Value-Based Software Architecture Knowledge Management

Developed by:

Nida Ahmad
Reg # 132-FAS/MSSE/F06

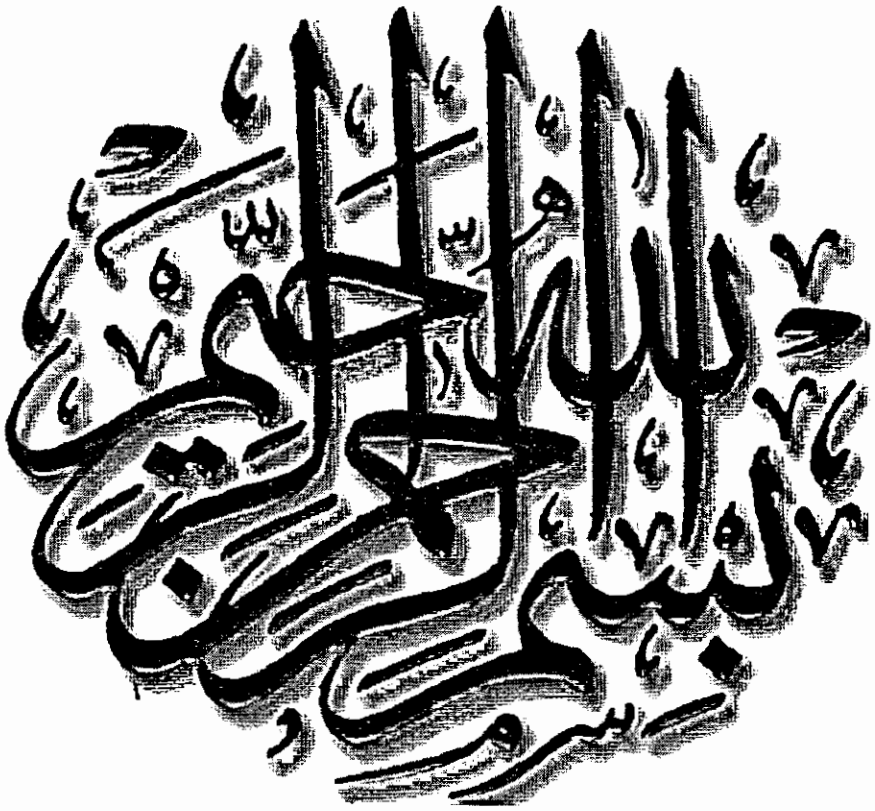
Supervised by:

Dr. Naveed Ikram

Co. Supervised by:

Mr.Muhammad Usman

Department of Computer Science
Faculty of Basic and Applied Sciences
International Islamic University, Islamabad
(2009)



IN THE NAME OF ALLAH
THE MOST BENEFICENT
THE MOST MERCIFUL

International Islamic University, Islamabad
Faculty of Basic and Applied Sciences
Department of Computer Science

Dated: 28-03-2009

FINAL APPROVAL

It is certified that we have read the thesis submitted by Nida Ahmad Reg No. 132-FAS/MSSE/F06. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad for the degree of Master of Science in Software Engineering.

Committee

External Examiner
Dr. Arshad Ali Shahid
Associate Professor,
Chairman, Department of Computer Science,
FAST-NU, Islamabad.

Internal Examiner
Mr. Usman Nasir
Lecturar,
Department of Computer Science,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad.

Supervisor
Dr. Naveed Ikram
Associate Professor,
Department of Computer Science,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad.

Co-Supervisor
Mr. Muhammad Usman
Lecturar,
Department of Computer Science,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad.

Dedicated

To my Dear Parents
Who are an embodiment of diligence and honest,
Without their prayers and support
This dream could have never come true.

Also to my beautiful,
Extraordinary Daughter Hiba,
Who makes everything worthwhile,
And to whom I love the most.

A Dissertation submitted as

Partial Fulfillment of Requirements

For the degree of Master of Science in

Software Engineering

DECLARATION

I hereby declare and affirm that this thesis neither as a whole nor as part thereof has been copied out from any source. It is further declared that I have completed this thesis and accompanied software application on the basis of my personal efforts, made under the sincere guidance of my supervisors. Where as necessary references and acknowledgment has been made. If any part of this report is proven to be copied out or found to be a reproduction of some other, I will stand by the consequences. No portion of the work presented in this report has been submitted in support of an application for other degree or qualification of this or any other University or Institute of learning.

Nida Ahmad
132-FAS/MSSE/F06

ACKNOWLEDGEMENT

In times of acknowledgement all too often we forget the one who is behind it all Allah. The author deepest gratitude and appreciation goes to Almighty Allah who gave her power and wisdom to work and plan with full devotion.

The author expresses her most sincere appreciations to Mr.Muhammad Usman Co.Supervisor/Lecturar (IIUI), for his invaluable guidance and endless support during the research. Thanks are due to Mr.Muhammad Usman for suggesting the topic. He gave her full support whenever she needed and showed his personal interest in this research. Mr.Usman's supervision and suggestions at all stages of the work made this report possible.

With a debt of gratitude, this cannot be adequately expressed in words, the author thanks to Dr. Naveed Ikram Supervisor/Associate Professor (IIUI), for his advice, guidance and encouragement. His practical and sharp vision in research has been invaluable for author's work on this thesis. Dr. Naveed's supervision and insightful comments at all stages of the work made this report possible.

The author wishes to extend her deep appreciation towards her family members especially her parents and parents-in law who did their best to make the world a better place for the author. For their inspiration, love and endurance, the author thanks them and expresses her everlasting love and gratitude.

Lastly, the author is deeply indebted to her husband, Salman Alam, who gave her an unconditional support and love through all this long process. It would not have been possible to write this thesis without his support and wonderful advice. Thanks for the encouragement, patience, support and for always believing in me. Thank you for being my best friend and a great husband.

THESIS IN BRIEF

Thesis Title: Value-Based Software Architecture Knowledge Management

Organization: International Islamic University, Islamabad, Pakistan.

Objective: The objective of this research work is to improve efforts of tool support for managing, sharing and storing architectural knowledge.

Undertaken By: Nida Ahmad

Supervised By: Dr. Naveed Ikram (Associate Professor)
Mr. Muhammad Usman (Lecturer)
Department of Computer Science,
Faculty of Basic & Applied Sciences,
International Islamic University, Islamabad.

Started On: February 2008

Completed On: March 2009

Research Area: Managing, Sharing and Storing Architectural Knowledge.

Tools: ADDSS,PAKME,AREL-Tool-Set,Archium,Knowledge Word Plug-in,MySQL,Enterprise Architect..

ABSTRACT

Proper management of architectural knowledge (AK) is essential in order to reduce high evolution and maintenance costs and to avoid architectural erosion. Architecture Knowledge is an important piece during the architecting process that must be explicitly documented. Researchers have proposed different tools and techniques for managing, sharing and storing architectural knowledge, but practitioners are reluctant in applying such tools and techniques because of certain inhibitors such as extra time and effort required, unclear benefits for documenting AK etc. To deal with such inhibitors, there is a need to manage architectural knowledge in a value based manner. This thesis describes a Value-Based approach for managing architectural knowledge. Value-Based approach takes into account value consideration of stakeholders and only documents the information required by stakeholders.In this thesis, a web-based tool is described which is able to manage, share and store AK in a value-based manner. The main work is the application of Value-Based Software Engineering principles on an existing tool. Moreover, it also describes some other features implemented to an existing tool which are missing from that tool found during the survey.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

In current research period, more and more things rely on software-intensive systems. The number of systems that become software-intensive is ever increasing and so are the demands that are put on them. This means that the quality of these systems becomes increasingly dependent on the quality of the software. Software architectures are a vessel which can be used to reason about the expected quality of a software system. They can be used by engineers to predict the consequences of design decisions for an envisioned or existing software system before such design decisions are actually implemented [1]. This means that engineers can use architectural analysis as a means to investigate which kind of system would best fit their needs without having to implement the various candidate systems first.

Software architecture provides a high-level abstraction of a system. It is an important area of research in recent years because it lays the structural foundation of a system. In this thesis the basic discussion is about one of the discipline of software architecture i.e. Architectural Knowledge. This chapter includes research area, problems in that area, research problems, research questions, thesis contributions etc.

1.1 Software Architecture

Software architecture plays an important role in developing high quality software intensive system. It represents the design for describing the main parts of a software system. Traditionally, "*software architectures have been considered as a set of interrelated components and connectors*" [1]. This means that the functionality of a software system is mainly described by means of a set of interrelated components and connectors [2],[3]. Software architecture has become the principal means by which requirements are transformed into a working, implemented system. Recently, research [4] shows that the software architecture plays a significant role in organizing the complex interactions as well as dependencies between stakeholders. Moreover, software architecture also provides an essential artifact that can be used for reference.

The Rational Unified Process® (RUP)[5] defines software architecture as "*the set of significant decisions about the organization of a software system: selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements, composition of these structural and behavioral elements into larger subsystem, architectural style that guides this organization*". Software architecture also involves usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs, and aesthetic concerns [3].

The Perry/Wolf model of software architecture defines software architecture as "*elements, form, and rationale*" [6]. This model described the rationale and principles that guide the design and evolution of software architectures. This rationale is considered when adopting an architecture centric approach. However, the main goal for representing architectural design decisions is to bridge the gap between software requirements and architectural products.

1.1.1 Architectural Knowledge

Nowadays, the research trends in software architecture focus on the treatment of architectural design decisions as first-class entities and their explicit representation in architectural documentation [4]. From this point of view, software architecture is no longer perceived as interacting components and connectors only, but also as a set of architectural decisions that convey the architectural principles underlying a particular design [4]. Software architecture is not only just a diagram but also contains architectural design decisions, their rationales and alternatives etc.

"*Architectural Knowledge (AK) is defined as the integrated representation of the software architecture of a software-intensive system (or a family of systems), the architectural design decisions and their rationale, and the influences of the external context /environment*" [7].

According to Kructhen[8] , AK = "*Design + Design decisions + Assumptions + Context*"

1.14. Motivation[46]
1.15. Cause [46]
1.16. Context [27,28,29]
1.17. Notes [27,28,29]
1.18. Date/Versions of decision made[8,9]
1.19. Obsolete decision [8,9]
1.20. Consequences [17]
1.21. Validity of decision [8]
1.22. Related decisions[27,28,29]
2. Design Decision rationales [21]
2.1. Reasons behind design decisions
2.2. Justification for it
2.3. Other alternatives considered
2.4. Tradeoffs evaluated
2.5. Argumentation that led to the decision
2.6. Past rationales
3. Architectural rules [8]
4. Design pattern [23]
5. Architectural views [8]
6. Architecture diagrams/images/ figures [8]
7. Architectural pattern [23]
8. Architectural tactics [23]
9. Architectural styles [8]
10. Scenarios [23]
10.1. General Scenarios
10.2. Concrete Scenarios
11. Quality Attribute [23]
12. Requirements [8,9,23,27]
13. Analysis models [23]
14. Issues [27,28,36]
15. Concerns[36]

16. Choice[36]
17. Reference architecture [8]
18. Design options [23]
19. Design History [8]
20. Tradeoff made [23,36]
21. Architecture variation points[23]
22. Domain analysis [23]
23. Assumptions any other[27]
24. TraceLinks[46]
25. Relationships [23]
26. Ranking [23]
27. Risks/Non Risks [23]
28. Architecture environment[17]
29. Architecture description[8]
30. Stakeholder/Decision Maker Name [8]
31. Stakeholder role and responsibilities [8]

1.2 Architectural Knowledge Management and its importance

The subject of architectural knowledge is complex and covers many issues, both general and domain-specific. Managing and Sharing architectural knowledge is very important issue these days and main area of my research. The software architecture community has recently gained an increasing interest in managing, sharing and storing architectural knowledge [15]. Management of architectural knowledge is clearly related to management of knowledge in general. Architectural knowledge management is defined as

"Software architecture knowledge management is an approach to improving software architecture process outcomes by introducing practices for identifying and capturing architecture knowledge and expertise, and making it available for reuse across projects".[15]

Architectural Knowledge consists of architectural design as well as design decisions, their assumptions and context. It supports the development of the architectural design. During this development, issues arise. These issues lead to architectural choices between two or more alternatives. Based on rationale, engineers make a decision for an architectural choice. Making these decisions often involves making trade-offs between quality attributes. However, the design decisions and their underlying rationales are usually ignored at architectural level and during the development life cycle. Existing notational and documentation approaches to software architecture typically focus on the components and connectors and provides less focus on documenting the design decisions and the rationale underlying the design decisions. Architectural knowledge that is not shared eventually dissipates, as people tend to forget it. So my **problem area** is how to manage, share and store Architectural Knowledge.

The need for documenting, sharing and managing design decisions has been recognized in recent workshops especially in workshops and conferences **[4][7][16][52]**. The truth that design decisions are rarely recorded complicates architecture reconstruction. This difficulty to recreate lost or non documented decisions is one of the main reasons to record them. Hence, documenting AK enables not only to track the overall architecture along the construction process, but also to support future maintenance and evolution activities.

1.3 Problems

Problems arise if we don't manage architectural knowledge properly are:

1. **High evolution and maintenance costs [7]:**
   During the evolution of any software system, architecture erosion may cause high maintenance and evolution costs because the decisions made in the past were not documented, and this architectural knowledge is vaporized.
2. **Poor stakeholder communication [7] :**
   The stakeholders come from different backgrounds and have different concerns that the architecture document must address. If the architectural decisions are not documented and shared among the stakeholders, it is difficult to perform tradeoffs, resolve conflicts, and set common goals, as the reasons behind the architecture are not clear to everyone [17].

3. **Limited reusability of architectural assets [7]:**

It is difficult to perform architectural reuse when the architectural decisions and their underlying rationales are implicitly hidden in the architecture.

4. **Complicates architecture reconstruction[18]:**

The fact that design decisions are never recorded complicates architecture reconstruction. This difficulty to recreate lost or non documented decisions is one of the main reasons to record them.

5. **Poor traceability between requirements, architecture and implementation [7]:**

As design decisions are ignored, traceability between requirements, architectural design decisions and architectural solutions are also ignored.

6. **Precludes organizations from growing their architectural capabilities [19]:**

Existing notational and documentation approaches to software architecture typically focus on the components and connectors and provides less focus on documenting the design decisions that resulted in the architecture as well as the organizational, process and business rationale underlying the design decisions [7].

7. **Changes on existing teams:**

Change of team member from the project also causes problem as the new member have to rethink why this decision were made.

8. **Explain the rationale by which the decisions were made:**

If the design decisions are not documented then we cannot explain the rationale by which the decisions were made.

9. **Efficiency of architectural processes becomes low [20]:**

One of the main problems in architecture processes is the lack of capture and access to knowledge underpinning the design decisions and the processes leading to those decisions. This causes the efficiency of architecture process low.

10. **Difficult to identify design errors [21]:**

Architecture knowledge when not managed properly, we are unable to identify design errors which occur during the construction and maintenance phase.

11. **Difficult to track the overall architecture along the construction process [22]:**

By not storing Architecture knowledge, it is difficult to track the overall architecture along the construction process.

12. Decisions quality becomes worst [20]:

Lack of documentation of design decisions causes the design decision's quality worst.

13. Maximize architectural risks and much time consumed:

As architectural knowledge is not properly managed, this maximizes architectural risks. It takes so much time for understanding any architecture and especially during construction phase.

1.4 Sub-discipline in Architectural knowledge: Tools and techniques

Current research shows that architectural knowledge has brought along some promising research directions. One of which is tools and techniques. Nowadays, there has been an increased demand for suitable techniques and tools that support organizations in capturing, sharing and managing architecture design decisions [4][7].

Proper methodological and tool support for managing AK helps to solve the above mentioned problems. The complex role of architectural decisions requires systematic and partially automated approaches that can explicitly document. As design decisions and their rationale were not rigorously documented. One of the main reasons for this was lack of suitable methodological and tool support. Due to lack of methodological support, some effort has been spent now on developing techniques and tools for effectively managing knowledge pertaining to software architecture [8, 9,17,23,27,28,36] as discussed in chapter 2 and 3. The main focus of this research is on tools and techniques for managing architectural knowledge.

1.5 Inhibitors in managing architectural knowledge

There are different ways for managing architectural knowledge [4,7,52].Researchers and practitioners have proposed various tools [8,17,23,33,35] and techniques [27,28,36,37,38] for its management. Indeed there are certain tools and techniques for architecture knowledge management, practitioners still avoid to do so due to certain inhibitors [27,28,29]. These are:

1. **Critical timing [27,28,29]:**

The period in which design decisions are taken is usually critical for the success of the software project. The project deadline or project pressure is one of the main reasons for not documenting the design decisions and its rationale. Suggesting people for documenting design decisions and their rationale at that critical time is perceived useless.

2. **Extra effort and time required:**

Several architectural knowledge documentation techniques that already exist usually focused on maximizing the consumer (people who are in charge to evolve a system) benefits rather than minimizing the producer (original designers) effort. As a result of this people involved in the documentation and maintenance activities are supposed to spend a lot effort and time for recording and documenting the architecture knowledge.

3. **Overhead [27,28,29]:**

As people who are involved in the documentation and maintenance activities are supposed to spend a lot effort and time for recording and documenting the architectural knowledge. This more effort and time increases the effect of overhead.

4. **Unpredictable information [27,28,29]:**

The architecture knowledge documentation consumer and producer are often different persons. Therefore, it is not clear which Architecture Knowledge information for the project would be relevant for whom.

5. **Unclear benefit [27,28,29]:**

Decision-makers do not know many times for which purpose it is useful to learn the rationale of a design decision. Therefore, some previous training is particularly needed, as the users can perceive the usefulness of documenting the architectural knowledge.

6. **Lack of motivation [27,28,29]:**

People who developed the system for the first time don't act in the role of maintainers. This inhibitor is usually caused by the absence of personal interest because the people who developed the system do not perceive the expected benefits to waste time recording the design decisions.

7. **Owner of the knowledge [27,28,29]:**

The producer of architectural knowledge sometimes doesn't want to transfer or communicate this knowledge to others.

8. **Potential inconsistencies [27,28,29]:**

Architectural knowledge documentation implicitly represents the results of the design. If Architectural Knowledge documentation is not well updated, potential inconsistencies in case of decision changes might occur.

1.6 Value-Based Software Architecture Knowledge Management and its importance

Current research trends in software architecture focus on the proper management of architectural knowledge [4,7,52]. However, due to above mentioned inhibitors; practitioners are reluctant to manage architectural knowledge. A lot of architectural knowledge is there to document and maintain, but the benefit of managing all architectural knowledge is not clear. If benefits for managing AK are not clear, the above-mentioned inhibitors like critical timing, overhead etc will have an impact.It will take more time and effort which also increases the overhead in order to manage AK. From this it is concluded that to get to know the benefit of managing AK, there is a need to manage architectural knowledge in a value based manner which will also mitigate the effects of these mentioned inhibitors [27,28,29].

This idea has been taken from Boehm's work [26], who proposed a Value-Based Software Engineering (VBSE) agenda and from Davide Falessi's work [27,28,29] who used Boehm's idea for documenting design decisions rationale(Value-Based Design Decision Rationale Documentation). Boehm [26] proposed a Value-Based Software Engineering agenda with the objective of integrating value considerations into the full range of existing and emerging software engineering concepts and practices e.g. value-based requirements engineering, architecting & design etc, and of developing an overall framework in which they compatibly reinforce each other.

Basically, Value-Based Software Architecture Knowledge Management is an emerging trend in software architecture community. By managing architecture knowledge in a value based manner is one of the most valuable steps for advancing the software

architecture state of the art by preventing its high costs of change and diminishes the effort and time required [28,29]. This diminished effort has the effect of mitigating the overhead. A Value-Based approach helps to document only the set of required information based on the choice of different stakeholders. With this approach one doesn't need to manage AK which is not required at that time for that stakeholder. Value-Based Software Architecture Knowledge Management basically focuses on the choice of all the stakeholders who are involved in specific project and will have a choice to get only that type of information which they required.

1.7 Research Problem

As the above study shows that there are tools and techniques for managing, sharing and storing architectural knowledge but practitioners are reluctant in applying such tools and techniques because of certain inhibitors such as extra time and effort required, unclear benefits for documenting AK etc. Therefore, we need to introduce an approach which mitigates the effect of above mentioned inhibitors. With this context, the problem statement of this research involves the following questions. Also, in order to establish the usefulness of this research, I aim to investigate the following questions:

1. How to make architecture knowledge management tools and techniques practical?
2. How to reduce time and effort for managing and storing architectural knowledge?
3. How to reduce overhead for managing and storing architectural knowledge?
4. How to make knowledge capture cost-effective?

The above-mentioned research questions will be addressed by Value-Based approach.Three workshops on Sharing and Reusing Architectural Knowledge have been held on 2006 [4] , 2007 [7] and 2008 [51] by Lago, P. & Avgeriou. All of them focused on current approaches, tackling above mentioned problems: methods, languages, notations, tools, techniques to extract, represent, share, use and re-use architectural knowledge. As these workshops held in 2006, 2007 and, 2008, so this area is not an old one and I can do further research in this area as it provides a lot of research directions. The other reason of choosing this area is that, it is necessary to solve the above mentioned inhibitors as it plays an important role in making projects successful and in understanding the benefits of managing AK and if we don't mitigate the effect of above mentioned inhibitors, it causes other problems like lack of resources, management issues, financial

problems etc. Authors like Muhammad Ali Babar [21,23,24,19,20], Remco C. de Boer [22,24] , Rik Farenhorst [24,25], Rafael Capilla [8,9,26] and others have done a lot of work in this area and also address the importance of sharing ,managing, documenting and reusing AK.

1.8 Research Methodology

In order to answer the research questions a qualitative and human-centered approach has been used. The research process consists of three stages:

a) A literature survey and analysis of techniques and tools for managing, sharing and storing Architectural Knowledge.
b) Evaluation of the surveyed tools.
c) Build a working prototype of an architecture knowledge management tool that incorporates value based concepts to address research questions discussed in section 1.7.

Since the above mentioned research questions are inter-related, they need to be examined together in a holistic way. The research approach in this thesis is to first survey and analyzes the existing techniques for managing, sharing and storing architecture knowledge. As a result of this survey, a Value-Based approach has been found for Documenting Design Decisions Rationale, known as Value-Based Design Decision Rationale Documentation (VB-DDRD). This is a useful approach as it documents the set of required information based on its purpose. Existing tools for managing Architectural Knowledge have also been studied and evaluated on the basis of certain attributes. It has been observed that none of the tools supports Value-Based Software Engineering concepts. Therefore, an open source tool is selected that covers additional features relative to other tools. This tool has been used to develop a Value-Based Software Architecture Knowledge Management tool. Certain other limitations have been found during the survey and evaluation of tools. Special features have been added to overcome various limitations as discussed in chapter 3.In this way we are providing a working prototype.

