

OPEN SOFTWARE ARCHITECTURE EVALUATION SYSTEM

By

Chintan Barad

Bachelor of Computer Engineering, Gujarat Technological University, India (2008-2012)

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Computer Science

in the Graduate Academic Unit of Computer Science

Supervisor: Weichang Du, PhD, Computer Science

Examining Board: Virendra Bhavsar, PhD, Computer Science
Yevgen Biletskiy, PhD, Electrical and Computer Engineering
Dhirendra Shukla, PhD, Electrical and Computer Engineering

**This thesis is accepted by the
Dean of Graduate Studies**

THE UNIVERSITY OF NEW BRUNSWICK

April, 2015

© Chintan Barad, 2015

ABSTRACT

Software architecture evaluation is a process of identifying potential issues with the designed architecture for a software system. It is performed prior to the software system's construction phase, to determine the system's architectural feasibility and to evaluate the system's ability to meet the given quality requirements and business goals. In this thesis, we propose an open and distributed software architecture evaluation system to support standard and user-defined scenario-based software architecture evaluation methods. The system supports three types of software architecture evaluations: conventional scenario-based evaluation, open evaluation, and crowdsourcing based evaluation. The system is composed of six components: evaluation creation management, evaluation methodology management, team organization management, evaluator management, evaluation execution management, and post evaluation management. Those components can be configured for different evaluation types. A prototype of the proposed system has been implemented. A case study is also described in the thesis for architecture evaluation of a real-world software system using the proposed software architecture evaluation system.

ACKNOWLEDGEMENTS

I am glad to thank those who made this thesis possible. Therefore I would like to especially thank my family, who supported me while writing this thesis. In particular I would like to thank my advisor Dr. Weichang Du, from the University of New Brunswick. His strong motivation and guidance helped me to finish my thesis. I am obliged and feel honored to accomplish my research under him. Without their help, writing this thesis would have not been possible.

Table of Contents

ABSTRACT.....	ii
ACKNOWLEDGEMENTS.....	iii
Table of Contents.....	iv
List of Figures.....	viii
List of Abbreviations	x
Chapter 1 Introduction	1
1.1 Importance of Software Architecture Evaluation	2
1.2 Thesis Overview	3
1.3 Thesis Structure	6
Chapter 2 Background and Related Work	9
2.1 General Process of Software Architecture Evaluation.....	9
2.2 Scenario-Based Software Architecture Evaluation Methods.....	13
2.2.1 Software architecture analysis method (SAAM)	14
2.2.2 Architectural trade-off analysis method (ATAM)	15
2.3 Distributed Software Architecture Evaluation.....	18
2.3.1 ACE (ATAM collaborative environment)	19
Chapter 3 Concepts and Requirements	21
3.1 Actors of the Software Architecture Evaluation	21
3.2 Open Software Architecture Evaluation	22

3.2.1 Openness based on participation.....	23
3.2.2 Openness for large scale evaluation.....	25
3.2.3 Openness based on output availability.....	27
3.2.4 Additional explanation.....	28
3.3 Distributed software architecture evaluation	29
3.3.1 Open and closed evaluation in distributed environment.....	30
3.3.2 Distributed evaluation models	31
3.3.3 Scheduling in distributed evaluation.....	31
3.4 Requirements for Online Software Architecture Evaluation System	35
3.4.1 Requirements to support open evaluation.....	35
3.4.2 Requirements to support distributed evaluation	39
3.5 Summary	44
Chapter 4 System Design.....	45
4.1 System Architecture.....	45
4.2 Components of the System	48
4.2.1 Evaluation creation management.....	49
4.2.2 Evaluation methodology management.....	52
4.2.3 Team organization management	53
4.2.4 Evaluator management.....	55

4.2.5 Evaluation execution management	59
4.2.6 Post evaluation management.....	62
4.3 Different Types of Software Architecture Evaluation Realizations	64
4.3.1 Conventional software architecture evaluation.....	64
4.3.2 Open team software architecture evaluation.....	67
4.3.3 Software architecture evaluation by crowdsourcing.....	69
4.4 Summary	71
Chapter 5 Custom Evaluation Method Creation	72
5.1 Common Elements of Software Architecture Evaluation Methodologies.....	73
5.2 Common Tasks of Software Architecture Evaluation Methodologies	75
5.3 XML based Specification Language for Software Architecture Evaluation Methodologies.....	78
5.3.1 Input element	81
5.3.2 Roles element.....	82
5.3.3 Process element.....	84
5.3.4 Outcome element	87
5.4 Implementation Strategy of XML Specification.....	88
5.5 Summary	92

Chapter 6 Implementation.....	93
6.1 Technical Overview	93
6.1.1 Details on technologies used.....	94
6.1.2 Database insights	95
6.2 Implementation Details of Architectural Components of the System	99
6.3 Summary	111
Chapter 7 Case Study.....	112
7.1 Evaluation Organization	112
7.2 Evaluation Process	118
7.3 Post Evaluation Activities.....	130
Chapter 8 Conclusions	132
8.1 Summary	132
8.2 Evaluation	133
8.3 Contribution	136
8.4 Future Work	138
Bibliography	140
Appendix A Evaluation Methodology Specification Schema and ATAM Specification	143
Appendix B Database Design	150
Curriculum Vitae	

List of Figures

Figure 4.1 Architectural Diagram of the System	45
Figure 4.2 Evaluation Creation Management and Evaluation Methodology Management	49
Figure 4.3 Team Organization Management Component.....	54
Figure 4.4 Evaluator Management Component	56
Figure 4.5 Evaluation Execution Management Component	59
Figure 4.6 Post Evaluation Management Component	62
Figure 5.1 Sample XML file for evaluation methodology specification	80
Figure 5.2 Evaluation Methodology Management	89
Figure 6.1 Class Diagram of Evaluation Creation Management	100
Figure 6.2 Sequence Diagram of Evaluation Creation	101
Figure 6.3 Sequence Diagram of Custom Evaluation Methodology Creation	102
Figure 6.4 Sequence Diagram of Evaluation Methodology Execution	103
Figure 6.5 Class Diagram of Evaluation Methodology Management	104
Figure 6.6 Class Diagram of Evaluator Management and Team Organization Management.....	105
Figure 6.7 Sequence Diagram of User Sign in and Registration	106
Figure 6.8 Sequence Diagram of Evaluation Team Formation	107
Figure 6.9 Class Diagram of Evaluation Execution Management.....	108
Figure 6.10 Sequence Diagram of Evaluation Execution Process.....	109
Figure 6.11 Class Diagram of Post Evaluation Management	110
Figure 6.12 Sequence Diagram of Post Evaluation Activities.....	111

Figure 7.1 Signup on the System	113
Figure 7.2 Create an Evaluation.....	115
Figure 7.3 Homepage.....	119
Figure 7.4 Create a New Presentation.....	120
Figure 7.5 Layered view of Online Transaction Manager (OLTM)	122
Figure 7.6 Architectural Approach Creation	124
Figure 7.7 Quality Attribute Utility Tree	125
Figure 7.8 List of Architectural Approaches	126
Figure 7.9 Voting and Prioritizing Scenarios	127
Figure 7.10 View Evaluation Outcomes	129
Figure 7.11 Participant Feedback Form.....	131

List of Abbreviations

ATAM:	Architectural Trade-off Analysis Method
SAAM:	Software Architecture Evaluation Method
CBAM:	Cost Benefit Analysis Method
ALMA:	Architecture Level Modifiability Analysis
FAAM:	Family Architecture Analysis Method
XML:	Extensible Markup Language
API:	Application Programming Interface
OAuth:	Open standard for Authorization
WWW:	World Wide Web
UI:	User Interface
SQL:	Structured Query Language
DAO:	Database Access Object

Chapter 1

Introduction

The term software architecture instinctively denotes the high level structures of the software system. It can be defined as the set of structures needed to understand the software system, defining the software elements, the relations between them, and the properties of both elements and relations. The architecture of a software system can be considered as an analogy to the architecture of a building [1].

The person, who designs the software architecture, is known as Software Architect. Similar to the building with more than one architect, a large software system can have a group of architects. Experienced architects know that they may make some mistakes while designing the entire system or some parts of it. And, the earlier that they can detect a problem with their architecture, the better the project will be because, the longer a fault goes undetected, the costlier it will be to correct. **Software architectural evaluation** is a process of identifying potential issues with the proposed architecture, prior to the implementation phase, to determine its architectural feasibility and to evaluate its ability to meet its quality requirements.

1.1 Importance of Software Architecture Evaluation

In recent years, many software organizations have introduced software architecture evaluation as a critical and crucial component of the software-development life cycle [2]. They have also started including it as a phase in a software-development life cycle. Here one thing should be noted that evaluation of the software architecture is not another layer of process with the potential to slow the project down. The evaluation of software architecture is important for risk mitigation. It is good for the organization, and it's good for anyone as an architect. It leads to successful design of the system and hence saves an organization from a possibility of a huge loss.

The rationalization behind the evaluation of software architecture in an organization is as described in [2]. The bottom line is that architecture evaluation produces better and effective architectures that ultimately results in the delivery of better software systems. Quite often, systems are released with performance issues, security risks, and availability problems; and those issues are resolved in future releases. The architectures need to be defined early in the project life cycle, but the resulting defects and pitfalls can be discovered much later. We cannot migrate the process of designing the architecture to the other phase, because architecture is a blueprint on which the entire software is built. So as a result all the defects and pitfalls were exposed when the project was affected most negatively by change, when downstream artifacts were too costly to restore. Another most significant benefit of evaluation is to reassure stakeholders that the candidate architecture is capable of supporting the current and future business objectives and specifically it can meet its functional and non-functional requirements. Moreover, the

quality attributes of a system - such as performance, availability, extensibility, and security - are a direct result of its architecture and therefore, quality cannot be altered easily to the system late in the software development life cycle. In a nutshell, an evaluation of the architecture, while it is still an aspirant specification, can reduce project risk greatly and can benefit the entire organization in many ways.

Along with that, software architecture's evaluation can improve the architecture indirectly as well. Firstly, the evaluation process necessitates the unambiguous delivery of the system's predefined quality requirements. If the requirements are too obscure to evaluate architecture against, they must be elaborated upon in early phases in order to prevent loss of direction. Poorly specified requirements result in aimless architectures. Evaluation also forces architects to document the architecture clearly, so that it can be reviewed by other peers. Furthermore, as architects participate in regular evaluations of their work (architecture), they learn to anticipate the questions that will be asked by evaluation team members and the typical criteria against which their work will be measured and evaluated. With the passage of time, this process promotes stronger architectural skills in architects.

1.2 Thesis Overview

The architecture of the software needs to be evaluated to identify potential issues before actually constructing the system to determine its architectural feasibility. Generally this architecture is evaluated against various aspects of the respective software system such as business drivers, quality attributes, requirements from customers etc. The main purpose

of this research is to propose a software architecture evaluation system where anyone can create, participate and contribute in one or more evaluations at any time, following any of the evaluation methodologies in open and distributed environment. Moreover, this system allows the registered user to create a customized evaluation methodology or to modify the existing scenario-based evaluation methodologies.

Recent proposals of software architecture evaluations in distributed environment show the improved quality of scenarios in distributed meetings and barriers in current groupware system [10],[12]. Moreover, when this thesis is written, software industry faces following problems with the current scenario-based evaluation methodologies and distributed collaborative environment. Firstly, available distributed collaborative environment doesn't follow any of the scenario-based models. That implies we do not have a distributive environment to support scenario based evaluation methodologies. Hence, the advantages of scenario-based evaluation methods lead current software industry to face-to-face evaluation. Secondly, if the organization defines a new evaluation methodology or modifies some of the current ones, may face difficulties in creating and strictly maintaining the new methodology throughout the organization. Moreover, we have been concerned with organizations that have necessary human resource with them in order to evaluate the architecture. Consider a case where an organization needs to contract experts from other organization or outsource to freelancers who may not be collocated at the same place. Next, a group of entrepreneurs works on a software and in order to implement it, they hire some architects and developers. What if they want to evaluate the architecture? Even though experts across the countries will be ready to

contribute, with the current distributive environment they won't be able to follow or create the evaluation methodology. Moreover, they will have to find appropriate experts in order to evaluate the software architecture, as we don't have a smart evaluation system that suggests suitable participants for the evaluation.

The thesis proposes an open and distributed software architecture evaluation system as a solution to aforementioned problems. The system consists of six components: evaluation creation management, evaluation methodology management, team organization management, evaluator management, evaluation execution management, and post evaluation management. Each of these components is responsible for management of different evaluation phases. Firstly, evaluation creation is responsible for different activities of evaluation organization. The information provided to this component is transferred to the evaluation execution component, to support the actual evaluation process based on the selected evaluation method. The evaluator management component is responsible for all the activities of all the registered users of the system. This component provides all the necessary information about each participant to the team organization management component, which is responsible for managing a number of evaluation teams formed on the system to evaluate respective software systems. Lastly, the evaluation methodology creation component is responsible for handling creation and possession activities of standard and user-defined scenario-based software architecture evaluation methods. Depending on the type of software architecture evaluation, these components form the best suitable evaluation system to realize the objective of the thesis.

The thesis proposes open (open call for) and crowdsourcing based software architecture evaluation. These are unique ways of team organization for software architecture evaluation which allow evaluation creators to bring as much appropriate expertise as possible in the evaluation of prospective system's architecture. The proposed system supports the architecture evaluation by conventional and customized evaluation methodologies. Hence, to let people define such customized evaluation methodologies, the thesis proposes rules and grammar for XML based specification markup and mechanism to execute such specifications. The proposed system also helps participants create an open collaborative network for software architecture evaluation to invite and involve the expertise that they want in the evaluation. In order to bring the most suitable participants for the evaluation, the system provides various facilities such as smart suggestion, efficient search and email invitations to the members.

1.3 Thesis Structure

The thesis is structured in following manner. It starts with the background information related the importance of software architecture evaluation and two very well-known scenario-based software architecture evaluation methodologies, provided in the second chapter. This chapter also briefly describes two tools related to the proposed system, a distributed evaluation model called Groupware system and ATAM Collaborative Environment (ACE).

The third chapter proposes different concepts for openness and distribution in terms of software architecture evaluation. In addition to that the requirements to realize the proposed concepts are described. This chapter provides the fundamentals for the proposed thesis, based on which the following chapters elaborate creation, design, implementation and case study of the open and distributed software architecture evaluation system.

The forth chapter discusses the architecture of the proposed evaluation containing six components. Moreover, this chapter also elaborates how these components are configured to support different types of open and distributed evaluations. One of the components supports a phase of evaluation methodology creation, separate from the actual evaluation process. So, the fifth chapter is dedicated to that phase called custom evaluation management. This chapter provides a unique way of evaluation methodology specification through XML markup and the basic grammar and other rules for it along with the detailed process on such evaluation methodologies retain ability, creation and usage in software architecture evaluation in open and distributed manner.

Having discussed architecture and detailed description of each component of the proposed evaluation system, the sixth chapter describes technical aspects of the system based on the prototypal implementation done for the research work. In addition to that, the sixth chapter discusses each component's internal structure and functionalities along with necessary UML diagrams. At last, the seventh chapter is about case study which briefly explains the evaluation process of the Nightingale system from start to end

through the ATAM evaluation methodology. This chapter provides enough screenshots and real time statistics to thoroughly explain the real time evaluation process through the proposed system in open and distributed environment.

Chapter 2

Background and Related Work

This chapter starts with some information about software architecture and its evaluation. Following to that, it provides background information and briefly describes and cites other research works related to this thesis work. Background information mainly contains brief description of various scenario-based software architecture evaluation methods. Following that, as related work, a distributed evaluation model called Groupware system [12] and ATAM Collaborative Environment (ACE) software architecture evaluation tool which supports one of the scenario-based evaluation methods in a collaborative web-based environment [13] are explained.

2.1 General Process of Software Architecture Evaluation

Software architecture and its evaluation can be defined and described as follows. Basically, “it is a process by which conclusions can be drawn about the suitability of architecture.” Architectural decisions are evaluated to determine how they enable or restrict the ability of a system to meet its architecturally significant requirements. This question can be asked differently to architectural evaluation and every time it rationalizes its importance. The primary output of the evaluation process is a comprehensive report that describes the evaluation-and-review findings. This document needs only to be as formal as required by the project, but it should serve as a brief summary of the assessment that can be communicated to the project team, as well as the stakeholders.

The report should include the scope of the review, evaluation-and-review objectives, architecturally significant requirements list, findings and recommendations, and an action plan with that of backup plan as well.

The scope of software architecture's evaluation describes the boundaries of a specific instance of an evaluation. For example, the architecture of the entire system can be evaluated, or only part of the system. What exactly should be evaluated? Based on the defined objectives and scope, a list of the specific criteria can be created, against which the architecture will be measured. The list might include system-wide properties, significant functional requirements to deliver, and general attributes of quality architectures. The goal is to evaluate and assess how each item on the list is affected by the architectural decisions that are made. Reliability, security, availability, extensibility, manageability and portability are most of the quality attributes that can be considered in an architecture evaluation [2].

The objective of the selection process of the appropriate participants is to ensure that people with the right skills and relevance to the project are assigned to support the effort effectively, without creating a crowd that is too large to be efficient. Ideally, there should be active representation from three prospects i.e. evaluation team members, project decision makers and system architects, and stakeholders (developers, testers, performance engineers and quality engineers).

The evaluation team contains the actual evaluators who document all the notes and points found in the existing architecture. In large organizations, an evaluation team often contains practitioners who circulate through the team in between other projects. Staffing the evaluation team with practitioners from the target project should be avoided, if possible, to maintain the highest degree of objectivity. For very small projects, however, self-assessments and peer reviews are completely acceptable. It is critical that members of the evaluation team possess respect and credibility as architects, so that their conclusions will weigh with the project representatives and stakeholders. Stakeholders are the people who have specific architectural concerns and authorized interest in the resulting software system. Most of the architectural requirements were specified by these stakeholders, so that their participation in the evaluation is critical, especially in evaluating and prioritizing scenarios. System architects and component designers are the key project representatives and are responsible for communicating the architecture and presenting their motivations for design decisions. Other project representatives to include are project and program managers, system administrators and component vendors.

The most appropriate time for the evaluation is as early as possible in development life cycle, if only one evaluation is to be performed. The motive behind that is to identify any areas of concern as early as possible, while they are still relatively easy, cheap and convenient to adjust, mend and balance in order to restore the system. For agile software development, evaluation can take place during any iteration whenever architectural decisions have been made. Evaluations also can be conducted on legacy systems, to assess their ability to support future business objectives.

The detailed description on how this evaluation should be performed is as follows. Before the evaluation, architect(s) should gather inputs that describe the architecture and explain the rationale behind the architectural decisions that are made. Examples of typically selected inputs are the architecturally significant requirements, an architectural description or software architecture document, an architectural-decisions document, and an architectural proof of concept. The primary activity of the evaluation process is the assessment of the architecture. A proven technique involves the use of scenarios, which allow the quality attributes of the architecture to be evaluated in specific contexts. Walking through the steps of a scenario provides an architect with the opportunity to describe how architecture will respond to specific demands that are placed upon it. This technique is known as ATAM (Architectural Trade-off Analysis Method) and that is briefly described in following sections of this thesis.

The final step of the evaluation process is to document the findings, and communicate them to the project team and stakeholders. When architectural concerns or deficiencies are exposed, it is critical to provide recommendations for improvement that are actionable. The whole point of the investigative approach is to uncover issues that otherwise might have been overlooked.

After the review and the evaluation report is complete, architect typically is given an opportunity to respond to the findings and recommendations. The report then is forwarded to the stakeholders for use in planning the next steps for the project. Sometimes, an evaluation will identify the need for trade-offs. For example, if the

architecture cannot support a specific performance requirement, stakeholders must determine if the benefit of strengthening the architecture to achieve that requirement is worth the cost. Hence, including stakeholders later in the process is very important. Following an evaluation, the architectural decisions should be updated, requirements refined and prioritized, and the project adjusted as necessary. While each evaluation produces different results, the goal is always the same: to produce a better architecture and hence a better system.

2.2 Scenario-Based Software Architecture Evaluation Methods

In recent times, a number of new scenario-based software architecture evaluation methods have been developed. They usually restrict themselves to a particular class of systems. Many of these evaluation methods are adaptation of SAAM (Software Architecture Analysis Method) or ATAM (Architectural Trade-off Analysis Method), developed by Carnegie Mellon University – Software Engineering Institute. Later in this section, both of these models will be briefly described.

Currently available methods, for software architecture evaluation, are given below:

1. SAAM, Software Architecture Analysis Method [3], [5]
2. ATAM, Architecture Trade-off Analysis Method [3], [4]
3. CBAM, Cost Benefit Analysis Method [3], [6]
4. ALMA, Architecture Level Modifiability Analysis [7], [8]
5. FAAM, Family – Architecture Analysis Method [9]

2.2.1 Software architecture analysis method (SAAM)

SAAM has been the first and foremost widely published scenario-based software architecture analysis method. It is a methodology used to determine how specific quality attributes of the software system can be achieved and how much future changes in the architecture will affect those quality attributes [5]. Quality attributes that can be utilized by evaluation methodology include mainly modifiability, robustness, portability, extensibility and functional coverage. The method can also be used to assess quality aspects of software architectures such as performance or reliability.

The effectiveness and usability of SAAM depend on how it is used to analyze/evaluate the architecture(s) of a software system. Firstly, if a single architecture of the software system is evaluated, this method indicates both weak and strong points especially where the architecture fails to meet its quality requirements. Secondly, if two or more different nominee architectures of the same software system (providing the same functionality), are correlated with respect to their quality attributes, SAAM can produce an analogous ranking between the two architectures.

Software system's quality attributes, scenarios with the descriptions of interaction of a user with the specific system, the software system architecture's description available for all stakeholders, the reference artifact, which the quality scenarios are mapped onto, are the primary inputs to an SAAM evaluation. As output, a mapping of the brainstormed scenarios onto the architecture regarding future changes of the software system and the

estimation of costs and effort to be performed for required changes of the architecture are generated.

Participants of each categories i.e. external stakeholders, internal stakeholders and SAAM team members perform following six steps in order to evaluate the software architecture by SAAM [5]. First step is to develop scenarios in which activities that the system must support are brainstormed. Second step it to describe architecture(s) in which the candidate architecture(s) is clearly and understandably presented to all participants. In the third step aforementioned scenarios are classified and prioritized.

The fourth step - individually evaluate indirect scenarios - is performed in two parts. Firstly, the architect demonstrates a direct scenario and how that would be executed by the architecture and in case of an indirect scenario the architect describes how the architecture would need to be changed to accommodate the scenario. In step 5 - assess scenario interaction – affected components of the system on which more than one scenario interact are modified or divided into sub-components. Finally in step 6 – create an overall evaluation - gravity is assigned to each scenario in terms of its relative importance to the success of the system.

2.2.2 Architectural trade-off analysis method (ATAM)

In this subsection, the Architecture Trade-off Analysis Method (ATAM) is explained. ATAM is an accurate and complete way to evaluate software architecture [4]. The name

is ATAM because it exposes at what extent architecture satisfies quality goals and it provides perception how quality goals trade-off (cooperate) [3]. It is an architecture evaluation methodology developed by the Software Engineering Institute (SEI) and has been tested in real-life software development projects [3]. Based on the results drawn from these real-life ATAM projects, it seems that the ATAM is an effective method for guiding a software architecture evaluation phase. ATAM's main purpose is to guide ATAM session participants through an evaluation process via a series of nine steps. These steps can be bifurcated into two phases. Phase 1 consists of six steps and is more architecture-centric, involving architects, evaluators and other system developers. Phase 2 makes up Steps 7–9, which are more stakeholder-centric as it relies on stakeholder input.

The ATAM requires collaboration of three groups as participants. Firstly, the evaluation team which is external (sometimes internal) group consists of three to five people. Each member is assigned a one or more tasks to be performed during evaluation. Secondly, project decision makers containing project manager, system architect and a person assigning the evaluation in it. These people are in charge of speaking for the development and authorized change makers in the software system. Finally, stakeholders are a group including developers, testers, integrators, maintainers, performance engineers and users. The aforementioned participants of ATAM perform various activities in following steps.

Phase one, in step one, the evaluation team presents the ATAM to the participants. In step two, the evaluation team presents the system in a business context. During step three, the lead architect presents the system architecture. In fourth step, the architectural

approaches and styles are identified. In step five, the quality attribute utility tree is generated which shows all the quality attributes that need to be satisfied by the system along with their level of importance to the system. In step six, the architectural approaches identified in step four are analyzed which will result in a list of architectural approaches or styles, the questions associated with them and the architect's response to these questions.

Phase two, in step seven, the stakeholders and evaluation team brainstorm and prioritize scenarios by voting. During eighth step, the prioritized scenarios are mapped onto the architectural approaches that were previously acquired to determine if these scenarios fit the various approaches. This step is primarily a testing phase with expectation that little new information will be uncovered. At last in step nine, the result of the whole process is presented.

The output [3] of an ATAM is a brief presentation and/or a written report that includes the major findings of the evaluation. Output consists of the set of scenarios and their prioritization, the set of attribute-based questions, the utility tree, discovered risks and non-risk, the sensitivity points and trade-off points.

Main benefits [3] of ATAM includes clarified quality attributes requirements, improved architecture documentation, documented basis for architectural decisions, identified risks early in the life cycle, increased communication among stakeholders and improved system architecture. The ATAM aids in eliciting sets of quality requirements along

multiple dimensions, analyzing the effects of each requirement in isolation, and then understanding the interactions of these requirements.

2.3 Distributed Software Architecture Evaluation

As we just saw, most of the well-known software architecture evaluation methods are scenario-based and involve several Face-to-Face (F2F) meeting based activities [10]. That gets expensive most of the time. Moreover, as complex system development becomes an increasingly global activity, distributed collaboration is widely used to support software architectural evaluation with the less cost than that of scenario-based evaluation methods. In these types of evaluations, stakeholders are able to contribute to the evaluation whenever and wherever it matches with their schedule so it is possible to encourage them and evaluation team members for more constructive and collaborative participation in the architectural evaluation process. However, that depends on how distributed architectural evaluation is applied [10].

Research work in [10] proposes an effective and efficient way of involving physically distributed stakeholders in architecture evaluating processes without making them travel and relocate, while improving the overall process. They propose a framework (Groupware systems/tools) for distributed evaluation process along with the experimental findings showing the effectiveness of the proposed idea. A controlled experiment including large number of participants to assess the effectiveness of groupware has been presented [11]. The findings show that the quality of the scenarios generated in

distributive environment is better than the quality of scenarios generated in face-to-face environment. On the other hand, questionnaires to participants indicated that most of them (82%) preferred the face-to-face arrangement and 60% thought the distributed meetings were less efficient. Moreover, [12] describes collaborative environment called CAB (Collaborative Architecture Browser) to support distributed evaluation of complex system architecture. But, CAB has some barriers, which includes eliciting measurable contributions, cultural norms, critical mass and production paradox. Along with that, collaborative software has some design and implementation issues, they are - authentication, concurrency, bandwidth and persistence.

2.3.1 ACE (ATAM collaborative environment)

ACE [13] is a collaborative environment built for the ATAM. Its aim is to support an entire ATAM software architecture evaluation process, from the presentations to the brainstorming and voting steps until the final report generation step. In ACE, stakeholders, project managers/decision makers and evaluation team members can evaluate the given software system's architecture by following ATAM steps anytime and anywhere without having to be geographically collocated.

Main features available to all users are; request to join an evaluation, scenario brainstorming and all times available chat service. Evaluator specific features include creating an evaluation project, accepting a stakeholder, start/stop any step and view/edit utility tree. ACE supports the entire ATAM process where the current implemented

prototype concentrates on the brainstorming and voting process. When proposed, it was in an incomplete state and hence practical testing to completely verify these hypotheses was not yet possible. This approach has some inherent shortcomings or trade-offs, for instance, the impersonal nature of interacting with the machines instead of a human. As an outcome, this may affect getting stakeholders and evaluators encouraged to participate in an ATAM evaluation via ACE.

