase the utility of the architecture. To survive in the competitive market, the existing system must be able to accommodate changing the requirements which can be accomplish by restructuring of the software The author describes how to make evolution to achieve goals of business by using limited resources, infrastructure required to get the target system and principle tradeoffs between time and development effort and how to make and represent the evolution plan. The author describes an evolution path which are considered as class entities and support reuse, correctness and completeness checking. Evolution path is a path from initial state to final state. This paper describes evolution styles, Evolution style is a family of the evolution paths that has similar properties and has same set of constraints .The main idea is that by identifying the evolution paths of the particular families we can define the constraints that each path in family must follow. An evolution style contains many evolution paths having same properties. This paper describes the modeling of evolution style by specifying the architecture, specifying the families of architecture, specify evolution path properties, relating architecture in path, specify path constraints and using evaluation paths **(Barnes & Schmerl et.al, 2012)**.

Goltz& Reussner et.al, describes [16] the concepts of software engineering, methodologies, and that tools helps in the evolution of complex software system and challenges in co-evolution of the software system. The first challenge is coping with the technologies which has newly emerged. The second challenge is that there is aging of the software in which the software does not continuously adapt the changes so it will age with respect to the environment. The third challenge is that there is less understanding between the functional and the quality requirements the software system .There is also the challenge of the time and the complexity of the developing an application. The author describes the evolution lifecycle which include design phase, construction phase and the operation phase. The author describes the structure of three themes which are used to integrate the activities in the development, operation, monitoring and maintenance **(Goltz& Reussner et.al, 2014)**

Souza& Mylopoulos et.al, describes [17]the importance of the evolution systems and the adapted systems which are related concepts. Adaptive systems have an architecture mechanism which help them to change their behavior at runtime in order to fulfill the requirements. Support to the adaption and the evolution system is important in the requirement engineering since the software change in the software system is triggered by change in the requirements of the stakeholders. The author describes the goal oriented requirement engineering in which the requirements of the stakeholders are taken as a goals. Goal includes the tasks which will operate the goals. It also include the soft goals which are the non-functional requirements of the system **(Souza& Mylopoulos et.al, 2012).**

Dougherty& White et.al, proposes [18] that software evolution helps to extend the functionality and it is life of Distributed Real Time and Embedded (DRE) Systems. The author describes the 4 aspects to evolve legacy DRE system configurations. Firstly, we describe software evolution analysis with resources technique for converting legacy DRE system configurations, and then analyzing the replacement components. Secondly, we describe the formal steps for accessing the validity of evolved system configurations. Third, we will get the results of these experiments which is that some algorithms measure better than other algorithms, too large or small problems give nearby solutions and no algorithm is universally superior **(Dougherty& White et.al, 2011)**

Ghezzi & Gall, describes [19]the concepts of Software Analysis as a Service which is the platform for analyzing the evolution of software. Software analysis are provided has services that can be accessed and composed into the work flows. The author describes that the data of evolved software is stored in repositories such as version control, bug and tracking of the issues is very important to composing this analysis into the workflow which consists of custom made model language and composition infrastructure for service offering. This approach helps the user to execute in more and accurate way and in a flexible manner which is give correct results **(Ghezzi & Gall, 2013).**

Medvidovic & Jakobac, [20] describe the concept of the architectural recovery. A concept of architectural recovery is used to cope with this problem. Architectural recovery is a process of extracting architecture of system from implementation. The main goal of architectural recovery is recovery of system architecture fully. The author describes an approach to architectural recovery which is called focus which is light weight approach to OO systems. The architectures are recovered incrementally. The basic function of focus is to recover the three architectural building blocks. The focus approach is based on assumption that little or no reliable document exists for system which is being modified .Focus approach focuses on three facets. Focus uses the requirements of system evolution to recover the fragments of system of architecture which is affected by evolution. Secondly it recovers the architectural notion. Focus not only deals with the architectural erosion but may lead to re -architect the architecture **(Medvidovic & Jakobac, 2006).**

Godfrey & German, describes [21] how the software evolution is different from the software maintenance. Software evolution provides the concept of essential changes that maintenance do not provide. Software architecture evolution is making new architectures from old architectures. The author describes the Lehman's laws which are continuous change, complexity is increased, large program, conservation organization stability, conservation of familiarity, growth continuity, quality decline, feedback system. The author describes lifespan of the software as initial development, active evolution, servicing, phase out. This paper describes challenges of model building. These challenges are how do we proceed, how we can create general laws of software evolution, how will we anticipate how system will respond to environmental pressure, and modeling of economic tradeoffs and risks **(Godfrey & German, 2005).**

Zhu& Aurum et.al, [22]describes the evaluation of software architecture involving analyzing the different architectural design alternatives. Analytic Hierarchy Process AHP is used to give ranking to alternative architectures .The ranking which is produced is sensitive i.e. if there is smallest change in the priority weights can change final order of the alternative architectures. AHP takes into account the priority weights of alternative architectures for individual quality attributes and priority weights among quality attributes. ATAM is architectural analysis method in which quality constraints of software are gathered from stakeholders and by analyzing the requirements .If there is conflict between stakeholders perspective then aggregation is used to get the final result .The elements which affect architecture are identified and if some of these elements affect different quality attributes .These elements attributes are considered as tradeoff points .These points leads to the decision making in tradeoff .Stakeholders use Win Win model to resolve their conflicts.. This paper also describes non-functional requirements. The requirements of quality attributes is known as Non Functional Requirements **(Zhu& Aurum et.al, 2005).**

Lung& Bot et.al, have presented the [23] approach for accessing the software architecture for evolution and reuse. The main idea is to select existing architecture for evolution or reuse in the future project in same problem .This paper describes the framework for gathering various types of information and then analyzing that information and then analyzing the software architectures and comparing them. Various architectural views are static view, map view, dynamic view, resource view. This paper describes the objectives of the stakeholders, architectural objectives and quality attributes. This paper describes the architectural styles which consist of many evolution paths having similar properties. This paper provides the better understanding of target system, communication among various stakeholders **(Lung& Bot et.al, 1997)**