1.9 Survey Outcomes

This section summarizes the results of the literature survey. Seven techniques and five tools have been studied and surveyed for managing, sharing and storing Architectural Knowledge. Different techniques, frameworks, methods, approaches have been investigated for managing AK. From the survey of techniques, a Value-Based Design Decision Rationale Documentation(VB-DDRD) technique [27,28,29] has been found as a useful technique that focuses on documenting the set of required information based on its purpose. Basically in that technique, Value-Based Software Engineering principles have been applied to document Design Decisions Rationale. Moreover, the existing tools have been analyzed and studied. The features of each tool have been compared with others and there are certain features which one tool is covering but not the other one. Tools are evaluated on the basis of certain attributes like usability, open source etc. It has been observed that Architecture Design Decision Support System (ADDSS) is only an open-source tool and it covers additional features relative to other tools. Moreover, by comparing and evaluating tools, certain limitations and drawbacks have been found in all the tools. Limitations like no tool is supporting for Value-Based software engineering principles, no support for software product families etc are found.

The results indicate that there are tools for managing and storing architectural knowledge but none of these tools support Value-Based Software Engineering principles. So for managing architectural knowledge, a Value-Based tool should be implemented. Moreover, certain limitations found during survey must be implemented. Details are mentioned in chapter 2 and 3.

1.10 Thesis Contributions

This thesis enhances the understanding of architectural knowledge. The main focus is on providing some methodological support for managing, sharing and storing architecture knowledge. There are two major contributions of this thesis: Survey of tools and techniques and modification of a tool to make it value-based. This work is implemented on an existing tool Architecture Design Decision Support System (ADDSS) [8,9] which is an open source tool and covers more features relative to other tools. Basically certain

features found from the literature survey have been incorporating into the tool which is missing from the tool.

The thesis makes the following specific contributions:

1. A literature survey and analysis of tools and techniques and the evaluation of tools on the basis of certain attributes. This is one of the major contributions of this thesis.
2. Application of Value-Based Software Engineering principles into ADDSS tool. A Value-Based Software Architecture Knowledge Management (VB-SAKM) process has been proposed which focuses on documenting only the set of required information based on choice of different stakeholders. This concept has been taken from Davide Falessi's work [27, 28, 29] who proposes a Value-Based approach to DDRD (VB DDRD) based on Boehm work [26] studied in the literature survey of techniques; this thesis proposes a Value-Based approach to ADDSS. Modified tool provides the opportunity to all the stakeholders to choose the required design decisions information by giving score to each attribute of design decisions. Architect can mark these attributes as required, useful or optional on the basis of the score provided by each stakeholder. This is the other major contribution of this work as this feature is not present by any of the studied tool and due to this feature; the modified tool is Value-Based and named as Value-Based Software Architecture Knowledge Management Tool.
3. Provides a set of templates to document architecture and design tactics as an artifact of architecture knowledge. Also provides a catalogue of architecture and design tactics. Records the functional requirements and non functional requirements according to their type.
4. Captures scenarios (general or concrete), which can be elicited from a stakeholder or extracted from a pattern. Also captures principles and artifacts for some particular architecture.
5. Captures architecture patterns and supports multiple views with each single architecture. Also shows the categorization of risks and non risks associated with each design decisions.
6. Evaluation of modified tool i.e. Value-Based Software Architecture Knowledge Management (VB-SAKM) tool on the basis of certain attributes.

1.11 Thesis outline

The rest of this thesis is organized as follows:

Chapter 2 explores the related work of techniques for managing, sharing and storing architecture knowledge published in the literature .In this chapter, the techniques/ method/ concept/ framework are studied and surveyed along with certain limitations.

Chapter 3 explores the related work of tools for managing, sharing and storing architecture knowledge published in the literature. In this chapter, these tools are also evaluating on the basis of certain attributes. A comparison is made to highlight the merits and limitations of these tools.

Chapter 4 describe the main contribution i.e. implementing the work in some existing tool to make that tool more mature for managing, sharing and storing Architecture Knowledge. In this chapter, the features which are implemented in the tool have been described. Details of each feature along with the figures are described in this chapter.

Chapter 5 concludes this thesis by outlining the major contributions and benefits of this work. It also discusses the future work in this area.

CHAPTER 2

LITERATURE SURVEY AND ANALYSIS OF TECHNIQUES

2. LITERATURE SURVEY AND ANALYSIS OF TECHNIQUES

There are different ways to manage and store architectural knowledge. Current research shows the importance of documenting and managing design decisions along with their rationales. Recently, researchers [27,28,37,38] have proposed various ways to capture architecture knowledge. There are different techniques for managing, sharing and storing architectural knowledge. Different terms are used for techniques as framework, method, approach or concept. This thesis is using a general term "techniques" for all above mention terms.

This chapter provides a brief description about the work done in managing, sharing and storing architectural knowledge as techniques alongwith their main limitations. The surveyed techniques, framework, method or approaches are discussed below:

2.1 Review of Existing Techniques / Frameworks / Method /Approach

Seven techniques have been studied and reviewed in this thesis which is as follows:

2.1.1 Decision Goal and Alternatives DDR Framework (DGA-DDR)

Main Feature: For documenting design decision rationales.

2.1.1.1 DESCRIPTION

Falessi et al.[27,52] have proposed a framework that focuses on the reasons why design decisions have been taken. The framework contains a specific design decision rationale documentation technique called DGA DDR, which is driven by the decision goals and design alternatives available. In DGA DDR technique, the rationale behind a design decision documents the attributes of CBAM. According to DGA, no matter what the software context might be, design decisions depend on basic decision goals and inter-decision relationships. These decision goals include Functional requirements, Non-

functional requirements (quality attributes and constraints), Business goals, Decision relationships.

The framework [27] aims not only to document decisions previously taken, but also to support decision makers in taking these decisions. According to DGA, DDR documentation consists of two stages i.e. it consists of two main activities, which aim to: (i) *understand what* to document and (ii) *enact* the documentation, respectively. In the first activity the project objectives and constraints are defined and it is investigated which decision relationships are appropriate for the project. In the second activity the knowledge is further refined and described in phases and tables. However, this framework does not take into account the influence of certain factors like granularity, hierarchies, architectural pattern, and architectural styles.

2.1.1.2 LIMITATIONS

Certain limitations have been found by studying and analyzing the research papers and technical reports [27,52] as discussed below:

1) Decision Goal and Alternatives DDR Framework doesn't provide any type of tool support.
2) Value-Based Software Engineering (VBSE) [26] principles play an important role while capturing architecture knowledge. It doesn't support VBSE principles. This technique doesn't documents only the required set of information based on a priori understanding of who will benefit later on, from what set of information, and in which amount.
3) Software product families [43] [44] are recognized as an effective approach to reuse in software development. Architectural knowledge plays an important role for the architectures in software product families. This technique/framework is not useful for software product families as it has been checked from the research papers and other documentation.

2.1.2 COVAMOF Framework [Variability Modeling Principles to Capture Architectural Knowledge]

Main Feature: For capturing architectural knowledge.

2.1.2.1 DESCRIPTION

COVAMOF [30] is a framework which is useful for managing variability.COVAMOF framework solves the issues that arise when the practitioners are relating quality attributes to architectural design decisions. Basically, the effects of architectural decisions on the functional and quality aspects of a system are often hard to make explicit. Researchers mentioned that the same issues arise in the field of product families, as arises when configuring products. The COVAMOF variability modeling framework was developed which should be able to deal with the imprecise and incomplete nature of the effect of decisions on quality attributes. The problems with incomplete and imprecise knowledge do not only occur in the context of architectural knowledge, but have also been identified in the field of software product families. COVAMOF consists of models, tools and processes that support engineers in the development of product families as well as the configuration of individual products from a product family.

Researchers [30] argue that the concepts of COVAMOF can be used to capture the architectural knowledge. According to them, this mapping makes it possible to use the COVAMOF tool suite and method for capturing architectural knowledge. It has been shown how the concepts and issues of architectural knowledge map to concepts and issues that are addressed by COVAMOF. By using the approach adopted by COVAMOF for capturing AK, it is possible to explicitly deal with the implications attached to tacit, documented and formalized architectural knowledge, as it does not require a complete and fully formalized model in order to be useful during architecture design.

The idea behind COVAMOF is to enable tool support to manage complexity and reduce the dependency of organizations on experts. The first benefit of COVAMOF is incremental externalization which includes documenting expert knowledge,incorporating existing documentation,collecting reference data,formalizing documented knowledge.Second benefit is to reduce derivation cost which includes reducing expert involvement,providing structured documentation,configuration guidance,automatic inference,automatic consistency checking and automatic QA estimation.However, this approach is time consuming. The actual taking of decisions will slowly move from architecture time, to development time and eventually to runtime.

2.1.2.2 LIMITATIONS

Certain limitations have been found by studying and analyzing the research paper [30] as discussed below:

1) Value-Based Software engineering (VBSE)[26] principles play an important role while capturing architecture knowledge. It doesn't support VBSE principles. This framework doesn't captures only the required set of information based on a priori understanding of who will benefit later on, from what set of information, and in which amount.
2) As it documents all the architectural knowledge in a framework, which is useful for communicating stakeholders but it is difficult to access by all stakeholders. So this framework is not useful for building stakeholder communication.

2.1.3 METHOD [Flag, Filter, and Form]

Main **Feature:** For capturing architectural design decisions.

2.1.3.1 DESCRIPTION

A method [37] has been proposed which aimed to make the process of capturing architectural design decisions easier by dividing it into three steps: Flag, Filter, and Form. This method reduces the amount of work needed to capture design decisions. It allows software designers to casually flag decisions in various documents like e-mail, webpages, books, and pictures, so that a software designer could spend less time creating the list of decisions and more time on manipulating and evaluating captured decisions. Flagging is the capture of candidate decisions "in the raw". Filtering is sifting through the captured decisions to find applicable decisions to a project. The designer would collect a set of decision references when the time is over. Once the designer identifies a decision reference to be valid, the designer promotes the decision to represent a part of the software design, which is represented as a formal decision entity. Forming creates a formal decision entity for the decision and provides descriptive attributes such as priority, category, or state for each decision. The designer generates a first-class representation of the design decision, once a decision has been approved. The formed decision is stored into a decision repository for manipulation, association and analysis.

Researchers mentioned that the method[37] is iterative and each step can span across multiple sessions; Supports decision and design traceability; Provides immediate benefit to those capturing their design decisions by acting as a "memory-aid" service; and finally, applies to other areas of software development, such as coding, testing, and support. A tool is being developed that will assist designers in capturing software architectural design decisions using the three-step method described above. An empirical validation of the method has been planned by evaluating a tool that implements the proposed method.

A main issue revolves around the architectural knowledge itself, as AK can be considered as intellectual property, and hence are highly confidential. Moreover, decisions can also be personal or political, in which legal issues may arise if they were documented. The method may be unintuitive or overly complicated. Complications arise when the source of many design decisions are directly from the designer. The documentation of sources across various mediums is challenging.

2.1.3.2 LIMITATIONS

Certain limitations have been found by studying and analyzing the research paper [37] as discussed below:

1) This method does not being applied industrially.
2) Method doesn't support VBSE principles. This method doesn't documents only the required set of information based on a priori understanding of who will benefit later on, from what set of information, and in which amount instead this is recording all the architectural knowledge information.
3) Architectural knowledge plays an important role for the architectures in software product families. This method is not useful for software product families and it has been checked from the research paper.
4) As the information is flagged and filtered through different sources, sometime if one cannot access the information from any source it might be difficult to communicate with stakeholders. Complications arise when the source of many design decisions are directly from the designer.

2.1.4 Derivational Analogy: An approach for Capturing and Replaying Architectural Knowledge.

Main Feature: For capturing and replaying Architectural Knowledge mainly architectural design decisions.

2.1.4.1 DESCRIPTION

Ibrahim Habli and Tim Kelly [38] tackle the topic of recovering architectural knowledge in existing system architecture. They propose to use derivational analogy to reconstruct the decision-making process and document architectural drivers, decisions, and subsequent analysis. Instead of reusing past solutions directly, derivational analogy replays the process leading to these past solutions. In doing so, particular design steps or routes are skipped if the design assumptions do not hold in the context of the new problem. An approach [38] to define new software architectures through the use of derivational analogy.

The main actions in architecture design replay using derivational analogy can be summarized as follows: 1) Capturing and Representing Architecture Knowledge: Not only does derivational analogy require the recording of the final design decisions, but also the goals, requirements, constraints, preconditions, rationale, assumptions, dependencies , and alternatives of the chosen design. Three steps for capturing and representing architectural knowledge: (1) recording architectural drivers, (2) recording architectural design decisions, and (3) recording the analysis of these decisions in achieving the architectural drivers. In order to capture and analyze the architectural knowledge, three methods have been used, developed by the SEI, namely: Quality Attribute Scenarios, Attribute Driven Design Method (ADD) and Architecture Tradeoff Analysis Method (ATAM). 2) Architecture Derivation through Process Replay: A new architecture is derived by replaying the architectural knowledge captured as described in step 1. The replay of the architectural process entails the retrieval of the design decisions and their rationale. Then, a gap analysis is carried out where only relevant design sequences are reapplied in the context of the new architecture. Adaptation is required when mismatches are encountered. Finally, the new architecture is evaluated against its own specific requirements. Three steps are: (1) Relevance and Retrieval (2) Adaptation (3) Evaluation.

Although it produces an analyzable architecture but this concept is a time-consuming activity.

2.1.4.2 LIMITATIONS

Certain limitations have been found by studying and analyzing the research paper [38] as discussed below:

1) Derivational Analogy approach doesn't provide any tool support. This approach should be applied on any tool.
2) Approach doesn't support VBSE principles. This approach doesn't documents only the required set of information based on a priori understanding of who will benefit later on, from what set of information, and in which amount instead this is recording all the architectural knowledge information.
3) Architectural knowledge plays an important role for the architectures in software product families. This approach is not useful for software product families and it has been checked from the research paper [38].

2.1.5 Extensibility Approach [Exploring Extensibility of Architectural Design Decisions]

Main Feature: For documenting and reusing architectural design decisions.

2.1.5.1 DESCRIPTION

Explores how design decisions can be documented, and how they affect the synthesis architecture. This approach [18] explores extensibility ideas from software product lines to show how synthesis architectures can be extended on the basis of design decisions. Researchers [18] introduce design decision documentation in such synthesis architectures. An approach for product line synthesis architecture, where design decisions are introduced to promote its reuse.

This work focuses on how systems are synthesized in FOMDD. Feature Oriented Model Driven Development (FOMDD) is a blend of FOP and MDD that shows how products in a software product line can be synthesized in an MDD way by composing features to create models, and then transforming these models into executables. Synthesis in

FOMDD 1) initially using scripting (repetitive, time-consuming and cumbersome),2) recently, new approach (scripts generated from abstractions).

The documentation of design decisions constitutes a first step towards AK reuse, but documentation alone does not imply reuse. Approach basically documents the design decisions that happen along synthesis and use step-wise refinement to extend synthesis architectures and their design decisions. Architecture Extensibility includes steps: Synthesis Architecture, Extensibility, Architecture Composition, Documenting Design Decisions, Traceability and Design Decisions. This approach documents the decisions but basically useful for reusing architectural knowledge.

2.1.5.2 LIMITATIONS

Certain limitations have been found by studying and analyzing the research paper [18] as discussed below:

1) Approach doesn't provide any tool support. This approach should be applied on any tool.
2) Approach doesn't applied industrially and it has been checked from the research paper[18].
3) This approach doesn't helps building stakeholder communication as it is quite complex approach as compared to other approaches.
4) Approach doesn't support VBSE principles. it doesn't documents only the required set of information based on a priori understanding of who will benefit later on, from what set of information, and in which amount instead this is recording all the architectural knowledge information.

2.1.6 Using Patterns to Capture Architectural Design Decisions

Main Feature: For capturing architectural design decisions.

2.1.6.1 DESCRIPTION

Architecture patterns [39] are suggested here for capturing structural and behavioral information. The relation between patterns and decision making are discussed here and it is described that how architects can use patterns to capture certain architectural design

decisions in practice. Architecture patterns make information easier and faster to document architectural decisions. In applying architecture patterns, architects make decisions that encourage them to both reflect on those decisions and consider related issues. Pattern selection is indispensable to the architecting process, so architects can record related decisions with little effort. Patterns follow an easily understood form, which is highly compatible with proposed description templates for architectural decisions.

This approach[39] first compares the Patterns and Architectural decisions and then explains the pattern-decision relationship. Using a pattern in system design is, in fact, selecting one of the alternative solutions and thus making the decisions associated with the pattern in the target system's specific context. Although patterns and decisions have different origins, one can investigate their relation by comparing how they're documented.

Patterns have potential for providing very useful AK that architects can turn into application-specific knowledge and can document as an architectural asset. However, patterns can't help the architect of all the responsibility for documenting decisions. Firstly, the architect must still document application- specific decisions. Secondly, not all decisions have appropriate patterns. One can't capture some architectural decisions in terms of patterns because these patterns depend on the project's concrete scope and domain. An important challenge there is with patterns is what to do if developers use the wrong pattern but don't discover this until well into the implementation phase. As with any architectural decision, backing out is difficult.

2.1.6.2 LIMITATIONS

Certain limitations have been found by studying and analyzing the research paper [39] as discussed below:

1) Approach doesn't provide any tool support. It should be applied on any tool.
2) Approach doesn't support Value-Based Software Engineering principles. It doesn't documents only the required set of information based on a priori understanding of who will benefit later on, from what set of information, and in which amount instead this is recording all the architectural knowledge information.

3) Architectural knowledge plays an important role for the architectures in software product families. This approach currently is not useful for software product families but if its drawbacks will be removed then it might be useful for software product families.

2.1.7 A Value-Based Approach for Documenting Design Decisions Rationale (VB-DDRD)

Main Feature: For documenting design decisions rationales using value based approach.

2.1.7.1 DESCRIPTION

An approach [27,28, 29] for systematic DDR use that follows value-based software engineering principles. A value-based approach is proposed for documenting the reasons behind design decision (VB DDRD), based on a priori understanding of who will benefit later on, from what set of information, and in which amount. The basic idea of this approach is that all the information included in a design decision rationale documentation (DDRD) might be useful but sometimes some information are mere optional. Basically, researchers focuses on an approach which says not all the information is required to document all the time to relevant persons. Only required design decisions rationale information must be documented according to the choice of relevant person. Moreover, the amount of importance related to the information included in the DDRD depends on the DDRD Use-Case (DDRD UC). The adoption of a tailored DDRD, consisting only of the required set of information, would mitigate the effects of DDRD inhibitors. Several usage scenarios for DDRD (i.e. DDRD Use-Case) are considered/provided and characterized by specific payees, business context and product characteristics, type of DDR activity, benefits, and required DDR information. A set of thirteen scenarios have been selected.

Researchers[28,29] analyzed the feasibility of this approach, which includes certain aspects relevant for every software engineering practice, such as: *Where* (project context), *Who* (the beneficiary stakeholders), *When* (DDRD type of use), *Why* (process and product metrics) and *How* (DDRD required information), through a replicated experiment which adopts this approach and the importance of different DDRD information categories. This applied experiment employs 50 subjects, 25 decisions, 5 different DDRD UC(s), and 250

DDRD UC(s) executions. Each subjects practically used the documentation to enact all the five Use Case(s) by providing an answer and a level of utility for each category of DDRD. Researchers observed several difficulties in the reasoning activity to answer the questions that motivated each DDRD UC. Thus, from the reasoning activity of the subjects it is deduced that a good description of the decisions is needed in order to avoid confusion when reusing decisions made by others.

Researchers mentioned that DDRD consumers require specific categories of DDRD information according to the DDRD UC to enact. This suggests that the DDRD producer should include only the information required for the specific DDRD UC(s) that is expected to be enacted (i.e. the value-based approach). This concludes that for the use of DDRD that should be captured and used according to the user or organizational needs and use agile methods to reduce the effort both in creating and consuming such relevant architectural knowledge. Thus, documenting this tailored architectural knowledge explicit may be seen as a "new" cross-cutting architecture view i.e. "Decision View".

2.1.7.2 LIMITATIONS

Certain limitations have been found by studying and analyzing the research papers [27,28, 29] as discussed below:

1) Approach doesn't provide any tool support. VBSE principles have not yet applied to any tool.
2) Approach currently not being applied to software product families but it can be useful for software product families, as to get only the required information for product line software, this approach saves time and effort.

The above study shows the work in techniques/framework/method/approaches for managing, sharing and storing architectural knowledge. From this survey, a Value-Based approach has been found for documenting the Design Decisions Rationale, known as Value-Based Design Decision Rationale Documentation (VB-DDRD). Basically in that approach, researchers [27,28,29] have been applying Value-Based Software Engineering principles for documenting design decisions rationale. This is a useful approach as it documents only the set of required information based on its purpose, so we can do further work with this approach.

CHAPTER 3

LITERATURE SURVEY AND EVALUATION OF TOOLS

3. LITERATURE SURVEY AND EVALUATION OF TOOLS

Recently various studies show that the software architecture community has begun to recognize that knowledge management is vital for improving an organization's architectural capabilities [21]. There has been an increased demand for suitable methods, techniques, and tools that support organizations in capturing and maintaining the architecture design decisions and their rationales [15]. Architectural knowledge can be valuable throughout the software development lifecycle. Researchers and practitioners have proposed various approaches to capture and manage architectural knowledge [8,23,27]. Many of these approaches have been adapted from knowledge extraction techniques used in artificial intelligence and in social science disciplines. One of the main objectives of these approaches is to help making explicit what is known by architects or implicitly embedded in architecture. This may include knowledge about the domain analysis, architectural patterns used, design alternatives evaluated, and assumptions underpinning design decisions [21].