The differences between ACE and the proposed open and distributed software architecture evaluation system are as follows. ACE only supports software architecture evaluation by Architectural Trade-off Analysis Method (ATAM) in a web based environment while the proposed system supports software architecture evaluation by any of the existing scenario-based evaluation methodologies (ATAM, SAAM, CBAM etc.) along with the user defined (or customized) evaluation methodologies. ACE is just a web based version of ATAM evaluation methodology where all the members collaborate to perform a closed evaluation of the software system's architecture. On the other hand, proposed evaluation system supports open and crowdsourcing based software architecture evaluations along with the closed one. In addition to these features, proposed evaluation system allows people define a new evaluation methodology, and helps them to manage it and use it in the evaluation of their software systems architecture.

Chapter 3

Concepts and Requirements

This chapter outlines and elaborates concepts expected in an open and distributed software architecture evaluation system. These concepts are openness and distribution. They are explained in context of the software architecture evaluation system. Even though significance of openness and distribution differ in various domains, this section demonstrates that concepts form the heart of the work in the evaluation system. Along with that, this chapter also contains the explanation of the requirements of the system in order to substantiate aforementioned concepts.

3.1 Actors of the Software Architecture Evaluation

Firstly, two terms that can get confusing are software owner and evaluation creator. As mentioned earlier, as this proposed system I will be providing a highly scalable web-based open and distributed software architecture evaluation system in which anyone around the world can create and/or participate in multiple evaluations. So software owner can be a person or a group of people or an organization who owns the software. While evaluation creator is the person who creates and sets up the evaluation for the respective software system where other participants will collaborate and contribute based on their roles. These two can be combined or played by different people. If software owner(s) and evaluation creator(s) are same, they will set up the evaluation and various participants will play different roles in order to evaluate the architecture of given software system. On

the other side, if these are not the same, creator will create the evaluation and outsource the evaluation team and/or other participants based on the evaluation requirements. In either of the case, evaluation creator will be the final authority, responsible for monitoring the whole process and verifying the final output.

In addition to that, actors of the evaluation are generally divided into evaluators and evaluation creators. However, based on the evaluation methodology these actors are further segregated in various subgroups. For example, in architectural trade-off evaluation method, evaluators are divided into project decision managers, architects, project decision makers and stakeholders (developers, testers, performance evaluation developers). Evaluation creators mainly create evaluations, select participants, monitor evaluation processes and view evaluation results. On the other hand, functionalities of evaluators include participating in evaluations and following various duties according to one's expertise. However, the system should let the member to play different and multiple roles in the same evaluation.

3.2 Open Software Architecture Evaluation

Openness is generally characterized by an emphasis on transparency and unrestricted access to knowledge, information and expertise. And in case of a collaborative environment for software architecture evaluation, openness stands for collaborative and/or cooperative management and decision making instead of a central authority. In the context of the software architecture evaluation system, openness stands for exposure in

various contexts. Before actually getting into that variety of openness in this context, some definitions are given below. The detailed explanation will be covered in following sections of the thesis. The remaining part of this section briefly describes openness in various contexts and following to that, requirements of the system in order to justify those aspects.

3.2.1 Openness based on participation

Essentially, openness can be bifurcated into open call for participation and invite only participation. Open call for evaluation substantiates, participants around the world can request participation in the corresponding evaluation and necessary details about the given software system's architecture will have been made publicly available. These details will help participants to determine their eligibility for the evaluation and also will give them perception about the role they can play in corresponding evaluation.

This case comprises of two possibilities. Specifically, evaluation process in open call for evaluation type may or may not be open. In other words, even though participants are assembled by open call for the evaluation, the evaluation process itself will be closed and will not be visible to everyone or vice-versa. In that case, the basic information about the corresponding evaluation will be made publicly available in order to facilitate the open call but all other information such as output of the evaluation, details of evaluation methodology and participant information will not be publicly visible. Summing up this type, evaluation will be open and requesting participants will be added on first come first

serve basis. Creator will have final authority and one can accept or reject (ignore) the requests.

Next is invite only participation and that on the other hand is completely different than the previous type of open call. Here, evaluation details and/or output may be publicly available but only evaluation creator will have authority to add participants and grant them appropriate access rights for the corresponding evaluation. In this type, participants will not be able to request creator for the participation, rather creator will be responsible to find and add each participant in the respective evaluation to play an appropriate role.

The later type, that is inviting only participation, involves some more detailed conceptual analysis as follows. As mentioned earlier, only evaluation creator can invite appropriate participants and bring the expertise one wants in the corresponding evaluation, this can be facilitated in many different ways by the evaluation software system.

Firstly, creator should be able to look for appropriate participants by name, expertise and relevant experience. Here, if a creator knows someone who can actively participate in the evaluation, creator can invite one in the evaluation either by name if one is already a registered user of the system or through email invitation otherwise. Moreover, in this open system, each registered user will have detailed profile publicly open; containing domain expertise and that can be used by creator to determine the propriety of the participant with whom one can collaborate in a particular evaluation. Likewise the expertise, the open profile may contain the details of one's experience in the past

evaluations that can also be helpful to creator in deciding the appropriateness of participant. Here, not only creator should be able to examine the expertise and experience, but also one should be able to specify these as searching criteria and should get pertinent results.

3.2.2 Openness for large scale evaluation

Previous openness concepts are mostly effective in small teams where a fairly limited number of participants evaluate the architecture of a prospective software system by this evaluation system. In case of larger teams and large scale evaluation, where great number of stakeholders are entitled to participate in the evaluation of the forthcoming software system (large scale system), aforementioned ways of evaluation might get challenging.

Large scale evaluation comprises of two cases, either for larger open source software system or closed large scale software system. In both the cases, evaluation by multiple self-organized teams can be a better approach. Here self-organizing team implies a team where members have to manage themselves and they have to have self-governance. Self-organizing team chooses the best way to accomplish their work, rather than being directed by others outside the team [14].

In case of open source software, a creator should be able to provide necessary details for the prospective software system to be available to members of the evaluation system. As a result, number of self-organizing teams will be formed and evaluate the architecture

given necessary details and specifications by evaluation creator(s). The teams should be independent to follow any evaluation model and should be able to carry out evaluation one or more times with the software architecture evaluation system. Here, openness of architectural details of prospective system can also be modified. Owner can choose to make details open to registered users with specific skills and relevant experience rather than every end user. As an instance, authority of the avionics software system can make the details available to professionals with relevant experience and knowledge in avionics area. As a result, only those users will be able to be able to form self-organizing teams and to execute the evaluation.

On the other hand, evaluation creator will form various balanced self-organizing teams within the organization and propagate the fundamental information of the prospective software system, in closed large scale software's evaluation. Following to that, teams should be able to evaluate the architecture with the selected evaluation selected and multiple times. Here, balanced self-organizing teams mean, teams will have balanced participants with the necessary skills. In other words, each team will have equivalent evaluation team leader, project managers, architects and necessary stakeholders.

In both the cases i.e. open source and closed large scale evaluation, software owner (either a person or organization) and/or final authority should be able to monitor all the ongoing evaluation processes. In other words, each evaluation by self-organizing team should be open to its owner. One should be able to audit one or more steps of the evaluation at any moment of time and also review final outputs and other findings. One

should also be able to compare the final results of every evaluation and decide the best one of combination of some evaluations. And if not satisfied, various combinations of team members and evaluation models can be tried. And if asked for more details by evaluation teams during the evaluation, owner should be able to provide further details about forthcoming software system.

As mentioned above, the provided necessary information by the prospective software's authority may or may not contain the architecture of the system itself. If it comprises of the architecture, various self-organizing teams will perform the evaluation on the evaluation system. While on the other hand, if authority does not provide the architecture, various teams should be able to suggest the architecture designed on the basis of provided essential information of the planned software system first and then they should perform the evaluation of the proposed architecture. Furthermore, various teams can provide different architectures and the team can evaluate one or more of those proposed architectures. As a result, the evaluation will be treated as a rationalization of the corresponding architecture and authorities are likely to get the best architecture along with its evaluation details.

3.2.3 Openness based on output availability

Openness can be defined based on the output availability of the corresponding evaluation. As we know, each evaluation at last generates some output preferably in a report format. And that gives the necessary insight to its participants, and if they are not satisfied with

that they can re-evaluate the architecture either by different participants or by different evaluation method. Now, the output can be either made publicly available or private. In case of public availability, the registered user will be able to examine the output to determine the suitability of the evaluation model for the corresponding system requirements and its architecture and participants. That can help one in gaining necessary insights in selecting evaluation model and participants for corresponding evaluation. For the beginners, this can help them understand basic ground rules and protocols of the evaluation system.

3.2.4 Additional explanation

With the support of all preceding openness concepts, the evaluation system should support the private and closed software architecture evaluations, too. With this feature, a team of the organization will be able to use the system for the evaluation of the prospective software system while keeping it private and isolated from other users. While using the system for such an evaluation, rest of the features, for instance, selecting the evaluation model and various ways to invite participants, should be available to the evaluation creator and rest of the participants. All the necessary details about the software system and its architecture, output and member details must remain private. Evaluation leader should also be able to remove any related detail documents from the evaluation system's database after evaluation completion.

In all of the above cases, no two users can collaborate without connecting with each other through the system. So, with the passage of evaluations of different (or same) software systems, a wide social network of collaborators will be formed. That network should be open and every registered user should be facilitated and benefitted equally. That depicts the openness concept pertaining to the software evaluation system differently. Here the network will be ever-growing and creators with high degree centrality will have more choices for participants than those with the lower degree centrality. Many other social network analysis concepts can be applied here to analyze the network and to develop a better algorithms for forming the network.

From aforementioned various openness concepts in diverse situations, one hypothesis can be described as follows. In any of the above cases, different group of people may generate different outputs for the same evaluation probably following the same architectural evaluation model. As an instance, a group of participants in self-organizing team may produce a different set of trade-off points than that of the other group of participants evaluating the same system architecture through architectural trade-off analysis method (ATAM).

3.3 Distributed software architecture evaluation

Distributed literally means dividing tasks or responsibilities in geographically scattered area. Distributed evaluation of the software system means each member of evaluation should be able to contribute from any part of the world either synchronously or

asynchronously. Furthermore, distribution can also be defined as various evaluations of the same software system and any participant should be able to contribute in multiple evaluations at the same time. It can also be the case where large scale software system whose architecture is divided into number of blocks and further those blocks of the architecture are evaluated in distributed manner. In case of the small scale evaluation, various tasks for the evaluation can be divided into team members and they can execute the evaluation.

3.3.1 Open and closed evaluation in distributed environment

As discussed in previous section, evaluation can be either open or closed and both should be supported by the evaluation system. In both the cases, architecture of the prospective software system will be evaluated in distributed manner. In case of closed evaluation, participants will perform the duties from any part of the world either synchronously or asynchronously. Here all the details and the final evaluation output will be kept isolated and private from participants other than the evaluation team. While in case of open evaluation, everything will be made publicly available. Moreover, if self-organizing teams for the large scale evaluations will be executing the evaluation, various evaluations will take place in distributed environment and owner should be able to monitor this simultaneous process.

In any of the above cases, the registered user should be able to participate in multiple distributed evaluations to play a specific role. Here, role is to be specified by final

authority; however, access rights can be requested by participants. Moreover, registered user can play same and/or different roles in various evaluations and one should be able to divide one's efforts as per requirements in one or more evaluations.

3.3.2 Distributed evaluation models

Distribution can also be defined in terms of evaluation model. Evaluation model means the evaluation methodology to be followed in the corresponding evaluation on the prospective software system. Here, evaluation of the system should be distributed in different evaluation models. In other words, same and/or different team members may evaluate the same architecture by following different evaluation methods and can generate different output so that they can choose the best evaluation. The evaluation system should support various evaluations of the same software system by following any evaluation model in parallel.

3.3.3 Scheduling in distributed evaluation

One of the critical elements to make distributed evaluation successful is time. In other words, scheduling in collaborative evaluation results in an effective evaluation of prospective software system. Moreover, research demonstrates that network-based collaboration may provide opportunities for more equality in group work than actual face-to-face group work since in the latter approach group "decision-making" is often depend upon which participant has the loudest voice or who has the most confidence in

the target language [15]. Hence, the evaluation system should equal each participant's participation by abolishing the barriers of face-to-face evaluation and by giving them sufficient time to think and perform their duties. However, system should also allow leaders to make the evaluation duration organized at some extent to avoid the evaluation running forever.

Based on the time, distributed evaluation can be divided into two categories viz. synchronous and asynchronous. Synchronous distributed evaluation implies each participant must participate and contribute in each phase on pre-decided time and finish the duties in pre-specified duration. Synchronous evaluation can get more exacting if participants are asked to perform duties at the same time irrespective of their location. For example, in a strictly synchronous evaluation 'A' following ATAM (Architectural Trade-off Analysis Method) evaluation model, evaluation team members and project decision makers are asked to be available at the same time in order to perform the duties for first step.

Synchronization in a distributed software architecture evaluation can be attained in two ways. Firstly, time constraint in which each phase needs to be finished in a specific duration. This duration can be either decided by an evaluation leader or by mutual opinion. As an instance, if in the evaluation a group of participant decides to finish a phase in certain amount of time, an evaluation system should be able to facilitate them by all means in the scheduling.

The second way to achieve synchronization in the distributed evaluation is by milestone also known as point of workflow. Points of workflow can be defined as a series of markers placed along the evaluation process (irrespective of the phase boundaries) at intervals of parts of the whole evaluation process. The software architecture evaluation system should be able to aid this type of scheduling by letting participants cross the point of workflow only after all participants are done with their duties for that milestone. To demonstrate the case, consider the example of a self-organized team evaluating the architecture and following ATAM evaluation model in synchronous mode with respect to milestones. If they treat each phase as a point of workflow, after presentation phase participants will be able to perform analysis phase. So, until each group of participants (viz. evaluation leaders, project managers and architects) finish the presentations of evaluation plan, business drivers and architecture of the system respectively, they should not be able to start working on next phase (or point of workflow) which is analysis phase.

The other type of distributed evaluation is asynchronous distributed evaluation. In this type, no various steps of evaluation occur at the same time. In this type of evaluation, creator starts the evaluation if only one team is evaluating the prospective system's architecture. In case of self-organizing teams, software owner provides with all the necessary details about architecture of the system to be evaluated. In both the cases, every participant should be able to contribute as per one's convenience and no strict schedule should be followed. As a result, evaluation can last for longer period and so where strict deadlines are not crucial this type of evaluation can be a better choice. However, the

evaluation system should avail creators to trigger some reminders to the participants in order to avoid evaluation running forever.

The best approach to carry out an evaluation in a distributive environment would be the combination of asynchronous and synchronous software architecture evaluation. Here, each phase can be made synchronous and within each phase all the steps can be made asynchronous. As an instance, a team can allocate ten days in one phase and each participant can perform the duties as per one's convenience. In other words, participant from any part of the world should be able to contribute or to examine other's executions at his suitable situation in no longer than ten days. The participants should be able to customize the synchronization as per their requirements and to impose temporal deadline as strict as they want on the evaluation system.

In both the aforementioned cases, after one of the evaluation team members (preferably pre-decided scheduler) will mark the current step as finished, then only next step can be started. As an instance, a group of participants are evaluating architecture 'A' by following ATAM (Architectural Trade-off Analysis Method) evaluation model, once the scheduler marks the first step as finished, all the participants should then be able to perform the duties for the next step. However, in order to allow participants to pre-plan the afterward step, some functionalities can be made available to them for the prospective step.

3.4 Requirements for Online Software Architecture Evaluation System

This section comprises of requirements of the online software architecture evaluation system in order to justify aspects mentioned in previous sections (openness and distributed).

3.4.1 Requirements to support open evaluation

Openness in this context can be described as an emphasis on transparency and unrestricted access to various evaluations and associated data. Previous section describes openness in various contexts; as a result this section elaborates requirements for the proposed system to support them.

In the context of participation, the system has been divided into two parts, namely open call for evaluation and invite only participation. To support the open call to the participants for the evaluation, the system should allow all participants to search for the evaluation and request the creator to be a part of it. In this case, a creator will receive a request and can decide either to accept it or reject it. On the acceptance of the request, a creator will have to decide the role of the requesting participant and then he can play that role in the respective evaluation. With the association of registered users in one or more evaluations, a network of users will be generated. In that network, all the users will share the relation based on their participation in one or more evaluations. As a result, a user can team up with the same person(s) for future evaluation through the system.

Next, in order to invite the participants in invite only evaluation, a creator can be given various options. Firstly, he can simply write an email address and a role (viz. architect, stakeholder, evaluation team member) of the participant and the participant will get an email with a link. Through that link he can register on the system and can be a part of the evaluation to play a specified role. Secondly, if a participant is already a registered user of the evaluation system, a creator can simply search for him and invite him to be a part of an evaluation with the specified role. With the acceptance of the invitation, he can be a part of the evaluation. Next, LinkedIn API [6] can be used to find and suggest appropriate professionals from LinkedIn users. Finally, the proposed evaluation system should also be smart enough to suggest participants to the creator (a person or the owning organization) or allow creator to search based on various parameters. In order to do that, the system should store the data for each user in the form of profile and with natural language processing of those self-claimed credentials the system can populate the suggestions based on the corresponding evaluation and its requirements. In case of searching, the system should index various details of the user and allow creator to search by name, location, expertise or experience and many other criteria.

According to the discussion from previous section, large organizations encourage independent evaluation by multiple self-organizing teams. These teams can either be from anywhere in the world or within the organization. The proposed system should control the privacy for either of the cases. Not only the proposed systems should facilitate creators to create various self-controlled teams given number of willing participants but also should automate the team creation process. With the provided information about

users, the system needs to be capable of using that data to create automated balanced self-controlled teams. However, prospective software creator(s) can later alter the teams if unhappy with the team formation by the system. Within each of the teams, election should be supported for various roles by the system. User can ask the system to automate the access rights distribution within each team, too.

There may be the case where software authorities want self-organized teams to not only evaluate the architecture but also create the architecture based on the details provided. For that, the system needs to provide various collaboration tools in efficient manner to let team members create the architecture for the prospective software system. This generally happens in an open source project, where inventors propose the idea and various teams around the world will participate and create architecture.

For the large scale evaluation where large number of self-organizing teams is evaluating the architecture, software authorities should be able to control, compare and analyze various evaluations through the proposed system. The system needs to facilitate the creator to control all the evaluations with minimum complications. Moreover, software authorities should be able to compare the outputs of various evaluations and get insights from that to choose the best evaluation or combinations of several evaluations. For that, the system should let them generate various graphs and charts based on the outputs. The system should also be capable enough to integrate an analytics tool to get the insights from the large amount of data generated through various evaluations.

One of the openness concepts from the preceding section defines openness based on the output availability of the corresponding evaluation. Here, the system should let creator specify during, before or after the evaluation to keep output either private or public. If kept public, the registered user of the evaluation system should be able to get the insights about the output of the corresponding evaluation and in case of private, only the participants can view the output and ask the system to remove it forever. In case of large scale evaluation where number of self-organized teams will evaluate (or propose) the evaluation, details of the evaluation can be made available to users with specified skills and/or experience through the system.

In order to support smart suggestions for relevant participants to the creator, the proposed system needs to possess information pertaining to each user in order to process that. And for that, like any other social media channels, the proposed system can retain user's information (lists of personal interests, contact information, other personal information, such as employment status, relevant education and experience etc.) and later can process these information to provide relevant and smart participant suggestions to the evaluation creating authority. In order to ensure the provided information is legitimate, data can be accumulated from other well-known social media channels and the proposed system should be capable enough of integrating this using the OAuth [19] algorithm. Not only can this information be used by the system for smart suggestions, but also can be made publicly available to help evaluation creator determine the eligibility of the participant for the prospective evaluation. For that, system needs to show public profile (information of the user made publicly available) to other users. And preferably, the system should let

users to connect with each other before actually showing the profile. As a result, the open social network among users will be created where users can share much information and re-connect for the prospective evaluations. That network will be ever-growing, and the proposed evaluation system needs to be capable of handling such network efficiently.

3.4.2 Requirements to support distributed evaluation

To create web-based software evaluation system, that can support multiple evaluations with a number of participants in distributed environment either synchronously or asynchronously, the system needs to have following characteristics. The web-based system should be able to scale if it continues to be consistently available and functional as the number of users and their requests, and various evaluations grow, even to very high numbers [16]. The system should have better performance with a number of users and their evaluations. The proposed evaluation system should have multi-tenancy as one of the essential features. Multitenancy here implies that, the system should be designed in such a way that it provides every user a dedicated share of its data, configuration, processors and other resources [17]. Furthermore, as previously mentioned, the user can participate in a number of evaluations at the same time, playing same and/or different role. To facilitate this, the proposed system should maintain the access rights of different users properly with respect to the corresponding evaluation. Along with that, system should facilitate the software evaluation authority to assign those access rights to appropriate participants and that even can be altered by the authority only. However, the proposed system should allow users to request for specific rights. And, evaluation creator

should be able to get the insights about participants through this system in order to determine the appropriate rights for them. Consider an example, where a user ‘A’ is participating in two evaluations at the same time, but playing different roles in them. In evaluation 1, the user is an evaluation team member and in evaluation 2, the user is a project decision maker. So, authorities of both the evaluations should be able to provide appropriate rights to user ‘A’ for corresponding evaluations and ‘A’ should be able to play different roles with different access rights. Here, roles and the respective access rights are decided by the evaluation models.

In order to support multiple evaluation models, where a team can use one of the evaluation methods for the evaluation and a participant can be part of one or more evaluations following any of the selected evaluation methods at the same time, the proposed evaluation system should be capable of supporting and implementing multiple evaluation models. The proposed system should be able to contain multiple types of evaluation models while keeping scalability, performance and multitenancy unaltered. Here scalability, performance and multitenancy are referred to the evaluation system and the supported evaluation models, too. In other words, a supported evaluation model should be with all the aforementioned qualities in order to let participants collaborate in distributed and open environment.

As mentioned in the previous section, the proposed evaluation system should equal each participant’s participation and should abolish barriers of face-to-face evaluation. On the other hand, research findings in [11] show the preference of face-to-face evaluation over

distributed arrangement (with no evaluation method) by participants. So, the proposed system should facilitate evaluation in open and distributed environment without annihilating all the features of face-to-face evaluation.

Along with support for distributed and open evaluation, the proposed software architecture evaluation system should facilitate participants to contribute in both synchronous and asynchronous manner. To support synchronization, the system needs to provide time keeping facility to each user independently for each evaluation he has participated into. The timeline should be available to each participant respective to one's time-zone and should reflect schedule for each evaluation accordingly. As mentioned in previous section, synchronous distributed evaluation may get more exacting so in that case the evaluation system provides invariable performance and scalability to a number of participants committing to multiple evaluations at the same time.

Previous section describes two ways of cultivating distributed evaluation in synchronous manner namely by time constraint and milestone (workflow). In order to support phase completion on pre-decided time, the scheduler should be able to keep synchronization points in the corresponding evaluation. A pre-decided evaluation team member (e.g. scheduler) should be only allowed to mark current phase as completed and move the process to the next phase. The system needs to show the details of uninitiated phases without letting team members to perform any of the duties. Hence, through this evaluation system, participants can maintain the evaluation model during the process among all the participants. The system should be capable of letting evaluation creator

decide boundaries (time or workflow) or pass to other participants where they can decide synchronization points through some mechanism (such as voting).

While in case of asynchronous distributed evaluation, the system should preserve state and data for each participant and pertaining evaluation. And that data should be accessed with the same efficiency irrespective of the duration of the evaluation. In other words, no matter for how long the evaluation and its data have been untouched, it should be persisted and returned with the same ease when requested by the participant.

At last, the proposed system should support distributed environment in both (synchronous and asynchronous) manner combined. As mentioned in the previous section, the combination of synchronous and asynchronous manner is the best approach to carry out an evaluation in distributed environment. Hence, system is supposed to let participants customize the synchronization in the corresponding evaluation. In other words, the creator should be able to keep evaluation synchronous between any two phases but within each phase, various steps can be made asynchronous or vice-versa.

Since the proposed system allows the registered user to participate in one or more evaluations and as a creator to monitor all the evaluations by self-organizing teams, the proposed system should avail user to subscribe to receive updates of all the concerned evaluations. Participants should be able to subscribe to the respective evaluations for updates like their turn to contribute, other participants' activities and modification in resulting document through this system. They should also get notified with respect to

scheduling deadlines in distributed synchronous evaluations by the system. Here notification can be email, text message or internal notification on the system with necessary information regarding the update. As a result, each participant will be up-to-date with each participating evaluation. In case of large scale evaluation where numbers of self-organizing teams are evaluating the architecture, the system is required to facilitate the creator to subscribe to any team and for any update with the specific priority to stay up-to-date with all the evaluations with the least time investment.

In order to support distributed evaluation in a true sense, the proposed software architecture evaluation system requires providing branching facility in each evaluation. Here, participants will be able to create a small branch in a particular evaluation with a specific goal. Following that, that branch will be treated as a separate unit and they can treat that as an independent workflow. The system should let them invite other participants to that workflow, too. Newly added participants will have only access to that workflow with evaluation information limited to that branch. As a result, the output of that workflow will be used in the corresponding evaluation as input for other steps. The system should allow participants to create as many branches as they want in any phase (step) of the evaluation irrespective of the evaluation model. In other words, the evaluation system should allow participants to make the evaluation iterative and incremental.

3.5 Summary

In a nutshell, this chapter elaborates two concepts viz. openness and distributed in terms of software architecture evaluation. Open software architectural evaluation is described in different ways. Firstly, openness in terms of participation describes open call and invitation for participation. Following to openness for larger teams and large scale evaluation is discussed. Moreover, this chapter also describes how large scale software architecture evaluation can be different from the conventional software architecture evaluation. Next, openness with respect to output availability is discussed, containing visibility of evaluation outcomes to other registered users of the evaluation system.

The proposed software architecture evaluation system supports various aspects of distribution. Firstly, both open and closed types of evaluations are discussed in terms of distributed environment. Following to that, detailed description of distributed software architecture evaluation models is elaborated. Lastly, synchronous and asynchronous software architecture evaluations in distributed environment are discussed along with all the possible types of scheduling.

Based on both the concepts and their different cases, this chapter gives the explanation of the requirements of the system in order to substantiate all the cases of open and distributed software architecture evaluation.

