International Islamic University, Islamabad, Pakistan

Faculty of Basic and Applied Sciences

Department of CS & Software Engineering

Date:

Thesis Acceptance Certificate

Thesis *"Impact of Architecture Design and Evaluation on Risk, Schedule and Cost Estimation for PSP Engineer"* has accepted by the Faculty of Basic and Applied Sciences, Department of Computer Sciences and Software Engineering in partial fulfilment of the requirements in *"Master of Science in Software Engineering"* with specialization in *"Software Process Engineering"*.

Submitted By:
Name: Waqar Ul-Hasnain Khokhar ,
Registration # 213-FBAS/MSSE/F08
CNIC # 35202-7940764-1

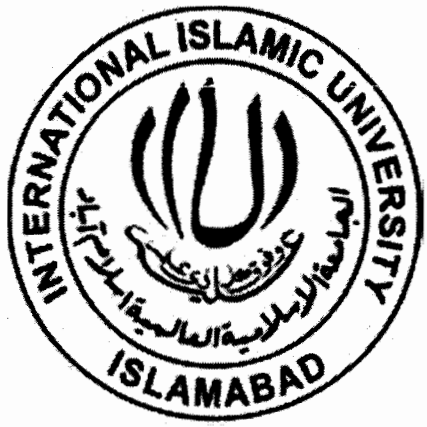
Committee

External Examiner
Dr. Umer Khan
Assistant Professor
Department of Mechatronics, Faculty of Engineering
Air University, Islamabad, Pakistan

Internal Examiner
Mr. M. Usman
Assistant Professor, DCS&SE, FBAS, IIUI, Pakistan

Supervisor
Mr. Shahbaz Ahmed
Assistant Professor, DCS&SE, FBAS, IIUI, Pakistan

IMPACT OF ARCHITECTURE DESIGN AND EVALUATION ON RISK, SCHEDULE AND COST ESTIMATION FOR PSP ENGINEER



Submitted by:

Waqar Ul-Hasnain Khokhar

Registration No. 213-FBAS/MSSE/F-08

Submitted in partial fulfilment of the requirements for the Master of Science in Software Engineering with specialization in Software Process Engineering at the faculty of Basic and Applied Sciences.

Department of Computer Science & Software Engineering,
International Islamic University,
Islamabad, Pakistan.

Supervisor:
Mr. Shahbaz Ahmed
Assistant Professor
DCS&SE, FBAS, IIUI,
Pakistan.

2012-December-7
1434-Muharram-23

MS
005.1
KHI

-1 Software Engineers
-2 Computer Science

Impact of Architecture Design and Evaluation on Risk, Schedule and Cost Estimation for PSP Engineer

ABSTRACT

Personal Software Process (PSP) has a long history of its implementation in industry and academic. It can be applied in 21 key process areas of CMMI. But in past the main emphasis was on planning, requirement, design, code, test and post-mortem phases of software development. Identification of risk at system structure level for estimation of cost and schedule is the key focus of PSP. However, current implementation of PSP does not provide any process for risk, schedule and cost estimation at the architecture level. The architecture design and evaluation phases highly influence other phases thus creates needs to be integrated within the development process. Also, architecture analysis for developer is proposed in literature. Therefore, this work presents a Personal Software Process with integration of architecture design and internal evaluation. The proposed process *"Personal Integrated Process (PIP)"* is designed and executed in the university computer lab. The impact of integration of architecture design and evaluation was determined by a case study. The process was automated to further facilitate the adoption of the process. The integration of architecture design and evaluation proved to be helpful in identification of technical risks early at the architecture level. The process integration demonstrated better estimation of cost and schedule. The tool found to be help in process learning. It was also found that process automation can substantially reduce project cost. Software quality management system developed to automate the process is open source software and can be used for research

purpose or it can also be used in any CMMI company for process automation and process improvement.

Impact of Architecture Design and Evaluation on Risk, Schedule and Cost Estimation for PSP Engineer

Copyright

Author Name: Waqar Ul-Hasnain Khokhar

Address: 190-Q-Block, Model Town, Lahore, Punjab, Pakistan.

Mobile # 92+300-4028714

Phone# 92+42+35861255

Email# waqarulhk@gmail.com.

TABLE OF CONTENTS

Chapter 6: Case Study

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

4+1VM-E	4+1 View Model Extension
ABAS	Attribute-Based Architectural Styles
ABC	Activity Based Costing
ABC/DD	Architecture Based Component composition/ Decision-oriented Design
ABD	Architecture Based Design
ACCA	Architecture-Centric Concern Analysis
ACE	Architecture-Centric Engineering
ADD	Attribute Driven Design
ADF	Application Development Framework
ADL	Architecture Description Language
AIS	Advanced Information System
ALMA	Architecture Level Modifiability Analysis
APA	Architecture Potential Analysis
API	Application Programming Interface
APTIA	Analytic Principles and Tools for the Improvement of Architecture
AREL	Architecture Rationale and Elements Linkage
ARID	Active Reviews for Intermediate Designs
ASC	Architectural Separation of Concerns
ASD	Architecturally Significant Decision
ASR	Architecturally Significant Requirement
ATAM	Architecture Tradeoff Analysis Method
BAPO	Business Architecture Process and Organization
BIS	Business Information System
CBAM	Cost Benefit Analysis Method
CDS	Commanding Display System
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CMMI-DEV-1.2	CMMI for Development Version 1.2

DFD	Data Flow Diagram
DMAIC	Define, Measure, Analyze, Improve, and Control
EBBS	Electronic Bulletin Board System
EFI	Electronic Fund Transfer
ERP	Enterprise Resource Planner
E-SAAM	Extended SAAM
FODA	Feature-Oriented Domain Analysis
FORM	Feature-Oriented Reuse Method
FSQMS	Firmsoft Software Quality Management System
GMSAD	General Model for Software Architecture Design
HMI	Human Machine Interface
HP-QC	HP Quality Center
JPA	Java Persistence API
JSF	Java ServerFaces
LTSP	Level TSP
MAD	Methodological Architectural Design
OPC	Open Process Components
PASA	Performance Assessment of Software Architecture
PSPA	Personal Software Process Assistant
PBC-GZ	People's Bank of China Guangzhou
PBX	Private Branch Exchange
PIP	Personal Integrated Process
PPMP	Personal Software Engineering Project Management Process
PROBE	Proxy Based Size Estimation
PROM	PRO Metrics
PSP	Personal Software Process
PSPBOK	PSP Body of Knowledge
PSP-DROPS	PSP Data RepOsitory and Presentation System
PSP-EVA	Personal Software Process Expert Visualization Agent
PWP	Personal Writing Process
QADA	Quality-Driven Architecture Design and Analysis
QADAG	Quality Attribute Directed Acyclic Graph
QAW	Quality Attribute Workshop
RAMRTS	Rate Monotonic Analysis for Real Time Systems

RDP	Rational Design Process
RMA	Rate Monotonic Analysis
RSE	Reflective Software Engineering
SAAER	Software Architecture Analysis for Evolution and Reusability
SAAM	Software Architecture Analysis Method
SACAM	Software Architecture Comparison Analysis Method
SARA	Software Architecture Review and Assessment
SBSAD	Scenario Based Software Architecture Design
SBSAR	Scenario Based Software Architecture Reengineering
SCADA	Supervisory Control and Data Acquisition
SEI	Software Engineering Institute
SPE	Software Performance Engineering
SPICE	Software Process Improvement and Capability dEtermination
SQL	Structured Query Language
SSPMT	Six Sigma Project Management Tool
TSP	Team Software Process
TTSP	Tailored TSP
UML	Unified Modeling Language
V&B	View and Beyond
VDM	Vienna Development Method

CHAPTER 1
INTRODUCTION

1.1 PERSONAL SOFTWARE PROCESS

The *"Personal Software Process (PSP)"* [1] was designed for individual software engineer or a team of three to five engineers. PSP is the initiative of *"Software Engineering Institute (SEI)"* developed to improve the performance of software engineers especially for small organizations. It is a structured, disciplined, and measureable software process for engineers. PSP helps in improving quality and productivity of engineers and make their result more predictable. Using PSP engineers can identify areas where they can improve from their process feedback. An engineer can use PSP to improve the quality and productivity. PSP is based on five principles which emphasize the measuring and tracking work progress, applying suitable methods consistently, and use of defined process for evaluating his own work. Engineer can use consistent personal practices and become effective member of the team.

PSP introduces a set of seven personal processes PSP0, PSP0.1, PSP1, PSP1.1, PSP2, and PSP2.1. PSP 3 is the cyclic process. PSP0 is the baseline personal process which introduces defect recording and time logging activities. PSP0.1 introduces the use of coding standard. Preparing process improvement proposal and measuring software size. PSP1 is also refers to as personal planning process, which introduces software size and resources estimation using PROBE. Software testing activity is focused and test reports are introduced in this process. PSP1.1 introduces task planning and schedule planning. PSP2 introduces code review and design review activities. Finally PSP2.1 introduces design templates. PSP3 is cyclic personal process for developing large scale programs.

It is a structured, disciplined, and measureable software process for engineers. The PSP helps in improving quality and productivity of engineers and make their

result more predictable. Using PSP engineers can identify areas where they can improve from their process feedback. However, initial work reported focuses only on requirement, design, code, test, and post-mortem phases.

1.2 PROCESS SIGNIFICANCE

The PSP has successful track of industrial application especially in Digital Equipment Corporation, Hewlett Packard Corporation and AIS Corporation [1]. Another publication [6] reports the use of PSP in Motorola Paging Products Group, Union Switch & Signal Inc. and Advanced Information Services Inc. The most projects executed at these organizations contain one to three software engineers. However, project executed at AIS involved two groups with three to five software engineers. The most projects executed at Motorola contain zero defects which exposes the significance of using PSP in industry.

1.3 PROBLEM STATEMENT

Identification of risk, estimation of cost and schedule at system structure level is the key focus of PSP.

"You must examine the system structure, assess the principal development risks, and select a strategy to fit your situation" [2]

However, current implementation of PSP does not provide any process for risk, schedule and cost estimation at the architecture level. The architecture design and evaluation phases highly influence other phases and need to be integrated within the development process.

PSP3 is cyclic process for developing large software, i.e. PSP3 is large scale personal process. However, PSP3 process addresses only module level development within a component. **It does not address component level development within a product.** A software engineer who wants to develop a large complex product with number of components and each component may contain number of modules will need a process that addresses these activities. In other word current implementation of PSP does not provide processes for architecture design and architecture evaluation.

The current version of PSPBOK [3] addresses the seven necessary competency areas: *"Foundational Knowledge"*; *"Basic PSP Concepts"*; *"Size Measuring and Estimating"*; *"Making and Tracking Project Plans"*; *"Planning and*

Tracking Software Quality"; *"Software Design"*; *"Process Extensions and Customization"*. **However, it does not address architecture design and evaluation competencies.** The first competency area is foundational knowledge which comprises of four knowledge areas. These knowledge areas are process definition, process elements, measurement principles and statistical elements. The basic PSP process has planning phase, development phase and post-mortem phase. The main four elements of PSP are scripts, forms, measures and standards used mainly in coding, counting line of code and defects. The third knowledge area "measurement principles" describe essential product and process measures. Two types of measures are involved in PSP one is artifact measure which is used for measuring product size or defects. The other measures are for historical process measures and current process measures. The knowledge area "statistical elements" focus measurement and analysis using statistics. The second competency area is "basic PSP concepts" which involves knowledge areas such as process fidelity, data collection, data measure, data analysis and process improvement. The third competency area "size measuring and estimating" describes knowledge areas such as size measure, size data, size estimating principles, proxies, the PROBE estimating method, combining estimates and size estimation guidelines. Making and tracking project plans is the fourth competency area which focuses on six knowledge areas. These knowledge areas are planning principles, planning framework, software size and effort, task and schedule planning, use of earned value for schedule tracking, planning and tracking project issues. Planning and tracking software quality is the fifth competency area which involves six knowledge areas. These knowledge areas are quality principles, quality measures, quality methods, code reviews, design reviews, and review issues. Software design is the sixth competency area. This competency area involves six knowledge areas which are software design principles, design strategies, design quality, design documentation, design templates, and design verification. Finally the last and seventh competency area is "process extensions and customization". This competency area focuses on three knowledge areas which are "defining a customized personal process", "process evolution", and "professional responsibility".

Engineer can use PSP principles in any of 21 key process areas of CMMI-DEV-1.2 [4]. PSP is the class room implementation of CMMI. Initial reported work on PSP focuses only on requirement, design, code, test, and post-mortem phases. **However, it does not address architecture design and evaluation. These are the**

limitations of PSP and are focus of the research. PSP does not contain team processes. The *"Team Software Process (TSP)"* [5] is designed for managing team of software engineers. TSP is beyond the scope of this research.

The PSP is designed for small organizations with few engineers and does not define team software processes. However, PSP engineers can be guided through a team oriented approach i.e. Team Software Process (TSP). PSP does not depends on the use of any special tool, but PSP engineers have to go through many phases and have to fill out many forms and prepare script which utilizes huge development time. Therefore, automation of PSP will reduce development time and cost The automation will also help in process learning as emphasized below.

"With CASE facilities to automatically log time, track defects, maintain data, and present statistical analyses, the PSP likely would be easier to learn and more efficient to use."[1]

The Personal Software Process need its automation to reduce cost and time spent in learning process and manual work. However, there are some issues in current PSP process that need to be addressed. The PSP needs to be integrated with architecture design and evaluation processes.

The *"Software Architecture Analysis Method (SAAM)"* was proposed to address the analysis of context dependent quality attributes. It is a scenario based analysis of software architecture. The method was developed to compare software architecture. The major activities involves in this method are *"describe the candidate architecture"*, *"develop scenarios"*, *"evaluate scenario"*, *"reveal scenario interaction"* and finally *"weight scenario and scenario interactions"*. The method was successfully applied in a case study. The senior development staff and managers found this method very helpful. They were able to identify many problems early in the software development stage. These problems were identified at very low cost. They commented about architectural analysis this way:

"It has convinced management that developers need architectural analysis up front."[7]

The *"Team Software Process (TSP)"* has recently been combined with *"Architecture-Centric Engineering (ACE)"* [8]. The major process components of ACE are *"Quality Attribute Workshop (QAW)"*, *"Attribute Driven Design (ADD)"*, *"View and Beyond (V&B)"* approach to document the software architecture, *"Architecture Tradeoff Analysis Method (ATAM)"*, and *"Active Reviews for Intermediate Designs (ARID)"*.

However, risk identification process ATAM together with other process components have combined with TSP and is designed for coaches or managers and not for individual engineer.

1.4 PROPOSED SOLUTION

The *"Personal Integrated Process (PIP)"* is proposed to address above mentioned limitations. It is a consistent, systematic, disciplined, and repeatable process for individual software system engineer. The proposed process is explained in detailed in chapter-5.

1.5 OBJECTIVES

The main objectives of the research are given below. These objectives were derived from issues and limitations in current software processes. These objectives also address the needs of "research question".

- To determine the impact of architecture design and evaluation on risk identification.
- To determine the impact of architecture design and evaluation on project cost.
- To propose a process for individual software system engineer.
- To automate the proposed process i.e. "personal integrated process".

1.6 RESEARCH QUESTION

The process variations highly impacts the cost associated with the software project. Any modifications in software process can dramatically increase or decrease productivity. Research question is designed so that factors affecting process can be examined.

RQ: What is the impact of architecture design and evaluation on risk, schedule and cost for PSP Engineer?

1.7 SUMMARY OF RESULTS

Process components necessary for software system architecture design and evaluation were integrated in PSP. The proposed process was executed successfully. A case study was designed to address the research question. Data related to research question was gathered and numbers of analysis were carried out. The results of these

analysis shows that proposed process is suitable for individual software system engineer. Only 15% of the cost was associated with the non coding activities such as architecture design and evaluation.

CHAPTER 2
RESEARCH METHODOLOGY

2.1 INTRODUCTION

To achieve the objectives of the research, literature survey on two topics were carried out followed by a case study. The research method used in this research was quantitative in nature. However, the analysis carried out during literature survey was qualitative.

2.2 LITERATURE SURVEY

Literature surveys on two topics were carried out. The first topic of literature survey was "Personal Software Process" and the second topic of literature survey was "architecture design and evaluation process". These literature surveys were sent for publication before conducting the case study.

2.3 CASE STUDY

Case study is chosen as method for study design for carrying out research. It is an empirical method for conducting research in software engineering [9], [10]. Case studies are descriptive and observational with or without some rational. These studies are used for validating research. In software engineering case study is used for comparing or evaluating processes, tools and methods. Case study is a scientific method for raising research question regarding investigation. Collection of row data during research and analysis of data set is purely scientific in nature. Finally the research findings and conclusion can be published scientifically. Case study scientific method of investigation has many strengths and weaknesses some of them are discussed below [11].

2.3.1 Strengths

Case studies provide sufficient information for the judgment of the technology, process or tools. They are suitable for comparing processes, tools or

technology. They are scientific investigation methods within the field of software engineering. Case studies methods of investigation are important for industrial evaluation of software engineering processes, tool and technology. Case studies are easier to plan as compare to formal experiments. There are number of strengths of case studies over other methodology such as: case studies are useful for evaluating software engineering methods and tools; case studies avoid scale-up problems; easier to plan; preferred when process changes are wide-ranging; case studies can be used to find out which process is better; finally case studies can be used to control improvement. The case study is suitable for settings where software researcher have little control over variables.

2.3.2 Weaknesses

The results obtained for case study are sensitive and cannot be compared with formal experiments. Formal experiment involves replication whereas case study does not require any replication. Case study investigation is carried out for a typical project undertaken. Case studies results are harder to interpret when compare to experiments. The research findings from case studies cannot easily be generalized. The results are context dependent.

2.3.3 Research Question

Case study was selected for empirical investigation of the impact of process change on project risks, schedule implication and project cost. The research question posed for investigation is given below.

What is the impact of architecture design and evaluation on risk, schedule and cost for PSP engineer?

2.3.4 Research Design

The case study by design was the single case holistic design. However, case study is based on theory and appropriate links to the existing literature has been established. The subject of the case study was the software processes. The reason for selecting single case was the uniqueness of the case. The source of research evidences collected during case study includes documentation, archival records, direct observation and physical artifact. The study design based on the reference period is

retrospective-prospective. However, there is no control group in the study design. Other evidences that would be used for construct validity includes PSP standard forms.

2.3.5 Case Study Protocol

The case study generic protocol template [12] and review checklist [13] were modified to adapt the specific needs of the case study. The case study protocol ensures consistent planning process. It increases the rigour of the case study. When research circumstances change case study protocol makes it easier to adjust new circumstances.

2.3.6 Case Study Context

The main context of the case study includes: Design of process for software system development was in the context of the case study. Development of software quality management system for Personal Software Process (PSP) and Personal Integrated Process (PIP) was the one of the context of the case study. Determination of impact of architecture design and evaluation on cast, schedule and risk was also in the context of the case study.

2.3.7 Hypothesis

Architecture design and evaluation affect project cost and risk identification.

2.3.8 Variables

Various dependent and independent variables were involved in different analyses. Detail of these variables will be discussed as these analyses will be explained in next chapters.

2.3.9 Ethical Issues

Case study design and research proposal were approved before conducting research.

2.3.10 Data Gathering

PSP standard forms and modified PSP forms were used for data gathering during investigation. Narrative observation was also used during data collection.

2.3.11 Results and Analysis

Case study involved number of analysis. These analyses include project software size, time, defects, effort, cost, and risks. All these analysis were quantitative. However, some qualitative analyses were also conduced. These analyses were carried out during literature review for comparing PSP tools and architecture design and evaluation processes. Results of some of these analyses have been published and other is sent for publication.

2.3.12 Construct Validity

Construct validity is the establishment of operational measures that are correct for the concept being studied. Construct validity can be explained in term of intentional validity, representation validity, and observation validity. Construct validity can also be defined as what is being measured was intend to measure. The product size, development time and system defects were the three basic measures. These measures adequately represent the quantities which were intended to study. Other measures were project cost that was measured in standard units. Project risks such as availability risks, security risks, performance risks were also measured in standard units. These concepts were operationalized for measuring correctly and given in the Appendix-A. Multiple source of evidence, establishment of chain of evidence, and expert review were used to ensure construct validity.

2.3.13 Internal Validity

Internal validity is the establishment of causal relationship. If the value of dependent variable is the result of independent variable the relationship is said to have an internal validity. To achieve internal validity confounding variables are eliminated. The internal validity is mostly applied to explanatory and causal studies only. The case study by nature was comparative. The case study was used to contrast standard Personal Software Process with Personal Integrated Process. This comparison helped in understanding the limitation of PSP. The causal relationship between outcome and treatment were also determined in the case study.

2.3.14 External Validity

External validity is the establishment of domain for which the case study can be generalized. Theory was used in single case study to ensure external validity.

2.3.15 Experimental Reliability

Experimental reliability is the degree of results to which the case study can be repeated. The documentation of case study is the prerequisite for experimental reliability. The case study protocol was used for ensuring experimental reliability.

2.3.16 Measurement Procedures

The data were collected in architecture design; architecture evaluation; detailed design; code; compile; test; planning; and post-mortem phases of process. Data were collected using standard PSP forms and modified PSP forms.

Instrument Validity: The validity of research instrument is its ability to measure the quantity for which this instrument was designed. There were many research instruments involved in the case study and the validity of these research instruments were determined using logic evidences. Expert reviews were used to ensure the validity of research instruments.

Face and Content Validity: Face and content validity was established by linking research question with objectives. Table 2.1 gives details of validation.

TABLE 2.1
Face and Content Validity

Objectives	Process-1
To determine the impact of architecture design and evaluation on risk, schedule and cost.	How many risks were identified in process-1?
	Process-2
	How many risks were identified in process-2?
	How much architecture design and evaluation impacted on project cost?
	How much architecture design and evaluation impacted on project schedule?

Instrument Reliability: The reliability of research instrument is the degree of accuracy and precision by which it makes measurements. PSP involves three basic measures size, time and defects. The program size was measured using automated

software tool. Time was noted using digital clock, and defects were counted manually. Software performance and security were also measured using software tools. These software instruments always provided accurate and precise results.

2.3.17 Sampling

Sample size was the whole population, i.e. all project risks identified were measured and analyzed. Project cost was estimated and compared with actual for both processes.

2.3.18 Research Setting

The case study was conducted in university computer lab. The place was selected based on the computer and networking facilities available in the lab. The type of study by nature was comparative. Two processes were compared during the investigation.

2.3.19 Reporting

The literature reviews conducted on two topics were submitted for publication. The analysis, results, findings and conclusion drawn from the case study were also submitted for publication.

2.3.20 Schedule

The case study took about 36 months, during first 12 months literature review on two topics was conducted. The analysis and findings of the literature review were published in international journal. In second phase research question was designed and approved from the research committee. The sort coming in the literature review, research problem and case study protocol was addressed in third phase. The second and third phase took about 6 months. In forth phase research proposal was approved for execution. During this phase process was designed, implemented and tested. Software quality management system was designed, implemented and tested. Row data was gathered and analysis was carried out over this data. The fourth phase took about 12 months. The analysis, results, findings and conclusion drawn from the case study were sent for publication in the fifth phase.

2.3.21 Unit of Analysis

The unit of analysis refers to as a "case" in a case study. The unit of analysis is based on the research question under investigation. The unit of analysis in the case study was a "process".

2.3.22 Case Study Findings

The project cost that was associated with the architecture design and evaluation was about 15%. By this investment project risks were identified early in the development stage. These risks if not identified would result in poor estimation of project cost and schedule.

2.3.23 Limitations

More case studies and experiments are required to generalize the findings. The major phases that were studied includes requirement; architecture design; architecture evaluation; detailed design; coding; testing; planning; and post-mortem. Software maintenance was not the focus of the study. There is also need to validate the process for other domain to ensure external validity these major domains include embedded firmware and industrial software.

2.3.24 Significance of Study

The proposed process is designed for development of enterprise resource planner or business information systems. The process can be used to build such commercial system in software industry. The proposed process helped in identification of risks at the early stage of software development. The identification of risks helped in better estimation of cost and schedule for project. The automation of process will significantly reduce the development time and context switching time for engineer. The PSP tool will help in process learning. The tool can be used in research activities carried out in universities or it can be used commercially in any CMMI organization. Software quality management system developed for PSP and PIP is an open source software project. This open source software project will help in knowledge and technology transfer.

CHAPTER 3

LITERATURE SURVEY ON PERSONAL SOFTWARE PROCESS

3.1 THE GAP IDENTIFICATION

To find the gap in research a literature survey was conducted. Considering institutional requirement survey sources were selected. Keywords and index terms were used to find the research articles from these research sources. Survey sources, index terms, and quality criteria for research articles selection are discussed below.

3.2 SURVEY SOURCES

The survey sources include digital libraries, scientific database, publishers, search engines, and references/bibliography. These sources include:

IEEExplore Digital Library; ACM Digital Library; CiteSeer Digital Library; IEEE Computer Society Digital Library; ScienceDirect; Web of Science; British Computer Society; Wiley InterScience; Springer Verlag; Springer Science + Business Media; Kluwer Academic Publishers; Elsevier; and Google.

3.3 INDEX TERMS & KEYWORDS

The keywords selected from research question were "PSP", "Personal Process", and "Process Tool".

3.4 SEARCH STRING

The search string was formulated based on keyword and index terms selected in previous section. The string was designed so that only related articles for the survey may be searched. Following search strings were used in survey.

String-1: "Personal Software Process".

String-2: "PSP".

String-3: "Personal Software Process Tool".

3.5 STUDY SELECTION

The search string was used to find articles from selected survey sources. The articles selected during primary study were based on search string appeared in publication title, abstract, or keywords. The total publications selected in phase-1 were 98. These publications were downloaded after reading title and abstract. Most of publications downloaded in phase-1 were selected from reference/ bibliography. So these publications already fall in the inclusion criteria.

Inclusion Criteria: The reviewed article includes research papers from journals, conferences, workshops, and symposiums. Copyrights technical reports, doctoral dissertation and books were also reviewed.

Exclusion Criteria: White papers, technical reports, books and website articles without copyrights were not considered. The total selected research papers after applying quality criteria in phase-2 were 87. These publications are shown in Fig.3.1.

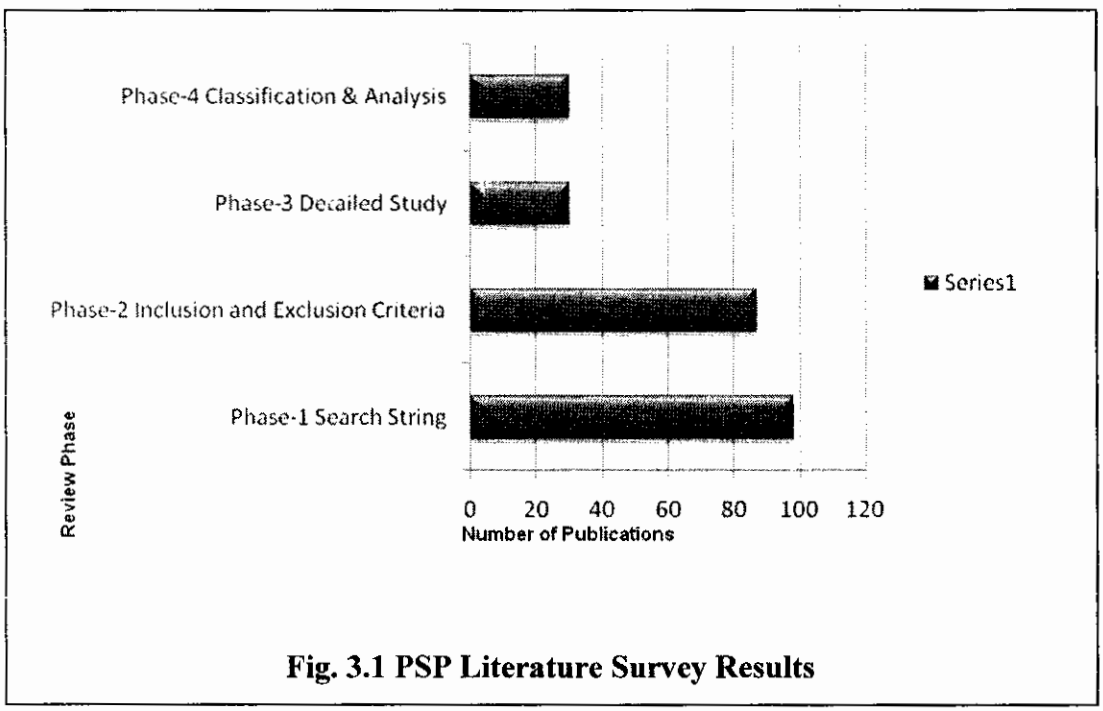


Fig. 3.1 PSP Literature Survey Results

In phase-3 the total selected research papers for detailed study were 87. The study includes those publications which fulfilled the inclusion and exclusion criteria. These articles were studied completely and selected for analysis. However, 57 articles were related to standard PSP. As the focus of the research was to find those articles that deals with modification of PSP or integration of other process components with

PSP. These articles were further separated to keep the focus of the research. In phase-4 the total research papers selected for analysis were 30 and can be seen in Fig. 3.1.

3.6 CLASSIFICATION AND ANALYSIS

The total of 30 publications that were selected during phase-4 of the literature survey was studied in detailed. It was found that 12 publication described just the automation of personal software process, 6 publication described integration of other process components with personal software process, 3 publication proposed the modified process, 7 publication proposed the modification of process as well as they automated the proposed process, and finally 2 publications proposed the modification in Team Software Process (TSP).

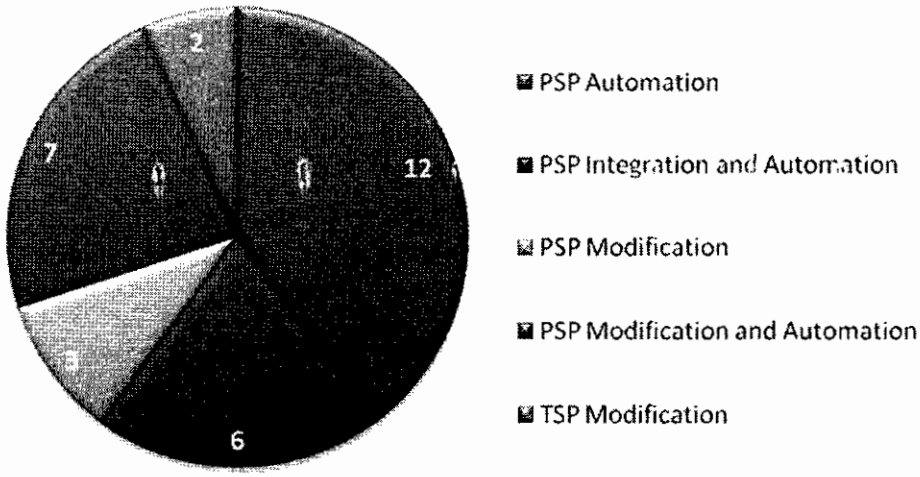


Fig. 3.2 Classification of Articles

Analyses were performed for the selected survey population. The sample size was 30 which is the total number of classified publications. Fig.3.2 shows the classification of these publications selected during literature survey. The selected 30 publications were studied in detailed. A study was conducted to find out whether these publications proposed modification or integration of architecture design and

evaluation in PSP. **No publication was found that proposed architecture design and evaluation for PSP. However, only one case study was found that proposed the architecture design and evaluation for individual engineer. This study did not proposed process for PSP.** Also no PSP tool could be found that provides facilities for managing architecture design and evaluation knowledge in PSP context. Therefore, there is a strong need for a process with integrated architecture design and evaluation for personal software process engineer. This integration will enhance the knowledge, skill and competency of the software engineer [15]. The process automation will reduce the learning process learning time. It will also help in better estimation of cost, time, and risk early in the development. The purpose of this research is to find the impact of architecture design and evaluation on cost, schedule and risk. The study will open new area for research to further investigate the impact of architecture design and evaluation.

3.7 PSP AUTOMATION

The research publications that proposed PSP automation software tool are discussed in this section. All the publications discussed in this section addressed standard PSP with automation support. The publications that were classified in this section were 12. Out of these twelve publications three publications addressed same process. Therefore, rest of 10 publications are discussed here in brief.

3.7.1 PSP-DROPS

The *"PSP Data RepOsitory and Presentation System (PSP-DROPS)"* is a web based tool [16] that support Personal Software Process. The tool is developed at *Embry-Riddle Aeronautical University* to automate the PSP process and to facilitate the teaching of PSP. PSP-DROPS helped in cut down the work load of management and analysis of personal software process data. Since it is web based tool students can access this tool anywhere in the world using internet. The tool was developed to address the issues in adaptation of PSP such as: extensive effort required in recording process data; Process involves too many different forms; and lack of immediate assistance on how to record the PSP data. PSP-DROPS provide assistance in filling different form, performs calculation on collected data, store process data in database, and generates graphical analysis. These reports and analysis result can be generated

anywhere in the world. The web base architecture of PSP-DROPS fulfils the goal of its development for PSP data for academic and industrial use. The four major software architectural components of the PSP-DROPS are PSP database server, *"CS1 module"*, *"CS2 module"* and a *"Teaching Assistant module (TA)"*. The security feature of PSP-DROPS allows only authorized and authenticated user to login and retrieve his own PSP data. However, student can only enter data and are not allowed to update the data.