This chapter includes the study and review of existing tools for managing and storing architectural knowledge. Evaluation of existing tools for managing Architecture Knowledge is one of the contributions of this thesis has also been discussed in this chapter. Tools are evaluated on the basis of different attributes. Following section describe the evaluation criteria defined for existing tools:

3.1 Evaluation Criteria for existing tools

This section describes the attributes which have been taken to evaluate the studied tools. These attributes consists of general attributes which are usually taken to evaluate the tools as well as the specific attributes related to architectural knowledge [8,9,23,27,31]. The evaluation criteria attributes have been taken out after the literature mapping. First the Architectural Knowledge itself has been defined. The attributes of architectural knowledge has been listed down as shown in Table 1.1.It has been checked either the tools are covering the AK attributes or not. Table 3.2 that compares and contrasts the features of each tool is also helpful for providing few attributes. Further, some of the attributes have taken out by reading and surveying the area related papers. Knowledge

Management literature have been proposed out few things e.g. we come to know about Value-Based Software Engineering concept that the tools must support. General attributes have also been considered as there are some things which the tools must have e.g. usability attribute etc. Following are evaluation criteria attributes:

1. **Usability**: Usability is a quality attribute that assesses how easy user interfaces are to use[47]. It also refers to methods for improving ease-of-use during the design process. Usability is defined by five quality components: Learnability, Efficiency of use, Memorability, Error Prevention, Satisfaction. This attribute has been added to check how usable and learnable the selected tools are. Usability attribute is an important attribute and helps to get to know how learnable, efficient, memorable, safe and pleasant the tools are. Usability is a necessary condition for survival. It is important to find usability of any tool because if a tool is difficult to use, people leave. If users get lost on a tool, they leave. If a tool's information is hard to read or doesn't answer users' key questions, they leave. By using all the studied tools personally, it is possible to proof how learnable, efficient, memorable, safe, and satisfying these tools are for a given set of users. For evaluating tools with usability attribute, the main focus is on the interface of tools, Usable interfaces must possess five basic attributes :
    a. **Learnability.**—Tools must be easy to learn. Novice users must be able to complete basic tasks in a short period of time, with a minimum of training.
    b. **Efficiency of use.**— Once experienced users have learned the design, they must perform the task quickly.
    c. **Memorability.**—Tools must be easy to remember. Users can return to them after an absence and complete tasks without retraining.
    d. **Error Prevention.**—Users must experience only few errors while using the tools, and recover quickly from them.
    e. **Satisfaction.**—Tools must be pleasant to use.
2. **Industrially used:** This attribute shows that the selected tools are industrially used. This attribute is important because if the tool is not industrially used then it is impossible to find any evaluation report from the researchers and impossible to validate it through case studies. From the research papers and tools' website, it is easy to get to know either the tools are industrially used or only a research prototype.
3. **Open Source:** This attribute shows either the source code is available to the general public for use and/or modification from its original design free of charge or not. This

attribute is important if we want to do further iterations in the source code. Availability of code for a particular tool validates this attribute. Tool's website or researcher's website can be helpful for this attribute.

4. **Coverage:** Coverage attributes shows how much the selected tools covers architectural knowledge attributes as well as other features of these tools. For evaluating tools related to this area, coverage attribute plays an important role. This attribute is important to find limitations of each tool. From the research papers, technical reports and by using the tools personally it is possible to get to know how much these tools covers architectural knowledge attributes as well as the other features of these tools. A table 3.2 is developed that compares and contrast different features of each tool with one another as mentioned in the end of the chapter. The coverage attribute is evaluated on the basis of this table.
5. **Useful for software product families:** Software product families [43] [44] are recognized as an effective approach to reuse in software development. The basic reuse philosophy of software product families is intra-organizational reuse through the explicitly planned exploitation of the commonalities between related products. As architectural knowledge plays an important role for the architectures in software product families, this attribute tells either tools are useful for software product families or not and it has been checked from the research papers, web sites and other documentation of these tools.
6. **Support of Value-Based software engineering principles:** Value-Based Software Engineering agenda has emerged, with the objective of integrating value considerations into the full range of existing and emerging software engineering principles and practices, and of developing an overall framework in which they compatibly reinforce each other [26]. VBSE principles play an important role while capturing architecture knowledge. The importance of this attribute is that with this one can get required set of AK information that benefits different persons. Also saves a lot time and effort. This attribute tells us either these tools support this concept or not. This attribute has been evaluated on the basis of research papers, personal usage and other documentation.
7. **Useful in evolution and maintenance activities:** Software evolution is used to refer to the activity of adding new functionality to existing software [45]. Maintenance refers to the activity of modifying software after it has been put to use in order to maintain its usefulness [45]. Evolution and maintenance activities are very much

important for a particular architecture, as whenever some change occurs to existing software, the architecture of that software also changed, this is the reason why this attribute has been taken. From the research papers and personal usage, it is easy to get to know either these tools are useful in evolution and maintenance activities or not.

8. **Integrated with other modeling tools:** This attribute tells that either the tools are a separate tool or integrated with any other tool. The importance of this attribute is that from this one can check is it easy to use tool or a complicated tool. This attribute has been checked from research papers and personal usage. Personal usage involves either these tool are integrated within any other modeling tools or either some other tool are integrated in these tools.
9. **Accessible for geographically distributed stakeholders:** Stakeholders comes from different background and have different concerns. As stakeholders are distributed in different areas, this attribute tells either these tools are accessible for different stakeholders placed at different places or not. This attribute is important to check as stakeholder plays an important role for the projects and they can be distributed all over the world. If the tool is web-based then all the stakeholders can easily access that tool. From the research papers, this attribute have been validated for all the tools.
10. **Performance:** Performance is about timing [1]. Performance refers to responsiveness: either the time required responding to specific events or the number of events processed in a given interval of time [47]. Performance attribute plays an important role to check how efficient the tools are and how much time tools take to complete the task. By personally using the studied tools the performance of the tools has been checked. By providing same set of data and same actions to all the tools, we can evaluate tools with performance attribute on the basis of following requirements:
    a. Time the user takes to complete basic flow.
    b. Time to perform any process/flow of operation i.e. to save data, delete any data.
    c. Response time of any flow of operation i.e. retrieve data from the database.
    d. Time to start the application.
11. **Security:** Security attribute is a measure of system's ability to resist unauthorized attempts at usage or behavior modification, while still providing service to legitimate users. Security attribute is an important attribute that has been added in order to check how secure these tools are. In this case, Authentication property has only been checking either these tools possess this property or not. This

attribute has been checked by personally using the tools and from the research papers.

3.2 Review of Existing Tools

There are five tools for managing, sharing and storing architectural knowledge which have been evaluated by personally using these tools. Research papers, technical reports and other documentation have also been used for evaluating the studied tools.

3.2.1 PAKME (Process-based knowledge management environment)

An architectural knowledge management framework [21,23] has been proposed by the researchers of National ICT Australia (NICTA). This framework incorporates concepts from knowledge management, experience factory, and pattern-mining. It consists of various approaches to capture design decisions and contextual information. This framework involves an approach to distill and document architecturally significant information from patterns, and also involves a data model to characterize architectural constructs, their attributes and relationships. A web-based architecture knowledge management tool, called Process based Architecture Knowledge Management Environment (PAKME) [23,48], has been developed to support the proposed framework. PAKME is a tool for providing knowledge management for software architecture development. It has been built on the top of Hipergate, an open source groupware platform which includes collaborative features, project management facilities and online collaboration tools.

Some of the features and components of PAKME [21,23, 48] are as follows:

1) PAKME consists of five components:
    a. **Knowledge acquisition service:** the user interface implemented with JSP and HTML pages.
    b. **Knowledge maintenance service:** knowledge management component provides the services necessary to store and update AK.
    c. **Knowledge retrieval service:** the search component which defines three different searching mechanisms (i.e.: keywords, logical operators, and navigation).

d. **Knowledge presentation service:** for retrieving artifacts, the reporting component provides services for representing AK and describing the relationships between different architectural artifacts. Generates automatically PDF documents/ Web based report for describing decisions etc.

e. **Repository Management:** Offers the services needed to maintain the data which is currently implemented in PostgreSQL.

2) Captures Architectural Design Decisions and their underlying rationales i.e. it captures both Contextual Information (Design Rationale) and Technical Information (Patterns, Styles, Tactics, Analysis models, Scenarios).

3) PAKME divides AK into organizational (generic) and project-specific (concrete).

a. *Generic*: including general scenarios, patterns, quality attributes, design options.

b. *Project specific:* including concrete scenarios, contextualized patterns, quality factors, and architecture decisions.

4) Incorporates AKM features for geographically distributed stakeholders involved in the software architecture process.

5) Improves architecture-based software development.

6) Provides different templates, knowledge repository, and various features to capture, manage, and present architectural knowledge.

7) Capture, manage, use, reuse and retrieve AK captured from Human Sources and patterns.

8) Capture and present scenarios.

9) Support for design and analysis methods.

10) In PAKME, architecture decisions were categorized as risk or nonrisks.

11) Supports the architecture evaluation process and systematizes evaluation process of an industrial collaborator.

3.2.1.1 EVALUATING PAKME

1. **Usability:**

PAKME is user-friendly tool. One can learn the tool easily as it provides proper user-interfaces which enables user to add/retrieve knowledge easily. User can easily complete task in a short period of time as PAKME provides different templates, knowledge repository, and various features to capture, manage, and present

architectural knowledge easily. Moreover, as PAKME provides proper templates to capture and manage architecture knowledge, also provides Knowledge maintenance, acquisition, retrieval, presentation and repository management services so an experience user can easily use this tool. Moreover proper menu and navigation services are provided with this the tool become more efficient.

As proper interfaces are provided so one can easily remember this tool. With the knowledge retrieval service, one can easily return to this tool and get the required data after an absence. PAKME generates automatically PDF documents/Web based report for describing decisions etc which enables users to easily found architecture design decisions as well as their rationales for a particular architecture of any project. Only few errors have been found due to the absence of JSP server and found duplication of workload of requirements. Besides few errors, PAKME is pleasant to use tool. The usability of PAKME has been checked and evaluated by personally using the tool.

2. **Industrially used:**

   PAKME has been tested industrially for an aircraft system. PAKME is trialled to help systematize the architecture knowledge management and evaluation process of an industrial collaborator. The opinion about this attribute has been formed from the research papers [23,48], technical and evaluation reports[21,48].

3. **Open Source:**

   PAKME is not an open source tool as tool's source code is not available. The opinion about this attribute has been formed from the research papers [23], technical reports [21] and from the PAKME's developers.

4. **Coverage:**

   Table 3.2 that compares and contrast the features of each tool with one another can be useful to evaluate PAKME with this attribute. PAKME captures Architectural Design Decisions and their underlying rationales. Also captures both Contextual and Technical Information. This tool covers many features like searching, presenting knowledge, support for evaluation process etc. Table 3.2 shows PAKME covers many features which other tools except ADDSS are not covering.

5. **Useful for software product families:**

   Papers [23] shows that tool is not useful for software product families as there is not anything mentioned in the paper about the usage of this tool for software product

families. This attribute has been checked from the research papers [23,48], evaluation report and other documentation of PAKME [21].

6. **Support of value-based software engineering principles:**

   PAKME doesn't support the VBSE concept as tool doesn't capture design decisions information according to user's choice. User's have no choice to capture required design decisions information. All the design decision information has been captured not focusing only on required set of information. This attribute has been evaluated with the help of research papers [23,48] and technical reports [21].

7. **Useful in evolution and maintenance activities:**

   Fully supports the evolution and maintenance activities and this attribute have been checked from the research papers [48] and its technical report [21].

8. **Integrated with other modeling tools:**

   Tool is not integrated with other modeling tools. This attribute has been checked from research papers [23] and personal usage.

9. **Accessible for geographically distributed stakeholders:**

   PAKME incorporates AKM features for geographically distributed stakeholders involved in the software architecture process. From the research papers[23,48], this attribute have been validated and this tool fully supports it.

10. **Performance:**

    PAKME is good performance wise as it provides proper templates so with less time one can easily store and retrieve architecture knowledge information, but due to few problems as discussed below, it is concluded that PAKME partially supports the performance attribute. Although tool takes less time to complete basic flow but there is one drawback i.e. duplication of workload of requirements which some times slows down its performance. PAKME needs to be designed to be heavily customized at deployment time. PAKME takes 4-5 seconds to save information whenever we press the save button. It takes 1-2 seconds for deleting information. PAKME takes time while retrieving knowledge. Whenever we gave command for viewing architecture knowledge information, system responses in 20-30 seconds. Improving the speed of knowledge retrieval by using the task-based retrieval techniques is needed in this tool. PAKME has proper interface so its takes few seconds to start the tool. The performance of PAKME has been measured by personally using the studied tool

11. Security:

Tool doesn't support this attribute. Tool doesn't ask for any login/password. This attribute has been checked by personally using this tool.

3.2.2 ADDSS (Architecture Design Decision Support System)

The Architecture Design Decision Support System (ADDSS) 1.0, was available at [36], developed in 2005-2006 [8]. ADDSS [8,9] is an open web-based tool developed with PHP, HTML and MySQL. ADDSS is an open source tool whose code is also available at their website [36]. Tool focuses on recording, managing and documenting architectural design decisions under an iterative development process. ADDSS creates the architecture under an iterative process and where one or more design decisions are made for each of the iterations.The new ADDSS 2.0,[9] which has been released, supports the status of decisions and the date when the decision was made. Also, supports versioning for recording and tracking the history of a particular decision and supports the decision making process. Tool describe a flexible approach in the form of mandatory and optional attributes for characterizing architectural design decisions that can be tailored to the specific needs of each particular organization. Researchers [9] are fixing these mandatory and optional attributes according to their choice.

Some of the features of this tool [8,9] are:

1) Supports the creation, use, maintenance, and documentation of architectural design decisions. Enables to document design decisions as first class entities under an iterative approach.
2) Design decisions easily visualized understood and replayed.
3) Supports the implementation of decision view.
4) An easy web interface provides access to the functionality of the tool.
5) Tool can access through username and a password.
6) Users can upload figures from external files representing architectures. For each of the iterations, a thumbnail image of the architecture is shown to the user.
7) Allows the storage of several projects and architectures. Multi-perspective support for different stakeholders.
8) Add or remove well-known architecture and design patterns and architectural styles to the database.

9) PDF documents containing the project and architecture descriptions with the design decisions can be automatically generated.
10) Support for functional and non-functional requirements and for different architectural views.
11) Support the decision making process.
12) Supports traceability between requirements, decisions, and architectures, but detecting inconsistencies of decisions is not implemented.
13) Useful for maintenance and evolution activities. Supports modeling and documenting the evolution of ADD.

3.2.2.1 EVALUATING ADDSS

1. **Usability:**

ADDSS is a user friendly tool. One can easily learn the tool as it provides an easy web interface which offers access to the functionality of the tool. ADDSS is quite usable but as the code is in Spanish language so at some places in the interface Spanish language is used which is difficult for the users to understand the tool. User can easily complete task in a short period of time as the tool provides different templates to capture, manage, and present architectural knowledge easily. A simple user interface using web technologies are used for representing the decisions and architectures as well as the multi-perspective support. This tool provides proper knowledge management, retrieval and presentation services for managing and storing architecture knowledge. Design decisions easily visualized understood and replayed. Moreover proper menu and navigation facilities are provided with this the tool become more efficient.

As proper interfaces are provided so one can easily remember this tool. With the knowledge retrieval service, one can easily return to this tool and get the required data after an absence. The decisions made by the user can be visualized afterwards in order to understand the rationale behind them. Users can easily visualize the history of the architecture and the iterations performed. Few errors have been found in saving like saving architecture information, images etc. 'Page cannot displayed" error appears on certain pages. Besides few errors, ADDSS is pleasant to use tool. The usability of ADDSS has been checked and evaluated by personally using the tool.

2. **Industrially used:**

   ADDSS is a research prototype. The opinion about this attribute has been formed by visiting tool's website[36].

3. **Open Source:**

   ADDSS is an open-source tool because tool's source code is available from the website [36]. Although the code of the tool is available but it is written in Spanish language so this make sometimes difficult for the developers who do not understand this language. This attribute has been validated by personally visiting the website [36].

4. **Coverage:**

   Table 3.2 that compares and contrast the features of each tool with one another can be useful to evaluate ADDSS with this attribute. ADDSS supports the creation, use, maintenance, and documentation of architectural design decisions. Enables to store and document design decisions as first class entities under an iterative approach. Support for integrated representation of architecture. Tool captures the rationales underlying the design decisions. This tool covers many features like searching, presenting knowledge, support for evaluation process etc. Table 3.2 shows ADDSS covers more features as compared to other tools. Table 3.2 also shows there are few features which other tools are covering and doesn't support by ADDSS.

5. **Useful for software product families:**

   ADDSS is not useful for software product families as there is not any thing mentioned in the paper about the usage of this tool for software product families. This attribute has been checked from the research papers[8,9] and web sites[36] .

6. **Support of Value-based software engineering principles:**

   ADDSS doesn't support the VBSE [26] concept as tool is not capturing design decisions information according to user's choice. Although ADDSS provides a flexible approach of mandatory and optional attributes for characterizing design decisions information but they are fixed by the researcher's and users don't have the choice to mark fields as required, useful according to their choices. One cannot mark the mandatory field as optional or optional field to mandatory. This attribute has been evaluated with the help of research papers [8,9].

7. **Useful in evolution and maintenance activities:**

   ADDSS supports modeling and documenting the evolution of ADD. This attribute have been checked from the research papers [8,9].

1) Retrieve architectural decision and their rationales. Allows for run-time addition of additional design decisions.
2) Tool's compiler translates Archium code in a combination of Java and ArchJava code, which is then transformed into Java byte code.
3) Also a component language, which extends Java for describing components, connectors, and design decisions with tool support.
4) Provides visualization facilities for the decisions made using a dependency graph
5) Archium also captures consequences of an architectural decision.
6) Supports the tracing of requirements to architectural decisions and is able to check which of these requirements are addressed in one or more decisions.
7) Checks implementation against architectural design decisions. Weaves architectural decisions into architectural models and connects them to the implementation.
8) Check for consistency and for completeness. Check for superfluous architectural decisions and circularity of set of decisions is also provided (not automatically).
9) Captures rationale in customizable rationale elements.
10) It decreases the effects of knowledge vaporization.

3.2.3.1 EVALUATING ARCHIUM

1. **Usability:**

   ARCHIUM is not a user-friendly tool. Novice user cannot learn tool easily as it doesn't provide user-interface. The tool is not usable because sometimes one doesn't want to learn the coding language or to install the whole J2EE environment. ARCHIUM integrates an architectural description language (ADL) with Java to describe the elements from a component & connector view and making explicit the design decisions and its rationale that lead to a particular architecture description[17]. For novice user it is difficult to use this tool without having knowledge of Java language. A lot of training is required for them to understand this tool. One need to first setup java environment, its path etc. User without having knowledge of java cannot access this tool. Novice user cannot set java path easily. This makes also difficult for experienced users.

   Although one can retrieve architectural decision and their rationales from Archium but as there is no proper interface, it is difficult to remember the tool. As every time to start the application one have to set path and have to go through from code, this

makes difficult to remember the tool if one is absent for a while. Archium provides a visualization of an architectural decision but the way to view these decisions is difficult which makes hard to remember the tool. Many errors have been found during compilation and at run time. Run time addition of design decisions is a difficult task. ARCHIUM is not a pleasant to use tool. The usability of ARCHIUM has been checked and evaluated by personally using the tool.

2. **Industrially used:**

   ARCHIUM tool has not being tested yet in an industrial setting. The opinion about this attribute has been formed from the research paper [17] and tool's website [34].

3. **Open Source:**

   ARCHIUM is not an open-source tool because tool's source code is not available. This attribute has been validated by from the research paper[17] and website[34].

4. **Coverage:**

   Table 3.2 that compares and contrast the features of each tool with one another can be useful to evaluate Archium with this attribute. Tool captures architecture design decisions and their rationales. Partially support for integrated representation of architecture. Table 3.2 shows tool covers less number of features as compared to other features.

5. **Useful for software product families:**

   Tool is not useful for software product families as it is difficult to set path etc for software product families as this tool is time consuming and require much effort. This attribute has been checked from the research paper[17] and web site[34] .

6. **Support of Value-based software engineering principles:**

   ARCHIUM doesn't support the VBSE concept as we are not capturing design decisions information according to user's choice. User's have no choice to capture required design decisions information. All the design decision information has been captured not focusing only on required set of information. From the research paper, this attribute has been checked.

7. **Useful in evolution and maintenance activities:**

   ARCHIUM is partially useful in evolution and maintenance activities. Design decisions and their rationales are recorded in this tool at compile time which causes difficult for the maintainers as all the maintainers are not familiar with the java language. This attribute have been checked from the personal usage and research paper [17].

8. **Integrated with other modeling tools:**

   Tool is not integrated with other modeling tools. This attribute has been checked from research paper [17] and personal usage.

9. **Accessible for geographically distributed stakeholders:**

   Tool doesn't support this feature. As this tool is not a web-based tool this makes difficult for geographically distributed stakeholders to access the tool. From the research paper[17], this attribute have been validated.

10. **Performance:**

    ARCHIUM is not good performance-wise as it doesn't provide proper interface. To perform any task one need to compile and run the code which takes a lot of time. To save information it takes 4-5 minutes. Similarly retrieving data is also a difficult task. Whenever we send command, system responses in 1-2 minutes. Every time to start the application the compiler translates Archium code in a combination of Java and ArchJava code, which is then transformed into Java byte code. Compiling and running the code takes a lot of time to start the application. The performance of ARCHIUM has been measured by personally using the studied tool

11. **Security:**

    Tool doesn't support the security attribute. No security feature is added in this tool like username etc. The opinion about this attribute has been formed from the research paper[17] and tool's website[34].

3.2.4 KNOWLEDGE ARCHITECT WORD PLUGIN

It is the part of Griffin project [33]. Griffin project consists of methods, tools, and techniques to manage architectural knowledge. It is a software architecture project memory to manage know-why and know-how. Knowledge Architect Word [32] is a tool to capture Architectural knowledge. This tool allows architects to make their Architectural Knowledge (AK) explicit in architecture documents written in Microsoft Word. Installing the Knowledge Architect Word Client creates a button bar in Microsoft Word. Tool connects to a server before you can use the other features of the client.

Some of the features of this tool [32] are:

1) It is a .Net tool and easy to use tool.

2) This Plug-in covers the following attributes of Architectural knowledge: Knowledge Entity, Concern, Requirement, Risk, Decision Topic, Alternative, Decision, Quick Decision, Specification.
3) Knowledge Entity Form in this plug-in allows users to give name of knowledge entity and the Knowledge Entity type can be selected.
4) The user can specify a custom status, or select one that is predefined. User can add notes to a Knowledge Entity.
5) Allows creating Knowledge Entity table. Specified tables can be generated by specifying what Knowledge Entity types should be shown and what connections to show.
6) Tool shows summaries of all Annotations and their connections. Can export the Annotations to an XML file or import all Annotations into the backend.
7) Allows customizing the default colors for knowledge entities. You can specify both font color as well as the background color.
8) Provides different context menus depending on what user selects,
    a. Plain text selection ,Single Annotation ,Overlapping Annotations ,Annotation with Completeness Coloring.
9) Shows multiple errors if occur. Different error level is shown by colored flags, with higher level errors on top in the list.
10) Provides the facility of Completeness Check. Completeness is based on a number of tests.

3.2.4.1 Evaluating Knowledge Architect Word plug-in

1. **Usability:**

   Knowledge Architect Word is a user friendly tool. One can easily learn the tool as it allows architects to make their Architectural Knowledge (AK) explicit in architecture documents written in Microsoft Word. Novice users can easily work in Microsoft word document so they can easily complete their basic task with this tool with minimum training. Little training is required as there are certain terms which the tool is using from which a common user is not familiar.

   Knowledge Architect Word is integrated within Microsoft word so this does not provide separate interface. Tool provides screen within the word document so

experience users can produce their work but with little effort and it take some time first to understand this tool. As tool documents the relevant architecture knowledge within word document, so the tool is easy to remember as the previous documented AK is saved and retrieved easily. Users can return to it after an absence and complete tasks without retraining. Error occurs when the Word plug-in cannot find the web services on the server. With this one cannot able to run the tool. Due to above error and training required, this tool is partially pleasant to use. The usability of tool has been checked and evaluated by personally using the tool.