Chapter 4

System Design

4.1 System Architecture

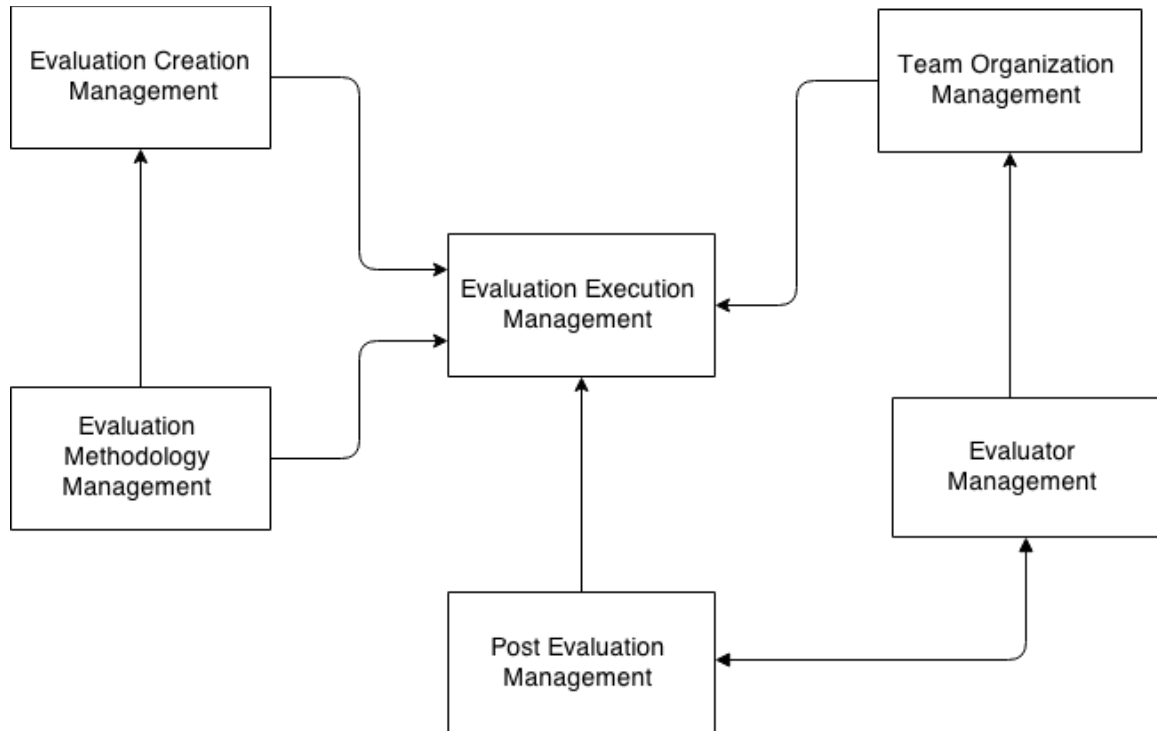


Figure 4.1 Architectural Diagram of the System

Figure 4.1 shows the architectural diagram of the proposed software architectural evaluation system. The architecture comprises of six components: evaluation creation management, evaluation methodology management, evaluation execution management, post evaluation management, team organization management and evaluator management.

The diagram also shows the relations between components. Here, direction of the relation shows a component depending on the output of other component(s).

The first component is evaluation creation management which is responsible for evaluation details management. All the necessary details for the respective evaluation are stored and processed in this component. As shown in the architecture diagram, this component intakes list of available evaluation methodologies from the component called evaluation methodology management. This component lets a user create customized evaluation models. It is responsible for all the necessary user interactions and enabling them to create desired evaluation model which fulfills their needs.

Next is the evaluator management component which is responsible for managing evaluators in all phases of an evaluation. From basic functionalities like login, signup to helping evaluation creators find appropriate participants, this component does it all. This component publishes its output to another component called team organization management component. The team organization management component is responsible for evaluation team creation and retention, access rights management and self-organized team management. Another component that is connected with the evaluator management component is post evaluation management. The post evaluation management handles activities after the completion of an evaluation such as, participant rating, outcomes comparison etc.

The central component of the system is the evaluation execution management component. This component is the most important in the whole software architecture of the proposed system. This component is responsible for all the functionalities, user interactions and scheduling of evaluation process irrespective of underlying evaluation models. As shown in the figure, this component intakes the output of three components namely, evaluation creation management component, team organization management component and evaluation methodology management component. All the components are briefly described in the following subsection.

Components shown in the architectural diagram are loosely coupled with each other. Each has its own user interaction and database. Depending on the evaluation specifications various components form the evaluation system and interact with each other to give the evaluation creator a satisfactory output. The following subsection elaborates how different components collaborate and form the evaluation system in different types of evaluations.

All architecture evaluations independent of selected evaluation methodology comprise of several fixed states. We can relate each step with each component. However, functionalities within each phase differ from evaluation to evaluation but there are also some commonalities between them. These phases are also carried out in a particular sequence and that sequence remains same most of the time. Common phases in a software architecture evaluation include creating evaluation, selecting appropriate participants, performing the evaluation activities, and post evaluation activities. The

evaluation creation component realizes the first phase. Next, the evaluator management and the team management components get involved in the second phase and lastly, the post evaluation management component comes in to play.

The only remaining component is the evaluation methodology management. That component itself is a separate phase which does not fall into the aforementioned sequence. The users that specifically interact with that component may not be the participants and/or evaluation creators. However as shown in the diagram, this component sends its output to the evaluation creation component and the evaluation execution management component.

The rest of this chapter explains each component in detail with all the functionalities. Each component is depicted with its subcomponents. Following to that, how all components can realize various types of open and distributed software architecture evaluations as well as closed and conventional software architecture evaluation is described.

4.2 Components of the System

This section elaborates each component in detail along with its diagram. The diagram contains the internal structure of the component, relations between parts and various use cases associated with them. This section also provides details about the user interaction

with each component and how various components are connected and depend on each other's outputs.

4.2.1 Evaluation creation management

This is the first and foremost component of the proposed software architecture evaluation system. This component is responsible for managing evaluation related details such as evaluation profile, selected evaluation method, openness of the evaluation and other scheduling details. Freelance evaluators, looking for appropriate evaluation to participate in, interact with this component.

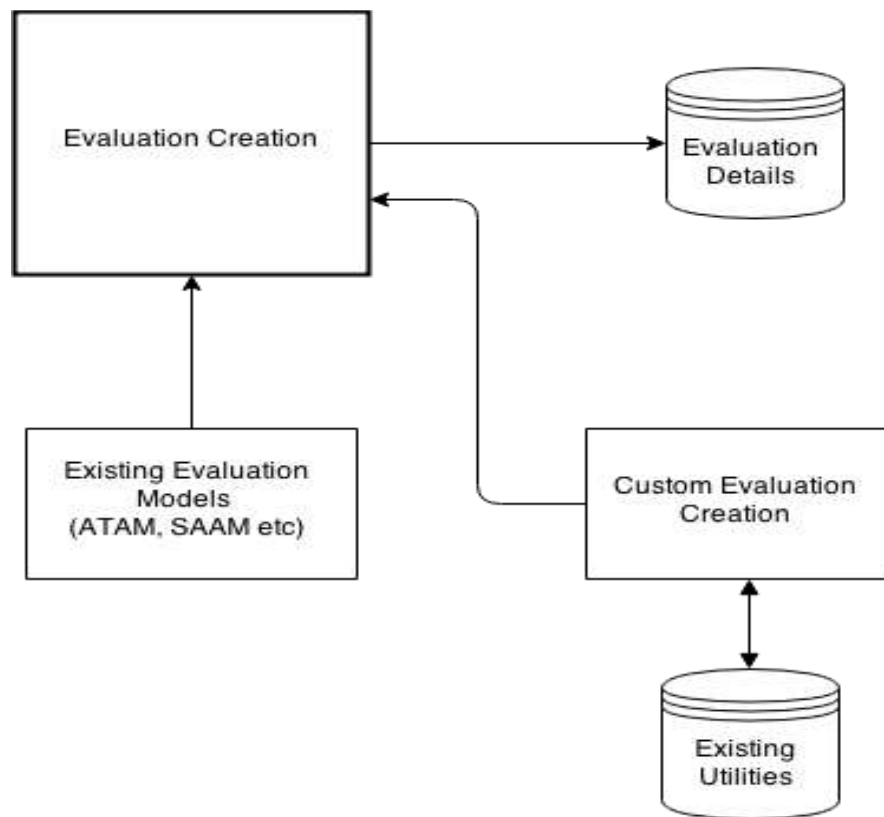


Figure 4.2 Evaluation Creation Management and Evaluation Methodology Management

Firstly, when an already registered user logs in, this component is where the user will interact. From here, the registered user will be able to create a new evaluation by filling out necessary details. As shown in the figure 4.2, all the necessary details will be stored in the database. Necessary details for the evaluation comprise of two parts namely evaluation profile and system details. Here, the evaluation profile contains various details such as selected evaluation model, scheduling details, various privacy levels and participation specifications etc. System details comprised of architectural documentation, system documentation, architectural approaches etc.

In the first phase, the evaluation creator needs to select the evaluation method to be used throughout the whole evaluation process. Various evaluation participants, their roles and rights, generated output and many other things depend on the underlying evaluation model. An evaluation model also known as evaluation method can be defined as a sequence of steps through which evaluation participants can assess the system's architecture against the requirements of its customers, the architects and all other stakeholders. Along with all other details, the selected evaluation model will also be persisted and passed over to the execution component. As shown in figure 4.2, this component takes custom evaluation method details from the custom evaluation method unit to let the evaluation creator specify a customized evaluation models among existing scenario-based evaluation methods.

Evaluation scheduling details can be provided to this component. These details mainly contain various specifications pertaining to synchronous and/or asynchronous evaluation.

All the scheduling specification will be forwarded to the component of execution unit that is responsible for managing the schedule of the evaluation. Moreover, the evaluation creator needs to provide the participation details related to evaluation. Participation details comprise of availability of the evaluation to other registered users for the participation. It can be open call or closed or invite only type of participation. Team organization unit will behave according to this specification which is discussed in following subsection.

System details also need to be provided to this component that later can be used by other participants. System details mainly consist of architecture documentation, system documentation, overall architectural approaches taken and all the required quality attributes with optional priority specifications. These details will be made available to each participant to get the good idea of the system and other necessary details. As shown in figure 4.1, this component is connected with the evaluation execution management component. Through this relation, this component lets the execution management unit access all the aforementioned information related to the evaluation to successfully enable the effective participation of participants. Some of the aforementioned details will also be made available to all registered users (for open evaluation) to let all registered users get the access to basic idea of the system and its architecture. These details will help them determine appropriateness of the evaluation for their skills and interest. Along with that, the evaluation creation management component will also use this information to do necessary processing and allow freelance registered users to participate in appropriate evaluation.

All registered users will also be shown the list of all the evaluations in which one is participant; a user can then select the desired evaluation and continue the process. This component will help users for selecting appropriate evaluations. With the suitable evaluation found, a user can request for the evaluation.

4.2.2 Evaluation methodology management

As discussed in previous component's description, at the time of creating an evaluation, a registered user can choose an evaluation model from the available evaluation models list. The selected model will be applied to the respective evaluation and will remain unaltered till the end. In the list of available evaluation models, a registered user can choose existing scenario-based evaluation methods such as ATAM, SAAM, CBAM, ALMA and FAAM.

In addition to the well-known evaluation methods, this component allows registered users to create custom evaluation methods and modify existing ones. Chapter 5 on custom evaluation creation gives the detailed description of this phase. A custom evaluation methodology creator can also specify different roles for each step. Instead of creating a new step, a creator can look into repository of steps and can use the existing one if its functionality fulfills the requirements. This repository may be filled by other users while creating their customized evaluation models. In addition to that, a registered user can modify any of the supported evaluation models e.g. ATAM, SAAM etc., or override some of the existing evaluation models from the repository. Well-known scenario-based

software architecture evaluation methodologies (ATAM, SAAM, etc.) are implemented and contained within this component.

Each step in an evaluation method is usually a sequence of one or more utilities/functionalities. In other words, combined utilities create a step in an evaluation model. This system has a large pool of implemented utilities along with their documentation. So, a custom evaluation methodology creator can choose and arrange them based on the requirements with the customized access rights to rest of the participants. More details on this component are elaborated in chapter 5.

4.2.3 Team organization management

Figure 4.3 shows the structure of the team organization management component. Another important thing for a software architecture evaluation is the evaluation team and its management. After providing necessary information related to the evaluation and selecting evaluators, the evaluation creator will be redirected to this component where he will be able to manage evaluation team members. This component allows the evaluation creator to grant access rights to various participants. If needed, the creator can also assign more roles to the same member who will have a union of access rights of all the granted roles. This component should be able to retain information of various teams with respect to their evaluations and roles in one or more evaluations. Set of roles depend on the evaluation model, so this component grabs details on roles with respect to the evaluation model from the evaluation methodology management component.

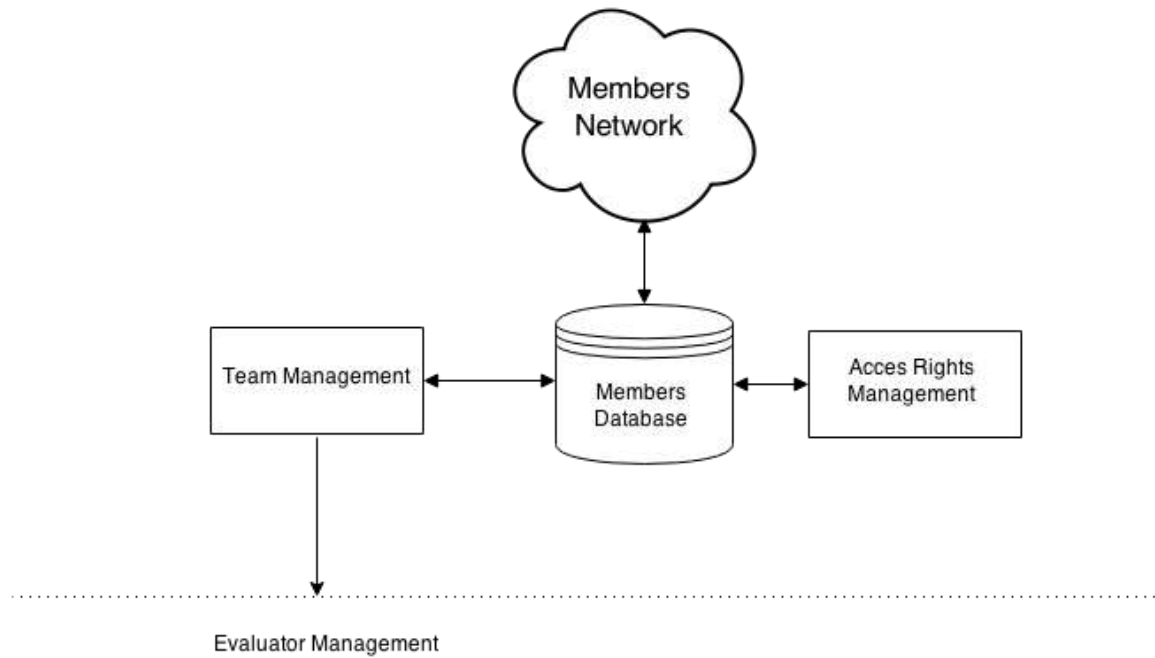


Figure 4.3 Team Organization Management Component

Not only the evaluation creator can manage various teams related to particular evaluations, but also he can monitor the performance of evaluation teams at any point during the evaluation. In case the same architecture needs to be reevaluated, with this performance monitoring facility the creator can determine what changes need to be implemented to the current team(s) so that the performance and output can be improved.

As shown in figure 4.3, the member database is shared between the evaluator management component and the team organization management component. This database contains all the necessary information related to all registered users in the system. How the evaluator management component can access and update data in this database is described in upcoming subsection. This database also contains the information of relations between registered users. As mentioned in chapter 3, no two

users can collaborate without connecting with each other through the system. Thus this component is responsible for managing the connection network of all the registered users. With this network, evaluation participants can collaborate with the same and/or different participants in future evaluations and hence growing the social network among evaluation participants. In order to enhance the system, various social networking algorithms are implemented in this component.

Also mentioned in chapter 3, large scale software architecture evaluation needs number of evaluation teams. This component helps the evaluation creator to manually form multiple evaluation teams from the large pool of interested users to perform evaluations of the software architecture using same or different evaluation models.

4.2.4 Evaluator management

As mentioned in chapter 3, a registered user of the software architectural evaluation system is either an evaluator or an evaluation creator. However, the same registered user can be both evaluator and creator as any user can play different or same roles in one or more evaluations at the same time. So, the component responsible for managing all the evaluators of the system is called evaluator management component.

As shown in the figure 4.4, this component comprises of several units namely role and rights management, participant selection management, member registration unit and

session management. Along with that, this component also has the database called member database which is shared with the team organization management component.

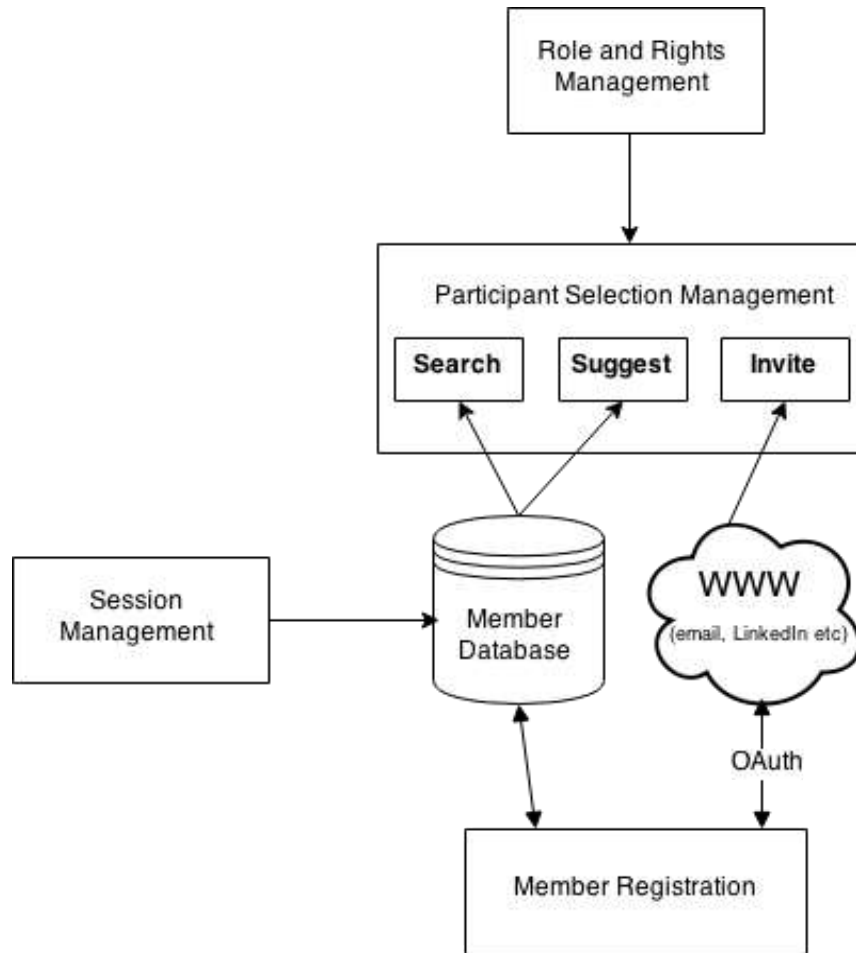


Figure 4.4 Evaluator Management Component

The member registration unit allows a user to register by creating a profile. This unit is responsible for user profile creation and management of user specific credentials, too. A user profile consists of personal data associated with a specific user. A user is free to decide what details of the profile can be visible to other users and what should be private.

As shown in figure 4.4, this component is also responsible for relating user profiles with outer social media. A registered user can (at a time of profile creation or later) connect one's profile with one's account in other social networking services such as LinkedIn, Facebook, and Twitter etc. As a result, more information regarding users can be obtained from connected other social media. For an instance, rather than providing professional experience information while creating the profile, a user can let this component to retrieve such information from LinkedIn's connected account. While connecting this system with other social networking services, user authorization must be reliable. Hence, this component uses OAuth [20] to get the secured delegated access to users' profiles from other social networking services. All the access tokens generated in this process (associating users and social networking service) are stored, updated and retrieved from the member database.

As mentioned earlier, a user creates a profile containing one's information and self-claimed credentials as a part of the registration on the software architecture evaluation system through this component. This information is stored and used in many ways by the system and may be viewed by other registered users. The component uses this information to provide effective suggestions to evaluation creators in order to help them identify appropriate participants for their evaluations.

The user profile is also used by the search unit of this component. This information can be exploited by the system taking into account the users' characteristics and preferences. The search utility parses various fields of the profile and tries to match search

specifications. The user can then use these search results to connect and/or collaborate with other registered users.

Another important unit of this component is login and session management unit. This component is responsible for checking the validity of the credentials before displaying information regarding evaluation histories of users. This unit also manages the session of every user in various evaluations at the same time. It redirects users to respective evaluations with necessary session commencement according to the roles of the users in respective evaluations. The information about roles and rights of each user for an evaluation can be retrieved from the member database. This unit is also responsible for destroying user evaluation sessions when logging out.

As shown in figure 4.1, this component is connected to the post evaluation management component. This connection implies that the evaluator management component provides necessary information regarding evaluators to facilitate the feedback procedure. And with the feedback provided, this component reflects the changes users' profiles and uses that in suggesting future evaluation participants.

Figure 4.1 shows; this component is connected with the team organization management component. This relation realizes that the evaluator management component takes role information of team members with respect to an evaluation from the team organization management component. This information is to be associated with each evaluator and used for redirecting users to their evaluations.

4.2.5 Evaluation execution management

As shown in figure 4.5, the evaluation execution management component is responsible for managing the actual evaluation process by various participants using selected evaluation methodology. As shown in figure 4.1, this component takes evaluation details (along with the methodology) from the evaluation creation management component and team(s) details from the team organization management component. A registered user (except evaluation creator) is directed to this component after selecting the evaluation to participate. Based on the specification of the selected evaluation and one's roles, the user can perform evaluation related activities. In case of evaluation creator, after creating the evaluation he is redirected here to be a participant and/or to monitor the evaluation.

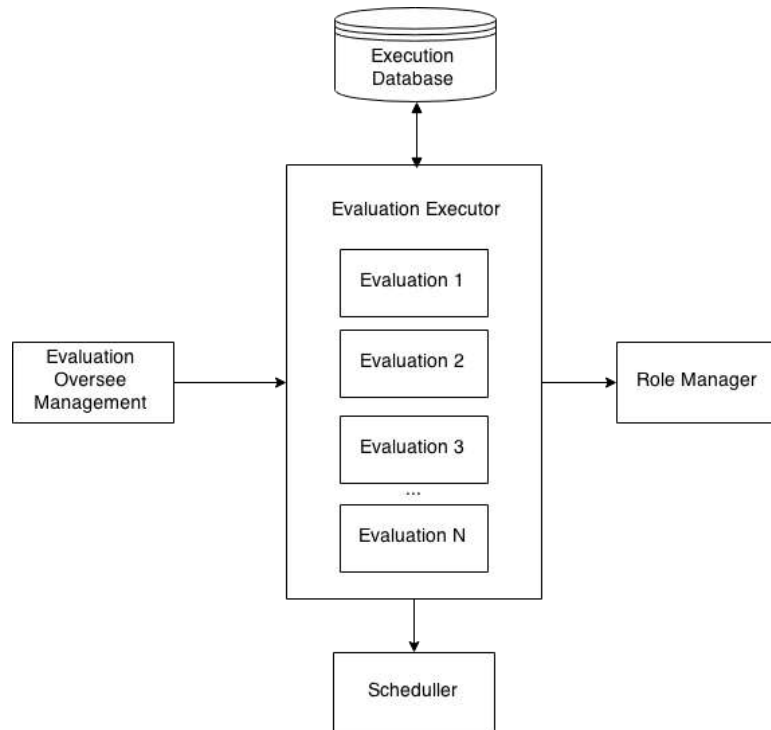


Figure 4.5 Evaluation Execution Management Component

The main unit of this component is the evaluation executor. This unit is responsible for creating an instance of the evaluation execution engine configured with the underlying evaluation methodology. As shown in figure 4.1, this unit obtains an instance of the evaluation execution engine from the evaluation methodology management component as specified in the evaluation creation phase. During the evaluation process, it retains the state of the engine and the evaluation, so an evaluator can resume participation at any time. Moreover, it also maintains a single instance per evaluation process and shares among all the participating evaluators.

Also shown in the figure 4.5; this unit retrieves data from the role manager subcomponent which contains information pertaining to role and access rights of the users redirected to this component. The role manager takes information from the team organization management component related to team and roles to participants mapping. As a result, this unit allocates privileges to users to let them perform necessary activities related to the evaluation. The definition of roles may differ as in different evaluation models. So, as defined in the implementation of an evaluation model, the executor facilitates users accordingly.

As specified in the previous section, a user can play same or different roles in one or more evaluations at the same time. This component is responsible for facilitating that. Based on selected evaluation and user's role in it, the executor unit loads appropriate data and provide platform to let the user perform duties based on the selection.

Software architecture evaluation can be synchronous or asynchronous (or combination of both) as discussed in chapter 3. The scheduling management unit is responsible to maintain timetable of any synchronous evaluation. This unit can be configured by the evaluation creator (or scheduler) with respect to the evaluation disparate to the evaluation method. The scheduling management unit manages each phase of an evaluation with respect to temporal deadlines and informs participants about remaining time for the given point of the evaluation's workflow. As shown in figure 4.5, this component obtains the scheduling specification from the evaluation creation component and the evaluation creator can also configure this unit. Moreover, the scheduling unit also controls the commencement of any phase of the given evaluation. In other words, a new phase (or step) of the evaluation process cannot be started before its scheduled start time.

Another unit of this component is evaluation oversee management. This unit lets the evaluation creator(s) monitor ongoing evaluations at any point of time. Since this unit has access to all the information pertaining to any evaluation, it can show the statistics about evaluation. When an evaluation creator navigates to this component, one can choose to monitor any evaluation from the list of evaluations. An evaluation creator can also grant monitoring access to other team members of the evaluation. By selecting an evaluation from the given list, the evaluation oversee management unit shows various statistics such as progress of the evaluation, participations and contributions of team members, various scheduling stats etc. As a result, the evaluation creator can get insight of any evaluation at any given time and can take necessary steps.

4.2.6 Post evaluation management

After evaluation completion, participants are redirected to this component for all the after-activities of the evaluation. This component mainly handles all the post evaluation activities with its own user interface and database. As shown in the figure 4.6, this component comprises of several subcomponents such as participant feedback management, output storage along with comparator and other output management units.

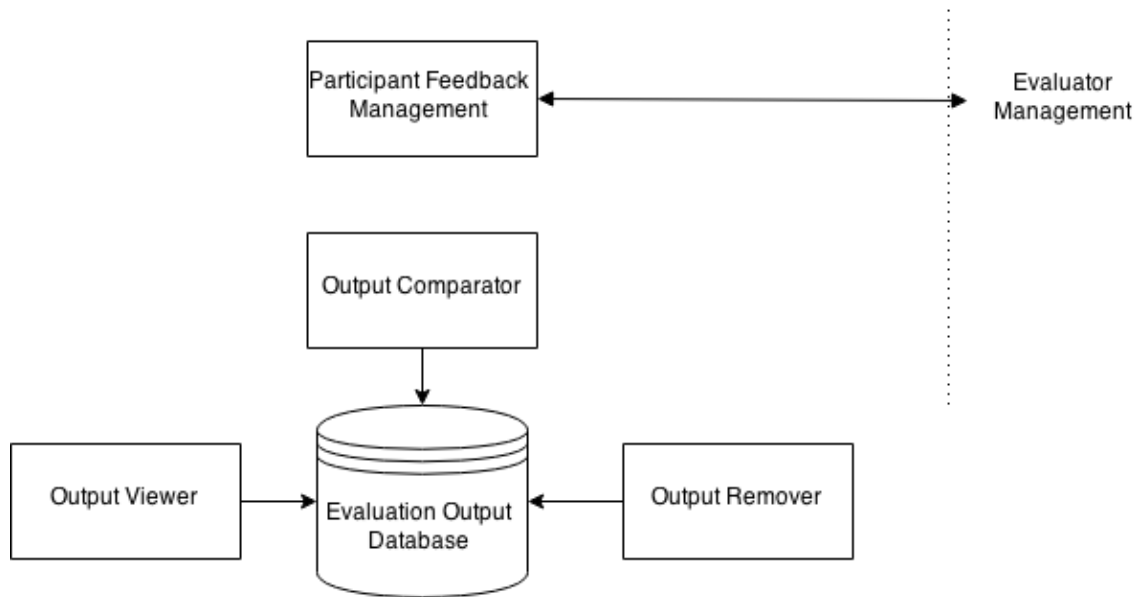


Figure 4.6 Post Evaluation Management Component

The participant feedback subcomponent lets participants provide their feedbacks about all other participants related to the experience in particular evaluations. The component is responsible for recording the feedbacks. As shown in figure 4.1, this component supplies feedback related information about each participant to the evaluator management

component and as a result that information is used to provide smart suggestions to the evaluation creator to select future participants.