3.7.2 Hackystat

The *"Hackstat"* [17] is one of PSP metrics collection and analysis tool which reduce development time and cost thereby increase productivity of PSP engineer, but introduces some adoption barrier.

3.7.3 PROM

The *"PRO Metrics (PROM)"* data collection and analysis tool [18] is developed for Personal Software Process. PROM provides automated data collection and analysis facility for both code and PSP process measures. This tool not only process PSP data but also data of procedural metrics, object oriented metrics, and custom metrics. PROM provides data collection and analysis facilities for personal, workgroup and enterprise levels. The architecture of PROM is based on plug-in technology and use SOAP to communicate with various component and subsystems of the architecture. This architecture based on Package-Oriented Programming that makes the development and integration of its components easier and extensible. PROM is consists of four components *"Database"*, *"PROM server"*, *"plug-in server"* and *"Plug-in"*. Database of PROM is used to store data regarding PSP data, software metrics and project activities. PROM server use SOAP web services to communicate with other components. Plug-in server collect data from plug-ins provide caching facility and communicate with PROM Server for data storage. The fourth component is plugs-in for IDE. The plug-in communicates with PROM server using SOAP protocol. PROM not only provides metrics and process support to developer but also to the manager. Developers can simultaneously login for pair programming and can access PSP data, and software metrics for analysis and improvement. PROM is fully automated to support the context switching problem in

process recording and product development. PROM is developed using Java technology and it component communicates using XML and SOAP. However, PROM also supports manual data insertion facilities.

3.7.4 PSPA

The *"Personal Software Process Assistant (PSPA)"* [19] designed and developed to facilitate the automatic collection of PSP data, viewing and editing PSP logs and reports. The tool also performs automatic classification and ranking of defects. This tool was developed to solve the adaptation issues of Personal Software Process. The tool has the capabilities of recording compile defects of programs written in Java and C. It not only automatically collects these defects but also classifies for individual software engineer and for team. PSPA is written in .Net(C#) with plug-in support written in Java. It uses centralized local database.

PSPA provides daily schedule tracking facilities, automatic line of code (LOC) counting facility, compile defects recording facility, size and time estimation facility, Time and defect recording facility, and facility of automatic classification and ranking of defects. The data related to these facilities is collected through plug-ins and stored in a local centralized database. The plug-in communicates with Eclipse open source IDE. PSPA provides tracking individual engineer task with the help of Gantt chart and a timer attach to it. Timer is invoked automatically when an engineer starts working PSPA tools. Time is automatic calculated using timer and logged in the database. These basic measures are used to calculate defect density and productivity of individual engineer. Defects are logged and classified using standard PSP defect type standard. These defects are ranked based on the frequency of their occurrence. PSP logs and reports are automatically produced, these reports include *"Productivity per Task"*; *"Yield per Task"*; *"Defect Density per Task"*; *"Estimate Size versus Actual Size"*; *"Estimated Time versus Actual Time"*; *"Estimated Cost versus Actual Cost"*; *"Productivity of Team"*; and history of member. The consolidated team Gantt chart provides information regarding individual and team productivity. This information is only available to manager for project monitoring and control. The defects information provides the quality at individual and at the team level.

3.7.5 DuoTracker

The *"DuoTracker"* is software tool [20] was designed and developed to address the process tracking and analysis needs of individual software engineer and for organization. Its defect classification capabilities and data collection is based on ISO 9001 and CMM standard. The tool is developed to address the issues related to adoption barrier of PSP. These issues includes manual entry of data; switching between applications; and collection of rigid data. DuoTracker provides the solution to these problems by integrating PSP data collection tool in ISO 9001 and CMM based defect tracking tool. The eleven mandatory categories of IEEE standard 1044-1995 was used for defect classification scheme and implemented in CMM based software lifecycle. These defects are collected using defect logger implemented as one of the component of DueTracker. DueTracker has ability of integrating with organization wide defect tracking and analysis tool. Software engineer can select any type of defect for his own analysis for the available defect type in this tool. Defect classified in IEEE standard 1044 are automatically logged whereas some PSP specific and unique field are manually entered. DueTracker provide a facility for comparing individual PSP quality data with organization wide product quality data. The tool provides a time logging facilities for each defect, such as the time at which defect was found and fixed. It allows engineer to analyze estimated defect fix time as well as actual fix time. However, in DueTracker recording of compilation errors are not automated. Another issue with the tool is security of PSP data which is restricted from viewing using some measures. The defect records in DuoTracker system is viewed by *"Defect Viewer"*. This viewer provides the necessary information regarding defect classification and logged-on user. DueTracker provides two different ways of updating defects one is for assigned defects and other is for unassigned defects.

3.7.6 PSP-EVA

The system software architecture of the *"Personal Software Process Expert Visualization Agent (PSP-EVA)"* tool is based on software agents [21]. The agents that were used in the tool were the combination of *"semi-intelligent agents"*, *"informative agents"*, and *"interface agents"*. The major agents that exist within the system are *"InterfaceAgent (IA)"*, *"TaskAgent (TA)"* and *"SearchAgent (SA)"*. These agents are used for visual representation of performance. These agents perform

several tasks including planning, scheduling and visual representation of defect density. The tool provides the statistical performance of the individual software engineer or for a team. The agents in the system work as a personal assistant for the software engineers. They provide the visual representation of the performance of the software engineers. The tool provides process support from PSP0.1 level through PSP3 level. It used PHP and AJAX language to communicate with the web. The project manager can generate Gantt chart for project tracking. For scheduling Gantt chart is provided in the tool for graphical representation of milestone. This visual representation of milestone in Gantt chart is accomplished with the help of agent that enhanced PSP tool. Performance visualization is accomplished with the help of interface agent. The user can monitor his performance with the help of these interface agents. These agents send alert messages to software engineers when they are slow. Finally the PSP-EVA provides analysis support for both software engineers and for project manager.

3.7.7 PSP-EAT

The *"PSP-EAT"* [22] tool is the enhancement of Excel worksheets based automation of software process. They developed this tool for providing assistance to teachers and students of *Master in Software Engineering* program conducted at *Monmouth University*. They observed that about five hundred calculations were involved in a single assignment given to a student. These assignments took about two to eight hours of instructors. They developed and enhanced the existing tool to reduce the manual time involved in the process. They were able to reduce manual work by 35%. They were also able to reduce post-mortem time by 64%. The features provided by the tool include the addition of a student, database support and reports on specific assignments.

3.7.8 Extended GESIP

Software quality management system such as GESIP [23] was extended to support Personal Software Process. This support includes the time logging for PSP activities. The second module was developed to support the software size information. Third module was added to manage project defects. The tool supports the automatic numeration of defects and creates links between these defects. The historical

information about the quality of the project can be determined using the module. Tool provides the secure login for software engineers. The access to resources was maintained using private key. The tool also provides the support for group work on the projects. They developed and implemented tool in ISO 9000 certified company. The company has established process based improvement. It has also achieved Silver Q which is the grants for achieving 4000 points in the European Foundation for Quality Management.

3.7.9 JTemporal API Based Tool

The tool discussed in [24] support Personal Software Process with focus on problem of context switching and recording overhead faced by the users while recording time log. The tool used the speech sensor in order to record activities. The tool provides the facilities to record start and end time during the particular activity. Microsoft speech recognition API was used in the tool to recognize the speech of user for recording start and end time. Activity duration estimation was performed using additional sourced of information such as information automatically obtained from secondary sensor and information obtained from schedule stored in database. The tool uses sixty rules and seven algorithms to record start and end time of an activity automatically.

3.7.10 Open Process Components

The research work on *"Open Process Component (OPC)"* [25] explains the application of component based technique to process modeling. The proposed work describes how open process component approach can be applied to Personal Software Process. The proposed component based framework support process activities such as building of process models its execution and analysis. The framework is proposed to study the process technology can be used to construct processes, its execution and analysis. The OPC provides a distributed process components environment. In this way process components are located and adapted relative to the process. These components can also be optimized for such process. OPC provides a process enabled environment where distributed components can interact. The OPC based tool for PSP was implemented in Java. The processes can be viewed as dynamically interacting components. These components are reused in process model. The distributed,

dynamic and reusable process components may be created by OPC. Processes such as PSP and ISPW-6 have been implemented using OPC.

3.8 PSP INTEGRATION AND AUTOMATION

Six research publications are discussed here which proposed integration of other process with PSP and automated these processes.

3.8.1 SSPMT

The research work [26] proposed a framework for software process improvement. The framework is based on the integration of Six Sigma with PSP for software process improvement. Six Sigma DMAIC (Define, Measure, Analyze, Improve, and Control) methodology is used for software process improvement. The framework is supported by a web based tool *"Six Sigma Project Management Tool (SSPMT)"*. The tool has integration with software project management tool and PSP process tool. They introduced a conceptual model for software Six Sigma for software process improvement.

3.8.2 PSP-Six Sigma

The proposed work on PSP-Six Sigma integration [27] provides a mapping table by which PSP features are enabled by Six Sigma. They provided a relationship between PSP and Six Sigma tools. Statistical problems such as determination of 2-sample t-test can be done using these tools. They proposed an approach for continuous process improvement. The improvement is based on Six Sigma based reduction of variation, focus on consistency and high product quality. The process improvement and its acceleration with PSP were discussed.

3.8.3 Mercury

The research work [28] proposed agent based process management model. The knowledge management system provides the process management of knowledge workers and service workers. They tried to improve the process execution accuracy by the application of process visualization and standardization. They integrated agents in their PSP system which provides accurate guideline in process execution. The PSP agent provides the process execution data where as Advisor agent provides the

guideline on project execution. The process areas are improved by collecting data regarding competency of individuals and subject of the process execution. The agents in Mercury system behaves like a program which can automatically execute activity for the users. These agents are intelligent, mobile, autonomous and collaborative in nature. The agent based system is designed by integrating PSP agents with Six-Sigma DMAIC methodology. The user identity management is established in the system. It loads the required process from process repository and executes it for the specific user. The system is designed for small team of individual engineers or managers. The system was proposed to manage process accurately, and to control the project schedule and cost.

3.8.4 Six Sigma and PSP based Process Management

Six Sigma and PSP based approach was proposed to support business process management in [29]. The proposed technique was considered to be helpful for process operation. The technique also was considered helpful for the identification of areas for process improvement. They used the technique for the schedule management. The time data collected was used for identifying areas for process improvement. The process management technique is designed for process definition, process execution, and process measurement. The overall scope of this technique was business process of a company. The business interests such as valued customers, product quality and services provided by the organization are directly related with revenue. They consider an organization can maximize return on investment by using well-defined processes. They also proposed tools for their process management technique. These tools include schedule management tools, schedule data measurement tools and process definition tools.

3.8.5 Process Platform

The *"Process Platform"* [30] proposed to improve software process. It focuses on software process improvement models such as CMMI (Capability Maturity Model Integration) and "*SPICE (Software Process Improvement and Capability dEtermination)*". They integrated Six Sigma and PSP/TSP within their proposed process platform. They also utilized the 6-Sigma tools for process improvement within process platform. The process improvement environment so called Process

Platform was discussed for process definition, process execution, and process measure. The process data and service data for a process is identified by process platform. The architecture of process platform has features such as integrated environment for defining, executing and improving process. The tested processes are presented to user for knowledge based environment. The software process operation is provided for each process maturity level. The conceptual model of the process platform has process instances, process applications. The process framework contains CMMI and SPICE. The process repository is used for making decisions. Message handlers and message brokers are used in process middleware. Finally the process OS contains process such as PSP and TSP.

3.8.6 OPC for Group

The research work described in [31] proposed personal software process for group projects. The process supports was automated with *"Open Process Component (OPC)"*. The OPC were developed at *Arizona State University*. The OPC is a toolset developed for process definition and enactment. They developed a tool using OPC with the support of planning and post-mortem. The integrated process consists of *"Planning Process Components"*, *"Personal Software Activities Components"* and finally the *"Post-mortem Components"*.

3.9 PSP MODIFICATION

Total of three publications were found on PSP modification. These research articles are discussed in this section one by one.

3.9.1 PSP Control Theory

PSP was combined with control theory to analyze the PSP evolution process, discovery of critical factors, and to improve performance. It provides a feedback control for training adjustment and feed forward training requirement for new engineer [32].

3.9.2 B-PSP

The PSP modified process with B-method was proposed to achieve high-quality and reliable software on time and within budget [33]. The process introduces the

formal specification approach in the early stage. The developed defects type standard related with formal specification based state machine.

3.9.3 Collaborative Software Process

The practices of pair programming are integrated with Personal Software Process [34]. The resulting *"collaborative software process (CSP)"* leverages the power of two programmers. The process is defined and repeatable for any two programmers. The process is supported by set of process scripts, forms and templates to ensuring completeness. It provides the measurement-based feedback for measuring progress of programmers. CSP introduces different levels for improving skills. The level 0 introduces collaborative baseline, level 1 introduces collaborative quality management, and finally the level 2 introduces collaborative project management activities.

3.10 PSP MODIFICATION AND AUTOMATION

This section is designed to address the proposed processes which are modified version of the PSP and also provides the automation support. Seven publications are classified in this section. However, out of these seven publications only five addressed process in the right context. The remaining two publications are not discussed here because of the research context.

3.10.1 The Team PSP

The Team PSP process [35] is designed for team of PSP engineers. They designed Team PSP based on process model with measurement goals and related measures. They introduced *"Team PSP0"* where team effort is aggregated. *"Team PSP1"* introduces planning and tracking procedures for individual and for project team. Finally *"Team PSP2"* introduces three events compilation of modules, integration of components, and the release of software product. The process is supported with a tool TT for managing process. The tool provides privacy of data for individual engineers, data storage and analysis, and problem tacking. The TT tool is developed in Visual Basic with supported database.

3.10.2 VDM over PSP

The VDM over PSP process is proposed to introduce the *"Vienna Development Method (VDM)"* to the PSP engineers [36]. The process tool is supported by a software tool VDM-SL Toolbox. They applied VDM over all of the PSP levels. However, VDM over PSP uses PSP2.1 as baseline. The VDM-SL syntax review, Type checking, and Validation introduced by VDM over PSP is supported by software tool.

3.10.3 PPMP

Personal Software Engineering Project Management Process (PPMP) is modified process with more emphasize on project management activities with tool support [37]. The process is designed for large team of software engineers. The focus of the process is personal project management. It can be used at higher levels of PSP 2.0 through PSP3. The data gathering and analysis is carried out similar to the PSP. The application of methods such as PROBE, regression, prediction interval and multiple linear regressions is same as in PSP. It introduces the concept of *"Personal Writing Process (PWP)"*, a process support for writing reports.

3.10.4 Reflective Software Engineering

The *"Reflective Software Engineering (RSE)"* process is proposed in [38]. It is based on PSP basic idea and is supported by a software tool. The *"Leap toolkit"* is designed for process automation. The toolkit is designed using Java language. The toolkit is light-weight and does not introduce new substantial effort. The software process is empirical in nature, and address measurement dysfunction. The tool proposed for the process is portable and can be deployed in different environments.

3.10.5 PSP.NET

The proposed process is designed to support collaborative sharing of defects information [39]. It introduces the concept of anti-freezing of process definition in the process. The research work presented tool to support personal software process. The tool is proposed to address the overhead in data collection, excessive use of manual work, freezing of process definition, and finally the privacy issues. The tool not only supports standard PSP but software engineer can define his own process.

3.11 TSP MODIFICATION

In this section those publications are discussed which addressed the TSP modification.

3.11.1 ACE

ATAM along with QAW, ADD, V&B, and ARID form a systematic process and is called Architecture-Centric Engineering (ACE) [8]. The ACE is recently combined with Team Software Process (TSP) which provides an accelerated process with focus on quality early in the architecture phase. However, the current implementation of architecture design and evaluation method is for TSP coaches or managers and not for PSP engineers.

3.11.2 TTSP

Another study [40] which proposed a modified Team Software Process to support PSP introduced two team processes *"Tailored TSP (TTSP)"* and *"Level TSP (LTSP)"*. Both TTSP and LTSP do not provide support for architecture design and evaluation.

3.12 COMMERCIAL TOOL

HP-Quality Center which was formerly known as *"HP TestDirector for Quality Center"* is a commercial available software [72]. HP-QC is designed for managing software quality. It is industry leading quality management system available in J2EE and .Net. The system is designed to support Oracle and MS SQL Server databases. The main modules of HP-QC are requirement module, release and cycle module, test plan module, test lab module, defects management modules, dashboard reporting module. The *"HP Quality Center Premier Edition"* is advanced version of HP-QC family and have many advance features.

3.13 ANALYSIS AND RESULTS

The last phase of the literature review was involved the analysis of the selected publication. Two type of analysis was conducted one for process support and other for process automation support. A comparative analysis is given in table 3.1.

TABLE 3.1
Comparative Analysis

Reference	Abbreviation	Research Purpose	Commercial Purpose	Tool Support				Process Support					
				Technology	Database	Open Source	Requirement Traceability	PSP	QAW	ADD	Views and Beyond	ATAM	CBAM
	FSQMS	Y	Y	JSF	Oracle	Y	Y	Y	Y	Y	Y	Y	Y
[16]	PSP-DROPS	Y	N	CGI	Y			Y	N	N	N	N	N
[17]	Hackstat	Y	N	SOAP	XML	Y		Y	N	N	N	N	N
[18]	PROM	Y	N	Java	Y	Y		Y	N	N	N	N	N
[19]	PSPA	Y	N	Net(C#)	MySQL	Y		Y	N	N	N	N	N
[20]	DuoTracker	Y	N	Java				Y	N	N	N	N	N
[21]	PSP-EVA	Y	N	PHP	Y			Y	N	N	N	N	N
[22]	PSP-EAT	Y	N	Excel	Y	Y		Y	N	N	N	N	N
[23]	E-GESIP	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[24]	JTemp	Y		Net(C#)	MySQL			Y	N	N	N	N	N
[25]	OPCTool	Y	N	Java				Y	N	N	N	N	N
[26]	SSPMT	Y	N	HTTP				Y	N	N	N	N	N
[27]	PSP-Six Sigma-1	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[28]	Mercury	Y	N					Y	N	N	N	N	N
[29]	PSP-Six Sigma-2	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[30]	Process Platform	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[31]	OPC Group	Y	N	Java				Y	N	N	N	N	N
[32]	PSP Control Theory	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[33]	B-PSP	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[34]	CSP	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[35]	TT	Y	N	VB	Y			Y	N	N	N	N	N
[36]	VDM-PSP	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[37]	PPMP	Y	N					Y	N	N	N	N	N
[38]	RSE (Leap)	Y	N	Java				Y	N	N	N	N	N
[39]	PSP.NET	Y	N	HTTP	MySQL			Y	N	N	N	N	N
[8]	ACE	Y	Y	N/A	N/A	N/A	N/A	Y	Y	Y	Y	Y	N
[40]	TTSP	Y	N	N/A	N/A	N/A	N/A	Y	N	N	N	N	N
[72]	HP-QC	N	Y	J2EE/ .Net	Oracle, MS SQL Server	N	Y	N	N	N	N	N	N

3.14 CONCLUSION

The analysis and results discussed in previous section shows that no publication addressed the integration of process components for architecture design and evaluation. The personal integrated process (PIP) which is the proposed process has all components for software architecture design and evaluation along with PSP. The literature survey also shed light on the available publication on PSP modification. Only 30 publications were found in this regard. Therefore, more research on PSP modification or integration with other processes can produce high quality, disciplined, repeatable, systematic processes. These processes if focus on domain and technology can produce process that will be able to estimate project cost more accurately.

CHAPTER 4
LITERATURE SURVEY ON SOFTWARE ARCHITECTURE DESIGN AND EVALUATION

4.1 THE GAP IDENTIFICATION

To find the gap in research a literature survey was conducted. Software engineering survey [14] presents the comprehensive coverage of research literature from 1972 to 2002. There is another study [79] which proposed framework for classifying and comparing software architecture evaluation methods. The study presents the comparison of various evaluation methods from 1976 to 2004. The extended form of the framework is given in [80]. The comparison of these two frameworks can be found in [81]. Since large number of software system evaluation methods have been proposed after the publication of above mentioned surveys this requires the need for a fresh survey. Considering institutional requirement survey sources were selected. Keywords and index terms were used to find the research articles from these research sources. Survey sources, index terms, and quality criteria for research articles selection are discussed below.

4.2 SURVEY SOURCES

The survey sources include digital libraries, scientific databases, publishers, search engines, and references/bibliography. These survey sources are given below: IEEExplore Digital Library; ACM Digital Library; ScienceDirect; Web of Science; British Computer Society; CiteSeer Digital Library; Wiley InterScience; Springer Verlag; Springer Science + Business Media; Kluwer Academic Publishers; Elsevier; IEEE Computer Society Digital Library; and Google.

4.3 INDEX TERMS & KEYWORDS

The keywords selected from research question were "Software Architecture", "Architecture Evaluation Processes" and "Architecture Design Processes".

4.4 SEARCH STRING

The search string was formulated based on keyword. The research stings used for search are given below.

String-1: Software Architecture Evaluation Process.

String-2: Software Architecture Design Process.

4.5 STUDY SELECTION

These search strings were used for searching publications from selected search sources. In study selection process publication title, abstract, keyword or references were considered. The total publications that were selected in phase-1 were 155. However, more than thousand publications are available on architecture design and evaluation. The limited numbers of publications are due to accessibility to the digital libraries, and institutional requirements. Other factors are related with research project cost and time.

Inclusion Criteria: The research work published in journals, conferences, workshops, and symposiums were selected for review. The copyright technical reports, books and doctoral dissertations were also selected for review.

Exclusion Criteria: White papers, technical reports, books and website articles without copyrights were not considered.

The total selected research papers after applying inclusion and exclusion criteria in phase-2 were 76.

In phase-3 the total selected research papers for detailed study were 76. The study includes those publications which fulfilled the inclusion and exclusion criteria. These articles were studied completely and selected for analysis. In phase-4 the total research papers selected for analysis were 32. Fig.4.1 shows the results of literature survey 155 publications were selected in first phase, 76 were selected in second phase and third phase, and finally 32 were selected in forth phase.

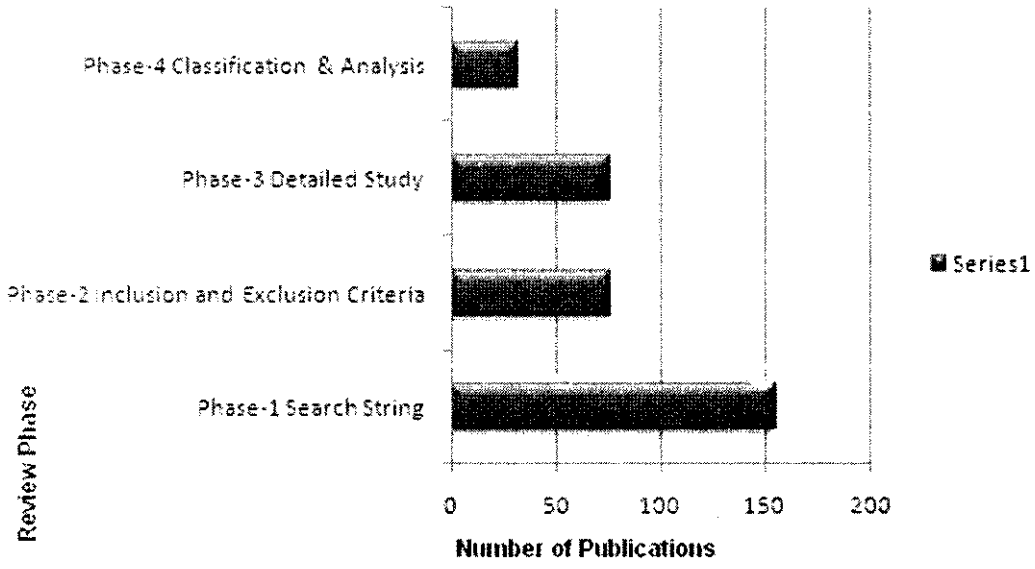


Fig.4.1. Literature Survey Results

4.6 CLASSIFICATION AND ANALYSIS

The analysis was carried out after detailed study of 32 publications. Out of these 32 publications seven publications were found to be related with architecture design process, twelve publications were found to be related with architecture evaluation process, and thirteen were found to be related with architecture design and evaluation. Fig.4.2 shows the results of review in graphical format.

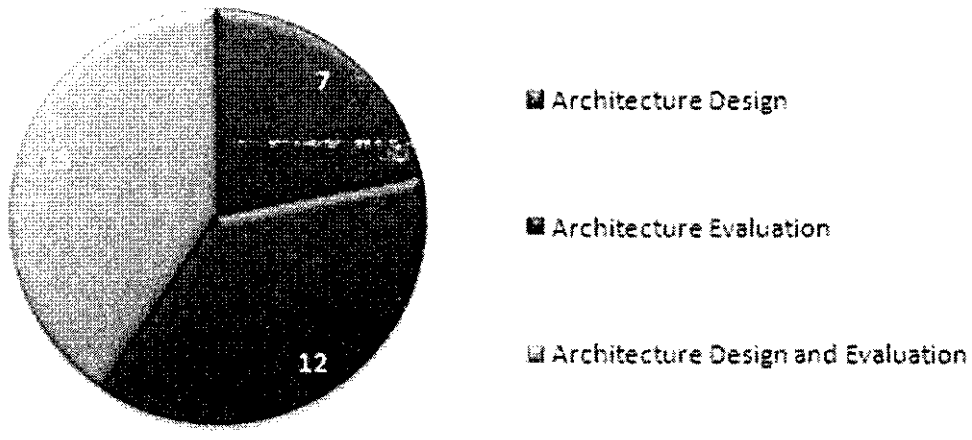


Fig.4.2. Classification of Publications

4.7 ARCHITECTURE DESIGN PROCESSES

This section contains the research publications related with architecture design process, model or methods. Total of 16 publications were found to be related with architecture design process, model or method. These research publications are discussed in this section.

4.7.1 Rational Design Process

The *"Rational Design Process (RDP)"* [42] was discussed to derived program systematically from precise requirements. A rational process has number of benefits such as the understanding of the process provides designer how to proceed, easier to adapt the process, design better and backtrack less. It becomes easier to transfer people and software from one project to another project. Project progress can be measured and review can be conducted easily. The main steps of RDP are explained below.

1- Establish and Document Requirements

The requirements are elicited and documented in this step that help designer to design the module structure. The requirement document contains complete information to write software. The requirement statement should be valid. The requirement document is completed in this phase. However, incomplete requirements are identified and marked. The requirement document is organized as a reference document and this saves labour. The requirement document is written by software developer and approved from the user representatives. The mathematical models are used in requirement specifications. The requirement document is organized in such a way to ensure the separation of concerns which helps in achievement of completeness of requirements. The major sections that a requirement document should have are computer specification section with information regarding specification of the computer system. The *"Input/ Output Interface Section"* contain the information regarding specification of the interfaces for communication with input/ output devices. *"Output Value Specification Section"* contains the information about state and history for each output. *"Timing Constrains Section"* provides timing information for each output. *"Accuracy Constraints Section"* contains information of accuracy

level for each output. *"Likely Change Section"* is designed for programmers to decide the changes that are most likely. *"Undesired Event Handling Section"* contains information about those events that may occur.

2- Design and Document the Module Structure

In this step the module guide is produced which contains the information about work assignments and design decisions that reflects the responsibilities of particular module. The module guide helps in achievement of separation of concerns and maintenance of defected module. The module guide is arranged in tree structure with information about all modules and sub-modules of the system.

3- Design and Document the Module Interfaces

The module interface specifications are written for each module in this step in such as way that it provides a formal representation of the module interfaces. The specification document includes information of programs that can be invoked by the module, the parameters information, timing constrains and accuracy constraints.

4- Design and Document the Uses Hierarchy

In this step the uses hierarch of the modules is designed in binary matrix form which contains the information of modules and their access programs.

5- Design and Document the Module Internal Structures

The module internal structure may contain process that is designed and documented in this step. The internal data structure is designed and documented for each access program or function. The module return values and its associated mathematical functions are provided.

6- Programming:

The coding activity is performed for a specific design of module.

7- Maintenance:

During maintenance if a design document is changed all other concerning documents are updated.

4.7.2 Feature Oriented Reuse Method

The *"Feature-Oriented Reuse Method (FORM)"* [44] is the extension of *"Feature-Oriented Domain Analysis (FODA)"* method developed at Software Engineering Institute Carnegie Mellon University. FODA uses feature model for requirement engineering. However, FORM addresses the feature model for software

design phase for development of domain architecture with reusable components. To design domain architecture with reusable components commonalities and differences across related software systems are discovered systematically. In FORM reusable architectures and components are developed using process called *"domain engineering"*. In domain engineering the method analyzes the commonalities in terms of services, operating environment, domain technologies, and implementation technologies. These commonalities are captured as features of product and are arranged in the form of AND/OR graph. The feature model is constructed in this way from this graph analysis. The domain architecture thus constructed from three viewpoints. Subsystem viewpoint is used to capture service features, Process viewpoint is used to capture operating environment features, and module viewpoint is used to capture feature associated with domain technology and implementation techniques. The FORM was applied on electronic bulletin board system (EBBS) and private branch exchange (PBX) domains to investigate development of common domain language for developers and feature model for identifying reusable components. The feature model use term features as characteristics of product that can be implemented, tested, and maintained. These features of product are considered as first class objects for development.

4.7.3 Architecture Based Design Method

The *"Architecture Based Design (ABD)"* [48] method is developed by scientists of Software Engineering Institute and Robert Bosch GmbH for designing high level software architecture for software product line engineering. The ABD method is supported by Rational Rose tool for recording decisions made during the ABD method execution. The input to the ABD method is abstract functional requirements; concrete functional requirements in the form of use cases; abstract quality and business requirements; concrete quality and business requirements in the form of quality scenarios; architecture options; and constraints for initial conceptual architecture. The requirement elicitation goes in parallel during conceptual architecture design activity. This conceptual architecture represents first design choice for developing a product. Decomposition of function; realization of quality and business requirements; and software templates are the foundations of architecture based design method. *"Logical View"*, *"Deployment View"* and *"Concurrency View"*

are used for functional decomposition. The output of the ABD method is abstract components in logical view, deployment view and concurrency view. Software templates; constraints; and concrete requirements are also the output of the ABD method. Based on these outputs concrete component design is prepared. The ABD method is used to decompose the system and to capture application portion and infrastructure portion of the system. The system is first decomposed into conceptual subsystems with associated subsystem templates. The responsibilities are allocated to these conceptual subsystems according to logical view, deployment view and concurrency view. These conceptual subsystems are then decomposed into conceptual components along with its component template. These conceptual components are then considered for responsibility using logical view, concurrency view and deployment view. The next step is to decompose the conceptual components into concrete components which may correspond to software element such as class. This step produces "design elements" with their relationship. However, the order of generation of these design elements is based on some consideration. These considerations are architect's knowledge of the domain; incorporation of new technology; and personnel experience in particular portion of the architecture. The design elements are decomposed into children design elements with associated set of requirements; templates; and constraints. The sequence and steps involved in decomposing design elements are: *"Define Logical View"*; *"Define Concurrency View"*; *"Define Deployment View"* and a feedback of verification of quality scenarios and constraints.

4.7.4 Quality Attribute Workshop

The *"Quality Attribute Workshop (QAW)"* [50] is an eight steps method for elicitation, identification and refinement of quality attribute of software intensive system. It is system-centric and its main focus is system stakeholder for eliciting requirements regarding driving quality attributes of software architecture. The QAW engages systems stakeholders for better communication before the creation of the software architecture. The QAW has the ability to complement ATAM for analyzing architecture for tradeoffs points and sensitivity points. The method is not designed to provide absolute measure of quality. However, it provides a systematic way of elicitation, documentation, and prioritization of quality attributes. The system

engineer use these prioritized scenarios for analyzing architecture. It is their responsibility to prepare risk mitigation strategy and identify quality attribute concerns. QAW is a scenario based method and describes each scenario in term of stimulus, response and environment from. "*Stimulus*" is a factor that causes system to initiates, and reaction of this system is "*response*". QAW is designed to address the challenges such as: precise meaning of quality attributes; elicit, identify, prioritize quality attributes; engaging system stakeholders in disciplined and repeatable process; processing and utilizing this information. The basic concern of QAW is system-level for identifying and prioritizing quality attributes. The QAW stakeholders group may range from five to thirty, during which they receive "*participant handbook*". The workshop requires focus and active participation of stakeholders.

Step-1: QAW Presentation

In this step facilitators and stakeholders have brief introduction about their role and responsibilities in the organization. Facilitator then presents standard slide presentation of QAW for the purpose of motivation, and to explain the steps involved in the method.

Step-2: Present Business and/or Mission

Facilitator in this step note the key quality attributes drivers during management presentation. In this step the management representative presents business concerns and/ or mission concerns of the system along with functional requirements, constraints and quality attribute.

