Chapter 3

Architectural Analysis
Chapter 3 Architectural Analysis

Architectural analysis consists of the following context:-

i. Architectural Requirements

ii. Quality Attributes

iii. Analyzing Software Architecture

iv. Performing Architecture Assessment

3.1 Introduction

In this 21st century, the computational way has dramatically changed for the past 10 years. New technology has been a strong driving force in revolutionizing the business being done. As the size and complexity of software systems increases, the design problems goes beyond the algorithms and data structures of the computation for instance designing and specifying the overall systems structure.

There is considerable work including module interconnection languages, templates and frameworks for systems that serve the needs of specific domains and formal models of component integration mechanisms. Thus, software architecture under the level of design comes into the picture. Software architecture expresses how systems should be built from various components and how those components should interact.

Therefore, it is very vital to recognize common paradigms so that high-level relationships among systems can be understood and new systems can be built as variation to new systems. The right architecture is crucial to the success of a software system design. Detailed understanding of software architecture allows the engineer to make principled
choices among design alternatives. Hence, the architectural system representation is often essential to the analysis and description of the high-level properties of a complex system [6].

In business context, software architecture is no longer an enabler of business success but it is an essence of competing in the hyper-competitive information age due to it acts as an mechanism or catalyst for creating and recreating customer value. Architecture must be positioned so that the business can use it to become a hyper-competitive predator. Software architecture must be positioned so that the business will prevail over its competitors. Software architecture must enable an organization to move swiftly and decisively.

3.2 Architectural Requirements

A requirement is a feature of the system or a description of something that the system is capable of doing in order to fulfill the system's objectives. It describes not only the flow of information to and from the system but also the constraints on the system's performance. Requirement elicitation is an especially critical part of the process. Variety of techniques is used to determine user’s need and customer’s want. Requirement identifies the ‘what’ of the system and the design identifies the ‘how’ of the system.

Requirement elicitation enables to explain the requirement definition of the system. Requirements definition is a complete listing of everything the customer expects the proposed system to do. It represents an understanding between customer and developer of what the customer need or wants and it usually written jointly with developer. On the other hand, the requirement specification restates the requirement definition in technical terms appropriate for the development of a system design. It is the technical counterpart to the requirement definition document where is written by requirement analysts [11].
In software architecture, requirements can be subdivided into those related to the functions of the systems and those related to the architecture. In a domain, requirement must be suitable to a product line. Architectural requirements derive from one of three sources: the quality goals for the system, or the business goals for the system, or the business goals for the people who will work on the system. Steps in defining architectural requirements are as follows:

i. To identify the architectural drivers.

ii. To enumerate the architectural requirements.

The requirements are an enumeration of the consequences of the architectural drivers and bring in other important architectural requirements.

As overall, architectural requirements are created by developing organization and are influenced by the technical environment and system architect’s own experience. There are three outputs from this process: an enumeration of functional requirements made concrete by use cases, an enumeration of specific architectural requirements and an enumeration of a collection of quality scenarios that contribute concrete tests for the architectural requirements. Figure 3-1 shows eliciting the architectural requirements.
3.3 Conceptual Architecture View

There are three phases in the conceptual architecture views, which are global analysis, central design tasks and final design task. Global analysis reviews the product requirements, uses cases and system requirements and their interactions. Besides, it analyzes the technological and organizational factors.

In the central design tasks, strategies used during the global analysis, is acting as a guide to design decisions and to improve the ability of the system in respond to change. The components and connectors are identified in order to build and construct the system. Map the system functionality to the components and connectors focusing on functional behavior and the control aspects. And define the instances of the component and connector types.

The results from the central design tasks are utilized to fed into the final design task, which is the resource budgeting. The available resources are assigned to configuration and budget is refined. The initial budget will assist to trace the resource problems. These constraints are then inputted to the central design tasks. Figure 3-2 shows the conceptual architecture view.
3.4 Quality Attributes

In the Architecture Business Cycle (ABC), business considerations determine qualities that must be accommodated in the system’s architecture. Quality and functionality of the system always interrelated. The mapping of a system’s functionality into software structures determines the architecture’s support for qualities.

3.4.1 Architectures and Quality Attributes

Quality must be considered at all phases of design, implementation and deployment. However, different qualities manifest themselves differently during these phases. Many aspects of the quality of usability are not architectural because they are almost always encapsulated within a single component.

In any discussion of quality attributes, it is vital to remember that they exist within complex systems and their satisfaction can never by achieved in isolation. The achievement of any quality attribute will create an effect sometimes positive or negative. Listed below are the quality attributes discernable at runtime:

- **Performance.** Performance refers to responsiveness of the system, which is the time required to response to stimuli or event in some interval time. Performance qualities are expressed by the number of transaction per unit time or by the amount of time taken for a transaction with the system to complete. Therefore, performance is often a function on how much communication and interaction occur among components in the system. In the
history of software engineering, performance has been the driving force in system architecture.