The other subcomponent is the output management and comparison unit. Each type of evaluation methods generates some set of outputs at last. As an instance, the architectural trade-off evaluation method (ATAM) generates a set of architectural approaches, utility tree, a set of risks and non-risks, risk themes and large number of scenarios. This component provides user interface where all participants can view the output of the evaluation in text format or other graphical representation. The evaluation creator can not only view the output but also point out the important findings from the output.

As mentioned in chapter 3, in case of large scale evaluation more number of self-organized teams perform the evaluation following same or different evaluation methods. They generate different sets of output in the end. Moreover, as per the hypothesis presented in chapter 3, different groups of evaluation participants following the same evaluation model may generate different sets of output. So to help evaluation creators get the best findings, this component helps them compare various findings generated by different teams.

This component is also responsible for making evaluation output visible and letting other users view that to get the insights regarding the evaluation system. As a result, a registered user can search for the evaluation and view its output along with details of the software system and its architecture. There is an output availability flag which can be

specified in the evaluation creation process. The evaluation creator can decide about making output available during this phase. In the case of conventional (closed) evaluation, this component lets the evaluation creator remove (hide) the findings so as not to let other users to view that.

4.3 Different Types of Software Architecture Evaluation Realizations

Having discussed system architecture and components, their functionalities and relations, this section provides detailed description of how aforementioned components form suitable systems for different types of software architecture evaluations. There are mainly three types of evaluations which can be created in this open and distributed software architecture evaluation system. They are conventional (closed and invite only) software architecture evaluation, open (open call for) software architecture evaluation and fully open (crowdsourcing based) software architecture evaluation. The following is the description of how every component plays its role independently yet depending on the output from other components in the given three types of evaluations and supports participants performs their evaluation activities accordingly.

4.3.1 Conventional software architecture evaluation

Conventional software architecture evaluation, also known as closed software architecture evaluation with invite only participation is the most basic type of software architecture evaluation. Most of existing scenario-based software architecture evaluation methods is designed for a closed team to evaluate the prospective system's architecture at

any time during software development lifecycle. In this type of evaluation the close evaluation team can use the proposed system for the evaluation of the prospective software system while keeping it private and isolated from other users.

In this type of evaluation, pre-selected evaluators use system to evaluate the prospective software architecture evaluation. Firstly, the evaluation creator creates the evaluation with all the necessary information. Here, the evaluation creator can select the evaluation models from the existing evaluation models contained in the system or define customized evaluation methodology through the custom evaluation method management component. Along with the evaluation model, the evaluation creator specifies other information pertaining to the prospective software system through the evaluation creation component which later can be used by other team members during the evaluation.

Following to that, the evaluation creator needs to associate all the team members to the respective evaluation. This process is facilitated by the evaluator management and team organization management components. As all team members will use the system for their evaluation, each team member has to be registered user of the system. So, as a next phase, the evaluation creator asks all the members to create profiles and register themselves to the system. After that, the evaluation creator identifies them in the system to form the team. This is carried out through the evaluator management component. As shown in figure 4.1, the evaluator management component transfers team information to the team organization management component through which the evaluation creator

assigns role(s) to each team member. This role (access rights) assignment is based on the evaluation model.

Having decided evaluation methodology and participants to the evaluation, the actual evaluation begins where each member plays respective role(s) and performs necessary actions. This phase is aided by the evaluation execution component. The scheduler of this component manages the temporal deadlines of the evaluation process as specified by the evaluation creator. The oversee management unit lets the evaluation creator monitor the evaluation process by providing various statistics and insights of the evaluation at any point of time. As the system supports evaluation in distributed environment; team members can collaborate and perform their duties at their suitable time at distributed locations.

At last, after the evaluation of the prospective system, team members can view the outcomes of the process. Again, this output also depends on the evaluation model. If the team is not satisfied with the output, they can carry out the evaluation process by making subtle changes. Since it is closed software architecture evaluation, all the necessary details about the software system, its architecture and output remain private. The evaluation creator can remove any related detail from the evaluation system's database after evaluation completion.

4.3.2 Open team software architecture evaluation

Open software architecture evaluation with open call for participation is another type of software architecture evaluation. In this type, any registered user can be a part of the evaluation and play assigned role(s) by the evaluation creator.

In open software architecture evaluation, the evaluation creator provides necessary information about the prospective software system along with its architecture. Along with that, the evaluation creator provides other specifications regarding the evaluation in this phase such as evaluation model, number of participants, duration and other scheduling information. The evaluation creator can avail other information pertaining to the software system to be evaluated which later can be used by participants during the evaluation.

The evaluation creator can create an open evaluation through the evaluation creation management component. As a result, any registered users can view the evaluation information and request for participation. The evaluation creator can accept or reject (ignore) the requests. To help the evaluation creator determine the suitability of requesting users for the evaluation, the evaluator management component shows details of users such as their skills, self-claimed credentials, previous participations and reviews in users' profiles.

In addition to that, the evaluator management component facilitates searching suitable participants by letting the evaluation creator search for the expertise from the wide catalog of members in the system. Here, the evaluation creator can search based on

various criteria such as experience, skillset and other self-claimed credentials of the user. Moreover, the evaluator management component is equipped with the smart suggestion unit which refers appropriate potential participants to the evaluation creator. To do that, the component takes evaluation details and self-claimed credentials of all the registered users into account to provide smart suggestions. The evaluation creator can then invite the suggested users to participate in the evaluation. If the evaluation creator wants to collaborate with potential evaluators, external to the system, he can invite them to be the registered users of the evaluation system via emails through the evaluator management component.

After the team members' selection, the team organization component helps the evaluation creator to assign role(s) to each team member. This role (access rights) assignment is based on the evaluation model. Here, team member can request for specific role to the evaluation creator for the respective evaluation.

After deciding evaluation model and selecting (invited and/or accepted) participants to the evaluation, the actual evaluation begins where each member plays assigned role(s) and performs necessary actions. This phase is supported by evaluation execution component similar to the conventional (closed) evaluation. Scheduling unit manages the temporal deadlines of the evaluation process as specified by the evaluation creator. The oversee management unit lets evaluation leader monitor the evaluation process by providing various statistics and insights of the evaluation at any point of time. The

evaluation leader can stop or alter the ongoing evaluation process based on the current progress.

In the last phase, after the evaluation process is completed, all the team members can view the outcome of the process. As its open software architecture evaluation, all the necessary details about the evaluated software system, its architecture and output are made available to all users of the system. This helps other registered users to gain the useful insights about any previous software architecture evaluations and guidance of using the system for future evaluations. All registered users, can search and scrutinize previous relevant open software architecture evaluations and their outputs.

4.3.3 Software architecture evaluation by crowdsourcing

Crowdsourcing for software architecture evaluation is another type of software architecture evaluation. In this type, an evaluation creator creates an evaluation by making necessary information about the prospective software system available openly. After that, number of self-organizing teams can be formed to evaluate the provided software system's architecture separately. The self-organized teams are independent in terms of choosing evaluation methods and can carry out their evaluations using proposed system.

Self-organized evaluation teams are facilitated by the evaluator management component and the team organization management component. Since the information pertaining to

the evaluation is publicly available, any interested registered user can request to participate in with the preferred role. Here, search, invite and suggest units of the evaluator management component can also be used as explained in the previous subsection. As a result, the evaluator management provides pool of interested users to the team organization management component where team members are chosen manually. An evaluation creator or a designated team leader is responsible of forming number of balanced teams and allocating roles and rights to members. He can take self-claimed credentials, past experience, previous participations and requested roles into account while dividing users in number of teams. This whole process is supported by the team organization management component.

After team formations, participants start evaluation in a self-governance manner by following arbitrary evaluation model. Every team member plays designated role(s) in a particular evaluation and performs necessary duties. This whole execution process is aided by the evaluation execution management component. As mentioned earlier, the evaluation system lets a registered user to participate in any number of evaluations at the same time; this component maintains sessions in each evaluation. The scheduling unit of this component manages the temporal deadlines of the evaluation process as specified by the evaluation creator of each team. The oversee management unit provides evaluation teams various statistics and insights about all the ongoing evaluations at any point of time.

After the evaluation process is completed, the numbers of teams have evaluated the architecture in different ways. The evaluation creator can scrutinize output of each team and compare them to find the best or combination of best. Likewise the details about the software system, results of all the evaluations are made publicly available. As mentioned in the previous section, this helps other registered users to gain the useful insights about previous software architecture evaluations and guidance of using the evaluation system for future crowdsourcing based evaluations. Along with that, each participant is asked to provide the feedback about other participants based on one's experience with them during the process. These reviews are further used by the evaluator management to provide smart suggestions.

4.4 Summary

This chapter is all about the system's design, its various components and how they all work together to form the best system for these types of software architecture evaluations. The system consists of six components namely evaluation creation management, evaluation methodology management, evaluation execution management, post evaluation management, team organization management and evaluator management. Each component has its own set of subcomponents and all the functionalities and importance are elaborated in sections. At last, the chapter describes how the system and its components realize conventional, open and crowdsourcing based software architecture evaluations.

Chapter 5

Custom Evaluation Method Creation

The open and distributed software architecture evaluation system supports all the existing evaluation methodologies along with customized evaluation models. Here, a customized evaluation model can either be a newly defined evaluation methodology or a modified version of existing evaluation methodology. The component “Evaluation Methodology Management” shown in figure 4.1, allows users to create custom evaluation methods and modify any existing ones (ATAM, SAAM etc.).

Two well-known evaluation methods ATAM (Architectural Tradeoff Analysis Method) [21] and SAAM (Software Architecture Evaluation Method) [22] have been exemplified and kept in the center of the discussion. Both are scenario-based software architecture evaluation methods. Here, scenarios describe interactions between users and the system and it is the primary input to any scenario-based evaluation methodology. ATAM is an enhanced version of SAAM. Software Architecture Analysis Method (SAAM) has been proposed by the software engineers who were looking for a method to express different quality allegations of software architecture(s) with respect to different scenarios and business drivers and to evaluate them. ATAM as an extension to SAAM, not only assesses the quality attributes and business drivers, but also explores the quality aspects and their interdependencies emphasizing trade-off points and risks.

This chapter provides the detailed description on this phase of evaluation methodology creation. Firstly, the commonalities among existing scenario-based software architecture evaluation methods are discussed. Following to that, a detailed diagram of the component (Evaluation methodology management) is given. In addition to that, a detailed description of the process of creating customized evaluation methodologies is given.

5.1 Common Elements of Software Architecture Evaluation Methodologies

All proposed scenario-based software architecture evaluation methods are for evaluating software system's architectures against the requirements of customers. As a result all these methodologies share some similarities. This section describes similarities among various scenario-based evaluation methodologies in terms of their elements with the focus on comparing mainly ATAM and SAAM and exemplifying them.

Every evaluation methodology comprises of different evaluation phases. Generally evaluation phases have different forms for certain evaluation stages such as preparation for evaluation, analysis of scenarios, testing of architectural aspects and summarization. As an instance, ATAM method consists of four phases: presentation, investigation and analysis, testing, and reporting. Each phase consists of a collection of steps. Further, each step is a sequence of one or more evaluation functionalities, sometimes executed in parallel with different sets of users.

As mentioned above, each phase consists of one or more steps with involvement of some team members. The first phase called preparation for the evaluation is similar in all scenario-based evaluation methods. During this phase, participants assemble essential inputs about the software system to be evaluated by participants to use later in the evaluation. At a time of evaluation creation, following details are provided in ATAM and SAAM. Software system's quality attributes and business drivers, list and detailed description of scenarios and, system architecture description.

Another similar component for every software architectural evaluation methodologies is role. Roles represent the access rights and duties participants are supposed to perform during the evaluation process. All the participants are divided based on this parameter. All the evaluation methods have three sets of participation roles and they are classified as external stakeholders, internal stakeholders and evaluation team. The evaluation team may or may not have direct involvement in the development of software system to be evaluated. The evaluation team consists of evaluation team leader, analysts, scheduler, writer etc. Here external stakeholders are product customers, project managers, end users (may be different from customers), investors, etc. On the other hand, internal stakeholders are architects, designers, testers, performance engineers, database administrators etc. However, depending on the evaluation method these sets of roles can be further categorized.

Lastly, the common component among different scenario-based evaluation methods is outcomes of the evaluation process. Irrespective of the evaluation methodology, every

evaluation process generates a set of outputs. Based on the expected outcomes of evaluation methodologies, an appropriate evaluation methodology can be selected with respect to the prospective software system and potential evaluators' skill and experience. However, as mentioned in the previous section, the proposed system also supports reevaluation of the same architecture with different evaluation models if unsatisfied with the current evaluation output. Outcomes of SAAM are- brainstormed scenarios, their relation with the architecture, necessary changes in the scenarios and required time, cost and efforts to make such changes. As mentioned above, ATAM is an enhancement of SAAM. So, in addition to outputs of SAAM, outcomes of ATAM are the interrelationship between scenarios and quality attributes, architecture extraction based on scenarios (use cases and interactions), risks and non-risks. Therefore with the support of functionalities required to present and generate the ATAM output, SAAM is supported too.

5.2 Common Tasks of Software Architecture Evaluation Methodologies

As mentioned in the previous section, in each step of evaluation there is a set of tasks to be performed either sequentially or in parallel by participants. The behaviors of tasks depend on the evaluation methodology. Along with the common components, different scenario-based software architecture evaluation methods also have some functionalities in common. However, each evaluation methodology has a unique set of functionalities. This implies that, if a superset of functionalities/tasks (union of the functionalities of all evaluation models) are implemented and contained in the evaluation system, a type of

evaluation model can be created by selecting and combining some tasks. Moreover, this superset can be made richer by adding user-defined functionalities through predefined interfaces. This section provides detailed description of commonalities in terms of functionalities among various evaluation methodologies with focus on ATAM and SAAM.

After selecting an evaluation methodology, the evaluation creator needs to provide the necessary details to be used by participants during the evaluation process as a first step. However, the required details may differ in various evaluation methodologies. Following to that, an appropriate group of participants need to present the details to other participants of the evaluation, for them to familiarize the system. As an instance, architect presents the existing architecture of the prospective software system to the rest of the stakeholders and the evaluation team members. In a nutshell, all evaluation methodologies have some steps only to brainstorm and present the prerequisites of the evaluation. SAAM has steps called present architecture and scenarios and ATAM has presentation phase which consists of presentations of the method itself, business drivers and system's architecture.

Every scenario-based software architecture evaluation method mainly depends on various scenarios of the software system to be evaluated. After presenting and brainstorming inputs, a scenario-based evaluation methodology needs to have some steps for analyzing scenarios and other provided details through team collaboration. Here, analysis of scenarios consists of evaluating scenarios against quality attributes, system's architecture

and customer requirements, prioritization and scenario interaction assessment etc. SAAM has steps for individual scenario evaluation and scenario interaction assessment and ATAM has step for brainstorming and prioritizing scenarios. Along with scenario evaluation, some evaluation methods also consist of scenarios development. SAAM has the first step for scenario development and in ATAM after generating scenarios in analysis phase; stakeholders can generate even more scenarios later in the evaluation.

The previous section mentions that every evaluation methodology has a specific set of outcomes. So, the last step of an evaluation method is to generate and present the outcomes of the evaluation process. This output depends on the whole evaluation process and involvement of the participants. Similar to initial phase of presentation, the evaluation process finishes with the output submission. As a result, all scenario-based software architecture evaluation methods share functionalities of output generation and presentation. SAAM has scenario presentation and prioritization in terms of its relative importance to the success of the prospective system as the last step. With ATAM the evaluation team summarizes and presents the results to the stakeholders based on the information collected during the session. However, output is only presented in the end of the evaluation process but different outcomes are generated in almost each phase of the evaluation process. After presentation, the outcomes can be made publicly available to let other non-participants get the useful insights of the software system being evaluated.

In the previous sections, commonalities amongst evaluation methodologies have been discussed in terms of common components and functionalities by emphasizing on the

analogy of two well-known software architecture evaluation methods ATAM and SAAM. Apart from these two, there are three more scenario-based evaluation methods that have been proposed, they are CBAM - Cost Benefit Analysis Method [3],[6], ALMA - Architecture Level Modifiability Analysis [7],[8] and FAAM - Family Architecture Analysis Method [9]. CBAM is an architecture-centric method for analyzing the costs, benefits and schedule implications of architectural decisions. CBAM has very similar components and functionalities to ATAM, and it starts when ATAM ends. ALMA is another scenario-based analysis method built on top of SAAM just like ATAM. FAAM is based on the same general evaluation principles like SAAM and ATAM. In a nutshell, any other scenario-based software architecture evaluation methods are either based on or similar to SAAM and/or ATAM. So, commonalities analyzed in this section also apply to other scenario-based evaluation methodologies.

5.3 XML based Specification Language for Software Architecture Evaluation

Methodologies

In the previous two sections, commonalities among various scenario-based software architecture evaluation methodologies are discussed. These similarities are drawn in two aspects namely common components and common tasks. As mentioned above, each evaluation methodology follows almost the same structure. In other words, any of the above software architecture evaluation methods roughly pursues some sequence of the following phases: taking necessary details about the software system to be evaluated, discussing and/or generating architectural approaches, business drivers, scenarios etc. and

present useful findings. All scenario-based evaluation methods take, manipulate or generate scenarios during the evaluation process.

So, creation of new scenario-based software architecture evaluation methodologies will follow the same structure and have same basic elements and functionalities. Hence, scenario-based software architecture evaluation methodologies (either existing or new/customized one) can be described by a generalized XML specification. This specification contains annotations based on commonalities described in the previous sections.

Not all the evaluation methodologies are exactly same, but they also vary at some extent. These variable entities make them unique and best suitable for particular evaluations. In order to support variable entities, the specification (XML) file has different attributes specification associated with each XML element. The specification file is designed in such a way that not only attributes customize the methodologies but also the elements can vary, too. As an instance, an evaluation method can have a number of “step” elements, but according to the strict specification structure, they must be within a “phase” element. The detailed description about the specification structure and all the accepted elements is given below. Since a user can create a customized evaluation methodology by providing the XML specification, one can skip element(s) without disobeying the structure specification. The evaluation methodology execution engine is designed in such a way that it can take care of the skipped element. A complete description of the engine and its functionalities is given in the following subsection.

```

<evaluationMethodology name="MyEvaluation">

  <prerequisite>
    <input name="organization" optional="true" title="Organization Name"
type="text">
      <input name="architecture" optional="false" title="Architecture Diagram"
type="file" format=".jpg">
        [...]
    </ prerequisite >

  <roles>
    <role category="Evaluation Team" name="Scheduler"
type="predefinedRole1">
      <role category="Internal Stakeholder" name="Architect" type=
"predefinedRole2">
        [...]
    </roles>

  <process>
    <phase name="Presentation Phase">

      <step name="Present Architecture" number="1" duration="5">
        <task order="1">
          <functionality type="presentation" argument="architecture">
            <player> Evaluation Team </player>
            <visible> Internal Stakeholder </visible>
            [...]
          </functionality>
          [...]
        </task>
        [...]
      </step>
      [...]
    </phase>
    [...]
  </process>

  <outcome availability=public>
    <output name="risks" type="list" step="6" title="Risks">
      <output name="utility tree" type="tree" step="7" title="Utility Tree">
        [...]
      </outcome>

```

Figure 5.1 Sample XML file for evaluation methodology specification

Figure 5.1 depicts the structure and all the possible elements of the XML specification for an evaluation methodology specification. As shown in figure, the root element is “evaluationMethodology” having name attribute for the title of the methodology. The structure is similar to any typical XML file, having fixed hierarchy of various elements. Each element has a set of attributes, required to specify variable entities of a customized evaluation model. The engine (described in subsection 5.4) only compiles and parses XML specification file whose structure (hierarchy) has been followed strictly. Detailed description and importance of each element are given below. See the full XML schema for evaluation methodology specification in appendix A.

5.3.1 Input element

First child of root element is input element. As mentioned above, every scenario-based evaluation method needs some input details given by the evaluation creator. These details are then used by other participants during the evaluation process. In this element, the custom evaluation creator can specify different input elements required for the evaluation. As a result, when an evaluation creator uses this customized evaluation model, he needs to provide specified input materials before starting the evaluation.

As shown in the sample XML file, input element can have any number of sub-elements representing different inputs. However, there must be only one input element which can contain any number of inputs. Input sub element can have an attribute called optional that can be either true (default) or false. This attribute makes the input as a required field to be

filled in while creating the evaluation. The title attribute will be rendered as a title for the respective input. Type attribute helps the evaluation methodology execution engine to determine the type of input such as text, dropdown, multi-select, file, etc. in order to render and process appropriately. With type element's value being 'file', it comes with format attribute to restrict file type (such as .pdf, .gif, .jpg) to be provided by the evaluation creator. The name attribute is used to refer a particular input, to be used in the evaluation process.

The sample XML file shows two different input elements. The first one called 'organization' is displayed as an optional text field with title "Organization Name" when creating an evaluation. Next one lets the evaluation creator browse one's computer to upload any .jpg file that contains the architecture diagram of the prospective software system to be evaluated.

5.3.2 Roles element

Another common component of an evaluation model is role. Role represents rights and responsibilities during the evaluation of a participant. Even though all the roles are divided in three types, evaluation team, internal stakeholders and external stakeholders, each evaluation has its own set of roles. So, this element lets the custom evaluation methodology creator to specify different roles with respect to the evaluation method. As a result, while adding a participant to the evaluation, an evaluation creator can assign some of these roles to participants.

This element is written as “roles” and contains any number of sub-elements “role”. However, there must be only one roles element within the XML file. Here, each sub-element represents a role that can be assigned to participant(s) of the evaluation using the created evaluation methodology. The access rights, responsibilities and duties depend on the type of role a participant is assigned to.

After defining various roles within this element, the custom evaluation methodology creator can refer them in the process element to specify the duties of each role with respect to a step. As an instance, the evaluation creator can reference role named ‘architect’ as a player for the functionality of architectural presentation in the first step.

As shown in the sample XML file, “role” element has three attributes namely “category”, “name” and “type”. Category attribute refers to any of the three categories of participants’ viz. evaluation team, internal stakeholders and external stakeholders. This specification helps the evaluation methodology execution engine to provide appropriate basic access rights to the participants. Next required attribute is name, which specifies the name of the role with respect to the evaluation. This name is used to refer role in “process” element’s step sub-elements. More details on “process” element are in subsection 5.3.3. The last required attribute is type, which represents the type of role. Here, type points to the member of implemented (pre-defined) set of roles.

The evaluation methodology management component contains the superset of all the roles for scenario-based software architecture evaluation methodologies. The

implementation also contains basic properties of each role, such as scheduling management by scheduler. However, the duties to be performed within a step of the evaluation process are specified by the custom evaluation methodology creator in the “process” element. In other words, the features and duties of each role depend on the specification of the “process” element. As an instance, it can be defined in a specification that the project manager will present his scenarios in some step, while other stakeholders will also present their scenarios in another step.

5.3.3 Process element

As mentioned earlier, software architecture evaluation methods have some functionality in common. This implies a set of the common functionalities of all evaluation methodologies can be implemented and contained in the evaluation system, to create different types of evaluation methodologies. This knowledge base of implemented functionalities can be made open for users to design their own evaluation methods and develop new functionalities.

The central and the most important element of the evaluation methodology is process, where the actual evaluation takes place. As described in section 5.2 each evaluation methodology has a certain set of phases, each of which containing one or more steps. As shown in the sample XML file, process element is the parent of the whole evaluation process. That contains phase elements as per specifications. These phases contain a set of

steps. As an instance, ATAM evaluation methodology has four phases, presentation, investigation and analysis, testing, and reporting where each phase contains three steps.

“Phase” element has only one attribute called name. This attribute gives any phase a suitable title. As shown in the sample specification file, phase is titled as “Presentation Phase”. Child element “step” under phase element has three attributes, “name”, “number” and “duration”. Similar to phase’s name attribute, step’s name attribute gives it a title. Number attribute is used in relating the step with the output element as described in output element’s section. Duration attribute is optional and used for scheduling purpose which takes number of days as acceptable values. If a step is not marked as completed before the end of the duration, the evaluation creator is notified so that one can take necessary steps.

Further, a step is divided into one or more tasks. The reason behind that is to let different participants to run different tasks in parallel within a step. However, steps are to be followed in sequence. So, step element has only one attribute called order which takes number as accepted values. Here, the same numbered (ordered) tasks within a step will be executed in parallel.

Task element consists of different functionalities. Here, functionality elements directly refer to the set of pre-written utilities, contained in the evaluation system. Any registered user can define and implement a new functionality to be used in a customized evaluation methodology. Functionality may depend on some already generated details. In that case,

argument attribute lets the evaluation creator to specify any of the existing output/input to be passed in to the functionality. Here, the existing input/output is referred by its name attribute. All the functionalities contained in the system are made publicly available. As a result, a custom evaluation methodology creator can choose suitable set of functionalities from the large pool of pre-written functionalities.

Here, the interface is defined which is required to be implemented in order to write and contain a functionality in the evaluation system. The interface of the functionality takes two arguments which are children elements of functionality element namely, ‘player’ and ‘visible’. Player elements specify the sets of participants who can perform necessary operations according to the written functionality. The possible values for player are either any existing role or category (i.e. evaluation team, internal stakeholders and external stakeholders) of roles. Second and optional element is visible which lets the methodology creator to specify participants whom the player’s activities are visible. Possible values are true and false, and if not specified the activities are visible to all the participants of the respective evaluation. One thing should be noted that, both “player” and “visible” elements only accept pre-specified roles or category of the roles.

Note that the sample XML specification file shows the step called “Present Architecture” with sequence number one and duration of five days. Further, the step has task of order one containing functionality type presentation. As mentioned above, the type attribute refers to the large pool of pre-written functionalities. To be more specific, the functionality element has evaluation team member as a player (presenter) and the

presentation is to be visible to internal stakeholders. The functionality element has architecture as an argument, which refers to the architecture diagram provided as input.

5.3.4 Outcome element

Each scenario-based software architecture evaluation methodology produces a unique set of outputs at the end of the evaluation process. With such diversity in outcomes, the evaluation engine allows scenario-based software architecture evaluation methodologies generate outputs in a predefined format sets. So, as part of the implementation, the engine is provided with a superset of different output formats and hence can support customized evaluation methodologies with different outcomes.

Outcome element consists of any number of “output” elements representing all possible outcomes by the evaluation methodology. However, there is only one outcome element per specification file. As shown in the sample XML file, output element consists of three attributes namely “type”, “name”, “title” and “step”. Title attribute gives the title to the output. Type attribute refers to the implemented display of the output. Here, based on the UI specification, different types of displays can be implemented. As an instance, list can be made more colorful, interactive and alterable through themes, too. Third optional attribute is the step also known as generating step. Here, custom evaluation methodology creator can reference the step (through number attribute of the step) in which the output must be generated. The respective step cannot be marked as completed otherwise. The

name attribute is used to refer the generated output, to be used in other step in the evaluation process.

The sample XML file shows two different output elements. The first one is risks of the type list to be generated in the sixth step. The second one is utility tree of type tree to be generated in the seventh step. Here, tree type of representation is implemented using various open-source JavaScript libraries and contained in the evaluation system.

5.4 Implementation Strategy of XML Specification

Figure 5.2 shows the component diagram of the evaluation methodologies management component with the engine that executes custom software architecture evaluation methodologies defined by XML specification. The diagram consists of four sub-components namely prewritten codebase, evaluation specification compiler, implemented evaluation methodologies and evaluation methodology execution engine. The sub-component called evaluation methodology specification (.xml) represents the storage of specification files each of which specifies a customized evaluation methodology. The rest of the section briefly describes each sub-component and details on these units work together to let user create a customized evaluation methodology, as well as using it in the evaluation.