Step-3: Present Architectural Plan

In this step a technical stakeholder presents initial high-level system description and system context diagram. The technical representative presents architectural plan and strategies for meeting business and or mission requirements. While technical requirements and constraints of the system are presented, facilitator captures information about architectural drivers.

Step-4: Identify Architectural Drivers

To identify architectural driver facilitator captures information regarding functional requirements, business concerns, mission concerns, goals, objectives, and system quality attributes. This information is transformed into list of key architectural drivers. The stakeholders then distilled the list of architectural drivers by some addition or deletion. This final list of distilled and key architectural drivers is used during subsequent brainstorming section.

Step-5: Scenario Brainstorming Process

Scenario brainstorming process is initiated by the facilitator. Facilitator reviews the generated scenarios by the stakeholders. During two round-robin passes of QAW at least two scenarios are contributed by the stakeholders. The facilitator ensures that collected scenarios are well formed and represented in the form of stimulus, response and environment.

Step-6: Consolidate Scenario

Similar scenarios are merged if facilitator finds that similar scenarios will not contribute anything.

Step-7: Prioritize Scenario

Consolidated scenarios are prioritized by the system stakeholders. This prioritization of scenarios is based on voting activity which is done in round robin two passes. The number of votes determines the priority of the scenarios.

Step-8: Scenario Refinement

In scenario refinement process top five scenarios are refined and documented in the form of stimulus; response; source of stimulus; environment; artifact stimulated; and response measure for the particular scenario.

4.7.5 Methodological Architectural Design

The *"methodological architectural design (MAD)"* process [51] is designed to address the limitations of Attribute-Driven Design (ADD) method such as making design decision that satisfy the driving requirements. The process provides the reasoning framework for making design decision based on accuracy of quality attribute model. Since such models are available for performance so the method can be used for it. The process helps in coupling one quality attribute requirements to decision regarding architecture with objectives of providing analysis support to designer for precise quality attribute based design. The work demonstrates the systematic relationship between concrete scenarios, general scenarios, design fragments and architectural tactics. Concrete scenarios are instances of general scenarios and are system specific quality attribute requirements. General scenario which is the precise system independent specification of quality attribute consists of a stimulus; a response; a source of the stimulus; an environment; a stimulated artifact and a response measure. Tactic or architectural tactic is the architectural design

decision used to control a quality attribute response measure through the use of quality attribute model. Thereby architectural tactic bridge the quality attribute model and the architectural design. It represents codified knowledge of relationship between decision of architecture and quality attribute parameters. To create quality attribute model knowledge of reasoning frameworks are used. This process provides a set of relations for moving from each scenario, through tactics, to design fragments i.e. it provides a way for deriving design fragments from concrete scenarios. These set of relations are: Concrete scenarios are instances of general scenarios; Quality attribute parameters are contains in general scenarios; Reasoning framework comprise a set of dependent parameters and independent parameters of quality attributes along with their relationships; Quality attribute model is an instance of quality attribute reasoning framework; finally reasoning framework rules and architectural design rules together form the tactics.

The first step of the method emphasize for picking a concrete scenarios. In second step it is verified that the concrete scenario is well formed and have all necessary elements of a general scenario. The general scenario generation table can be used to verify the concrete scenario. The other activity of this step is to identify responsibilities from elements of the concrete scenario. In third step candidate reasoning framework is identified by eliminating the unwanted reasoning framework by using concrete scenario information. The association between reasoning framework and general scenarios also help in identification of candidate reasoning framework. The set of bound parameters that reflects decisions already been made and set of free parameters that reflects decisions yet to be made are determined in forth step. These parameters can be found from candidate reasoning framework, or from concrete scenarios. However, dependent parameters are not considered. The fifth step is to determine the tactics associated with the free parameters. In sixth step initial set of values are assigned to the free parameters determined in the last step. The value is assigned to free parameters based on the knowledge of designer. The value may be arbitrarily chosen or based on prototype implementations. During seventh step architectural tactics are selected for application of bind values for the achievement of quality attribute response measure. These binding must be determined because of the availability of multiple free parameters. In case if there is a single concrete scenario with one free parameter and one modeling framework. The candidate tactic is relevant or not is determined by adjusting the value of the free parameter to a new value.

However, for multiple free parameters simultaneous adjustment of all free parameter is required. Finally in the eighth step responsibility is allocated to the architectural elements of the design fragments.

4.7.6 Attribute Driven Design

The *"Attribute-Driven Design (ADD)"* process [52] developed by Carnegie Mellon University Software Engineering Institute is a software architecture design process based on software quality attributes. The ADD method consists of eight steps for decomposing system by applying architectural tactics and patterns. The inputs to ADD method are prioritized functional requirements; design constraints; and quality attribute. These inputs are referred to as requirements of the system that an architect address in the ADD process. System design is the output of ADD method. The system design addresses software elements; their responsibilities; roles; properties of software elements; and relationship among software elements. This system design is documented using appropriate architectural views and refined using eight steps which are explained below.

Step-1: The prioritized functional requirements based on business and mission goals are documented along with the impact of these requirements on the system architecture. The prioritization of requirements is performed in consultation with the stakeholders. The architect makes sure that the information about quality attribute is sufficient and is expressed in term of *"stimulus-response"*. These measurable qualities attribute should be expressed in the six parts: *"Stimulus Source"*; *"Stimulus"*; *"Artifact"*; *"Environment"*; *"Response"*; and *"Response Measure"*. Design patterns and architectural tactics are applied on the system architecture based on the information of quality attributes. These quality attribute scenarios are documented and refined based on availability of architectural information. However, no design decisions are made at this stage.

Step-2: In this step architect select a design element to decompose it. Since it is a loop step, an architect can enter in this step in two ways: First when the system is not decomposed and he is going to decompose it first time. In this case the design element to decompose is the whole system. Second when the system is decomposed and partially designed. In this case decomposed designed element will be the focus of subsequent steps. In second case the major concerns of architect may be knowledge of

architect; difficulty in achievement of requirement; risk involved; business criteria; and organizational criteria.

Step-3: In this step candidate architectural drivers are identified. This identification is done with the help of ranking again the requirement with their relative impact on the architecture. The second ranking is performed by assigning "High", "Medium", or "Low" impact to architectural significant requirements. This ranking help in dividing requirement into number of groups based on importance of requirements to stakeholders and impact of requirement on the architecture. The final selected requirements with indicated importance of stakeholders, but may or may not have impact on architecture are called *"Candidate Architectural Drivers"*. Those requirements that have impact on the architecture are referred to as *"Architectural Drivers"*.

Step-4: In fourth step architect select a "*design concept*" for architectural drivers. Major types of design element and types of relationships based on design constraints and quality attribute requirements are selected. This is carried out in six steps: A- Identification of design concerns; B- Creation of list of alternative patterns for selected design concern; C- Selection of appropriate pattern that satisfy candidate architectural drivers. D-Based on design decision determine relationship between design element for selection of a pattern or combination of patterns. E- Using architectural view describe the selected patterns. F- Evaluate the design for selected architectural drivers and resolve inconsistencies in the design concept. In this step some design decisions are made which includes, selection of design concept with major types of elements and types of relationships; Identification of functionality associated with elements; Software elements mapping; Communication among elements; Resource allocation to elements; and dependencies between various types of elements.

Step-5: In this step software architectural elements are instantiated and responsibilities are assigned based on their types. The parent architectural element's requirements are expressed in sequence for child element.

Step-6: In this step *"Interfaces"* are defined for each instantiated element. The interfaces are basically services and properties *"Required"* and "*Provided"* by the software architectural element.

Step-7: In this step functional requirements, design constraints and quality attributes are verified and refined for decomposed instantiated elements.

Step-8: This step provides a loop back to step 2 for decomposing design element and for refining it.

4.7.7 Architecture Rationale and Elements Linkage

The *"Architecture Rationale and Elements Linkage (AREL)"* model [55] is developed for architecture design rationale modeling. The AREL model is validated by a case study conducted on Electronic Fund Transfer System (EFT) at People's Bank of China Guangzhou (PBC-GZ) branch. The model captures relationship between *"Architecture Rationale (AR)"* and *"Architecture Elements (AE)"*. The AREL model uses entity-relation diagram to model AR and AE entities. The modeling support is provided using UML. The AREL provides an automated support for design reasoning traceability using Enterprise Architect. The traceability techniques in the AREL model are used for impact analysis and root-cause analysis. A traceable design rationale is used to trace relationships between the design objects. The rationale-based architecture model was designed to address issues such as conflicts, inconsistencies, and omissions in architecture design because of the absence of design rationale. The reasons behind architecture design decisions are captured using design rationale. The design rationale if capture and trace appropriately can help in understanding architecture design during verification and maintenance of large systems. Therefore, AREL was introduced to support architecture design rationale capture. The architecture design rationale is captured using quantitative design rationale and qualitative design rationale. Any arguments regarding design alternative whether in favour or against is captured using qualitative design rationale. However, for capturing the quantitative design rational cost, benefit and risks associated with the design alternative are quantified.

The AREL model considers two forms of design reasoning one is *"motivational reasons"* and other is *"design rationale"*. Requirements, goals, constraints or design objects are considered as motivational reasons as they motivates in design. The AREL model is the implementation of *"conceptual model"* of architecture rationale. The main entities of conceptual model are *"Architecture Rationale"*; *"Motivational Reason"*; and *"Design Outcome"*. The actual implementation of AREL conceptual model is represented using UML notation. The architecture element is represented by AE. When AE participates in a decision as an input it is called motivational reason.

Architecture Elements: The architecture element is an artifact concerning business requirements, technical constraints or assumptions about architecture design. The architecture design elements are produced through architecture design process. These architecture elements are classified in a set of related concerns. These concerns are modeled using business viewpoints, data viewpoints, application viewpoints, and technology viewpoints in IEEE standard 1471 format. Functional requirements, non-functional requirements, information system environment, business environment, and technology environment are contained in business viewpoint. They act as architecture drivers of the architecture design. Data models are contained in data viewpoint. Application models contained in application viewpoints and technology models are contained in technology viewpoint. Data viewpoint are used to represent data being used by an application. Application viewpoint are used for processing logic and structure. Technology viewpoint represents technology used to implement system. Requirements, assumptions, constraints, and design objects are motivational reasons and act as architecture elements.

Architecture Rationale: Three justifications *"Qualitative Rationale (QLR)"*, *"Quantitative Rationale (QNR)"*, and *"Alternative Architecture Rationale (AAR)"* are considered in architecture rationale. The architecture rationale AR captured in AREL model simplified process and it only capture the justifications of the decisions. The design rationale is encapsulated in the architecture rationale. The direct dependencies between decisions in a chain and architecture elements are used to understand design reasoning. Qualitative rationale contained information regarding issue of the decision; design assumptions; design constraints; strengths and weaknesses of a design; tradeoffs involved; risks involved in design option; assessment and decisions; and supporting information. This information provides qualitative rationale for decision. However, quantitative rationales are not considered in AREL. Finally, the alternative architecture rationale AAR contains "alternative behaviour" and "alternative design". The AREL model is extended to adapt the evolution. The extended model that can capture the evolution history is eAREL. The model provides three types of traceabilities "forward traceability", "backward traceability" and "evolution traceability". The impact analysis of the design is provided through forward trace. The root-cause analysis is performed using backward trace. Finally the analysis of the evolution of a decision is provided by evolution trace. These traceabilities are automated with Enterprise Architect which is a UML design tool.

4.8 ARCHITECTURE EVALUATION PROCESS

The research publications proposed architecture evaluation process are discussed in this section.

4.8.1 SAAM

The *"Software Architecture Analysis Method (SAAM)"* [7], [57] is a five-step method for analyzing software architectures. This method accompanied with an architectural description language which was used to analyze three competing architectures with respect to modifiability. This language describes the structural perspective of the competing architectures. With this language one can describe architecture consistently, having a common level of understanding for architecture comparison. It is a practical and proven method and applied for examining architectures of user interface portion of interactive system. There are two important things about SAAM, i.e. first it uses a common vocabulary for analyzing architectures. Second it not only concentrate on functional features of architecture but also the quality concerns within the system software life cycle. It uses three perspectives for understanding and describing architectures which are functionality, structure, and allocation.

In first activity characterization of canonical functional partition for a given domain is considered. The second activity involves mapping of this partition over the structural decomposition of architecture. The third activity involves selection of quality attribute for architecture assessment. The forth activity involves identify set of concrete tasks which test the selected quality attribute. Finally the fifth activity involves the evaluation of architecture for these tasks. SAAM is an abstract evaluation method for architecture for its functional and non-functional quality requirements. It is not metric based evaluation method instead it is a qualitative evaluation of software architecture. It uses three perspectives for describing software architecture functional partitioning, its structure, and allocation. The functionality or behaviour of system may be based on single function or set of functions. This functionality can be decomposed using *"Structured Analysis"*, *"Object Oriented Analysis"*, or *"Domain Analysis"*. For component based software a component may be consider the smallest unit of the structure. These components communicate with each other and have control

relationship. SAAM used graphical language for these component and connections. In a domain allocation choices are used to differentiate architecture because these choices identify how domain functionality is realized in software structure. Modifiability was the key quality attribute used in SAAM case study. However, it can be used for other quality attributes. This quality attribute was chosen because SAAM used *"Evolutionary Development Life Cycle"* at Carnegie Mellon University and these projects has life time between 20 to 30 years. Project with long life modifiability is the paramount important. This includes adaptation to new operating environments and extensions of capabilities.

4.8.2 SAAER

The *"Software Architecture Analysis for Evolution and Reusability"* [58] (shortly as SAAER) is a framework and a set of architectural views designed to evaluate software architecture for evolution and reuse. It is based on SAAM which is a scenario based approach for software evaluation. This framework consists of four phases *"Gathering"*, *"Modeling"*, *"Analyzing"* and *"Evaluating"*.

In first phase four different set of information is gathered which includes *"Stakeholder Information"*, *"Architecture Information"*, *"Quality Information'* and *"Scenarios"*. However, information categories may be extended. This phase helps in gap analysis i.e. what information is available and what information is required. Stakeholder information provides who are participating and influencing system such as designers, managers, end users etc. Architecture information on the other hand provides information regarding critical design principles, architectural objectives, and architectural views. Architectural views are considered to be important for evolution and reuse. These architectural views help in comparing systems developed using different paradigm such as functional decomposition or object oriented design. Quality information refers to the non-functional requirements such as availability, modifiability, or integrability. Scenarios describe the system's functionality in term of use cases. These scenarios help in detecting possible flaws in the system. Modeling which is the second phase of the framework, information is aligned across information categories and mapped information into usable artifacts. In the modeling phase breadth aspect describes the relationships between objectives of stakeholders, architecture objectives, quality attributes and scenarios. The depth aspect deals with

the levels of abstraction and will affect the cost of the analysis. Analyzing is the third phase of the framework in which Software Architecture Analysis Method (SAAM) is used for further analysis of various artifacts generated in the last phase. These artifacts include Domain Models, Architectural Views, Trade-off, Scenarios, Environmental assumptions and Constraints. Evaluating is the fourth and last phase of the framework in which recommendations are made, risk and their mitigation strategies are suggested, and common reference models are identified. They applied this framework over a large telecommunication call center software system and found that their framework helped in better estimation of cost, schedule, and risk at early stage of software development for evolution and reusability.

4.8.3 E-SAAM

The extended SAAM method [59] is developed to overcome the limitations of SAAM for reuse of architecture knowledge in domain centric development processes. It considers reuse of architecture assets by integrating SAAM in domain centric process. The reusability analysis helps them in determining the degree of variability and the corresponding limitations of reusable architecture. However, the reuse scenarios should be sufficiently concrete. They found the concrete scenarios help to determine flexibility of architecture and its limitation. During reuse of existing architectures their comparison with each other and the effort required to reuse is also considered. Then interaction patterns are applied along with the modification of existing architectural elements. Same set of concrete scenarios are applied on resultant architecture to analyze the adaptation. This ensures that the architecture does not deviate from the initial design principles. An analysis templates is consists of elements such as proto-scenarios, classification hints, and evaluation protocol etc. This template is reused in SAAM analysis. The analysis template helps in reusability of architecture knowledge in the domain-oriented development. These reuses of architecture assets help in reduction of cost of development. Analysis template also facilitates the identification and mitigation of risk of neglecting crucial issues. It also helps in comparing different architectures in a domain. The analysis template together with architecture helps in determining the risky decisions for developing application that will be based on this architecture. In case of architecture-specific analysis templates, proto-scenarios describe the elements of architecture. The domain specific

and project specific scenarios are evaluated separately. However, architecture specific analysis templates increase the expressiveness of the evaluation. This reduces the critical risk of the architecture. It helps in identifying the commonalities and differences of the constructed software. The architecture specific analysis templates and domain specific analysis templates provides similar advantages such as reduction in cost and time of development. To reduce subjectivity it combines analysis templates of architecture-specific, domain-specific, and project-specific scenarios. The E-SAAM method emphasizes the use of domain-specific experience and knowledge to reduce cost and development time. This method helps in comparing the effort required for analysis.

4.8.4 ARID

The *"Active Review for Intermediate Design (ARID)"* [60] is a technical review which is a blend of four approaches. These approaches are *"stakeholder-centric"*, *"scenario-based"*, *"Active Design Review"*, and use of ATAM for *"architecture evaluation"*. This technical review is conducted after the architecture phase and before the start of design specification document. It is conducted for a portion of software architecture. It is one of the reason that such technical review conducted in pre-release stages are effective for discovering errors, inadequacies and inconsistencies in design. ARID was developed to overcome the limitations of ATAM and ADR. ARID adapts strong qualities of these techniques such as active participation from ADR and stakeholder-generated scenarios from ATAM.

ARID review is conducted using ARID *"Review Team"*, *"Lead Designer"*, and *"Reviewers"*. However, the number of stakeholders involved in ARID may vary. The review team consists of *"Facilitator"*, *"Scribe"*, and *"Process Observer"*. Facilitator prepare for the review meeting. Scribe note down inputs of reviewers, and process observer monitor the process for any suggestion to improve process. The lead designer is responsible for presenting design for review. Software engineers are the major stakeholders for the design to be reviewed. They review the design to judge the quality of design such as its adequacy and usability.

There are two phase in ARID technical review. In first phase four activities are involved, where as in second phase five activities are involved. During first step lead designer and facilitator select people for review. In second step design is presented.

This presentation is delivered by designer. In this phase the design is reviewed by facilitator to ask possible question that reviewer may ask. This helps the designer to improve his presentation. In third step designer together with facilitator creates seed scenarios that may be used for actual evaluation. These scenarios are presented for acceptance or rejection from reviewers. During forth step preparation of the review meeting is completed. The preparation involves the distribution of seed scenarios, review agenda, presentation and schedule to reviewers. In fifth step ARID is explained to the participants. During sixth step the lead designer delivers his presentation. The lead designer presents design. The facilitator ensures that only question of factual clarification will be asked. The scribe notes down each such question and prepare a list of potential issues. The designer addresses each critical question before declaring design to be complete.

During brainstorm activity which is the seventh step, seed scenarios are put together with the other scenarios to filter those scenarios that observed to be repeated. The reviewers use their vote to prioritize scenarios. The scenario with high number of votes is used to test the design for usability of the portion of software architecture. In step-8 the review is performed for scenario that receives more votes. The designer does not give any hint in this review. Only reviewers use examples to review the design. The scribe record any issues that may produce any hindrance for non-expert to proceed with design. The review ends if either review time, highest priority scenarios are reviewed or conclusion has been reached. Finally in the ninth step issues raised in previous step are listed and counted. The effectiveness of reviews is determined from participants.

4.8.5 ATAM

The *"Architecture Tradeoff Analysis Method (ATAM)"* [61] is a spiral model of architecture design, it is iterative in its nature and its each iteration is used to reduce risk that could result from competing quality attributes. This method is designed to characterize the interactions of these competing quality attributes and identifies the tradeoff points between such quality attributes. Hence, it provides a framework for comparing multiple architectures in a continual, spiral and iterative way. It is designed for general quality attributes such as performance, security, availability, reliability and so forth. This method helps better communication between all stakeholders early in

the requirement and architecture phase for possible requirement conflicts. ATAM is a structured method that is refined and upgraded form of the Software Architecture Analysis Method (SAAM). SAAM was developed for single attribute. However, ATAM is designed to compare multiple and competing architectures. This method is iteratively, qualitative, and quantitative in nature, and has four phases in iteration. These phases are explained below:

In first phase two steps are executed which are interchangeable. These steps are used to gather stakeholder's attribute based requirements, system usages scenarios, technical and business constraints and environmental details. In second phase various architectural views such as module view, process view, dataflow view and class view may be used. The process view can be used to analyze performance. Any change in architectural views will result in new architecture of the system. The selection of architectural views, scenarios and requirements provides a basis for initial system architecture. The each iteration of this phase results in a refined and competing architectures. These architectures now must be compared and analyze in the subsequent phase for each quality attribute.

In third phase attribute specific analyses are performed. These attribute specific analyses are analyzed in isolation by individual quality attribute experts. In this phase attribute models for quantitative and qualitative analysis is built. Any change in architecture's functionality, structural elements, and coordination model will have strong affect on these models. In this phase two steps are performed. In fifth step sensitivity points are identified. In sixth step trade-off points are identified.

4.8.6 CBAM

The *"Cost Benefit Analysis Method (CBAM)"* [62] decision making process can be conducted in two phases with six steps in either phase 1 or in phase 2. These steps are listed below.

Step-1: To conduct first step of CBAM prioritized list of scenarios form ATAM with high importance are selected. This importance is based on the desired improvement to the architecture. This step also results in selection of architectural strategies for high importance scenarios.

Step-2: In this step benefit evaluation function is calculated for architectural strategies and quality attribute score *"QAscore"* is also calculated in this step.

Step-3: Quality attribute score *"QAscore"* calculated in previous step are used to evaluate the individual architectural strategy (AS). In this step another variable contribution *"Cont"* is evaluated which is used to represent ranking of architectural strategies for each stakeholders. Thus benefit evaluation function "*Benefit (AS_i)*" is computed and finally for each stakeholder *"concordance coefficient"* is calculated.

Step-4: In this step rough estimate of cost and schedule of implementing each architectural strategy is calculated.

Step-5: In fifth step architectural strategy are ranked using *"desirability"* metric.

Step-6: In this step architectural strategies are plotted for corresponding benefits and costs. In this plot each AS surrounds an ellipse of uncertainty region. This helps in selection of AS with high benefits low cost and low uncertainty.

The CBAM process uncovers the optimal set of architectural strategies for all stakeholders with the possible impact of cost, schedule, and benefits with associated uncertainty. It is a repeatable method that stakeholders can use to make architectural choices with better investment decision. Cost and benefit are two business quality attribute that are trade-off in CBAM in addition to the technical quality attributes trade-off in ATAM.

4.8.7 The 4 +1 View Model Extension (4+1VM-E)

The approach proposed in [63] is an extended form of 4 + 1 View Model consisting of 9 major activities divided into three main phases. The first phase is *"Prepare the Evaluation"*, the second phase is *"Execute the Evaluation"* and third phase is *"Complete the Evaluation"*.

Preparation: In first phase of architecture evaluation three major steps are involved. First step is to define the template for requirements, second step is to generate evaluation contract, and third step is to define templates for software architecture. This phase helps in defining and documentation of goals and scope of architecture using 4 + 1 View Model. These views are documented using UML for functional and non-functional requirements analysis. In second step evaluation contract is produced which helps in defining scope and goals of an evaluation for required quality attributes. For this functional requirements are ranked and scope is clarified. Non-functional requirements and their relationship are identified. In third step architecture is documented using 4 + 1 View Model in UML.

Execution: In the second phase of software architecture evaluation execution is performed. The execution is performed in four steps from step-4 through step-7. In the forth step architectural sub-designs are identified from functional requirements. This step consists of three activities identification of use cases, views from quality attributes and sub-design.

In fifth step architectural design decisions are determined. This step consists of two activities, determination of decision variables for design problems. And determination of decision values for design problems. These decision values represent solutions to each problem. In the sixth step rationale for these design decisions is determined. And in the final step of this phase relationship among these architectural decisions is finalized. The output of this step is the tradeoffs among quality attributes and dependencies among the decisions.

Finalize: In third and last phase of the method two steps are performed step-8 and step-9. In the step-8 finalized decisions are grouped based on quality attributes. The step-9 is the last step in which prediction data is generated. The structure of prediction data is organized in four layers *"Qualities"*, *"Rationale"*, *"Architectural design decisions"*, and *"Sub-designs"*. This help in identification of risks and fitness of architecture with respect to quality attributes.

4.8.8 SARA

The *"International Working Group"* on *"Software Architecture Review and Assessment (SARA)"* has developed a process for system architecture review and assessment [64]. The process provides guidelines on which steps to follow for review, question to ask from stakeholders, information to elicit, collect and document. SARA provides guidelines for managing social and technical issues. However, it does not provide any information for architecture design.

SARA provides a roadmap for finding and structuring *"Architecturally Significant Requirement (ASR)"* and finding *"Architecturally Significant Decision (ASD)"*. These reviews are conducted on concrete architecture artifacts such as architectural description documents, business case, stakeholder's concerns, standards, and requirements. SARA reviewers may use one or more methods and techniques such as SAAM, ATAM, RMA, and 4 +1 Views. The objective of review may be for

analyzing the conformance to specific standard, quality assessment of the architecture, architecture improvement and for effective communication between stakeholders.

Review and Assessment Inputs: The inputs to SARA for review and assessment includes: Review Objectives; Review Scope; Architectural Artifacts (System Description, Architecture Descriptions, Architectural Decisions, Reused Solutions, Guidelines and Rules, Architecture Supporting Evidence); Architecturally Significant Requirements (ASRs); Product Strategy and Product Planning; Requirements; Standards and Constraints; Quality Assurance Policies; Risk Assessment Artifacts.

Review and Assessment Outputs: The SARA review outputs are: Assessment Report that includes: Objectives; Scope; Methodology; Evaluation Criteria for Architecture; Architectural Foundation and Approaches; Architecture Analysis, Findings and Recommendations; Executive Summary; Lesson Learned.

Activity-1: Identify Type of Review and Its Business Objectives

The objective of this activity is to identify the type of review required, its goals and issues.

Activity-2: Identify Key Stakeholders and Review Scope

The objective of second activity is to identify project stakeholders, system and its required quality attributes.

Activity-3: Identify Review Objectives

In this activity list of review objectives are prepared.

Activity-4: Plan Preparation for Approval

The objective of this activity is to prepare review plan and getting its approval.

Activity-5: Identify, Describe and Prioritize ASRs

The fifth activity is for identification of architectural significant requirements, prioritization of ASRs and its approval from key stakeholders.

Activity-6: Development of Architectural Description

The objective of this activity is to develop the architectural description.

Activity-7: Analyze Architecture Description against ASRs

The objective of this activity is analyzing architecture description for ASR, finding out risk, issues, tradeoff points and sensitivity points.

Activity-8: Summarizing and Reviewing

In this activity summary of review finding is discussed with architecture owner before documentation. The outputs of this activity are strengths, weaknesses, issues, risks, tradeoff points, recommendations and action plan.

Activity-9: Presentation

The objective of this activity is to present review report and recommendations to architecture owners and stakeholders. The outputs of this activity are review report, review presentation, and corrective actions.

Activity-10: Refinement of Review Method

The objective of this activity is to analyze the process improvement opportunity.

4.8.9 SACAM

The *"Software Architecture Comparison Analysis Method (SACAM)"* [65] was developed to compare and analyze the software architectures. It is architecture centric, qualitative, goal-oriented analysis approach. This method is developed using scenario generation concept of Architecture Tradeoff Analysis Method (ATAM) and Quality Attribute Workshop (QAW). SACAM is based on the assumption that the software architecture addresses best level of abstraction for organization's business goals. The scenarios are used to compare the candidate architectures for any commonality or variability analysis of software product. However, it emphasizes on use of existing architecture documentation or architecture reconstruction techniques for analysis. The stakeholders score each scenario that leads to the selection of architecture. The technical reuse context in which SACAM was developed helps in reuse of architectural design.

The goal of SACAM is achieved by two objectives i.e. the extraction of comparable architectural views, and criteria collation and analysis. The method consists of six steps with total of 24 activities that are carried out to compare two architectures.

Step-1: This step is further consists of nine activities such as clarify and identify the application context. Architecture candidate for comparison are selected. Candidate stakeholders are identified. SACAM is presented. Business goal for the system is identified. The candidate architecture is presented. The availability of architecture related documents is analyzed. SACAM initial plan is prepared.

Step-2: The comparison criterion is derived for organization's business goals that reflect the quality attributes scenarios. The criterion is then prioritized using stakeholder's scores and used as yardstick for comparison.

Identify Criteria, Prioritize Criteria, and Refine Criteria are the major activities of this step.

Step-3: This step consists of two activities with quality attribute scenario and architectural documentation as input. In first activity *"viewtypes"* are collected that are common across architectures for comparison. The second activity is conducted to collect architectural patterns, tactics, metrics and styles.

Step-4: This step consists of four activities. The first activity involves the view extraction for architecture documentation. The second activity involves the identification of used tactics, patterns and styles. The third activity involves the data collection for different metrics. The forth activity involves the view verification with candidate stakeholders.

Step-5: The fifth step consists of four activities. The first activity involves the selection of quality attributes model and response calculation. The second activity involves the scoring and reasoning. The third activity involves the collection of individual quality attribute scores. In forth activity score is analyzed.

Step-6: The inputs to this step are quality attribute scenarios; architectural views; scoring and reasoning. Two activities are performed in this step: Result is presented, and Summary report is prepared.

4.8.10 ACCA

The *"Architecture-centric concern analysis (ACCA)"* [66] method is analogous to *"defect causal analysis (DCA)"* method and it uses root causes of concern in software architecture. The architectural concerns are triggered by architectural evaluation process. These architectural concerns are characterized by tradeoff points, sensitivity points, and risks at architecture stage. Hence, by knowing root causes of associated concerns early in the requirement phase will help in mitigation of these risk, prevention and reduction of architectural problems. ACCA method uses a meta-model which is the extended form of Ramesh & Jarke's *"Requirement Traceability Reference Model"*. Attribute Driven Design (ADD) is used in its architectural design stage and ATAM method is used in architecture evaluation stage. ACCA method is based on iterative process which helps in detection and prevention of architectural problems early in the requirement engineering phase. This results in a high quality

architecture, stakeholder satisfaction and low utilization of resource. This method is composed of eight processes which are listed and explained below:

1- Requirements Engineering Process

In this process requirements are elicited and documented, the output of requirement engineering phase goes to *"Architecting Process"* and *"CT-Map Construction Process"*.

2- Architecting Process

ADD is used in the architecting process for designing system using UML. This process has requirement documents as an input. The output is the data and documents related to *"Architecture Template"*. This process falls in first region therefore, it is termed as *"Data Collection"*.

3- Architecture Evaluation Process

In data collection process ATAM is used to evaluate software architecture. The architecture template data is used in architecture evaluation process and data triangulation process. The output of this process is architecture evaluation documents. These documents are used as an input to concern analysis and prioritization process.

4- Concern Analysis and Prioritization Process

In *"Concern Analysis and Prioritization Process"* set of concerns collected during architecture evaluation are prioritized, duplicates concerns are eliminated. This is done by validating concerns for any overlaps, ambiguities, and inconsistencies. This process takes evaluation documents as an input and a process input from data triangulation process. The outputs of this process are refined and prioritized concerns.

5- Data Triangulation Process

The *"Data Triangulation Process"* takes two documents as inputs one is *"Concerns Documents"* and other is architecture evaluation documents. Other input for this process comes from *"CT-Map"* construction process. In this process a table is constructed for gathering decisions and for constructing CT-Map. The output of this process is documented form of rationale, assumptions, and decisions in the form of *"Triangulation Table"* that are used as an input document for CT-Map construction process.

6- CT-Map Construction Process

In *"CT-Map Construction Process"* requirements document, traceability meta-model, and data regarding concerns, triangulation table data, and feedback data are used as an inputs. This process also has one input from CT-Map analysis process. The output of

this process is feedback to data triangulation process. The accurate construction of CT-Map is guided by a meta-model. This traceability meta-model defines a schema for all the node types, relationship types, implied entities, and attribute necessary for CT-Map construction.

7- CT-Map Analysis Process

Characterization meta-model is the input to the *"CT-Map Analysis Process"* along with the feedback data, and CT-Map data. The output of this process is feed to CT-Map construction process. *"Root Causes Data"* is the output data produced by this process i.e. the analysis process will result in one or more root causes for each architectural concern. This process is repeatedly validated in 10 steps. The requirements are analyzed in 10 steps using the goal oriented approach. These steps are: derivation of goals, identification of alternative decision, decision analysis, conflict analysis, requirement analysis for conflicts identification, requirements analysis for problem identification, repetition of previous six steps with examination of *CT-Map* and *scenarios* for potential causes; eighth step involves the elimination of items not causing specific concern. In ninth step root causes of concern are identified by asking series of questions. The process terminates in tenth step if all valid questions have been asked, and all nodes are visited.

