IT6602 SOFTWARE ARCHITECTURES

UNIT I

INTRODUCTION AND ARCHITECTURAL DRIVERS

Introduction – What is software architecture? – Standard Definitions – Architectural structures – Influence of software architecture on organization-both business and technical – Architecture Business Cycle- Introduction – Functional requirements – Technical constraints – Quality Attributes.

1.WHAT IS SOFTWARE ARCHITECTURE?

- The software architecture of a program or computing system is a representation of the system that aids in the understanding of how the system will behave.

- Software architecture serves as the blueprint for both the system and the project developing it, defining the work assignments that must be carried out by design and implementation teams.

- The architecture is the primary carrier of system qualities such as performance, modifiability, and security, none of which can be achieved without a unify(combined) architectural vision.

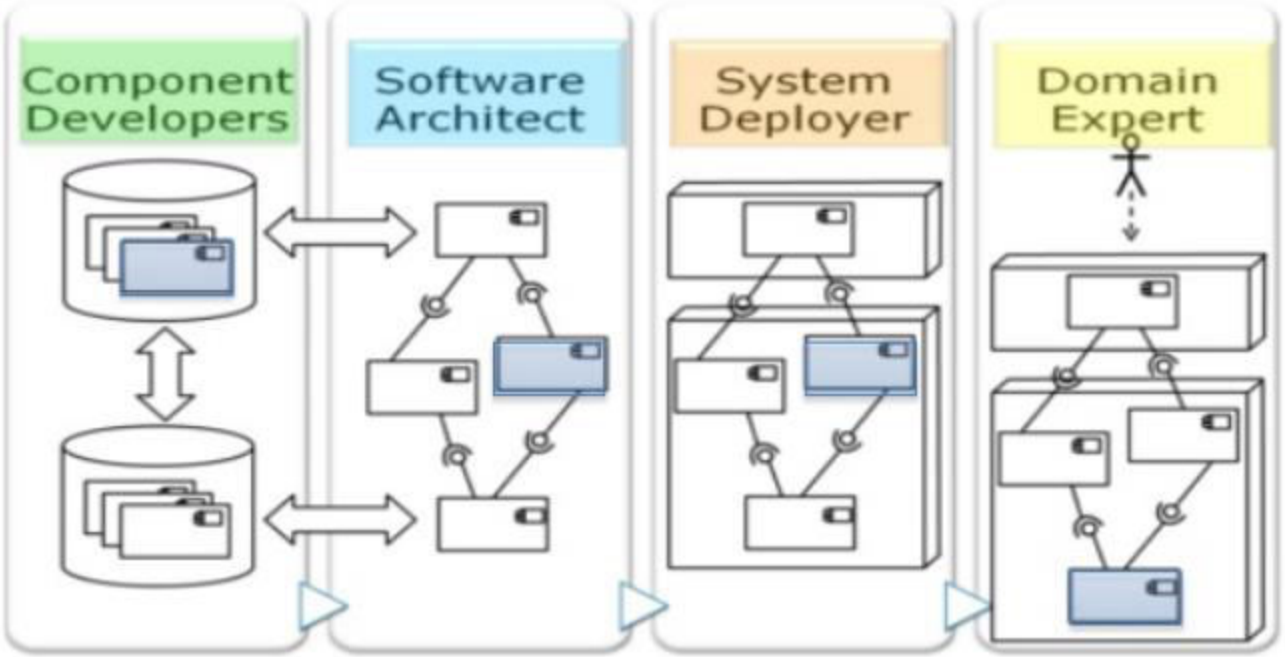
- Architecture is an artifact(hand-made object) for early analysis to make sure that a design approach will yield an acceptable system.

- By building effective architecture, the design risks can be identified and can be mitigate(diminished) early in the development process.

- If a project has not achieved system architecture, including its rationale(foundation), the project should not proceed to full-scale system development.

- Specifying the architecture as a deliverable enables its use throughout the development and maintenance process.-Barry Boehm

SOFTWARE ARCHITECTURE



The software architecture of a program or computing system is the structure or structures of the system, which comprise **software components** [and connectors], **the externally visible properties of those components** [and connectors] and **the relationships among them**."

1.1 What Software Architecture is and What it isn't?

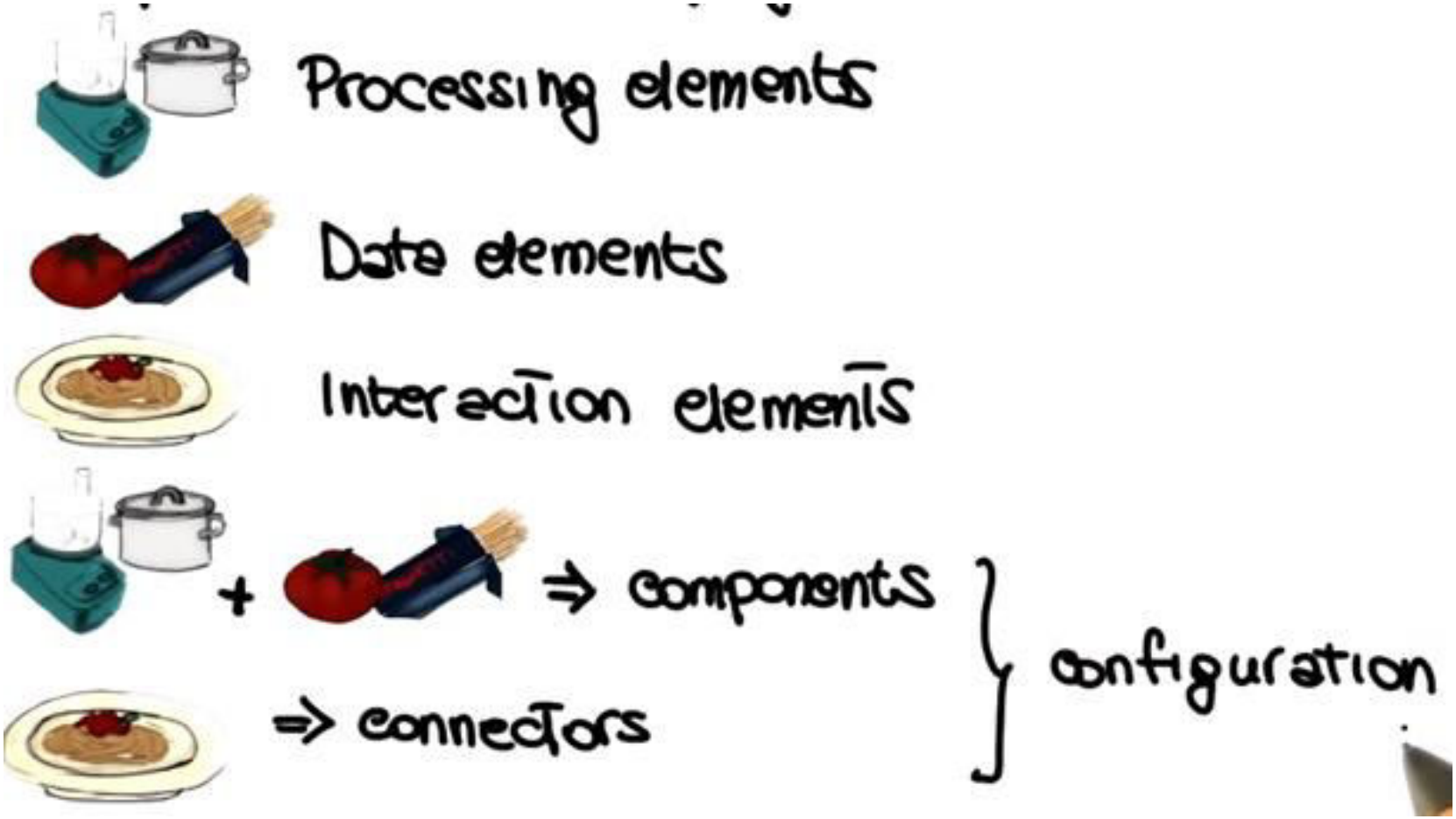
- The **software architecture** of a system is the set of structures needed to reason(explain) about the system, which comprise software elements, relations among them, and properties of both.