- **Security.** Security is the measure of the system’s ability to resist unauthorized attempts at usage and denial of service while still providing its service to legitimate users. Security often involves identifying special components, separating these from the rest of the system’s functionality and arranging the coordination and interaction of other components through them. All of this solutions are architectural based.

- **Availability.** Availability measures the proportion of time the system is up and running. Basically, it is the measurement of the length of time between failures as well as by how quickly the system is able to resume its operation in the occurrence of failure. This bring to the equation of availability:

  \[
  \alpha = \frac{\text{mean time to failure}}{\text{mean time to failure} + \text{mean time to repair}}
  \]

  It is closely related to reliability, which is the ability of the system to keep operating over time. Reliability is usually measured with mean time to failure. These attributes are both tied to architecture. Availability refers to both mean time to failure and mean time to repair, where they are addressed through architectural means.

- **Functionality.** Functionality is the ability of the system to do the work for, which it was intended. Functionality is orthogonal to structure, which means that it is largely non-architectural in nature. In fact, the achievement of functionality is only requirement. However, software architecture constraints the allocations of functionality to structure when other quality attributes are vital.

- **Usability.** Usability consists the areas of learnability, efficiency of systems, memorability, error detection, error handling and user satisfaction. By ensuring the right
information is available to user at the right time and allowing user get to the right destination in the system is architectural due to the information must flow across appropriate components via the connectors.

There are also some quality attributes that not discernable at runtime. Listed below are the attributes:

- **Portability.** Portability is the ability of the system to run under different computing environments. A system is considered portable to the extent that all of the assumptions about any particular computing environment are confined to one component. Encapsulation of platform-specific considerations in architecture typically takes the form of a portability layer, which is an abstract interface to its environment. A portability layer results from a straightforward application of the design principle of information hiding.

- **Modifiability.** Modifiability is the attribute that related most closely to the architecture of a system. It is largely a function of the locality of any change. Since the architecture defines the components and the responsibilities of each component, it also defines the circumstances under which each component will have to change. Modification is often flow from changes within the business needs in an organization.

- **Reusability.** Reusability often means the way of designing a system so that the system’s structure or some of its components can be reused in other applications in the future. It is related to software architecture in the architectural components are the units of reuse and to what extend the reusability a component depend on how tightly coupled to other components.

- **Integrability.** Integrability is the ability to ensure the separately developed components of the system to work correctly together. Basically, it depends on the complexity of the
components, their interaction mechanism and protocols and all architectural issues. Besides, it also depends upon how well and completely the interface to the components is specified.

- **Testability.** Testability refers to the ease of software can be made to portrait its fault. Particularly, it refers to the probability. A system’s testability relates to structural and architectural issues where it is the level of architectural documentation, its separation and the degree to which the system uses information hiding.

### 3.4.2 Qualities of the Architecture

In general, quality is related to the business environment in which the system is being constructed. However, there are also qualities that related directly to architecture. As conceptual integrity unifies the design of the system at all levels, architectural integrity does for the other stakeholders.

Correctness and completeness are necessary for the architecture, which they allow the meeting of all of the system’s requirements and runtime resource constraints. Buildability is the quality of the architecture that allows the system to be completed on time and changes as the systems development progresses. It refers to the ease of developing a desired system. Usually, it is measured in terms of cost and time. Another aspect of buildability is the state of knowledge about the problems being solved.

### 3.4.3 Software Architecture Quality Assessment

Conventional design methods focus on achieving the required system functionality, rather than software qualities. However, most methods focus on single quality attribute and treat all
others as secondary importance. Therefore, software architect needs to balance the various quality attributes for any realistic system. In order to assess quality attribute on handling software quality requirements, an architectural reengineering method is used to provide more objective and practical approach. Figure 3-3 shows the graphical method:

![Diagram](image)

**Figure 3-3 Software Architecture Quality Assessment**

The steps involved in the software architecture quality assessment are as follows:

*Incorporate new functionality requirements in the architecture.*

Although software architect generally will not design a system less reliable or reusable, the software qualities are not explicitly addressed at this stage. The result from this is a first version of the application architectural design.

2. *Software quality assessment.*
Each quality attribute (QA) is estimated, using initial scenario-based analysis for assessment technique. If all estimations are as good or better than required, the architectural design process is finished. Otherwise, next step needs to proceed.

iii. Architecture transformation.

During this stage, QA-Optimizing transformations are used to enhance the architecture. Each set of transformation results in a new version of architectural design.

iv. Software quality assessment.

The design is again evaluated and assessed, as the process is repeated from (iii) until all software quality requirements are met or until the software architect decides that no feasible solution exists.