The prewritten code base consists of already written code of common elements, described in the previous section. The code base contains different functionalities, implemented

roles and related access rights and necessary code to display different types of outputs. As described in the previous section, functionalities are written by implementing the predefined interface in this sub-component. In addition to that, different roles and respective access rights (common for all scenario-based software architecture evaluation methods) are defined in this sub-component. As an instance, it can be a step from ATAM to present the architecture by architect and let other internal stakeholders and evaluation team members scrutinize. It implies that the step consists of two functionalities i.e. presentation and discussion forum, the architect as player and visibility set to internal stakeholders and evaluation team members. So, all the functionalities and roles written in this component can be referenced in the specifications.

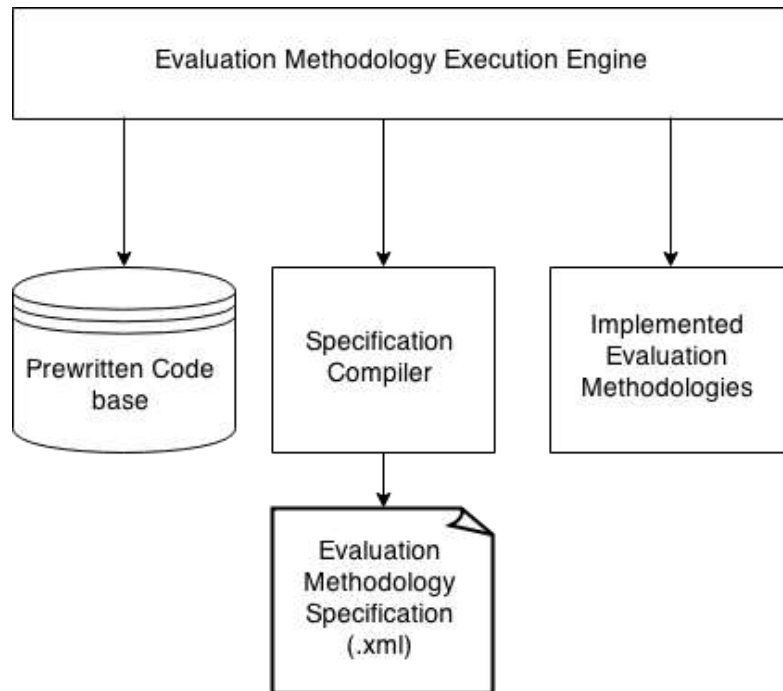


Figure 5.2 Evaluation Methodology Management

The next sub-component is the container and compiler sub-component for the XML specification files. Here, a specification file specifies a customized evaluation model based on the format described in the previous subsection. The execution engine uses this file to execute the respective customized evaluation methodology. This sub-component contains rules and grammar of the specification file. This sub-component is also responsible for compiling the specification file and then retaining and keeping it unaltered in the container. When compilation errors found in a provided specification file, this sub-component provides insightful messages to the methodology creator. As described earlier, the evaluation creation component lists all the available evaluation methodologies by retrieving them from this component. So, successfully compiled specification files representing evaluation methodologies will be listed by the evaluation creation component to let evaluation creators create their own architecture evaluation using the specified methodologies.

The implemented evaluation methodologies sub-component contains the prewritten version of existing scenario-based software architecture evaluation methodologies. This component will have a complete implementation of all the existing software architecture evaluation methods i.e. ATAM, SAAM, CBAM, ALMA and FAAM. For prototyping, architectural trade-off analysis method (ATAM) has been implemented and contained in the evaluation system. The detailed description about ATAM is covered in the following (seventh) chapter. When the evaluation creator chooses any of these well-known implemented evaluation methodologies, the engine will access code base from this sub-component. On the other hand, when the evaluation creator chooses any of the

customized evaluation methodologies, the engine will retrieve the related specification file from the specification file container and necessary prewritten utilities from the prewritten code base component.

The core of the (custom) evaluation methodology management component is the evaluation methodology execution engine. This engine is responsible for executing any prewritten or specified scenario-based software architecture evaluation methodology. As shown in figure 5.2, the engine depends on the three other sub-components viz. prewritten codebase, evaluation specification compiler and implemented evaluation methodologies. When the engine is used to execute an implemented scenario-based software architecture evaluation methodology, it retrieves the code of the complete implemented version of the evaluation methods from the implemented evaluation methodologies' code base. In case of customized evaluation methodology, the engine itself contains the implementation of commonalities of scenario-based evaluation methods and for variable entities, the engine retrieves the specification from the storage of specification files and related codebase (functionalities, roles etc.) from the prewritten codebase sub-component. Engine grabs the arguments to be passed in the pre-written functionalities from the respective specification. With combination of static and variable codebases, the engine internally generates an implemented version of the selected software architecture evaluation methodology and runs in accordance with the evaluation execution management component.

An evaluation process of the system starts by creating a new instance of the engine with the selected evaluation methodology as an argument. Based on the argument passed at a time of initialization, the engine will be configured and the state of engine is retained in the evaluation database. So, these instances are shared and accessed by different participants and maintained one instance per evaluation with one-to-one mapping. With this association, the evaluation execution management component uses the instance of the evaluation methodology execution engine with respect to the evaluation.

5.5 Summary

This section describes commonalities among various scenario-based evaluation methodologies with respect to their common elements and functionalities, with focus on analogy between the two well-known software architecture evaluation methodologies. After that, XML based specification language has been defined to define the scenario based evaluation methodology. The presented XML specification file structure consists of four elements viz. input, roles, process and outcome. Each element is further divided into number of sub elements. Here, all the elements have certain attributes which let methodology creators specify variables in defining their methodologies.

The evaluation methodology management component has four sub-components namely prewritten codebase, evaluation specification compiler, implemented evaluation methodologies and evaluation methodology execution engine. All these sub-components work together to let evaluation methodology creators create customized evaluation methodologies and use them in future evaluations.

Chapter 6

Implementation

This chapter is about the implementation details of the proposed open and distributed software architecture evaluation system. The implementation details are divided into two main sections: technical overview and implementation details of the system's components. The technical overview elaborates the practical aspects of the system containing information about relevant technologies and database insights. The implementation details section describes how each component is implemented with UML class diagrams and sequence diagrams.

6.1 Technical Overview

This section is further divided into two subsections; details on technologies used and database insights. Firstly, technological details contain brief information about all technologies and programming elements used in development of the evaluation system. In the next sub-section, the database technology used in the system along with various database design details is discussed. The prototype implementation of the evaluation system supports ATAM evaluation methodology. More details on that are covered in chapter 7 as case study.

6.1.1 Details on technologies used

In order to fulfill the system requirements, the proposed evaluation system is implemented as software as a service on Google's cloud platform called Google App Engine [23]. Google App Engine (GAE) is a Platform as a Service (PaaS) to let users build and run their applications on Google's infrastructure. Google's cloud platform made it easy for us to build and deploy our system that runs reliably even under heavy load of users and with large amounts of data. Being software as a service on Google's cloud platform, the evaluation system is centrally hosted with one configuration and all the data are being replicated by Google infrastructure to the servers worldwide.

With use of Google App Engine (GAE) as a platform to deploy the evaluation system, it inherits certain features from GAE. The system can automatically be scaled based on the current data size and traffic. Load balancers of GAE send requests to appropriate servers based on the load of each server. GAE provides runtime environment support for standard Java technologies, the evaluation system has been written in Java along with its ecosystems.

In order to manage user interactions with the evaluation system, to communicate asynchronously with the server and update display contents accordingly, there is also some client side (in browsers) implementation. Dojo Toolkit (a very well-known JavaScript library) [25] has been used to attain aforementioned client side features. All browsers are contained with the JavaScript engine which is also called JavaScript

interpreter. As an instance, the V8 JavaScript Engine is an open source and developed by Google for the Google Chrome web browser [26].

Dojo toolkit is an open source JavaScript library designed to facilitate the accelerated cross-browser design and development, efficient interaction with the backend through Ajax and easy management of large code base on client side. In order to give users desirable usability, various web pages of the system have been designed using HTML5 [27] (Hypertext Markup Language) and CSS3 [28] (Cascading Style Sheets). By proper use of Dojo toolkit along with HTML5 and CSS3, the evaluation system has been implemented as an RIA (Rich Internet Application) [29].

6.1.2 Database insights

We also implemented a highly available relational database in order to store, process and retrieve a large amount of data for various software architecture evaluations. The database residing in Google cloud along with Google Cloud SQL satisfies all the requirements with easy integration, configuration and management of large amount of data. Google Cloud SQL is a completely managed MySQL (a renowned standard query language for relational database) service hosted on Google cloud platform providing a database foundation for the evaluation system [30]. Google's cloud data store is fully managed, easily configurable and auto encrypted data store.

Google manages replication, encryption, patch management and backs up its data store. Evaluation creators of the evaluation system can create, configure, manage and monitor their database instances with real time data and traffic of requests. Moreover, all data stored in Google's cloud data storage is encrypted. Along with that, to make data access even more secured, Google cloud SQL provides user grants that can control access at the database, table, or even column level. All data pertaining to different evaluations, users and outputs are replicated multiple times at multiple locations for great durability and availability.

The whole database sub-system of the evaluation system has been divided into five databases. Every database except member database is coupled with one component of the system and only accessible from that component. The evaluation database connected with the evaluation creation management component. The evaluation methodology database is coupled with the evaluation methodology management component. The evaluation execution database is coupled with the central component i.e. the evaluation execution management component. The outcome database contains data of outputs from different software architecture evaluations. The member database is shared between two components, the evaluator management component and the team organization management component. Please see detailed database design in appendix B.

The reason for dividing the whole database system into five smaller databases is that they can be scaled, managed and configured independently. Moreover, data replication and sharding can be achieved in only heavy loaded databases independently. The data from a

database can only be accessed by the coupled component(s). If a sub-component depends on data from another component's database, it can access such data only through the coupled component. As a result, if one database or component goes down, rest of the system will function normally.

The evaluation database, associated with the evaluation creation component, persist all the details pertaining to each evaluation created on the evaluation system. Information pertaining to a software architecture evaluation can be of two types, evaluation profile and prospective system details whose architecture is to be evaluated. This database has several normalized tables containing details of each created evaluation such as evaluation model, scheduling details, and various privacy level and participation specifications. As described earlier, the evaluation creator provides details about the architecture to be evaluated in the early phase irrespective of the selected evaluation methodology. These details including architecture documentation, system documentation, overall architectural approaches taken and all the quality attributes with optional priority specification, are stored in this database.

The member database is shared between two components, the evaluator management component and the team organization management component. This database stores all the information about every user on the evaluation system. As this database is also coupled with the team organization management component, all the details of each team for all evaluations are stored in this database. The reason for sharing this database between the two components is to avoid number of inter-database calls. As this database

contains information about all the registered users, the team organization component can map data of teams to relevant users. Moreover, this database has all it needs to retain the information relation between registered users in order to perform social network analysis.

The evaluation methodology database is coupled with the evaluation methodology management component. As mentioned in the previous chapter, a new evaluation methodology can be defined by providing a specification file (.xml). This database contains all such specification files; as a result the evaluation execution engine can load a stored specification to create an implemented version of a customized evaluation methodology. Moreover, the superset of variable functionalities catalog for customized evaluation methodologies is also contained in this database.

The largest and most dynamic database is the evaluation execution database connected with the evaluation execution management component. Depending on the evaluation methodology, every evaluation generates lots of findings in each step except the final output. As an instance, discussion between team members for some topics create a discussion forum, as a result the entire discussion along with the related attachments are stored in this database. This database also stores events information for all participants during their evaluation activities for different purposes.

This database has all the pre-defined tables for existing scenario-based software architecture evaluation models (ATAM, SAAM, etc.). But in case of customized evaluation methodologies, there will be a number of user defined functionalities with

their own data table(s) requirements. So, based on the functionality implementation, the execution component creates tables at run-time to retain the findings of each step during the evaluation.

All the post-evaluation information is stored in the outcome database which is coupled with the post evaluation management component. This database contains outputs of each evaluation performed on the evaluation system. As a result, a user looking for previous evaluation outcomes to get useful insights interacts with this component and the post evaluation management component retrieves data from this database. Moreover, this database retains feedback about participants, submitted by peer participants, which will be forwarded to the evaluator management component to improve future selection of evaluation participants.

6.2 Implementation Details of Architectural Components of the System

This section describes some implementation details of the six components of the system. Each component's internal structure is described by a UML class diagram. Also, the main functionalities of the evaluation system are described as sequence diagrams. Each sequence diagram shows the sequence of process and interaction among different sub-components of the associated components.

Figure 6.1 shows the class diagram of the evaluation creation management component. The diagram consists of three classes, namely EvaluationCreationManager,

EvaluationDAO and EvaluationMethodologyManager. EvaluationMethodologyManager class is responsible for retrieving the list of available evaluation methodologies. EvaluationCreationManager class is responsible for validating evaluation profiles and storing all the details provided by evaluation creators to the evaluation database through the evaluation database access object. EvaluationDAO is responsible for connecting to the evaluation database to perform various database accesses.

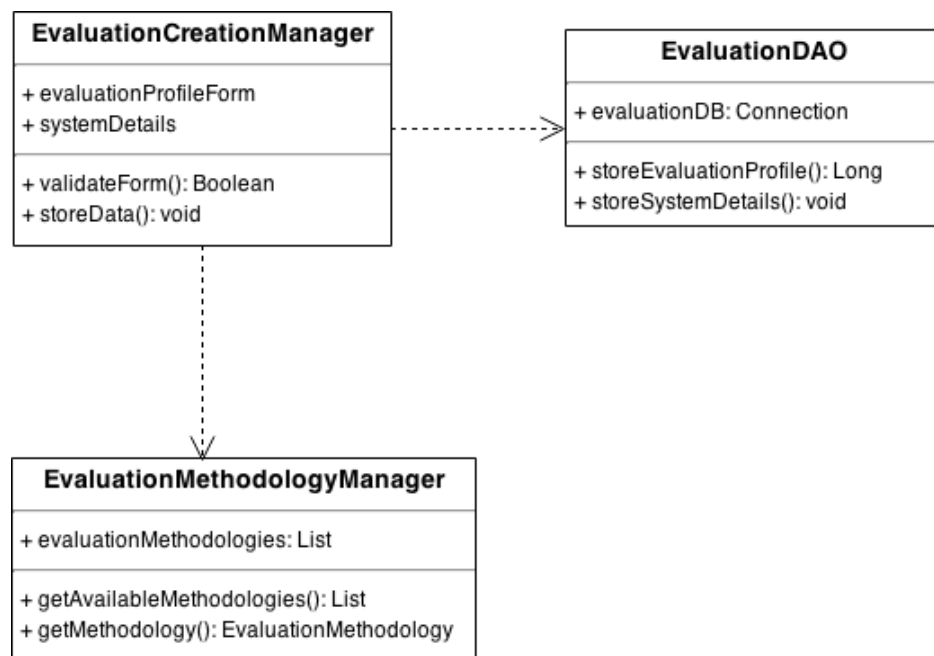


Figure 6.1 Class Diagram of Evaluation Creation Management

Figure 6.2 shows the sequence diagram of the evaluation creation process by the evaluation creation management component. After signing in, an evaluation creator fills necessary details and creates an evaluation profile. In order to associate an evaluation methodology, the evaluation creation manager class retrieves the list of available

evaluation methodologies from the evaluation methodology management component through the evaluation methodology manager. Before sending all the data to EvaluationDAO to store them into the evaluation database, the evaluation creation manager validates those details against the pre-specified business rules.

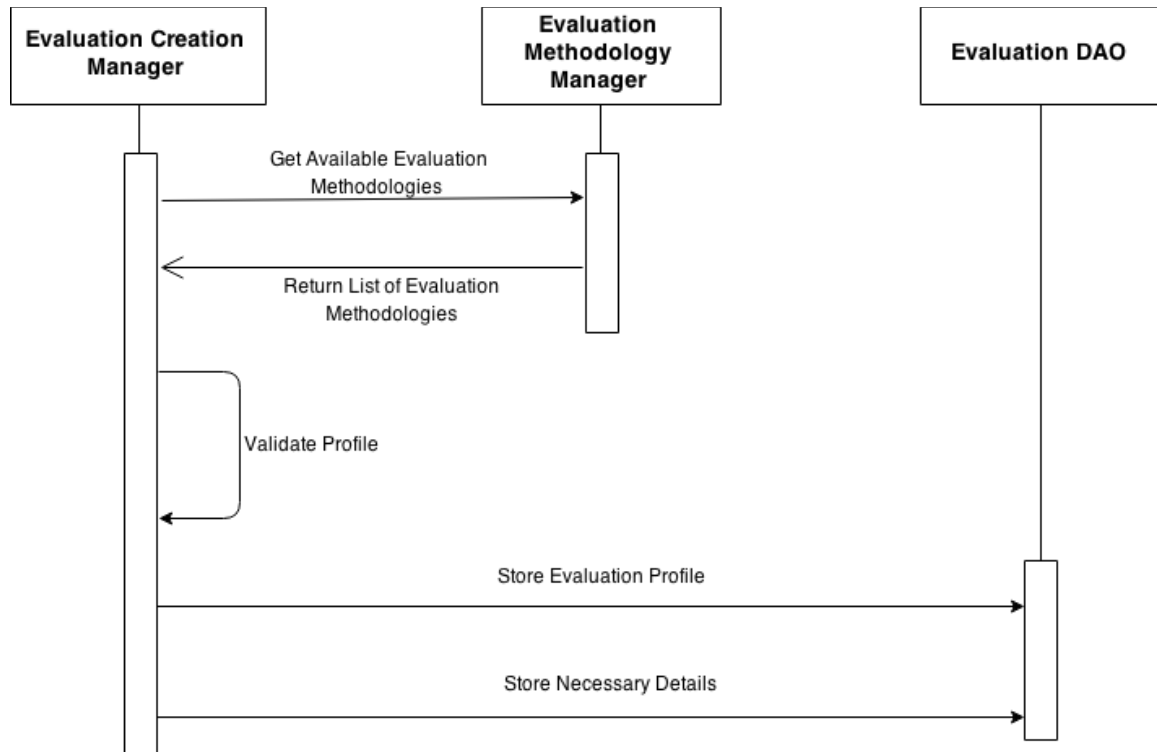


Figure 6.2 Sequence Diagram of Evaluation Creation

In Figure 6.5, a class diagram of the evaluation methodology management component is depicted. EvaluationMethodologyCreator class is responsible for managing all the operations and user interactions for generating customized evaluation methodologies. A methodology creator retrieves all the pre-written utilities from the evaluation methodology database and generates the specification for a new evaluation methodology.

Before storing the specification into the evaluation methodology database, compiler class compiles the specification against the specification grammar. This process is shown in figure 6.3, by the sequence diagram.

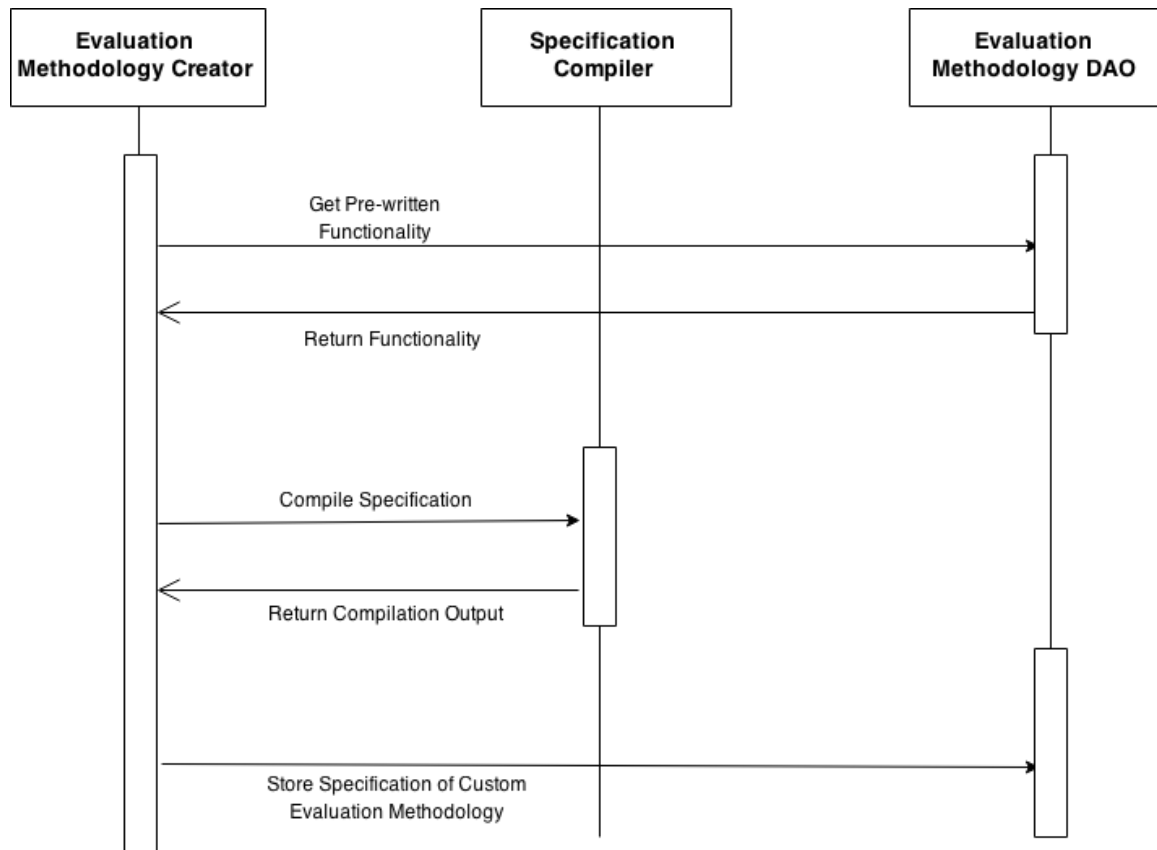


Figure 6.3 Sequence Diagram of Custom Evaluation Methodology Creation

Along with methodology creation, this component also configures and creates instances of the evaluation methodology execution engine to be used by the evaluation execution management component. EvaluationMethodologyManager class creates instance of Engine class, configured with an evaluation methodology and sends instances to evaluation execution management component. As shown in the figure 6.5, the engine

holds the references of all the pre-written functionalities with respect to the specification provided at the time of configuration. The sequence diagram in figure 6.4 shows such process.