Michele & Giuseppe et.al, presents [24] an approach for automate architecture recovery process of the software systems. This approach is built on information retrieval and clustering techniques, and, in particular, uses Latent Semantic Indexing (LSI) to get similarities among software entities and the k-means clustering algorithm to form groups of software entities that implement similar functionality. In order to improve computational time in the context of the software evolution and then reduce energy waste, the architecture recovery process can be also applied by using fold-in and fold-out mechanisms that, respectively, add and remove software entities to the LSI representation of the understudy software system. The approach has been implemented in a prototype of a supporting software system as an Eclipse plug-in **(Michele & Giuseppe, 2011).**

Antonia & Paola et.al, Describes [25] the role of software architecture in testing and analysis. SA is means for managing complexity and improving reuse, by supporting the decomposition of a system into its high level components and their interconnections. The SA model correctly implements the desired requirements, ABA aims at using the produced architectural artifacts to select the right architecture that at best satisfies expected system qualities and stakeholder concerns. The author describes how to analyze an architecture or architectural model with respect to functional and non- functional qualities. This papers describes about the Architecture evaluation (AE) consists in evaluating the software architecture compliance to quality requirements .The author describes about how to architect the fault tolerant systems **(Antonia & Paola, 2013).**

Martens & Heiko et.al, presents [26] that to design a different architectures that provides a good trade-off between several quality attributes is difficult. The author explains that the manual solutions takes more time and can be error prone. The author presents an automated approach for searching the good design solutions. The author begins with initial development of the architecture and evaluation of the architectural model and then applying the genetic algorithm to software architecture modelled with the Palladio Component Model. It provides the support to the quantitative performance, reliability and cost prediction. The author describes how to search the problem formulation and Evolutionary optimization **(Martens & Heiko, 2010).**

Wang & Chen, presents [27] the reliability-oriented evolution method. The main focus of this method is to improve the reliability of the architecture of the software by analyzing the contribution degree of the component. The contribution degree of the components is considered because various components play different roles in reliability-oriented architecture of the software. The author applies the reliability-oriented evolution method of the software architecture based on contribution degree of the component is applied to ATM system. The evolution process shows that which component is playing an important role in the process of reliability-oriented evolution of the software. The evolution process is illustrated by the method, and it shows that which component is playing an important or crucial role in the process of reliability-oriented evolution of software architecture. The result will show the improvement in the reliability of the software architecture **(Wang & Chen, 2012).**

Franco & Barbosa et.al, presents [28] the importance of the software architecture which is used to make a structure and the behavior of the system and supports the early decisions making and the other quality attributes. There is need for the system to be evolved over time so software architecture is used as early indicator of the impact of planned evolution on the quality attributes. The author proposes an automated approach to evaluate the effect of the reliability of the architecture evolution. The approach provides the correct information to the artifacts to make prediction about the impact of the reliability of the components and reliability of the structure of the system. The analysis of the possible architecture modifications is performed by translating the system architecture description written in Architecture Description Language to a model. The author also describes the different behavior of the failures beside the probability of failure on demand. With these enhancements in approach it would give support to practitioners and researchers to avoid, prevent and detect undesired or infeasible architectural redesigns which could result in a loss in overall system reliability **(Franco & Barbosa).**

Wermelinger & Lozano et.al, presents [29] the importance of the historical perspective on the generic laws, guidelines and principles such as Lehman’s laws of software evolution and the Martin’s design principles to gain the multifaceted process and the assessment of the structure of the systems architecture evolution. The author presents the structural model with historical metrics.

The author describes the assessment of Eclipse SDK. The main focus on checking generic principles is to see whether there are some lessons which can be learned of the architecture evolution and to get the thorough detail about the applicability of such principles **(Wermelinger & Lozano, 2011).**

Semetti & Zereik et.al, presents [30] the goals of planet explorations which are reached through the development of the smart robots which are able to work on their own without requiring the human supervision but with safety and reliability. The author presents the development of the architectures which deals with both hardware and software issues. The author presents an architecture Test Bench for Robotics and Autonomy which develops the test bench for these missions. This architecture helps in researches regarding planet explorations **(Semetti & Zereik, 2011).**

Barnes & Garlen presents [31] that how the theoretical models provide software architects with support for arrangement and carrying out the most evolution of software systems. The author describes the challenges in development of a tool as a plug-in to the architecture model which includes firstly setting up the project, then the functionality of magic draw is extended and then the development process starts. The author describes the development of a prototype which includes operation of evolution model, identification of evolution paths, interface for applying the operators, operator parser, metadata handling, evolution path constraints. The author described the problems for development of the tools for architecture evolution modeling which includes that Magic Draw turns out to be less flexible and difficulty for a small team to learn the concept quickly. The author describes the key factors which must be weigh while considering framework **(Barnes & Garlen)**.

Guan & Chen et.al, presents a method to [32] estimate the relationship between software reliability and software development cost with regards to the complexity for developing the software system and the size of the software intended to develop during implementation phase of the software development life cycle. Software development process consist of a several phases where estimation cost means different things with respect to the different phases of software development life cycle.

The author presents the two assumptions of RCR model. The first assumption is Software development cost is inversely proportional to the number of system failures. The second assumption is Software development cost is proportional to the complexity and size of the software system. The author makes the use of data to validate the correctness of the proposed model by comparing the result with other existing models **(Guan & Chen, 2010).**

Hesse, presents the meaning of the [33] ontology which stands for formal specification of the shared conceptualization. Software Engineering is a field where conceptualization plays a vital role that is in the early phases of the software development. Ontology deals with the possibilities and the conditions of being. The author presents the ontology development by three phases which are Analysis, Design, Implementation, Operational use. The author compares the conceptual models with the corresponding life cycle models **(Hesse)**

CHAPTER-3

PRESENT WORK

In this chapter, we are going to present the problem of our research work, its objectives, the methodology that we used for our purposed approach and the introduction of the developed tool. In the 3.1 section we explain how we formulated our problem and what the approach we are going to use. In the 3.2 and 3.3 section the objectives and the methodology of the work done. In the methodology the flow of our work with the help of flow chart is explained.

3.1 PROBLEM FORMULATION

Architecture evolution [5] is process of developing software and repeatedly updating it for various reasons. To survive in the competitive market, the existing system must be able to accommodate changing requirements which can be accomplished by restructuring of the software. Broadly, we can say it must follow evolution process.