3.5 Analyzing Software Architecture

Due to the importance of architectural decisions, it is suitable to have a close inspection. Particularly, it is often cost-effective to evaluate software quality as initial as possible in the software life cycle. When the problems are detected in the early phase of the development life cycle, it is easier and definitely cost lesser to correct. Software quality cannot be appended late in a project but it should be inherent as early as the beginning of the project. Therefore, it is essential to have an effective assessment technique to evaluate candidate architecture before it becomes the blueprint of the project.

During the acquisition of a large and complex software system that will have a long lifetime, it is vital that an organization develops an understanding on the underlying architecture. The inherent complexity is further compounded by application of distributed processing technologies such as CORBA, COM+ and Enterprise JavaBeans. Hence, this will
assist the assessment of the fitting of the candidates with respect of the qualities of importance.

It is well known that architecture cannot guarantee the quality or functionality of a desired system. Poor design, implementation and testing can always undermine an acceptable architecture. Decisions whether from high-level design to coding or implementation, will definitely affect the quality of the developed system. However, an architecture-based assessment provides only one dimension of a system’s quality characteristics. It only assesses the ability of the architecture to support the required qualities. Refinement of the architecture into an implementation preserves those qualities. Figure 3-4 shows the analysis and design of software architecture flow.
Figure 3-4 Analysis and Design Software Architecture Flow
Define a Candidate Architecture

i. Develop Design Guidelines

ii. Prioritize Use Cases

iii. Architectural Analysis

iv. Use Case Analysis

v. Submit Change Request

Refine the Architecture

i. Prioritize Use Cases

ii. Describe the Run-time Architectures

iii. Describe Distribution

iv. Identify Design Mechanism

v. Identify Design Elements

vi. Incorporate Existing Design Elements

vii. Review the Architecture

viii. Structure the Implementation Model

Analyze Behavior

i. Use Case Analysis

ii. Identify Design Elements

iii. Plan System Integration

iv. Review the Design

Design Components

i. Class Design

ii. Subsystem Design
iii. Use Case Design

iv. Review the Design

v. Implement Component

vi. Perform Unit Tests

> **Design Real Time Components**

i. Class Design

ii. Capsule Design

iii. Subsystem Design

iv. Use Case Design

v. Review the Design

vi. Implement Component

VILPERFORM UNIT TESTS

> **Design the Database**

i. Class Design

ii. Database Design

iii. Review the Design

iv. Implement Component
3.5.1 Software Architecture Analysis Method (SAAM)

Software architecture analysis method (SAAM) is a scenario-based method for analyzing architectures that provides an approach to characterize how well an architectural design responds to the demands resulted by a set of scenarios. Scenario is a set of specified sequence of steps involved in the use and modification of the system. Scenario actually serves as a representative for an entire class of scenarios.

Software architecture analysis method uses scenarios as benchmarks to compare and contrast the difference among the candidate of architectures. The purpose is to supply a way to validate assertion about the system quality from architectural description. This description identifies the components and connections and the overall coordinated behavior of the system. Eventually, these descriptions are measured to determine which architectural candidate supports the set of scenarios.
Figure 3-5 shows the sequential steps for Software Architecture Analysis Method:

![Diagram showing the steps of Software Architecture Analysis Method](image)

**Figure 3-5 Steps in Software Architecture Analysis Method**

1. **Scenario Development.**

   In scenario development, it is essential to capture all vital uses of a system, users of a system and qualities that a system is to satisfy. Hence, scenarios will represent tasks relevant to different stakeholders. Scenario elicitation needs a certain degree of skill and experience.

2. **Architecture Description**

   Candidates of the architectures should be described in an architectural notation in order all parties or stakeholders understand in the analysis. These architectural descriptions must indicate the system’s computational and data components where they are relevant connected. A typical representation will distinguish between data connections and control connections.
This straightforward lexicon gives a reasonable static representation of the architecture. This representation of the architecture is a description of the behavior of the system.

iii. Classification of Scenarios

A scenario is direct or indirect with respect to a particular architecture. When a scenario is not directly supported, there must be some change to the system that can be represented architecturally. Apparently, an architecture for which the scenarios of interest are directly supported is more capable than one that requires modification to support those similar scenarios. By having the distinction of direct or indirect is not completely reliable indicator of the complexity of a change.

iv. Individual Indirect Scenarios Evaluation

For the individual indirect scenario, the changes to the architecture are required to be listed and the cost must be estimated. The effect or set of changes that the scenario has on the architecture needs to be listed. At the end of this stage, there must be a summary of all listed scenarios whether it is direct or indirect. Weighting of difficulty need to embedded in this stage too. This summary is beneficial for comparing alternative architectural candidates due to it provides a simple way to determine which architecture better support a collection of scenarios.

v. Scenario Interaction Assessment

If two or more indirect scenarios need changes to a single component of a system, interaction in that component occurs. It is vital due to it exposes the allocation of functionality to the product’s design. Areas of high scenario interaction reveal potentially poor separation of concerns in a system component. It also guides the designers to focus their subsequent attention. Fundamentally, high interaction among scenarios that are different with respect to
low cohesion suggests high structural complexity. Thus, high interaction signals high cohesion. Eventually, scenario interaction is heavily correlated with defects in the product.