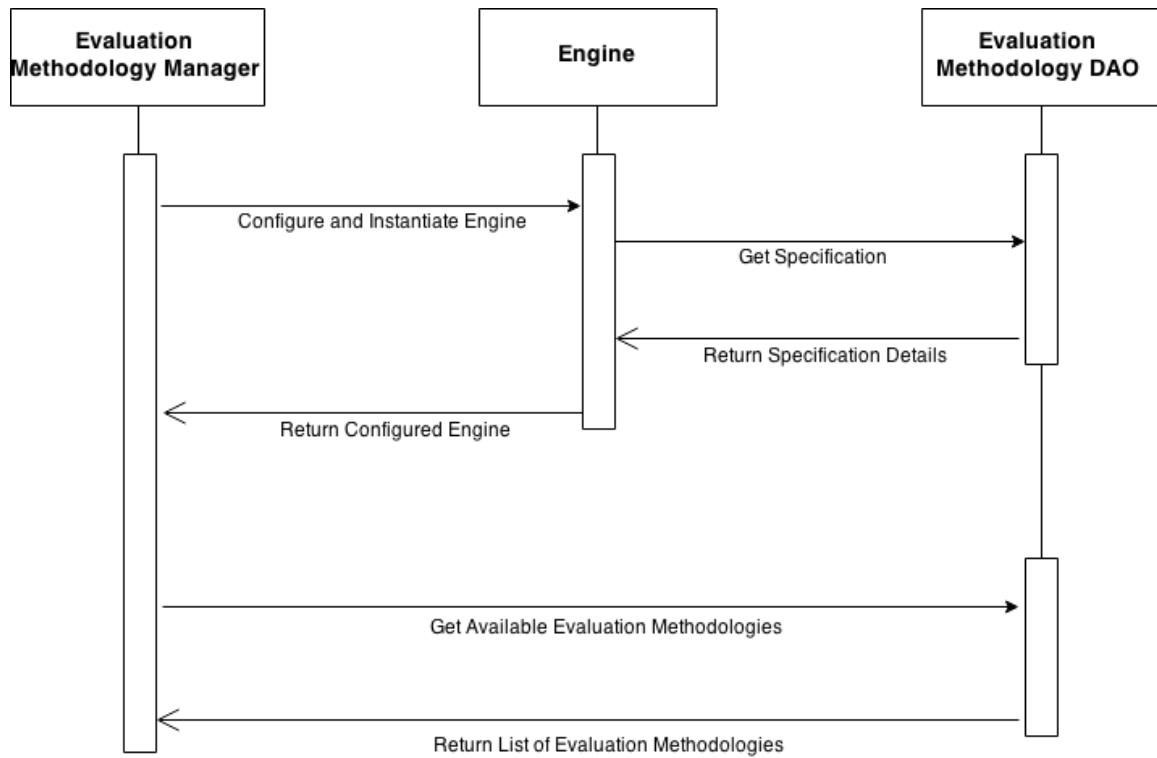


Figure 6.4 Sequence Diagram of Evaluation Methodology Execution

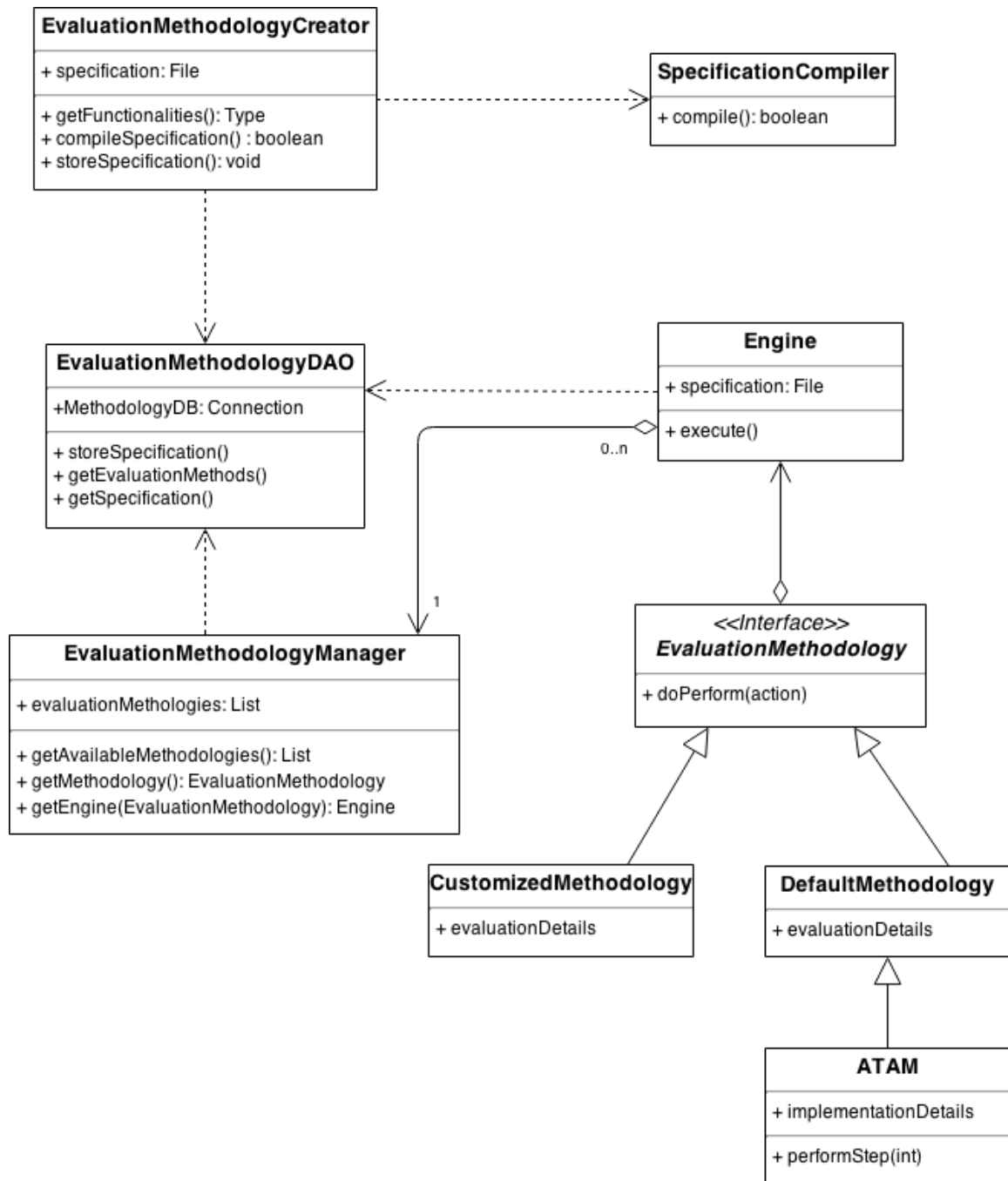


Figure 6.5 Class Diagram of Evaluation Methodology Management

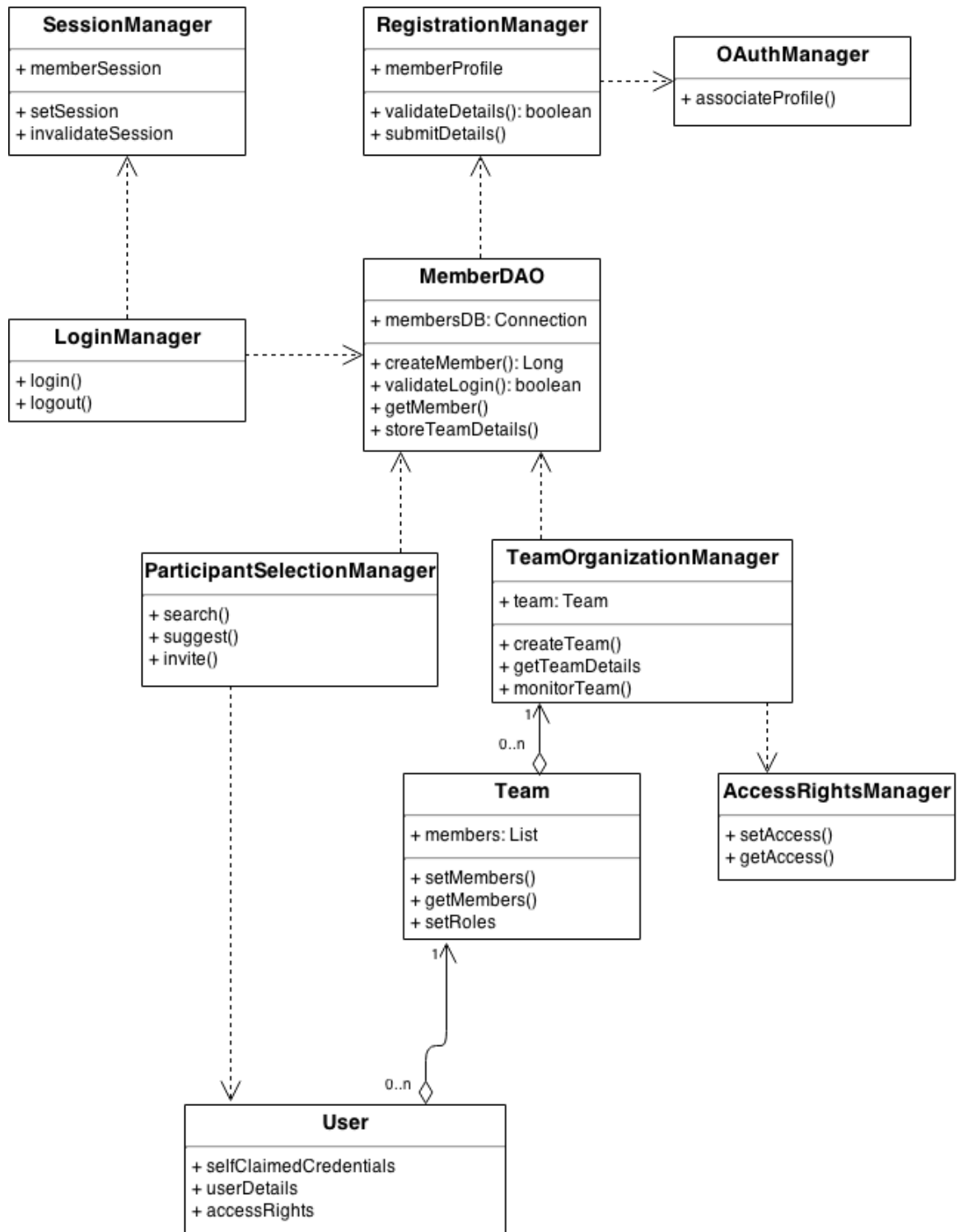


Figure 6.6 Class Diagrams of Evaluator Management and Team Organization Management

Figure 6.6 shows class designs of the evaluator management component and the team organization management component and figure 6.7 and 6.8 show the two main processes of these components. RegistrationManager class, after validating user profile, stores all the details of a new user to the member database through MemberDAO class. OAuthManager class is responsible for relating user profiles with other social networking services. A user then can use the same credentials to sign in to the evaluation system, which is managed by LoginManager class. After validating credentials, LoginManager class redirects the user to SessionManager class where a session of the signed user is created to manage subsequent activities from that user according to the user's role.

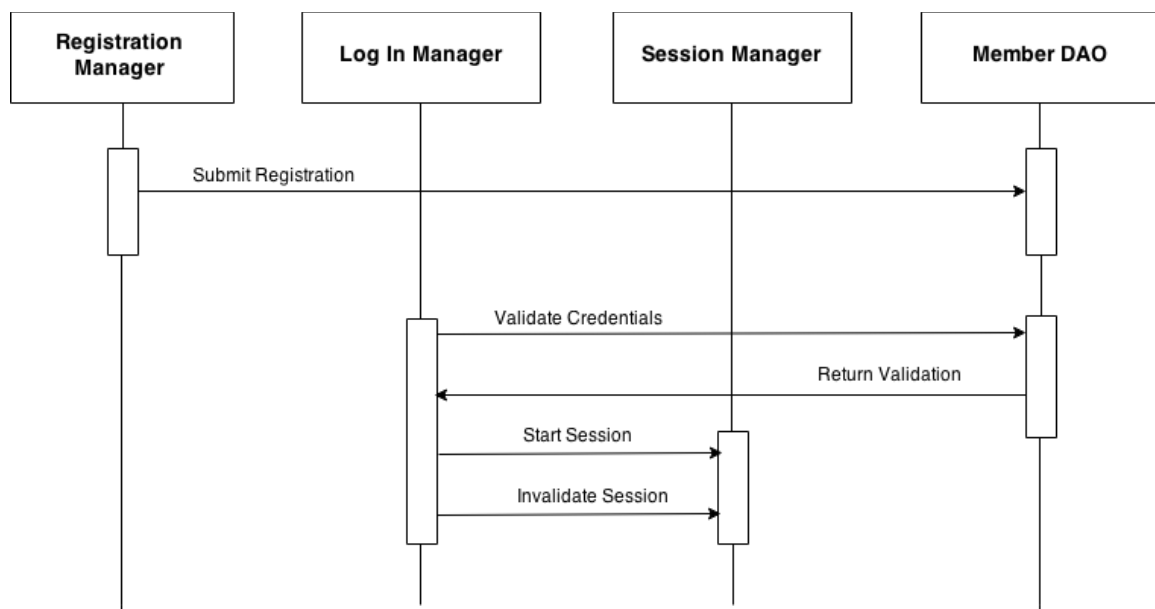


Figure 6.7 Sequence Diagram of User Sign in and Registration

Another important process is team formation and management, which is handled by ParticipantSelectionManager and TeamOrganizationManager. As shown in figure 6.7,

ParticipantSelectionManager lets an evaluation creator to search or invite other participants. Moreover, this class also provides suggestions to the evaluation creator based on users' credentials. After selecting participants, the evaluation leader grants access rights to each participant and add them to the team. TeamOrganizationManager is responsible for managing the team throughout the evaluation process. In case of large scale evaluation where multiple teams are evaluating same architecture, this class helps the evaluation creator to monitor progress of all teams at any given time.

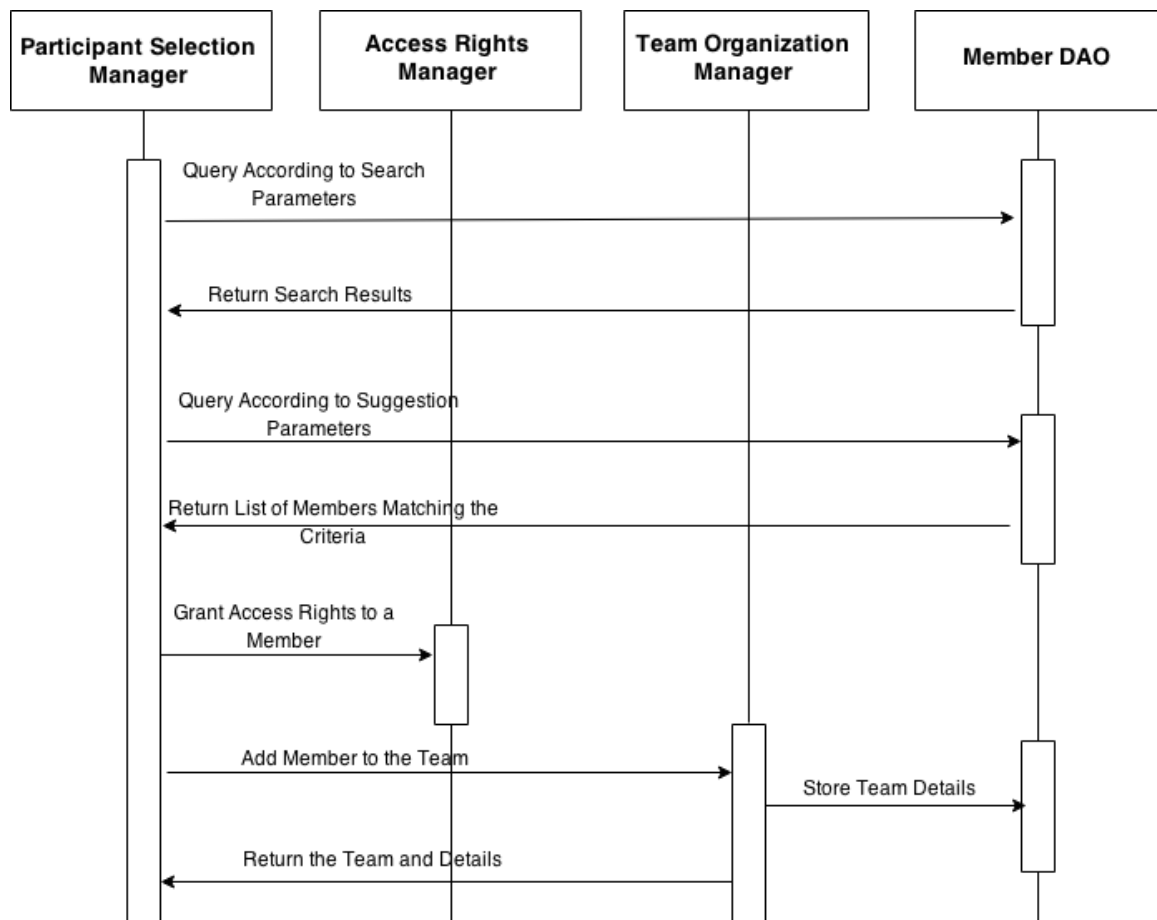


Figure 6.8 Sequence Diagram of Evaluation Team Formation

Figure 6.9 shows the class diagram of the evaluation execution management component. ExecutionManager class manages all the processes during the evaluation such as scheduling, monitoring and role/rights management. Further, this manager class uses Scheduler class to let the evaluation leader manage the schedule of synchronous software architecture evaluation. ExecutionManager uses OverseerManager for monitoring the evaluation and RoleManager to manage the roles of participants with respect to the evaluation. ExecutionManager retrieves the configured instance of the evaluation execution engine from the evaluation methodology management component and retains the instance into the execution database through EvaluationExecutionDAO. All of the aforementioned functionalities are depicted as a sequence diagram in figure 6.10.

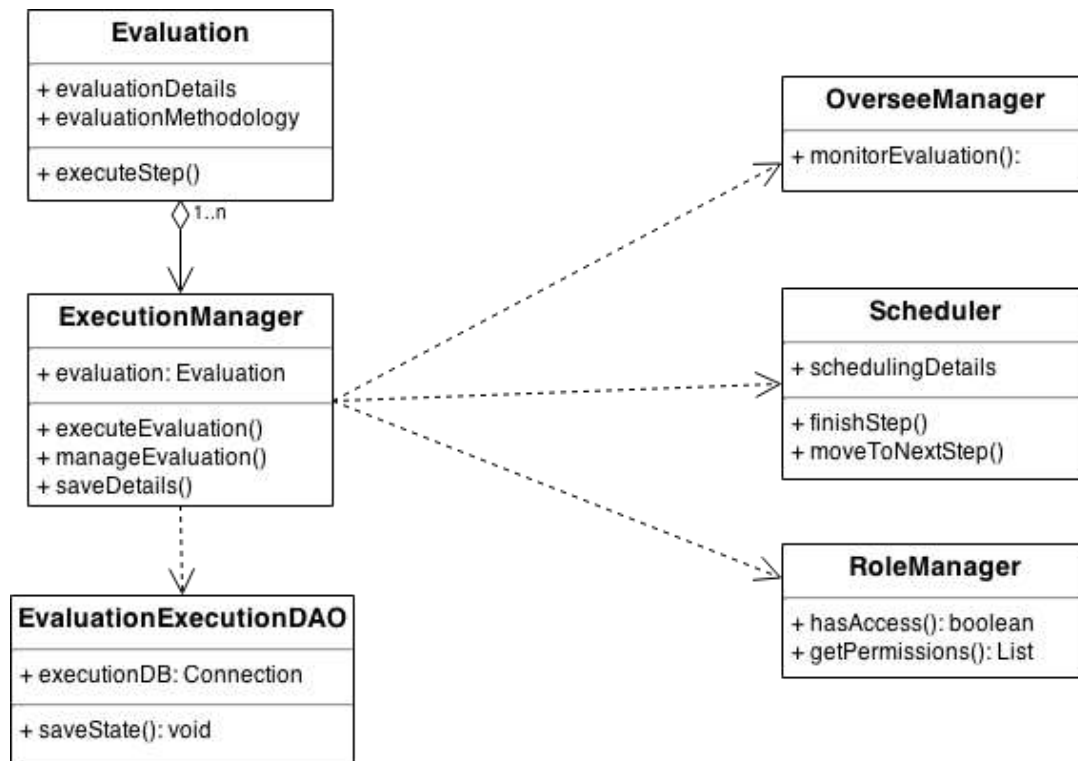


Figure 6.9 Class Diagram of Evaluation Execution Management

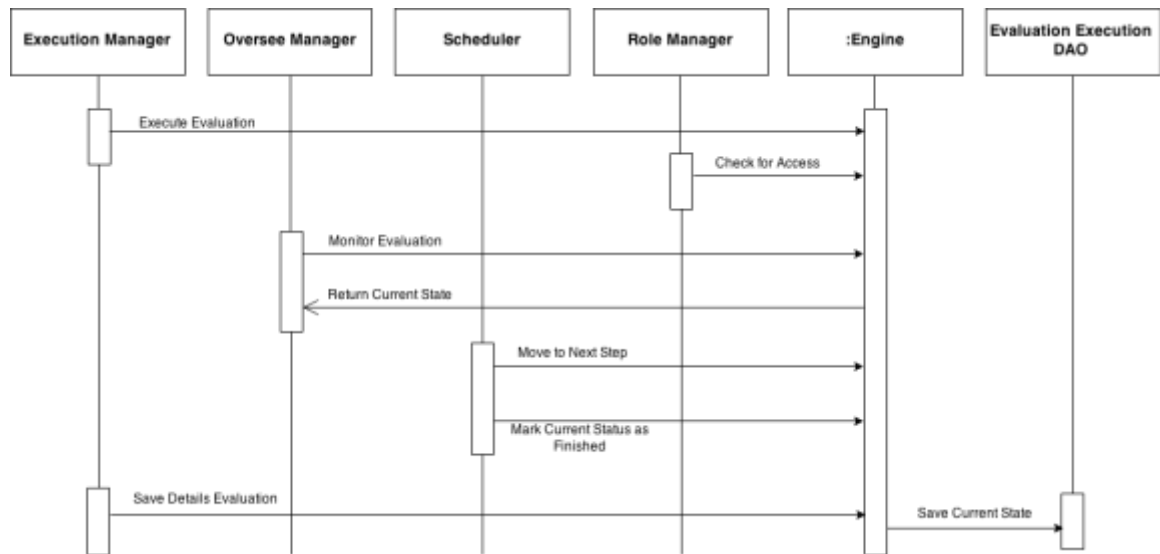


Figure 6.10 Sequence Diagram of Evaluation Execution Process

Figure 6.11 shows the class diagram of the post evaluation management component. The diagram consists of two manager classes i.e. feedback manager and output manager. The activities of the two managers include, providing feedback of peer team members after the evaluation and viewing, and comparing the outcomes of different evaluation processes. To insert, remove and retrieve the feedback and outcomes, OutputDAO (connected with output database) is used by both the management classes.

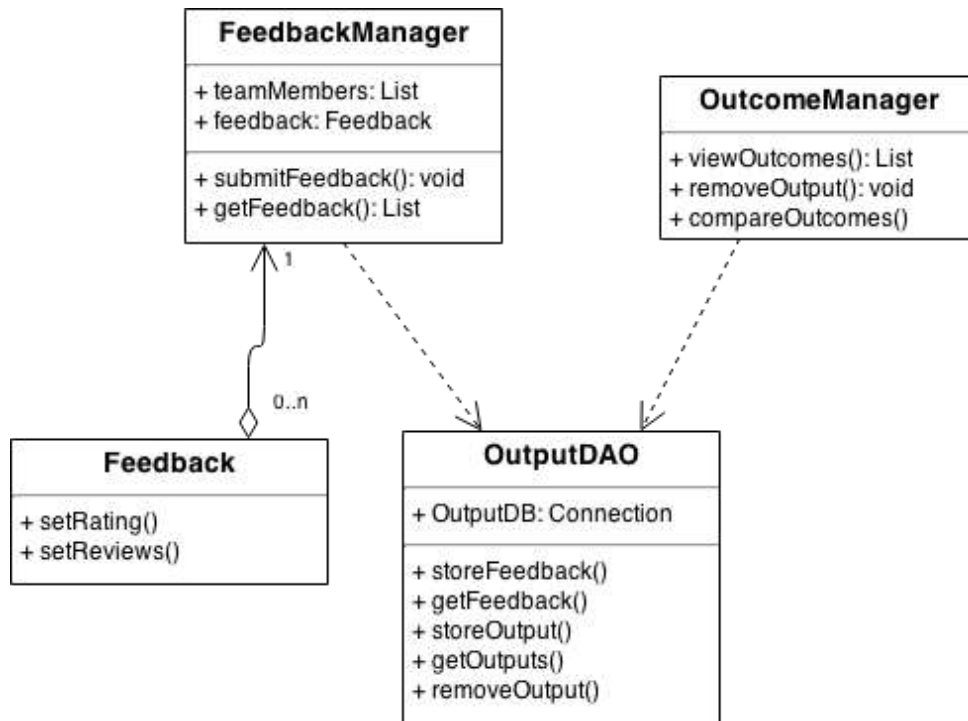


Figure 6.11 Class Diagram of Post Evaluation Management

Related to the previous class diagram in figure 6.11, figure 6.12 shows the sequence diagram of two activities performed by the component. Firstly, **OutputManager** is responsible for managing the output during and after the evaluation process. During the evaluation, the execution management component uses **OutputManager** to store the outcomes of various processes. The evaluation creator uses this class to view (and compare) outputs of different evaluation processes. **OutputManager** uses **OutputDAO** to access the output database in order to perform various retrieval, removal and insertion in the database. Secondly, at the end of an evaluation each user is asked to submit reviews on other team members by the feedback manager. The reviews are then stored in the output database through **OutputDAO**. The feedback can be retrieved by evaluation creators in future for new evaluation teams.

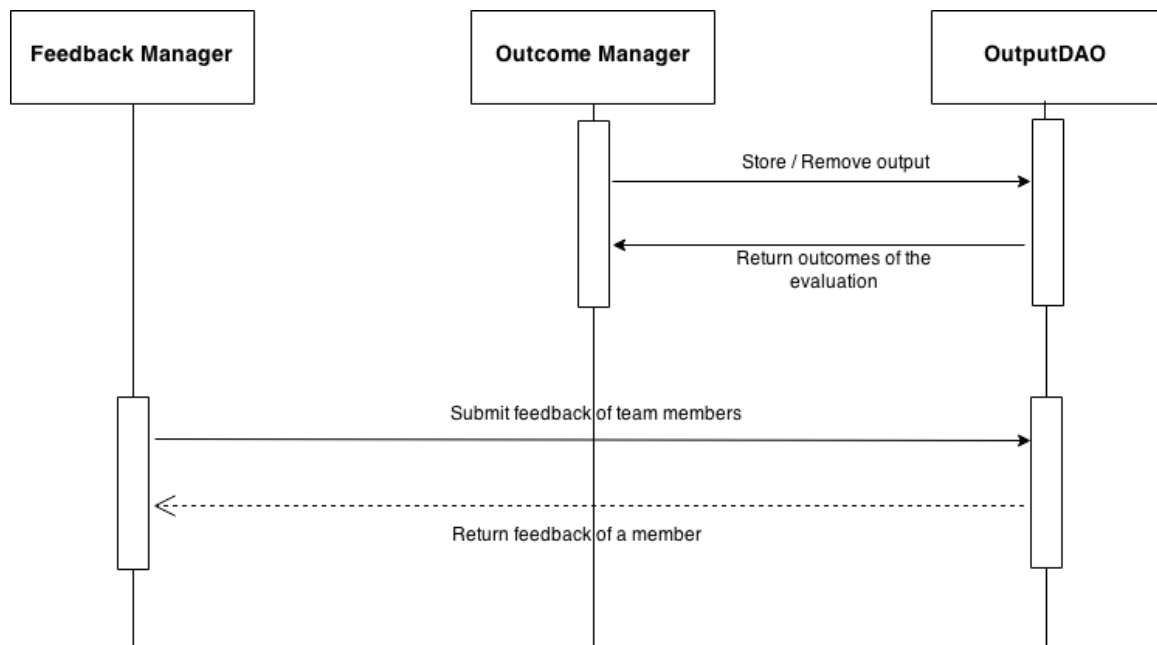


Figure 6.12 Sequence Diagram of Post Evaluation Activities

6.3 Summary

This chapter gives the detailed description on how the proposed evaluation system is implemented. The first part of the implementation details is technical overview. That section contains the details on various technologies used in order to make the system as a good collaborative system. Following to that, insights on the database are given. All the divisions of the database system are explained in detail. The implementation of all components of the software architecture evaluation system is further elaborated. The class diagram of each component consisting general classes and their relationships are described.

Chapter 7

Case Study

This chapter is about a case study of the proposed evaluation system. It describes evaluation of the architecture of the Nightingale health care system [31] through evaluation system from the beginning to the end. Firstly, member registration, evaluation creation and selected option (open call for evaluation) of team organization through the evaluation system are explained. Following to that, the evaluation process with detailed description on how each step of ATAM is performed on the system is explained. At last, all the post evaluation activities, independent from evaluation methodologies, through the evaluation system are explained.

7.1 Evaluation Organization

As it's already discussed, to use the proposed system each participant has to be a registered user of the system. So, while using the system for the first time, he signs up through the basic registration process. With this registration, he gets the login credentials for future use. All the details of the user are associated with him through his unique user identification.

Figure 7.1 shows the screenshot of the registration form containing various fields to help user create basic user profile. After this basic registration process, he has to fill in various details (self-claimed credentials) to enhance his profile in the system. These details

include, relevant experience, certifications, skills, expertise, and knowledge in any domain of the software development lifecycle specifically software architecture evaluation. This profile also contains reviews by other users with whom the user has participated with in the previous evaluations. More details are covered in sub-section 7.3.

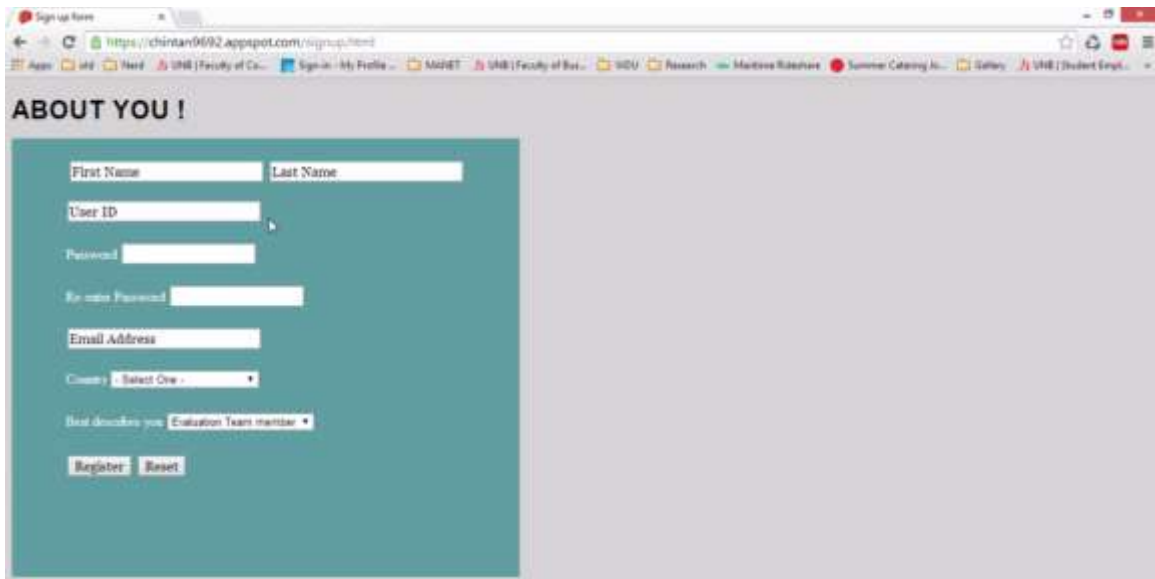
A screenshot of a web browser window displaying a signup form. The browser's address bar shows the URL 'https://chintan0692.appspot.com/signup.html'. The page has a title bar that says 'Sign up here'. The main heading of the form is 'ABOUT YOU !'. The form fields include: 'First Name' and 'Last Name' (text inputs), 'User ID' (text input), 'Password' (text input), 'Re-enter Password' (text input), 'Email Address' (text input), 'Country' (a dropdown menu with 'Select One' visible), and 'How do you know you?' (a dropdown menu with 'Evaluation Team member' selected). At the bottom of the form are two buttons: 'Register' and 'Reset'.

Figure 7.1 Signup on the System

In the evaluation of the Nightingale system, there are eight participants playing different roles. However, only five of them register in the beginning to the system through the aforementioned registration process. The remaining three will be invited by an evaluation creator, which is discussed later in this sub-section.