- **System architecture** is concerned with a total system, including hardware, software, and humans.

- Architecture is a set of software structures
  - (program) decomposition structures, component-and-connector structures (C&C), allocation structures
- Architecture is an abstraction(***is a technique for managing complexity of computer systems.*** *)*
  - Model of a software system

- Every software system has a software architecture
  - documented or undocumented
- Architecture includes behaviour
  - (external) behaviours/properties of architectural elements are part of architecture
- Not all architectures are good architectures
  - determined by architecture's quality attributes

SOFTWARE ARCHITECTURE ELEMENTS



- Software architecture must be distinguished from lower-level design (e.g., design of component internals and algorithms) and implementation, on the one hand, and other kinds of related architectures, on the other.

- Software architecture is not the information (or data) model, though it uses the information model to get type information for method signatures on interfaces.
  - *An Information Model is an organizational framework that you use to categorize your information resources.*

- It is also not the architecture of the physical system, including processors, networks, and the like, on which the software will run.

- It uses this information in evaluating the impact of architectural choices on system qualities such as performance and reliability.

- More obviously, perhaps, it is also not the hardware architecture of a product to be manufactured.

- While each of these other architectures typically have their own specialists leading their design, these architectures impact and are impacted by the software architecture, and where possible, should not be designed in isolation from one another.

- This is the domain of *system* architecting.

1.2 UNDERSTANDING SOFTWARE ARCHITECTURE

- An Example taken from a system description for an underwater acoustic simulation purports(highlights) to describe that system's "top-level architecture" and is precisely the kind of diagram most often displayed to help explain the architecture.

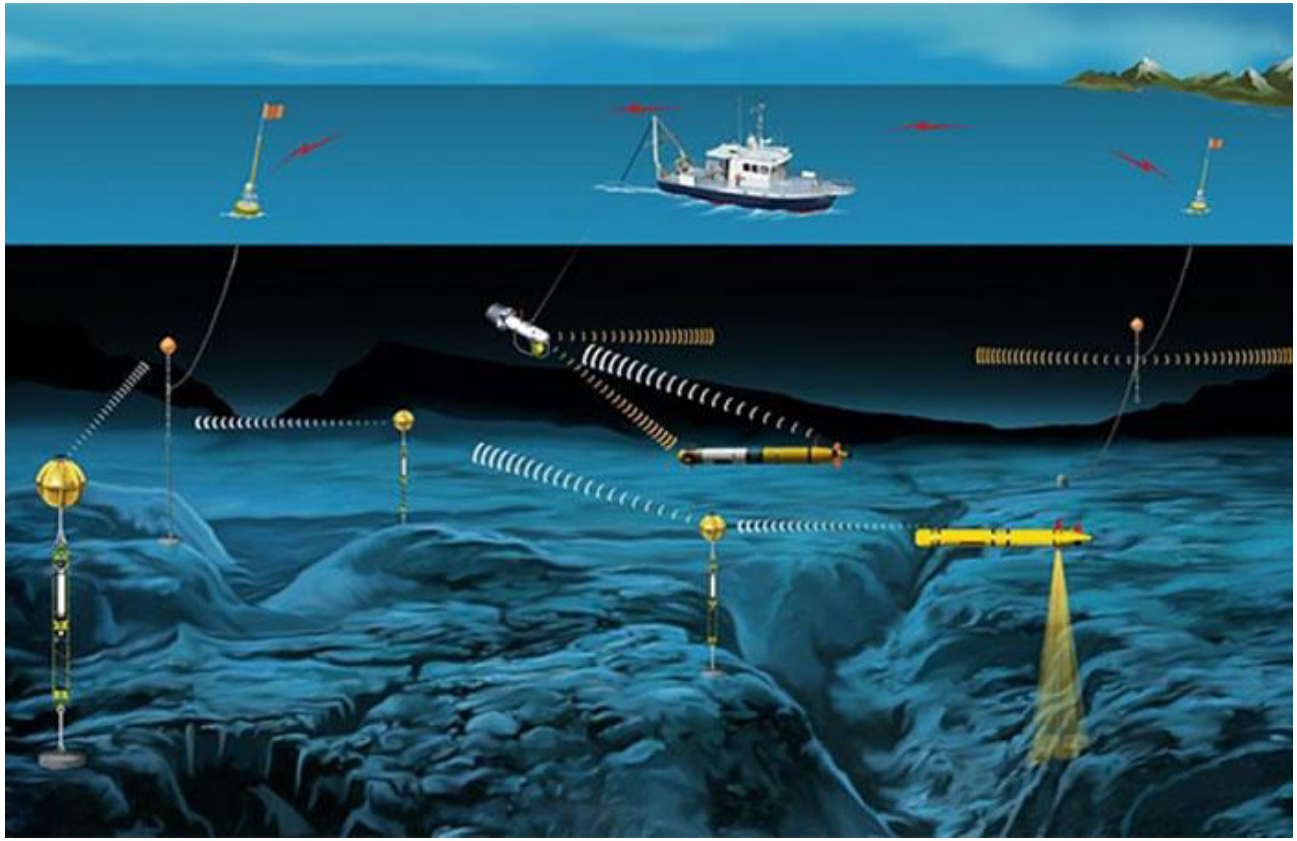
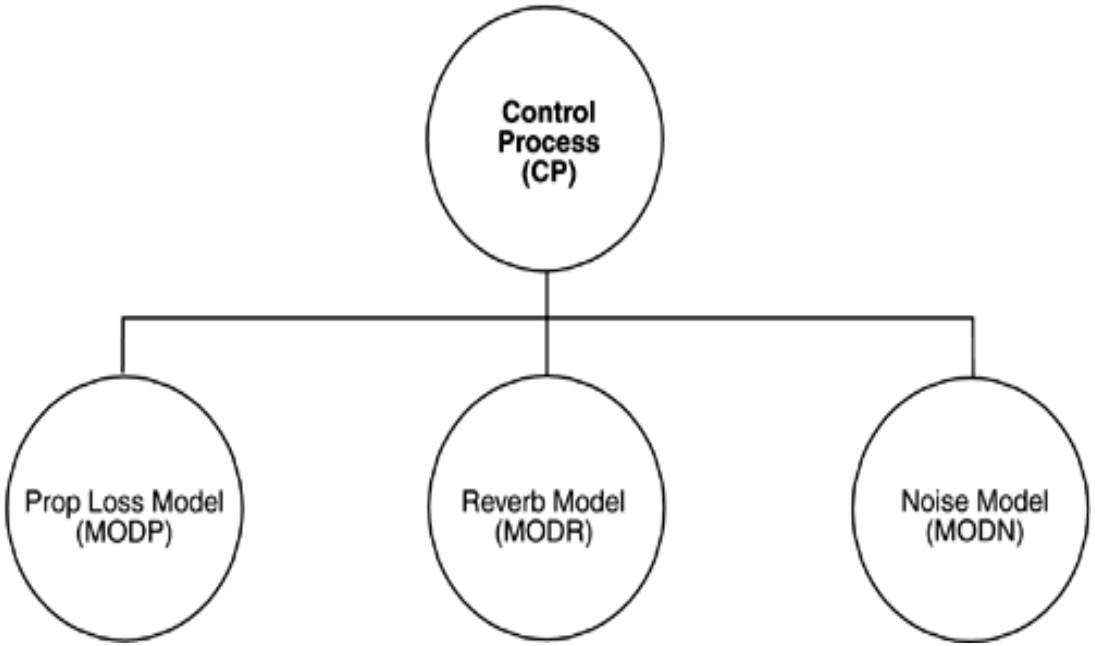
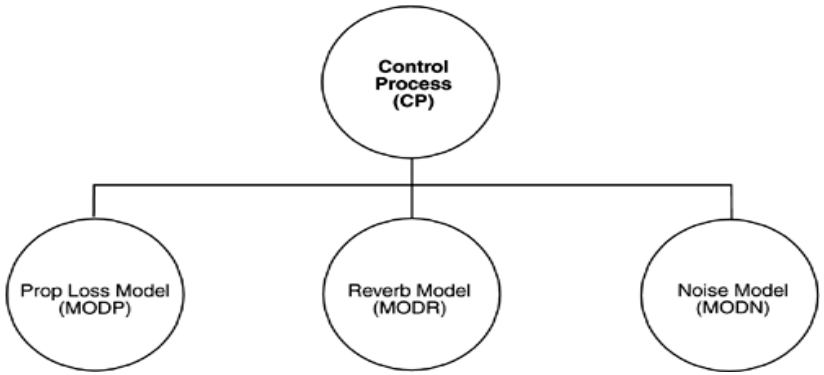