In our research we are going to identify the different evolution paths of architecture of Library Management System and choosing the best evolution paths on the basis of the quality attributes like reliability, accessibility, availability etc.

3.2 OBJECTIVE

Objectives of work:

1) To provide an evaluation process for selecting the best possible path from the different alternatives by performing the trade-off analysis.

2) Proposing quality attributes for analyzing the software architecture from the evolution point of view.

3) Enhancing the quality of the architecture.

Trade-off analysis of alternative evolution paths to choose best evolution path from alternative paths on the basis of the quality attributes. The path to be chosen should be conforming to all the requirements and should impose quality constraints. Then the identification of the evolution styles is performed which is collection of evolution paths.

3.3 RESEARCH METHODOLOGY

3.3.1 Steps for the approach are

Step1: Firstly we will specify the architecture and the families of the architectures.

Step 2: Then we will identify the alternative evolution paths.

Step3: Then we will specify the properties of evolution paths in terms of quality attributes.

Step4: Then evaluating the architectures in a path

Step5: After that we will specify path constraints

Step6: Finally we will use evaluation function to perform trade-off analysis between the alternative evolution paths.

Step7: Then we will choose the best path among all alternative paths.

In the first step, we will identify the architecture. Software architecture [5] use some formal modeling language such as architecture description language ADL. These languages are used to provide the structure of the software. An architecture is represented as a graph where vertices represent the piece of software system and edges describe how these pieces are related to each other. In second step, we will specify the families of the architecture. Architectural styles are used to represent the families of the architecture. Architecture styles is defined as the vocabulary of elements and the set of constraints on that elements. In the third step, we will identify the alternative evolution paths. All the alternative evolution paths are identified. In the fourth step, we will identify the properties of the evolution paths. Evolution path consist of nodes and the transitions with list of the properties. The information used by the constraints is contained by the properties. An evolution style specify list of the properties which are expected to be values on the nodes and the transitions. In fifth step, we will be relating the architecture. Given a node Ni and its successor Ni+1, most of the architectural elements are same in Ni and Ni+1.We use symbol of evolutionary identity. Elements which are evolutionary identical they would be assigned same evId. In sixth step, we will specify the path constraints. Path constraints provides the evolution paths which are used in evolution style. We use the linear temporal logic (LTL) to specify path constraints. In seventh step, we will use the evolution function. In eight step, we will do tradeoff analysis between the evolution paths and then select the best path among alternative evolution paths.

3.3.2 Flow Chart of proposed approach

~

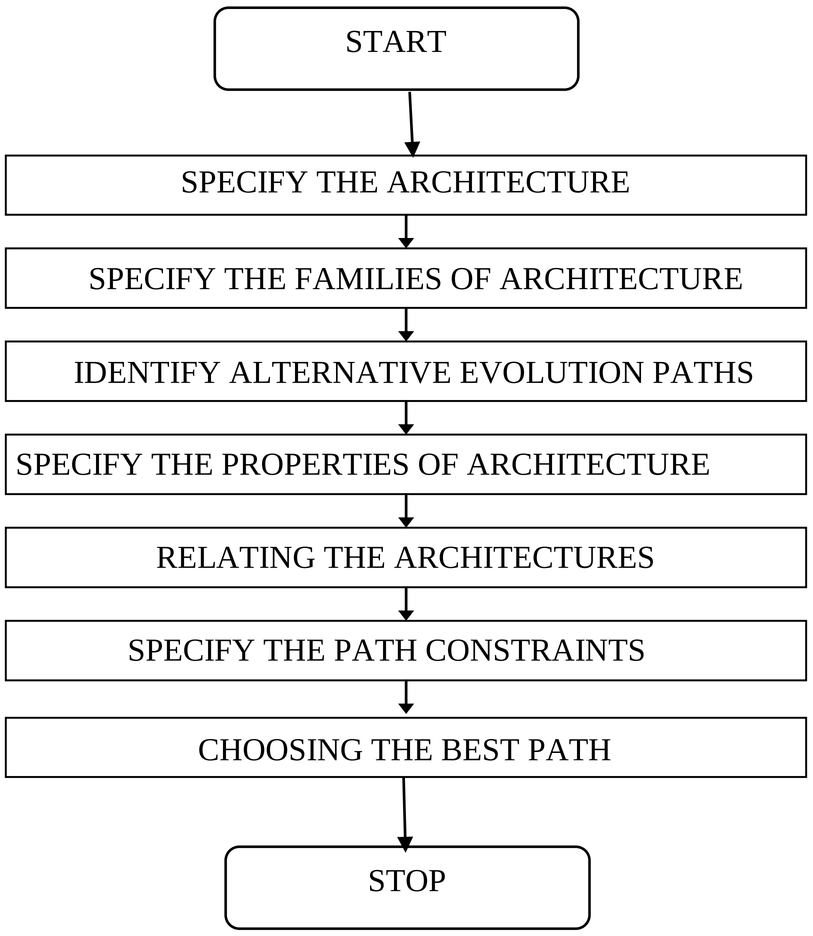

START
SPECIFY THE ARCHITECTURE
SPECIFY THE FAMILIES OF ARCHITECTURE
IDENTIFY ALTERNATIVE EVOLUTION PATHS
SPECIFY THE PROPERTIES OF ARCHITECTURE
RELATING THE ARCHITECTURES
SPECIFY THE PATH CONSTRAINTS
CHOOSING THE BEST PATH
STOP

Fig 3.1 Flow chart of the proposed approach

3.4 ECLIPSE JUNO

Eclipse is [34] general purpose open platform that provide the development of third party plug-ins. It is known as Integrated Development Environment.

3.4.1 Features of Eclipse:

1) It provides tool for coding, building, [34] running and debugging the application.

2) It is open source software.
3) It contains a workspace and the extensible plugin for customizing the environment.
4) It is originally designed for java but it supports many other languages such as C, C++.
5) Eclipse provides the rich client platform for development of the general purpose application.
6) Eclipse uses plug-ins to provide all the functionality within and on the top of the runtime system.
7) Eclipse is written in java and thus it needs installed JRE or JDK in which to execute.
8) Eclipse which is written in java can be used to develop the applications.
9) It can also be used to develop the packages.
10) The Eclipse Software Development Kit (SDK) which includes the java development tools which are designed for the java developers.
11) User can extend the abilities by installing plug-ins for Eclipse platform.
12) Eclipse makes the use of workspace.
13) In Eclipse all of your code [34] live under workspace.
14) A Workspace is nothing more than a location where we store our source code and where eclipse will write out our preferences.
15) Eclipse allows you to have multiple workspaces.