8- Validation Process

The validation process gets input data from concerns, CT-Map, and root causes. The output data of this process is used for feedback.

4.8.11 APA

The purpose of *"Architecture Potential Analysis (APA)"* process [67] is to provide a concise architecture evaluation method. This method use *"quality attribute directed acyclic graph (QADAG)"* to uncover the dependencies of the quality attributes such as performance and modifiability for expected cost. QADAG basically is a structure of quality attributes. Any change in the architecture is calculated using uncovered dependencies. The method provides a systematic way of documentation of architecture knowledge and helps in traceability of architectural decisions. First define a model for architectural representation along with their relevant properties. Second use quality rate to find the achievement of architecture goals using QADAG. Third evaluate the dependencies of the software system architecture. The dependencies

describe the influences that architecture may have on evaluation techniques. These dependencies have two classes. One class deals with the dependencies between architecture elements and quality attributes. Other class deal with the dependencies among quality attributes.

QADAG which is the hierarchical structuring of quality attributes is based on composite patterns. These quality attributes can be decomposed into sub-attributes or may consist of evaluation techniques. Quality attribute directed acyclic graph based evaluation structure is used to set dependencies into relation and for determining optimal potential. The main classes of QADAG quality attributes are scenarios, constraint, interpretation, and *"DataTypes"*. The other classes are evaluation technique, joining technique, *"LeafQA"* and *"CompositeQA"*.
The method facilitates the architecture evaluation process to be carried out either early in the design phases or after design process for ensuring quality.

The method emphasizes the use of *"Modular Performance Analysis (MPA)"* for performance analysis of architecture. This analysis is used to analyze communication performance and computation performance. The analysis can be applied in various architecture domains. MPA takes event stream models as input. The model contains detailed information on the communication behaviour. MPA also consider resource model that takes into account the resources in controller network along with execution times. They found that dependencies help in determining relationship between cost, performance and modifiability based on target market and available technology. By considering these dependencies one can uncover the optimization potential and development tendencies.

4.8.12 ALMA

The *"Architecture Level Modifiability Analysis (ALMA)"* [68] is designed to analyze the modifiability potential of software architecture design. This method is scenario based that can be used to assess the risk associated with modifiability, compare software architectures or to predict the effort required for modification. The goal of this method is to analyze software architecture for four different types of architecture analysis.

1- The candidate architectures are from different origins and architecture design differs significantly.

2- The candidate architectures are subsequent versions of the architecture to be evaluated.
3- The candidate architecture meets the required quality requirements.
4- The candidate architecture compared against the quality requirement of benchmarked virtual architecture.

The method is based on five step evaluation process in first step goals and aim of the analysis is identified. The second step involves describing candidate software architecture. In third step relevant scenarios are elicited. Evaluation for these set of scenarios is done in forth step. The fifth step involves reaching at some conclusion.

Step-1: Goal Setting

The first step of the method is to determine goal for software architecture analysis. These goals may include maintenance cost estimation, risk assessment, or architecture evaluation.

Step-2: Describe Candidate Architecture

The second step of software architecture evaluation method for modifiability is to describe candidate architecture using available most appropriate techniques. Such techniques may include using UML or some specific ADL. This method emphasize on responsibilities of software architect to choose most appropriate techniques for architecture description.

Step-3: Scenario Elicitation

The third step of the method is concerned with scenario elicitation. This step focuses on use of different techniques for scenarios elicitation. The scenario elicitation techniques depends upon the purpose of evaluation i.e. maintenance prediction, risk assessment, or software architecture comparison. The two techniques that involves are now discussed here one by one.

In first technique equivalence classes are used for change scenarios. The equivalence classes are used in limiting the number of change scenarios to consider. The second technique is used in identifying scenarios which justify selection criterion. This technique also used for defining stopping criterion. The change scenarios are elicited until the complete coverage of classification scheme. The method uses a bottom-up approach without having predefined classification scheme. Therefore, stakeholders play an important role in implicit categorization scheme. In this approach analyst moves from concrete scenarios to abstract classes of scenarios. If the purpose of the

analysis is maintenance prediction, interview with the stakeholder is considered to be the effective techniques for elicitation of change scenarios. However, if the aim of the analysis is risk assessment the top-down approach is recommended for scenarios elicitation. Complex change scenarios elicitation process uses interview with the stakeholder. The output of this process is the set of equivalence classes representing same complexity and account for same risks. If the analysis is carried out to compare two architectures top-down approach is used. Change scenarios that expose difference between two architectures are grouped using stakeholder interview.

Step-4: Scenarios Evaluation

In this step change scenarios are evaluated. The goal of analysis is to find the impact of change scenarios and their resulting ripple effects on the architecture. The scenario evaluation process is divided into three sub-steps. In first step affected components are listed. In second step modifiable operations are located. Finally in the third step ripple effect is determined. The result of scenarios evaluation may be qualitative or quantitative in nature. Different techniques may be used for expressing results of scenario evaluation.

Step-5: Result Interpretation

The fifth step involves the interpretation of results for specific goal of the analysis, i.e. whether the analysis is carried out for maintenance cost prediction, risk assessment or software architecture comparison. If the result are interpreted to predict maintenance cost, such model is recommended that are based on cost drivers of the maintenance process. However, if the results are interpreted for risk assessment, the analysis is carried out in consultation with stakeholders. The scenarios which pose more risk are group together in equivalence classes. On the other hand if the results are interpreted to compare candidate software architectures three different approaches may be used. The first technique involves the appointment of the best candidate for each scenario. The second technique is to rank these best candidates for each scenario, and third technique involves the estimation of effort for each scenario.

4.9 ARCHITECTURE DESIGN AND EVALUATION

4.9.1 4+1 View Model

The *"4+1 View Model"* [41] was developed to remedy the problem of software architecture representation using five concurrent views. These views include *"Logical*

View", *"Physical View"*, *"Process View"*, *"Development View"* and *"Use Case Scenarios"*. These views are used to select applicable architectural style. The model used these views and scenarios for modeling architecture. The model can be used to evaluate system quality attributes such as availability, reliability, scalability and portability. The process is evolutionary iterative development which involves prototyping, testing, measuring, and analysis. The analysis involves risk assessment and mitigation at the architecture level. The risk identification is performed using scenarios which are the instances of use case.

Step-1: In first step scenarios are selected based on risk and criticality these risk may be technical risks. These critical but small in number scenarios are synthesize by abstracting user requirements. The next activity is to script the scenarios followed by decomposing them into sequences of object and operation pairs.

Step-2: In this step architectural elements are organized in to the four views followed by implementation, testing and measuring of the architecture. The architecture analysis is performed at the end of these activities to reassess the risks and discovering any flaws.

Step-3: The analysis helps in identifying the additional architectural elements or changes. These changes are reflected using additional scenarios.

Step-4: The additional scenarios are implemented, tested, and measured for incremental architecture prototype. Now by using five views reuse is considered.

Step-5: After analyzing reuse of existing architectural prototype for evolutionary development the final system architecture is produced.

4.9.2 RAMRTS

The theory of *"Rate Monotonic Analysis for Real Time Systems (RAMRTS)"* [43] was developed at Software Engineering Institute. The *"Rate Monotonic Theory"* is concerned with assigning priorities as a monotonic function to a set of periodic processes. This analysis consists of set of analytical methods for real-time system engineering. RAMRTS addresses design and analysis of real-time resource management. The current theory is based on concepts of scheduling independent periodic tasks; scheduling both aperiodic and periodic tasks; synchronization of requirements; mode change requirements; hardware scheduling support; Ada scheduling rules; schedulability analysis of input/output paradigms; and algorithm

implementation in an Ada runtime system. By using rate monotonic theory one can achieve resources utilization up to 90 percentages. It is based on sporadic server algorithm which is the enhanced form of deferrable server algorithm and aperiodic server algorithm. In sporadic server the allocated budget is replenished after its consumption. It is based on real-time synchronization protocol known as *priority ceiling protocol.* The first property of this protocol is the freedom from mutual deadlock and second property is bounded priority inversion. The protocol provides a mathematical model of schedulability of set of tasks with their inequalities. The current theory also addresses the limitations of rate monotonic algorithm such as fixed task set with static priorities. Hence, to overcome these limitations Mode Change Protocol is introduced. The protocol considers deletion of some tasks or addition of new tasks or change in parameters in some tasks as the change in its mode. The extended theory addresses quality attributes such as compatibility, maintainability and performance (and applied on IEEE 896 Standard of Futurebus) for designing real-time systems. The Texas Instrument chip set TFB2010 is based on this standard.

4.9.3 SBSAR

The *"Scenario Based Software Architecture Reengineering (SBAR)"* [45] was initially introduced as method for software architecture design and passed through many refinements. The method has applied in measurement systems, fire alarm systems and in dialysis systems. SBAR now has the capability to reengineer existing software architecture for further improvement. The inputs of SBAR are updated requirements and the existing architecture which goes through many refinements and transformations to produce improved architectural design as output of the method. The improvement is carried out through three main phases which are: Functionality based Architecture Redesign; Quality Attribute Assessment; and Architecture Transformations. The main artifacts to these phases are software architecture and requirements specifications. These artifacts loop through four steps in these phases which are explained below.

A. New Functionality Based Architecture Redesign:

In this step the redesign activity is carried out on the updated requirement and existing architecture. The core abstractions which are modeled as objects of the system are identified and evaluated. The interactions between these abstractions are defined in

more details. The most relevant properties of the domain entities are modeled as architecture entities in a top-down approach.

B. Software Quality Requirement Assessment:

Software quality attributes are explicitly assessed using scenarios, simulation, mathematical modeling or experience based reasoning. The *"scenario based"* evaluation of architecture is carried out in three steps. In first step a representative set of scenarios is defined and developed to concretize the actual meaning of the quality attribute. In second step each individual scenario is analyze in its context for architecture analysis. In third step overall result of analysis is summarized in term of number of accepted scenarios versus rejected scenarios. The second approach recommended for assessment for quality attribute is *"Simulation"*. The operational quality attributes are assessed based on simulating application behaviour. The third recommended approach is *"Mathematical modelling"* for quality attributes evaluation. This approach allows static evaluation of architecture design for assessing operational quality attribute. The forth approach recommended for assessing quality attribute is experienced based reasoning or logical reasoning.

C. Apply Architecture Transformation:

Architecture transformations are applied after the assessment of architecture properties. If some quality attribute does not meet its required value then architecture transformations are applied. This transformation resulted in new version of the architecture with same functionality but with improved or different value of quality attribute. In this method five transformations are used which are

Architectural and design patterns, styles, quality attribute conversion, and requirements distribution among subsystems. The application of architectural styles results in to reorganization of software architecture. The reorganization result in improvement of certain quality attribute but may result in degradation of another quality attribute. The second transformation that is applied is application of architectural patterns. This transformation affects the large part of the architecture by imposing certain rules. The third transformation is applied by utilizing different design patterns. However, this transformation only affects small numbers of architectural components. In fifth transformation software quality requirements are converted into functionality of the system. Finally, the distribute requirement transformation is applied by distributing quality requirements in number of components or subsystems.

D. Evaluate Design:

Once the architecture transformations are applied on architecture it goes through assess quality attribute phase of the architecture reengineering method. If all requirements meet the final version of the architecture is released for implementation.

4.9.4 SBSAD

The *"Scenario Based Software Architecture Design (SBAD)"* [46] method is an iterative process for creating software architecture by applying design transformation. The method uses different techniques for the evaluation of quality attributes of architecture such as scenarios, mathematical modeling, simulation, and reasoning. The method uses architectural transformations for iteratively assessing the achievement of quality attributes. Five different architectural transformations are applied in this method which includes application of architectural styles, architectural patterns, and design patterns. The other architectural transformations that are applied include conversion of quality requirement into functionality and distributing quality requirement across different components or subsystems. The method is a rational design process that has the capabilities of balancing and optimizing quality requirements. The input to this method is requirements specifications and output of this method is architectural design. This architectural design is produced through application of various transformations which are applied iteratively. However, in first iteration only functional requirements are considered which results into architecture design of an application. This architecture is evaluated and analyzed with respect to required quality attributes. This assessment may be quantitative or qualitative in nature. If the quality attribute requirements are up to the expectation, the process is terminated and architecture design is released. In case quality requirements are not up to the expectation the architecture design process enters the second stage where architecture transformations are applied. The application of architectural transformation results into new version of architecture. This new version is then evaluated for required quality attribute requirements. If NFR not fulfil the loop is repeated otherwise the final architecture design version is released.

A. Functionality Based Architecture Design:

In this phase system architecture is designed by identifying the core abstractions of the system structure in top-down approach. These abstractions are modeled as objects

using a creative design process which involves analysis of domain entities. These domain entities are then modeled as architecture entities and interactions between abstractions are identified.

B. Assessment of NFRs:

System NFRs are explicitly assessed using scenarios, simulation, mathematical modeling or experience based reasoning. The scenario based evaluation of architecture is carried out in three steps. The second approach recommended for assessment for quality attribute is simulation of architecture. The third recommended approach is mathematical modeling for quality attribute evaluation. The forth approach recommended for assessing quality attribute is experienced based reasoning.

C. Apply Architecture Transformation:

Architecture transformations are applied after the assessment of architecture properties. In this method five transformations are used which includes application of architectural styles, patterns, quality attribute conversion in to functional requirements, and finally the requirement distribution among subsystems. The application of architectural styles results in to reorganization of software architecture. The reorganization results into improvement of certain quality attribute but may result in degradation of other quality attributes. The second transformation that is applied is application of architectural patterns. This transformation affects large part of architecture by imposing certain rules. The third transformation is applied by utilizing different design patterns. However, this transformation only affects small numbers of architectural components. In fifth transformation software quality requirements are concreted into functionality of the system. Finally, the distribute requirement transformation is applied by distributing quality requirements in number of components or subsystems.

D. Evaluate Design:

Once the architectural transformations are applied on architecture it goes through assessment of quality attributes phase of the architecture design. If all requirements

design. The reasoning framework may be based on qualitative or quantitative model of single quality attribute and can be used in architectural design and analysis. Attribute Based Architectural Style is based on a specific quality attribute and have additional property that they are attribute specific too. Therefore, for an architectural style with more than one quality attributes needs ABAS for each quality attribute associated with it. The method facilitates the reuse of attribute based architectural styles in design and analysis for precise reasoning and predictable repeatable properties. The publication exemplifies the use of ABAS for availability, modifiability and performance in architectural design. However, ABAS qualitative analysis can be used with Architecture Tradeoff Analysis Method (ATAM) for detail evaluation of architecture. When using ABAS one can use formal reasoning and mathematical models for quantitative analysis or informal reasoning for qualitative analysis. The standard characterization of quality attribute is the prerequisite for execution of ABAS. Every quality attribute is characterized in three categories first is stimuli; second is architectural parameter; and third is response. This standard characterization of quality attribute provides consistent way of documentation and analysis for architectural design decision. The common parts of any ABAS are *"Problem Description"*; *"Stimulus/Response Attribute Measure"*; and *"Architectural Style and Analysis"*. Initial work describes six ABASs which are: *"Synchronization ABAS"*; *"Data Indirection ABAS"*; *"Abstract Data Repository Sub-ABAS"*; *"Publish/Subscribe Sub-ABAS"*; *"Layering ABAS"*; and *"Simplex ABAS"*. With these pre-packaged unit of design and analyses for known problem provides an architect a reusable repository of knowledge for efficient architecture design and analysis.

4.9.6 QADA

The *"Quality-driven Architecture Design and Analysis (QADA)"* method [49] developed by VTT Technical Research Center of Finland provides a process and language for design and analysis of product line architecture. The method utilizes architectural styles and patterns for designing high quality software architecture. In QADA two level of abstraction are considered for developing product line architecture first is conceptual architecture design and analysis second is concrete architecture design and analysis. The method has applied on distributed service platform which embodies layered service architecture. QADA is a scenario based

architecture evaluation method used to assess the quality of single product-line architecture. In the first step two categories of changes are considered i.e. category of scenarios related to technical requirement of the platform and category of scenarios related to technical environment. In second step change scenarios are identified. These scenarios reflect the future expected changes to product-line architecture. In third step product-line architecture is described in with the help of abstract as well as concrete components. In forth step effect of scenarios is evaluated for product-line architecture. Finally in the fifth step scenario interaction is evaluated to find poor separation of concern in PLA. The main phases of QADA and sub-activities are given below.

a) Requirement Engineering:

The requirement engineering phase provides a link between requirement engineering phase and software architectural design phase of software development lifecycle. In this phase driving ideas of system are identified. The main steps involved in this phase are system requirement specification, system requirement analysis, defining system context, and determination of product line scope.

b) Conceptual Architecture Design:

During conceptual architecture design functionality and responsibilities are considered for conceptual structural components using structural view, behaviour view and deployment view. Conceptual structural components and conceptual structural relationship are two design elements that are considered during design of conceptual structural view. Conceptual structural view is designed in three iterative steps. In first step functional responsibilities are captured in textual form. The structural model is built using selected architectural styles, design principles and rules. The technical properties of the system are also considered in this step. In second step quality responsibilities of conceptual component are defined with selection of appropriate architectural style. In third step functional responsibilities are grouped in conceptual components and subsystems. Conceptual behaviour components and conceptual behaviour relationship are the two design elements that are considered during the design of conceptual behaviour view. This view is used to specify the behaviour of the system at higher level of abstraction. The three main step of the design process are: selection of collaboration scenarios; identification of service sets related to collaboration scenarios; and creation of collaboration model which is the aggregation of collaboration diagrams. The main elements involved in conceptual

deployment view are *"Deployment Node"*; *"Unit of Deployment"*; and *"Conceptual Deployment"* relationship. These elements along with deployment language and design steps are used to design conceptual deployment view. The design step is carried out in three steps in first step clustering is done for leaf component. In second step deployment nodes are identified. Finally, in the third step deployment units are allocated to nodes.

c) Conceptual Architecture Quality Analysis:

Conceptual architecture is analyzed using commonality analysis and by determination of any violation of architecture patterns and styles. Variability analysis is conducted for building flexible architecture. The points where product line architecture has variation points are analyzed. Such variability is represented using hot spots or patterns. The violation of architectural patterns and styles are determined. Missing components or links are traced to identify possible violation. These violations may result from system maintenance, understand-ability, or discrepancies involved in documentation.

d) Concrete Architecture Design:

The conceptual structural components and their relationship are refined in this phase using three views. During design of concrete structural view step architectural elements, structural language, and design steps are used to describe a concrete structural view. *"Capsule"*, *"Port"*, *"Connector"* and *"Protocol"* are used as architectural elements. ROOM method is used for concrete structural language. To design concrete structural view as a first step functional responsibilities are refined and documented as capsule documentation. In Second step design patterns and architectural styles are applied. Finally, the hierarchical structural diagram is constructed in third step. During design of concrete behaviour view concrete behaviour components and concrete behaviour relationships are the two design element involved in the design of concrete behaviour view. UML is used to describe behaviour of architectural elements. State diagram and sequence diagram are used to describe these behaviours. The design steps involved in designing concrete behaviour view are first to define inner behaviour of capsule using state machine. The second step involved is to refine collaboration scenarios which are expressed in the form of message sequence charts. During design of concrete deployment view concrete deployment hardware component; concrete deployment software component; concrete deployment hardware relationship; and concrete hardware-software deployment

relationship are used as four design elements. UML is used to describe concrete deployment view. The design is carried out in three steps. In first step capsules and protocols are physically packaged in a component. In second step deployment nodes are defined. Finally, in the third step deployment diagram is built up from selected hardware and software components.

e) Concrete Architecture Quality Analysis:

The quality analysis of concrete architecture is carried out using *"customer value analysis (CVA)"* and scenario-based analysis. Customer value analysis CVA is used to identify most important changes and their impact on cost of product and return from market. The architecture is evaluated using CVA by assigning weight to scenarios and scenario interaction. The analysis can be used to compare two candidate architectures.

4.9.7 APTIA

The *"Analytic Principles and Tools for the Improvement of Architecture (APTIA)"* is a process [53] for improvement and design of software architecture which is based on reusable pre-existing components of ten techniques. The method has been applied on a commercial information system with product line architecture and real time requirements. The method is based on three principles of software architecture. The first principle is related to the abstraction of software architecture i.e. the elements of software architecture should be coarse such that human intellectual can control it and for meaningful reasoning it should be specific. The second principle is concerned with business and mission goals i.e. quality attribute requirements should be determined form business and mission goals. The third principle is concerned with design and analysis of architecture i.e. the quality attribute requirements guides the design and analysis of architecture. These principles and component techniques together are used in APTIA. These techniques are explained one by one below:

1- The explicit elicitation of business and/or mission goals.
2- Active participation of stakeholders.
3- The explicit elicitation of rationale of architecture and documentation in standardized views.
4- The realization of mission and business goals in 6-part quality attributes scenarios.
5- 6-part quality attribute scenario mapping onto architecture.

6- The use of architectural tactics.
7- Using templates to capture information.
8- The explicit elicitation of cost, benefit and schedule associated with architectural decisions.
9- The use of quality attributes models for architecture analysis and design decisions.
10- The use of design principles based on quality attribute models to identify alternatives for improvements.

The APTIA method provides more detailed analysis and design alternatives for improvement of architecture. The APTIA is designed in such a way that it can be broken down into a number of modular steps with each step consist of proven techniques. These techniques are considered as "component" which can be combined to create a new method. APTIA is one of such method that is created from these component techniques tailored for specific need. APTIA has six phases. In its first phase ATAM is performed. In its second phase based on risk themes focus of analysis is determined. The third phase is concerned with use of quality attribute models for risk themes. In forth phase model based analysis and design principles are used to propose design alternatives. The fifth phase is concerned with ranking design alternatives based on cost and benefits. The design decisions are made in final phase. These decisions are made based on costs/benefits associated with the architecture design using existing component techniques in an agile way. The APTIA therefore, provides a deeper analysis of architecture design and suggests design alternatives with new design principles. The consistency in design is achieved using two templates one template for documenting outputs of APTIA for analysis and second template for documenting outputs of APTIA for architectural alternatives.

4.9.8 GMSAD

The *"General Model for Software Architecture Design (GMSAD)"* process [54] is based on the commonalities that can be found in five software architecture design and analysis methods. These five methods are: Software Engineering Institute's *"Attribute-Driven Design (ADD)"*; *"Siemens' 4 Views (S4V)"*; *"Rational Software RUP 4+1"*; *"Business Architecture Process and Organization (BAPO)"* developed at Philips Research; and Nokia Research's *"Architectural Separation of Concerns (ASC)"*. The method provides a framework for comparison of strengths and

weaknesses of these methods. The framework developed can be used for developing new methods. GMSAD is one such method that is developed using the framework which is based on the commonalities of the five industrial methods. The method presents three main and common activities in architectural design model which are explained below.

Architectural Analysis: This is the first main activity whose inputs are *"Context"* and *"Concerns"*. The output of this activity is the *"architecturally significant requirements"* (ASRs). The activity is performed to articulate the ASRs. The architectural analysis activity is performed to filter the requirements that are not relevant to the architecture. Based on architectural concerns and system context a set of architecturally significant requirement are identified. The architectural problems are also analyzed in this activity.

Architectural Synthesis: The input of architectural synthesis activity is the architecturally significant requirements whereas the *"Candidate Architectural Solutions"* are the output of this method. It is the core activity of architecture design. Based on architecturally significant requirements different solutions are proposed in this activity.

Architectural Evaluation: Candidate architectural solutions and architecturally significant requirements are the inputs of architectural evaluation activity. The *"Validated Architecture"* is the output of this activity. Based on architecturally significant requirements candidate architectural solutions are measured to validate that the design decisions are correct. The evaluation is carried out repeatedly to ensure the validation of architecture. The artifacts and sub-activities involved in this model are now explained below.

The definition of architectural concerns is based on IEEE 1471 standard. Architectural concerns reflect the interests of one or more stakeholders. The concern may include system consideration, mandated design decisions or regulatory requirements. All architectural concerns become the input to the architectural analysis activity. IEEE 1471 standard's definition of context of system is used to determine the setting and circumstances of political, developmental, or operational influences. The *"contexts"* and *"concerns"* together are the inputs to the architectural analysis activity. Architecturally significant requirements (ASRs) are extracted from system context or architectural concerns. The ASRs are those requirements that influence the software system architecture. Therefore, it is not necessary that all of the system

requirements will be ASRs. The candidate architecture solutions are design decisions about the structure of software. The output of the architecture synthesis activity is the candidate architectural solutions which may be partial solution or alternative solution. These candidate architecture solutions include design rationale and its traceability. The candidate architectural solutions that are mutually consistent and consistent with the architecturally significant requirements together form a validated architecture. This validated architecture also includes design rationale. "*Design Knowledge*" in the form of architectural styles, patterns, reference architecture, or use of ADLs comes from the architect. These are important inputs to design process. "*Analysis Knowledge*" in the form of analysis patterns and analysis models is another input to the design process. The other inputs that go to design process are the "*Knowledge of Evaluation Process*" and "*Realization Knowledge*". As the architecture design process progresses a backlog of issues or problems is built up. This "*backlog*" drives the architecture design process which is often non-linear. This backlog of need and issues is prioritized to resolve these problems. The architectural issues that are resolved by the architect are removed from backlog.

4.9.9 ABC/DD

The "*Architecture Based Component composition / Decision-oriented Design (ABC/DD)*" is an iterative process [56] for designing software architecture. In the first step architect elicit architecture design issues this phase is called issue elicitation phase. The next stage is called solution exploiting phase in which architect exploit the candidate solutions to each issue based on reusable design knowledge. The candidate architecture solution is synthesized automatically from various "issue solutions" in the solution synthesizing phase. The architecture is evaluated in the architecture deciding phase. The final phase is ration capturing phase in which architect captures architecture rationale and issue rationale. The ABC/DD approach for software architecture design is based on software architecture principles of "*decision-abstraction*" and "*issue-decomposition*". These principles are used to solve complexity and difficulty associated with the software architecture design. To bridge the gap between requirements and design the "decision-abstraction" principle is used, this principle consider architecture from the perspective of system wide design decisions. The decision-abstraction provides necessary high level abstraction on

problem space as well as on solution space for modeling of software architecture. The second principle "issue-decomposition" is used to consider the architecture design task as solving system wide problems. The design goals are decomposed into number of related design issues, these issues are used for making decisions. In ABC/DD an issue is referred to as *"Architecturally Design Issue (ADI)"* .The *"solution"* is used for solving an issue or all the issues related to architecture. The *"decision"* is used for acceptance or rejection of single candidate *"issue solution"* or one candidate architecture solution. Finally, a *"rational"* is used to reason about an *"issue decision"* or architecture decisions. This method provides a way for making architecture decisions based on architecture level problems. These architectural significant problems are elicited in the architecture design phase and solution is provided for these problems. In this activity stakeholders and architects participates for considering solutions for each issue. The decisions made in this activity are recorded in a tool that automates this process. The tool is developed as an Eclipse plug-in and implements the ABC/DD process meta-model. The ABC/DD method is successfully applied to Spaceflight Center Commanding and Control System. This system is used to receive data of space vehicles using Telemetry, Tracking and Control network. Space vehicles current status, its orbit calculation and control commands are monitored using this network. The second application of this method is on Commanding Display Systems (CDSs) which is the Air Traffic Control system. The ABC/DD method is applied on this system during its re-architecting. In this case study architect first elicited functional requirements and non-function requirements. Architect then elicited architecturally significant issues and solution for these issues. The instance model was then created for each issue solution. All these information was automated using tool and automated synthesis of architecture solution was produced. The architect and stakeholder use this solution for making architecture choices, tradeoff among quality attributes, and architecture evaluation for global considerations. These case studies provided the evidence that decision-oriented method which is stakeholder centric provides better architectural design approach then traditional artifact-oriented approach. The proposed approach provided them a systematic and rational design process. It also helped in reducing difficulties of software architecture design. The automation of the process provided them a solution synthesis of candidate architecture automatically and provides automatically elimination of unfeasible combinations of issue solutions. Multiple candidate

architecture solutions are possible for a single issue with some advantages, disadvantages and tradeoffs. Therefore, global impact on the whole architecture need to be consider by the architecture. The proposed solution has to be evaluated and compare for the candidate architecture. The ABC/DD decision oriented approach makes clear the use of decision and rationale in design process as well as in architecture design models.

4.9.10 SPE

The *"Software Performance Engineering (SPE)"* [69] is the process to evaluate performance of the software system. They proposed the process which is quantitative in nature. The process utilizes models for predicting performance aspect of the software. The process can be used to identify problems with the software system architecture, design and implementation. The models used in the process are used to predict performance goals. The process utilizes adaptive strategies which includes upper-bounds, lower-bounds estimates, best-case and worst-case analysis. These strategies are used for managing uncertainty in the measurement. These analyses are used to predict best-case and worst-case performance. The process utilizes different techniques for precise measurement of performance. The process uses models for identification of problems with the software system architecture, design and implementation. Different techniques are used for precise results. These techniques include the refinement of architecture, building performance prototypes and identifying resource requirement. The execution model of system and execution model of software are used performance assessment of architecture. The software execution model uses execution graph for workload scenarios. The static analysis of the mean-case, best-case and worst-case response time is carried out with software execution model. They consider the software execution models to be sufficient for identification of performance problems of the architecture. The software execution model is a dynamic model and is constructed in presence of workloads or multiple users. The input parameters are collected by solving software execution model. These parameters are used for system execution model. The system execution model provides precise metrics used for resource contention, sensitivity of performance metrics, effect of software on the system's service objectives, bottleneck resource, and finally the data for performance improvement.

4.9.11 CB-SPE

The *"Component-Based Software Performance Engineering (CB-SPE)"* [70] process is based on CB framework utilizes RT-UML profile for modelling. The process is applied on component layer for parametric performance evaluation of the components. CB-SPE is then applied on application layer to determine performance of the assembled components. The performance is evaluated on actual platform. The process deals with the automated compositional framework for performance analysis of component based system. They used the approach for the analysis of the performance attributes such as system execution time, component execution time, response time, and resource utilization. The approach supports the computation of performance parameters of application from its components. The component layer is used to obtain components with desired performance properties. CB-SPE at the application layer involves seven steps. The first step involves the determination of the usage profile. In this step different types of application users and different use case are defined. The usage profile is the collection of all the use case probabilities. Every use case represents an activity diagram. In second step those components that provide best performance with same services are selected. In third step application workflow is represented by sequence diagrams. The deployment diagram is created for representing available resources. The sequence and deployment diagram are annotated with the proper performance values and parameters. In forth step best-case performance analysis is performed. They performed the analysis with no resource contention. The results they obtained provided them optimal bound on the expected performance. The fifth step involves the CB-SPE based model generation. The queuing model is obtained in this step. The sixth step involves the QN model evaluation. Finally the seventh step involves the analysis on results obtained in the previous step. The analysis performed is used to modify the parameters for achieving desired performance results.

4.9.12 PASA

The *"Performance Assessment of Software Architecture (PASA)"* [71] is a method for evaluating performance of software architecture. The method is based on principles of software performance engineering (SPE). The method is used to identify

performance related architectural risks, and suggests mitigation strategies for these risks. PASA is scenario based evaluation of software architecture. The first step involves in the evaluation is concerned with the introduction of PASA process. Presentation is given to the participants includes the process of assessment of architecture, participant introduction, processing information, and potential outcomes of the process. The presentation also includes the overview of software performance engineering goals, methodology, data requirement and results. The steps of the process are explained. The tradeoff involved in quality attributes. The second step involves in the process describes the architecture of the system. The team member explains the current or planned architecture, followed by a question and answer session. They used scenarios elicitation technique in this step and found helpful. The step three involves the identification of critical use cases. The use cases for which there is sufficient performance risk are considered to be the critical use cases. The forth step involves the selection of key performance scenarios. In case of critical use case those scenarios are considered which found important for performance evaluation. The performance modeling is used for quantitatively assessing the performance. In step eight if performance of the system is found to be unsatisfactory alternative solution are found. Architectural styles and interaction between components is explored for possible solution to the performance problems. Performance anti-pattern are identified and refactored to improve performance. The step nine involves the presentation of architecture with possible performance solutions.

4.10 ANALYSIS AND RESULTS

The analysis was conducted in the last phase of literature survey. Total of 32 publications were selected for analysis. Out of these thirty two publications seven publications were related with architecture design. Twelve publications discussed architecture evaluation. Thirteen publications were classified in architecture designed and evaluation section. All these publications were analyzed for required technical and business quality attributes. Table 4.1 shows the results in tabular format.

