UNIT-I
ENVISIONING ARCHITECTURE

CHAPTER 1

WHAT IS SOFTWARE ARCHITECTURE :

WHAT SOFTWARE ARCHITECTURE IS AND WHAT IT ISN'T

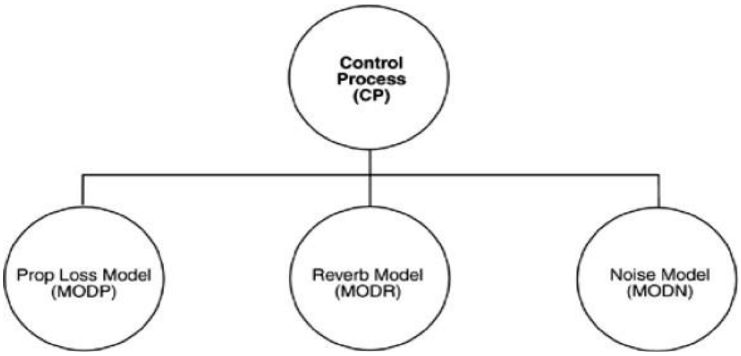


Figure 1.1 : Typical, but uninformative, presentation of a software architecture

Figure 1.1, taken from a system description for an underwater acoustic simulation, purports to describe that system's "top-level architecture" and is precisely the kind of diagram most often displayed to help explain an architecture. Exactly what can we tell from it?

- The system consists of four elements.
- Three of the elements— Prop Loss Model (MODP), Reverb Model (MODR), and Noise Model (MODN)— might have more in common with each other than with the fourth—Control Process (CP)—because they are positioned next to each other.
- All of the elements apparently have some sort of relationship with each other, since the diagram is fully connected.

Is this an architecture? What can we *not* tell from the diagram?

- *What is the nature of the elements?*
  What is the significance of their separation? Do they run on separate processors? Do they run at separate times? Do the elements consist of processes, programs, or both? Do they represent ways in which the project labor will be divided, or do they convey a sense of runtime separation? Are they objects, tasks, functions, processes, distributed programs, or something else?

- *What are the responsibilities of the elements?*

  What is it they do? What is their function in the system?

- *What is the significance of the connections?*

  Do the connections mean that the elements communicate with each other, control each other, send data to each other, use each other, invoke each other, synchronize with each other, share some information-hiding secret with each other, or some combination of these or other relations? What are the mechanisms for the communication? What information flows across the mechanisms, whatever they may be?

- *What is the significance of the layout?*

  Why is CP on a separate level? Does it call the other three elements, and are the others not allowed to call it? Does it contain the other three in an implementation unit sense? Or is there simply no room to put all four elements on the same row in the diagram?

This diagram does not show a software architecture. We now define what *does* constitute a software architecture:

Software Architecture Definition:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Let's look at some of the implications of this definition in more detail.

- Architecture defines software elements
- The definition makes clear that systems can and do comprise more than one structure and that no one structure can irrefutably claim to be the architecture.
- The definition implies that every computing system with software has a software architecture because every system can be shown to comprise elements and the relations among them.
- The behavior of each element is part of the architecture insofar as that behavior can be observed or discerned from the point of view of another element. Such behavior is what allows elements to interact with each other, which is clearly part of the architecture.

OTHER POINTS OF VIEW

The study of software architecture is an attempt to abstract the commonalities inherent in system design, and as such it must account for a wide range of activities, concepts, methods, approaches, and results.

- ***Architecture is high-level design**.* Other tasks associated with design are not architectural, such as deciding on important data structures that will be encapsulated.
- ***Architecture is the overall structure of the system.*** The different structures provide the critical engineering leverage points to imbue a system with the quality attributes that will render it a success or failure. The multiplicity of structures in an architecture lies at the heart of the concept.
- ***Architecture is the structure of the components of a program or system, their interrelationships, and the principles and guidelines governing their design and evolution over time.*** Any system has an architecture that can be discovered and analyzed independently of any knowledge of the process by which the architecture was designed or evolved.
- ***Architecture is components and connectors.*** Connectors imply a runtime mechanism for transferring control and data around a system. When we speak of "relationships" among elements, we intend to capture both runtime and non-runtime relationships.

ARCHITECTURAL PATTERNS, REFERENCE MODELS & REFERENCE ARCHITECTURES

An architectural pattern is a description of element and relation types together with a set of constraints on how they may be used.