3.4.2 Creating java project

Enter the name of the [34] project and then click.

3.4.3 Creating a class:

Right click on the src folder and click [34] on the class and then new class wizard will open.

From this you can specify:

- Package
- Class name
- superclass

3.5 MAGIC DRAW

Magic Draw is modelling tool [35] with collaboration support. It is a dynamic and versatile development tool that provides analysis and designing of object oriented systems and databases.

Magic Draw is a Unified Modelling Tools and CASE tool with the team work support which is designed for the software analysts, programmers, business analysts. It is a versatile development tool that provides the analysis and design of Object Oriented systems and databases.

3.5.1 Features of Magic Draw:

1) **Domain specific language:**
   Magic Draw [36]allows the customization of multiple GUI, model initialization, adding semantic rules and creating one's own specification dialogs and smart manipulators.
2) **Model decomposition:**
   Model decomposition is a function which can split projects and other work into independent parts. Read-Write modules allows module editing of the fragmented model.
3) **Template based documentation generation:**
   Customizable templates can be created in style and format preferred by the user. Reports can be exported into variety of the file formats. Magic Draw supports MS Word and Open Document Format template.
4) **Model refactoring:**
   Model refactoring is a technique used for modifying or improving an existing model. There are refactoring functions in Magic Draw such as Element conversion and Diagram extraction.
5) **Analysis facilities:**
   The following analysis facilities are there in Magic Draw such as:
   i. Traceability between different levels of abstraction
   ii. Visual model differencing allows viewing changes made between two different version of model
6) **Transformation:**

   Magic Draw provides the conversion of UML model into a specific XML Schema and DB models and vice versa and any to any transformation.

3.5.2 Magic Draw and Eclipse Integration functionality:

Magic Draw is integrated into Eclipse [35] environment which provides UML modelling using a current Eclipse project. This integration allows the synchronization of your model in Magic Draw you’re your code in Eclipse in same environment. Magic Draw uses the same windowing style as eclipse. The integration provides all of the Magic Draw functionality along with UML data update according to source code. To start the integration tool from main menu:

1) Click **Tools** > **Integrations**. The **Integration** dialog open
2) Select Eclipse and click the **Integrate/Unintegrate** button.

Opening Magic Draw project from Eclipse:

1. From the Eclipse shortcut menu, click on Magic Draw project.
2. Magic Draw starts and [35] new Project Wizard dialog box opens.
3. Specify the name of the project and location of the project.
4. Then choose the project type: local one or teamwork.
5. Choose which profile you want to import

CHAPTER -4

RESULTS AND DISCUSSIONS

In this chapter we have presented various results of our purposed approach with snapshots and compared with the existing work done. In this chapter we will compare the two different architectures of the Library Management System.

MATHEMATICAL FORMULAS USED

1) Adaptability

Adaptability means change in system to accommodate the change in its environment. Adaption of the software architecture(S) is caused by the change (δ) from old environment (E) to a new environment (É) and results in new software architecture (Ś) that meets the needs of a new environment (É).

Adaption: E *É*S →Ś where meet (Ś, need (É))

Adaption involves three tasks:

a) Ability to recognize the change δ.
b) Ability to determine change δ to be made to software architecture S according to δ.
c) Ability to effect the change in order to generate new system Ś.

EnvChangeRecognition: É-E →δ

SysChangeRecognition: δ*S→Ś

SysChange:δ*S→Ś

2) Reliability

Reliability is the ability of a system [36] to perform its required functions under stated conditions for a specified period of time. Software architecture comprises of three activities:

a) Error prevention
b) Fault detection and removal
c) Measures to maximize the reliability

Metrics of software reliability

a) MEAN TIME TO FAILURE (MTTF)

   MTTF is the time interval between [37] successive failure .A MTTF of 200 means that one failure can be expected every 200 time units. The time units are totally dependent on the software and even specified in number of the transactions. MTTF is relevant for the software with long transactions

b) MEAN TIME TO REPAIR (MTTR)

   MTTR is the average time it takes to track the errors causing the failure and to fix these errors which causes failures.

c) MEAN TIME BETWEEN FAILURE (MTBF)

   MTBF is the combination of the MTTF and MTTR metrics.

d) MTBF=MTTF+MTTR

e) RATE OF OCCURRENCE OF FAILURE (ROCOF)

   ROCOF is the number of the failures occurring in unit time interval. ROCOF is the frequency of the occurrence with which the expected behavior occurs.

f) PROBABILITY OF FAILURE ON DEMAND (POFOD)

   POFOD is defined as the probability that the system will fail when a service is requested. It is the number of system failures given a number of system inputs.

3) Maintainability:

   There are three types of maintenance:

   Corrective Maintenance

   Adaptive Maintenance

   Perfective Maintenance

Factors affecting the maintainability are:

1) Modularization

Modularization allows one [38] to decompose a system into functional units to develop independently useful subsystems.

2) **Programming Language**

   Programs written in high level programming language are usually easier to understand than low level programming language. If program is easy to understand then maintenance will be easy.

3) **Level of validation**

   Coding errors are relatively cheap to correct. Design errors are much more expensive as they may involve rewriting of one or more program units. Errors in requirement specification are most expensive to correct because of redesign which is involved,

4) **Complexity**

   Complexity of the software affects the maintainability. More complex the software, more difficulty in maintaining codes.

5) **Traceability**

   Traceability refers to the ability to trace design representation or actual program components back to the requirements.

Metrics of maintenance:

1) Lines of code(LOC)
2) Number of commented lines
3) Average number of lines per module.

Formula for calculating the maintenance of the software architecture

$$\textbf{Maintainability Effort} = \sum_{IA}\Big(\big(\sum_{CC_i} s_j\big) \cdot P_{cc} + \big(\sum_{NP_i} s_j\big) \cdot P_p + \big(\sum_{NC_i} s_j\big) \cdot P_{nc}\Big)$$

The result of the impact analysis process is a set of impact analyses, one for each change scenario, i.e. $IA = \{ia_1 \ldots, ia_u\}$, where ia_1 is defined as $IA_i = \{CC_i\, NP_i\, NC_i, R_i\}$. Here $_i$ is the, possibly empty set including a size estimate for the required change in lines of code, $CC_i = \{\{c_j, s_j\}, \ldots\}$. NP_i and NC_i contain similar information for new plug-ins and new components, respectively. R_i contains the new, changed and removed relations required for incorporating the change scenario.