6. Overall Evaluation

When the architecture is being compared, a weight should be assigned to each scenario and the scenario interaction should be used to determine an overall ranking of the candidate architectures. The purpose of assigning weight is to resolve the situation where the first candidate architecture score better in the first half of scenario but the second candidate has a better score in the other half. Assigning weight is subjective because it involves all the stakeholders in the system. Software architecture analysis method produces a collection of small metrics, which is a small set of per-scenario analyses. It has been used to gain a deeper and complete understanding of the competing architectures, where it is beneficial for performing comparative analysis. Developing scenarios and representing the architecture are interdependent steps. Making decision on the granularity for architecture will depend on the types of scenarios that are intended to evaluate. Determining a reasonable set of scenarios involves the activities that the system is expected to perform where is also reflected in the architecture.

3.5.2 Simulation-Based Assessment

Software architecture analysis method revolves in scenario-based assessment where it depends on the profile defined for the quality attribute. The effectiveness of this method is largely dependent on the representativeness of the scenarios. However, this method is rather
static in that no executable dynamic model is used. An alternative method is simulation-based assessment in which a high-level implementation of the software architecture is used.

The fundamental approach consists of an implementation of the components of the architecture and an implementation of the context of the system. The context where the software system is supposed to execute, should be simulated at a suitable abstraction level. Hence, the implementation can then be used for assessing the behavior of the architecture under various circumstances.

Once a simulated context and high-level implementation of the software architecture are available, scenarios from relevant profiles can be used to assess the relevant quality attributes. For instance, performance can be evaluated by executing the usage profile in the simulated context and architecture and collecting information about the throughput and response time to system events.

Figure 3-6 shows the sequential steps in Simulated-Based assessment:
1. Define and Implement the Context

In this method, the most initial step is to identify the interfaces of the software architecture to its context and to decide how the behaviour of the context at the interfaces should be simulated. It is essential to select the correct level of abstraction, which generally means removing most of the details normally present at system interfaces. In an embedded systems where time often plays a role, decision need to be made whether time-related behavior should be implemented in the system. Decision depends on the quality attributes and the required accuracy that are intended to assess. Implementation should only be implemented at the level of realism when performing the assessments.

2. Implement the Architectural Components
Once the system context has been defined and implemented, the components in the software architecture are constructed. The description of the architectural design should at least define the interfaces and the connections of the components so those parts can be taken directly from the design descriptions. The behaviour of the components in response to events or messages on their interface may not be specified although there is a common understanding and decide upon the level of detail associated with the implementation. The domain behaviour is dependent on the quality attributes that are intended to assess.

**iii. Implement Profile**

An associated profile need to be implemented on the system depending on the quality attributes that are intended to use during the assessment. It is implemented by using random selection based on the normalized weights of the scenarios. It is performed by using a scenario activator, which randomly selecting scenarios from the profile.

**iv. Simulate System and Initiate Profile**

The complete simulation including context, architecture and profiles is ready for use. Since the goal of the simulation is to assess the software architecture, the type of result depends on the quality attribute being assessed.

**v. Predict Quality Attribute**

The final step requires to analyze the collected data and to predict the assessed quality attribute based on the data. Amount of the collected data need to be condensed depending on type of simulation and the assessed quality attributes.

Simulation of the architectural design is not only useful for quality attribute assessment, but also for evaluating the functionality aspects of the design. It requires defining
the behavior and interaction of the architectural entities very precisely, which may lead to uncover inconsistencies in the design earlier than using traditional approach.

3.6 Performing Architecture Assessment

Architecture assessment is an iterative activity that is part of an iterative design process. Once the architecture is assessed, it will proceed into transformation phase. After transformation, the architecture will be assessed its quality attributes.

During the first time the architecture is assessed, the profiles for the relevant quality attributes should be defined. Since both the definition of the profile and the repetitive assessment process are time-consuming activities, only those quality attributes should be selected for explicit assessment that are crucial for the success of the system.

Once the required quality attributes have been chosen and the profiles for theses quality attributes have been defined, this lead to the next step to select an assessment technique. During the design iterations, the actual architecture assessment is performed during every iteration and quality attribute. Finally, it assembles the result and decides upon continuation, re-negotiation or termination of the design project.