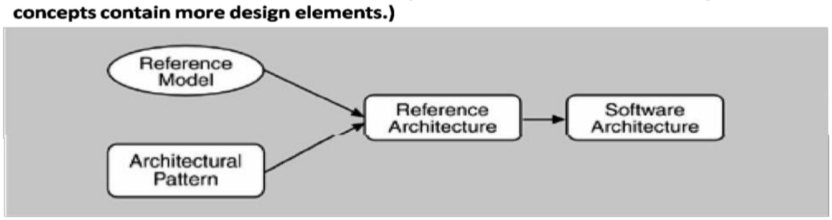
For ex: client-server is a common architectural pattern. Client and server are two element types, and their coordination is described in terms of the protocol that the server uses to communicate with each of its clients.

A reference model is a division of functionality together with data flow between the pieces.

A reference model is a standard decomposition of a known problem into parts that cooperatively solve the problem.

A reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them. Whereas a reference model divides the functionality, A reference architecture is the mapping of that functionality onto a system decomposition.

Figure 2.2. The relationships of reference models, architectural patterns, reference architectures, and software architectures. (The arrows indicate that subsequent concepts contain more design elements.)



Reference models, architectural patterns, and reference architectures are not architectures; they are useful concepts that capture elements of an architecture. Each is the outcome of early design decisions. The relationship among these design elements is shown in Figure 2.2. A software architect must design a system that provides concurrency, portability, modifiability, usability, security, and the like, and that reflects consideration of the tradeoffs among these needs.

WHY IS SOFTWARE ARCHITECTURE IMPORTANT?

There are fundamentally three reasons for software architecture’s importance from a technical perspective.

- ***Communication among stakeholders***: software architecture represents a common abstraction of a system that most if not all of the system’s stakeholders can use as a basis for mutual understanding, negotiation, consensus and communication.
- ***Early design decisions***: Software architecture manifests the earliest design decisions about a system with respect to the system's remaining development, its deployment, and its maintenance life.
- It is the earliest point at which design decisions governing the system to be built can be analyzed.
- ***Transferable abstraction of a system***: software architecture model is transferable across systems. It can be applied to other systems exhibiting similar quality attribute and functional attribute and functional requirements and can promote large-scale re-use.

We will address each of these points in turn:

❖ ARCHITECTURE IS THE VEHICLE FOR STAKEHOLDER COMMUNICATION

- Each stakeholder of a software system – customer, user, project manager, coder, tester and so on - is concerned with different system characteristics that are affected by the architecture.
- For ex. The user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways.
- Architecture provides a common language in which different concerns can be expressed, negotiated, and resolved at a level that is intellectually manageable even for large, complex systems.

❖ ARCHITECTURE MANIFESTS THE EARLIEST SET OF DESIGN DECISIONS

Software architecture represents a system's earliest set of design decisions. These early decisions are the most difficult to get correct and the hardest to change later in the development process, and they have the most far-reaching effects.

- **The architecture defines constraints on implementation**
  - This means that the implementation must be divided into the prescribed elements, the elements must interact with each other in the prescribed fashion, and each element must fulfill its responsibility to the others as dictated by the architecture.
- **The architecture dictates organizational structure**
  - The normal method for dividing up the labor in a large system is to assign different groups different portions of the system to construct. This is called the work breakdown structure of a system.
- **The architecture inhibits or enables a system's quality attributes**
  - Whether a system will be able to exhibit its desired (or required) quality attributes is substantially determined by its architecture.
  - However, the architecture alone cannot guarantee functionality or quality.
  - Decisions at all stages of the life cycle—from high-level design to coding and implementation—affect system quality.
  - Quality is not completely a function of architectural design. To ensure quality, a good architecture is necessary, but not sufficient.
- **Predicting system qualities by studying the architecture**
  - Architecture evaluation techniques such as the architecture tradeoff analysis method support top-down insight into the attributes of software product quality that is made possible (and constrained) by software architectures.
- **The architecture makes it easier to reason about and manage change**
  - Software systems change over their lifetimes.
  - Every architecture partitions possible changes into three categories: local, nonlocal, and architectural.
  - A local change can be accomplished by modifying a single element.
  - A nonlocal change requires multiple element modifications but leaves the underlying architectural approach intact.