Selection Criteria	Alternative 1	Alternative 2
Adaptability	+1	+2
Maintainability	+1	+2
Robustness	+1	+2
Scalability	0	+1
Portability	0	0
Performance	+1	+2
Interoperability	0	0
Net Score	4	9
Rank	2	1
Summary	Alternative1 that is existing software architecture is less robust than the enhanced architecture. The existing architecture is less adaptable than enhanced architecture. The existing architecture is less maintainable than enhanced architecture	The enhanced software architecture has more ability to cope with the errors. The enhanced architecture is more adaptable than existing architecture. The enhanced architecture has more ability to find defects and correcting those defects.

Table 4.1 Trade off Analysis for Alternative Paths

Table 4.1 shows a net score for each alternative path by subtracting the '_' ratings from the '+' ratings. At this stage, some alternatives are removed to simplify the trade-off analysis. The remaining options are due for more detailed analyses. A weighted scoring scheme is used to support the detailed analyses. The weights are assigned to each criteria and rates each criteria design alternative. Then ranking is done by calculating the total score for each alternative.

Comparison Basis	Existing architecture	Enhanced architecture
Comparing the architectures on the basis of quality attributes	The existing software architecture is less robust than the enhanced architecture. The existing architecture is less adaptable than enhanced architecture. The existing architecture is less maintainable than enhanced architecture	The enhanced software architecture has more ability to cope with the errors. The enhanced architecture is more adaptable than existing architecture. The enhanced architecture has more ability to find defects and correcting those defects.
Comparing the architectures on the basis of functionality.	The existing architecture has less features than enhanced architecture.	The enhanced architecture has additional features.
Comparing the architecture on the basis of the performance	The existing architecture takes more time to retrieve the data	The enhanced architecture takes less time to retrieve the data.
Comparing the architecture on the basis of the classes.	The existing architecture has not well defined classes and functions.	The enhanced architecture has well defined classes and functions.

Table 4.2 Comparison between existing and enhanced architecture

% of Maintainability	Existing architecture	Enhanced Architecture
	10	40

Table 4.3 Comparison of Maintainability

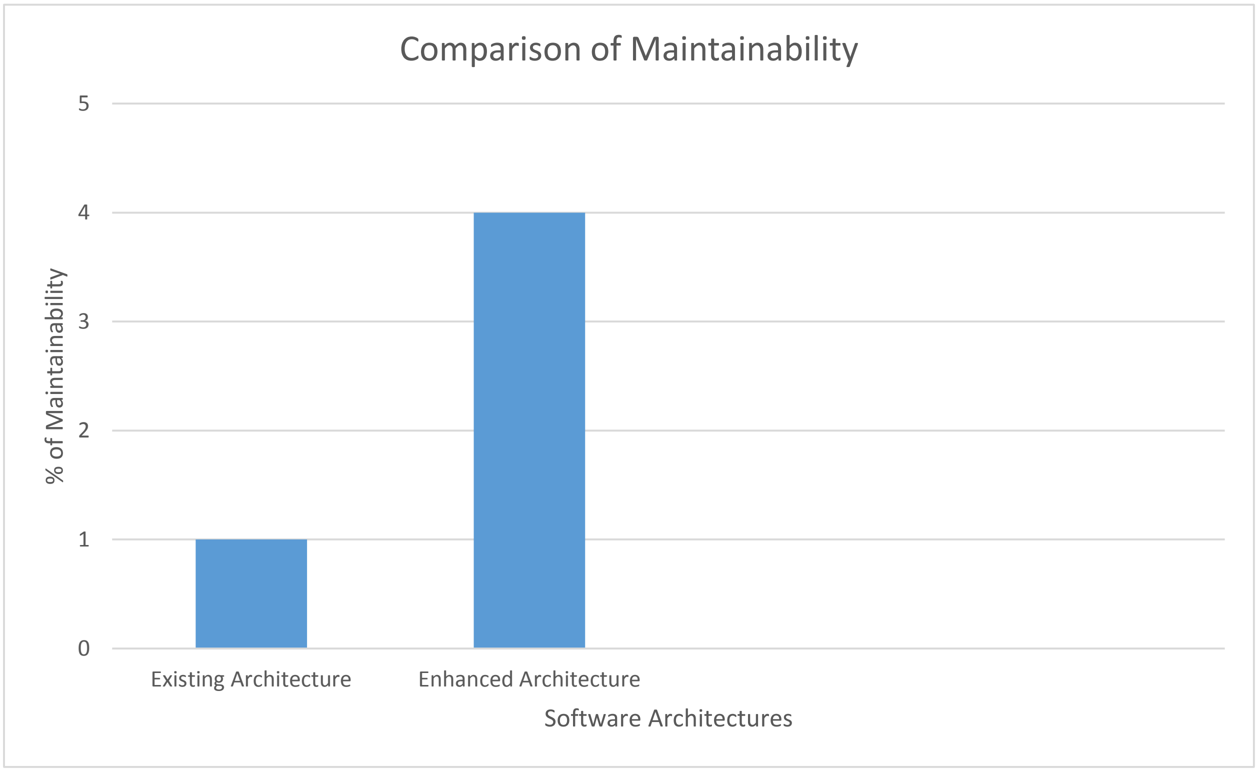


Fig 4.1 Architecture Comparison on the basis of Maintainability

In fig 4.1 the architectures are compared on the basis of maintainability. The x-axis represents % of Maintainability and the y-axis represents the architectures. Maintainability is the ease with which the software can be maintained in order to find the defects and the cause of the defect and then correcting those defects their causes. The fig 4.1 clearly shows that existing software architecture is 10% maintainable and the enhanced architecture is 40% maintainable. The enhanced software architecture has more ability to find the defects and correcting those defects and more changes can be accommodated in enhanced architecture.