2. **Industrially used:**

   Tool is partially industrially used as its developers are currently applying this tool on industrial case studies. It is partially industrially used as case studies from industry are applying on it. The opinion about this attribute has been formed from the tool's website and the tool's developers [32].

3. **Open Source:**

   Tool is not an open-source tool because tool's source code is not available. This attribute has been validated by from the website[32].

4. **Coverage:**

   Table 3.2 that compares and contrast the features of each tool with one another can be useful to evaluate tool with this attribute. Tool captures architecture design decisions. It partially covers the architectural knowledge attributes. Table 3.2 shows tool covers less number of features as compared to other features.

5. **Useful for software product families:**

   Tool is not useful for software product families as this is installed in MS Word and it might be difficult to use MS Word document to capture AK for software product families. This attribute has been checked from website [32] and other documentation of this tool.

6. **Support of Value-based software engineering principles:**

   Tool doesn't support the VBSE concept as we are not capturing design decisions information according to user's choice. User's have no choice to capture required design decisions information. All the design decision information has been captured not focusing only on required set of information. This attribute has been validated by personally using the tool.

7. **Useful in evolution and maintenance activities:**

   Tool is partially useful for maintenance and evolution activities as tool is integrated within the word document so to make changes within an already developed tool sometimes create problem. This attribute has been checked by personally using the tool.

8. **Integrated with other modeling tools:**

   Tool is integrated with Microsoft word. This attribute has been validated from tool's website [32] and personal usage.

9. **Accessible for geographically distributed stakeholders:**

   Tool doesn't support this feature as tool is integrated within the word document and it might be possible as all the stakeholders have not the latest document. Moreover, tool is not a web-based tool which creates problems for geographically distributed stakeholders. From the personal usage and tool's website[32], this attribute have been validated for the tool.

10. **Performance:**

    Performance wise it is partially good as it works in a document some times it is difficult to look into document. Moreover, tool is using certain terms which are not familiar to novice user. User takes time to complete the basic task. As the tool is integrated into Microsoft word, it takes 1-3 minutes to store information on a given set of data. However, the response time of a system for retrieving data is very good and i.e. 1-2 seconds. Tool provides a user friendly interface, so it takes few seconds to start the tool. The performance of tool has been measured by personally using the studied tool.

11. **Security:**

    Tool doesn't support this feature. No security feature is added in this tool like username etc. This attribute has been checked by personally using this tool.

3.2.5 AREL (Architecture Rationale and Element Linkage)

AREL Tool set [35,46] is a tool set comprises of three components, Enterprise Architect, Netica, and custom-built AREL Tool. This tool set enables the capture of architecture design and its design rationale. It provides an approach named as AREL. AREL approach is a model which relates decisions, architecture products and design rationale. It implements a conceptual model to relate ADD, architectures and the rationale of the

decisions. First, the Enterprise Architect tool is used to construct the AREL models and to capture design rationale by using extended UML profiles. Secondly, the complementary approach eAREL supports decision evolution and their tracing by means of versioning links. It supports backward and forward tracing through history. Each decision encapsulates its rationale, but there is only one link type, i.e. "depends-on", defined in this method.

Some of the features [35,46] are:

1) By using extended UML profiles one can capture architecture design rationale and architecture design.
2) Integrates a commercially available BBN tool to reason about the architecture design.
3) Improves the representation of design rationale for architecture development.
4) Improves and estimate change impact analysis.
5) Support consistency checking of the AREL models and AREL model tracing.
6) Captures qualitative rationale, quantitative rationale and alternative architecture rationale. Captures Requirements, Assumptions, Constraints, Design Objects.
7) Enables architects to have a better understanding of the problem, the associated costs and complexity of the design before committing to development.
8) Facilitates verification by peer review and stakeholders review.
9) By using the stereotype extension in Enterprise Architect, provides a convenient way to input design rationale using the design rationale capture templates.
10) Displays a screen to show any errors or warnings. A detailed error report is also produced.

3.2.5.1 EVALUATING AREL TOOL SET:

1. **Usability:**

   It is not a user friendly tool. No proper user interface available for capturing design rationales. Three different tools are comprises here, difficult to use for a common user. Certain errors found like AREL operations cannot be tightly integrated with Enterprise Architect. For instance, the AREL operations cannot be directly activated from the Enterprise Architect menu options to fully integrate AREL functionalities into Enterprise Architect. Second, we have no access to the source code of either Enterprise Architect or Netica, therefore there cannot be a seamless integration where

prior probabilities and conditional probability tables can be captured and computed within the UML tool.

Experience user finds this tool-set difficult as it requires a lot of training to understand it and its usage is quite time-consuming. One cannot remember this tool if he is absent for a while as he have to construct model for capturing design rationale and have nothing to write as text this makes it difficult to understand to complete task later on as he don't understand where he were. The above discussion concludes it is not pleasant to use tool. The usability of tool has been checked and evaluated by personally using the tool.

2. **Industrially used:**

   AREL tool-set has been partially tested industrially as researchers have applied case study on it. However researchers [35,46] mentioned that the AREL tool-set is only a proof-of-concept, its effectiveness in practice requires a formal evaluation. The opinion about this attribute has been formed from the research paper [46] and tool's website [35]

3. **Open Source:**

   Tool is not an open-source tool because tool's source code is not available. This attribute has been validated by from the tool's documentation [35].

4. **Coverage:**

   Table 3.2 that compares and contrast the features of each tool with one another can be useful to evaluate tool with this attribute. Tool captures design rationale and partially architecture design decisions. Support for integrated representation of architecture. Table 3.2 shows tool covers less number of features as compared to other features.

5. **Useful for software product families:**

   Tool is not useful for software product families as AREL is a tool-set in which three tools are involved and each tool works separately which causes difficult working for software product families. This attribute has been checked from the documentation [35] of the tool.

6. **Support of Value-based software engineering principles:**

   Tool doesn't support the VBSE concept as we are not capturing design decisions information according to user's choice. User's have no choice to capture required design decisions information. All the design decision information has been captured

not focusing only on required set of information. This attribute has been checked from the documentation [35] of the tool.

7. **Useful in evolution and maintenance activities:**

   AREL tool-set is partially useful for maintenance and evolution activities. Tool-set compromises three different tools and it is difficult sometimes for the maintainers to use the tool-set as they found hard to work on three different tools. This attribute has been checked from the tool's documentation [35].

8. **Integrated with other modeling tools:**

   Tool is integrated with Enterprise architect and Netica. The tool-set itself also requires better integration between UML modeling and BBN computation. This attribute has been checked from documentation [35] and personal usage.

9. **Accessible for geographically distributed stakeholders:**

   Tool doesn't support this feature. AREL Tool-set compromises three different tools and it is difficult to access geographically by different stakeholders. From the documentation [35], this attribute has been validate for the tool

10. **Performance:**

    Performance wise it is not good as we have to first capture rationale by designing it in enterprise architect which consumes a lot of time to complete the basic task. For a given set of information tool takes 10-15 minutes to capture design rationale. Similarly for retrieving the information, system responses in 4-5 minutes. No proper user interface available for capturing design rationales. AREL operations cannot be tightly integrated with Enterprise Architect i.e. the AREL operations cannot be directly activated from the Enterprise Architect menu options to fully integrate AREL functionalities into Enterprise Architect. As there are three different tools are comprises here, difficult to use for a common user. Tool takes a lot of time to run this tool-set. The performance of tool has been measured by personally using the studied tool.

11. **Security:**

    Security attribute has not being supported by AREL tool-set. No security feature is added in this tool like username etc. This attribute has been checked by personally using the tool.

Table 3.1 shows the evaluation criteria for the existing tools in summary form. For evaluating tools, ratings are defined. ✓ shows that the attribute is fully supported by the tool, • shows that the tool partially supports that attribute and X shows the tool doesn't support that attribute.

Table 3.1 Evaluation Criteria for tools

***Ratings: ✓: Fully supported, •: Partially supported, X: Unsupported**

Attributes	PAKME	ADDSS	ARCHUIM	KNOWLEDGE ARCHITECT WORD	AREL
1. Usability	✓	✓	X	•	X
2. Industrially used	✓	X	X	•	•
3. Open Source	X	•	X	X	X
4. Coverage**					
❖ Architectural Knowledge					
▪ Integrated representation of the software architecture	X	✓	✓	X	✓
▪ Architecture design decisions	✓	✓	✓	✓	•
▪ Rationales underlying the design decision	✓	✓	✓	✓	✓
▪ External context/ environment	✓	•	•	•	X
❖ Features	•	•	X	X	X
5. Useful for software product families	X	X	X	X	X
6. Support of value-based software engineering principles	X	X	X	X	X
7. Useful in evolution and maintenance activities	✓	✓	•	•	•

8. Not Integrated with other modeling tools	✓	✓	✓	X	X
9. Accessible for geographically distributed stakeholders	✓	✓	X	X	X
10. Performance	•	✓	X	•	X
11. Security	X	✓	X	X	X

***Coverage attributes shows the coverage of architectural knowledge attributes as well as other features each tool possess. Features attribute and architectural knowledge attributes are evaluated on the basis of table 3.2 which compares and contrasts the features of each tool with other. In this way we can easily find which tool covers more features then others.*

The following table shows the comparison and contrast between the features of each tool. Features have been divided into groups. For these features, ratings are defined. ✓ shows that the feature is fully supported by the tool, • shows that the tool partially supports that feature and X shows the tool doesn't support that feature.

Table 3.2 Compares and Contrast the features of each tool

***Ratings: ✓: Fully supported, •: Partially supported, X: Unsupported**

Features	PAKME	ADDSS	ARCHIUM	KNOWLEDGE ARCHITECT WORD	AREL
Group 1: Managing Architectural Knowledge	✓	✓	•	•	•
Capture, manage, present and retrieve Architectural knowledge	✓	✓	•	•	•
Share and Store architectural knowledge	✓	✓	•	•	•
Supports the creation, capturing, use, maintenance, and documentation of	✓	✓	•	•	•

architectural design decisions					
Captures rationales for design decisions.	✓	✓	✓	✓	✓
Captures Technical Information (Patterns, Styles, Tactics, Analysis models & Scenarios)	✓	•	•	X	X
Support and store design decisions as first class entities under an iterative approach	X	✓	•	X	X
Group 2: Supports architectural knowledge reuse	✓	✓	•	•	X
Reusing architectural knowledge	✓	✓	✓	✓	X
Reuse AK through different projects	✓	✓	X	X	X
Possible to reconstruct architecture	•	•	X	X	X
Group 3: Improves the efficiency of software *architecting process*	✓	✓	•	•	•
Support for the software architecture process	✓	✓	•	•	X
Improves architecture-based software development.	✓	✓	•	✓	•
Decrease the effect of knowledge vaporization.	✓	✓	✓	✓	X
Improves the representation of design rationale for architecture development	X	•	•	X	✓
Improves and estimate change impact analysis	•	•	•	X	✓
Effective communication of design rationale	•	•	•	X	✓
Complexity control	✓	✓	X	X	X
Highlight design complexity before implementation	•	•	X	X	✓
Group 4: Knowledge	✓	✓	•	•	•

acquisition service					
User Interface	✓	✓	X	✓	X
Users can upload figures/images	X	✓	X	X	X
Templates to capture, manage, store and present architectural knowledge	✓	X	•	X	•
Capture architecture design rationale and architecture design by using extended UML profiles	X	X	X	X	✓
Add Knowledge Entity, Context	X	X	X	✓	X
Create Knowledge Entity table	X	X	X	✓	X
Group 5: Knowledge maintenance service	✓	✓	X	X	X
Knowledge management	✓	✓	X	X	X
Repository Management	✓	✓	X	X	X
Allows the storage of several projects and architectures	•	✓	X	X	X
Group 6: Knowledge retrieval service	✓	✓	•	•	X
Search	✓	✓	•	•	X
Design Decisions easily replayed	✓	✓	•	X	•
Group 7: Knowledge presentation service/ Reporting	•	✓	•	•	•
Generates automatically PDF documents/ Web based report for describing decisions, architectural products etc	✓	•	X	X	X
Presents knowledge using representation mechanisms like utility, results, decision trees or network	✓	✓	•	•	X
Visual representation (explicit	X	✓	✓	X	✓

graphical notation)					
Design decisions easily visualized	•	✓	•	X	•
Establish dependencies and constraints between decisions.	•	✓	✓	•	X
Shows annotations lists and Context Menu	X	X	X	✓	X
Colors for knowledge entities	X	X	•	✓	X
Group 8: Support for different stakeholders	✓	✓	X	X	X
Multi-perspective support for different stakeholders	✓	✓	X	X	X
Different categories of users & permissions	•	✓	X	X	X
Group 9: Support for Patterns & Styles	✓	✓	X	X	X
AK captured from Human Sources and patterns	✓	•	X	X	X
Add or remove well-known design patterns and architectural styles to the database	✓	✓	X	X	X
Group 10: Support for different architectural views	•	✓	X	X	X
Implementation of decision view	X	✓	X	X	X
Group 11: Support for architecture evaluation process	✓	✓	•	•	•
Useful for evolution and maintenance activities	✓	✓	•	•	•
Modeling and documenting the evolution of ADD	✓	✓	X	X	X
Categorizes as risk or nonrisks.	✓	X	X	✓	X
Group 12: Supports traceability	✓	✓	✓	✓	✓

Improves traceability between requirements, architecture and implementation	✓	✓	✓	✓	✓
Several types of relationships among ADD can be handled	✓	✓	✓	✓	X
Validate the set of ADD against the requirements	✓	✓	✓	✓	✓
Group 13:Support for alternative decisions and requirements	✓	✓	✓	•	•
Support for alternatives decisions	✓	✓	✓	X	•
Support for functional and non-functional requirements	✓	✓	✓	✓	✓
Group 14:Support methods and processes	✓	•	•	X	✓
Support for design and analysis methods	✓	X	•	X	•
Supports Decision making activity	•	✓	✓	X	✓
Support design reasoning process	X	X	X	X	✓
Binds architecture decisions, models and system implementation.	X	X	✓	X	X
Relates decisions, architecture products and design rationale	✓	✓	✓	•	✓
Represents design rationale, design objects and their relationships	•	•	•	•	✓
Group 15: General Checks	X	•	✓	•	•
Check for consistency	X	X	✓	X	X
Check for superfluous & obsolete decisions	X	•	✓	X	X
Check for completeness	X	X	✓	✓	X
Get consequences of an ADD	X	X	✓	X	X

Shows multiple errors	X	•	✓	✓	•
Support consistency checking of the AREL models and to support AREL model tracing	X	X	X	X	✓

3.3 Limitations

To sum up the above study, certain limitations have been found from the above mentioned tools. There are some features which one tool is covering but the other tool is not covering those features. By evaluating each tool as discussed in section 3.2 on the basis of different attributes, the major limitations and drawbacks of each tool have been found. Table 3.1 clearly shows which tool has some limitations or drawbacks. Moreover by comparing and contrasting features of each tool with one another as discussed in Table 3.2, certain limitations have been found. Research papers, technical reports and other documentation of each tool have also been very useful for listing down the limitations of each tool. Every tool has some limitations some of which are mentioned below:

3.3.1 PAKME Limitations

Although PAKME covers a lot of features but still certain limitations have been found. Some of these [21,23,48] are:

1) No support of Value-Based software engineering principles.
2) PAKME's templates should be configurable based on organizational needs. Templates need to be configurable by users based on their needs.
3) Duplication of workload of requirements.
4) Does not supports diagrammatic modeling of design decisions rather its focus is on providing a handbook of architecture knowledge.
5) Should be integrated with their requirements management tool, if it is to be widely used within large environment.
6) PAKME needs to be designed to be heavily customized at deployment time.
7) Does not enable to store and document design decisions as first class entities under an iterative approach like ADDSS tool.
8) Not useful for software product families.

9) PAKME doesn't provide any security features. Need to improve the speed and accuracy of knowledge retrieval.
10) Does not check implementation against architectural decisions. Does not get consequences of an architectural decision.
11) Does not check for consistency, completeness and for superfluous decisions.
12) Does not differentiate the functional requirements and non functional requirements. Quality Attributes should be captured.

3.3.2 ADDSS Limitations

ADDSS tool is quite stable tool as compared to other four tools. This covers a lot of features which other tool doesn't cover but there are still some limitations in this tool which are as follows [8,9]:

1) ADDSS is a research prototype; this tool should be tested in an industrial setting.
2) No support of Value based software engineering in this tool.
3) Tool should allow the connection to other existing analysis and design tools in order to import/export requirements and architectures.
4) No proper templates to capture, manage, and present architectural knowledge like PAKME tool, it only contains web-forms.
5) Does not differentiate the functional requirements and non functional requirements. Quality Attributes should be captured.
6) No catalogue of architecture and design tactics.
7) Does not capture and present scenarios (general and concrete).
8) Does not support for design and analysis methods
9) Does not categorize risk and non risks
10) Does not check implementation against architectural decisions. Does not get consequences of an architectural decision
11) Does not check for consistency and for completeness.
12) Not useful for software product families.

3.3.3 Archium Limitations

Archium tool doesn't cover certain features which ADDSS and PAKME covers. So these are the limitations in that tool [17].

1) The Archium tool has not been tested yet in an industrial setting, so empirical verification data is not yet available.
2) No support of value based software engineering in this tool.
3) No proper templates to capture, manage, and present architectural knowledge like PAKME tool.
4) No catalogue of architecture and design tactics.
5) Does not capture and present scenarios (general and concrete).
6) Does not store architectural documents.
7) Does not support for design and analysis methods.
8) Does not support for standards such as IEEE 1471-2000 as PAKME.
9) Does not categorize risk and non risks.
10) Not useful for software product families.
11) Not a user-friendly tool as user interface has not provided. Difficult to understand tool as it is in Java and common user does not know how to set environment for the tool.
12) Does not provide any security features. Performance wise not good.
13) Does not provide multi-perspective support for different stakeholders.
14) Does not differentiate the functional requirements and non functional requirements. Quality Attributes should be captured.

3.3.4 Knowledge Architect Word Limitations

Knowledge word architect has certain limitations which are as follows [33]:

1) No support of Value Based Software Engineering in this tool.
2) No proper templates to capture, manage, and present architectural knowledge like PAKME tool, it captures AK in word document.
3) Does not enable to store and document design decisions as first class entities under an iterative approach.
4) Does not support diagrammatic modeling of design decisions.
5) No catalogue of architecture and design tactics.
6) Does not capture and present scenarios (general and concrete).
7) Does not support for design and analysis methods.
8) Does not store architectural documents.
9) Does not categorize risk and non risks.

10) Does not check implementation against architectural decisions. Does not get consequences of an architectural decision.
11) Does not check for consistency and for superfluous decisions.
12) Not useful for software product families.
13) Does not provide any security features and performance wise not good. Need to improve the speed and accuracy of knowledge retrieval.
14) Does not differentiate the functional requirements and non functional requirements.

3.3.5 AREL Limitations

Literature review of above mentioned tools show that AREL tool has a lot of limitations as compared to other tools. Some of these are [35,46]:

1) AREL operations cannot be tightly integrated with Enterprise Architect. For instance, the AREL operations cannot be directly activated from the Enterprise Architect menu options to fully integrate AREL functionalities into Enterprise Architect.
2) The AREL tool-set is a proof-of-concept and it is immature for real-life applications. This is because a number of usability features must be implemented if it is to be widely used in a commercial setting.
3) No support of value based software engineering in this tool.
4) No proper templates to capture, manage, and present architectural knowledge like PAKME tool.
5) Does not support properly the creation, capturing, use, maintenance, and documentation of architectural design decisions. Does not enable to store design decisions as first class entities under an iterative approach.
6) No catalogue of architecture and design tactics.
7) Does not capture and present scenarios (general and concrete).
8) Does not store architectural documents.
9) Does not support for design and analysis methods.
10) Does not check implementation against architectural decisions. Does not get consequences of an architectural decision.
11) Does not check for consistency and for superfluous decisions.
12) Not useful for software product families.

13) Does not differentiate the functional requirements and non functional requirements.

14) Does not provide any security features and performance-wise not good. Need to improve the speed and accuracy of knowledge retrieval.

CHAPTER 4

VALUE-BASED SOFTWARE ARCHITECTURE KNOWLEDGE MANAGEMENT TOOL

4. VALUE-BASED SOFTWARE ARCHITECTURE KNOWLEDGE MANAGEMENT TOOL

Architecture Knowledge Management [21] is very important for improving an organization's architectural capabilities. Value-Based Software Architecture Knowledge Management is an emerging trend in software architecture community. By managing architectural knowledge in a value-based manner is one of the most valuable steps for advancing the software architecture state of the art by preventing its high costs of change and diminishes the effort and time required [28,29]. This diminished effort has the effect of mitigating the overhead. Value-Based approach takes into account value considerations of stakeholders and only documents the information required by stakeholders. Different techniques have been studied and analyzed for managing, sharing and storing architectural knowledge in Chapter 2. As a result of techniques survey, a Value-Based approach for Documenting Design Decisions Rationale (VB-DDRD)[27,28,29] has been found as a useful technique as it focuses on documenting only the set of required information based on its purpose. Moreover, existing tools for managing and sharing Architectural Knowledge have been studied and evaluated in Chapter 3.These tools are evaluated on the basis of certain attributes. It has been found that none of the surveyed tools supports Value-Based Software Engineering concepts. Therefore, an open source tool was selected and has been used to develop a Value-Based Software Architecture Knowledge Management tool. The selected tool is Architecture Design Decision Support System (ADDSS)[8,9], which covers additional features relative to other tools as studied during survey. Certain other limitations have been found during the survey and evaluation of tools as discussed in chapter 3. Special features have been added to overcome these limitations.

This chapter involves the contribution of this thesis i.e. application of Value-Based Software Engineering principles [26] into ADDSS [8,9] tool along with the implementation of certain features which are missing from the tool. This chapter also describes the evaluation of Value-Based Software Architecture Knowledge Management tool i.e. the modified tool on the basis of different attributes. Certain features have been

added in this tool which is listed down below along with their benefits of adding in the tool.

4.1 Features

These features are listed down one by one:

4.1.1 Support of Value-Based Software Engineering Principles

Because architectures have high costs for change and may erode during the evolution of the system [11], architectural design decisions should be captured and documented to avoid knowledge vaporization. Hence, in order to prevent the erosion of software design and knowledge vaporization, we need to capture these decisions and their underlying reasons that led to any particular architecture. As Architectural Knowledge consists of architectural design as well as design decisions, and their underlying reasons that led to any particular architecture. So this means we need to manage and store architectural knowledge.