- **The architecture helps in evolutionary prototyping**
  - The system is executable early in the product's life cycle. Its fidelity increases as prototype parts are replaced by complete versions of the software.
  - A special case of having the system executable early is that potential performance problems can be identified early in the product's life cycle.
- **The architecture enables more accurate cost and schedule estimates**
  - Cost and schedule estimates are an important management tool to enable the manager to acquire the necessary resources and to understand whether a project is in trouble.

❖ ARCHITECTURE AS A TRANSFERABLE, RE-USABLE MODEL

The earlier in the life cycle re-use is applied, the greater the benefit that can be achieved. While code re- use is beneficial, re-use at the architectural level provides tremendous leverage for systems with similar requirements.

- **Software product lines share a common architecture**

A software product line or family is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

- **Systems can be built using large. Externally developed elements**

Whereas earlier software paradigms focused on programming as the prime activity, with progress measured in lines of code, architecture-based development often focuses on composing or assembling elements that are likely to have been developed separately, even independently, from each other.

- **Less is more: it pays to restrict the vocabulary of design alternatives**

We wish to minimize the design complexity of the system we are building. Advantages to this approach include enhanced re-use more regular and simpler designs that are more easily understood and communicated, more capable analysis, shorter selection time, and greater interoperability.

- **An architecture permits template-based development**

An architecture embodies design decisions about how elements interact that, while reflected in each element's implementation, can be localized and written just once.

Templates can be used to capture in one place the inter-element interaction mechanisms.

- **An architecture can be the basis for training**

  The architecture, including a description of how elements interact to carry out the required behavior, can serve as the introduction to the system for new project members.

ARCHITECTURAL STRUCTURES AND VIEWS

Architectural structures can by and large be divided into three groups, depending on the broad nature of the elements they show.

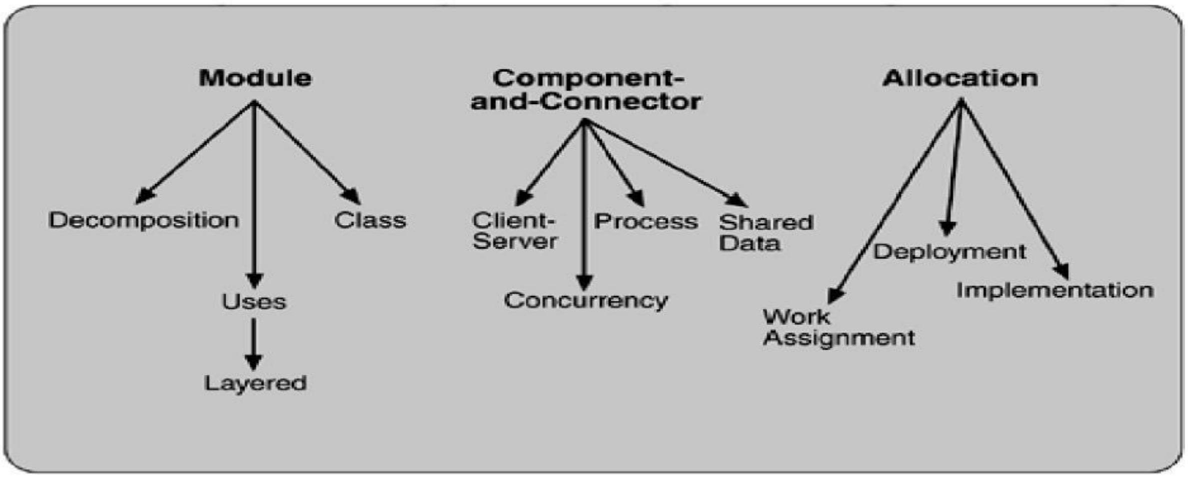
- ***Module structures*.**

  Here the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. They are assigned areas of functional responsibility. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?

- ***Component-and-connector structures*.**

  Here the elements are runtime components (which are the principal units of computation) and connectors (which are the communication vehicles among components). Component-and-connector structures help answer questions such as What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel?

Figure 2-3. Common software architecture structures



- ***Allocation structures***.

  Allocation structures show the relationship between the software elements and the elements in one or more external environments in which the software is created and executed. They answer questions such as What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of software elements to development teams?

SOFTWARE STRUCTURES :

- ***Module***

Module-based structures include the following structures.

✓ ***Decomposition:*** The units are modules related to each other by the "is a submodule of " relation, showing how larger modules are decomposed into smaller ones recursively until they are small enough to be easily understood.