% of Robustness	Existing Architecture	Enhanced Architecture
	20	50

Table 4.2 Comparison of robustness

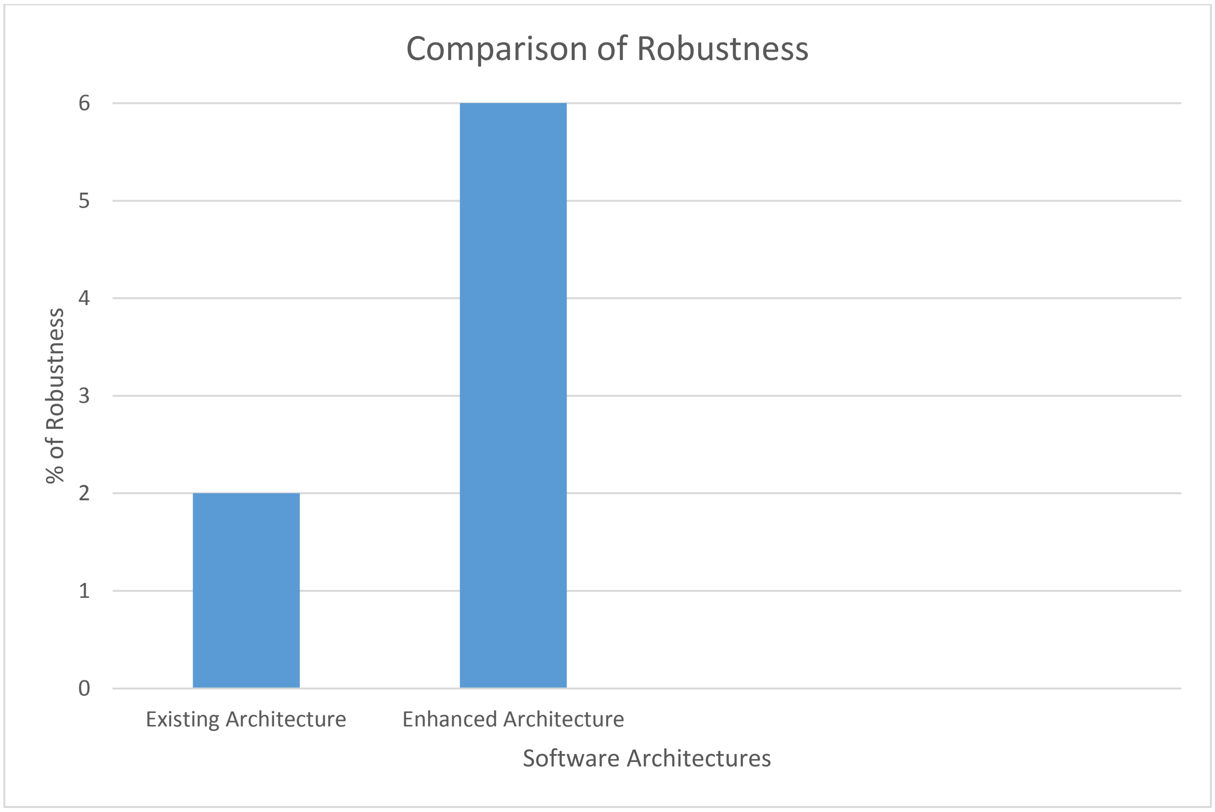


Fig 4.2 Architecture Comparison on the basis of Robustness

In fig 4.2 the architectures are compared on the basis of robustness. The x-axis represents the % of Robustness and the y-axis represents the architectures. The blue column represents the existing architecture and the green column represents the enhanced architecture. Robustness of the software is the ability of the software to deal with the errors and bugs which occurs during the execution of the software. Robustness is the ability of the software to perform its function despite of wrong inputs. The fig 4.2 clearly shows that existing software architecture is has 20% of robustness and the existing system has 50% of robustness. The existing system is less robust than the enhanced architecture. The enhanced software architecture has more ability to cope with the errors.