There are different ways to manage and store architecture knowledge. Different studies show the importance of documenting and managing design decisions along with their rationales [4,7,52]. Recently, various researchers [8,9,23,27] have proposed different tools and techniques to capture architectural knowledge. However the applicability of this work for managing AK in software engineering activities can be inhibited by certain factors [27,28, 29]. One of the main inhibitor for recording design decision and its rationales is that it takes a lot of time to record all the information about design decisions and its rationales. Nowadays during development or when the deadline is near no one has the time to enter all the information about architecture knowledge. That's why design decisions and their rationales are usually not properly documented. Following are the inhibitors for recording all the information about design decisions also mentioned in [27, 28, 29] are:

1. **Critical timing:** The period in which design decisions are taken is usually critical for the success of the software project. The project deadline or project pressure is one of the main reasons for not documenting this design decisions and its rationale.

10. **Potential inconsistencies:** Architectural knowledge documentation implicitly represents the results of the design. If Architectural Knowledge documentation are not well updated, potential inconsistencies in case of decision changes might occur.

These and other inhibitors may hamper capturing, using, and documenting the design decisions and its rationale.A lot of architectural knowledge is there and we have to document and maintain all the architectural knowledge.But the benefit of managing all architectural knowledge is not clear. It is not clear what information is required to save that will benefit whom. If benefits are not understandable whilst one has to manage all the architectural knowledge, the already defined inhibitors like critical timing, time and effort required and overhead will have an impact. It will take more time and effort which also increases the overhead in order to manage AK. From this it is concluded that not all the information is needed all the time as different people need different information. So there is a need to decide what information is required to save that will benefit whom. For this we have applied Value-Based Software Engineering principles on an architecturL knowledge management activity in order to mitigate the effect of above-mentioned inhibitors which also helps to get to know the benefit of managing AK. The idea has been taken from Boehm's work [26], who proposed a Value-Based Software Engineering (VBSE) agenda and from Davide Falessi's work [27,28,29] who used Boehm's idea for documenting design decisions rationale i.e. used a Value-Based approach to DDRD (VB DDRD).

This thesis applies the principles of Value-Based Software Engineering (VBSE) into an open-source tool; Architecture Design Decision Support System (ADDSS)[8,9] which was available at [36].Basically here architectural knowledge documentation has been tailored. The adoption of tailored architectural knowledge documentation, consisting only of the required set of information, would mitigate the effects of above mentioned inhibitors. ADDSS and other studied tools don't support the concept of value based software engineering. This is the main feature which is incorporated in ADDSS.

The basic idea of this approach is that all the information included in a documenting design decisions might be useful but sometimes some information are mere optional. The idea to prioritize attributes for the AK is similar to the use of mandatory and optional attributes for design decisions as in ADDSS [9]; it differs in the fact that the proposed

Value-Based Software Architecture Knowledge Management tool is focused on the choice of all the persons (beneficiary stakeholders) who are involved in specific project and will have a choice to get only that type of information which they required. In ADDSS mandatory and optional attributes are fixed, we cannot change any of the mandatory attribute to optional and optional to mandatory attribute. The researchers of ADDSS are making distinction between the mandatory and optional attributes by themselves. However, in this work, all the related stakeholders have the choice for selecting required set of design decisions information along with the architect. Besides moving towards working and applicability of this concept into ADDSS; we first need to understand the concept of value based software engineering.

4.1.1.1 VALUE-BASED SOFTWARE ENGINEERING

Most software engineering activities are practiced in a value neutral approach in which every fault, user requirement, test case, use case, risk etc. is treated equally [26]. The Standish Group CHAOS report [40] describes that value-oriented shortfalls like lack of user input, changing requirements, lack of resources and unrealistic time frames etc, are the common causes of most software project failures. A value-based software engineering (VBSE) agenda has emerged. The focus is to integrate value considerations into current and emerging software engineering principles and practices e.g. value-based requirements engineering, architecting & design etc, and to develop an overall framework in which they compatibly reinforce each other [26].

Basically, value based software engineering is an extension in traditional software engineering, as it tries to introduce value considerations into previously defined software engineering concepts and practices.In traditional software engineering (SE) the whole development process focuses mainly on successful development of the final product with lesser attention to the fulfillment of the values of stakeholders.On the otherhand,in VBSE the focus is taken(or atleast tried to be taken) beyond just the development of the software product.Here the main focus is on the value that the software has added/will be adding to the system.The traditional software engineering approach considers only the production/development whereas value-based approach also considers the system in which that software will be implemented. In this present work, a Value-Based approach has been proposed to Architecture Design Decision Support System[8,9], which focuses

on documenting only the set of required information based on the choice of different stakeholders.

4.1.1.2 APPLYING VALUE-BASED SOFTWARE ENGINEERING TO ADDSS TOOL

This section discusses how the Value-Based Software Engineering principles are applied on ADDSS. A process has been developed to manage architectural knowledge named as Value-Based Software Architecture Knowledge Management (VB-SAKM). Five steps are involved in the process which are as follows:

i. Identify success critical stakeholders.
ii. Elicit stakeholder preferences.
iii. Prioritize stakeholder preferences.
iv. Record design decisions information.
v. View of recorded design decisions information.

Process detail along with the working of tool is discussed as follows:

i. Identify success critical stakeholders

There are different stakeholders who are associated with each project. In software domain, stakeholder can be anyone who can affect or get affected by the system in any means (financially, personally etc). Basically, stakeholder is a general term that represents everyone having a stake in system e.g., developer, project manager, consumer or customer etc. For the successful completion of any project, it is important to bring-in all the Success-Critical stakeholders (SCSs).Moreover, every stakeholder doesn't need to store or use all the design decisions information. Therefore, it is important first to identify the success-critical stakeholders (SCSs) i.e. Who will get profit (beneficiary stakeholders) [27,28,29]. So the first step of this process is to identify success-critical stakeholders (SCSs). Architect can only identify the SCSs. Input of first step is a list of stakeholders involved in a specific project and the output is a list of the identified success-critical stakeholders. Value-based approach is aimed at making SCSs the winners and to ensure stakeholder satisfaction besides just focusing the successful product development [26,49].

To identify success-critical stakeholders (SCSs), tool is not concern about the technique the architect adopts. The modified tool is independent of any technique for identifying success critical stakeholders. However, architect can identify success-critical stakeholders by the Dependency theory as mentioned by Boehm [49].A key technique is the Results

Chain [50]. The Results Chain is a valuable framework by which software project members can work with their clients to identify additional non-software initiatives that may be needed to realize the potential benefits enabled by the software/IT system initiative. These may also identify some additional success-critical stakeholders who need to be represented and "bought into" the shared vision. Success-critical stakeholders have goals. Results chain is used to see what software and other initiative are required to fulfill goals of SCSs. But it is not necessary to identify success-critical stakeholders with the above mentioned technique, architect can identify with any other technique as well.

Figure 4.1 shows how success-critical stakeholders are identified in this tool. The tool opens a form that contains the privileges of each stakeholder along with their category. By checking the Critical Stakeholder option in front of any user, we can identify a stakeholder as success-critical stakeholder.

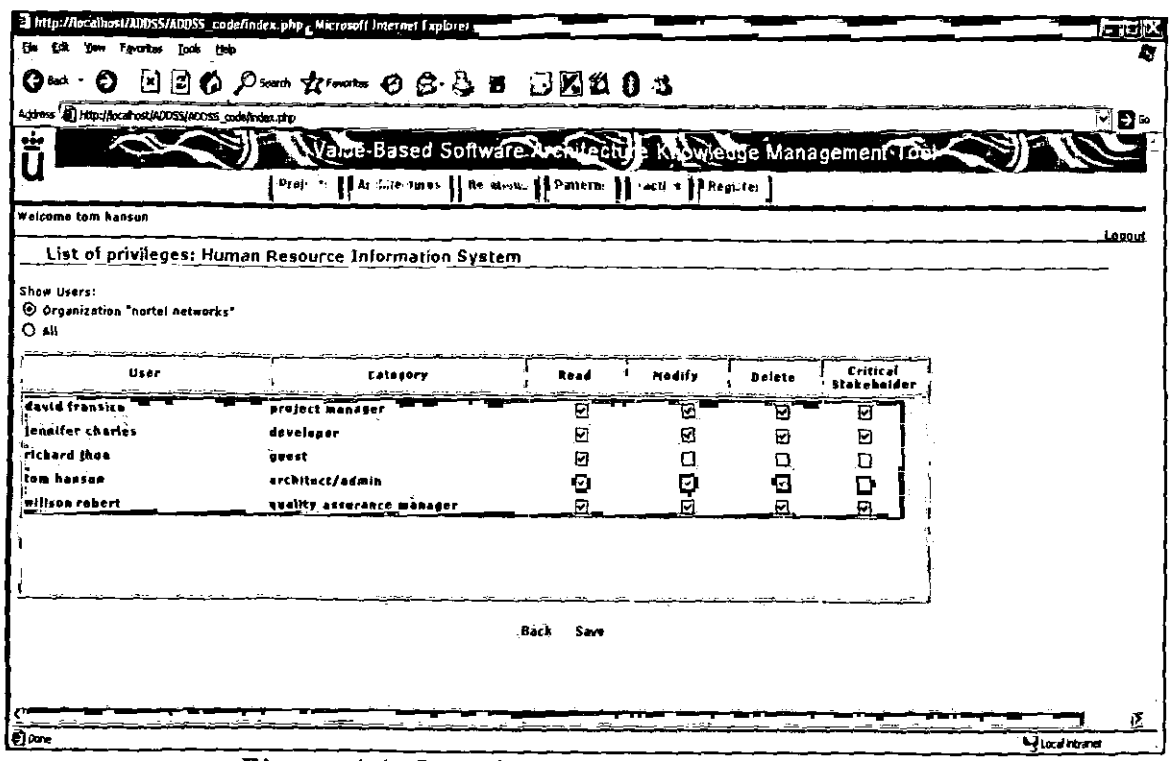


Figure 4.1: Identifying success critical stakeholders

ii. Elicit stakeholder preferences

The second step of VB-SAKM process involves the elicitation of stakeholder preferences. The identified success-critical stakeholders' gives preferences by providing value to each design decision attributes. According to the 'Theory of Value (economics)' [53,54], *"value is meant as economic worth of goods and services"* and it tries to explain the

worth of goods and services provided by some entity from different angles.This theory suggests that the value of some entity can be seen in different perspectives[54]. For example, it can be seen from intrinsic, subjective or objective angle. In this step, we are following Boehm's idea [26] for eliciting stakeholder preferences as the value of each design decision attribute purely depends on the stakeholders' perception and their choices.

Basically here the stakeholder preferences are elicited in the form of rating of each design decision attribute by giving score to each attribute. The input of second step is a list of identified success-critical stakeholders and output is stakeholders preferences for each design decision attributes. Stakeholder proposition value plays an important role for selecting required set of information.

For selecting only required set of information, the tool provides the facility to rate the design decision attributes as shown in Figure 4.2. The identified success-critical stakeholders can rate each design decision attribute by scoring each attribute from the scale 0-5.Five is the highest while zero is the lowest score. They can rate the attributes whatever information they think useful for them. SCSs can leave any field if he doesn't want to score that attribute, zero will be the default value for them.

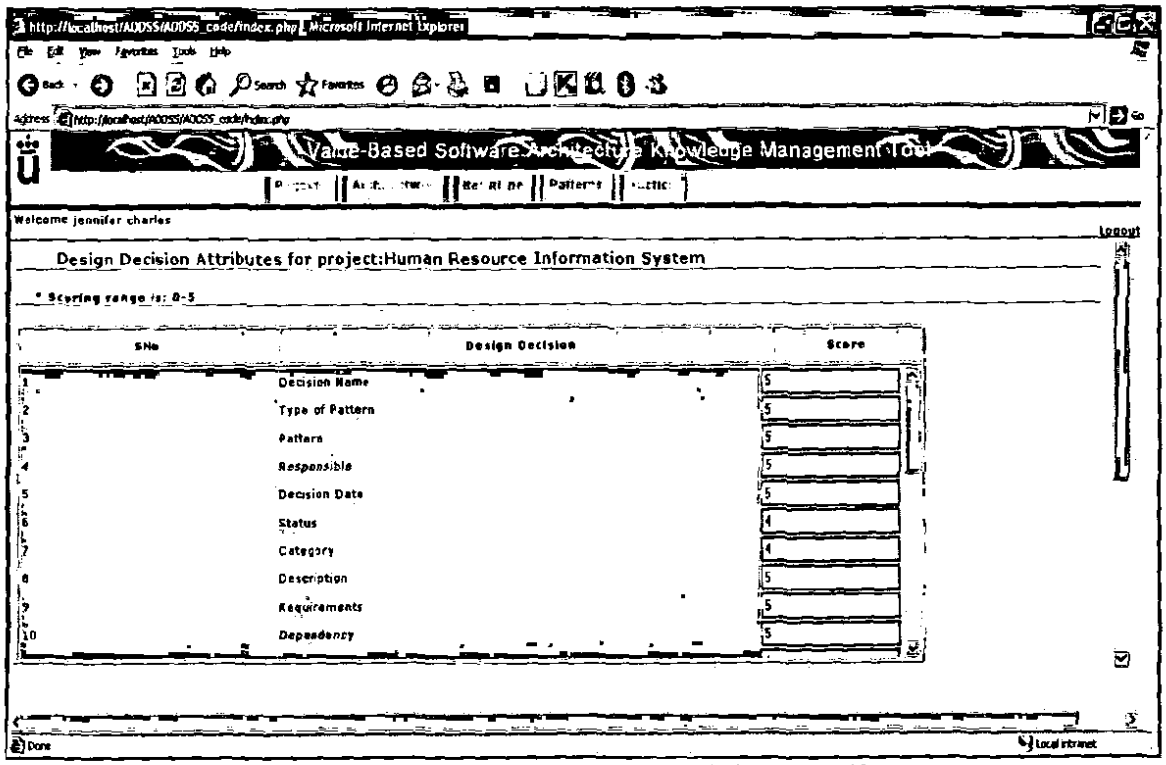


Figure 4.2: Rating of design decision attributes

A design decision (DD) has many attributes like rationale, alternatives etc.The attributes for design decision has been taken from different sources. The main focus is on the attribute which are used in [27,28,29]. Moreover some of the attribute has been considered which ADDSS tool has already been taken whilst others have been taken by reviewing all the other tools as studied above. These studied tools are considering these attributes for recording architecture knowledge. There are 29 attributes which tool is using. These attributes are dynamically added so one can add any attribute or can remove any of attribute from the list. The list of attributes which are using in this tool is as follows:

Table 4.1 List of Used Architectural Knowledge attributes

Attribute Names	
1. Decision Name	2. Type of Pattern
3. Pattern	4. Responsible
5. Decision Date	6. Status
7. Category	8. Description
9. Related Requirements	10. Decision Dependent
11. Issues	12. Assumptions
13. Constraints	14. Positions
15. Argument	16. Implications
17. Related Artifacts	18. Related Principles
19. Design Decision Rationales	20. Justification for the reasons behind design decisions
21. Other alternatives considered	22. Tradeoff evaluated
23. Argumentation that led to the decision	24. Notes
25. Pros/Cons	26. Alternative Decisions
27. Views	28. Tactics
29. Consequences	

iii. Prioritize stakeholder preferences

The elicited stakeholder preferences are prioritized here. Architect can only prioritize design decision attributes into required, useful or optional on the basis of total of each score associated with each attribute collected from different success-critical stakeholders.

For giving priority to required set of information, the following terms has been used as described in [28, 29]:

A. "Required information" refers to that kind of information without which the meaning of something cannot be understood to the readers
B. "Useful information", is a kind of information that helps to a small or large extent the readers to understand the meaning of something;
C. "Optional information" means information that is not required to understand something, but it can be useful.

Architect has all the scores associated with each design decision attribute from all success-critical stakeholders and total of each score associated with each attribute. Architect finalizes the prioritization of the DD attributes based on two factors, the total score associated with each attribute and the category of success-critical stakeholder. A mechanical process can be just to focus on total score for each attribute and treat all stakeholders equally important. But this can be unrealistic in a scenario where overall total score is in lower range and some important SCS e.g. project manager gives high score to an attribute. Architect is given discretion to prioritize the Attributes. In that case, he can prioritize the attribute as useful, no matters other stakeholders don't give preference to that attribute. Other stakeholders can neglect the useful attribute while recording design decision, if they don't require that. However, if architect does not prioritize that attribute as required or useful than it's an architect mistake. The input of this step is stakeholders preferences for each design decision attributes and output is prioritized stakeholders preferences for each design decision attributes.

The tool facilitates the process and the associated decision making by providing total scores for each attribute and individual stakeholder scores for attributes as well. Figure 4.3 shows a form that displays stakeholder's name, attributes and score associated with them, total of each score and priority choices in different colors. Architect has only the privilege to prioritize design decision attributes. Identified success critical stakeholders don't have the privileges to select design decision attributes. Architect must have to see the total score of each attribute and the category of success-critical stakeholder while prioritizing the design decision attributes. If there is no score provided for any attribute, architect can mark the attribute on his own choice. If the total score is in higher level then

it should be mark as "required". If the total score is greater then lowest level but not come under some higher level then it can be mark as useful.

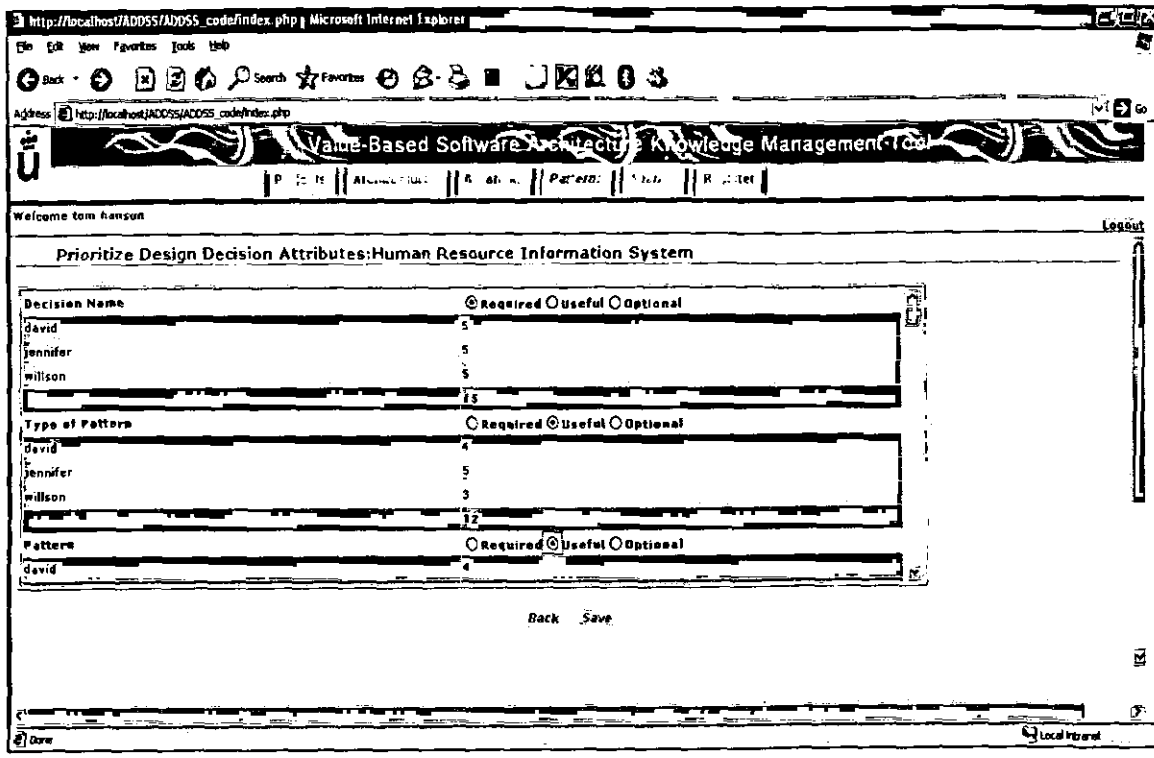


Figure 4.3: Selecting design decisions attributes

iv. Recording design decisions information

Architect or Success-Critical stakeholder can store only the required set of information after the design decision attributes have been prioritized, no need to store that information which is not required at that time. The input is the prioritized design decision attributes and output is stored required set of design decision information.

For recording design decision information a dynamic form is generated that contains only those fields which are marked as required or useful by architect as shown in Figure 4.4. 'Optional' fields are not visible in the form. 'Required' fields should be entered other wise the tool doesn't allow to save the information. For 'Useful' fields it is a user's choice either to fill them or not. The tool has provided a proper template like PAKME [23] to capture, manage and present architectural knowledge.

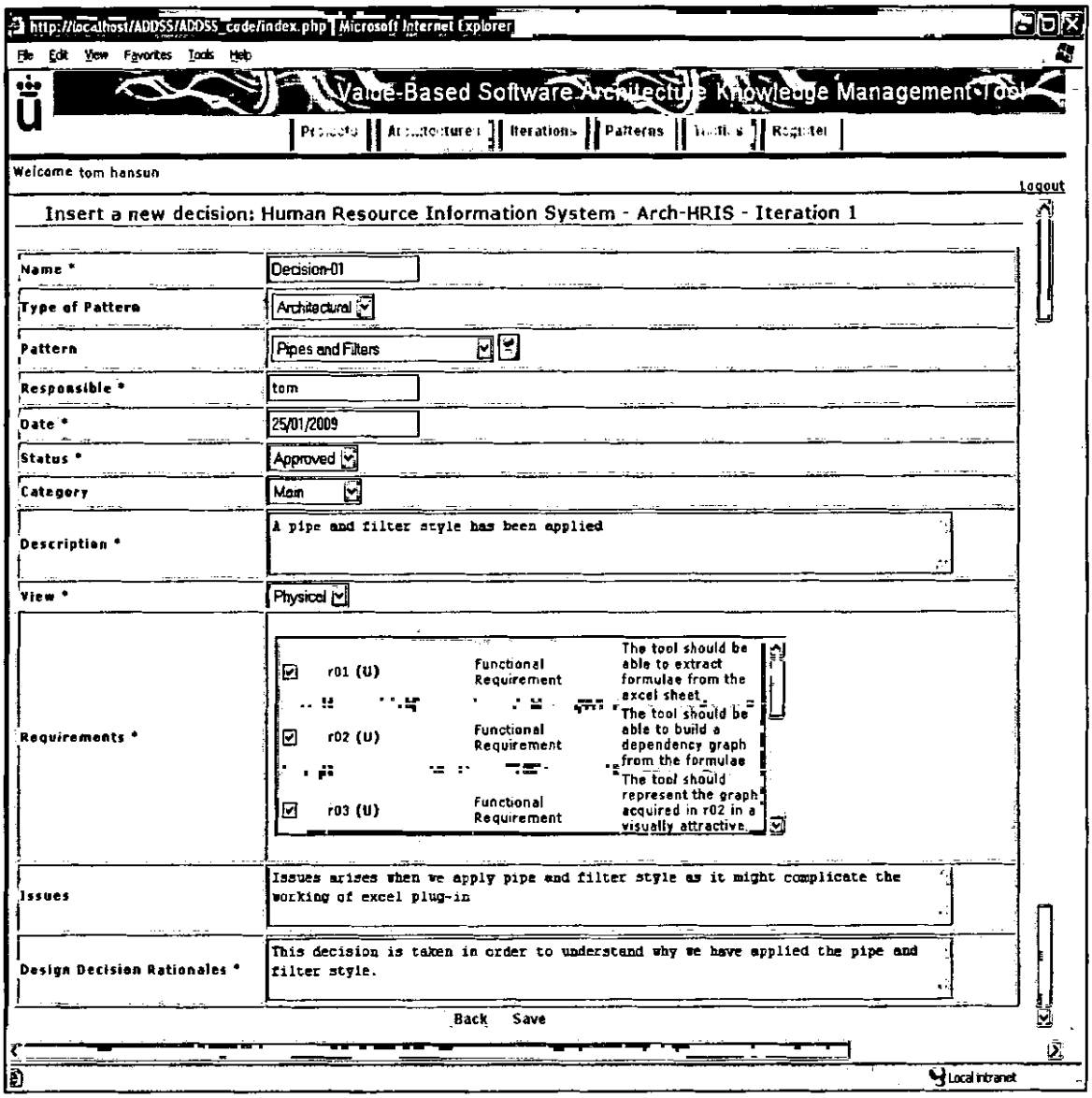


Figure 4.4: Recording of design decisions

v. View of recorded design decisions information

Architect or Success-Critical stakeholder has the choice to view any design decision information which is already stored. The input of this step is the recorded design decision information and output is view of recorded design decision information. User can choose from the list of fields which are prioritized as required and useful. One can view the entire field's information or can view any of them. Tool provides the choice of those fields as shown below.