✓ ***Uses:*** The units are related by the *uses* relation. One unit uses another if the correctness of the first requires the presence of a correct version (as opposed to a stub) of the second.

✓ ***Layered:*** Layers are often designed as abstractions (virtual machines) that hide implementation specifics below from the layers above, engendering portability.

✓ ***Class or generalization:*** The class structure allows us to reason about re-use and the incremental addition of functionality.

- ***Component-and-connector***

Component-and-connector structures include the following structures

✓ ***Process or communicating processes***: The units here are processes or threads that are connected with each other by communication, synchronization, and/or exclusion operations.

✓ ***Concurrency:*** The concurrency structure is used early in design to identify the requirements for managing the issues associated with concurrent execution.

✓ ***Shared data or repository:*** This structure comprises components and connectors that create, store, and access persistent data

✓ ***Client-server:*** This is useful for separation of concerns (supporting modifiability), for physical distribution, and for load balancing (supporting runtime performance).

- ***Allocation***

Allocation structures include the following structures

✓ ***Deployment:*** This view allows an engineer to reason about performance, data integrity, availability, and security.

- ✓ ***Implementation:*** This is critical for the management of development activities and builds processes.
- ✓ ***Work assignment:*** This structure assigns responsibility for implementing and integrating the modules to the appropriate development teams.

RELATING STRUCTURES TO EACH OTHER

Each of these structures provides a different perspective and design handle on a system, and each is valid and useful in its own right. In general, mappings between structures are many to many. Individual structures bring with them the power to manipulate one or more quality attributes. They represent a powerful separation-of- concerns approach for creating the architecture.

WHICH STRUCTURES TO CHOOSE?

Kruchten's four views follow:

- ✓ ***Logical.*** The elements are "key abstractions," which are manifested in the object-oriented world as objects or object classes. This is a module view.
- ✓ ***Process.*** This view addresses concurrency and distribution of functionality. It is a component-and- connector view.
- ✓ ***Development.*** This view shows the organization of software modules, libraries, subsystems, and units of development. It is an allocation view, mapping software to the development environment.
- ✓ ***Physical.*** This view maps other elements onto processing and communication nodes and is also an allocation view

Kruchten's 4+1 View Model :

4+1 is a view model designed by Philippe Kruchten for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views". The views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers and project managers. The four views of the model are logical, development, process and physical view. In addition selected use cases or scenarios are used to illustrate the architecture serving as the 'plus one' view. Hence the model contains 4+1 views.

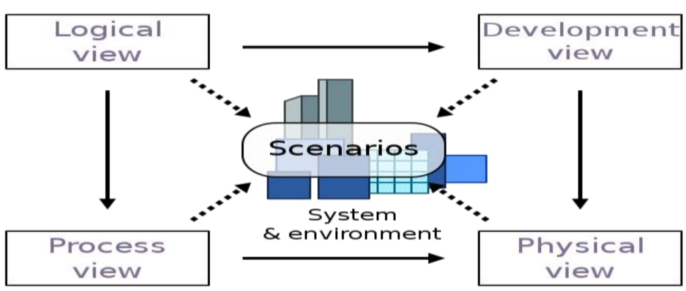


Figure : Kruchten's 4+1 View Model

- ✓ ***Development view:*** The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components.
  UML Diagrams used to represent the development view include the Package diagram.
- ✓ ***Logical view:*** The logical view is concerned with the functionality that the system provides to end-users. UML diagrams used to represent the logical view include, class diagrams, and state diagrams.
- ✓ ***Physical view:*** The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. This view is also known as the deployment view. UML diagrams used to represent the physical view include the deployment diagram.[2]
- ✓ ***Process view:*** The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML diagrams to represent process view include the activity diagram.[2]
- ✓ ***Scenarios:*** The description of an architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view. The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the use case view.