% of Adaptability	Existing Architecture	Enhanced Architecture
	20	40

Table 4.3 Comparison of Adaptability

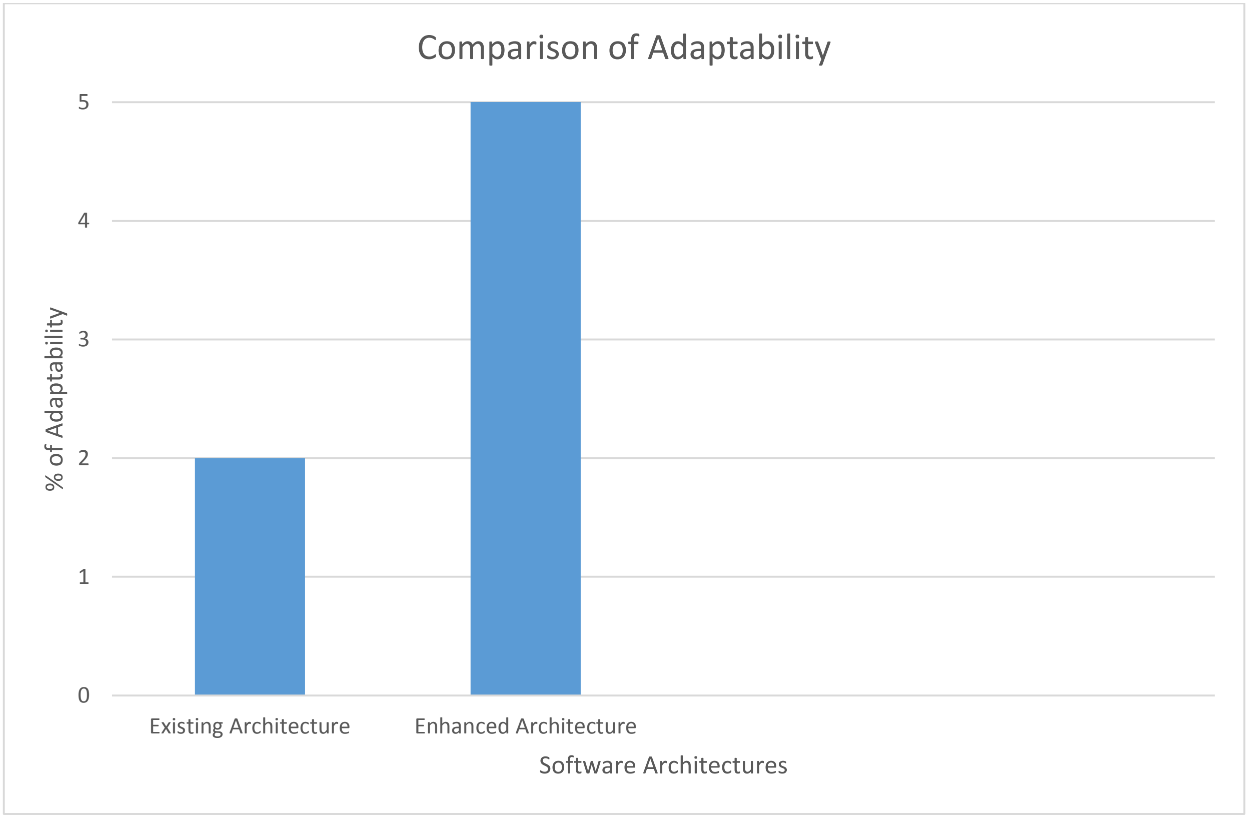


Fig 4.3 Architecture Comparison on the basis of Adaptability

In fig 4.3 the architectures are compared on the basis of adaptability. The x-axis represents the % of Adaptability and the y-axis represents the architectures. The blue column represents the existing architecture and the green column represents the enhanced architecture. Adaptability is the ability of the software architecture to change to fit to the occurring changes. Adaptability of the software is to adapt itself according to the changed conditions. Adaptive software is the open software that is able to change its behavior in response to the changed circumstances. The fig 4.3 clearly shows that existing software architecture is 20% adaptable and the enhanced architecture is 40% adaptable.

% Improvement	Existing Architecture	Enhanced Architecture	Difference
Maintainability	10	40	30
Robustness	20	50	30
Adaptability	20	40	20

Table 4.4 Overall Comparison

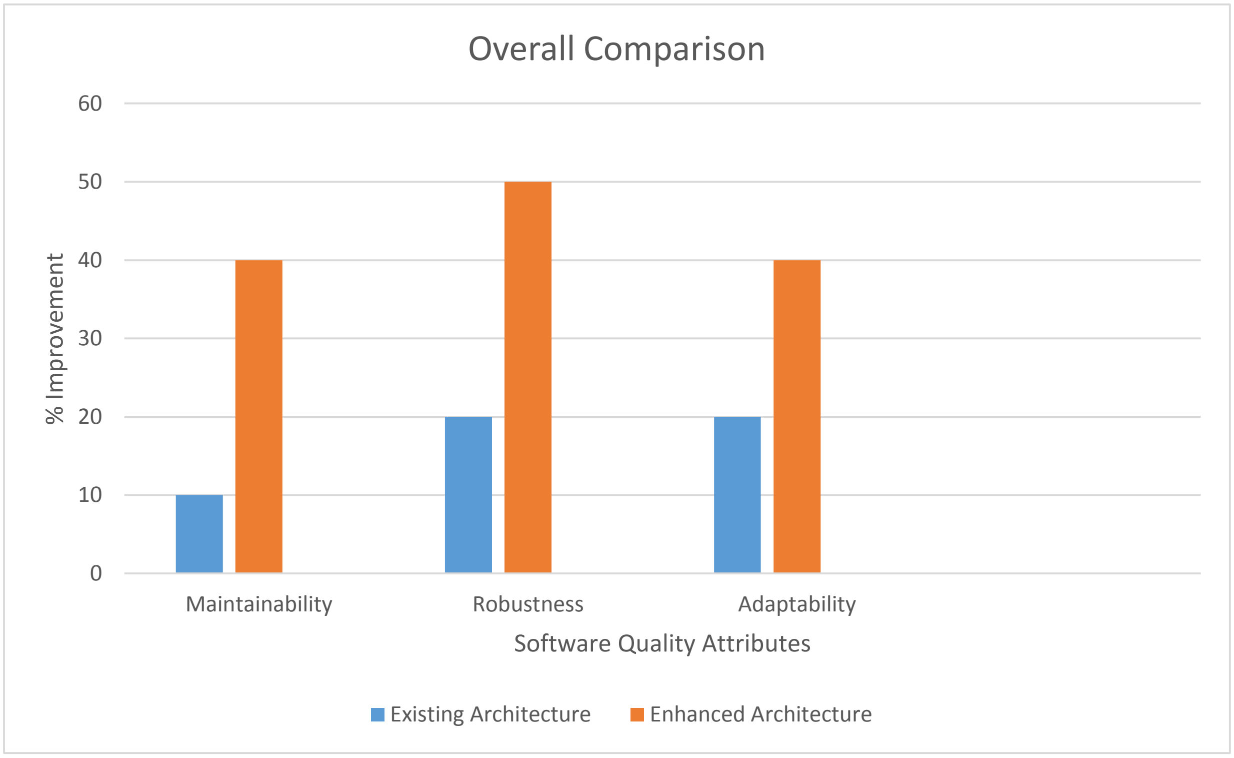


Fig 4.4 Overall comparison between existing architecture and enhanced architecture.

Fig 4.4 shows the comparison between the maintainability, robustness and adaptability of the Existing architecture and the enhanced architecture. The Fig 4.4 shows:

1) Enhanced system is 30% more maintainable than the enhanced architecture.
2) Enhanced architecture is 30% more robust than the existing architecture.
3) Enhanced architecture is 20% more adaptable than the existing architecture.

CHAPTER-5

CONCLUSION AND FUTURE WORK

CONCLUSION

In this proposal we have described the reasons for supporting the software architecture evolution. We identify alternative evolution paths. We describe the properties of the evolution paths and analyzing those properties. We specify and use the path constraints which are used to identify which evolution paths are used in evolution style. Then choosing the best path among alternative paths to achieve the business objective of the organization and to perform the tradeoff analysis between the alternative paths and choose best paths satisfying the requirements and the constraints.

FUTURE SCOPE

Software evolution is [21] a young field but its concepts changes continuously. There are many open questions which are still to be explored and researches must be carries out to accomplish these researches such as how to model the evolution in the complex environment in which the system is deployed.one question is modelling of tradeoff risks and problems which come across in building the model.