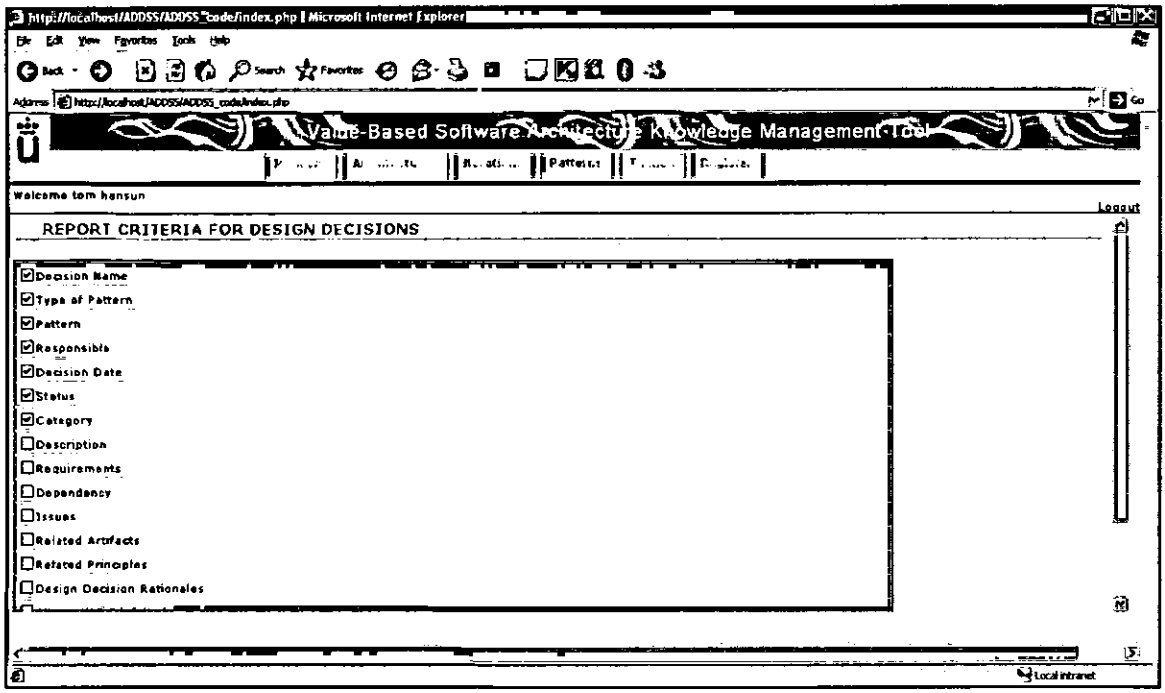


Figure 4.5:Report criteria for design decisions

After selecting any field of user's choice, a dynamically report is generated which shows the set of required information. Basically web based report is automatically generated for describing decisions. The more fields the user selects, the more fields will be displayed horizontally in the report. Figure 4.6 shows the report.

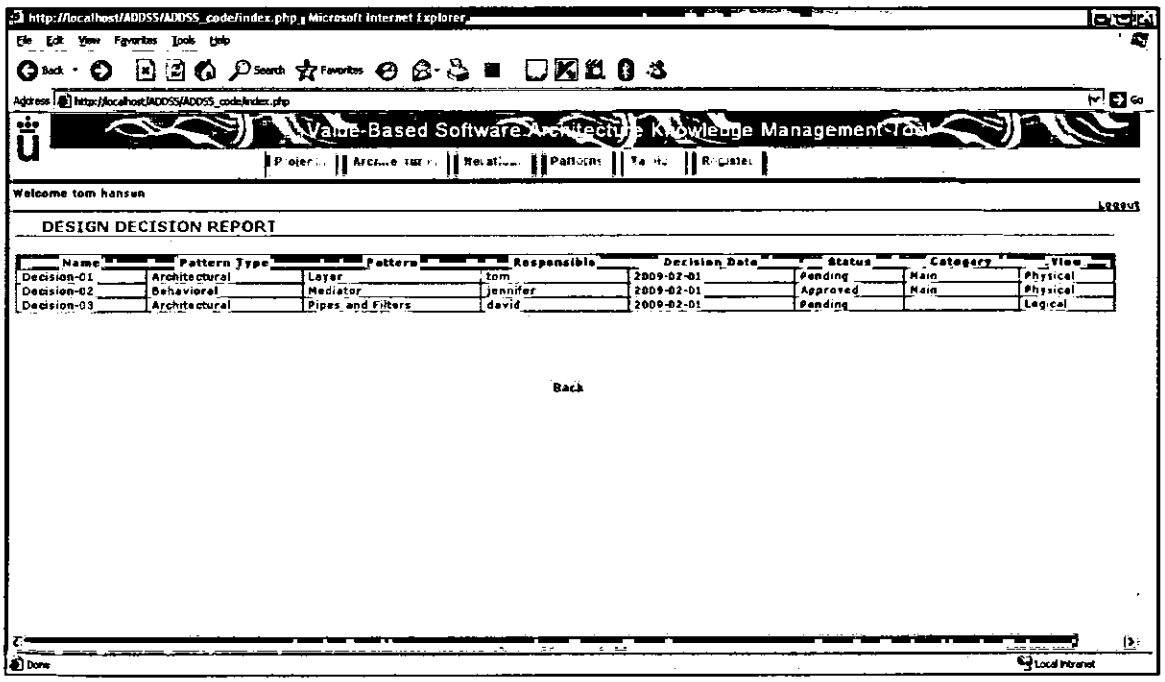


Figure 4.6: Design decision report

The above mentioned Value-Based Software Architecture Knowledge Management (VB-SAKM) process shows how to make architecture knowledge Value based. Moreover, this above discussion shows how the tool works by incorporating this process into the ADDSS tool. In this way we are determining a priority, who will profit from what information in which amount later on, in order to cope with the additional effort that has to be spend on recording design decisions and its rationale.

4.1.1.3 Benefits of applying Value-Based Software Engineering

Following are the benefits of applying this approach into ADDSS are:

1) **Less Time:** By recording only the required set of information for any architecture saves a lot of time as related persons are also performing other activities. The possibility to spend less time to produce the architecture knowledge documentation highly increases the possibility that people, who are busy to meet their projects deadlines, find enough time to develop such documentation.
2) **Less Effort:** By recording only the set of required information for any architecture, implies less information to document and maintain; hence less effort is required.
3) **Overhead:** A tailored architectural knowledge implies less information to document and maintain; hence, a diminished effort has the effect of mitigating the overhead.
4) **Lack of motivation:** The clear definition of who will profit from who allows the existence of a role (performed by real person or virtually) in charge of controlling that the specific producers provide, and the relate consumers use, the expected AK documentation.
5) **Delayed Benefit:** By recording only the required and useful information for any architecture according to the choice of different persons will not only help them for that architecture or project but also helps them in future projects. This will also helps building the organizational capabilities.
6) **Helpful for other persons:** By recording only the required and useful information for any architecture of some specific project can be helpful for the newcomers, who will associate with the same project or with any other project
7) **Potential inconsistencies:** The tailored architecture knowledge implies less information and hence less documentation. Less documentation implies both less required effort for architecture knowledge maintenance and less probability of

inconsistencies occurrence. In this way we can make knowledge capture cost-effective.

8) **Information unpredictability:** As the fields have been chosen on the basis of score given by different stakeholders so the producer can easily estimate what the consumer wants.
9) **Maturity:** As the literature survey shows ADDSS has more features as compared to other tools; this tool now covers VBSE principles so it is now quite mature and valuable tool.
10) **No conflict between stakeholders**: As for choosing the design decision attributes, right of choice is given to all critical stakeholders who are associated with that project, so there is no chance of conflict between them.
11) **Tells what type of architectural knowledge is useful:** As we know this support tells what type of architectural knowledge is required, useful or optional on the basis of scores given by all the stakeholders i.e. we are marking attributes as required, useful or optional. From this we get to know what type of architectural knowledge is useful.

4.1.2 Provide catalogue of architecture and design tactics

A set of templates have been designed to document different units of architecturally significant information (i.e. general scenarios, quality attributes, tactics) as an artifact of architecture knowledge. Figure 4.7 presents one of these templates. The template used to capture architecture and design tactics.

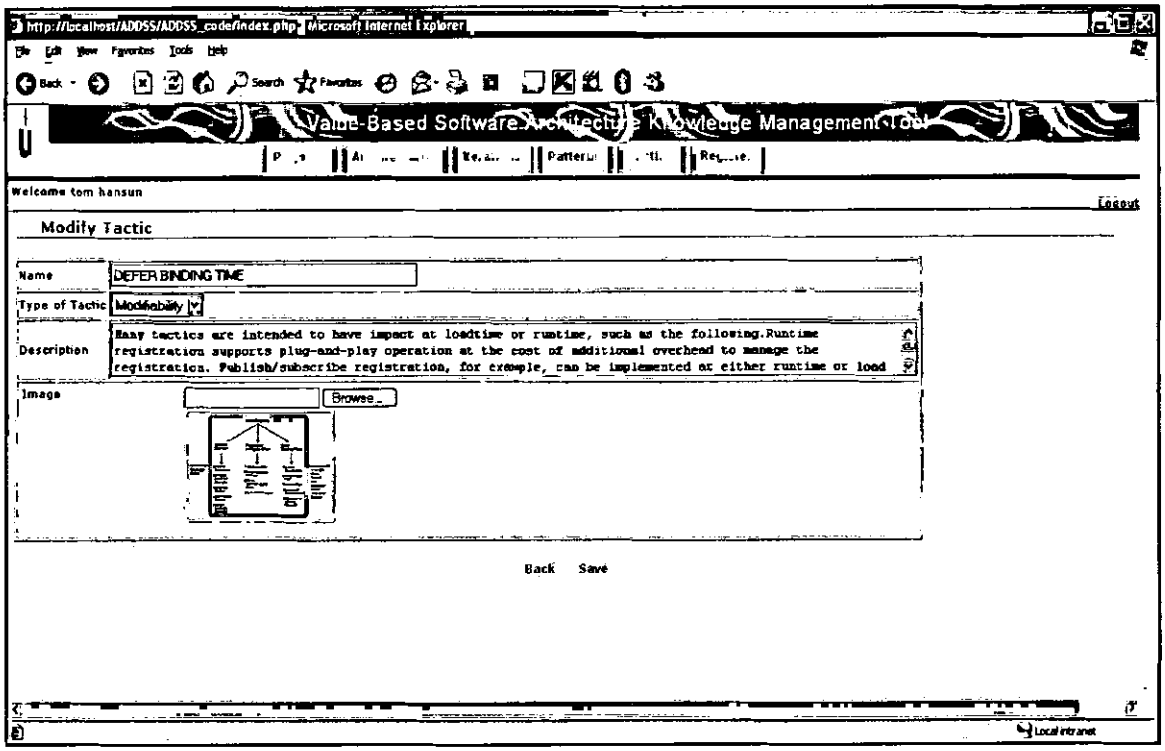


Figure 4.7: Template for recording tactics

A catalogue of architecture and design tactics has been provided to users as shown in Figure 4.8. Architect or stakeholders can select any tactic from the catalogue while recording design decision information.

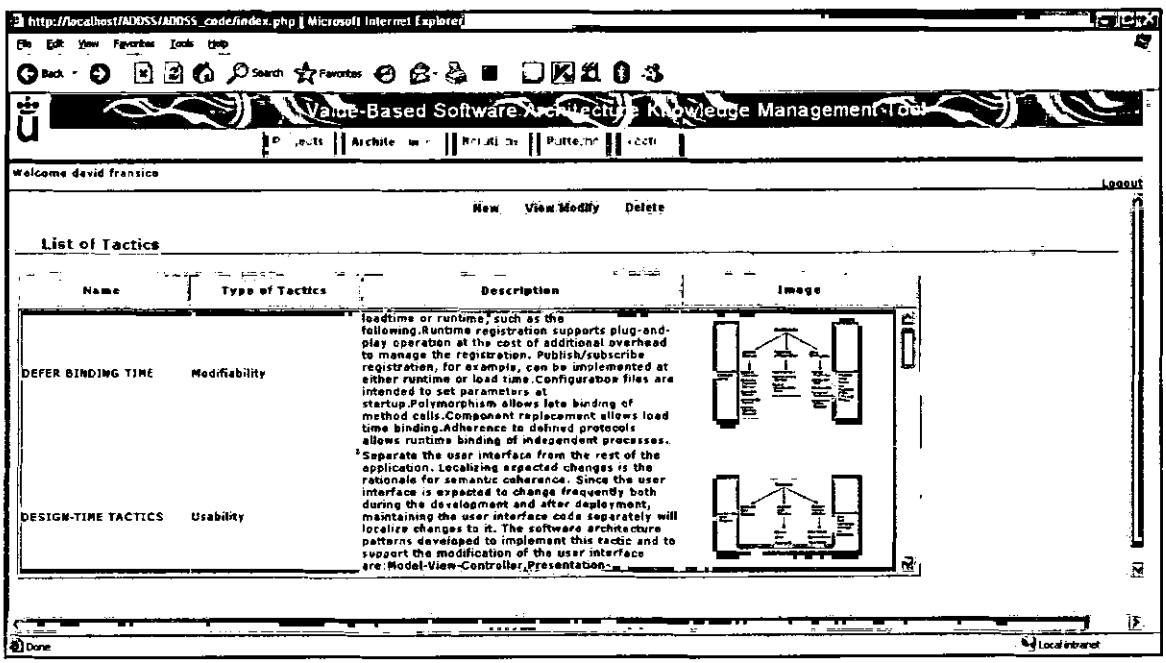


Figure 4.8: Catalogue of tactics

An architecture tactic is a transformation of the system from one state to other that affects one of the parameters defined by quality attributes [1]. A large number of tactics have been identified and catalogued in Bass et al [1,41]. The tactics are based on the quality

attribute addressed. Annotating the architecture documents with architectural tactics used while making architectural decisions helps to answer queries such as 1) did we use these tactics before and what was the result?

4.1.2.1 BENEFITS:

1) **Helpful in recording design decision information:** It is helpful for the users to select the related tactics while recording the design decision information from the catalogue as sometimes users doesn't know about tactics name. Also it answers did we use these tactics before and what was the result?
2) **Time Saving:** Selecting tactics from the catalogue saves a lot of time of each user as he doesn't have time to enter tactics and their description.
3) **Helpful for stakeholders:** Different stakeholders who are not related to this field, this catalogue helps them a lot as they can read and understand from this catalogue instead of reading from the book.

4.1.3 Differentiates the functional requirements and non functional requirements

Requirements play an important role for any architecture. Requirements are already taken in this tool but they are not differentiating the requirements by type i.e. either these requirements are functional or non-functional or business requirements. The existing form has been modified and now the requirements are recorded according to the type as shown in below Figure.

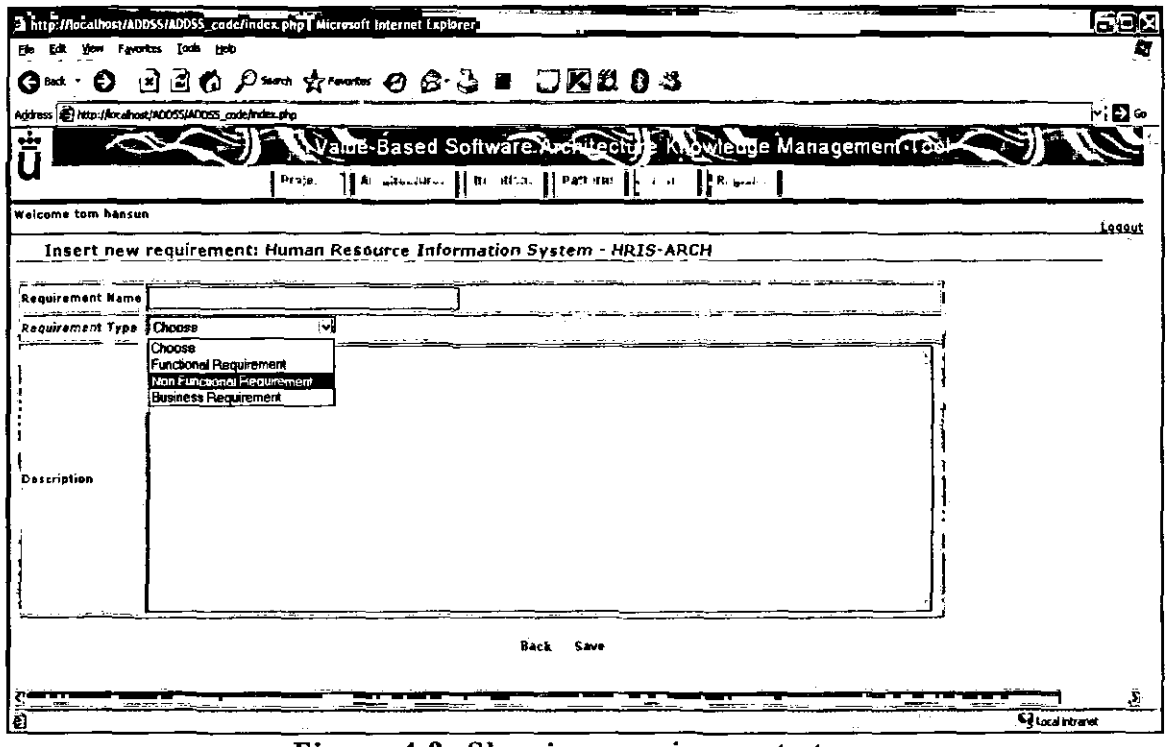


Figure 4.9: Showing requirements type

Taking requirements according to the type will be helpful while recording the design decision information as quality attributes should be taken separately so that we must know what quality attributes are affecting the design decision or have impact on them. Quality requirements are the architecture drivers for any successful development of the system. Also helpful while capturing scenario's as scenarios are used to characterize required quality attributes. Requirements along with their types helps the related persons understand the specific architecture.

4.1.4 Capture and present scenarios (general and concrete)

Figure 4.10 shows a form for capturing a general or concrete scenario, which can be elicited from a stakeholder or extracted from a pattern. Each scenario can have several attributes attached to it including scenario name, source type, quality attribute etc. Tool's repository can contains hundreds of general or concrete scenarios.

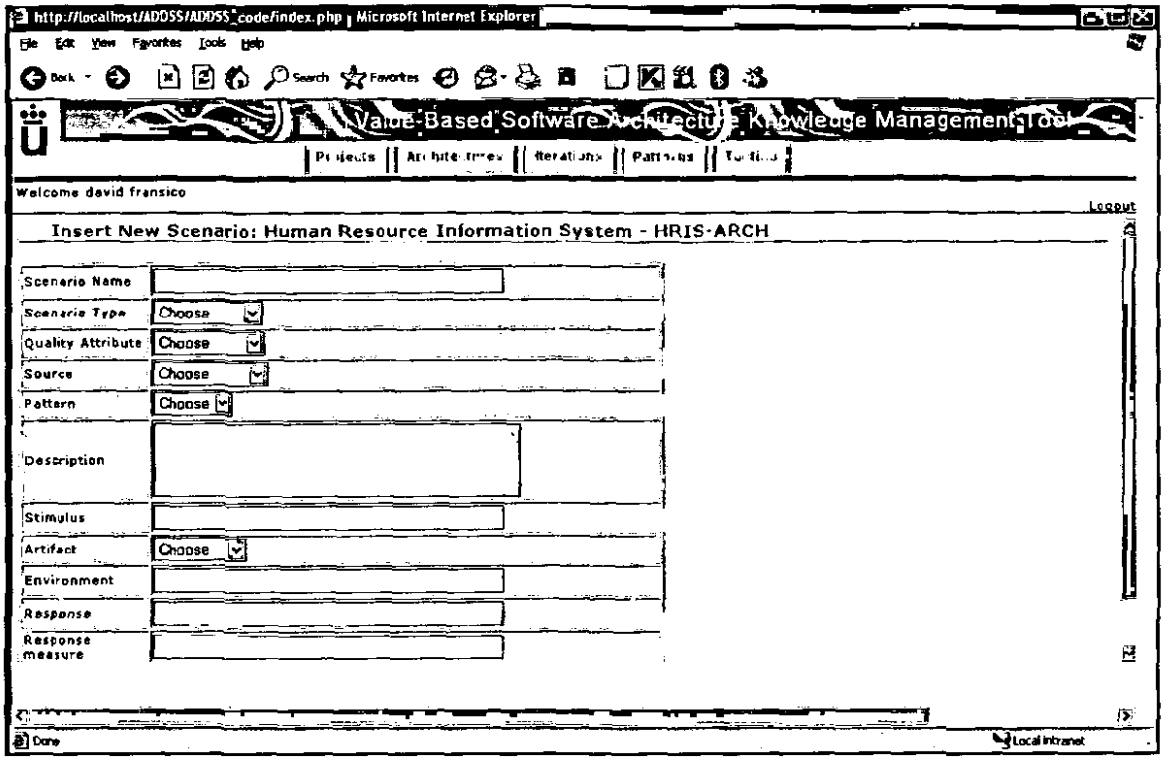


Figure 4.10: Capturing scenarios

Capturing scenarios helps the users to understand the requirements easily. These are an artifact to architecture knowledge. Scenarios are helpful in characterizing required quality attributes. The use of quality attribute scenarios is one of the core techniques for SEI's methods to characterize stakeholders' concerns. Tool can provide several hundred general scenarios, which can be concretized to specify quality attributes for a given system as implemented in [23].