CHAPTER 2

THE ARCHITECTURE BUSINESS CYCLE

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

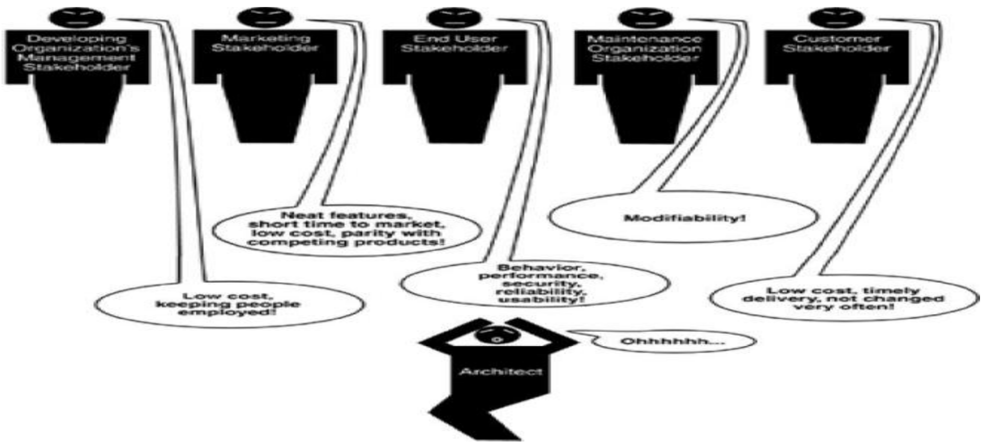
Software architecture is a result of technical, business and social influences. Its existence in turn affects the technical, business and social environments that subsequently influence future architectures. We call this cycle of influences, from environment to the architecture and back to the environment, the ***Architecture Business Cycle (ABC).*** This chapter introduces the ABC in detail and examine the following:

- ✓ How organizational goals influence requirements and development strategy.
- ✓ How requirements lead to architecture.
- ✓ How architectures are analyzed.
- ✓ How architectures yield systems that suggest new organizational capabilities and requirements.

ARCHITECTURE INFULENCES :

An architecture is the result of a set of business and technical decisions. There are many influences at work in its design, and the realization of these influences will change depending on the environment in which the architecture is required to perform. Even with the same requirements, hardware, support software, and human resources available, an architect designing a system today is likely to design a different system than might have been designed five years ago.

- ❖ **ARCHITECTURES ARE INFLUENCED BY SYSTEM STAKEHOLDERS**
    - ✓ Many people and organizations interested in the construction of a software system are referred to as stakeholders. E.g. customers, end users, developers, project manager etc.
    - ✓ Figure below shows the architect receiving helpful stakeholder “suggestions”.



- ✓ Having an acceptable system involves properties such as performance, reliability, availability, platform compatibility, memory utilization, network usage, security, modifiability, usability, and interoperability with other systems as well as behavior.
- ✓ The underlying problem, of course, is that each stakeholder has different concerns and goals, some of which may be contradictory.
- ✓ The reality is that the architect often has to fill in the blanks and mediate the conflicts.

❖ ARCHITECTURES ARE INFLUENCED BY THE DEVELOPING ORGANIZATIONS.

- ✓ Architecture is influenced by the structure or nature of the development organization.
- ✓ There are three classes of influence that come from the developing organizations: immediate business, long-term business and organizational structure.
  - An organization may have an immediate business investment in certain assets, such as existing architectures and the products based on them.
  - An organization may wish to make a long-term business investment in an infrastructure to pursue strategic goals and may review the proposed system as one means of financing and extending that infrastructure.
  - The organizational structure can shape the software architecture.

❖ ARCHITECTURES ARE INFLUENCED BY THE BACKGROUND AND EXPERIENCE OF THE ARCHITECTS.

- ✓ If the architects for a system have had good results using a particular architectural approach, such as distributed objects or implicit invocation, chances are that they will try that same approach on a new development effort.
- ✓ Conversely, if their prior experience with this approach was disastrous, the architects may be reluctant to try it again.
- ✓ Architectural choices may also come from an architect's education and training, exposure to successful architectural patterns, or exposure to systems that have worked particularly poorly or particularly well.
- ✓ The architects may also wish to experiment with an architectural pattern or technique learned from a book or a course.