TABLE 4.1
Quality Attribute Analysis

	Reference	Availability	Modifiability	Performance	Security	Usability	Reusability	Reliability	Traceability	Cost	Benefit	Schedule
Architecture Design												
RDP	[42]							G		G		
FORM	[44]			G	Y		Y			G		
ABD	[48]	Y	Y	Y				R				
QAW	[50]	Y	Y	Y	Y	Y	Y	Y				
MAD	[51]	G	Y	Y	G	G	G	G				
ADD	[52]	Y	Y	Y	Y	Y	Y	Y				
AREL	[55]								Y			
Architecture Evaluation												
SAAM	[57]	Y	Y	Y	Y	Y				P		N
SAAER	[58]	Y	Y	Y	Y	Y	Y			P		N
E-SAAM	[59]	Y	Y	Y	Y	Y	Y					
ARID	[60]					Y						
ATAM	[61]	Y	Y	Y	Y	Y			N	P	N	P
CBAM	[62]	Y	Y	Y	Y	Y			N	Y	Y	Y
4+1VME	[63]	Y	Y	Y	Y	Y				Y		Y
SARA	[64]	Y	Y	Y	Y	Y	Y			Y		Y
SACAM	[65]	Y	Y	Y	Y	Y				P		P
ACCA	[66]	Y	Y	Y	Y	Y				P		P
APA	[67]	P	Y	Y	P	P				Y		N
ALMA	[68]		Y							Y		
Architecture Design and Evaluation												
PIP		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4+1VM	[41]	Y	Y	Y	Y	Y		Y		Y		Y
RAMRTS	[43]			Y				Y		Y		
SBAR	[45]	G	G	Y	G	G	Y	G		G		
SBAD	[46]	G	G	G	G	G	G	G		G		
ABAS	[47]	G	G	G				G				
QADA	[49]	Y	Y	Y	Y	Y	Y	Y		Y		Y
APTIA	[53]	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
GMSAD	[54]	G	G	G	G	G	G	G				
ABC/DD	[56]			Y		Y				Y		Y
SPE	[69]			Y						G		
CB-SPE	[70]			Y								
PASA	[71]			Y						G		

Yes=Y; No=N; Partially Address=P; Generic=G

4.11 CONCLUSTION

All of these thirty two publications that were discussed in this chapter was studied and analyzed in detailed. **However, no publication was found that addressed architecture design and evaluation for Personal Software Process.** Therefore, to fill the gap in the literature Personal Integrated Process is proposed and will be discussed in the next chapter in detail. The process is designed to address the need for individual engineer for architecture design and evaluation.

CHAPTER 5
PROPOSED PROCESS
"PERSONAL INTEGRATED PROCESS"

5.1 INTRODUCTION

The *"Personal Integrated Process (PIP)"* is designed to address the need for individual software system engineer. It is the extended form of the *"Personal Software Process (PSP)"*. PSP was designed to address the need for individual software engineers or for the team of three or five engineers. It strongly addresses the need for risk analysis at the system structure level. However, it does not provide support for architecture design and evaluation. PIP contains major form, scripts, and tables of PSP for data collection. However, PIP was designed to address the limitations of PSP. Like PSP personal integrated process contains major process of CMMI-DEV. PIP introduces the *"Bottom-up Planning and Post-mortem"*. Bottom-up estimation is designed for individual engineers. However, PIP also introduces *"Top-down Planning and Post-mortem"*. Top-down estimation is designed for TSP coaches or managers. They can use this estimation for distribution of work among software engineers. The major process components of PIP includes the process components for architecture design, architecture evaluation, database development, computer hardware installation and configuration, computer network installation and configuration, and finally the process components for system software engineering. These process components are explained below.

5.2 SYSTEM ENGINEERING PROCESS

The PIP is designed to address the need for individual software system engineers. It is technology and domain aware process. The process has designed for development of large complex systems such as *Enterprise Resource Planners (ERPs)* or business information system. These systems are composed of large number of computers connected in a complex network topology. These systems are mostly designed for

The system engineering process is designed to address the need for such complex and large software systems. The process has many feedbacks from database development process, software development process, computer configuration process and network configuration process. These feedbacks of information are used for planning and post-mortem. Fig. 5.1 on previous page shows the system software engineering process. The details of activities involving this process are discussed below one by one.

5.2.1 System Development

The system software development process S-1 is designed for developing and designing software system. The process has major feedback from *"Stakeholder Requirement Definition Process (S-A)"*, *"Validation Process (S-C)"*, *"Verification Process (S-D)"* and *"Planning Processes (S-2.1, S-3.1, S-4.1)"*. The planning process is further composed of two processes one is *"System Requirement Engineering (SRE) Process"* and other is *"Domain Engineering (DE) Process"*. These processes are discussed in detail in [73]. However, these processes are tailored for the individual needs. The process scripts are given in the Appendix-B and Appendix-C respectively. The system requirement engineering process deals with requirement elicitation, requirement analysis, and transformation of requirement into design. The *"Unified Modeling Language (UML)"* specifications [74] are used to document the requirements and design. However, for precise modeling technology based UML modeling notations can be used [78]. The output of the *"Planning Process (S-1.1)"* goes to *"System Architecture Design Process (S-1.2)"*. Architecture of system such as business information system is designed using the process. The process is started with Quality Attribute Workshop (QAW) followed by the application of Attribute Driven Design (ADD). These processes QAW and ADD are tailored for the need of individual software engineers. The *"System Architecture Design Review (S-1.3)"* process is designed for personal review for architecture design. In this process Active Reviews for Intermediate Designs (ARID) is used for reviewing portion of architecture. The ARID process is tailored for the need of individual software engineers. The *"System Architecture Evaluation (S-1.4)"* process is conducted on initial design documented using View and Beyond (V&B) approach. This process is

consisting of Architecture Tradeoff Analysis Method (ATAM) and Cost Benefit Analysis Method (CBAM) for evaluation of system architecture. These methods are also tailored for the need of individual software system engineer(s). Major feedbacks for architecture evaluation process come from *"Product Integration (S-1.6)"* and *"System Testing (S-1.9)"* processes. These feedbacks are also used for design traceability, risk mitigation and comparison of project estimated cost with actual cost.

5.2.2 Product Development

The software product development process S-2 is introduced to develop software products. The process is composed of standard PSP3 components with support of architecture design. This *"Architecture Design (S-2.2)"* process is conducted for software components. The *"Architecture Design Review (S-2.3)"* process is executed for reviewing architecture of products. All other phases of S-2.1 are executed in the same manner as in PSP3.

5.2.3 Component Development

The component development process S-3 is executed in the same way as executed in PSP3. However, design process involves View and Beyond (V&B) approach for documenting software design. Mathematical design notations are used in addition with V&B. These mathematical design notations found to be very helpful during design traceability, validation and verification. These mathematical design notations were introduced in PSP.

5.2.4 Module Development

Software modules are designed using process S-4. However, design process for software module can vary based on development technology used and its application to domain. The design notation may include View and Beyond (V&B), mathematical notations, technology based notation such as OraclJDeveloper design notation, or domain based design notations such as used for Human Machine Interface (HMI), Supervisory Control and Data Acquisition (SCADA), or IEC 61499.

5.3.2 Product Development

Product development process D-2 is used for developing database application for each produce in the system. The *"Product Architecture Design (D-2.2)"*process consists of seven sub-processes. The *"Database Usage Analysis"* sub-process is used for determining database usage. Usage analysis is used to determine the usage paths or patterns. These usage paths and patterns are used for selection of file organization and data access method. The *"Volume Analysis"* sub-process is used for determining size (volume) of database utilization. Volume analysis is used to determine size of disk storage and their associated cost. The *"Data Distribution Strategy (D-2.2.2)"* sub-process is used for determining the location of data on the network. Four different data distribution strategies are used centralized, partitioned, replicated and hybrid. The *"File Organization (D-2.2.3)"* sub-process is used to select file organization techniques optimal for recording and arranging records of files on physical storage devices. Some of the techniques used for file organization include sequential, indexed and hashed. The *"Indexes (D-2.2.4)"* sub-process is used identify indexes for primary key value or for non-key attribute values. The sub-process is used in performance trade-off analysis involved in utilization and selection of indexes. Performance of database can be increased for data retrievals by the used of indexes. However, performance will be degraded during insert, update and delete operations. Finally *"Business Rules (D-2.2.5)"*, *"De-normalization (D-2.2.6)"*, and *"Redundant Keys Elimination (D-2.2.7)"* sub-processes are used in products development.

5.3.3 Component Development

The *"Conceptual Design (D-3.2.1)"* sub-process is used for developing detailed E-R diagram. The *"Process Design (D-3.2.2)"* sub-process is used for developing process model using *"Data Flow Diagram (DFD)"*. The *"Logical Database Design (D-3.2.3)"* process is used to construct data model for some class of database management system. Different logical database models are considered for implementation. These models includes network, hierarchical, relational, object-oriented, and object-relational. The *"Relational Design (D-3.2.4)"* is used to construct relations model of the components. Personal Integrated Process (PIP) is domain and technology aware process. This is one of the reasons that relational design sub-

process is used for modeling object-relational database such as Oracle 10g Express Edition. However, if different logical model is required for database design then that model's activities will used in place of D-3.2.4. The four activities involved in relational design process are represent entities, represent relationships, normalize the relations, and merge the relations.

5.3.4 Module Development

Database module development also includes *"Conceptual Database Design (D-4.1.1)"*, *"Logical Database Design (D-4.1.2)"* and *"Relational Design (D-4.1.3)"* processes. These processes have same activities as processes *D-3.2.1, D-3.2.3,* and *D-3.2.4* respectively. These database modules can be designed and documented using V&B approach, mathematical notation, or technology based notations such as OracleJDeveloper design notations. The OracleJDeveloper environment provides paperless environment for documenting software design and architecture. The environment also provides traceability support to design and architecture.

5.4 COMPUTER HARDWARE CONFIGURATION PROCESS

Business Information Systems (BIS) such as ERPs are designed for handling hundreds or even thousands of users. These systems may be designed using client-server architecture, web-based architecture or service-oriented architecture. Therefore, process is required to handle installation, configuration and cost estimation of large number of computers. A process script for handling these activities is given in the Appendix-D.

5.5 COMPUTER NETWORK CONFIGURATION PROCESS

Designing computer network designed for large number of computers, network components and accessory also requires a process. This process can provide useful information for estimation of project cost. A process script for designing, installation, configuration and cost estimation process is given in the Appendix-E.

5.6 PROCESS AUTOMATION

Process automation support for the Personal Integrated Process is provided by developing software quality management system. The software quality management

system designed for PIP is *"Firmsoft Software Quality Management System (FSQMS)"*. The FSQMS is designed to address the adaption barrier to manual process for data entry and data logging. Detail of process automation is given in the next chapter.

5.7 CASE STUDY

A case study was designed and executed to find the effectiveness of the process for individual software engineer. Both processes i.e. PSP and PIP were executed in university computer lab. These processes were used to develop an n-tier database enable complex and large web-based system. The Personal Integrated Process was first designed and implemented. Then PIP was executed manually without any automation support. However, line of code counting tool was used for measuring software size. Details of case study are given in the chapter2.

5.8 ANALYSIS AND RESULTS

The results of the Personal Integrated Process were found to be excellent. Its top-down estimation facility found to be highly helpful in the process. Bottom-up estimation facility provided standard feature of PSP.

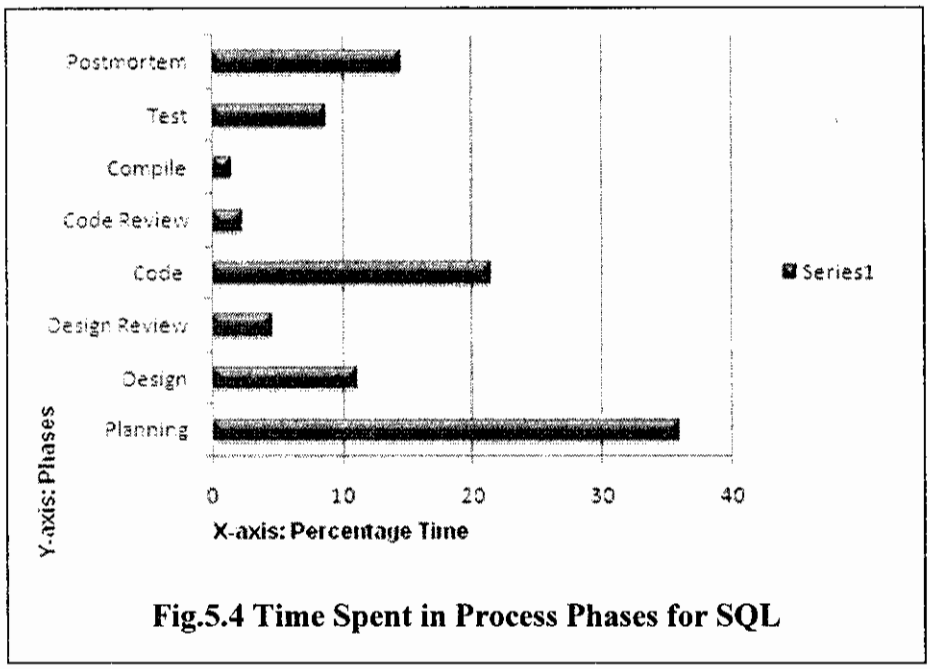


Fig.5.4 Time Spent in Process Phases for SQL

Three basic measures size, defect and time were estimated and compared with actual data. The process found to be helpful in providing precise estimation of cost,

schedule and defects. Personal Integrated Process introduces top-down and bottom-up planning for precise estimation of project cost, schedule and defects.

Time distribution among different phases of the Personal Integrated Process (PIP) for SQL code is shown in Fig.5.4. About 50% of the project time was spent in planning and post-mortem phases. This time can be reduced by automating process the whole process. Time spent in other phases of software development is quite reasonable. The bottom-up time and top-down time for both planning and post-mortem were considered during analysis.

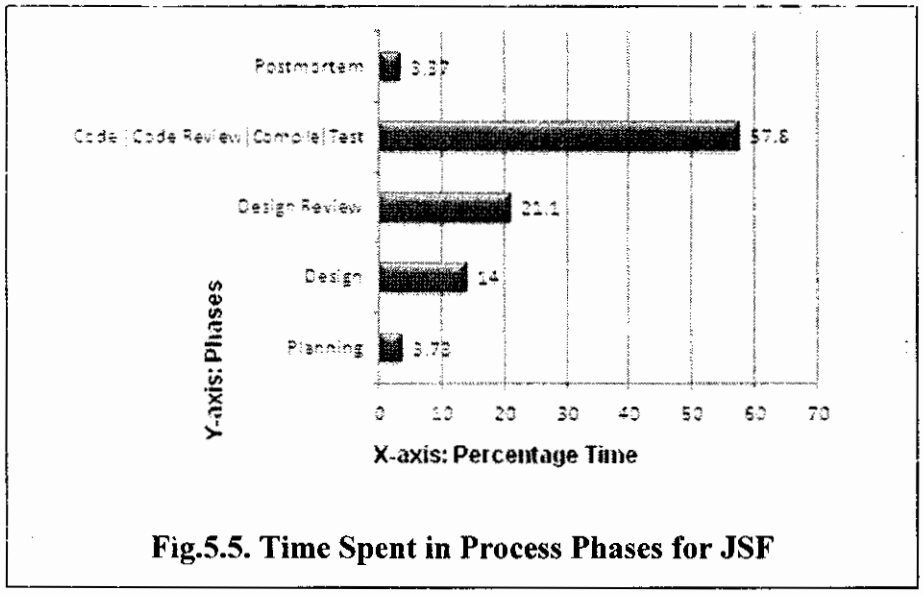


Fig.5.5. Time Spent in Process Phases for JSF

Time spent in JSF development is shown in Fig.5.5. About 6% time was spent during planning and post-mortem. Time analysis for JSF only includes the bottom-up planning and post-mortem.

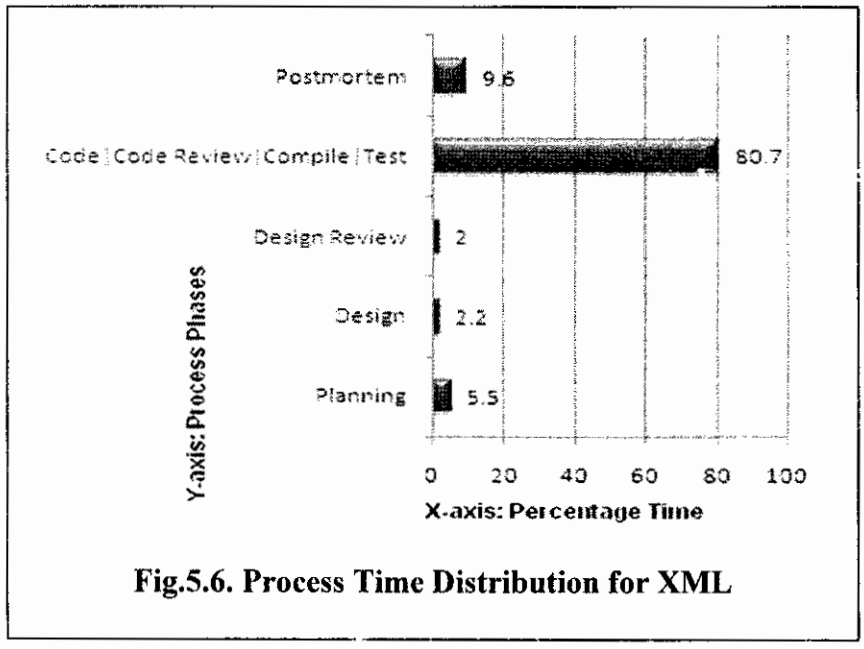


Fig.5.6. Process Time Distribution for XML

PIP introduces the concept of *"integration time"*. Product integration time is now measureable and is shown in Fig.5.6. The process utilizes the concept of *"Agile"* software development. Prototyping are used to evaluate portion of the design. It introduces the concept of *"Joint-Time"* for measuring time of Code| Code Review| Compile| Test phases.

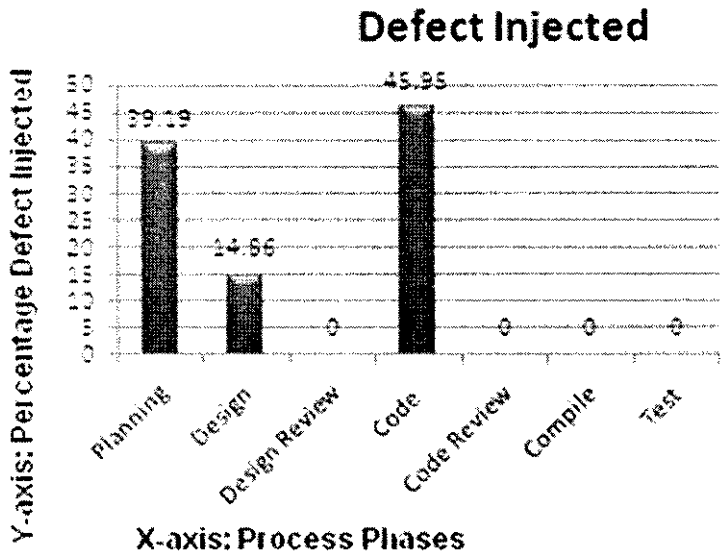


Fig.5.7. Defects Injected

Defect distribution of injected defects and removed defects can vary with the selection of development technology. When using Oracle 10g Express Edition defect removed in code review could be zero. This defect distribution is the result of use of code generation technology. The defect injected during different phases of development is shown in Fig.5.7. Most defects were injected during planning and coding phases. Only 14.86% defects were injected during design phase. Highest defect injection percentage was shown in coding phase. In this phase 45.95% defects were injected. However, planning phase introduced defect injection of 39.19%.

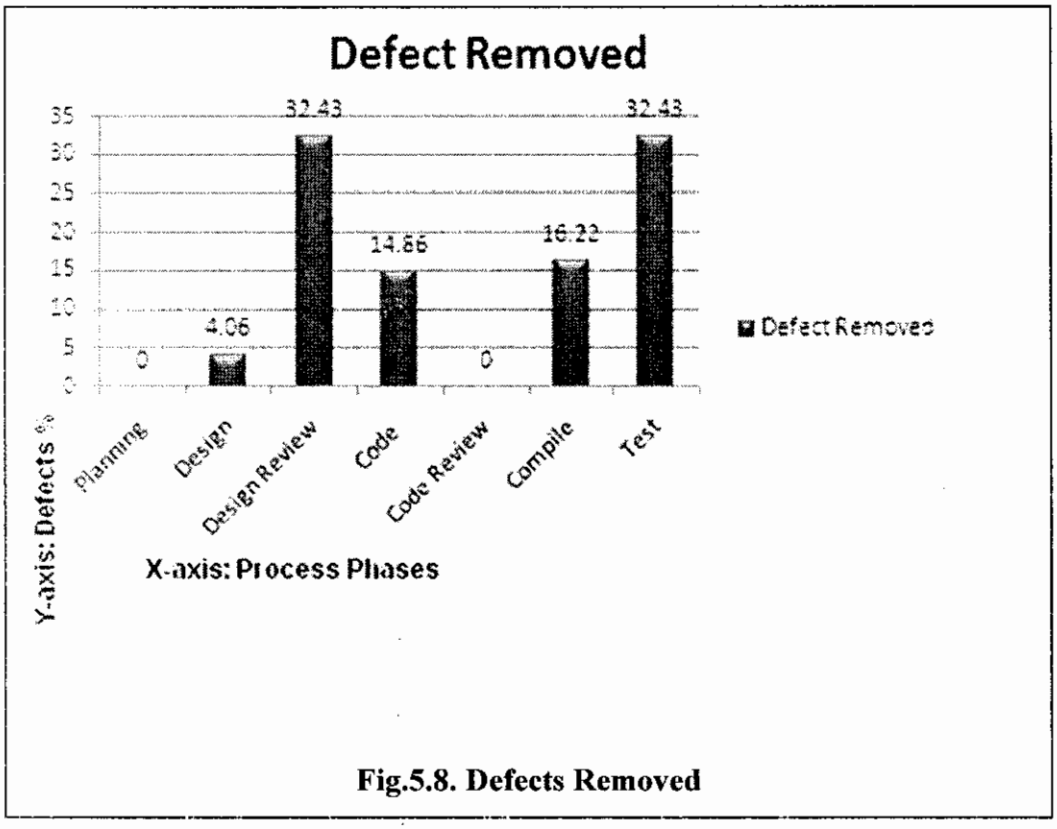


Fig.5.8. Defects Removed

Because a stable design process was used in PIP, which was found to be very helpful for defect removal. Percentage of defect removed in design phase was 4.05%. Similarly, design review was found to be very efficient where 32.43% defects were removed. Testing was also found to be efficient and 32.43% defects were removed. Fig.5.8 shows defects removed from different phases of software development process.

5.9 FINDINGS

The major finding of this case study regarding process design, implementation and execution are discussed in this section. PIP introduced *"Joint-Time"* concept for *"Agile"* software development. About 50% of the time was spent in planning and post-mortem. This time can be reduced by process automation. Use of Oracle 10g Express Database during case study execution was found to be useful in reducing injected defect during code review phase. The process provided accurate estimates of cost, schedule, defects and size. It was also found that the deployment phase is necessary as many defect which did not detected during compilation phase were appear in the deployment phase. This phase will be introduced in the next version of the process.

5.10 CONCLUSION

The process was designed and implemented in the university computer lab. The process was evaluated by executing a case study. The results of the process found to be exceptional. The major finding of process implementation results in better estimation of project cost, schedule, defects and software size. PIP is technology and domain aware process. This process was designed for database driven web-application. The process components for designing database were found to be very helpful during implementation.

CHAPTER 6

CASE STUDY

6.1 CASE STUDY DESCRIPTION

A case study was conducted to validate the effectiveness of the "Personal Integrated Process (PIP)". Details of case study are discussed in chapter 2. In this case study Personal Integrated Process was executed successfully, data regarding case study was collected and number of analyses were performed. In first section of the case study process automation and its impact is discussed. In second section of the case study analysis, results and finding regarding architecture design and evaluation processes are discussed. In third section analysis, results and finding regarding architecture based software system cost estimation is discussed.

6.2 Case Study Part-1: PROCESS AUTOMATION

Web-based software quality management system can provides fast, reliable, global accessible historical data for process improvement. Globally distributed teams of software engineers can store, access and perform analysis for process improvement. Such systems are currently been used in different software houses worldwide. These systems provide support in decision analysis and resolution, quality management, and process improvement. Automation of the software process can dramatically reduce development time. About 50% of the project time was spent in planning and post-mortem phases of the Personal Integrated Process (PIP). This was because the process contains bottom-up planning and top-down planning as well as bottom-up post-mortem and top-down post-mortem. Also PIP involves many calculations in planning, development and port-mortem which consume huge amount of process time. Manual work in PSP and PIP processes has many limitations and problems such as it cannot be accessed globally, process consume huge amount of time, results in increase in project cost, and slow access to historical data for analysis. Context switching is another problem in manual process. There was also found great emphasize for process automation in the literature:

"With CASE facilities to automatically log time, track defects, maintain data, and present statistical analyses, the PSP likely would be easier to learn and more efficient to use.[1]"

Therefore, a software quality management system is designed to address these limitations and problems.

Many software quality management systems or software tools have been proposed in the literature. Some of these tools were proposed for automation of Personal Software Process where as other were proposed for the automation of modified PSP. A literature survey was conducted to find the gap in the research literature details of this is given in the chapter 3. Out of 98 publications 30 were found to be related with the PSP automation or modification. However, none of these publications addressed PSP modification or automation with architecture design and evaluation components for individual software engineers. Therefore, *"Firmsoft Software Quality Management System (FSQMS)"* was designed to address the needs of process integration and automation.

6.2.1 Development Technology

A case study was conducted in the university computer lab to find the effectiveness of the proposed process. Detail of the case study has discussed in chapter 2. Here case study is discussed in brief. During case study execution a process was designed to address the limitation of PSP. The proposed process "PIP" was executed in the university computer lab. Process data was collected and analysis was performed to find the impact of process integration on cost, schedule, risks, defects, and size. However, in this chapter only process automation is discussed.

Software quality management system *"FSQMS"* was developed using OracleJDeveloper's *"Application Development Framework (ADF)"* [76]. Oracle *"Fusion Web Application"* technology was used in designing web interface. Oracle 10g Express Edition was used for database. Oracle was selected because it can be used in implementing n-tier architecture. The database was used for implementing online connection. Business tier was constructed using EJB 3.0. Model View

Controller was selected as architectural pattern. Java ServerFaces / facelets were used in implementing web-tier. Java Persistence API (JPA) was used in POJO persistence model. It was used for object-relational mapping. OracleJDeveloper was selected because it provides UML modeling facility. Business components such as EJB3.0 can be modeled using the technology. Task flow feature is used for designing page flow, navigation and security. The Oracle ADF provides two types of task flow one is unbounded task flow and other is bounded task flow. Bounded task flow is used in implementing region based permissions in ADF security. Oracle ADF Authentication and Authorization security model was used in implementing security of the software quality management system. ADF Form-Based security was selected which can be used to provide optimal security for users. Application roles were assigned to users and resource grants were given to the users. All these features when implemented require corresponding code in XML, JSF or Java.

6.2.2 System Major Components

FSQMS a software quality management system was designed to support Personal Software Process and Personal Integrated Process. Fig.6.1 shows system requirements of FSQMS in term of use cases. Two actors are involved in system operation one role is manager where as other role is engineers. Most of the activities are related with the engineer and manager can only view project plan summary.

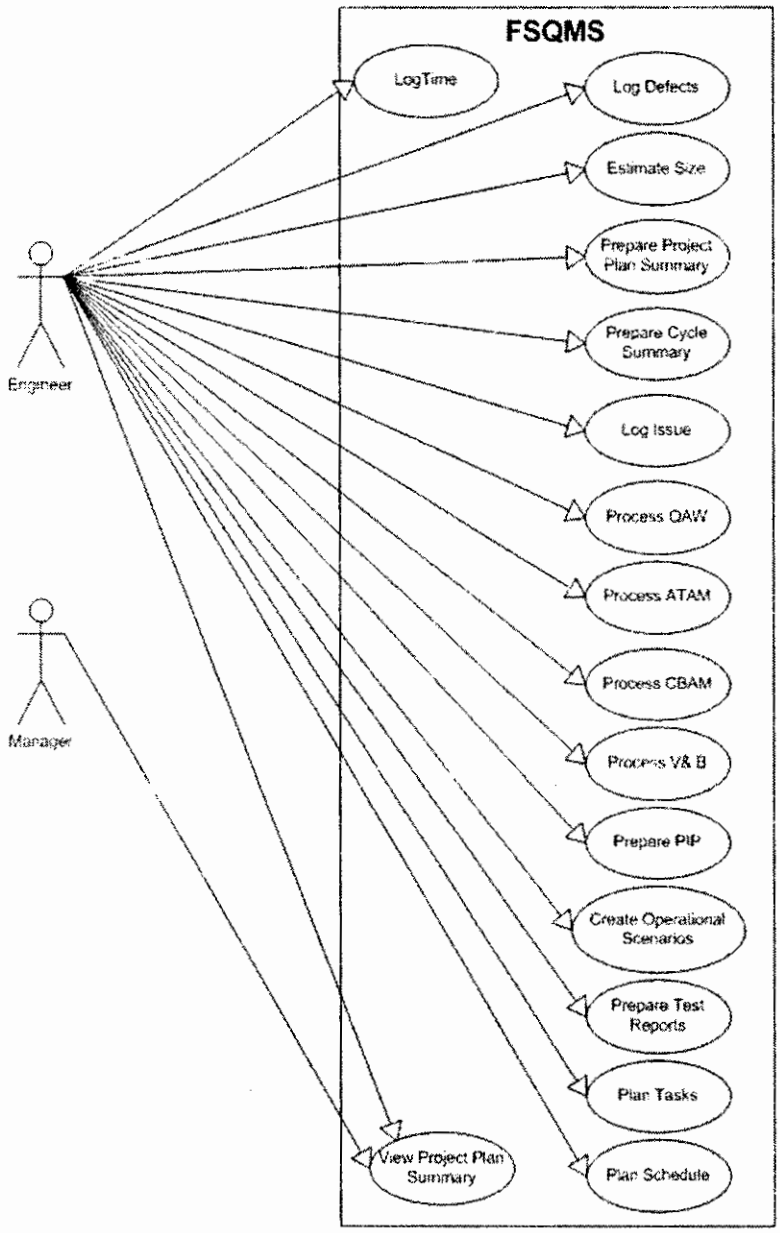


Fig.6.1. System Use Case Diagram

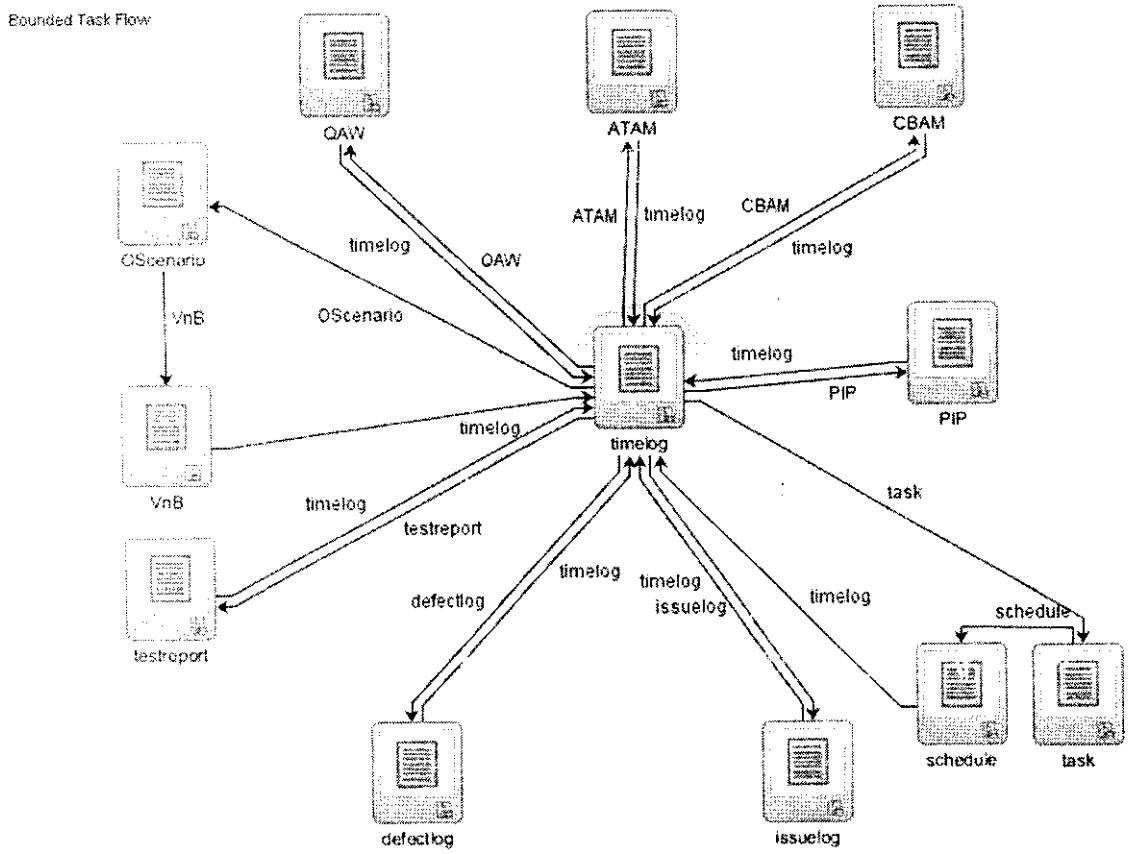


Fig.6.2. Bounded Task Flow

These use cases are implemented using OracleJDeveloper Fusion Web Application technology. Web user interface components their interactions, and control flows are shown in the Fig.6.2. The components are shown in the figure using OracleJDeveloper design notation. The OracleJDeveloper design notations are more expressive and informative than V&B notations. These visual components have complete traceability support with source code. Major components of the tool are discussed in detail one by one.