4.1.5 Captures principles

A template is designed to capture principles for each architecture. Figure 4.11 shows the form that captures principles that guide decisions for an architecture. These principles are enterprise principles (business or others). Tool displays a catalogue of principles that creates a link between architecture design decisions and enterprise principles.

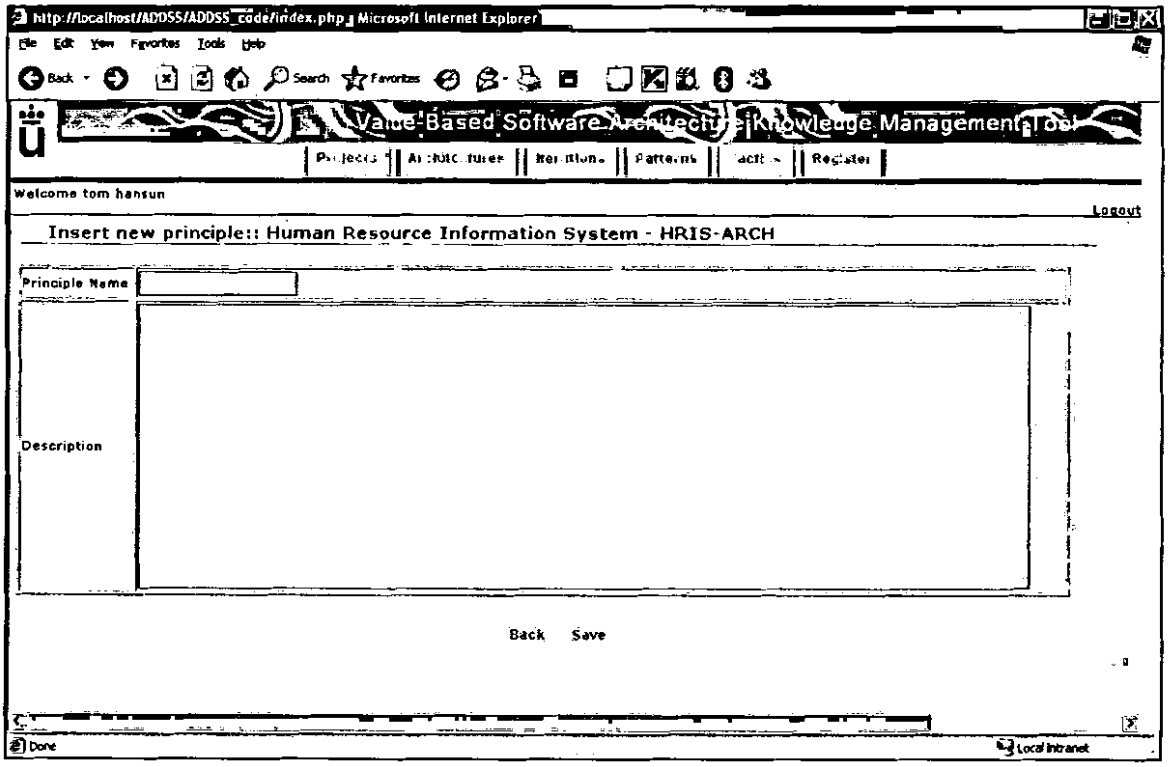


Figure 4.11: Capturing principles

Principles are captured which helps recording the design decisions as these principles helps guiding the design decisions i.e. it develops a link between the design decisions and these principles. One can easily select related principles from the catalogue of principles while recording design decision information.

4.1.6 Captures Artifacts

A template is designed to capture artifacts for architecture of a specific project. Figure 4.12 shows the form that captures artifacts that have an impact on architecture design decisions. Tool displays a list of catalogue of artifacts.

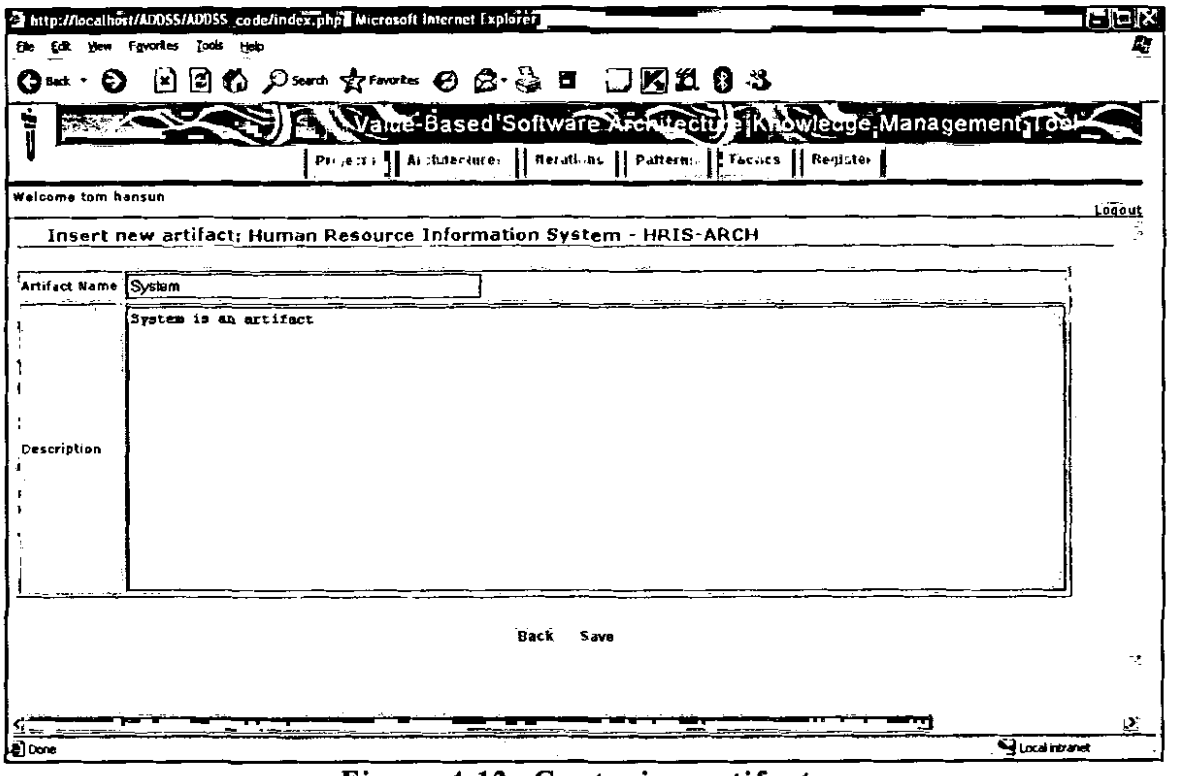


Figure 4.12: Capturing artifacts

Artifacts are stored which helps recording the design decisions as these artifacts have an impact on the design decisions. One can easily select related artifact from the catalogue while recording design decision information.

4.1.7 Captures architecture patterns

ADDSS captures only design patterns. Modified version now also captures architecture patterns. Figure 4.13 shows a catalogue which now also contains architecture patterns.

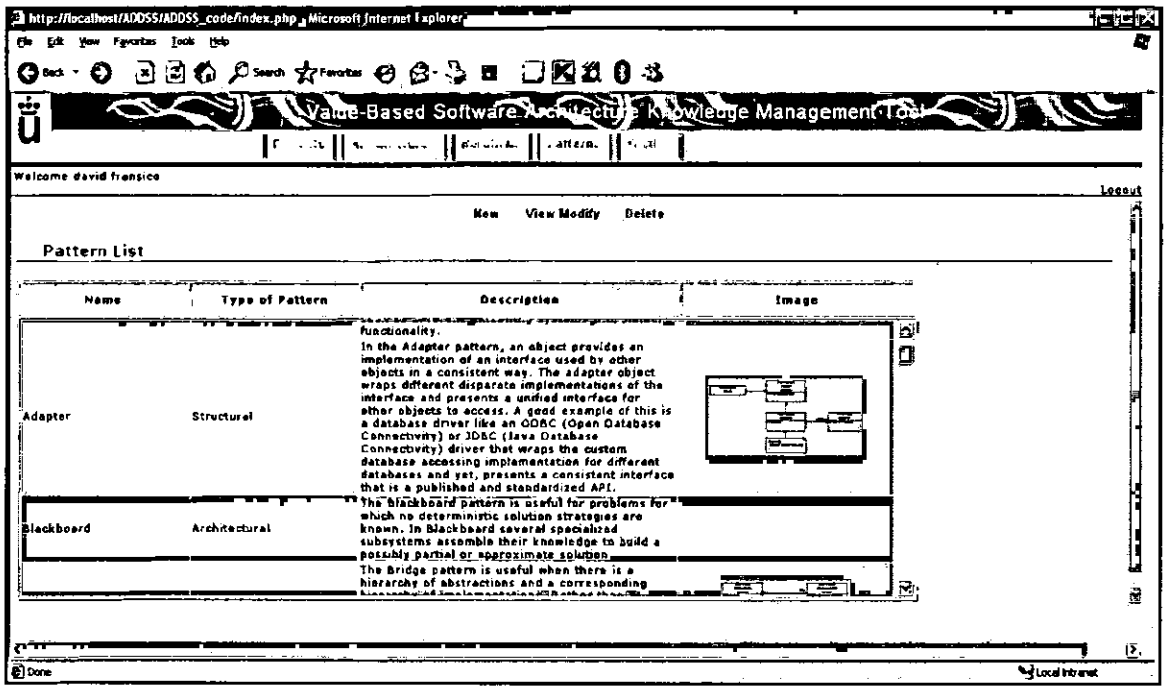


Figure 4.13: Catalogue of architectural patterns

Pattern helps in documenting design decisions. Similarly, architecture patterns also helpful in documenting design decisions. They are also helpful in capturing scenarios.

4.1.8 Multiple Views

Figure 4.14 shows multiple views associated with single architecture. ADDSS tool associates each architecture with single view however this tool now provides multiple views associated with single architecture.

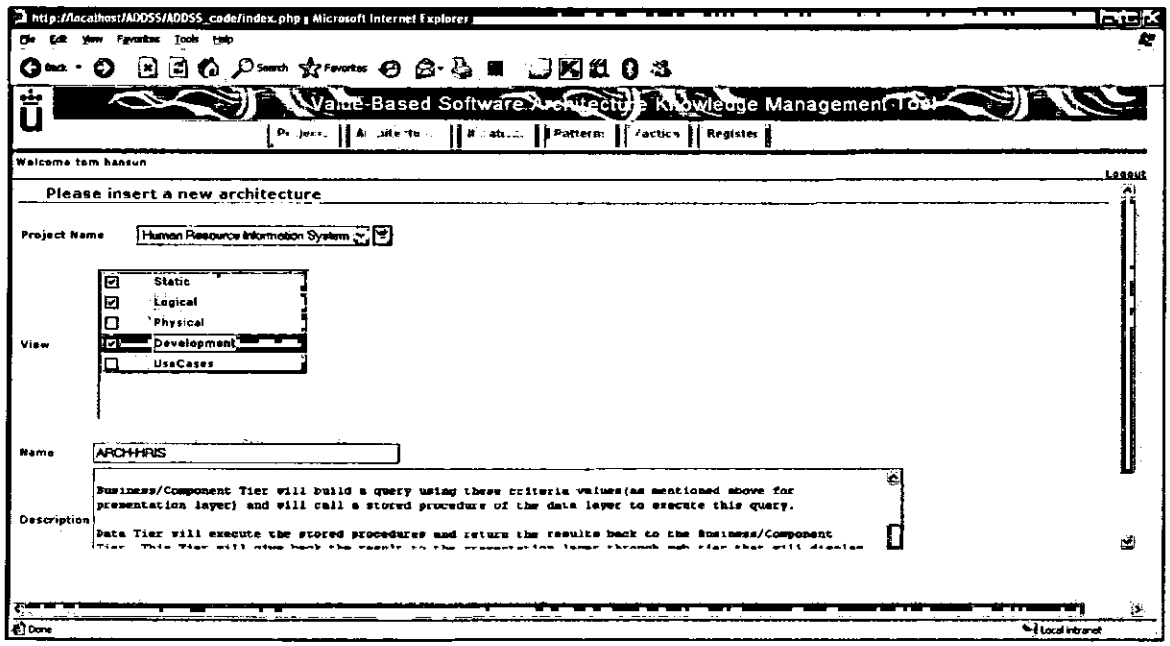


Figure 4.14: Recording multiple views with single architecture

Multiple views with each single architecture helps recording design decisions information as design decisions have an impact on these views.

4.1.9 Categorizes risk and non risks for decisions

Figure 4.15 shows the risks associated with design decisions. It also shows the categorization of risks and non risks.

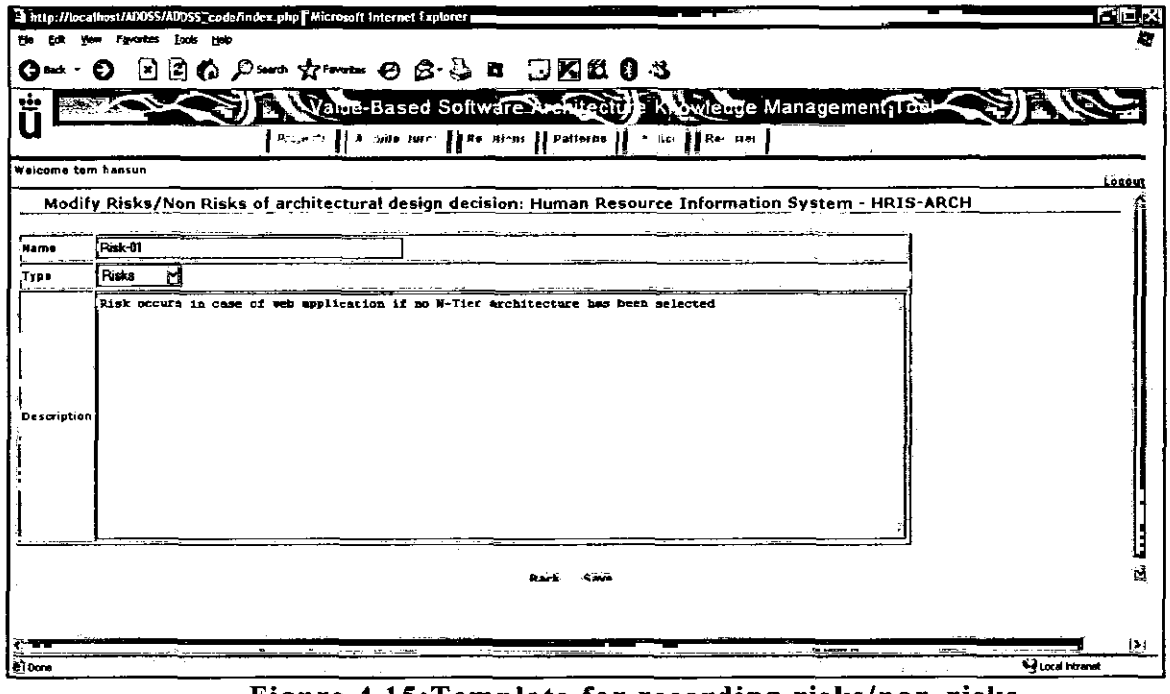


Figure 4.15:Template for recording risks/non-risks

With this we can get to know which decision is better to take. We can rank the design decisions as well.

4.1.10Other Features

There are some small features which are also added, these are:

Tool warns and prohibits violations of the decisions on which other decisions are dependent. Figure 4.16 shows how it warns the users whenever he wants to delete that decision.

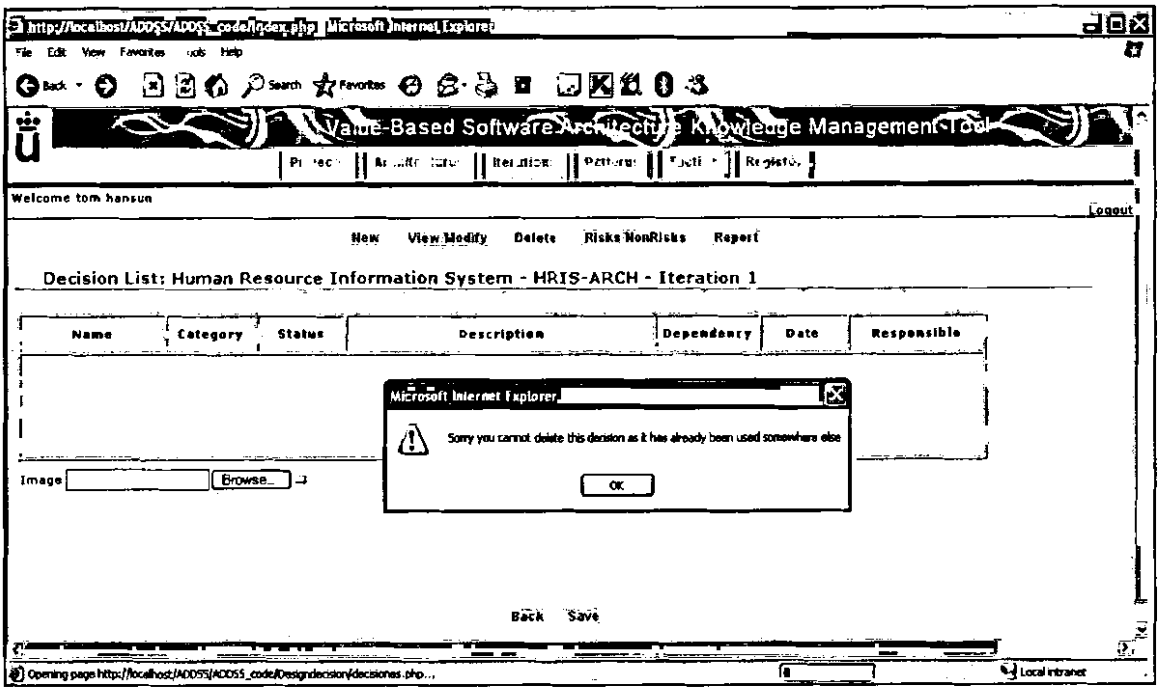


Figure 4.16: Warning from the violations of the decisions

By this check one doesn't lost data if some decision is depending on it.

All checks are provided for required and useful fields. Tool warns if the required fields are not filled. Numerous checks also implemented to ensure consistency. Tool is also getting consequences of design decisions as taken in Archuim [17].

4.2 Evaluation of Value-Based Software Architecture Knowledge Management tool

This section describes the evaluation of currently modified tool i.e. Value-Based Software Architecture Knowledge Management (VB-SAKM) tool. VB-SAKM tool has been evaluated on the basis of certain attributes like usability, performance etc. These attributes are the same as mentioned and used in Chapter 3 for evaluating the surveyed tools. Description about each attribute has also been mentioned in Chapter 3.By evaluating the modified tool personally and from the literature ,we can easily find either this tool provides better support of discussed attributes as compared to other surveyed tools or not. As discussed in the thesis that the developed tool is the modified form of existing tool ADDSS, so the modified tool also supports evaluation criteria attributes which is already supported by ADDSS. In the last of this section, a table is described which shows the evaluation of Value-Based Software Architecture Knowledge Management tool in

4. **Coverage:**

   Table 3.2 shows ADDSS covers many features which other tools are not covering. VB-SAKM is the modified form of ADDSS. So this covers all the features of ADDSS along with the newly implemented features. The modified tool is a value-based tool. It supports the Value-Based Software Engineering concept. This tool also covers certain other features which are missing from the tool i.e., catalogue of tactics, capturing of principles, artifacts, architectural pattern etc. The above discussion about this attribute shows that this tool covers more features as compared to all the other studied tools as none of the other tool is Value-Based. This attribute has been checked from the tool's thesis.

5. **Useful for software product families:**

   VB-SAKM tool is not useful for software product families as this feature has not yet being implemented into this tool. This attribute has been checked from the tool's thesis.

6. **Support of Value-based software engineering principles:**

   This is the main feature which is supported by Value-Based Software Architecture Knowledge Management (VB-SAKM) tool. Tool supports the Value-Based Software Architecture principles. It helps to document only the set of required information based on its purpose. Also focuses on the choice of all the stakeholders who are involved in specific project and will have a choice to get only that type of information which they required. With this tool, we are determining *a priori, who will profit from* what information in which amount later on, in order to cope with the additional effort that has to be spend on recording design decisions and its rationale. This attribute has been evaluated with the help of tool's thesis.

7. **Useful in evolution and maintenance activities:**

   Value-Based Software Architecture Knowledge Management tool supports modeling and documenting the evolution of ADD as it is the modified form of ADDSS and ADDSS supports this feature. This attribute have been checked from the tool's thesis.

8. ***Integrated with other modeling tools:***

   Modified tool is not integrated with other modeling tools, decisions can be stored in parallel at the same time the designers use modeling tools to depict the architecture. In future there is a plan to integrate this tool with other modeling tools. This attribute has been validated from the future work of tool as mentioned in tool's thesis and by personally using the tool.

9. **Accessible for geographically distributed stakeholders:**

   Value-Based Software Architecture Knowledge Management tool incorporates AKM features for geographically distributed stakeholders involved in the software architecture process. VB-SAKM tool provides the choice to all the stakeholders to document design decisions information. From the tool's thesis, this attribute has been validated.

10. **Performance:**

    Modified tool is good performance-wise as it provides proper template for capturing and sharing design decisions so with less time one can easily store and retrieve information from the well-defined templates. User with less time can complete the basic task. As the tool is value-based, this makes easy for a common user as he only stores and retrieves the relevant information. No need to confuse with irrelevant information. VB-SAKM tool takes 1-2 seconds to save information whenever we press the save button. Same time is consumed for deleting data. As only required information is saving so it saves a lot of time of user. Tool takes less time while retrieving knowledge. Whenever we gave command for viewing any information, system responses in 1-3 seconds. Time depends upon the choice of user's. As we can retrieve information about design decisions on the choice of user if he wants to retrieve information of all the design decision attributes, system responses in more time as compared to if we choose less number of design decision attributes. Tool has proper interface so its takes few seconds to start the tool. The performance of tool has been measured by personally using the studied tool.

11. **Security:**

    Tool can access through username and a password. Registered users have different permissions for accessing the information. Security attribute has been evaluated by personally using the tool.

Following table shows the evaluation of Value-Based Software Architecture Knowledge Management (VB-SAKM) tool in summarized form. For evaluating VB-SAKM tool, ratings are defined. ✓ shows that the attribute is fully supported by the tool, • shows that the tool partially supports that attribute and X shows the tool doesn't support that attribute.