Figure.1 Typical but uninformative, presentation of a software architecture



1.2.1 WHAT IS OBSERVED FROM THIS DIAGRAM?

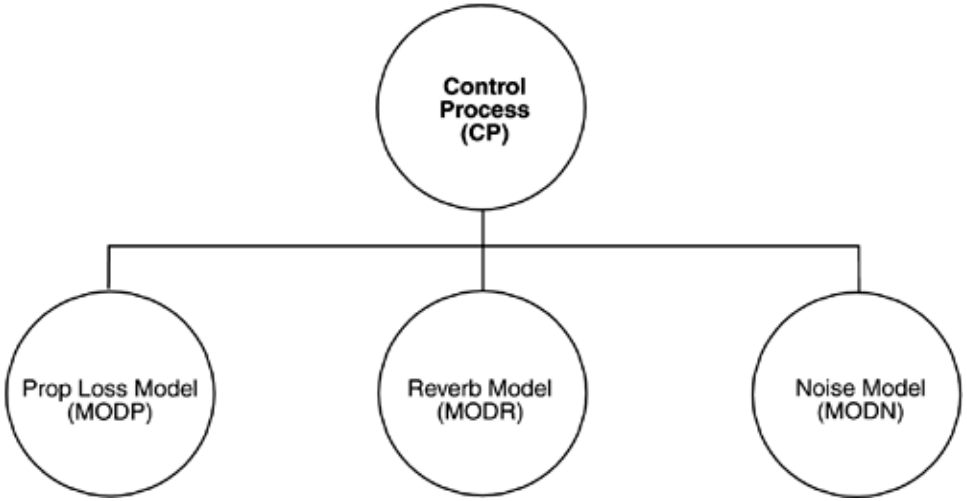
- The system consists of four elements.
- Three of the four elements may have something in common because they are positioned next to each other.
- All elements have some sort of relationship with each other since they are fully Connected
- Does the diagram show software architecture?



1.2.2 WHAT CANNOT BE DETERMINED FROM THIS DIAGRAM?

Nature of the elements

- What is the significance of their separation?
- Are they separate processes?
- Do they run on separate processors?
- Do they run at separate times?
- Are they objects, tasks, modules? Classes? Functions? What are they?

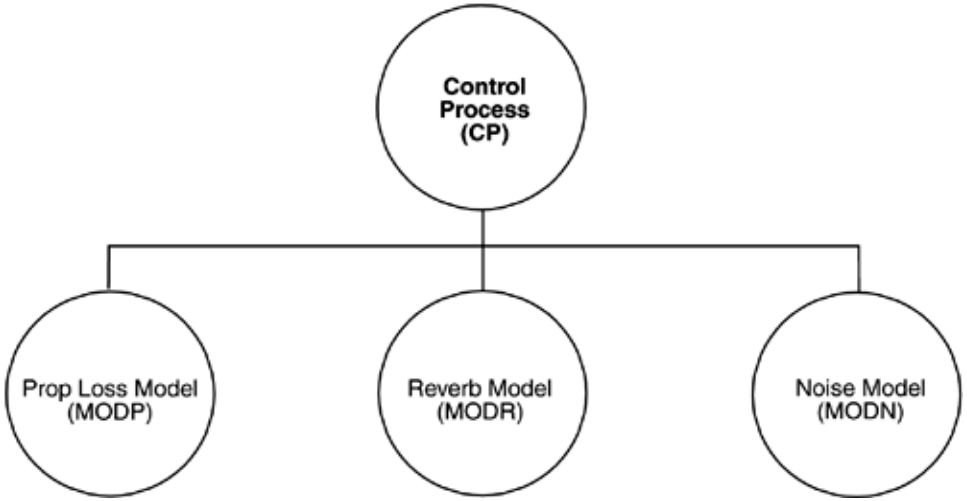


Responsibilities of the elements

- What are the functionalities of the elements?

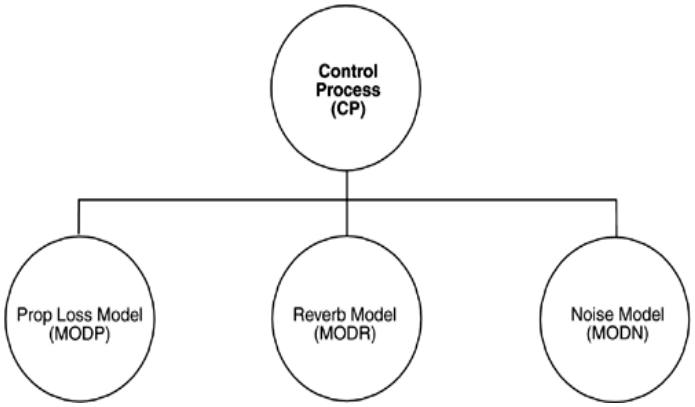
Significance of the connections

- Do the elements communicate with each other?
- Do they control each other?
- Do they send data and use each other?
- Do they invoke each other and share some information hiding secrets?