6.2.2.1 Time Log Component

Time log component *"timelog"* was designed using 3-tier architecture. Web-tier was implemented using JSF, business tier was implemented using ADF business components and database tier was implemented with Oracle 10g database. Time log component was designed to automate manual time logging. The component is created within the bounded task flow and can only be accessible for authenticated and authorized user.

6.2.2.2 Defect Log Component

Defect log component *"defectlog"* has same architecture as time logging component. The component is used for logging process defects.

6.2.2.3 Operational Scenario Component

Operational scenario component *"OScenario"* is designed and implemented for automating operational scenarios table in PSP and PIP.

6.2.2.4 Test Report Component

Test report component *"testreport"* automated the test reports involved in PSP and PIP.

6.2.2.5 Issue Tracking Component

Issue tracking component *"issuelog"* provides process automation support to issue log.

6.2.2.6 Task Planning Component

Task planning component *"task"* provides basic task planning of PSP and PIP.

6.2.2.7 Schedule Planning Component

Schedule planning component *"schedule"* is designed and implemented to support PSP and PIP schedule planning activities.

6.2.2.8 Process Improvement Proposal Component

Process improvement proposal component *"PIP"* provides automated support for both processes for preparing process improvement proposal.

6.2.2.9 QAW Component

QAW component *"QAW"* provides process support for Quality Attribute Workshop. The component is designed and implemented for PIP.

6.2.2.10 ATAM Component

ATAM component *"ATAM"* provides process support for Architecture Tradeoff Analysis Method and is the part of PIP.

6.2.2.11 CBAM Component

Like ATAM component CBAM component *"CBAM"* provides process automation support for Cost Benefit Analysis Method.

6.2.2.12 V&B Component

V&B component *"VnB"* is designed and implemented for PIP automation needs.

6.2.3 Results of Process Automation

All of these twelve components of FSQMS were implemented using same architectural pattern and open source software technology. Major development languages involved in developing software quality management system were SQL,

XML, JSF, and Java. About 3827 lines of code was consists of SQL, 3444 lines of code was consists of JSF and 364 lines of code was consists of XML. However, Java code generated automatically was not considered during project planning, post-mortem and analysis. Fig.6.3 shows programming languages contribution in developing software quality management system.

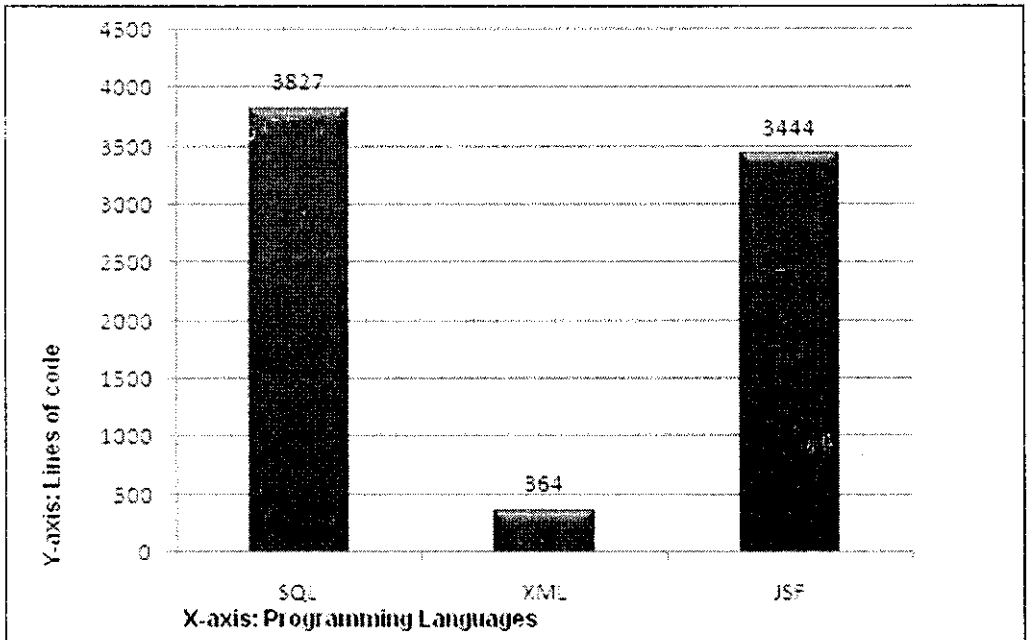


Fig.6.3. Programming Languages Contribution

6.2.4 Research Findings Regarding Process Automation

The integration of architecture design and evaluation in PSP was found to be very effective during tool development. Project cost, schedule, defects, and size were estimated and compared with the actual results and found to be extremely useful. Personal Integrated Process was automated with latest technology and resulting system was found to be highly secure. Technical risks were identified during development. Technical risks were mitigated using different strategies. These risks will not be identified if PSP was used. Project cost was estimated at architecture level. The cost estimation at architecture design and evaluation would be impossible if PSP was used.

6.2.5 Conclusion Drawn from Process Automation

Software quality management system such as "FSQMS" is highly important for software process improvement, project cost and schedule estimation. However, technology of software tool plays an important role in mitigation of technical risks. Architecture design and evaluation for such large and complex system is highly desirable. Personal Integrated Process with the support of architecture design and evaluation make such thing possible. The process automation support can reduce tremendous amount of time and project cost.

6.3 Case Study Part-1: Personal Integrated Process Implementation "ARCHITECTURE DESIGN"

Architecture design is the core activity of Personal Integrated Process. As discussed in the chapter 5, PIP is designed for individual software system engineers. Quality Attribute Workshop (QAW), Attribute Driven Design (ADD), and Active Reviews for Intermediate Design (ARID) are the process components used in architecture design process. These process components are discussed in this section one by one.

6.3.1 Quality Attribute Workshop

The process was tailored for the need of individual engineers. Nine use case scenario, two growth scenarios, and three exploratory scenarios were identified by QAW. Architectural drivers were identified, during brainstorming it was ensured the scenarios are in three part stimulus, environment and response. Similar scenarios were merged and priority of each scenario was established. After refinement scenarios were in six parts.

6.3.2 Attribute Driven Design

Inputs of the ADD were functional requirements, design constraints and quality attribute requirements. During process execution completeness of the requirements were made sure. System was decomposed in to number of elements and then these elements were further decomposed. Candidate architectural drivers were

identified. Design concepts were selected for these architectural drivers. Decomposed architectural elements were instantiated and responsibilities were allocated. Interface for the instantiated elements were defined. Finally verification was performed.

TABLE 6.1
Architectural Drivers

Candidate Architectural Drivers	Quality Attribute	Stakeholders	Stakeholders Priority	Impact on Architecture	Architectural Drivers
Scenario ID:1	Performance	End user	M	H	NO
Scenario ID:2	Security	End user Engineer	H	H	YES
Scenario ID:3	Availability	Engineer	L	H	NO
Scenario ID:4	Availability	Engineer	L	M	NO
Scenario ID:5	Reliability	Engineer	L	L	NO
Scenario ID:6	Modifiability	Engineer	L	L	NO
Scenario ID:7	Modifiability	Engineer	H	L	NO
Scenario ID:8	Reliability	Engineer	L	H	NO
Scenario ID:9	Availability	Engineer	H	H	YES
Design Constraints-1 Capacity Restriction	N/A	Engineer	H	H	YES
Design Constraints-2 Persistence Storage Service	N/A	Engineer	H	H	YES

Medium =M; High=H; Low=L

Table 6.1 shows the architectural drivers indentified in ADD process. Use case scenario 2, use case scenario 9, design constraint-1 and design constrain-2 were found to be architectural drivers. These candidate architectural drivers were considered to be architectural drivers because they had high stakeholder priority and high impact on architecture.

6.3.3 Active Reviews for Intermediate Design

ARID was performed when small portion of the system architecture was reviewed. Inputs to the process were quality attributes requirements, functional requirements, and portion of system architecture. During ARID seed scenarios were prepared, and prioritized after brainstorming. Finally review was performed using

question set. Issues and problems along with the reviewed portion of the architecture was the output of the process. All quality attributes were under discussion during ARID. Each quality attribute was discussed individually. The quality attribute that were under discussion include reliability, security, performance, and availability. Some of questions that were used during review are given below.

Q1: Introduce exception handling in the modules for reliability.

Q2: Write down the data type of each entity in a database table.

Q3: Maintain traceability of each object.

Q4: Ensure that the column name is not oracle reserved word.

Q5: Write down the foreign key constraints for each object.

Q6: Ensure there is no spelling mistake for each object.

Q7: Ensure foreign key is assigned to write entity.

Q8: Ensure users are mapped with appropriate role during application of Oracle ADF security.

Q9: Ensure application roles have appropriate resource grants.

Q10: Ensure application role has grant of appropriate bounded task flow before application of security.

Q11: Ensure application role has grant of all regions (web pages) in a bounded task flow for security.

Q12: Ensure connection pooling for performance reason.

Q13: Ensure application server is used for high availability.

6.4 Case Study Part-1: Personal Integrated Process Implementation "ARCHITECTURE EVALUATION"

Architecture Tradeoff Analysis Method is the core process component of Personal Integrated Process. The process was tailored for the needs of individual software system engineers. Quality attributes such as performance, security, availability and modifiability were evaluated by applying ATAM. Nine use case scenarios, two growth scenarios, and three exploratory scenarios identified and refined during QAW were the input to the ATAM. Other inputs to the process were business drivers and architecture of the system. Different architectural approaches were considered during the process. Quality attribute utility tree was generated, and

quality attribute specific questions were used to analyze architecture. Scenarios were prioritized after brainstorming. Outputs of the process were architectural views, sensitivity point, tradeoff point, and technical risks. During ATAM architecture business cycle was created, and quality attribute utility tree was constructed. Growth scenarios and exploratory scenarios after applying ATAM are given below.

Growth Scenarios:

1- Add a new data server to reduce latency within 2-person-week.

2- The size of table space of database is doubled without effecting the average retrieval time of one second.

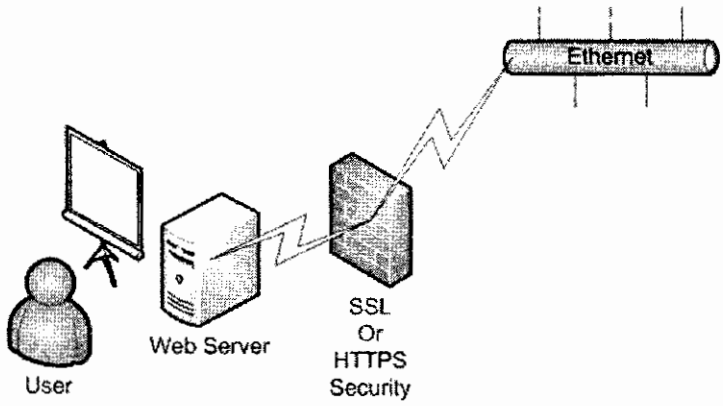
Exploratory Scenarios:

1- Server operating system may change from Microsoft Windows to Linux.

2- Improving the system availability from 99.9% to 99.999%.

3- One out of two backup servers goes down without effecting normal operation.

TABLE 6.2
System Use Case Scenario

Scenario ID:	UCS1
Scenario Description:	A PSP engineer query data from web server and data is displayed in less than 3 seconds.
Quality Attribute:	Performance
Risk:	R1- Connection pooling implementation decisions are not considered.
Tradeoff Point:	T1-100 concurrent connections can degrade performance. To achieve required number of concurrent connection will required increase number of servers. The will result in increase project cost.
Sensitivity Point:	S1- Concurrent connection might be sensitive to the bandwidth of the network.
Architectural Decision (Non-Risk):	Considering cost limitations only one server is considered for handling requests. Only ten concurrent connections will be allowed.
Reasoning:	Currently only one system is available.
Architecture Diagram:	

Use case scenarios were identified and prioritized using QAW process. These scenarios were further refined in term of six part using ADD process. Technical risks, sensitivity points and tradeoff points were identified by applying ATAM. These sensitivity points and tradeoff points were converted into non-risk by ATAM process. Use case scenarios after applying ATAM process is given in table.6.2.

TABLE 6.2 (Continued)
System Use Case Scenario

Scenario ID:	UCS2
Scenario Description:	The system allows login only to authorized and authenticated users.
Quality Attribute:	Security.
Risk:	R1- Decision regarding form based security is not considered yet.
Tradeoff Point:	T1- Form based security can take more time to develop components. This will result in increase in project cost.
Sensitivity Point:	S1-
Architectural Decision (Non-Risk):	Form base security will be implemented. Only two login will be implemented.
Reasoning:	Due to schedule constraints form based security will be implemented for only two users
Architecture Diagram:	N/A

TABLE 6.2 (Continued)
System Use Case Scenario

Scenario ID:	UCS3
Scenario Description:	A temperature spike causing hard disk failure and system is working in normal condition.
Quality Attribute:	Availability
Risk:	R1- Decision regarding software/hardware RAID is not considered yet.
Tradeoff Point:	T1- Achieving availability using software RAID may result in degrades in performance.
Sensitivity Point:	S1- Software RAID might be sensitive to the number of hard disks.
Architectural Decision (Non-Risk):	RAID will not be implemented i.e. only single drive server will be used.
Reasoning:	Due to cost constraints RAID will not be implemented. It will be considered in future projects
Architecture Diagram:	N/A

TABLE 6.2 (Continued) System Use Case Scenario	
Scenario ID:	UCS4
Scenario Description:	The system switch to another power supply after one power supply fails.
Quality Attribute:	Availability
Risk:	R1- Decision regarding dual power supply is not yet finalized.
Tradeoff Point:	T1- Achieving reliability using dual power supply will result in increase in cost.
Sensitivity Point:	S1-
Architectural Decision (Non-Risk):	Single power supply server will be used. Server with single power supply will be used.
Reasoning:	Considering cost constraints only one power supply per server is finalized
Architecture Diagram:	N/A

TABLE 6.2 (Continued) System Use Case Scenario	
Scenario ID:	UCS5
Scenario Description:	The web application server sent a time synchronization request, but unable to receive response.
Quality Attribute:	Reliability
Risk:	R1-
Tradeoff Point:	T1-
Sensitivity Point:	S1-
Architectural Decision (Non-Risk):	COST components will not be used.
Reasoning:	Implementation of COST component to synchronize time with internet time server is deferred
Architecture Diagram:	N/A

TABLE 6.2 (Continued) System Use Case Scenario	
Scenario ID:	UCS6
Scenario Description:	Software engineer want to add service component to synchronize the time of server with internet time within ten man-days.
Quality Attribute:	Modifiability
Risk:	R1-
Tradeoff Point:	T1-
Sensitivity Point:	S1-
Architectural Decision (Non-Risk):	COST components will not be used.
Reasoning:	Implementation of COST component to synchronize time with internet time server is deferred because of lack of development time.
Architecture Diagram:	N/A

TABLE 6.2 (Continued) System Use Case Scenario	
Scenario ID:	UCS7
Scenario Description:	Software engineer wants to add a COST to the system to support automated counting of line of code.
Quality Attribute:	Modifiability
Risk:	R1-
Tradeoff Point:	T1-
Sensitivity Point:	S1-
Architectural Decision (Non-Risk):	COST components will not be used.
Reasoning:	Implementation of COST component for automated line of code counting is deferred.
Architecture Diagram:	N/A

TABLE 6.2 (Continued) System Use Case Scenario	
Scenario ID:	UCS8
Scenario Description:	Exceptions occur during commit operation and notified to user by sound beeps and or text messages.
Quality Attribute:	Reliability
Risk:	R1- Decision regarding database architecture (centralized or distributed) is not yet considered.
Tradeoff Point:	T1- Distributed database implementation will increase in cost.
Sensitivity Point:	S1- Achieving availability using centralized database might be sensitive to the network availability.
Architectural Decision (Non-Risk):	Centralized database will be implemented i.e. single server will be used.
Reasoning:	Cost and schedule constraints.
Architecture Diagram:	N/A

TABLE 6.2 (Continued) System Use Case Scenario	
Scenario ID:	UCS9
Scenario Description:	Using web application server to achieve high availability.
Business Goal:	Return on Investment (ROI); Profitability; Customer Trust.
Quality Attribute:	Availability
Risk:	R1- Use of application server is not considered yet.
Tradeoff Point:	T1- Use of application server will result in increase in project cost.
Sensitivity Point:	S1- Using application server for availability might be sensitive to achieve performance.
Architectural Decision (Non-Risk):	Oracle WebLogic Server will be used as an application server.
Reasoning:	OracleJDeveloper IDE integration.
Architecture Diagram:	N/A

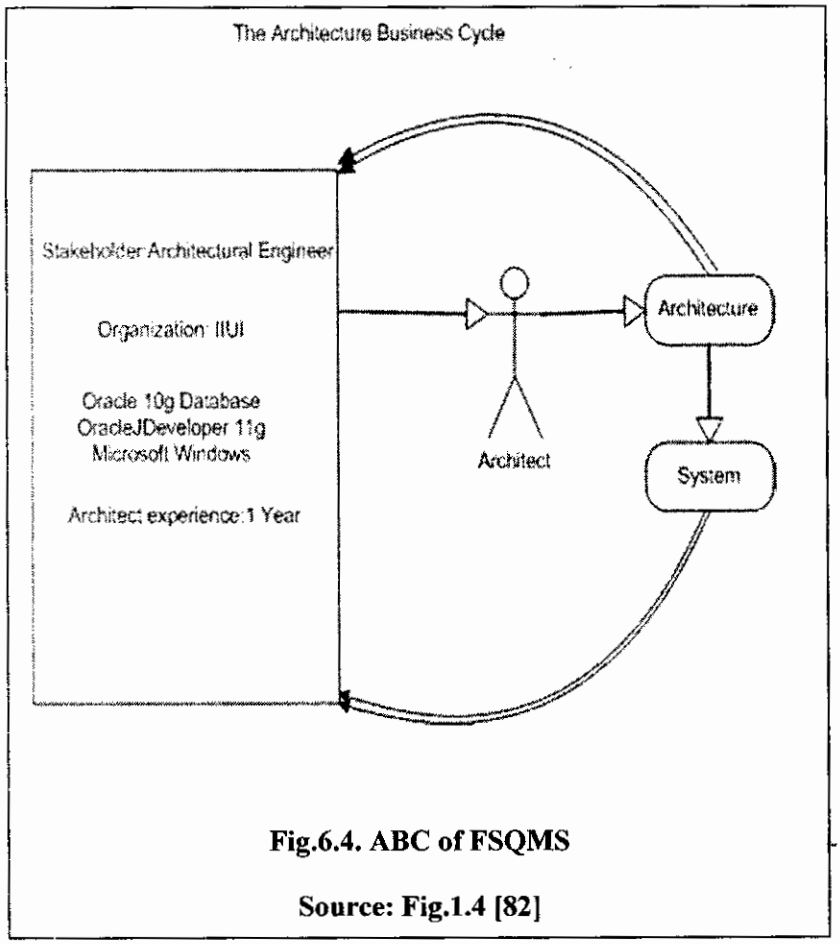


Fig.6.4. ABC of FSQMS

Source: Fig.1.4 [82]

During ATAM execution *"Architecture Business Cycle (ABC)"* constructed to identify experience of architect, organization, technology involved, and system. Fig. 6.4 shows the ABC of FSQMS and is adapted from Fig.1.4 [82]. From this figure experience of architect found to be one year, system engineer and end user was the stakeholder, and Oracle 10g Database was used for database, OracleJDeveloper 11g was used to construct JSF based front end, and quality attribute requirements were related with performance, security, reliability, availability and modifiability.

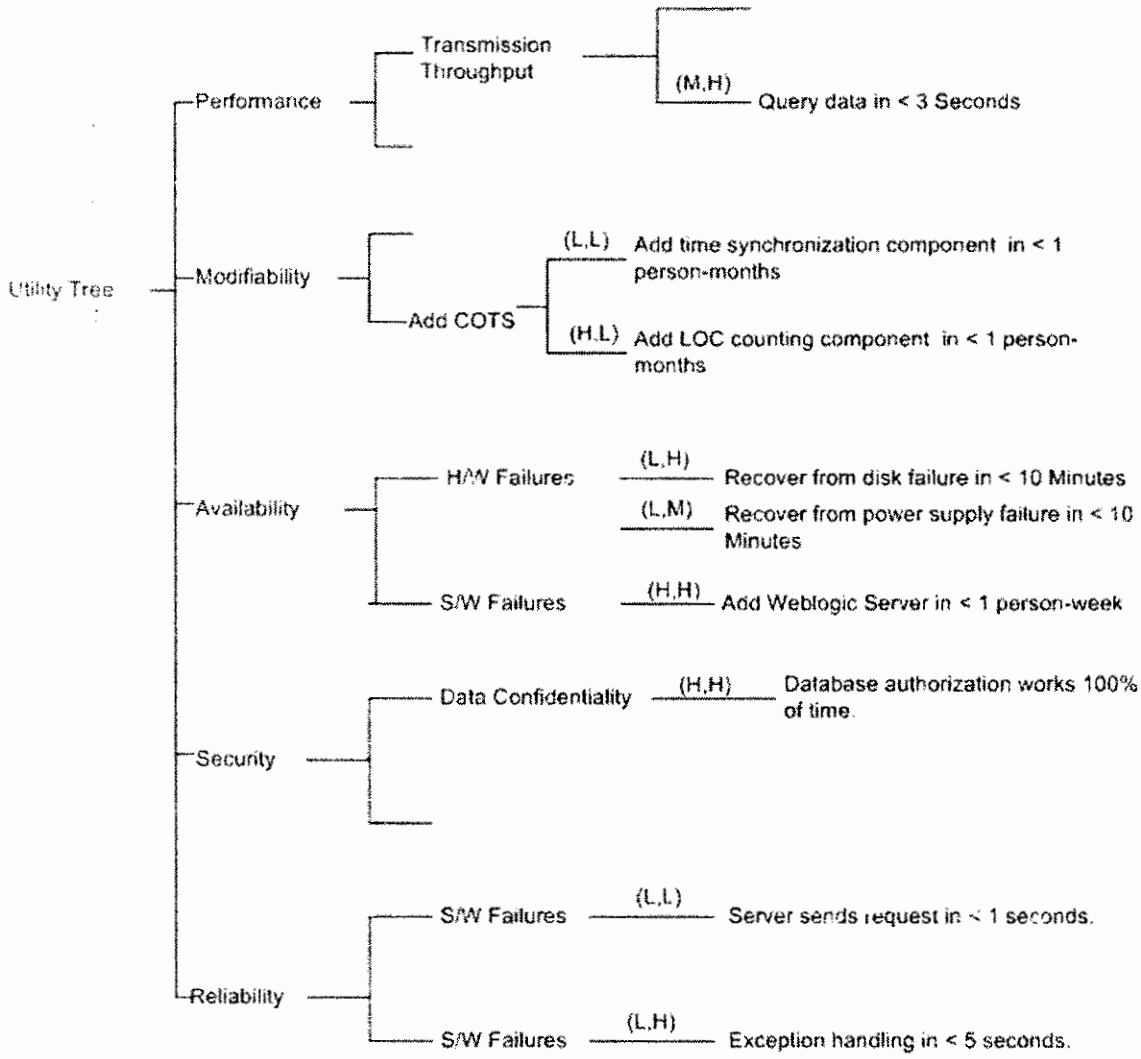


Fig.6.5. Utility Tree of FSQMS

Source: Table.3, pp.8 [83]

Construction of utility tree is one of the activities of ATAM process and is shown in Fig.6.5 this figure is adapted from Table.3 [83]. Utility tree is the graphical representation of quality attribute requirements need for a system. In case of FSQMS performance, modifiability, availability, security and reliability were the quality attribute requirements.

6.4.1 Performance

Performance of the system was evaluated manually without the use of any automated performance evaluation tool. This was because system involved only twelve web pages, and the performance was evaluated for only one user. The performance of the system was found to be satisfactory. However, in future it will be studied for 500 concurrent users.

6.4.2 Availability

Availability of the system was the second attribute to be studied. One availability risk was identified and deferred to reduce project cost. Second availability risk was identified and mitigated. Oracle WebLogic Server was used for achieving high availability.

6.4.3 Reliability

System reliability risks were identified and mitigated using OracleJDeveloper Fusion Web Application's built-in exception handling.

6.4.4 Modifiability

Modifiability risk were identified and deferred to reduce project development time.

6.4.5 Security

Use case scenario 2 was related with system security. Security risk were identified and then mitigated by applying Oracle ADF Form-Based security.

6.4.6 Results of Architecture Evaluation

By applying Personal Integrated Process (PIP) it was possible not only to identify the technical risk, but these risk were mitigated using different strategies. About 22.22% modifiability risks, 33.34% availability risks, 22.22% reliability risks, and 11.11% performance and security risks were identified. The technical risks and their frequency are shown in Fig.6.6.

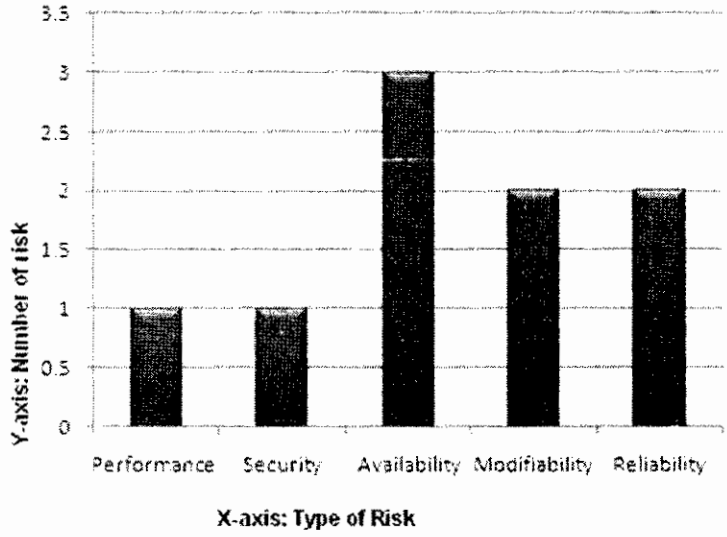


Fig.6.6. Technical Risk Analysis

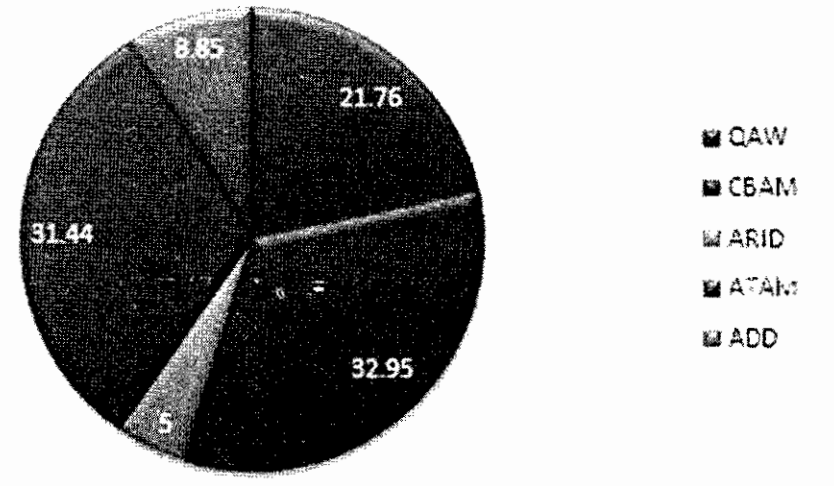


Fig.6.7. Time Distribution of Processes

The time spent in different process such as architecture design and evaluation is shown in Fig.6.7. Time spent in ATAM was about **31.44%**, QAW was **21.75%**, ADD was **8.85%**, and CBAM was **32.95%**.

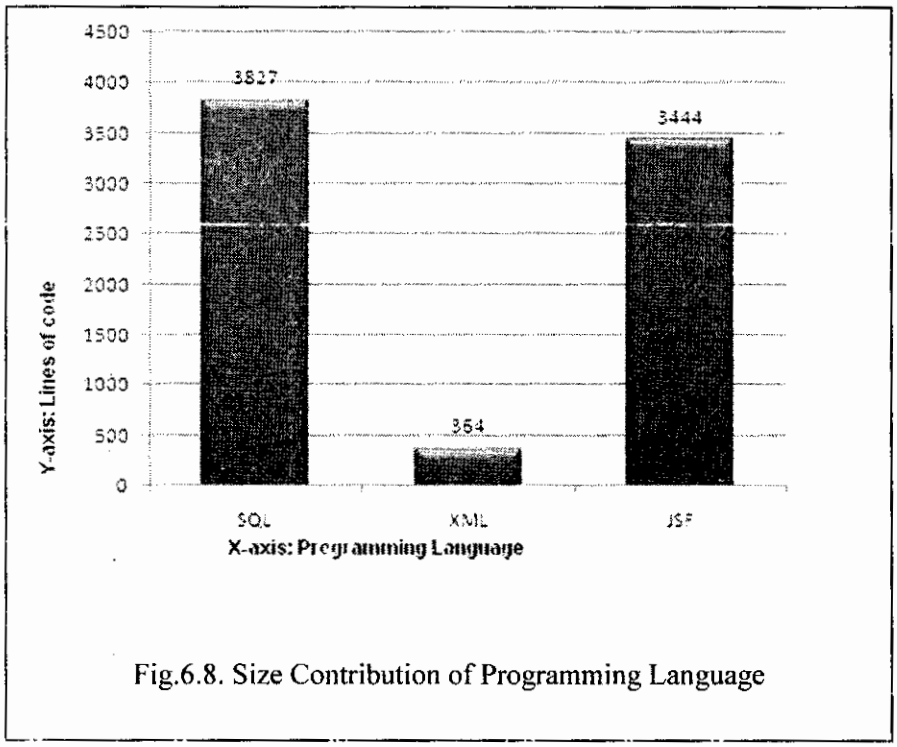


Fig.6.8. Size Contribution of Programming Language

Project time, cost and size were estimated accurately. For example the XML code used for implementing bounded task flow and security was about **364** lines of code. XML code along with SQL and JSF code is shown in Fig.6.8.

6.5 Case Study Part-1: Personal Integrated Process Implementation "COST AND BENEFIT EVALUATION"

Personal Integrated Process (PIP) is the extended form of Personal Software Process (PSP). PIP uses PROBE for estimation. Defects, size and schedule is estimated and planned in the same way as in PSP. However, PSP does not address architecture design and evaluation. These processes have many activities which do not involve coding. Therefore, such processes cannot precisely be estimated using PROBE.

PSP introduces the concept of estimation using expert judgment or estimation using historical data regarding process time, defects, or size. However, in case of non-coding process activities which consume substantial amount of time cannot be estimated using process time. Therefore, such process activities should be estimated using more precise approach such as activity based costing. Capers Jones has emphasized the use of activity based costing.

"Indeed, we now know that on some projects (such as large defence systems) the cost to produce paper documents is twice as much as the cost to produce the code itself". [77]

Keeping in this in view Personal Integrated Process (PIP) was designed to support activity based costing. It not only support PSP based estimation but also activity based cost estimation. However, it used lines of code as unit of measurement for software size instead of function point metric.

6.5.1 Cost Estimation

As architecture based cost and schedule estimation was the focus of the case study. A software quality management system was developed to find the impact of architecture design and evaluation on project cost and schedule. Cost Benefit Analysis Method (CBAM) is the process component used in Personal Integrated Process (PIP) to find the architecture based cost estimation. CBAM was tailored for the need of individual software engineers. The resulting process component was used to analyze cost and benefit of architectural strategies. Before the use of CBAM for cost and schedule estimation ATAM was applied to find the architectural risks. Quality attribute goals, high priority scenarios, utility tree, architectural views, and architectural risks were the inputs to the CBAM. Detail of application ATAM are given in the chapter7. Selection of the architectural strategies for each use case scenario was the objective of the first step of the CBAM. To achieve the objectives appropriate architectural strategies were selected for each use case scenario. For each architectural strategy desired improvement to the architecture was identified. These architectural strategies along with desired improvements are given in the table 6.3.

PSP introduces the concept of estimation using expert judgment or estimation using historical data regarding process time, defects, or size. However, in case of non-coding process activities which consume substantial amount of time cannot be estimated using process time. Therefore, such process activities should be estimated using more precise approach such as activity based costing. Capers Jones has emphasized the use of activity based costing.

"Indeed, we now know that on some projects (such as large defence systems) the cost to produce paper documents is twice as much as the cost to produce the code itself". [77]

Keeping in this in view Personal Integrated Process (PIP) was designed to support activity based costing. It not only support PSP based estimation but also activity based cost estimation. However, it used lines of code as unit of measurement for software size instead of function point metric.