Table 4.2 Evaluation Criteria of Value-Based Software Architecture Knowledge Management (VB-SAKM) tool

***Ratings: ✓: Fully supported, •: Partially supported, X: Unsupported**

Attributes	PAKME	ADDSS	ARCHUIM	KNOWLEDGE ARCHITECT WORD	AREL	VB-SAKM
1. Usability	✓	✓	X	•	X	✓
2. Industrially used	✓	X	X	•	•	X
3. Open Source	X	•	X	X	X	✓
4. Coverage**						
❖ Architectural Knowledge						
▪ Integrated representation of the software architecture	X	✓	✓	X	✓	✓
▪ Architecture design decisions	✓	✓	✓	✓	•	✓
▪ Rationales underlying the design decision	✓	✓	✓	✓	✓	✓
▪ External context/ environment	✓	•	•	•	X	•
❖ Features	•	•	X	X	X	✓
5. Useful for software product families	X	X	X	X	X	X
6. Support of value-based software engineering principles	X	X	X	X	X	✓
7. Useful in evolution and maintenance activities	✓	✓	•	•	•	✓
8. Not Integrated with other modeling tools	✓	✓	✓	X	X	✓

9. Accessible for geographically distributed stakeholders	✓	✓	X	X	X	✓
10. Performance	•	✓	X	•	X	✓
11. Security	X	✓	X	X	X	✓

** *Coverage attributes shows the coverage of architectural knowledge attributes as well as other features each tool possess. Features attribute and architectural knowledge attributes are evaluated on the basis of the features mentioned in the Chapter 4 as well as the features of ADDSS tool as described in table 3.2. In this way we can easily find which tool covers more features then others.*

This chapter discusses a Value-Based Software Architecture Knowledge Management process and its applicability on an open source tool. Value-Based Software Architecture Knowledge Management is recently recognized to be one of the most valuable trends in software architecture community. A value-based approach has been applied to Architecture Design Decision Support System (ADDSS). Modified form of ADDSS is known as Value-Based Software Architecture Knowledge Management (VB-SAKM) Tool. Different tools and techniques have been studied and reviewed for Architecture Knowledge Management. Certain limitations of tools have been found. These limitations have been incorporated in ADDSS tool. The main feature is to provide the support of Value-Based Software Engineering principles to ADDSS. Different benefits are already discussed above. Basically we are determining a priori, who will profit from what information in which amount later on, in order to cope with the additional effort that has to be spend on recording design decisions and its rationale. It is suggested that the use of such tailored architecture knowledge documentation would mitigate the effects of inhibitors as mentioned above and emphasize on the effects of its benefits. Besides this feature, some more features have also been implemented like the tool now provides a catalogue of tactics, and requirements are separately captured. Multiple views are associated with single architecture etc. The implementation of all the above mentioned features helps us in managing, sharing and storing architecture knowledge. By evaluating the modified tool, we found that Value-Based software architecture knowledge management (VB-SAKM) tool is a user-friendly tool. Moreover, it a value-based tool

which focuses on documenting the required set of information based on the choice of relevant stakeholder. Modified tool covers more features for architecture knowledge management as compared to other tools and it is good performance wise. From the above discussion it is concluded that, with this tool we can manage and store AK with little effort and in a short time.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5. CONCLUSION AND FUTURE WORK

In this chapter, the summary of this research has been explained alongwith the thesis contributions. Moreover, the research questions have also been answered. Enhancements that can be done in this work are also suggested

5.1 Summary

Software architectures have been considered as a set of interrelated components and connectors [1]. Research trends in software architecture focus on the treatment of architectural decisions as first-class entities and their clear representation in architectural documentation. From this point of view, a software system's architecture is no longer perceived as interacting components and connectors only, but also as a set of architectural decisions that convey the architectural principles underlying a particular design [7]. Within architectural analysis, architectural knowledge (AK) [2, 42] plays an important role. Architectural Knowledge consists of architectural design as well as design decisions, their assumptions and context. Design decisions and their underlying rationales are usually ignored at architectural level and during the development life cycle.

Proper management of architectural knowledge (AK) is essential in order to reduce high evolution and maintenance costs and to avoid architectural erosion. The quality of system and software architecture design can be highly dependent on the person who designs it. How architecture is designed depends on an architect's experience, knowledge and decision making abilities. As such, design decisions and its rationale directly affect the architecture design and its quality. By not properly managing, sharing and storing architecture knowledge, it affects architecture design in three ways: first, design decisions information might be incorrect or incomplete but there is no explicit information or documented information for its verification; second, once the system development has been completed, the architecture design can be costly and difficult to change at that stage if it is incorrect or not optimal; finally, it is sometimes difficult to understand the architecture design for maintenance purposes if the architecture knowledge is not documented.

Problem domain of this research is to manage, share and store Architecture Knowledge and the main focus is on tools and techniques for managing, sharing and storing Architecture Knowledge. There has been an increased demand for suitable techniques and tools that support organizations in documenting, sharing and managing architecture knowledge. The complex role of architectural decisions requires a systematic and partially automated approach that can explicitly document.

There are different ways for managing architectural knowledge [4,7,52].Researchers and practitioners have proposed various tools [8,17,23,33,35] and techniques [27,28,36,37,38] for its management. Indeed there are different tools and techniques for architecture knowledge management, practitioners do not like to apply them due to certain factors e.g, critical timing, extra effort and time required, overhead etc. A lot of architectural knowledge is there to document and maintain, but the benefit of managing all the architectural knowledge is not clear. Therefore, in order to mitigate the effect of above-mentioned factors and to understand the benefits of managing AK, there is a need to manage architectural knowledge in a value-based manner [27,28,29].

The research questions will be reiterated here along with the results from the analysis and the main work which is done that can be used to provide answers for them. In summary, the following research questions have been addressed in this thesis.

1. How to make architecture knowledge management tools and techniques practical?
2. How to reduce time and effort for managing and storing architectural knowledge?
3. How to reduce overhead for managing and storing architectural knowledge?
4. How to make knowledge capture cost-effective?

5.2 Contributions

As a result of addressing the above mentioned research questions, we have achieved the following.

Firstly, a literature survey has been performed to study existing techniques and tools for managing, sharing and storing Architecture Knowledge. Seven techniques have been studied and surveyed in chapter 2. Different techniques, method, framework, process

have been investigated for managing, sharing and storing architecture knowledge. From the survey of techniques, a useful technique/approach has been found named as Value-Based Design Decision Rationale Documentation which focuses on documenting only the required set of information based on its purpose.[27,28,29].Basically in that technique, Value-Based Software Engineering principles have been applied for documenting Design Decisions Rationale.This technique is better as compared to other techniques because it's the only technique which helps to mitigates the effects of mentioned inhibitors. Moreover, five existing tools have been analyzed and studied in chapter 3. The features of each tool have been compared with others and there are certain features which one tool is covering but not the other one. Tools are evaluated on the basis of certain attributes like usability, open source etc. By comparing and evaluating tools, certain limitations and drawbacks have been found in all the tools. Limitations like no tool is supporting for Value-Based software engineering principles, no support for software product families etc are found. In this work, some of the features which are missing from the tool have been implemented.

Secondly, special features have been implemented to the existing tool i.e. Architecture Design Decision Support System(ADDSS) in order to overcome the limitations and drawbacks found from the literature survey. The reason for selecting ADDSS tool for further enhancements is that this tool is only an open-source tool and it covers additional features relative to other tools. Value-Based Software Architecture Knowledge Management Tool is the modified form of Architecture Design Decision Support System (ADDSS).

A Value-Based approach has been proposed to Architecture Design Decision Support System that takes into account value considerations of stakeholders and only documents the information required by stakeholders.

There are many inhibitors as described in chapter 4 which may hamper capturing, using, and documenting the design decisions and its rationale. Also not all the information is needed all the time as different people need different information. Basically, the benefit of managing AK is not clear.So there is a need to decide what information is required to save that will benefit whom, for this we have taken the idea of Boehm [26], who proposed a Value-Based Software Engineering (VBSE) agenda and from Davide Falessi's work [27,28,29] who used Boehm's idea for documenting design decisions rationale i.e.

proposed a Value-Based approach to DDRD (VB DDRD). In the present work, Value-Based Software Engineering principles have been applied on an open-source tool Architecture Design Decision Support System[8,9]. This tool provides the opportunity to all the stakeholders to choose the required design decisions information by giving score to each attribute of design decisions.In this tool the value provided to each design decision attribute purely depends on the stakeholders' perception and their choices Architect can prioritize these attributes as required, useful or optional on the basis of the score provided by each stakeholder. ADDSS and other studied tools don't support the concept of value based software engineering. This is the main feature which is incorporated in ADDSS and this feature mainly answers the mentioned research questions.

There are certain inhibitors e.g. increase of overhead, potential inconsistencies, extra effort and time required etc as discussed in chapter 4 hampers managing architecture knowledge. Due to these inhibitors practitioners are reluctant to manage architecture knowledge. This is the reason why the architecture knowledge management tools and techniques are not practical. A Value-Based approach helps to make architecture knowledge management tools and techniques practical as this approach helps to mitigate the effect of above mentioned inhibitors. By applying VBSE principles to some tool makes the tool more effective, valuable and useful as it helps to document only set of required information based on its purpose.

Tailored architecture knowledge implies less information to document and maintain; hence, a diminished effort has the effect of mitigating the overhead. As the architecture knowledge is properly managed and documented i.e. decisions made in the past were properly managed and documented, so less effort is required at the time of maintenance phase and during evolution of any software system. This means we don't need to put extra effort and it saves a lot of time.

By recording only the required set of information for any architecture saves a lot of time as related persons are also performing other activities. The possibility to spend less time to produce the architecture knowledge documentation highly increases the possibility that people, who are busy to meet their projects deadlines, find enough time to develop such documentation.

The tailored architecture knowledge implies less information and hence less documentation. Less documentation implies both less required effort for architecture knowledge maintenance and less probability of inconsistencies occurrence. In this way we can make knowledge capture cost-effective.

As we know the stakeholders come from different backgrounds and have different concerns that the architecture document must address. With this approach, one can resolve conflict. As for choosing the design decision attributes, right of choice is given to all critical stakeholders who are associated with that project, so there is no chance of conflict between them. This approach also helps building the organizational capabilities as by recording only the required and useful information for any architecture according to the choice of different persons not only help them for that architecture or project but also helps them in future projects.

Basically as we know this support tells what type of architecture knowledge is required, useful or optional on the basis of scores given by all the stakeholders i.e. we are prioritizing attributes as required, useful or optional. From this we get to know what type of architecture knowledge is useful. . It is suggested that the use of such tailored architecture knowledge documentation would mitigate the effects of inhibitors as mentioned above and emphasize on the effects of its benefits.

Besides applying VBSE principles to ADDSS tool, a catalogue of architecture and design tactics has also been implemented in this tool. A set of templates has been developed to document catalogues of architecture and design tactics. This will helpful in recording design decision information as well as saves times and helpful for stakeholders.

Moreover tool is taking requirements according to the type which will be helpful while recording the design decision information as quality attributes should be taken separately so that we must know what quality attributes are affecting the design decisions or have impact on them. Also helpful while capturing scenario's as scenarios are used to characterize required quality attributes.

The other contribution is that this tool is capturing a general or concrete scenario, which can be elicited from a stakeholder or extracted from a pattern. Capturing scenarios helps

the users to understand the requirements easily. These are an artifact to architecture knowledge. Scenarios are helpful in characterizing required quality attributes.

Moreover, a template is designed that captures principles which guide decisions for an architecture. These principles are enterprise principles (business or others). Tool displays a list of catalogue of principles that creates a link between architecture design decisions and enterprise principles. Principles are captured which helps recording the design decisions as these principles helps guiding the design decisions i.e. it develops a link between the design decisions and these principles. One can easily select related principles from the catalogue of principles while recording design decision information.

Modified tool also captures artifacts that have an impact on architecture design decisions. Tool displays a list of catalogue of artifacts. Artifacts are stored which helps recording the design decisions as these artifacts have an impact on the design decisions. One can easily select related artifact from the catalogue while recording design decision information.

Architecture patterns are also helpful in documenting design decisions. Tool now captures architecture pattern also. ADDSS tool associates each architecture with single view however our tool now provides multiple views associated with single architecture. Multiple views with each single architecture helps recording design decisions information as design decisions have an impact on these views. Moreover, tool is showing the categorization of risks and non risks.

Value-Based Software Architecture Knowledge Management (VB-SAKM) tool is also evaluated on the basis of certain attributes. These attributes are the same as mentioned in Chapter 3. From the evaluation, it is found that the tool supports Value-Based Software Engineering principles.Moreover; it is a user-friendly tool and performance-wise good.

By implementing Value-Based Software Engineering principles and other features into ADDSS tool, the modified tool is now quite valuable, mature and useful for managing, sharing and storing architecture knowledge.

5.3 Limitations

The main limitation of this work is that surveyed tools and the modified tool i.e VBSAKM-tool has not evaluated on a large scale.These tools are evaluated only by the author of this thesis not by multiple persons or any team.The results of evaluation of tools had not been checked by some other persons or by any prjoect team.Moreover, the modified tool has not tried out into a real project.So we are not aware of the limitations which can be found out if the modified tool has been tried out into a project.

5.4 Future Work

In future, we plan to deploy and use the Value-Based Software Architecture Knowledge Management (VB-SAKM) tool into a project and would perform analysis within that industrial setting. Based on the feedback from the analysis, the limitations encountered and subsequent enhancements would be applied on the currently modified tool.

Moreover, there is a plan to develop a comprehensive tool for managing architectural knowledge that will cover all the limitations already found from the literature survey. The limitations and enhancements resulted from the deployment of VB-SAKM tool into some real life project would also be implemented in the newly developed comprehensive tool. This comprehensive tool would be a value-based tool.

Currently, no tool for managing architectural knowledge is integrated with any existing case tools or with development phases. For this direction, there is a need for integration of AK management tools with other case tools (e.g. requirements management tools) to provide an integrated and unified environment to software engineers.

APPENDIX-A

GLOSSARY

A-1 Glossary

A-1.1 Acronyms and Abbreviations

Term	Description
AK	Architectural Knowledge
ADD	Architectural Design Decisions
VBSE	Value-Based Software Engineering
SCS	Success-Critical Stakeholders
PAKME	Process-based Knowledge Management Environment
ADDSS	Architecture Design Decision Support System
DDRD	Design Decision Rationale Documentation
VB-SAKM	Value-Based Software Architecture Knowledge Management

Table A-1: Acronyms and Abbreviations

REFERENCES & BIOBLIOGRAPHY

[1] Len Bass, Paul Clements, and Rick Kazman. ***Software architecture in practice*** 2nd ed. Addison Wesley, 2003.

[2] Kruchten, P., Lago, P., van Vliet, H. and Wolf, T. ***Building up and Exploiting Architectural Knowledge***, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).

[3] P. Kruchten. Architectural Blueprints. ***The 4+1 View Model of Software Architecture***. *IEEE Software*, 12(6):42–50, 1995.

[4] Lago, P. and Avgeriou, P. ***First Workshop on Sharing and Reusing Architectural Knowledge***, To appear in ACM Software Engineering Notes, ACM SIGSOFT Software Engineering Notes, 3(5), 32-36 (2006).

[5] IBM (2003). ***Rational Unified Process***, Version 2003.Cupertino, CA: IBM Rational Software.

[6] Perry, D.E. and Wolf, A.L. ***Foundations for the Study of Software Architecture***. ACM SIGSOFT Software Engineering Notes, 17 (4). 40-52.

[7] Lago, P. and Avgeriou, P. ***„Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*** (SHARK/ADI'07: ICSE Workshops 2007) , 20-26 May 2007

[8] R. Capilla, F. Nava, S. P´erez, and J. C. Due˜nas. ***A Web-based Tool for Managing Architectural Design Decisions***. In 1st ACM Workshop on Sharing Architectural Knowledge (SHARK), Torino, Italy, 2006.

[9] R. Capilla, F. Nava,and J. C. Due˜nas. ***Modeling and Documenting the Evolution of Architectural Design Decisions*** accepted in the 2nd Workshop on SHAring and Reusing architecture knowledge. Architecture, Rationale, and Design Intent, 2007, Minneapolis, USA.

[10] R. C. de Boer, R. Farenhorst, V. Clerc, J. S. van der Ven, P. Lago, H. van Vliet, ***A Model for structuring Software Architecture Project Memories***, Proceedings of the 8th International Workshop on Learning Software Organizations, 2006.

[11] J. Tyree and A. Akerman, ***Architecture decisions: Demystifying architecture***. IEEE Software, 22(2):19–27,2005.

[12] J. S. van der Ven, A. G. J. Jansen, J. A. G. Nijhuis, J. Bosch, ***Design decisions: The bridge between rationale and architecture***, In A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech (Editors), Rationale Management in Software Engineering, Chapter 16, Springer-Verlag, March 2006.

[13] J. Bosch, ***Software architecture: The next step***, Proceedings of the First European Workshop on Software Architecture (EWSA), Volume 3047 of LNCS, Springer, pp. 194-199, 2004.

[14] A.G.J. Jansen, J., Bosch, ***Software architecture as a set of architectural design decisions***, Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 5), 2005.

[15] Ali Babar, M., Boer, R., Dingsoyr, T., Farenhorst, R., ***Architectural Knowledge Management Strategies: Approaches in Research and Industry***, accepted in the 2nd Workshop on SHAring and Reusing architecture knowledge . Architecture, Rationale, and Design Intent, 2007, Minneapolis, USA.

[16] *5th Working IEEE / IFIP Conference on Software Architecture (WICSA 2005), 6-10 November 2005, Pittsburgh, Pennsylvania, USA*. IEEE Computer Society, 2005.

[17] Anton G. J. Jansen, Jan van der Ven, Paris Avgeriou, Dieter K. Hammer, ***Tool support for Architectural Decisions***, Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA 2007).

[18] S Trujillo, M Azanza, O Díaz,R. Capilla. ***Exploring Extensibility of Architectural Design Decisions*** accepted in the 2nd Workshop on SHAring and Reusing architecture knowledge . Architecture, Rationale, and Design Intent, 2007, Minneapolis, USA.

[19] Ali Babar, M., Gorton, I., ***Architecture Knowledge Management: Concepts, Technologies, Challenges***, Presented at the 6th working IEEE/IFIP conference on software architecture (WICSA) 2007, Mombai, India.

[20] Ali Babar, M., Gorton, I., ***Architecture Knowledge Management: Challenges, Approaches, and Tools***, presented at the 29th International conference on software engineering, 2007,Minneapolis, USA.

[21] Ali Babar, M., Gorton, I., Jeffery, R., ***Toward a Framework for Capturing and Using Architecture Design Knowledge***, Tehnical Report.

[22] Remco de Boer : ***Architectural Knowledge Discovery, Why and How?*** In 1st ACM Workshop on Sharing ARchitectural Knowledge (SHARK), Torino, Italy,2006.

[23] Ali Babar, M., Gorton, I, ***A Tool for Software Architecture Knowledge Management***, accepted in the 2nd Workshop on SHAring and Reusing architecture knowledge . Architecture, Rationale, and Design Intent, 2007, Minneapolis, USA.

[24] Ali Babar, M., Boer, R., Dingsoyr, T., Farenhorst, R., ***Architectural Knowledge Management Strategies: Approaches in Research and Industry***, accepted in the 2nd Workshop on SHAring and Reusing architecture knowledge . Architecture, Rationale, and Design Intent, 2007, Minneapolis, USA.

[25] Rik Farenhorst. "***Tailoring Knowledge Sharing to the Architecting Process***" Presented at the *1st ACM Workshop on SHAring and Reusing architectural Knowledge (SHARK)*, Torino, Italy, 2006.

[26] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P.Grünbacher. 2006 ***Value-Based Software Engineering***. Springer.

[27] D. Falessi, M. Becker, and G. Cantone. ***Design Decision Rationale: Experiences and Steps Ahead Towards Systematic Use.*** In 1st ACM Workshop on Sharing ARchitectural Knowledge (SHARK), Torino, Italy,2006.

[28] Davide Falessi, Rafel Capilla, Giovanni Cantone, ***Value-Based Design Decision Rationale Documentation: A Replicated Experiment***, Third Workshop on SHAring and Reusing architectural Knowledge (SHARK2008) In conjunction with the 30th Int. Conf. on Software Engineering (ICSE2008), Leipzig, Germany, 10 - 18 May 2008.

[29] Davide Falessi, Giovanni Cantone, Philippe Kruchten, ***Value-Based Design Decision Rationale Documentation:Principles and Empirical Feasibility Study***, Seventh Working IEEE/IFIP Conference on Software Architecture, (WICSA 2008), Vancouver, Canada, 18-22 February 2008.

[30] M. Sinnema, J. S. van der Ven, S. Deelstra, ***Using Variability Modeling Principles to Capture Architectural Knowledge***, Proceedings of the Workshop on SHAring and Reusing architectural Knowledge (SHARK2006), June 2006 .

[31] http://www.softwaretestinghelp.com/what-are-the-quality-attributes/

[32] Knowledge Architect Word plug-in,. http://www.rug.nl/informatica/onderzoek/programmas/softwareengineering/griffin/KnowledgeArchitectWordPlug-in.

[33] Griffin project website, . http://griffin.cs.vu.nl.

[34] Archium website, . http://www.archium.net.

[35] AREL website,. http://www.ict.swin.edu.au/personal/atang/

[36] ADDSS website,. http://triana.escet.urjc.es/ADDSS/

[37] Larix Lee Kruchten, P., ***Capturing Software Architectural Design Decisions***, appear on Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on, April 2007.

[38]Ibrahim Habli,Tim Kelly, ***Capturing and Replaying Architectural Knowledge through Derivational Analogy***, accepted in the 2nd Workshop on SHAring and Reusing architecture knowledge . Architecture, Rationale, and Design Intent, 2007, Minneapolis, USA.

[39] Neil B. Harrison, Paris Avgeriou, Uwe Zdun, ***Using Patterns to Capture Architectural Decisions***, July/August 2007 (Vol. 24, No. 4) ,IEEE.
[40] The-Standish-Group, CHAOS Report 1995,www.standishgroup.com, 1995.

[41] ***Design Patterns, Quality Attributes and Software Architectural Tactics***, Felix Bachmann,Len Bass, Mark Klein.

[42] Remco C. de Boer, Rik Farenhorst, Patricia Lago, Hans van Vliet, and Anton G. J. Jansen. ***Architectural Knowledge: Getting to the Core***. In Proceedings of the Third International Conference on the Quality of Software Architectures (QoSA 2007), volume 4880 of LNCS, pages 197–214, July 2007.

ABOUT AUTHOR

The author, Nida Ahmad, daughter of Mrs. Talmeez Ahmad and Mr. Mohammad Ahmad, was born in Islamabad on February 1983.She got married to Mr. Salman Alam in 2007 and now has a daughter named Hiba Alam. The author did her matriculation securing first division from Islamabad Model College for Girls, F-7/4 (Pakistan) in 1999. She passed her Higher Secondary School Certificate Examination in first division from the Islamabad College for Girls, Islamabad, in 2001. She did her Bachelor of Science in Computers Science from International Islamic University, Islamabad, Pakistan in 2006. The author got admitted in International Islamic University, Islamabad, Pakistan, to earn Master of Science Degree in Software Engineering in 2006. Her current fields of interest are Software Architecture, Web-based applications, Knowledge Management, Project Management and Software Engineering.