Significance of the layout

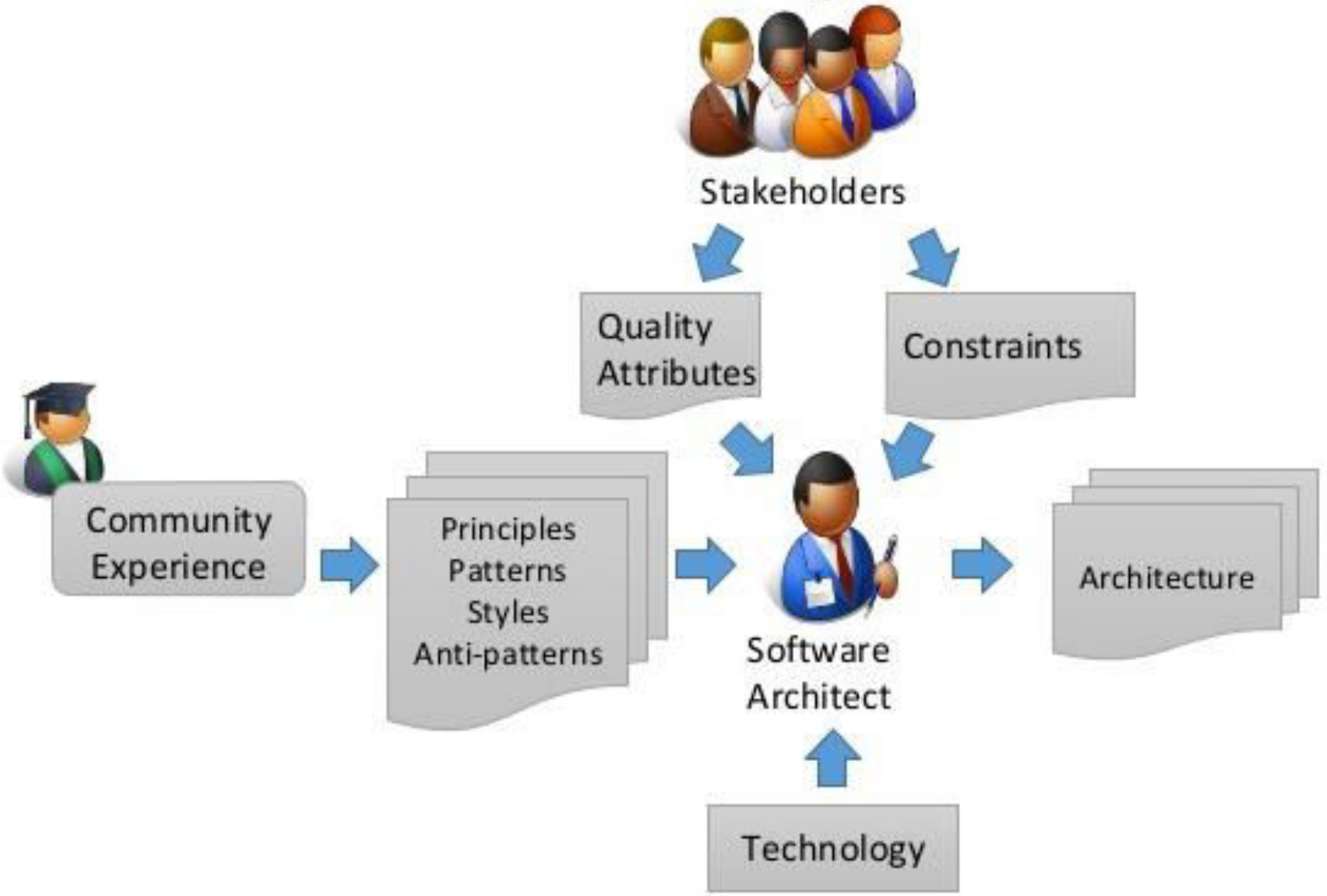
- Why is CP on a separate level?
- Does it call the other three elements?
- Are the others allowed to call it?



- Answers of all of the above questions are vital for the implementation team hence need to be discussed, defined and documented.

- Complete definition of **Architectural structures and views** are important.

Software architecting



2. STANDARD 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.*
- Primary building blocks were called "components," a term that has since become closely associated with the component-based software engineering movement
- "Element" was chosen here to convey something more general.

SOFTWARE ARCHITECTURE DEFINITIONS

Architectural Design: The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system. ***IEEE Glossary***

The Software Architecture of a program or a 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 in Practice, 2nd Edition, Len Bass, Paul Clements, Rick Kazman

The architecture of a system defines the system in terms of computational components and interaction between those components. Components are such things as clients and servers, databases, filters(compilers) and layers in a hierarchal system.

Software Architecture is the "structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time".

Garlan and Perry, guest editorial to the IEEE transactions on Software Engineering, April 1995

- **First**, architecture defines software elements. The architecture represents information about how the elements relate to each other.

- **Second**, the definition makes clear that systems can and do comprise more than one structure and that no one structure can finally claim to be the architecture.
- For example, all nontrivial(significant) projects are partitioned into implementation units; these units are given specific responsibilities and are frequently the basis of work assignments for programming teams.

- **Third**, the definition implies that every computing system with software has software architecture because every system can be shown to comprise elements and the relations among them.

- **Fourth**, the behavior of each element is part of the architecture insofar as that behavior can be observed or discerned(seperated) 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.

- This is another reason that the box-and-line drawings that are passed off as architectures are not architectures at all.

- Finally, the definition is indifferent as to whether the architecture for a system is a good one or a bad one, meaning that it will allow or prevent the system from meeting its behavioral, performance, and life-cycle requirements.

2.1 OTHER POINTS OF VIEW

- Software architecture is a growing but still young discipline; hence, it has no single, accepted definition. On the other hand, there is no shortage of definitions. A few of the most often heard definitions follow.

- Architecture is high-level design.

- Architecture is the overall structure of the system.

- 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.

- Architecture is components and connectors. Connectors imply a runtime mechanism for transferring control and data around a system. Thus, this definition concentrates on the runtime architectural structures.

3. ARCHITECTURAL STRUCTURES

- A view is a representation of a coherent(logical or valid) set of architectural elements, as written by and read by system stakeholders.

- It consists of a representation of a set of elements and the relations among them.

- A structure is the set of elements itself, as they exist in software or hardware.

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

3.1 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 manifest(visible) 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?