With respect to the case study, two users have relevant experience as evaluation team members that too with ATAM evaluation model. One user claims to be an architect and

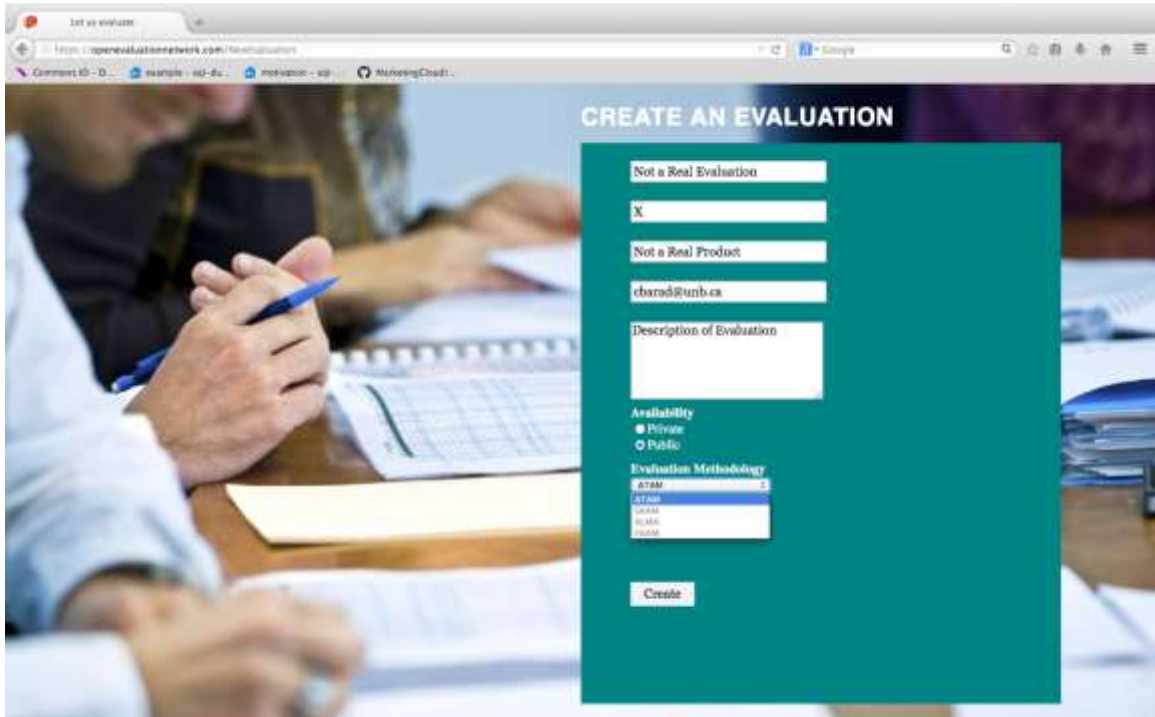
has associated LinkedIn profile backing up this claim. One user has some project management certifications. Two users are software developers and one is tester of the Nightingale system.

Having finished the registration and profile creation process, one of them chooses to be an evaluation creator. He then creates an evaluation by filling out necessary details as shown in the screenshot 7.2. These details contain evaluation profile, selected evaluation method, privacy of the evaluation and any of the three ways of team organization. After that, creator has to provide details of the software system to be evaluated. These details are comprised of architectural documentation, system documentation, architectural approaches etc.

Here, all the available evaluation methodologies are listed including existing and customized ones. Moreover, the system supports three ways to organize team(s) in the evaluation system: conventional (closed and invite only), open and crowdsourcing. To evaluate the Nightingale system's architecture, the evaluation creator selects open call for evaluation to organize an evaluation team for the Nightingale system.

In open type of team organization for the evaluation, creator creates the evaluation for the Nightingale software system with the necessary information pertaining to Nightingale system's architecture available. After that, creator uses the evaluation system's suggestion functionalities to get relevant suggestions of potential participants. Also, he invites some familiar people (not users of the evaluation system) through email. Based on

the previous experience with some participants, creator can collaborate with any of its connected participants again for the evaluation. In addition to that, some users request for participation and the evaluation creator accept some of the requests and give them the appropriate access rights.



The screenshot shows a web browser window with the URL <https://openevaluationnetwork.com/evaluation/new>. The page title is "CREATE AN EVALUATION". The form is set against a teal background and includes the following fields and options:

- ☐ Not a Real Evaluation
- ☒ X
- ☐ Not a Real Product
-
-
- Availability**
 - ☒ Private
 - ☐ Public
- Evaluation Methodology**
 - ☒ STAM
 - ☐ LTPM
 - ☐ NMA
 - ☐ NMA
 - ☐ NMA
-

Figure 7.2: Create an evaluation

He also makes the evaluation public so that the generated outcomes of the evaluation will be made available to the users of the evaluation system to get useful understanding about the proposed system. The provided details about the Nightingale system are given below. As its open call for evaluation, these details are available to the users of the evaluation system. So they can request for participation if the details are found relevant and

interesting. This information is also used by the system to suggest the evaluation to freelance evaluators, looking for appropriate evaluation to participate in.

Nightingale is large scale healthcare system software used by a large hospital chain throughout the southwestern United States [31]. The system serves as the information backbone and knowledge base for the health care institution customers. It stores, enriches and provides data about patients' treatment history as well as information about their insurance and other payments. It also provides a data-warehousing competence to help identify and analyze trends such as prediction of recurrence of certain diseases or viruses. The system produces a large number of reports according to the institution's specific requirements.

Nightingale system consists of two subsystems: OnLine Transaction Manager (OLTM) and Decision Support and Report Generation Manager (DSRGM). Nightingale system depends on commercial off-the-shelf (COTS) software, including the central data store, a rules engine, a work-flow engine, Common Object Request Broker Architecture (CORBA), a Web engine and a software distribution tool.

For the evaluation of Nightingale system, the evaluation creator got a suggestion of one evaluation team member who had a relevant experience. Several users requested for participation, but after careful analysis of their credentials, the evaluation creator chose one of them to be the evaluation team member. Evaluation creator can transfer the leadership to other evaluation team member through a normal role assignment, but he

chooses to be one in this evaluation. It is preferable that project manager and architect should have extensive knowledge about the system to be evaluated. So in this evaluation, both architect and project manager are not users of the evaluation system. Hence, evaluation creator sent them the email invite to register themselves on the system and ultimately participate in the evaluation. Fortunately, both the developers and a tester are already users of the system, so they voluntarily joined the evaluation.

At a time of adding a user to the evaluation, creator has to decide the role. In order to do that, he considers relevant experience, knowledge, skills and requested role with respect to the evaluation. In the evaluation of Nightingale system, the evaluation creator assigns different roles to each participant, as shown in the table below.

Member	Role
1 (Evaluation Creator)	Evaluation Team Leader
2	Evaluation Team Member - Scheduler
3	Evaluation Team Member - Scribe
3	Architect
4	Project Manager
5, 6	Software Developer
7	Software Tester

7.2 Evaluation Process

After successful registration, with the next login a user is redirected to his homepage as shown in the screenshot 7.3. This page is a main entry point to start different activities such as evaluation creation, evaluation methodology creation, participation in any ongoing evaluation and team organization for the evaluation. As shown in figure 7.3, the first item in the title bar leads a user to create a new software architecture evaluation. Create evaluation method redirects a user to the evaluation methodology management component for creating a specification of an evaluation method. For the given evaluation existing scenario based evaluation method ATAM is being used for the evaluation so, creator uses the existing implemented version of the evaluation model. However, ATAM is also defined using the proposed evaluation methodology specification. Please refer to appendix A for a complete specification of architecture trade-off analysis method.

Find participant menu lets a user add participants to the evaluation team. It has all the options required to form a team for any of the three types of team organization. The central area contains the list of all the evaluations the user is associated with. A user performs the activities according to his role by clicking and starting the evaluation. As a result, a user plays same and/or different roles in multiple evaluations at the same time. As shown in the screenshot, user “cbarad” is the creator and an evaluation team leader of the evaluation of Nightingale system’s architecture. So, he resumes his contribution with this evaluation by clicking on that tile.

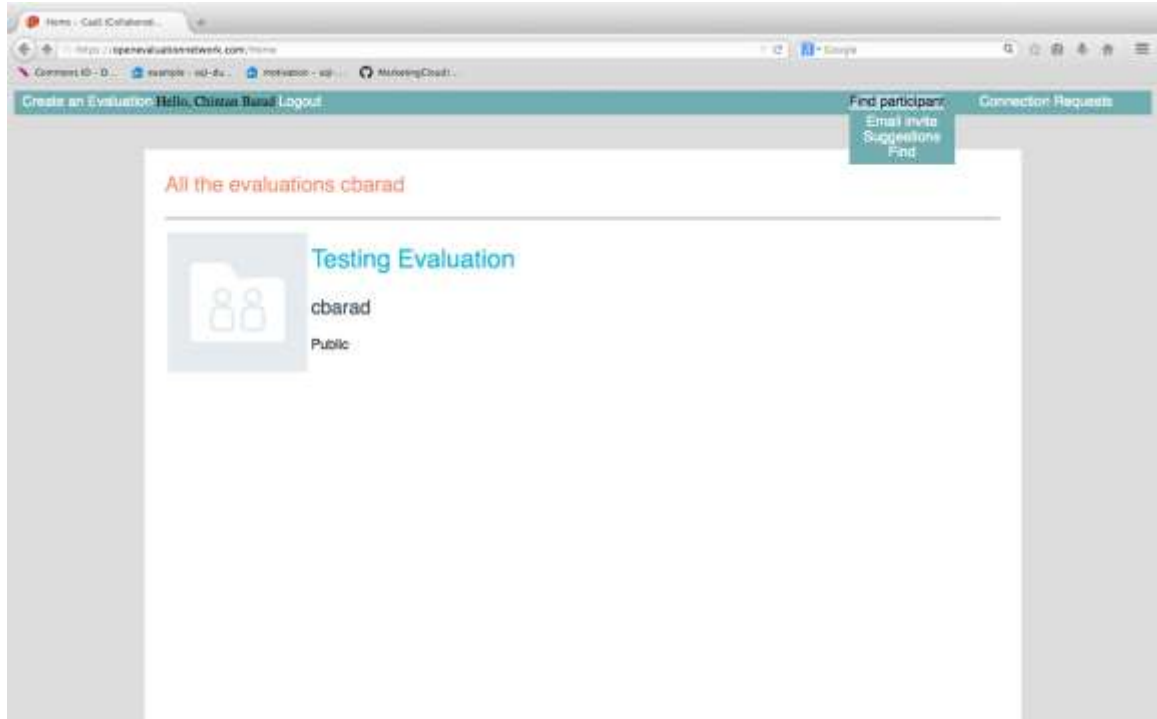


Figure 7.3 Homepage

After selecting the evaluation to participate in aforementioned way, a user is redirected to the main page of the respective evaluation. In the case study, a user is redirected to ATAM evaluation with the required session set according to his roles (and respective access rights). In each step, this role information set in session helps evaluation engine to execute a particular step. The remaining part of this sub-section discusses each of the nine steps of ATAM with respect to the Nightingale system's architecture evaluation.

Step 1, Present ATAM: First step is about presenting the general idea of ATAM to the team. In this step, the evaluation leader presents details about the evaluation methodology to the stakeholders in order to make them aware of the model they are going to follow. To

facilitate this step, the implemented evaluation model provides a functionality to upload a presentation in the form of Google slides, YouTube video or a simple Google document. As shown in the screenshot 7.4, an evaluation leader gets an access to upload the presentation for this step by filling in certain details. This presentation is made available to other team members to further discuss it by the evaluation leader through the evaluation system.

In the evaluation of the Nightingale system, the team leader prepares an hour long presentation laying out steps, phases, different foundations of ATAM and expected output from the evaluation process.

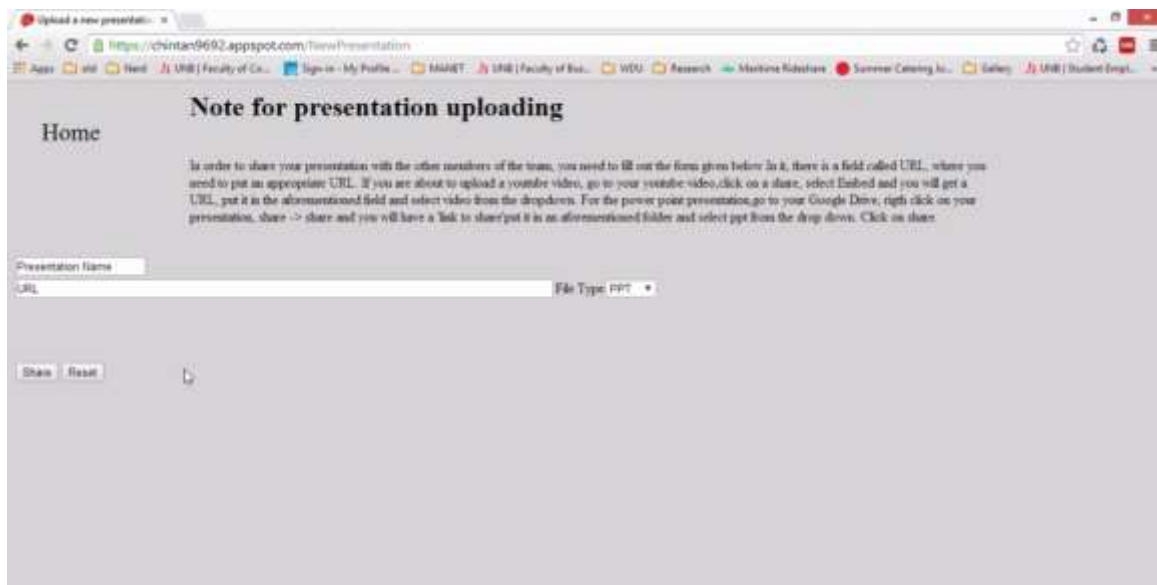
The screenshot shows a web browser window with the address bar displaying 'https://chintan9692.appspot.com/TeamPresentation'. The page has a header with 'Home' and 'Note for presentation uploading'. The note explains how to share a presentation with team members by filling out a form. Below the note, there is a form with a 'Presentation Name' field containing 'URL', a 'File Type' dropdown menu set to 'PPT', and 'Share' and 'Reset' buttons at the bottom.

Figure 7.4 Create a new presentation

Step 2, Present Business Drivers: This step is all about discussing business drivers of the software system to be evaluated. To support this step, features of presentation and related discussion forum provided are similar to the first step.

A project manager of Nightingale system presents business objectives, business requirements, business and technical constraints and quality attributes. Along with that, project manager of the Nightingale system enlisted and categorized all the business drivers, business constraints quality attributes with all the necessary details. They are given below.

Business requirements of the Nightingale system include creation of new version of the system to support their customers' diverse uses, ability to deal with diverse culture, regions and languages as fast as their current legacy system. Constraints for the system include no laying-off the existing employees, using off-the-shelf components as much as possible and finish the implementation of new system within two years. High priority quality attributes are performance, usability and maintainability while lower priority quality attributes are security, availability, scalability, modularity, testability and supportability. Note that the full list and description of each of these entities is given in [31].

Step 3, Present Architecture: In this step, an architect of the software system presents the architecture to the evaluation team and internal stakeholders (architect, project manager,

developers and testers). Likewise the previous two steps, features of presentation and related discussion forum are provided in this step, too.

For the evaluation of the Nightingale's system, some of the details were already made available by the evaluation creator during the evaluation creation as mentioned in subsection 7.1. Here the player (presenter) is the architect and audiences are evaluation team members and other internal stakeholders. The architect of the Nightingale system presents different architectural diagrams namely layered view of OnLine Transaction Manager (OLTM), a view showing communication, data flow, and processors of the OLTM, and data flow architectural view of the OLTM. The figure 7.5 shows the layered view of OLTM as presented by the architect during this step.

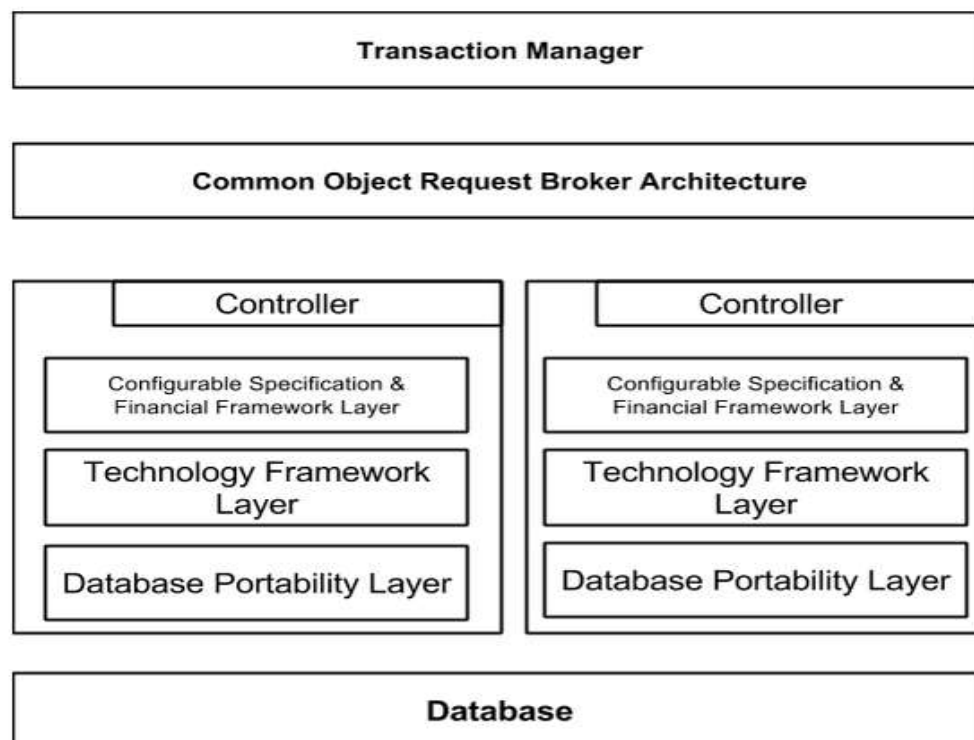


Figure 7.5 Layered view of Online Transaction Manager (OLTM)

With the third step, the first phase which is called presentation phase comes to an end. Note that each of the three steps was all about creating a presentation, presenting it to the team members and discussing various details through discussion forum. But, the presenter (player) in each step makes them distinct. At the end of it, presentations of ATAM, business drivers and architectural details of the Nightingale system are generated which can be viewed by a team member throughout the evaluation.

Step 4, Catalog architectural approaches: Having understood the architect of the system, in this step evaluation team enlists the architectural approaches they have heard, learned and understood from the architect. To support this step, functionalities of descriptive list making to enlist the architectural approaches and, a personal note making to make notes on each approach are provided. As shown in the screenshot given in figure 7.6, an evaluation team member (scribe) creates an architectural approach with the necessary details.

In the evaluation of the Nightingale system, the list of architectural approaches the evaluation team created in this step is as follows. OLTM layers, details about object orientation, use of configuration files, client server transaction processing and a data-centric architectural pattern.



Figure 7.6 Architectural Approach Creation

Step 5, Generate Quality Attribute Utility Tree: The quality factors that comprise system "utility" (performance, availability, security, modifiability, usability, etc.) are elicited, specified down to the level of scenarios, and prioritized. On the evaluation system, through the implemented ATAM evaluation model, project manager and evaluation team members together can create quality attributes; attribute refinements, scenarios by filling out necessary details in the given forms. Moreover, they can define priority criteria to associate with each scenario. The sample hierarchy along with its details is displayed in the form of a utility tree by the evaluation system. As shown in screenshot 7.7, a tree has utility as a root node; quality attributes on the second level, quality attribute refinements on the next level and scenarios as leaf nodes along with the associated priority criteria and level.

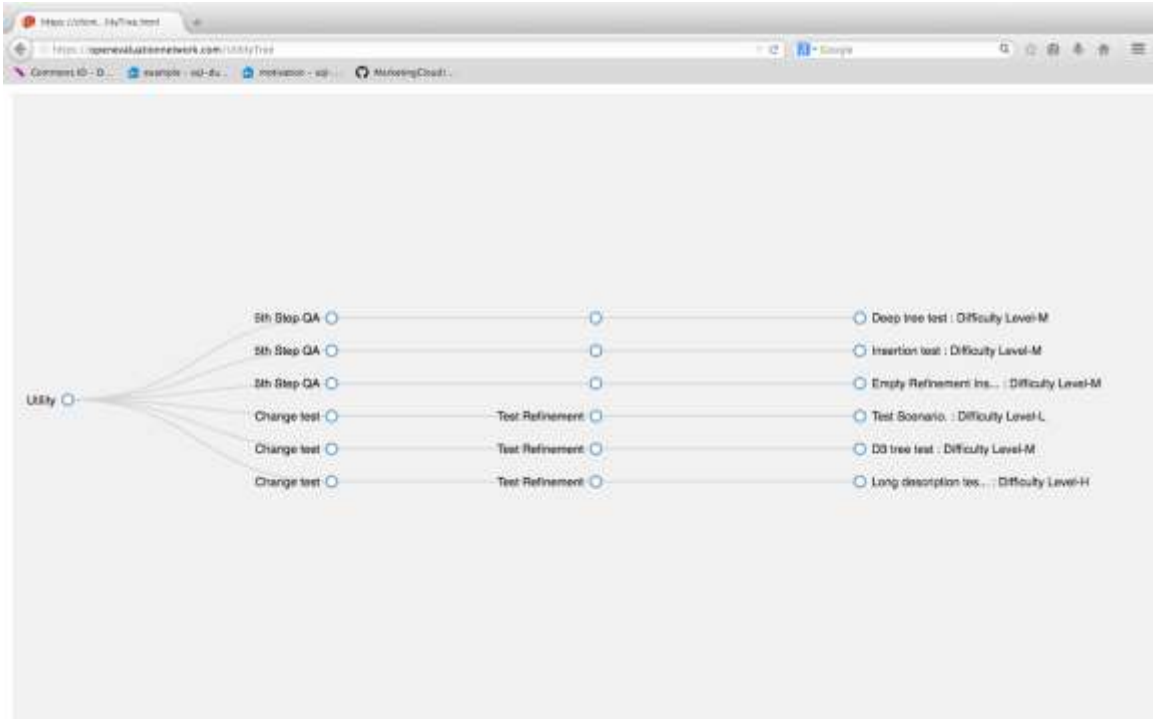


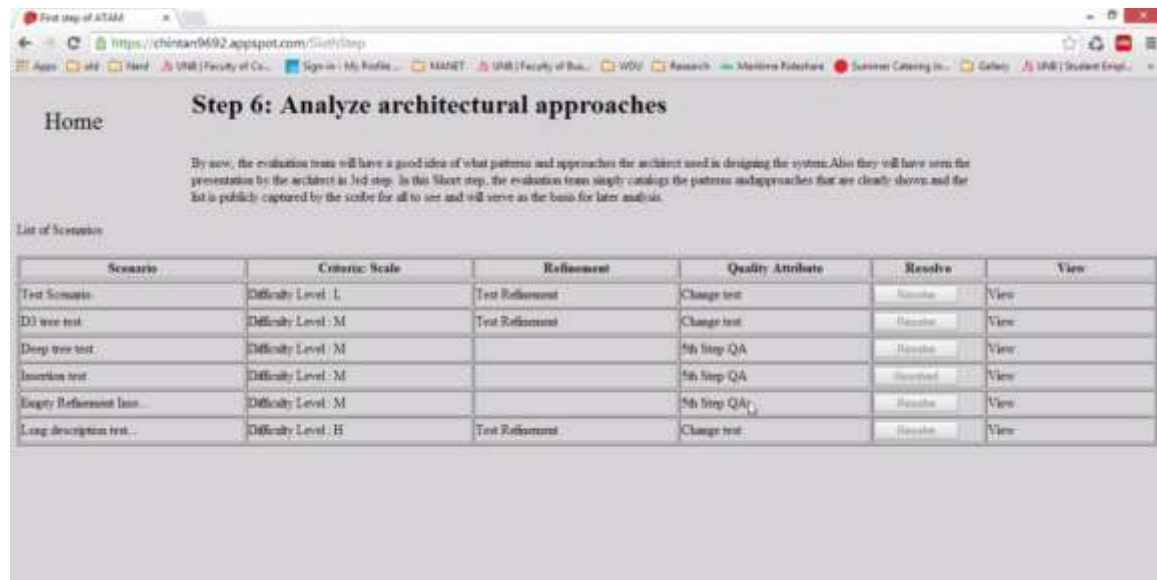
Figure 7.7 Quality Attribute Utility Tree

In the evaluation of the Nightingale system’s architecture, the participants refined the quality attributes generated in step 2 in a number of attributes and scenarios [31]. In order to prioritize them, they define two priority criteria namely importance and difficulty with the possible values of H, M, and L.

Step 6, Analyze Architectural Approaches: During this step, the architectural approaches that address the high-priority scenarios identified in Step 5, are analyzed. In this step, architectural risks, sensitivity points, and tradeoff points are identified. To facilitate this step, the analysis feature is provided on the evaluation system through which evaluation team members associate necessary details with all scenarios and share them with other team members. As shown in screenshot 7.8, an evaluation leader and architect gets a list

of sorted list of scenarios based on priority assigned in the previous step. Here, only the architect is given access rights to associate risks with each scenario and respective architectural changes. Evaluation team member can view the resolved scenario to get the useful insights and ask questions one may have with the provided resolution.

At this point, the evaluation team of the Nightingale system has recorded six sensitivity points, one tradeoff point, four risks, and five nonrisks [31]. When all the scenarios are resolved, the scheduler marks this step as complete and moves to the third phase starting with the seventh chapter.



Step 6: Analyze architectural approaches

By now, the evaluation team will have a good idea of what patterns and approaches the architect used in designing the system. Also they will have seen the presentation by the architect in 1st step. In this 6th step, the evaluation team simply catalogs the patterns and approaches that are clearly shown and the list is publicly captured by the system for all to see and will serve as the basis for later analysis.

List of Scenarios

Scenario	Criteria Scale	Refinement	Quality Attribute	Resolve	View
Test Scenario	Difficulty Level: L	Test Refinement	Change test	Resolve	View
DD tree test	Difficulty Level: M	Test Refinement	Change test	Resolve	View
Deep tree test	Difficulty Level: M		5th Step QA	Resolve	View
Insertion test	Difficulty Level: M		5th Step QA	Resolve	View
Empty Refinement tree	Difficulty Level: M		5th Step QA	Resolve	View
Long description test	Difficulty Level: H	Test Refinement	Change test	Resolve	View

Figure 7.8 List of Architectural Approaches

Step 7, Brainstorm and Prioritize Scenarios: With the involvement of developers and testers, a larger set of scenarios is elicited from them. This set of scenarios is prioritized

via a voting process involving all the participants except evaluation team members. The ATAM evaluation model contained in the evaluation system, helps participants to generate new scenarios, generate a voting mechanism, and prioritize the scenarios based on that mechanism. Note that appropriate access rights are given to each member according to one's role in the evaluation.

Evaluation team members have rights to create voting mechanism to be followed by other participants at the time of prioritizing the scenarios. The remaining participants get the access to create a new scenario, view existing scenarios (just like step 5) and vote to each of them. As per screenshot in figure 7.9, each developer and tester get a list of all the scenarios and to prioritize, they can use their total votes.