6.5.1 Cost Estimation

As architecture based cost and schedule estimation was the focus of the case study. A software quality management system was developed to find the impact of architecture design and evaluation on project cost and schedule. Cost Benefit Analysis Method (CBAM) is the process component used in Personal Integrated Process (PIP) to find the architecture based cost estimation. CBAM was tailored for the need of individual software engineers. The resulting process component was used to analyze cost and benefit of architectural strategies. Before the use of CBAM for cost and schedule estimation ATAM was applied to find the architectural risks. Quality attribute goals, high priority scenarios, utility tree, architectural views, and architectural risks were the inputs to the CBAM. Detail of application ATAM are given in the chapter7. Selection of the architectural strategies for each use case scenario was the objective of the first step of the CBAM. To achieve the objectives appropriate architectural strategies were selected for each use case scenario. For each architectural strategy desired improvement to the architecture was identified. These architectural strategies along with desired improvements are given in the table 6.3.

TABLE 6.3
Architectural Strategies and Desired Improvements

ID	QA Goal	Architectural Strategy ID	Architectural Strategy Description	Desired Improvement
UCS2	Security	AS-001	Using hard coded security	Reducing development time.
UCS2	Security	AS-002	Using JAAS Permission for business tier.	Increasing business tire security.
UCS2	Security	AS-003	Using hard coded basic authentication.	Reducing development time by using integrated authorization.
UCS2	Security	AS-004	ADF Authentication	Using built-in facility.
UCS2	Security	AS-005	HTTP Basic Authentication	Reducing development time.
UCS2	Security	AS-006	HTTP Digest Authentication	Reducing development time.
UCS2	Security	AS-007	HTTPS Client Authentication	Using enhanced security.
UCS2	Security	AS-008	Form-Based Authentication	Using optimal security

TABLE 6.4
Quality Attribute Score

Scenario ID	Quality Attribute	Quality Attribute % QAscore	Stakeholder Priority		Impact on Architecture		Total
UCS1	Performance	13.88	M	2	H	3	5
UCS2	Security	16.67	H	3	H	3	6
UCS3	Availability	11.11	L	1	H	3	4
UCS4	Reliability	8.33	L	1	M	2	3
UCS5	Reliability	5.56	L	1	L	1	2
UCS6	Modifiability	5.56	L	1	L	1	2
UCS7	Modifiability	11.11	H	3	L	1	4
UCS8	Reliability	11.11	L	1	H	3	4
UCS9	Availability	16.67	H	3	H	3	6
	Total % Score	100	Total Score				36

After identifying desired improvement to the architecture, quality attribute score is established based on stakeholder priority and impact on the architecture. Quality attribute score along with priority of stakeholder and impact on architecture is given in table 6.4.

Step-3: Benefit Score Calculation

Third step of the CBAM was to find the benefit score for each architectural strategy.

TABLE 6.5
Benefit Evaluation

Scenario ID	Quality Attribute	Contribution Cont	Architecture Strategy	QAscore	Benefit Score= QAscore x Cont
UCS1	Performance	+1	AS-008	13.88	13.88
UCS2	Security	+1	AS-008	16.67	16.67
UCS3	Availability	+1	AS-008	11.11	11.11
UCS4	Reliability	+1	AS-008	8.33	8.33
UCS5	Reliability	+1	AS-008	5.56	5.56
UCS6	Modifiability	+1	AS-008	5.56	5.56
UCS7	Modifiability	+1	AS-008	11.11	11.11
UCS8	Reliability	+1	AS-008	11.11	11.11
UCS9	Availability	+1	AS-008	16.67	16.67
Total Benefit Score for Architecture Strategy AS-008					+100 Benys
UCS1	Performance	+1	AS-007	13.88	13.88
UCS2	Security	+.5	AS-007	16.67	8.34
UCS3	Availability	+1	AS-007	11.11	11.11
UCS4	Reliability	+1	AS-007	8.33	8.33
UCS5	Reliability	+1	AS-007	5.56	5.56
UCS6	Modifiability	+1	AS-007	5.56	5.56
UCS7	Modifiability	+1	AS-007	11.11	11.11
UCS8	Reliability	+1	AS-007	11.11	11.11
UCS9	Availability	+1	AS-007	16.67	16.67
Total Benefit Score for Architecture Strategy AS-007					+91.67 Benys
UCS1	Performance	+1	AS-006	13.88	13.88
UCS2	Security	+.5	AS-006	16.67	8.34
UCS3	Availability	+1	AS-006	11.11	11.11
UCS4	Reliability	+1	AS-006	8.33	8.33
UCS5	Reliability	+1	AS-006	5.56	5.56
UCS6	Modifiability	+1	AS-006	5.56	5.56
UCS7	Modifiability	+1	AS-006	11.11	11.11
UCS8	Reliability	+1	AS-006	11.11	11.11
UCS9	Availability	+1	AS-006	16.67	16.67
Total Benefit Score for Architecture Strategy AS-006					+91.67 Benys
UCS1	Performance	+1	AS-005	13.88	13.88
UCS2	Security	+.2	AS-005	16.67	3.34
UCS3	Availability	+1	AS-005	11.11	11.11
UCS4	Reliability	+1	AS-005	8.33	8.33
UCS5	Reliability	+1	AS-005	5.56	5.56
UCS6	Modifiability	+1	AS-005	5.56	5.56
UCS7	Modifiability	+1	AS-005	11.11	11.11
UCS8	Reliability	+1	AS-005	11.11	11.11
UCS9	Availability	+1	AS-005	16.67	16.67
Total Benefit Score for Architecture Strategy AS-005					+86.67 Benys

Benefit score was determined by multiplying quality attribute score with its contribution score and resulting *"benys"* are given in the table 6.5.

TABLE 6.5 (Continued) Benefit Evaluation					
Scenario ID	**Quality Attribute**	**Contribution Cont**	**Architecture Strategy**	**QAscore**	**Benefit Score= QAscore x Cont**
UCS1	Performance	+1	AS-004	13.88	13.88
UCS2	Security	+.4	AS-004	16.67	6.67
UCS3	Availability	+1	AS-004	11.11	11.11
UCS4	Reliability	+1	AS-004	8.33	8.33
UCS5	Reliability	+1	AS-004	5.56	5.56
UCS6	Modifiability	+1	AS-004	5.56	5.56
UCS7	Modifiability	+1	AS-004	11.11	11.11
UCS8	Reliability	+1	AS-004	11.11	11.11
UCS9	Availability	+1	AS-004	16.67	16.67
Total Benefit Score for Architecture Strategy AS-004					+90 Benys
UCS1	Performance	+1	AS-003	13.88	13.88
UCS2	Security	+.4	AS-003	16.67	6.67
UCS3	Availability	+1	AS-003	11.11	11.11
UCS4	Reliability	+1	AS-003	8.33	8.33
UCS5	Reliability	+1	AS-003	5.56	5.56
UCS6	Modifiability	+1	AS-003	5.56	5.56
UCS7	Modifiability	+1	AS-003	11.11	11.11
UCS8	Reliability	+1	AS-003	11.11	11.11
UCS9	Availability	+1	AS-003	16.67	16.67
Total Benefit Score for Architecture Strategy AS-003					+90 Benys
UCS1	Performance	+1	AS-002	13.88	13.88
UCS2	Security	+.2	AS-002	16.67	3.34
UCS3	Availability	+1	AS-002	11.11	11.11
UCS4	Reliability	+1	AS-002	8.33	8.33
UCS5	Reliability	+1	AS-002	5.56	5.56
UCS6	Modifiability	+1	AS-002	5.56	5.56
UCS7	Modifiability	+1	AS-002	11.11	11.11
UCS8	Reliability	+1	AS-002	11.11	11.11
UCS9	Availability	+1	AS-002	16.67	16.67
Total Benefit Score for Architecture Strategy AS-002					+86.67 Benys
UCS1	Performance	+1	AS-001	13.88	13.88
UCS2	Security	+.1	AS-001	16.67	1.67
UCS3	Availability	+1	AS-001	11.11	11.11
UCS4	Reliability	+1	AS-001	8.33	8.33
UCS5	Reliability	+1	AS-001	5.56	5.56
UCS6	Modifiability	+1	AS-001	5.56	5.56
UCS7	Modifiability	+1	AS-001	11.11	11.11
UCS8	Reliability	+1	AS-001	11.11	11.11
UCS9	Availability	+1	AS-001	16.67	16.67
Total Benefit Score for Architecture Strategy AS-001					+85 Benys

Step-4: Cost and Schedule Analysis

Forth step of the CBAM is to determine the project cost and schedule implications.

TABLE 6.6
Cost and Schedule Evaluation

Scenario ID	Quality Attribute	% Time Required for Implementation	Effort (man-day)	Cost US $ $1000/ 30days	Architecture Strategy
UCS1	Performance	23.33	7	233.31	AS-008
UCS2	Security	33.33	10	333.3	AS-008
UCS3	Availability	1.67	0.5	16.665	AS-008
UCS4	Reliability	1.67	0.5	16.665	AS-008
UCS5	Reliability	1.67	0.5	16.665	AS-008
UCS6	Modifiability	1.67	0.5	16.665	AS-008
UCS7	Modifiability	1.67	0.5	16.665	AS-008
UCS8	Reliability	6.67	2	66.66	AS-008
UCS9	Availability	3.34	1	33.33	AS-008
		75.02%	Total Days = 22.5/30	749.925	AS-008
UCS1	Performance	23.33	7	233.31	AS-007
UCS2	Security	53.33	16	533.28	AS-007
UCS3	Availability	1.67	0.5	16.665	AS-007
UCS4	Reliability	1.67	0.5	16.665	AS-007
UCS5	Reliability	1.67	0.5	16.665	AS-007
UCS6	Modifiability	1.67	0.5	16.665	AS-007
UCS7	Modifiability	1.67	0.5	16.665	AS-007
UCS8	Reliability	6.67	2	66.66	AS-007
UCS9	Availability	3.34	1	33.33	AS-007
		95.02%	Total Days = 28.5/30	949.905	AS-007
UCS1	Performance	23.33	7	233.31	AS-006
UCS2	Security	13.33	4	133.32	AS-006
UCS3	Availability	1.67	0.5	16.665	AS-006
UCS4	Reliability	1.67	0.5	16.665	AS-006
UCS5	Reliability	1.67	0.5	16.665	AS-006
UCS6	Modifiability	1.67	0.5	16.665	AS-006
UCS7	Modifiability	1.67	0.5	16.665	AS-006
UCS8	Reliability	6.67	2	66.66	AS-006
UCS9	Availability	3.34	1	33.33	AS-006
		55.02%	Total Days = 16.5/30	549.945	AS-006
UCS1	Performance	23.33	7	233.31	AS-005
UCS2	Security	13.33	4	133.32	AS-005
UCS3	Availability	1.67	0.5	16.665	AS-005
UCS4	Reliability	1.67	0.5	16.665	AS-005
UCS5	Reliability	1.67	0.5	16.665	AS-005
UCS6	Modifiability	1.67	0.5	16.665	AS-005
UCS7	Modifiability	1.67	0.5	16.665	AS-005
UCS8	Reliability	6.67	2	66.66	AS-005
UCS9	Availability	3.34	1	33.33	AS-005
		55.02%	Total Days = 16.5/30	549.945	AS-005

Time required for implementing each quality attribute for each architectural strategy was calculated along with the effort required for development. Project cost was determined for each architectural strategy and results are given in the table 6.6.

TABLE 6.6 (Continued)
Cost and Schedule Evaluation

Scenario ID	Quality Attribute	% Time Required for Implementation	Effort (man-day)	Cost US $ $1000/ 30days	Architecture Strategy
UCS1	Performance	23.33	7	233.31	AS-004
UCS2	Security	33.33	10	333.3	AS-004
UCS3	Availability	1.67	0.5	16.665	AS-004
UCS4	Reliability	1.67	0.5	16.665	AS-004
UCS5	Reliability	1.67	0.5	16.665	AS-004
UCS6	Modifiability	1.67	0.5	16.665	AS-004
UCS7	Modifiability	1.67	0.5	16.665	AS-004
UCS8	Reliability	6.67	2	66.66	AS-004
UCS9	Availability	3.34	1	33.33	AS-004
		75.02%	Total Days = 22.5/30	749.925	AS-004
UCS1	Performance	23.33	7	233.31	AS-003
UCS2	Security	10	3	99.99	AS-003
UCS3	Availability	1.67	0.5	16.665	AS-003
UCS4	Reliability	1.67	0.5	16.665	AS-003
UCS5	Reliability	1.67	0.5	16.665	AS-003
UCS6	Modifiability	1.67	0.5	16.665	AS-003
UCS7	Modifiability	1.67	0.5	16.665	AS-003
UCS8	Reliability	6.67	2	66.66	AS-003
UCS9	Availability	3.34	1	33.33	AS-003
		51.69%	15.5/30	516.615	AS-003
UCS1	Performance	23.33	7	233.31	AS-002
UCS2	Security	56.66	17	566.61	AS-002
UCS3	Availability	1.67	0.5	16.665	AS-002
UCS4	Reliability	1.67	0.5	16.665	AS-002
UCS5	Reliability	1.67	0.5	16.665	AS-002
UCS6	Modifiability	1.67	0.5	16.665	AS-002
UCS7	Modifiability	1.67	0.5	16.665	AS-002
UCS8	Reliability	6.67	2	66.66	AS-002
UCS9	Availability	3.34	1	33.33	AS-002
		98.35%	29.5/30	983.235	AS-002
UCS1	Performance	23.33	7	233.31	AS-001
UCS2	Security	16.66	5	166.65	AS-001
UCS3	Availability	1.67	0.5	16.665	AS-001
UCS4	Reliability	1.67	0.5	16.665	AS-001
UCS5	Reliability	1.67	0.5	16.665	AS-001
UCS6	Modifiability	1.67	0.5	16.665	AS-001
UCS7	Modifiability	1.67	0.5	16.665	AS-001
UCS8	Reliability	6.67	2	66.66	AS-001

TABLE 6.7 Software and Licensing Cost			
Item	**Software Cost US $**	**License Cost US $**	**Total Cost US $**
Oracle 11g Database	70	17500	17570
Oracle WebLogic Suite	180	45000	45180
Oracle Java	100		100
Total Cost of All Items			62850

In the next step cost of software components was calculated and is given in the table 6.7. Finally the project cost was calculated and given below.

Total cost of development: Development time x cost per month = 12 x 1000$ = 12000 US $.

Total cost of equipments: 450 US $.

Total cost of project: 75300 US $.

Step-5: ROI Calculation

In fifth step Return on Investment (ROI) is calculated for each architectural strategy and is given in the table 6.8.

TABLE 6.8 Return on Investment (ROI)			
Architectural Strategy	**Cost (US $)**	**Benefit (Benys)**	**ROI Benefit/Cost (Benys/$)**
AS-001	583.275	85	0.1457
AS-002	983.235	86.67	0.0881
AS-003	516.615	90	0.1742
AS-004	749.925	90	0.1200
AS-005	549.945	86.67	0.1576
AS-006	549.945	91.67	0.1667
AS-007	949.905	91.67	0.0965
AS-008	749.925	100	0.1333

Step-6: Cost and Benefit Analysis of Architectural Strategies

Sixth step of the process involve cost-benefit analysis of all architectural strategies. In case of software quality management system architectural strategy AS-3 and AS-6 have high Benefit/Cost. However, AS-8 and AS-7 have high benefit and filtered for further considerations. However, architectural strategy AS-8 was selected when system was implemented. This strategy was selected based on the security requirement for the software quality management system. The strategy was based on the implementation of Oracle ADF Form-Based security. Fig.6.9 shows the cost benefit analysis for all architectural strategies this figure is partially adapted from Fig.2 [62].

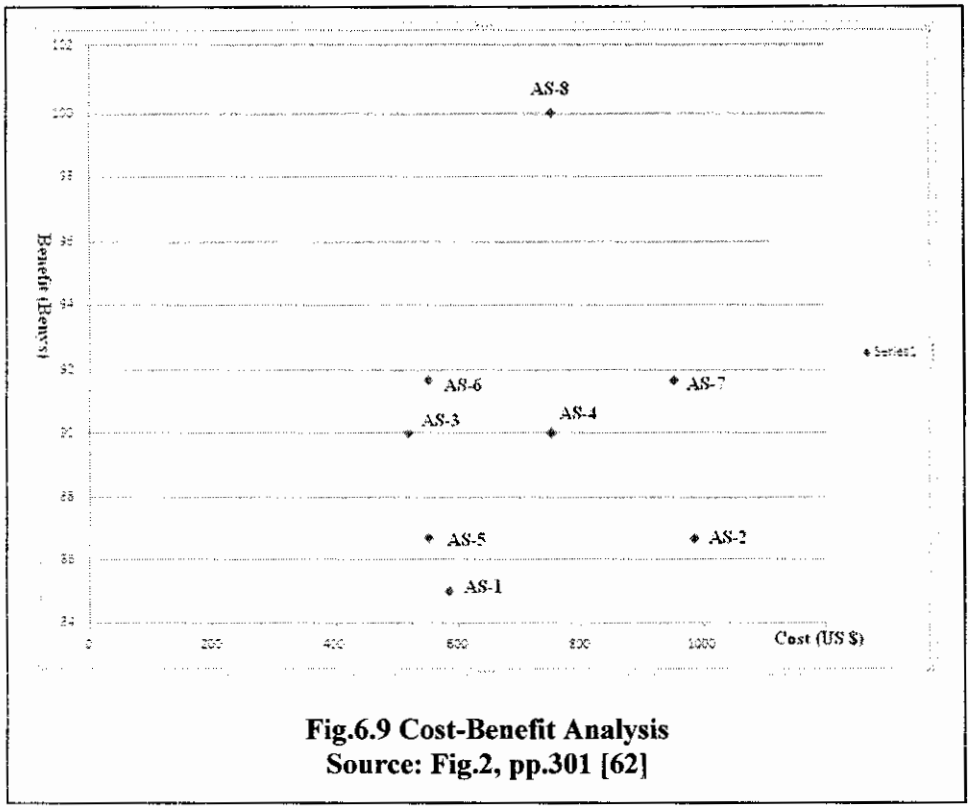


Fig.6.9 Cost-Benefit Analysis
Source: Fig.2, pp.301 [62]

6.5.2 Return on Investment (ROI) for Architectural Strategies

Return on Investment (ROI) for each architectural strategy is calculated by dividing total benefit B_i to the Cost C_i of implementing it i.e. $Ri = B_i/C_i$

Example: $R_{AS\text{-}001} = 85/583.275 = 0.1457$

6.5.3 Cost Estimation Results

CBAM was found to be extremely helpful for architecture base software cost and schedule estimation.

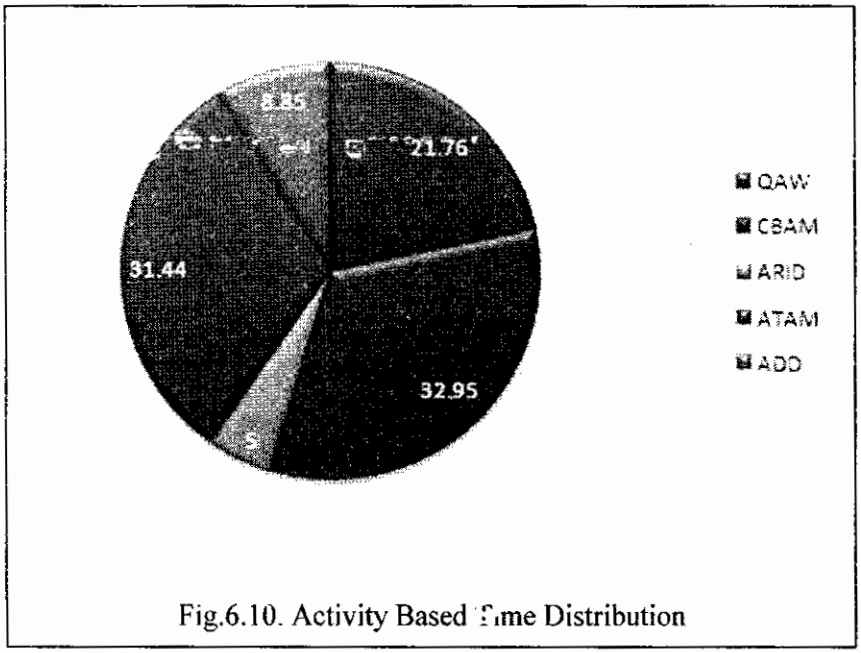


Fig.6.10. Activity Based Time Distribution

All the activities in this process component were non-coding and could not be estimated if Personal Software Process was used for estimation of cost or schedule. Therefore, activity based cost estimation was found to be very helpful for project cost and schedule estimation. The time distribution among different phase of the process is shown in Fig.6.10. CBAM took about 32.95% time, ATAM utilized 31.44%, QAW 21.76%, ADD 8.85%, and ARID consumed 5% of the project time.

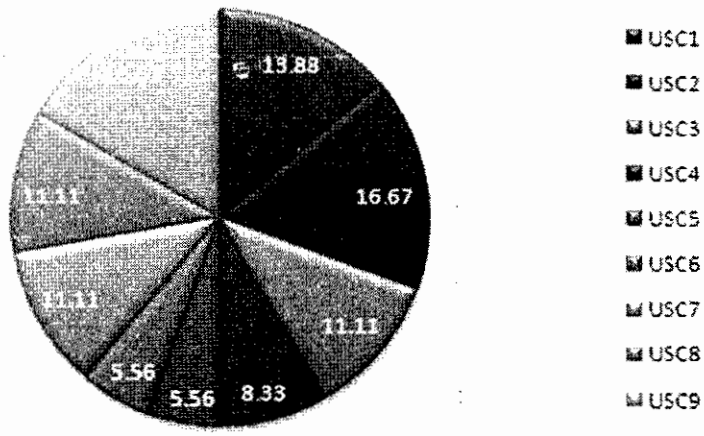


Fig.6.11. Quality Attribute Score Contribution

Use case scenario UCS2 concerning security and UCS9 concerning availability have high contribution and their quality attribution score is 16.67%. Quality attributes score contribution for all use case scenarios is shown in Fig.6.11.

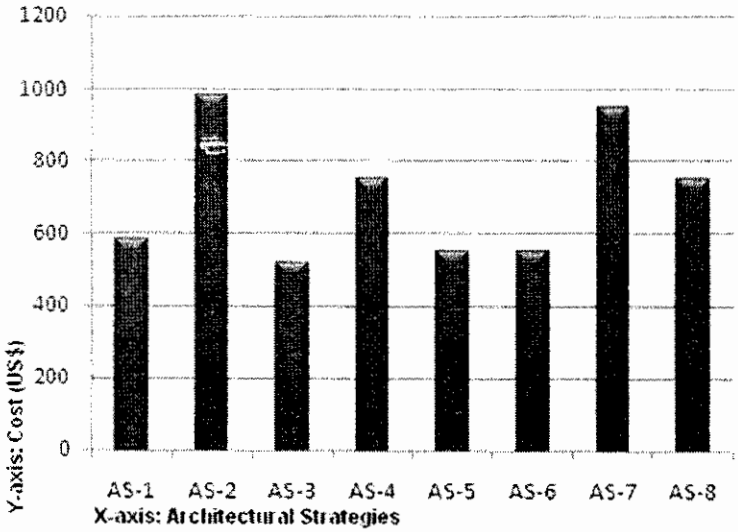


Fig.6.12 Cost Analysis

Architectural strategy AS-2 and AS-7 require more cost than AS-3, AS-5 and AS-6. However, AS-3 and AS-7 have high benefit/cost. Project cost requirement for all the architectural strategies is shown in Fig.6.12.

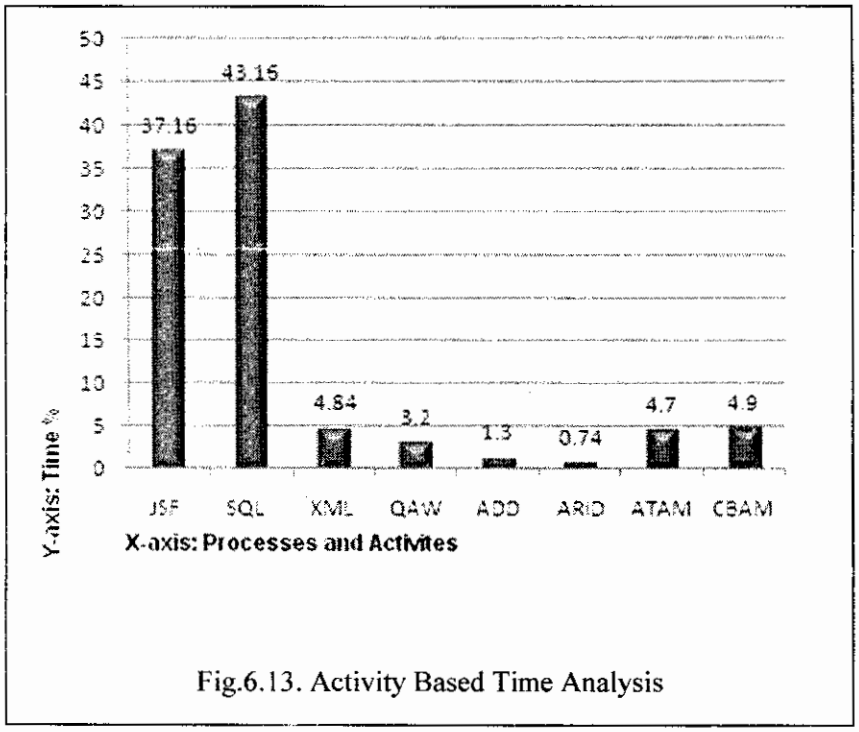


Fig.6.13. Activity Based Time Analysis

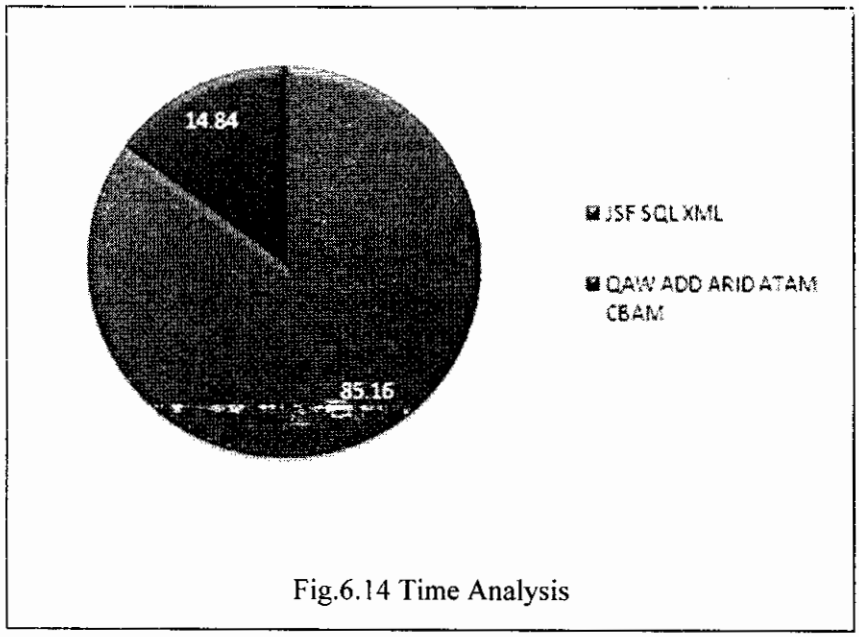


Fig.6.14 Time Analysis

The activity based time analyses were performed to estimate the project cost. These analyses are shown in Fig.6.13, and Fig.6.14. 85.16% project time was spent in coding activities such as XML, SQL and JSF. 14.84% time was spent in architecture

design and evaluation. These processes includes QAW, ADD, ARID, ATAM and CBAM.

6.5.4 Research Findings Regarding Cost Estimation

The integration of architecture design and evaluation processes were found to be extremely helpful for architecture based cost and schedule estimation. If PSP were used, cost of project at architecture level would be inaccurate and misleading. Also project risk were identified and mitigated before detailed design. If these risks were unidentified, stakeholder may have requirement conflicts. Furthermore project schedule and cost would be impossible to estimate accurately.

6.6 Case Study Part-2: Personal Software Process

Personal Software Process (PSP) was executed first and then Personal Integrated Process (PIP) was executed to find the impact of process change. Since PSP3 is cyclic and stand alone process and contains forms and scripts of all seven processes of standard PSP, therefore, PSP3 was integrated in PIP. Standard PSP contains seven processes PSP0, PSP0.1, PSP1, PSP1.1, PSP2, and PSP2.1 and executed separately. In this section only the standard PSP process will be discussed.

6.6.6 Case Study Part-2: Personal Software Process Results

In **PSP0** basic measures such as time and defects are involved. A program was developed in Structured Query Language (SQL) which took 7.75% time for planning, 23.26% time for design, 58.91% time for code, 1.94% time for compile, 1.94% time for test, 6.2% time for post-mortem. Only one defect was injected during code and removed in compile phase.

In **PSP0.1** seven programs were developed. Basic measures such as time, defects and program size was measured for all programs developed in PSP0.1. These seven programs contain reuse (R) value of 476 lines of code. Total new and changed lines of code (N) were found to be 452, and total lines of code (T) were found to be 932. The *"to-date percentage time"* spent in planning was 10.74%, in design it was 16.61%, in code it was 34.6%, in compile it was 2%, in test it was 15.14%, and in post-mortem it was 20.91%. The *"to-date percentage defects injected"* in design was 37.5% and in code it was 62.5%. However, no defect was injected in planning,

compile and test phases. The *"to-date percentage defects removed"* in compile was 50% and in test it was 50%.

In **PSP1** only one program was developed. This is because in standard PSP each process is different from other. Due to process changes it was not possible to accurately and precisely estimate the program size, defects, and time. Results of PSP1 are discussed in this paragraph. Productivity for this program was 29.22 line of code/hour. 255 lines of code were added (A) in the system. To-date value for *"total new and changed line of code (N)"* was 255, for *"total line of code (T)"* it was 323, and for *"estimated object lines of code (E)"* it was 312. To-date percentage value for time spent in planning was 45.7%, in design it was 10.8%, in code it was 20.3%, in compile it was 1.11%, in test it was 17.5%, and in post-mortem it was 4.6%. To-date percentage value for defect injected in design was 100%. To-date percentage value for defect removed in test was 100%.

In **PSP1.1** actual productivity was 41.00 lines of code/hour. Cost Performance Index (CPI) was 1.05, 723 lines of code was added (A) in the system. To-date value for total new and changed lines of code (N) was 723. To-date value for total lines of code (T) was 791. To-date value for percentage reused lines of code was 8.6%. To-date value for estimated object lines of code (E) was 616. To-date value for time spent in planning was 28.35%, in design it was 10.4%, in code it was 46.32%, in compile it was 1.89%, in test it was 6.9%, and in post-mortem it was 6.14%. To-date percentage value for defect injected in design was 40% and in code it was 60%. To-date percentage value for defect removed in code was 20%, and in compile it was 80%.

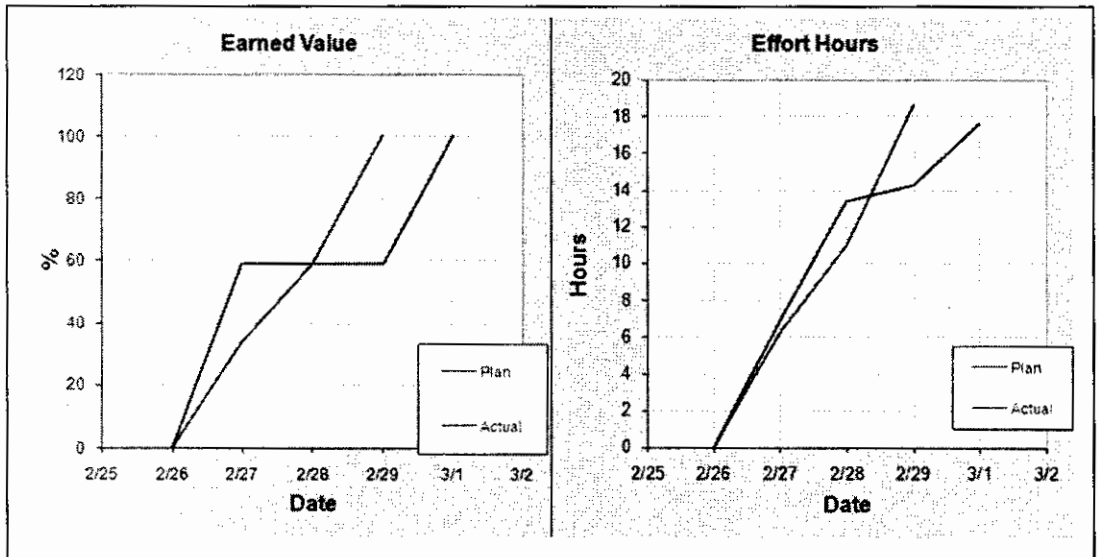


Fig.6.15. PSP1.1 Earned Value and Effort Hours

Fig.6.15 shows PSP1.1 results of earned value and effort hours in graphical format. This way PSP helps in schedule and task planning using earned value concepts. However, it is very short schedule but a practical step toward a project planning.