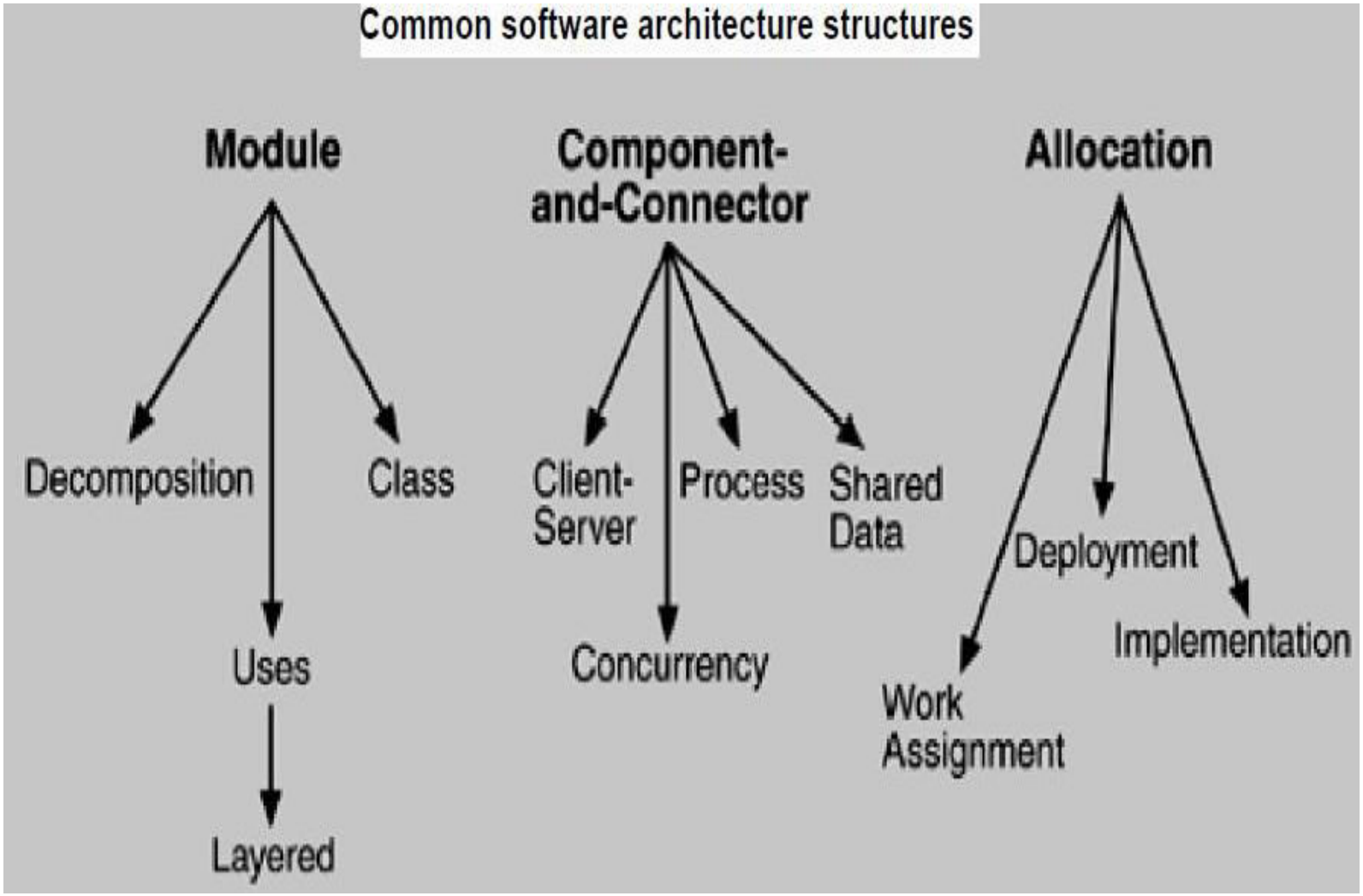
3.2 COMPONENT-AND-CONNECTOR STRUCTURES.

- 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?
    - How can the system's structure change as it executes?

3.3 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.

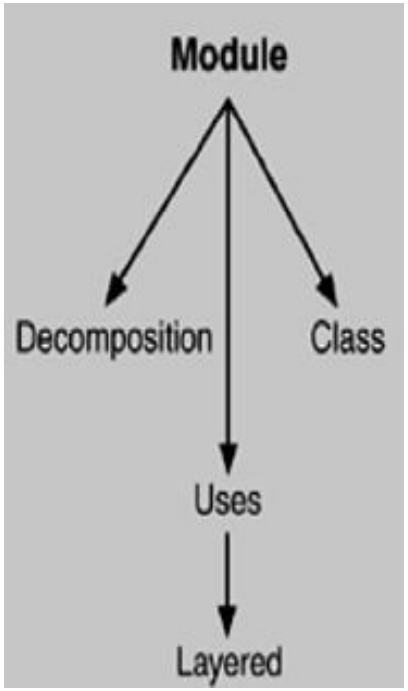
- They answer questions such as
  - What processor does each software element execute on?
  - In what files is each element stored during development, testing, and system building?
  - What is the assignment of software elements to development teams?


Common software architecture structures
Module
Component-
and-Connector
Allocation
Decomposition
Class
Uses
Layered
Client-
Server
Process
Shared
Data
Concurrency
Deployment
Implementation
Work
Assignment

- ***Module-based structures include the following structures.***

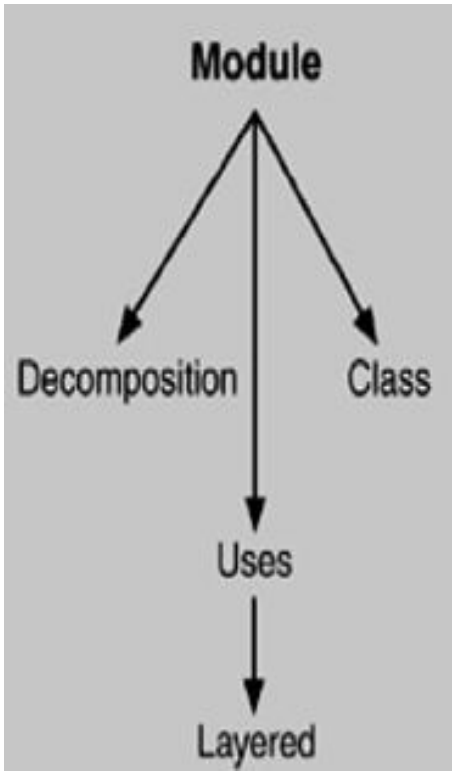
- **Decomposition:** The units are modules related to each other by the "is a submodule of " relation, showing how larger modules are decomposed into smaller ones recursively until they are small enough to be easily understood.

- **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.



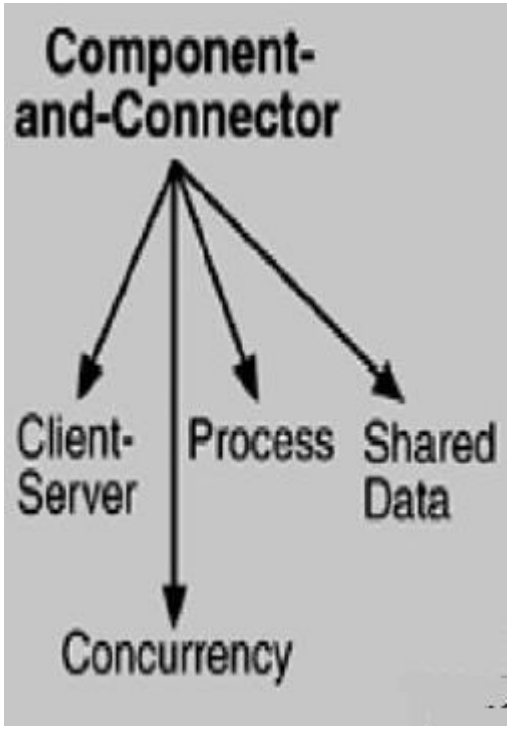
- **Layered:** Layers are often designed as abstractions (virtual machines) that hide implementation specifics below from the layers above, causing portability.

- **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.