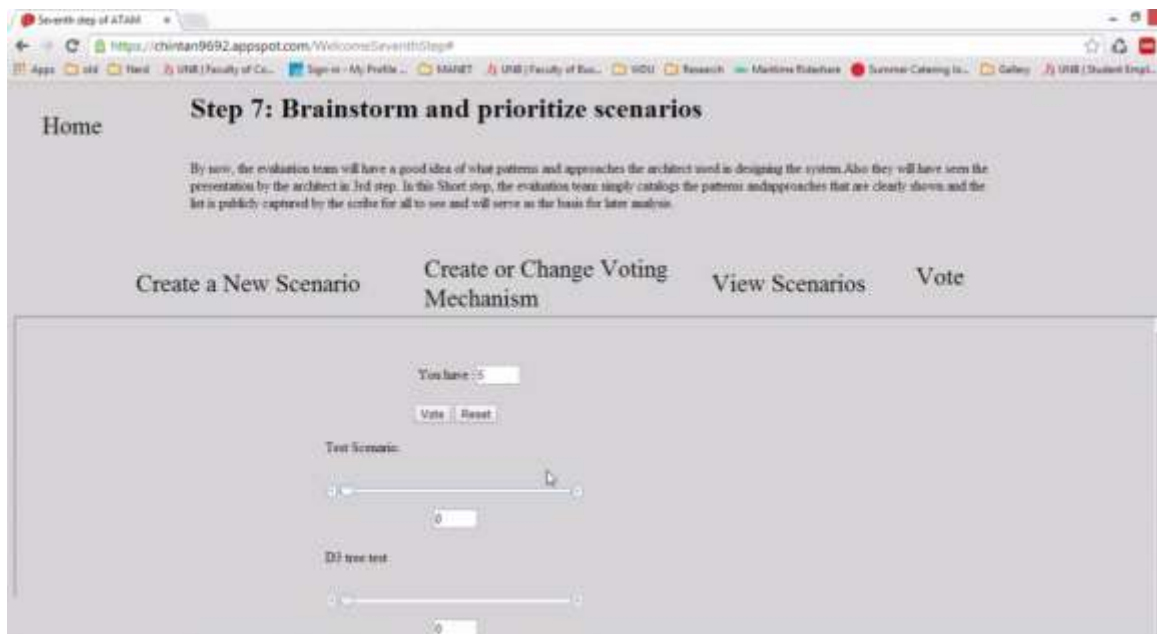


Figure 7.9 Voting and prioritizing scenarios

In the Nightingale evaluation, both the developers and tester are very much productive. They generate 22 scenarios in this step. Some of these scenarios are like, supporting German, data access is slow, payment response is slow sometimes etc. Note that the complete list of all the scenarios is given in [31]. Following to that, evaluation team creates a voting mechanism and assigns 20 votes to each stakeholder. These votes define the priority of each scenario, to be analyzed in step 8.

Step 8, Analyze Architectural Approaches: This step reiterates the activities of step 6, but this time using the highly ranked scenarios from step 7 instead of step 5. The reason behind analyzing them is that, this process may uncover more risks, sensitivity points, and tradeoff points, which can be also documented in the final output. The functionalities available for this step are similar to that of the step 6. Note that, this step depends on the output generated from step 7, hence cannot be started beforehand.

In the evaluation of the Nightingale system, the team analyzes additional seven highly prioritized scenarios one of which is as follows. Scenario states that the Nightingale is installed in a hospital and the hospital's existing database must be converted. The evaluation team noted one risk for not having the proper documentation for this transition and one non-risk about the architecture's support for a straightforward and effective data conversion and migration facility to support Nightingale installation

Step 9, Present Results: Based on the information collected in the ATAM (approaches, scenarios, attribute-specific questions, the utility tree, risks, non-risks, sensitivity points,

tradeoffs), the ATAM team presents the findings to the assembled stakeholders. The team can export these findings as a report for the future use. On the evaluation system, the same presentation and discussion features are provided in this step, too. Note that, here presenter is evaluation team leader and audiences are rest of the team members. As shown in figure 7.10, each participant of the evaluation can see the outcomes generated so far.

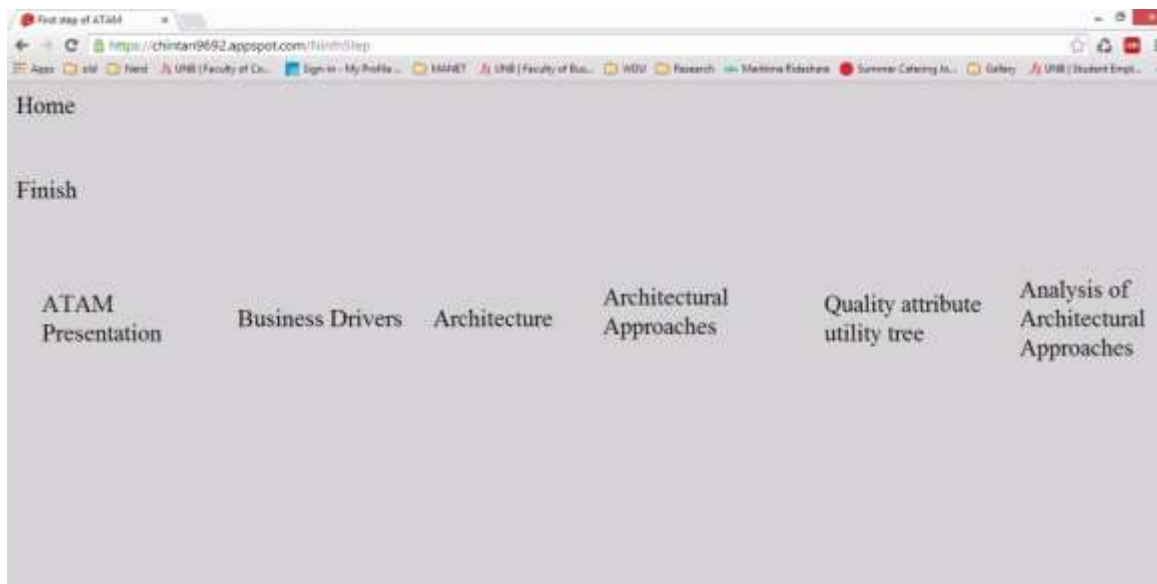


Figure 7.10 View evaluation outcomes

For Nightingale, the team identified three risk themes as follows. Over reliance on COTS products, lack of proper explanations about error recovery processes, and different documentation issues.

7.3 Post Evaluation Activities

After evaluation completion, the next phase contains after-activities of the evaluation. These activities mainly include handling and maintaining the outcomes generated throughout the evaluation process and reviewing peer participants with according to a user's experience throughout the evaluation.

With this case study, output of the evaluation of the Nightingale system by ATAM contains a list of risks, non-risks, sensitivity points, and tradeoff points. It also contains a catalog of architectural approaches used, the utility tree and brainstormed scenarios, and the record of analysis of each selected scenario. Since the evaluation creator has specified public output availability while creating the evaluation, these outcomes will be made available to all the users of the system.

If it were a crowdsourcing based evaluation, where a number of self-organized teams would evaluate the architecture of the Nightingale system without any involvement of evaluation creator, the creator would've shown all the outcomes generated by all the participating teams. He can then compare those outcomes and choose the best one or combination of several ones.

Participant Feedback

Please share your feedback of the peer participants, regarding the recent evaluation. We appreciate your candid responses.

	Not at all productive	A little productive	Neutral	Mostly productive	Extremely productive
How productive was User A ?	1	2	3	4	5

	Not at all productive	A little productive	Neutral	Mostly productive	Extremely productive
How productive was User B ?	1	2	3	4	5

	Not at all productive	A little productive	Neutral	Mostly productive	Extremely productive
How productive was User D ?	1	2	3	4	5

Please tell us how your overall experience and why you feel that way about this team.

Submit

[Report Abuse](#)

Figure 7.11 Participant Feedback Form

As shown in the screenshot given in figure 7.11, the feedback page lists all the peer participants with respect to the evaluation. An evaluation creator gets a list of remaining evaluation team members, project manager, architect, developers and tester. In order to write the reviews, a user gives rating to each and writes feedback in textual field. After submitting, these reviews are stored in the member database and used by the system to provide more sensible user suggestions. Also, these reviews are visible to other users, to help them determine the eligibility of the user for the evaluation. Note that, a user doesn't have the control over his reviews and its visibility.

Chapter 8

Conclusions

8.1 Summary

The research starts with proposal of concepts for openness and distribution in terms of software architecture evaluation. Moreover the requirements of the open and distributed evaluation system to realize these concepts are discussed. This is how the thesis first provides the fundamentals, based on which the remaining thesis elaborates different aspects of the proposed system.

Next, the system design is proposed, that contains six components: evaluation creation management, evaluation methodology management, team organization management, evaluator management, evaluation execution management, and post evaluation management. These components are discussed with their diagrams each of which contains the internal structure of the component, relations between parts and various use cases associated with them. Each of these components plays its role independently yet depending on the output from other components in given three types (conventional, open and crowd sourcing based) of evaluations and lets users evaluate their software system's architecture.

In addition to the existing scenario based evaluation models, the proposed system also supports the architecture evaluation by a customized evaluation methodology along with the ability to create and retain it. For that, the research proposes a new way of evaluation

methodology specification through XML markup called evaluation methodology specification. Further, the processes of creating and retaining customized evaluation methodologies are discussed with the detailed description of the evaluation methodology management component.

Following to that, technical aspects and implementation details of the system are also discussed. Here, details on the technologies and programming elements used in the development of the system are discussed. Furthermore, the database science used in the proposed web-based system and various database design details are discussed.

The prototype implementation of the evaluation system supports ATAM evaluation methodology. So, Nightingale system's architecture evaluation through the proposed system is explained with the real world statistics and necessary screenshots. All the phases of software architecture evaluation performed on the proposed system such as member registration, evaluation creation, team organization, ATAM evaluation process and post evaluation activities are discussed.

8.2 Evaluation

In this subsection, we discuss evaluation of the proposed open and distributed software architecture evaluation system. The system is evaluated against two criteria; firstly the system supports the standard software architecture evaluation processes, and secondly it

has extended features to support activities beyond conventional software architecture evaluation.

This research proposes an online system to support the process of software architecture evaluation. The system can fulfill the requirements for all the phases and activities of any-sized conventional software architecture evaluation processes, i.e. ATAM, SAAM etc., designed by CMU/SEI in online fashion. As already discussed, at a time of evaluation creation, the evaluation creator can select any of the standard scenario-based software architecture evaluation models and he can organize a closed team of group members which conforms the conventional (closed) evaluation in web-based environment.

The proposed system supports various activities beyond standard scenario-based software architecture evaluations. The proposed system is evaluated against the features provided to support such activities, such as multiple ways of team organization, various evaluation models (standard and customized), synchronous and asynchronous evaluation and post evaluation management. The detailed description on each feature based evaluation is given below.

- *Support different types of evaluation team organization:* The system supports two unique ways of team organization other than conventional (closed) evaluation: open (open call for) software architecture evaluation and fully open (crowdsourcing based) software architecture evaluation. In open software

architecture evaluation, the system lets the creator make necessary information about the prospective software system along with its architecture available openly. As a result, any registered user of the system can view the evaluation information and join the evaluation. In crowdsourcing based evaluation, the proposed system helps the evaluation creator in creation, management and governance of different self-organized evaluation teams. In both aforementioned types, the evaluation creator can accumulate as much appropriate participants as the conventional face-to-face evaluation methods, or even better. Since the evaluators around the world can collaborate, the evaluation quality may get improved more than that of conventional face-to-face evaluation methods.

- *Support for multiple standard evaluation methodologies:* Unlike ACE [13], the proposed system supports multiple standard evaluation methodologies. At a time of creation, the evaluation creator can decide from any of the standard evaluation methodologies (i.e. ATAM, SAAM etc.) according to his requirements and can follow the same steps as per specification and evaluate the architecture in open and distributed environment.
- *Support for user defined evaluation methodologies:* Beyond standard evaluation methods, the system supports software architecture evaluation by user defined evaluation methodologies, too. Moreover, the system supports rules and execution strategies for creation, possession and management of such customized evaluation methodologies.

- *Support for both synchronous and asynchronous evaluation processes:* The system lets each user to contribute to any number of different evaluations either synchronously (in traditional closed evaluations) or asynchronously (in open evaluations). While creating the evaluation, the evaluation creator can specify the type of evaluation i.e. synchronous or asynchronous. As a result, in synchronized software architecture evaluation each participant needs to contribute in each phase on pre-decided time and finish the duties in pre-specified duration. On the other hand, asynchronous evaluations are flexible in terms of time constraints and each participant can perform his duties according to his availability for a particular evaluation.
- *Support for post evaluation management and analysis:* Having done the evaluation in best suitable manner, the system provides different features for post evaluation management and analysis. These features include, gathering reviews about peer team members from each user to improve team organization, and providing output management facilities to view and compare outcomes of the evaluation process.

8.3 Contribution

The presented research work proposes a way to do the conventional scenario-based evaluation in distributed and asynchronous environment. In addition to that, the thesis

contributes open (open call for) software architecture evaluation and crowdsourcing based evaluation. Moreover, it also contributes XML based specification markup in order to create a scenario-based evaluation methodology that meets the requirements for the evaluation.

As mentioned in related work, available distributed collaborative environment does not follow any of the scenario-based models. We propose the evaluation system supporting all scenario-based methods along with user defined sequential methodologies in open and distributed environment. This system helps participants create an open collaborative network for software architecture evaluation to invite and involve the expertise that they want in the evaluation. In order to bring appropriate participants for open software architecture evaluation, the system provides various facilities such as smart suggestion, efficient search and email invitations.

Not the whole system but the prototype has been implemented as a cloud based web service covering following features. A prototypal evaluation system supports evaluation creation, team organization through search, suggestion and email based invitation to team members. It has ATAM evaluation methodology implemented and contained in it, supporting open and close evaluations in distributed environment. It supports participation in several evaluations at the same time and playing different or same roles in each of the evaluations.

In a nutshell, through this research work we contribute new ways of evaluation in open

and distributed environment, a way of creating a user defined evaluation methodology, an evaluation system to facilitate these evaluations and a working online software architecture evaluation web service (cloud based software as a service) to support software architecture evaluation by ATAM evaluation methodology.

8.4 Future Work

As mentioned in the previous section, not the whole system but a prototype has been implemented so, as a future work, a complete software architecture evaluation system in an open and distributed environment can be implemented. Moreover, the system can be designed and architecture can be enhanced in order to support all the openness and distribution concepts and to make a complete system to support all the phases of software architecture evaluation in as many open and distributed ways as possible.

In future, all the existing scenario-based evaluation methodologies should be implemented and contained in the evaluation system. This will help a wide community to create any number of evaluations using these methodologies to evaluate the architecture in open and/or closed environment. In addition to that, a functionalities pool can be made open for all the developers, so large number of utilities will be implemented. As a result, any user can create any number customized evaluation methodologies to evaluate the architecture of any type of software system in open and distributed environment. A repository of outcomes of all the public evaluations should be made available which can help evaluators improve the evaluations over the period of time.

Further, stronger algorithm can be implemented to improve the suggestion of participants and provide efficient way of suggesting appropriate participants for the evaluation. As a result, evaluation creator will end up having more useful software architecture evaluation and will be able to grow the collaboration network. In order to facilitate large scale evaluation, a smart system can be implemented to automate the self-organized teams' creation from the large pool of interested users. This system can take self-claimed credentials, past experience, previous participations and requested roles into account while creating balanced self-organized teams. These teams can then be modified by the evaluation creator. Moreover, to reduce the fake users and to remove the spams, some security and authenticity issues can be considered.

Bibliography

- [1] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., and Stafford, J. (2010). *Documenting Software Architectures: Views and Beyond*, Second Edition. Boston: Addison-Wesley. ISBN 0-321-55268-7
- [2] Cook, D., *Architecture Evaluation and Review Practices*, published in Skyscraper, an online resource provided by Microsoft, June 2007
- [3] Clements, P., Kazman, R., and Klein, M. *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley, 2002
- [4] Software Engineering Institute, *ATAM: Method for architecture evaluation*, (October, 2014) http://www.sei.cmu.edu/ata/ata_method.html
- [5] Kazman, R., Bass, L., Abowd, G., and Webb, M. *SAAM: A Method for Analyzing the Properties Software Architectures*, Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 1994, pp. 81-90.
- [6] Software Engineering Institute, *CBAM: Cost Benefit Analysis Method* (October, 2014) http://www.sei.cmu.edu/ata/products_services/cbam.html
- [7] Nico, L., Ph.D. Thesis, *Architecture-Level Modifiability Analysis*, University Amsterdam, February 2002
- [8] Bengtsson, P., Ph.D. Thesis, *Architecture- Level Modifiability Analysis*, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Sweden, 2002
- [9] Dolan, T. J., Ph.D. Thesis, *Architecture Assessment of Information-System Families*, Department of Technology Management, Eindhoven, University of Technology, February 2002

- [10] Ali Babar, M., Kitchenham, B., and Gorton, I., *Towards a Distributed Software Architecture Evaluation Process, a Preliminary Assessment*, Proceedings of the 28th international conference on Software engineering, May 2006
- [11] Ali Babar, M., Kitchenham, B., and Jeffery, R. *Distributed Versus Face-to-Face Meetings for Architecture Evaluation: A Controlled Experiment*. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering
- [12] Haynes S., and Singel, J., *Collaborative Architecture Design and Evaluation*. Proceedings of the 6th conference on Designing Interactive systems, June 2006
- [13] Maheshwari, P., and Teoh, A., *Supporting ATAM with a collaborative Web-based software architecture evaluation tool*, Science of Computer Programming (2005) pp. 109–128
- [14] Article published on self-organized teams (2014, November),
<http://www.infoq.com/articles/what-are-self-organising-teams>
- [15] Dooly, M., Lang, P., *Constructing Knowledge Together*, Extract from Telecollaborative Language Learning. A guidebook to moderating intercultural collaboration online (pp. 21-45)
- [16] Barish, G., *Building scalable and high-performance java web applications using J2EE technology*, published by Addison Wesley, ISBN – 0-201-72956-3
- [17] Krebs, R., *Architectural Concerns in Multi-tenant SaaS Applications* (2012)
- [18] LinkedIn developers API (November, 2014), <https://developer.linkedin.com/api>
- [19] The OAuth 1.0 Protocol ([RFC 5849](http://tools.ietf.org/html/rfc5849)) (November, 2014),
<http://tools.ietf.org/html/rfc5849>

- [20] OAuth and the Road to Hell (December, 2014), <http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/>
- [21] Software Engineering Institute, Architectural Trade-off Analysis Method, (December, 2014) <http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm>
- [22] Kazman, R., Bass L., Abowd, G., and Webb, M., *SAAM: A Method for Analyzing the Properties of Software Architectures*
http://resources.sei.cmu.edu/asset_files/WhitePaper/2007_019_001_29297.pdf
- [23] Google, Google App Engine (GAE) (February, 2015), <https://cloud.google.com/appengine/docs/whatisgoogleappengine>
- [24] Oracle, What is Java? (February, 2015), <https://www.oracle.com/java/get-started.html>
- [25] Dojo online documentation (February, 2015), <http://dojofoundation.org/>
- [26] Google, Introduction to V8 engine (February, 2015), <https://developers.google.com/v8/intro>
- [27] HTML, the web's core language (February, 2015), <http://www.w3.org/html/>
- [28] Cascading style sheets (March, 2015), <http://www.w3.org/Style/CSS>
- [29] Rich internet application (RIA), (March, 2015), http://en.wikipedia.org/wiki/Rich_Internet_application
- [30] Google, Google cloud SQL (March, 2015), <https://cloud.google.com/sql/docs>
- [31] Bass, L., Clements, P., and Kazman, R., *The ATAM: A Comprehensive Method for Architecture Evaluation*, Software Architecture in Practice

Appendix A

Evaluation Methodology Specification Schema and ATAM Specification

A.1 Evaluation methodology specification schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="evaluationMethodology">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="prerequisite">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="input">
                <xs:simpleContent>
                  <xs:complexType>
                    <xs:attribute name="name" type="xs:string" use="required"/>
                    <xs:attribute name="optional" type="xs:boolean" default="true"/>
                    <xs:attribute name="title" type="xs:string" use="required"/>
                    <xs:attribute name="type" type="xs:string"/>
                  </xs:complexType>
                </xs:simpleContent>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="roles">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="role">
          <xs:simpleContent>
            <xs:complexType>
              <xs:attribute name="category" type="xs:string" use="required"/>
              <xs:attribute name="name" type="xs:string" use="required"/>
              <xs:attribute name="type" type="xs:string" use="required"/>
            </xs:complexType>
          </xs:simpleContent>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="process">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="phase">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="step">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="task">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="functionality">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="player" type="xs:string"/>
                              <xs:element name="visible" type="xs:string"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="order" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="number" type="xs:string" use="required"/>
<xs:attribute name="duration" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

```

```

        </xs:sequence>
        <xs:attribute name="step" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="outcome">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="output">
                <xs:simpleContent>
                    <xs:complexType>
                        <xs:attribute name="step" type="xs:string" use="required"/>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                        <xs:attribute name="type" type="xs:string" use="required"/>
                        <xs:attribute name="title" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:simpleContent>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="availability" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

A.2 ATAM specification

```
<evaluationMethodology name="ATAM">

  <prerequisite>
    <input title="System Requirements" optional="false" type="textarea"
    name="requirements" >
    <input title="Architecture Diagram" optional="false" type="file" format=".jpg"
    name="architectureDiagram" >
    <input title="Architectural Description" optional="true" type="textarea"
    name="architectureDescription" >
  </prerequisite>

  <roles>
    <role category="Evaluation Team" name="Leader" type="evaluationLeader">
    <role category="Evaluation Team" name="Scenario Scribe"
    type="ScenarioScribe">
    <role category="Evaluation Team" name="Proceedings Scribe"
    type="proceedingsScribe">
    <role category="Evaluation Team" name="Timekeeper" type="timekeeper">
    <role category="Evaluation Team" name="Process Observer" type="observer">
    <role category="Evaluation Team" name="Process Enforcer" type="enforcer">
    <role category="Evaluation Team" name="Questioner" type="questioner">
    <role category="External Stakeholder" name="Project Manager"
    type="projectManager">
    <role category="External Stakeholder" name="System Admin"
    type="administrator">
    <role category="Internal Stakeholder" name="Architect" type="architect">
    <role category="Internal Stakeholder" name="Designer" type="designer">
    <role category="Internal Stakeholder" name="Tester" type="tester">
    <role category="Internal Stakeholder" name="Developer" type="developer">
  </roles>

  <process>
    <phase name="Presentation Phase">
    <step name="Present ATAM" number="1" duration="3">
    <task order="1">
    <functionality type="presentation">
    <player>Leader</player>
```

```

<visible>Internal Stakeholder, External Stakeholder</visible>
</functionality>
</task>
</step>

<step name="Present Business Driver" number="2" duration="5">
<task order="1">
<functionality type="presentation">
<player>Project Manager</player>
<visible>Evaluation Team</visible>
</functionality>
<functionality type="discussion">
<player="Evaluation Team, External Stakeholder"/>
</functionality>
</task>
</step>

<step name="Present Architecture" number="3" duration="5">
<task order="1">
<functionality type="presentation">
<player>Architect</player>
<visible>Evaluation Team</visible>
</functionality>
<functionality type="discussion">
<player>Architect, Evaluation Team </player>
</functionality>
</task>
</step>

</phase>

<phase name="Investigation and Analysis Phase">
<step name="Identify Architectural Approaches" number="4" duration="7">
<task order="1">
<functionality type="listCreation">
<player>Architect</player>
<visible>Evaluation Team</visible>

```

```

</functionality>
</task>
</step>

<step name="Generate Quality Attribute Utility Tree" number="5" duration="7">
<task order="1">
<functionality type="treeGeneration">
<player>Evaluation Team, Project Manager</player>
</functionality>
</task>
<task order="2">
<functionality type="voting">
<player>Evaluation Team, Project Manager</player>
</functionality>
</task>
</step>

<step name="Analyze Architectural Approaches" number="6" duration="6">
<task order="1">
<functionality type="analysis" argument="Architecture elicitation">
<player>Architect, Evaluation Team</player>
</functionality>
</task>
</step>

</phase>

<phase name="Testing Phase">

<step name="Brainstorm and prioritize scenarios" number="7" duration="6">
<task order="1">
<functionality type="generateScenarios">
<player>Internal Stakeholder</player>
</functionality>
<functionality type="brainstorm">
<player>Internal Stakeholder</player>

```

```

</functionality>
</task>
<task order="2">
<functionality type="voting">
<player>Internal Stakeholder, Evaluation Team</player>
</functionality>
</task>
</step>

<step name="Reanalyze Architectural Approaches" number="8" duration="4">
<task order="1">
<functionality type="analysis" argument="Architecture elicitation">
<player>Architect, Evaluation Team</player>
</functionality>
</task>
</step>
</phase>

<phase name="Reporting Phase">
<step name="Present Results" number="9" duration="3">
<task order="1">
<functionality type="presentation">
<player>Evaluation Team</player>
<visible>Internal Stakeholder, External Stakeholder</visible>
</functionality>
</task>
</step>
</phase>

<outcome availability="public">
<output name="Architecture elicitation" type="text" step="6,8">
<output name="Risks" type="list" step="6">
<output name="Utility Tree" type="tree" step="6">
<output name="Tradeoff Points" type="text" step="6">
<output name="Quality Scenarios" type="text" step="7">
</outcome>
<evaluationMethodology/>

```

Appendix B

Database Design

Evaluation database

Evaluation
(
evaluation_id,
evaluation_profile_id,
evaluated_system_details_id
)

EvaluationProfile
(
evaluation_profile_id,
evaluationName,
organizationName,
creator_id,
description,
availability,
evaluation_methodology_id,
team_organization_type_id,
teamSizeLimit
)

EvaluatedSystemDetails
(
evaluated_system_details_id,
architectural_diagram_ids,
system_details_id,
systemName,
systemDescription,
systemUsage
)

ArchitecturalDetails
(
architectural_details_id,
architectural_diagram_id,
diagramName,
diagramDescription,
diagramImage
)

```
SystemDetails
(
  system_details_id,
  detailName,
  detail
)
```

Member database

```
RegisteredUser
(
  user_id,
  name,
  username,
  password,
  email,
  security_question,
  security_answer,
  external_profile_ids,
  organization,
  avatarURL
)
```

```
UserExternalProfile
(
  external_profile_id,
  user_id,
  network_type,
  profile_url
)
```

```
UserProfile
(
  credential_id,
  user_id,
  credentialName,
  details
)
```

UserFeedback

```
(  
feedback_id,  
user_id,  
evaluation_id,  
role,  
rating,  
comments  
)
```

TeamOrganizationTypes

```
(  
team_organization_type_id,  
name,  
description  
)
```

Teams

```
(  
team_id,  
creator_id,  
participant_ids,  
name,  
creationDateTime,  
evaluation_id  
)
```

Participation

```
(  
participation_id,  
user_id,  
team_id,  
role_id  
)
```

Roles

```
(  
role_id,  
roleName,  
type,  
description,  
evaluation_id,  
functionality_ids  
)
```

```
UserConnection
(
  user_connection_id,
  first_user_id,
  second_user_id,
  connectionDateTime
)
```

Methodology database

```
EvaluationMethodology
(
  evaluation_methodology_id,
  name,
  description,
  specification_id
)
```

```
Specification
(
  specification_id,
  name,
  description,
  creationDateTime,
  compilationDate,
  specification_file
)
```

```
Functionalities
(
  functionality_id,
  name,
  creator_id,
  creator_name,
  creationDateTime,
  executableFile
)
```

Execution database

```
EventTypes  
(  
  event_type_id,  
  eventTypeName,  
  description  
)
```

```
EventDatanase  
(  
  event_id,  
  actor_id,  
  eventDateTime,  
  evalaution_id,  
  event_type_id  
)
```

Note: As already discussed in the chapter sixth, this database is dynamic in nature and contains dynamically generated tables during evaluation. The set of tables are generated for any particular evaluation to retain the information about findings generated in each step. These tables depend on the functionalities used in the evaluation methodology.

PostEvaluation database

```
EvaluationOutput  
(  
  evaluation_output_id,  
  outcome_ids,  
  evaluation_id,  
  availability,  
  contributor_ids  
)
```

```
Outcomes  
(  
  outcome_id,  
  outcome_name,  
  outcome_type,  
  evaluation_id,  
  generatedStepNumber,  
)
```

```
Feedback
(  
  feedback_id,  
  user_id,  
  evaluation_id,  
  role_id,  
  rating,  
  extraComments  
)
```

Curriculum Vitae

Candidate's full name: Chintan Vijaykumar Barad

Universities attended:

Bachelor of Computer Engineering, Gujarat Technological University, India

Master of Computer Science, University of New Brunswick, Canada

Publications:

Framework for open and distributed software architecture evaluation,

23rd International Conference on Software Engineering and Data Engineering (SEDE 2014), Held in October 2014 at New Orleans, LA, USA. In the proceeding of ISCA (ISBN 978-1-880843-96-3)