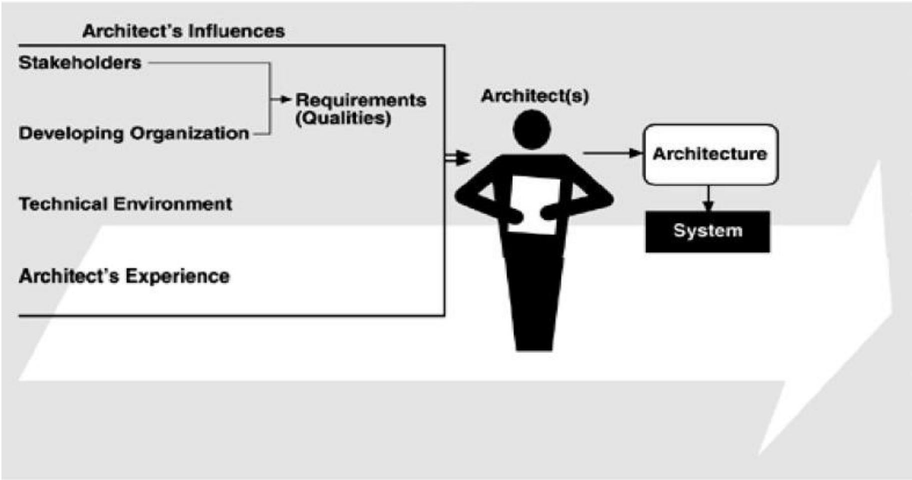
❖ ARCHITECTURES ARE INFLUENCED BY THE TECHNICAL ENVIRONMENT

- ✓ A special case of the architect's background and experience is reflected by the *technical environment*.
- ✓ The environment that is current when an architecture is designed will influence that architecture.

- ✓ It might include standard industry practices or software engineering prevalent in the architect's professional community.

❖ RAMIFICATIONS OF INFLUENCES ON AN ARCHITECTURE

- ✓ The influences on the architect, and hence on the architecture, are shown in Figure 1.3.

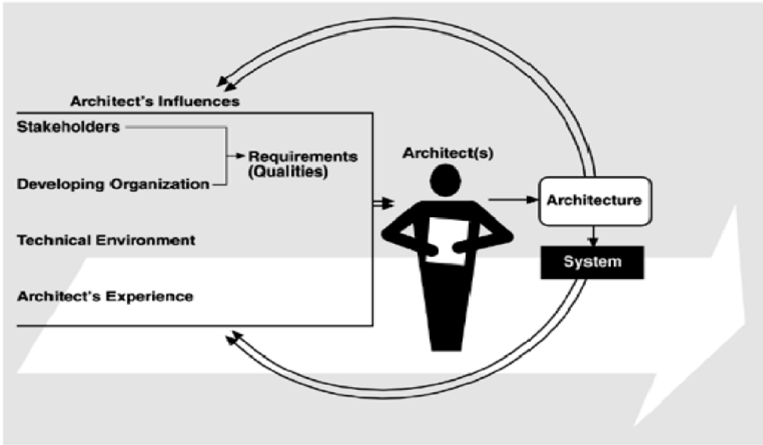


- ✓ Influences on an architecture come from a wide variety of sources. Some are only implied, while others are explicitly in conflict.
- ✓ Architects need to know and understand the nature, source, and priority of constraints on the project as early as possible.
- ✓ Therefore, *they must identify and actively engage the stakeholders to solicit their needs and expectations*.
- ✓ Architects are influenced by the requirements for the product as derived from its stakeholders, the structure and goals of the developing organization, the available technical environment, and their own background and experience.

❖ THE ARCHITECTURE AFFECTS THE FACTORS THAT INFLUENCE THEM

- ✓ Relationships among business goals, product requirements, architects experience, architectures and fielded systems form a cycle with feedback loops that a business can manage.
- ✓ A business manages this cycle to handle growth, to expand its enterprise area, and to take advantage of previous investments in architecture and system building.
- ✓ Figure 1.4 shows the feedback loops. Some of the feedback comes from the architecture itself, and some comes from the system built from it.

Figure 1.4. The Architecture Business Cycle



1) The architecture can affect the goals of the developing organization. A successful system built from it can enable a company to establish a foothold in a particular market area. The architecture can provide opportunities for the efficient production and deployment of the similar systems, and the organization may adjust its goals to take advantage of its newfound expertise to plumb the market. This is feedback from the system to the developing organization and the systems it builds.
2) The architecture can affect customer requirements for the next system by giving the customer the opportunity to receive a system in a more reliable, timely and economical manner than if the subsequent system were to be built from scratch.
3) The process of system building will affect the architect's experience with subsequent systems by adding to the corporate experience base.
4) A few systems will influence and actually change the software engineering culture. i.e, The technical environment in which system builders operate and learn.

SOFTWARE PROCESSES AND THE ARCHITECTURE BUSINESS CYCLE

Software process is the term given to the organization, ritualization, and management of software development activities.