In **PSP2** actual to-date productivity was 65.75 lines of code/hour. CPI was 1.6. To-date percentage reused was 1.49%. Total defects/KLOC was 0.75. Actual percentage yield (Y) was 100%. 1326 lines of code was added (A) in the system. To-date value for new and changed lines of code (N) was 1326. To-date value for total lines of code (T) was 1346. To-date value for estimated object lines of code (E) was 1178. To-date percentage time spent in planning was 31.32%, in design it was 11.57%, in design review it was 11.57%, in code it was 30.16, in code review it was 0.41%, in compile it was 0.9%, in test it was 3.3%, and in post-mortem it was 10.77%. To-date percentage value for defect injected in design was 100%. To-date percentage value for defect removed in design review was 100%. Defect removal efficiency for design review was 0.43.

Fig.6.16 shows PSP2 schedule and task planning results in graphical format. Cumulative planned value and cumulative earned value are drawn along y-axis in percentage, where as time is plotted along x-axis in term of date of task completion.

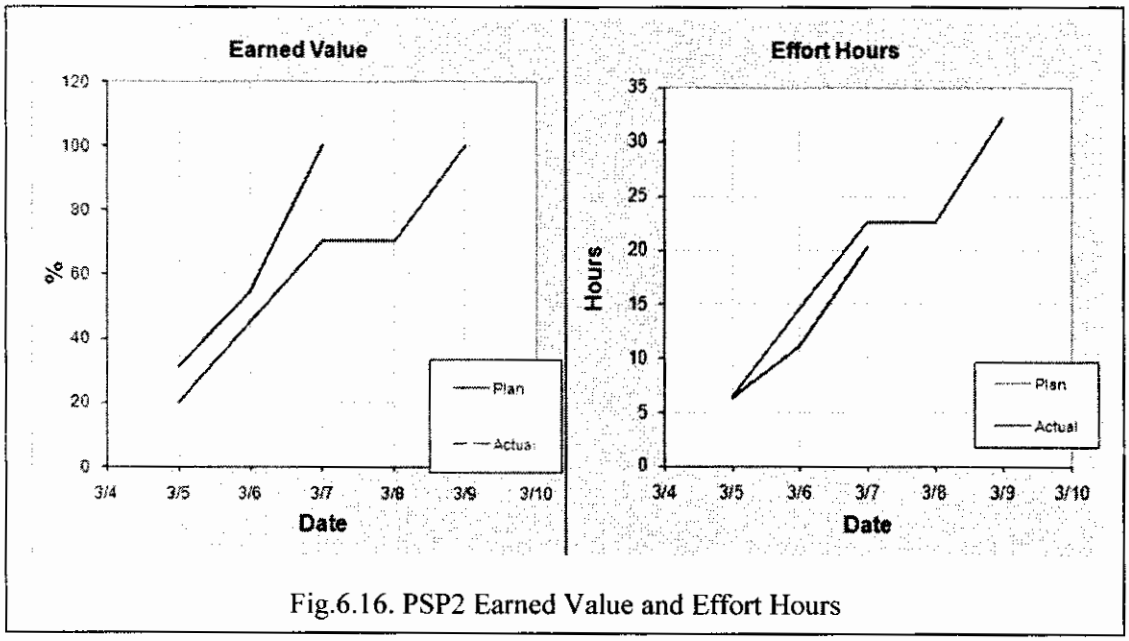


Fig.6.16. PSP2 Earned Value and Effort Hours

In **PSP2.1** actual to-date productivity was 34.7 lines of code/hour. CPI was 0.66. To-date percentage reused was 3.16%. To-date value for total defects/KLOC was 9.5. To-date percentage yield was 83.33%. To-date percentage appraisal COQ was 14.4%. To-date percentage failure COQ was 4.74%. To-date value for COQ A/F

ratio was 3.03. 622 lines of code was added (A) in the system. To-date value for total lines of code (T) was 632. To-date value for new and changed lines of code (N) was 622. To-date value for estimated object lines of code (E) was 548. To-date percentage time spent in planning was 40.2%, in design it was 4%, in design review it was 7.4%, in code it was 22.7%, in code review it was 7.3%, in compile it was 0.7%, in test it was 4.1%, and in post-mortem it was 13.6. To-date percentage defects injected in code was 100%. To-date percentage defects removed in code review was 83.33% and in compile its value was 16.67%. To-date defect removal efficiency for code review was 3.9 and for compile it was 8.57.

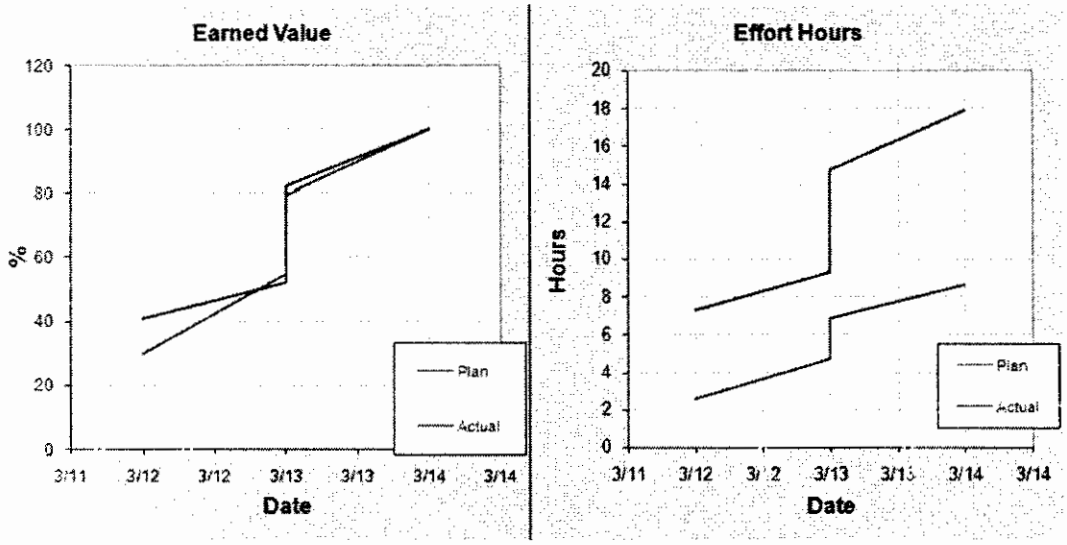


Fig.6.17. PSP2.1 Earned Value and Effort Hours

Fig.6.17 shows PSP2.1 schedule planning results in graphical format in this figure plan value and earned value have same data of completion. After going through various PSP exercises and availability of historical data regarding size, time and defect produces schedule planning and task planning results more precise and accurate.

Fig.6.18 shows correlation between program size "new and changed (N)" and development time (T). Correlation coefficient was calculated for PSP2.1 using PROBE. It was found that there is a strong correlation between program size and development time. However, results were not significant enough.

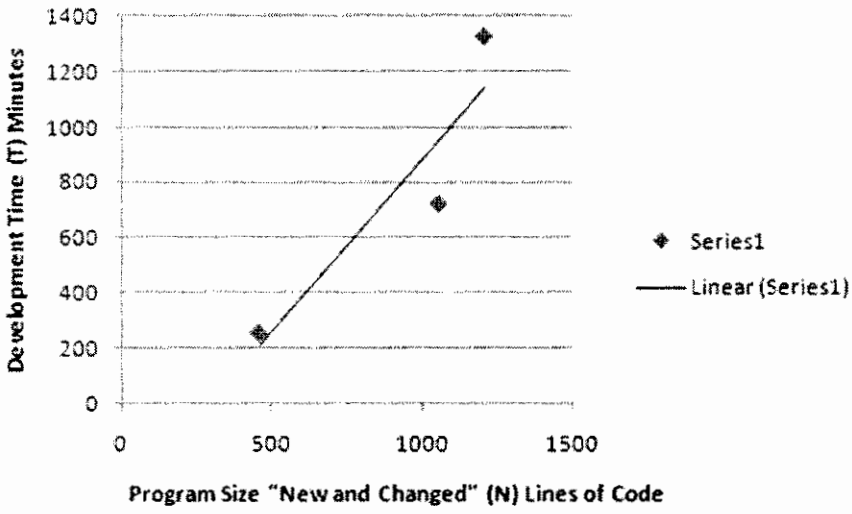


Fig.6.18. PSP2.1 Correlation between program size and development time

6.6.7 Case Study Part-2: Research Findings.

It was found that standard PSP involves many manual calculations, and took huge portion of development time. This also results in variation of precision and accuracy of development time. However, process automation can eliminate this inconsistency. It was also found that a separate process is required for requirement engineering, database development, and architecture design and evaluation. These suggestions were included in *"Process Improvement Proposal (PIP)"* during PSP execution.

6.7 Personal Software Process and Personal Integrated Process Comparison

Table 6.9 and table 6.10 provide a comparative analysis of two processes i.e. Personal Software Process (PSP) and Personal Integrated Process (PIP). Labor rates are high for PIP because of the technicality of the process. PIP involves advanced concepts and skills which normal software engineer do not possess. However, with this investment many risks were identified and mitigated. If these risks were not identified and mitigated project cost estimation would be incorrect or misleading. There is very slight difference in productivity for these processes. This shows that although PIP is difficult process and involves number of other process scripts but have very slight impact on productivity of software engineer.

TABLE 6.9
Process Comparison

	PSP	Personal Integrated Process
Time Measure	PSP0	Y
Defect Measure	PSP0	Y
Size Measure	PSP0.1	Y
Process Improvement Proposal	PSP0.1	Y
Coding Standard	PSP0.1	Y
% Reuse	PSP0.1	Y
% New Reuse	PSP0.1	Y
PROBE	PSP1	Y
Productivity	PSP1	Y
Schedule Planning	PSP1.1	Y
Task Planning	PSP1.1	Y
CPI	PSP1.1	Y
Design Review	PSP2	Y
Code Review	PSP2	Y
Total Defects/KLOC	PSP2	Y
% Yield	PSP2	Y
Program Size UPI 70%	PSP2	Y
Program Size LPI 70%	PSP2	Y
Defect Removal Efficiency	PSP2	Y
Defect Removal Leverage	PSP2	Y
% Appraisal COQ	PSP2.1	Y
% Failure COQ	PSP2.1	Y
COQ A/F Ratio	PSP2.1	Y
Functional Specification	PSP2.1	Y
Logic Specification	PSP2.1	Y
Operational Scenario	PSP2.1	Y
State Specification	PSP2.1	Y
Use Case Modeling	N	Y
System Sequence Diagram	N	Y
Operation Contract	N	Y
Domain Modeling	N	Y
Computer Hardware Selection Process	N	Y
Computer Network Design Process	N	Y
Database Design Process	N	Y
QAW	N	Y
ADD	N	Y
ARID	N	Y
ATAM	N	Y
CBAM	N	Y

TABLE 6.10
Process Comparison

	PSP	Personal Integrated Process
Development Language	SQL	SQL
Program Size (Lines of code)	3380	3827
Development Time (Minutes)	4690=78.167H	5587 Minutes=93.116 Hours
Productivity (Lines of code/Hour)	43.24	41.09
Labor Rates	US $ 50/1Hour	US $ 100/1Hour
Software Development Cost	US $ 3908.35	US $ 9311.6
Technical Risk Analysis	PSP	Personal Integrated Process
Availability Risk	N/A	33.34%
Performance Risk	N/A	11.11%
Modifiability Risk	N/A	22.22%
Reliability Risk	N/A	22.22%
Security Risk	N/A	11.11%

CHAPTER 7
INTRODUCTION, CONCLUSION, LIMITATIONS, FUTURE WORK, FINDINGS

7.1 INTRODUCTION

In this chapter research findings regarding literature review and case study will be discussed in brief. The limitations in case study regarding its application to other domain are discussed along with its application to technology. Conclusion drawn from different analysis is discussed in brief and finally the future work is give.

7.2 FINDINGS

Out of 98 research publications selected during literature survey on PSP only 30 publications were related with process modification or integration of PSP. So there is need for further work on process modification or integration. Proposed process if focus on domain and technology they can be used for precise estimation of cost, schedule, and identification of risks

The time spent in manual planning and post-mortem was 50% for PIP. This time can be reduced by using automated tool like FSQMS. Oracle 10g Express Edition was found to be helpful in reducing project defects during code review. This was because of the code generation facility provided by Oracle 10g Express Edition. It was also found that the deployment phase is necessary and will be integrated within the process.

Integration of architecture design and evaluation found to be very helpful in cost estimation. If however PSP was used the cost of project would be inaccurate or misleading. Project risk were identified and mitigated before detailed design. If these risks would not be identified stakeholder may had requirement conflicts. Project cost and schedule were better estimated.

7.3 LIMITATIONS

The case study has some limitation such as it was evaluated for developing software related to *"Business Information System (BIS)"*. So there is need to evaluate *"Personal Integrated Process (PIP)"* for embedded systems such as Microcontroller, FPGA, DSP based systems, and for industrial systems such as SCADA, PLC, DCS, HMI based systems. These systems may or may not require database development process. In case if database development process is not required process scripts regarding database development process may be excluded from the system development process.

7.4 FUTURE WORK

Business Information Systems (BIS) such as enterprise resource planners (ERPs) are consists of thousands of users spread around the globe. Project cost estimation for such large and complex system involves the estimation of all components such as software, hardware and network. Personal Integrated Process is designed to address the estimation of such large complex system. However, due to the time constraint project was implemented on relatively small project. In future PIP will be applied on such large complex system to estimate and compare actual values of cost, benefit, schedule, software size, defects, and risks. Return on Investment (ROI) was calculated for each architectural strategy. However, ROI need to be calculated for whole project and is the part of future work.

7.5 CONCLUSION

Case study was successfully designed, executed and analyses were performed to find the impact of architecture design and evaluation on project risk, cost and schedule. Software quality management system was successfully implemented using OracleJDeveloper Application Development Framework. System was developed using open source software Java. This system was designed to automate the Personal Integrated Process. Open source software was used to facilitate the transfer of knowledge and technology. The system can be used in the university to carry out further research in the field of software process engineering or it can be used in any commercial CMMI company for process improvement and automation.

REFERENCES

[1] W.S. Humphrey, "The Personal Process in Software Engineering," *Proceedings of the IEEE 1994 Third International Conference on the Software Process,* pp. 69-77, 1994, doi: 10.1109/SPCON.1994.344422.

[2] W.S. Humphrey, *A Discipline for Software Engineering.* ISBN 81-7808-435-X, Pearson Education Asia.

[3] M. Pomeroy, R. Cannon, T.A. Chick, J. Mullaney, W. Nichols, "The Personal Software Process (PSP) Body of Knowledge," version 2.0, Special Report CMU/SEI-2009-SR-018, Carnegie Mellon University, Software Engineering Institute, 2009.

[4] CMMI Product Team, "CMMI for Development," version 1.2, Technical Report CMU/SEI-2006-TR-008, Carnegie Mellon University, Software Engineering Institute, August 2006.

[5] W.S. Humphrey, "The Team Software Process (TSP)," Technical Report CMU/SEI-2000-TR-023, November 2000.

[6] P. Ferguson, W.S. Humphrey, S. Khajenoori, S. Macke, A. Matvya, "Results of Applying the Personal Software Process," *IEEE Computer,* vol. 30, no. 5, pp. 24-31, 1997. doi: 10.1109/2.589907.

[7] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-Based Analysis of Software Architecture," *IEEE Software,* vol. 13, no. 6, pp.47-55, November 1996, doi: 10.1109/52.542294.

[8] R.L. Nord, J. McHale, F. Bachmann, "Combining Architecture-Centric Engineering with the Team Software Process," Technical Report CMU/SEI-2010-TR-031, Carnegie Mellon University, Software Engineering Institute, December 2010.

[9] D.E. Perry, S.E. Sim, S.M. Easterbrook, "Case Studies for Software Engineers," *Proceedings of the 26th International Conference on Software Engineering (ICSE'04),* pp. 736-738, 2004, doi: 10.1109/ICSE.2004.1317512.

[10] D.E. Perry, S.E. Sim, S.M. Easterbrook, "Case Studies for Software Engineers," *Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop,* Tutorial Notes, pp. 96-159, April 2005, doi: 10.1109/SEW.2005.2.

[11] B. Kitchenham, L. Pickard, S.L. Pfleeger, "Case Studies for Method and Tool Evaluation," *IEEE Software,* vol. 12, no. 4, pp. 52-62, July 1995, doi: 10.1109/52.391832.

[12] P. Brereton, B. Kitchenham, D. Budgen, Z. Li, "Using a protocol template for case study planning," *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE),* University of Bari, Italy, pp. 26-27 June 2008.

[13] M. Host, P. Runeson, "Checklists for Software Engineering Case Study Research," *Proceedings of the IEEE First International Symposium on Empirical Software Engineering and Measurement,* pp. 479-481, 2007, doi: 10.1109/ESEM.2007.46.

[14] L. Dobrica, and E. Niemela, "A Survey on Software Architecture Analysis Methods," *IEEE Transactions on Software Engineering,* vol. 28, no. 7, pp.638-653, 2002, doi:10.1109/TSE.2002.1019479.

[15] P. Clements, R. Kazman, M. Klein, "Working Session: Software Architecture Competence," *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'07),* 2007, doi: 10.1109/WICSA.2007.50.

[16] I. Syu, A. Salimi, M. Towbidnejad, T. Hilburn, "A Web-Based System for Automating a Disciplined Personal Software Process (PSP)," *Proceedings of the Tenth IEEE Conference on Software Engineering Education & Training,* pp.86-96, 1997, doi: 10.1109/SEDC.1997.592443.

[17] P.M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, W.E.J. Doane, "Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined," *Proceedings of the 25th International Conference on Software Engineering (ICSE'03),* pp. 641-646, 2003, doi: 10.1109/ICSE.2003.1201249.

[18] A. Sillitti, A. Janes, G. Succi, T. Vernazza, "Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data," *Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03),* pp. 336-342, 2003, doi: 10.1109/EURMIC.2003.1231611.

[19] R. Sison, D. Diaz, E. Lam, D. Navarro, J. Navarro, "Personal Software Process (PSP) Assistant," *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05),* 2005, doi: 10.1109/APSEC.2005.87.

[20] O. Akinwale, S. Dascalu, M. Karam, "DuoTracker: Tool Support for Software Defect Data Collection and Analysis," *Proceedings of the International Conference on Software Engineering Advances (ICSEA'06),* 2006, doi: 10.1109/ICSEA.2006.261278.

[21] H. Hassan, M.H.N.M. Nasir, S.S.M. Fauzi, "Incorporating Software Agents in Automated Personal Software Process (PSP) Tools," *Proceedings of the IEEE 9th International Symposium on Communications and Information Technology,* pp. 976-981, 2009, doi: 10.1109/ISCIT.2009.5340991.

[22] D. Rosca, C. Li, K. Moore, M. Stephan, S. Weiner, "PSP-EAT- Enhancing a Personal Software Process Course," *Proceedings of the 31st Annual ASEE/IEEE*

Frontiers in Education Conference, pp. T2D-18, vol. 1, October 10-13, 2001, doi: 10.1109/FIE.2001.963883.

[23] I. Etxaniz, "Software Project Improvement through Personal Software Process in a R&D Center," *Proceedings of the IEEE EUROCON International Conference on "Computer as a Tool",* pp. 413-418, September 9-12, 2007, doi: 10.1109/EURCON.2007.4400502.

[24] A. Ibrahim, H. Choi, "Activity time collection and analysis through temporal reasoning," *Proceedings of the IEEE 11th International Conference on Advanced Communication Technology (ICACT'9),* vol. 1, pp. 579-584.

[25] K.A Gary, T.E. Lindquist, "Cooperating Process Components," *Proceedings of the IEEE Twenty Third Annual International Conference on Computer Software and Applications(COMPSAC'99),* pp. 218-223, doi: 10.1109/CMPSAC.1999.812704.

[26] Z. Pan, H. Park, J. Baik, H. Choi, "A Six Sigma Framework for Software Process Improvements and its Implementation," *Proceedings of the IEEE 14th Asia-Pacific Software Engineering Conference (APSEC'07),* pp. 446-453, 2007, doi: 10.1109/ASPEC.2007.43.

[27] Y. Park, H. Park, H. Choi, J. Baik, "A Study on the Application of Six Sigma Tools to PSP/TSP for Process Improvement," *Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse,* pp. 174-179, 2006, doi: 10.1109/ICIS-COMSAR.2006.13.

[28] J.A. Kim, J.Y. Taek, S.M. Hwang, "Study of Agent Based Process Management Environment – Mercury-," *Proceedings of the IEEE Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06),* pp. 132-136, 2006, doi: 10.1109/SERA.2006.65.

[29] J.A. Kim, "Process Management Technique Using 6 Sigma Tools and PSP," *International Journal of Software Engineering and its Applications,* vol. 1, no.1, pp. 1-18, July, 2007.

[30] J.A. Kim, S.Y. Choi, T.H. Kim, "Management Environment for Software Process Improvement," *Proceedings of the IEEE International Symposium on Computer Science and its Applications,* pp. 292-296, 2008, doi: 10.1109/CSA.2008.29.

[31] L.D. Sauer, T.E. Lindquist, J. Cairney, "Tracking Personal Processes in Group Projects," Proceedings of the IEEE Twenty Third Annual International Conference on Computer Software and Applications, pp. 364-369, 1999, doi: 10.1109/CMPSAC.1999.812740.

[32] Z. Yingying, Z. Xianzhong, J. Wang, "A Perspective of PSP Modeling Based on Control Theory," *Proceedings of the IEEE International Conference on Networking, Sensing and Control (ICNSC'10),* pp. 342-345, 2010, doi: 10.1109/ICNSC.2010.5461508.

[33] A. Babar, J. Potter, "Adapting the Personal Software Process (PSP) to Formal Methods," *Proceedings of the IEEE Australian Software Engineering Conference (ASWEC'05),* pp. 192-201, 2005, doi: 10.1109/ASWEC.2005.12.

[34] L. Williams, "Integrating Pair Programming into a Software Development Process," *Proceedings of the IEEE 14th Conference on Software Engineering Education and Training,* pp. 27-36, 2001, doi: 10.1109/CSEE.2001.913816.

[35] D. Escala, M. Morisio, "A Metric Suite for a Team PSP," *Proceedings of the IEEE Fifth International Software Metrics Symposium,* pp. 89-92, 1998, doi: 10.1109/METRIC.1998.731230.

[36] H. Suzumori, H. Kaiya, K. Kaijiri, "VDM over PSP: A Pilot Course for VDM Beginners to Confirm its Suitability for Their Development," *Proceedings of the IEEE 27th Annual International Computer Software and Applications Conference (COMPSAC'03),* pp. 327-334, 2003, doi: 10.1109/CMPSAC.2003.1245361.

[37] A.W. Brown, "Personal Software Engineering Project Management Process," *Proceedings of the IEEE International Conference on Software Engineering (ICSE'99),* pp. 669-670, 1999.

[38] C.A. Moore, "Lessons Learned from Teaching Reflective Software Engineering using the Leap Toolkit," *Proceedings of the IEEE International Conference on Software Engineering (ICSE 2000),* pp. 672-675, 2000, doi: 10.1109/ICSE.2000.870464.

[39] M.H.N.M, Nasir, A.M. Yusof, "Automating a Modified Personal Software Process," *Malaysian Journal of Computer Science,* vol. 18, no. 2, pp. 11-27, December 2005.

[40] H. Yu, X. Bao, S. Yang, "Research and Improvement of Team Software Process," *IEEE World Congress on Computer Science and Information Engineering (CSIE),* pp. 654-658, 2009, doi: 10.1109/CSIE.2009.911.

[41] P. B. Kruchten, "The 4 + 1 View Model of Architecture," *IEEE Software,* vol. 12, no. 6, pp. 42-50, November 1995, doi: 10.1109/52.469759.

[42] D.L. Parnas, P.C. Clements, "A Rational Design Process: How and Why to Fake it," *IEEE Transactions on Software Engineering,* vol. 12, no. 2, pp 251-257. February 1986.

[43] L. Sha, M.H Klein, J.B. Goodenough, "Rate Monotonic Analysis for Real-Time Systems," Technical Report CMU/SEI-91-TR-6, Software Engineering Institute, Carnegie Mellon University, March 1991.

[44] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh, "FORM: A Feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering,* vol. 5, no.1, pp, 143-168, 1998.

[45] P. Bengtsson, J. Bosch, "Scenario-Based Software Architecture Reengineering," *Proceedings of the IEEE Fifth International Conference on Software Reuse(ICSR'98)*, pp. 308-317, 1998, doi: 10.1109/ICSR.1998.685756.

[46] J. Bosch, P. Molin, "Software Architecture Design: Evaluation and Transformation," *Proceedings of the IEEE Conference and Workshop on Engineering of Computer-Based Systems (ECBS'99)*, pp. 4-10, March 1999, doi: 10.1109/ECBS.1999.755855.

[47] M. Klein, R. Kazman, "Attribute-Based Architectural Styles," Technical Report CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon University, October 1999.

[48] F. Bachmann, L. Bass, G. Chastek, P. Donohoe, F. Peruzzi, "The Architecture Based Design Method," Technical Report CMU/SEI-2000-TR-001, Software Engineering Institute, Carnegie Mellon University, January 2000.

[49] M. Matinlassi, E. Niemela, L. Dobrica, "Quality-driven architecture design and quality analysis method," VTT Publications 456, Espoo. ISSN 1455-0857. Technical Research Center of Finland, 2002.

[50] M.R. Barbacci, R. Ellison, A.J. Lattanze, J.A. Stafford, C.B. Weinstock, W.G. Wood, "Quality Attribute Workshops (QAWs)", Third Edition, Technical Report CMU/SEI-2003-TR-016, Software Engineering Institute, Carnegie Mellon University, August 2003.

[51] F. Bachmann, L. Bass, M. Klein, "Deriving Architectural Tactics: A Step Toward Methodical Architectural Design," Technical Report CMU/SEI-2003-TR-004, Software Engineering Institute, Carnegie Mellon University, March 2003.

[52] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, B. Wood, "Attribute-Driven Design (ADD)," version 2.0. Technical Report CMU/SEI-2006-TR-023, Software Engineering Institute, Carnegie Mellon University, November 2006.

[53] R. Kazman, L. Bass, M. Klein, "The essential components of software architecture design and analysis," *The Journal of Systems and Software,* vol. 79, no. 8, pp. 1207-1216, August 2006.

[54] C. Hofmeister, P. Kruchten, R.L. Nord, H. Obbink, A. Ran, P. America, "A general model of software architecture design derived from five industrial approaches," *The Journal of Systems and Software,* vol. 80, no. 1, pp. 106-126, January 2007.

[55] A. Tang, Y. Jin, J. Han, "A rationale-based architecture model for design traceability and reasoning," *The Journal of Systems and Software,* vol. 80, no. 6, pp. 918-934, June 2007.

[56] X. Cui, Y. Sun, S. Xiao, H. Mei, "Architecture Design for the Large-Scale Software-Intensive Systems: A Decision-Oriented Approach and the Experience," *Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems,* pp. 30-39, 2009, doi: 10.1109/ICECCS.2009.42.

[57] R. Kazman, L. Bass, G. Abowd, M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures", *Proceedings of the IEEE 16th International Conference on Software Engineering,* pp. 81-90, May 1994, doi: 10.1109/ICSE.1994.296768.

[58] C. Lung, S. Bot, K. Kalaichelvan, R. Kazman, "An Approach to Software Architecture Analysis for Evolution and Reusability," *Proceedings of 1997 Conference of the Center for Advanced Studies on Collaborative research (CASCON'97),* pp. 144-154, Oct. 1997.

[59] G. Molter, "Integrating SAAM in Domain-centric and Reuse-based Development Processes," *Second Nordic Workshop on Software Architecture,* 1999.

[60] P.C. Clements, "Active Reviews for Intermediate Designs," Technical Report CMU/SEI-2000-TN-009, Software Engineering Institute, Carnegie Mellon University, 2000.

[61] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and S. J. Carriere, "The Architecture Tradeoff Analysis Method," *Proceedings of the IEEE 4th International Conference on Engineering of Complex Computer Systems,* pp. 68-78, 1998, doi: 10.1109/ICECCS.1998.706657.

[62] R. Kazman, J. Asundi, M. Klein, "Quantifying the Costs and Benefits of Architectural Decisions," *IEEE Proceedings of the IEEE 23rd International Conference on Software Engineering (ICSE'23),* pp. 297-306, May 2001, doi:10.1109/ICSE.2001.919103.

[63] H. Choi, K. Yeom, "An Approach to Software Architecture Evaluation with the 4+1 View Model of Architecture," *Proceedings of the IEEE Ninth Asia-Pacific Software Engineering Conference (APSEC'02),* pp. 286-293, 2002, doi: 10.1109/APSEC.2002.1182998.

[64] W.G. SARA, "Software Architecture Review and Assessment (SARA) Report," version 1.0, 2002.

[65] C. Stoermer, F. Bachmann, C. Verhoef, "SACAM: The Software Architecture Comparison Analysis Method," Technical Report CMU/SEI-2003-TR-006, Software Engineering Institute, Carnegie Mellon University, 2003.

[66] Z. Wang, K. Sherdil, and N. H. Madhavji, "ACCA: An Architecture-centric Concern Analysis Method," *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05),* pp. 99-108, 2005, doi: 10.1109/WICSA.2005.8.

[67] B. Florentz, M. Huhn, "Architecture Potential Analysis: A Closer Look inside Architecture Evaluation," *Journal of Software*, vol. 2, no. 4, October 2007.

[68] P. Bengtsson, "Architecture-Level Modifiability Analysis," Doctoral Dissertation, Series NO. 2002-2, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, ISBN 91-7295-007-2, 2002.

[69] L.G. Williams, C.U. Smith, "Performance Evaluation of Software Architectures," *Proceedings of the ACM 1st International Workshop on Software and Performance,* pp. 164-177, 1998, doi: 10.1145/287318.287353.

[70] A. Bertolino, R. Mirandola, "Towards Component-Based Software Performance Engineering," *Proceeding of CBSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction,* May 2003.

[71] L.G. Williams, C.U. Smith, "PASA: A Method for the Performance Assessment of Software Architectures," *Proceedings of the 3rd International Workshop on Software and Performance,* pp. 179-189, 2002, doi: 10.1145/584369.584397.

[72] "HP Quality Center Software", Data sheet 4AA0-9587ENW Rev. 3, Hewlett-Packard Development Company, February 2009.

[73] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,* Third Edition, ISBN: 0-13-148906-2, Addison Wesley Professional, October 20, 2004.

[74] "Unified Modeling Language (UML) Specification: Infrastructure", version 2.0, OMG Adopted Specification, ptc/03-09-15, Object Management Group, December 2003.

[75] R. M. Fred, A. H Jeffrey, *Modern Database Management* Fourth Edition, The Benjamin/Cummings Publishing Company, Inc.

[76] *"Fusion Developer's Guide for Oracle Application Development Framework,"* 11gRelease2 (11.1.2.0.0) E16182-01, Oracle, May 2011.

[77] C. Jones, "Activity-based software costing," *IEEE Computer*, vol. 29, no. 5, pp. 103-104, May 1996, doi: 10.1109/2.494092.

[78] "Object-Oriented Application Analysis and Design for Java Technology (UML)", OO-226, Student Guide, Revision B, Sun Microsystems, Inc., March 2000.

[79] M.A, Babar, L. Zhu, and R. Jeffery, "A Framework for Classifying and Comparing Software Architecture Evaluation Methods," *Proceedings of the Australian Software Engineering Conference (ASWEC'04),* pp.309-318, 2004.

[80] M.A. Babar, I. Gorton, "Comparison of Scenario-Based Software Architecture Evaluation Methods," *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04),* pp.600-607, 2004, doi: 10.1109/APSEC.2004.38.

[81] R. Kazman, L. Bass, M. Klein, T. Lattanze, L. Northrop, "A Basis for Analyzing Software Architecture Analysis Methods," *Software Quality Journal,* vol. 13, no.4, pp. 329-355, December 2005.

[82] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice,* Second Edition, ISBN: 0-321-15495-9, Addison Wesley, April 11, 2003.

[83] M.R. Barbacci, R. Ellison, A.J. Lattanze, J.A. Stafford, C.B. Weinstock, W.G. Wood, "Quality Attribute Workshops", Second Edition, Technical Report CMU/SEI-2002-TR-019, Software Engineering Institute, Carnegie Mellon University, June 2002.

RESEARCH PUBLICATIONS

The following research publications were produced during *"Master of Science in Software Engineering"* thesis research project.

[1] H. Waqar, A. Shahbaz, "A Literature Review & Recommendations on Personal Software Process Tools," International Journal of Reviews in Computing, vol. 9, no. 4, pp. 26-33, April 10, 2012.

[2] H. Waqar, A. Shahbaz, "A Literature Review and Recommendations on Software Architecture Evaluation," International Journal of Reviews in Computing, vol. 10, no. 4, pp. 28-35, July 31, 2012.

[3] H. Waqar, A. Shahbaz, "Software Quality Management System," Journal of Emerging Trends in Computing and Information Sciences, vol. 3, no. 8, pp. 1205-1212, August 2012.