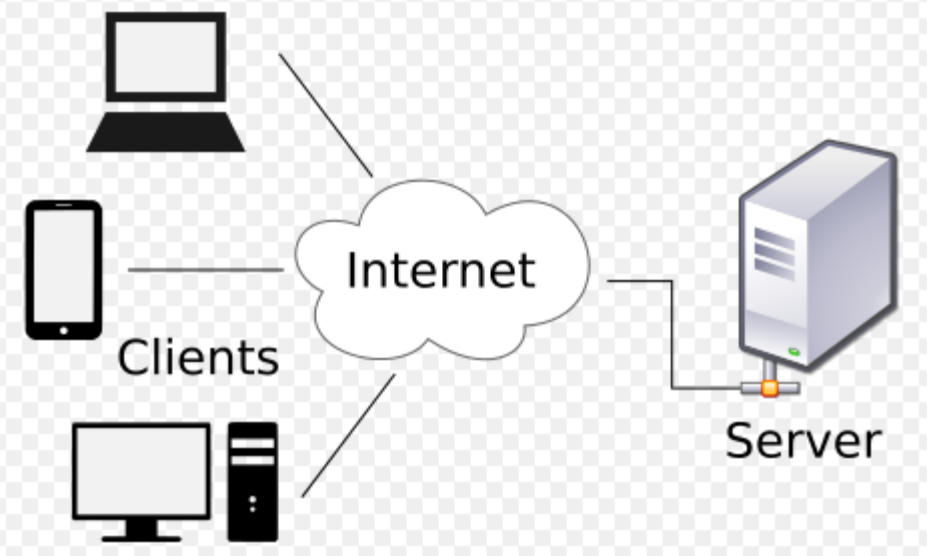


- A process is an executing instance of an application.
- A thread is a path of execution *within* a process.
- Software examples:

Concurrency

- Multiprogramming, or running a lot of programs concurrently (the O.S. has to multiplex their execution on available processors). For example:
  - downloading a file
  - listening to streaming audio
  - having a clock running
  - chatting

- **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

4. INFLUENCE OF SOFTWARE ARCHITECTURE ON ORGANIZATION-BOTH BUSINESS AND TECHNICAL

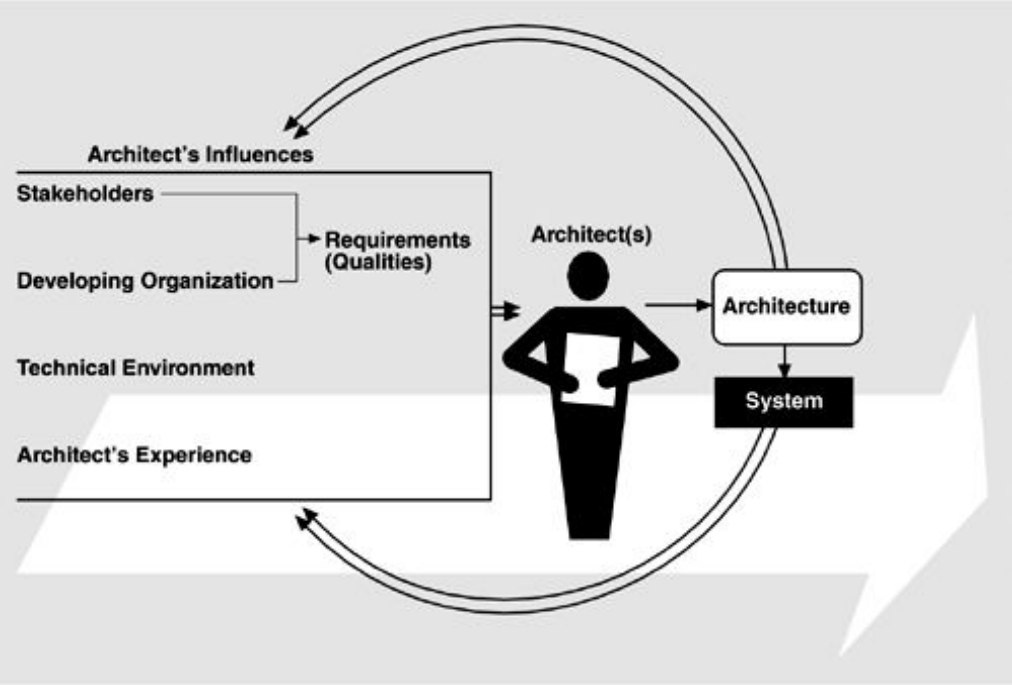
4.1 WHERE DO ARCHITECTURES COME FROM?

- 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
- Architectures are influenced by developing organizations
- Architectures are influenced by the background and experience of the architects
- Architectures are influenced by the technical environment



4.2 ARCHITECTURES ARE INFLUENCED BY SYSTEM STAKEHOLDERS

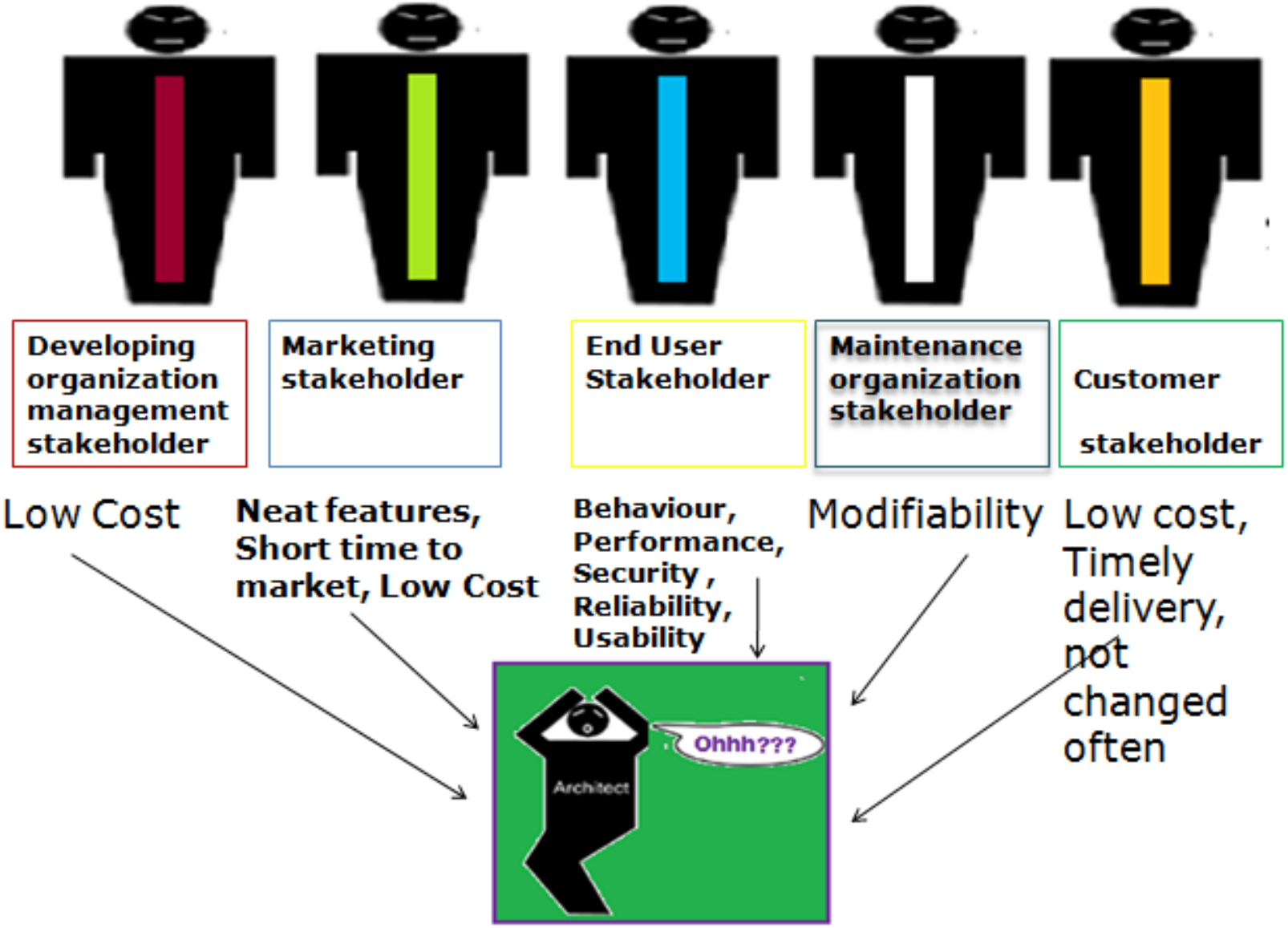
- Many people and organizations interested in the construction of a software system are referred to as stakeholders.
  - E.g. *customers, end users, developers, project manager* etc.