CHAPTER-6

REFERENCES

[1] L. Bass, P. Clements and R. Kazman, Software achitecture in Practice, Pearson Edition Inc., 2003.

[2] Taylor, Richard N., Nenad Medvidovic, and Eric M. Dashofy, *Software architecture: foundations, theory, and practice*, Wiley Publishing, 2009.

[3] R. N. Taylor and E. M. Dashofy, "Institute of software research," [Online]. Available: http://www.isr.uci.edu/classes/ics123s02/.

[4] P. Kruchten, "Architectural blueprints-The "4+1" view model of software architecture," pp. 42-50, Nov 1995.

[5] J. M. Barnes, D. Garlan and B. Schmerl, "Evolution styles: foundations and models for software architecture evolution," *Springer-Verlag Berlin Heidelberg,* vol. 13, no. 10, pp. 649-678, 2012.

[6] "AEvol-Architecture Evolution," 24 9 2009. [Online]. Available: http://www.cs.cmu.edu/~able/research/aevol.html. [Accessed 22 11 2014].

[7] I. Sommerville, Software engineering 6th edition, 2000.

[8] Breivold, Hongyu Pei, Ivica Crnkovic, and Magnus Larsson, "Software architecture evolution through evolvability analysis," *Journal of Systems and Software,* 85.11 (2012): 2574-2592.

[9] M. OUSSALAHA, N. SADOU and D. TAMZALIT, "A Generic Model For Managing Software Architecture Evolution," [Online]. Available: http://www.wseas.us/e-library/confrences/2005athens/cscc/papers/497-893.pdf.

[10] A. Amirat, A. Menasria and N. Gasmallah, "Evolution Framework for Software Architecture Using Graph Transformation Approach".

[11] Ghodsi, Ali, et al, "Intelligent design enables architectural evolution,"*Proceedings of the 10th ACM Workshop on Hot Topics in Networks* ACM, 2011.

[12] B. R. Maxim, *Software Evolution Planning*, Univ of Mich-Dearborn,Computer & Info Science.

[13] Morrison, Ronald, et al, "Support for evolving software architectures in the ArchWare ADL," *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*. IEEE, 2004.

[14] Kazman, Rick, et al, "The architecture tradeoff analysis method." *Engineering of Complex Computer Systems, 1998, ICECCS'98 Proceedings, Fourth IEEE International Conference on* IEEE, 1998.

[15] Goltz, Ursula, et al," Design for future: managed software evolution," *Computer Science-Research and Development* (2014): 1-11.

[16] V. E. Silva Souza, A. Lapouchnian, K. Angelopoulos and J. Mylopoulos, "Requirement-driven software evolution," *Springer,* 2012.

[17] B. Dougherty, J. White and D. C. Schmidt, "Automated Hardware And Software Evolution Analysis for distributed real time and embedded systems," *Springer,* vol. 1, no. 10, pp. 36-57, 2011.

[18] G. Ghezzi and H. C. Gall, "A framework for semi automated software evolution analysis composition," *Springer,* vol. 20, pp. 463-496, 2013.

[19] N. Medvidovic and V. Jakobac, "Using software evolution to focus architectural recovery," *Springer,* vol. 13, pp. 225-256, 2006.

[20] Godfrey, Michael W, and Daniel M. German, "The past, present, and future of software evolution." *Frontiers of Software Maintenance, 2008. FoSM 2008,*IEEE, 2008.

[21] L. ZHU, A. AURUM, I. GORTON and R. JEFFERY, "Tradeoff and sensivity analysis in software architecture evalution using analytic hierarchy process," *Springer,* vol. 13, pp. 357-375, 2005.

[22] C. H. Lung, S. Bot and K. Kalaichelvan, "An Approach to Software Architecture Analysis for Evolution and Reusability," pp. 144-154, 1997.

[23] M. Risi , G. Scanniello and Genoveffa , "Using fold in and fold-out in architeture recovery of software system," pp. 307-330, 2011.

[24] A. Bertolino and P. Inverardi , "Software architecture based analysis and testing-a look into achievements and future challenges," *Springer,* pp. 633-648, 2013.

[25] A. Martens , H. Koziolek and S. Becker, "Automatically Improve software architecture models for performance,reliability and cost using Evolutionary Algorithm," *WOSP/SIPEW*, 2010.

[26] Wang, Jun, and WeiRu Chen, "A Reliability-oriented Evolution Method of Software Architecture Based on Contribution Degree of Component," *Journal of Software* 7,8 (2012): 1744-1750.

[27] Franco, Joao M., Raul Barbosa, and Mário Zenha-Rela, "Reliability analysis of software architecture evolution," *Dependable Computing (LADC), 2013 Sixth Latin-American Symposium on* IEEE, 2013.

[28] Wermelinger, Michel, et al, "Assessing architectural evolution: a case study."*Empirical Software Engineering* 16,5 (2011): 623-666.

[29] Simetti, Enrico, et al, "A new software architecture for developing and testing algorithms for space exploration missions," *Intelligent Service Robotics* 4.2 (2011): 135-146.

[30] Barnes, Jeffrey M., and David Garlan, "Challenges in developing a software architecture evolution tool as a plug-in," *Developing Tools as Plug-ins (TOPI), 2013 3rd International Workshop on* IEEE, 2013.

[31] Guan, Hui, et al, "Estimation of reliability and cost relationship for architecture-based software," *International Journal of Automation and Computing* 7.4 (2010): 603-610.

[32] Hesse, Wolfgang, "Ontologies in the Software Engineering Process," *EAI*. 2005.

[33] "Computer Science and Electrical Engineering," [Online]. Available: http://www.csee.umbc.edu/courses/undergraduate/341/fall08/Lectures/Eclipse/intro-to-eclipse.pdf.

[34] "No Magic Inc," [Online]. Available: http://www.nomagic.com/files/manuals/MagicDraw%20Integrations%20UserGuide.pdf.

[35] "Magic Draw," [Online]. Available: http://en.m.wikipedia.org/wiki/MagicDraw.

[36] [Online].Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.4041&rep=rep1&type=pdf.

[37] [Online]. Available: http://www.ijiset.com/v1s3/IJISET_V1_I3_24.pdf.

[38] [Online]. Available: http://umijms.um.edu.my/filebank/published_arcticle/1695/19.pdf.