The various activities involved in creating software architecture are:

- **Creating the business case for the system**
    - It is an important step in creating and constraining any future requirements.
    - How much should the product cost?
    - What is its targeted market?
    - What is its targeted time to market?
    - Will it need to interface with other systems?

- These are all the questions that must involve the system's architects.
- They cannot be decided solely by an architect, but if an architect is not consulted in the creation of the business case, it may be impossible to achieve the business goals.

- **Understanding the requirements**
  - There are a variety of techniques for eliciting requirements from the stakeholders.
  - For ex:
    - ✓ Object oriented analysis uses scenarios, or "use cases" to embody requirements.
    - ✓ Safety-critical systems use more rigorous approaches, such as finite-state-machine models or formal specification languages.
  - Another technique that helps us understand requirements is the creation of prototypes.
  - Regardless of the technique used to elicit the requirements, the desired qualities of the system to be constructed determine the shape of its structure.
- **Creating or selecting the architecture**
  - In the landmark book *The Mythical Man-Month*, Fred Brooks argues forcefully and eloquently that conceptual integrity is the key to sound system design and that conceptual integrity can only be had by a small number of minds coming together to design the system's architecture.
- **Documenting and communicating the architecture**
  - For the architecture to be effective as the backbone of the project's design, it must be communicated clearly and unambiguously to all of the stakeholders.
  - Developers must understand the work assignments it requires of them, testers must understand the task structure it imposes on them, management must understand the scheduling implications it suggests, and so forth.
- **Analyzing or evaluating the architecture**
  - Choosing among multiple competing designs in a rational way is one of the architect's greatest challenges.
  - Evaluating an architecture for the qualities that it supports is essential to ensuring that the system constructed from that architecture satisfies its stakeholders needs.
  - Use scenario-based techniques or architecture tradeoff analysis method (ATAM) or cost benefit analysis method (CBAM).

- **Implementing the system based on the architecture**
    - This activity is concerned with keeping the developers faithful to the structures and interaction protocols constrained by the architecture.
    - Having an explicit and well-communicated architecture is the first step toward ensuring architectural conformance.
- **Ensuring that the implementation conforms to the architecture**
    - Finally, when an architecture is created and used, it goes into a maintenance phase.
    - Constant vigilance is required to ensure that the actual architecture and its representation remain to each other during this phase.

WHAT MAKES A "GOOD" ARCHITECTURE?

Given the same technical requirements for a system, two different architects in different organizations will produce different architectures, how can we determine if either one of them is the right one?

We divide our observations into two clusters: process recommendations and Structural recommendations.

Process rules of thumb are as follows:

- The architecture should be the product of a single architect or a small group of architects with an identified leader.
- The architect (or architecture team) should have the functional requirements for the system and an articulated, prioritized list of quality attributes that the architecture is expected to satisfy.
- The architecture should be well documented, with at least one static view and one dynamic view, using an agreed-on notation that all stakeholders can understand with a minimum of effort.
- The architecture should be circulated to the system's stakeholders, who should be actively involved in its review.
- The architecture should be analyzed for applicable quantitative

measures (such as maximum throughput) and formally evaluated for quality attributes before it is too late to make changes to it.

- The architecture should lend itself to incremental implementation via the creation of a “skeletal” system in which the communication paths are exercised but which at first has minimal functionality.
- This skeletal system can then be used to “grow” the system incrementally, easing the integration and testing efforts.
- The architecture should result in a specific (and small) set of resource contention areas, the resolution of which is clearly specified, circulated and maintained.

Structural rules of thumb are as follows:

- The architecture should feature well-defined modules whose functional responsibilities are allocated on the principles of information hiding and separation of concerns.
- Each module should have a well-defined interface that encapsulates or “hides” changeable aspects from other software that uses its facilities. These interfaces should allow their respective development teams to work largely independent of each other.
- Quality attributes should be achieved using well-known architectural tactics specific to each attribute.
- The architecture should never depend on a particular version of a commercial product or tool.
- Modules that produce data should be separate from modules that consume data. This tends to increase modifiability.
- For parallel processing systems, the architecture should feature well-defined processors or tasks that do not necessarily mirror the module decomposition structure.
- Every task or process should be written so that its assignment to a specific processor can be easily changed, perhaps even at runtime.
- The architecture should feature a small number of simple interaction patterns.