- 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.

- 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.

- Figure below shows the architect receiving helpful stakeholder "suggestions".



Influence of system stakeholders on Architect

4.3 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
  - 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 practice planned goals and may review the proposed system as one means of financing and extending that infrastructure.

- The organizational structure can shape the software architecture.

4.4 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.

- If their prior experience with this approach was unsuccessful, the architects may be hesitant 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.

4.5 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 established in the architect's professional community.

4.6 RAMIFICATIONS(OUTCOME) OF INFLUENCES ON AN ARCHITECTURE

- Influences on an architecture come from a wide variety of sources. Some are only implied, while others are explicitly in conflict(variance).
- Architects need to know and understand the nature, source, and priority of constraints on the project as early as possible.
- They must identify and actively engage the stakeholders to ask for 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.

5. 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 future architectures.

- This cycle of influences, from environment to the architecture and back to the environment, is called the Architecture Business Cycle (ABC).

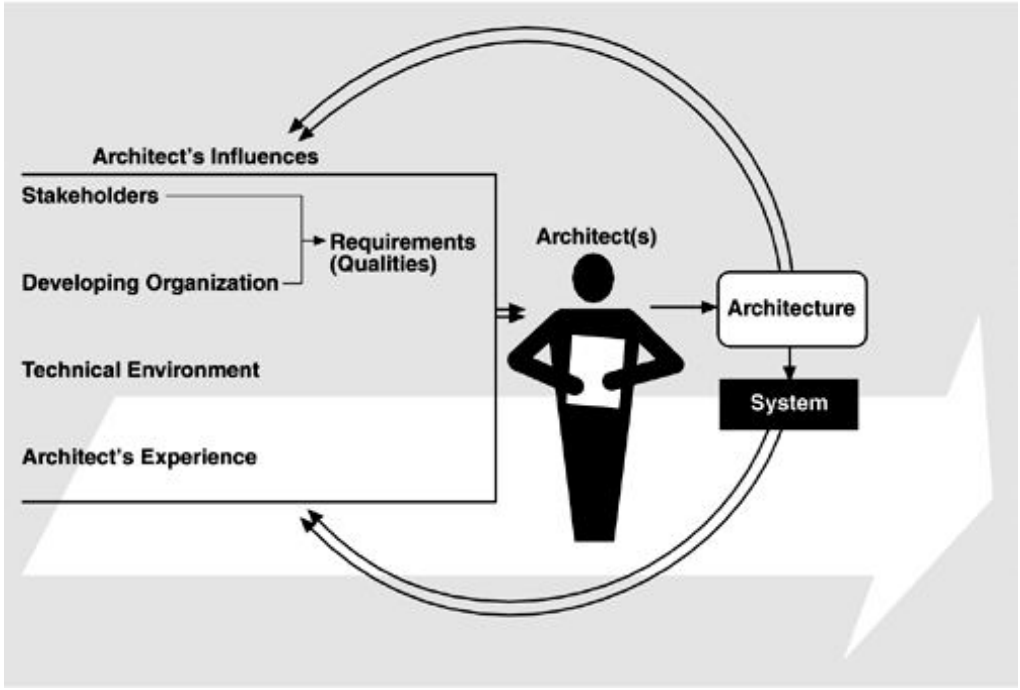
WORKING OF ARCHITECTURE BUSINESS CYCLE

- The architecture affects the structure of the developing organization.
- An architecture prescribes a structure for a system it particularly prescribes the units of software that must be implemented and integrated to form the system.
- Teams are formed for individual software units; and the development, test, and integration activities around the units.
- Schedules and budgets allocate resources in chunks corresponding to the units.

- Teams become embedded(fixed) in the organization's structure. This is feedback from the architecture to the developing organization.

- The architecture can affect the goals of the developing organization.
- A successful system built from it can enable a company to establish a foothold(grip) in a particular market area.

- The architecture can provide opportunities for the efficient production and deployment(use) 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.


Architect's Influences
Stakeholders
Developing Organization
Requirements (Qualities)
Technical Environment
Architect's Experience
Architect(s)
Architecture
System

- 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.
- The process of system building will affect the architect's experience with subsequent systems by adding to the corporate experience base.
- 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 architects are the professionals who develop the overall structure of software whereas a software engineer does its coding and modifications.

- Software architects have to communicate with the clients to understand about their requirements whereas a software engineer only works for what an architect instructs them.

5.1 ACTIVITIES INVOLVED IN CREATING SOFTWARE ARCHITECTURE

SOFTWARE PROCESSES AND THE ARCHITECTURE BUSINESS CYCLE

- *Software process* is the term given to the organization, reutilization, and management of software development activities.
- ***The various activities involved in creating software architecture are:***
    - Creating the business case for the system
    - Understanding the requirements
    - Creating or selecting the architecture
    - Documenting and communicating the architecture
    - Analyzing or evaluating the architecture
    - Implementing the system based on the architecture
    - Ensuring that the implementation conforms(obey the rules) to the architecture

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?
- Are there system limitations that it must work within?
- 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 extracting requirements from the stakeholders.
- For ex:
  - Object oriented analysis uses scenarios, or "use cases" to represent 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.
- 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 powerfully that conceptual integrity is the key to sound system design
- 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(clearly) 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(logical) 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(forced) by the architecture.

- Having an explicit and well-communicated architecture is the first step toward ensuring architectural conformance.

6. WHAT MAKES A "GOOD" ARCHITECTURE?/PROPERTIES OF A GOOD SOFTWARE ARCHITECTURE DESIGN

- 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?

- Divide our observations into two clusters: process recommendations and product (or structural) recommendations.

Process recommendations 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 expressed, 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-statistical analysis) 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(disagreement) areas, the resolution(decision) of which is clearly specified, circulated and maintained.

Product (structural) recommendations 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(policies) 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.

7. FUNCTIONAL REQUIREMENTS

WHAT DRIVES SOFTWARE ARCHITECTURE?

System requirements:

- **Functional needs** (what the system should do)
- **Quality needs** (properties that the system must possess such as availability, performance, security.,etc)

FUNCTIONAL REQUIREMENTS

- Functional requirements specify what the software needs to do. They relate to the **actions** that the product must carry out in order to satisfy the fundamental reasons for its existence.
- **Business Level**: defines the objective/goal of the project and the measurable business benefits for doing it.
- **User Level**: user requirements are written from the user's point-of-view.
- **System Level**: defines what the system must do to process input and provide the desired output.

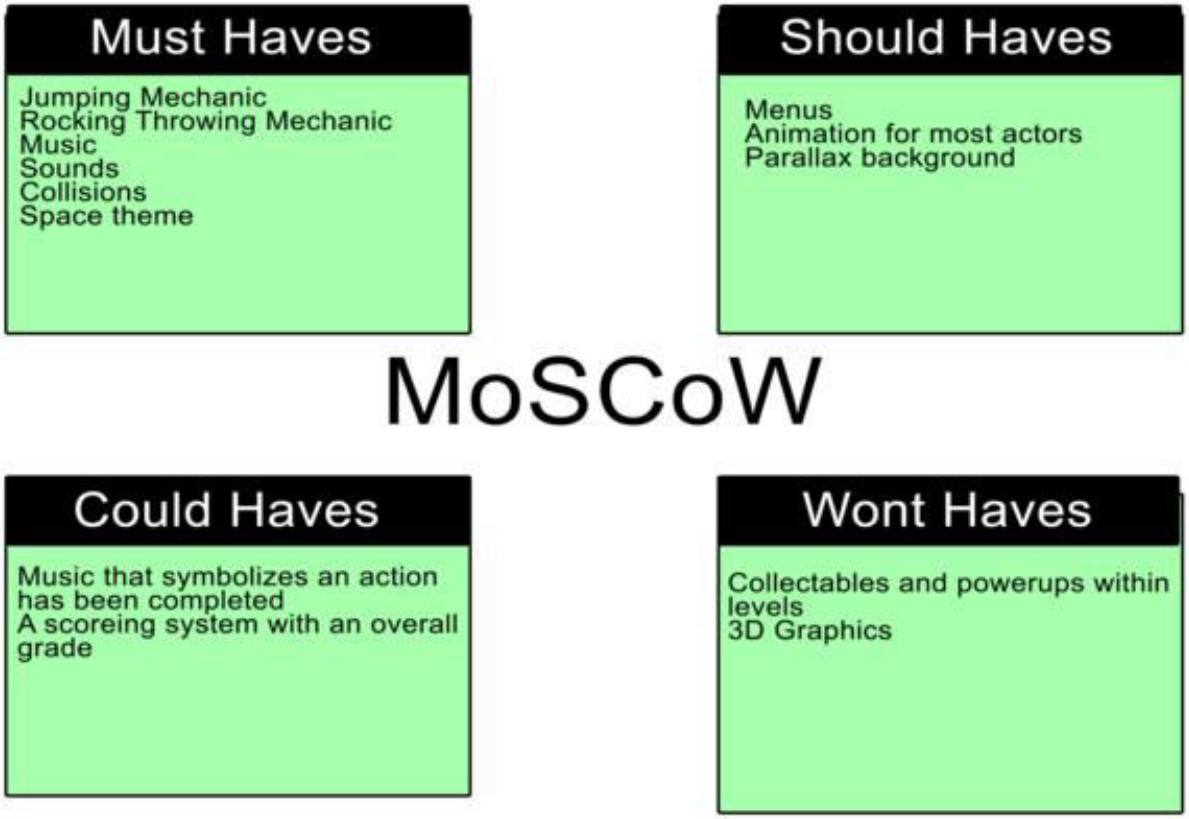
MOSCOW METHOD:

The **MoSCoW method** is a prioritization technique used in management, business analysis, project management, and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement - also known as ***MoSCoW prioritization*** **or** ***MoSCoW analysis*****.**

- **M - MUST**: Describes a requirement that must be satisfied in the final solution for the solution to be considered a success.

- **S - SHOULD**: Represents a high-priority item that should be included in the solution if it is possible. This is often a critical requirement but one which can be satisfied in other ways if strictly necessary.

- **C - COULD**: Describes a requirement which is considered desirable but not necessary. This will be included if time and resources permit.
- **W - WON'T**: Represents a requirement that stakeholders have agreed will not be implemented in a given release, but may be considered for the future.



FUNCTIONALITY AND SOFTWARE ARCHITECTURE

- It is the ability of the system or application to **satisfy the purpose** for which it was designed.
- It drives the **initial decomposition** of the system.
- It is the **basis** upon which all other quality attributes are specified.
- It is **related to quality attributes** like validity, correctness, interoperability, and security.
- Functional requirements often get the most focus in a development project.

8.QUALITY ATTRIBUTES

- Quality is a measure of excellence or the state of being free from deficiencies or defects.

- Quality attributes are the system properties that are separate from the functionality of the system.

- Implementing quality attributes makes it easier to differentiate a good system from a bad one.

- Attributes are overall factors that affect runtime behavior, system design, and user experience.

They can be classified as

Static Quality Attributes

- Reflect the structure of a system and organization, directly related to architecture, design, and source code.
- They are invisible to end-user, but affect the development and maintenance cost, e.g.: modularity, testability, maintainability, etc.

- **Functional requirement** defines a function of a system and its components. A function is described as a set of inputs, the behavior, and outputs.

- Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define *what* a system is supposed to accomplish.

- Functional requirements are supported by non-functional requirements (also known as *quality requirements*)

- **Non-functional requirement** is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.

Dynamic Quality Attributes

- Reflect the behavior of the system during its execution.
- They are directly related to system's architecture, design, source code, configuration, deployment parameters, environment, and platform.
- They are visible to the end-user and exist at runtime, e.g. throughput, robustness(strength), scalability, etc.

COMMON QUALITY ATTRIBUTES

- The following table lists the common quality attributes software architecture must have

Category	Quality Attribute	Description
Design Qualities	Conceptual Integrity	Defines the consistency and coherence of the overall design.
	Maintainability	Ability of the system to undergo changes with a degree of ease.
	Reusability	Defines the capability for components and subsystems to be suitable for use in other applications.
Run-time Qualities	Interoperability	Ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties.
	Manageability	Defines how easy it is for system administrators to manage the application.
	Reliability	Ability of a system to remain operational over time.

	Scalability	Ability of a system to either handles the load increase without impacting the performance of the system or the ability to be readily enlarged.
	Security	Capability of a system to prevent malicious or accidental actions outside of the designed usages.
	Performance	Indication of the responsiveness of a system to execute any action within a given time interval.
	Availability	Defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period.
System Qualities	Supportability	Ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
	Testability	Measure of how easy it is to create test criteria for the system and its components.
User Qualities	Usability	Defines how well the application meets the requirements of the user and consumer by being intuitive.
Architecture Quality	Correctness	Accountability for satisfying all the requirements of the system.

Non-runtime Quality	Portability	Ability of the system to run under different computing environment.
	Integrality	Ability to make separately developed components of the system work correctly together.
	Modifiability	Ease with which each software system can accommodate changes to its software.
Business quality attributes	Cost and schedule	Cost of the system with respect to time to market, expected project lifetime & utilization of legacy.
	Marketability	Use of system with respect to market competition.

9. ARCHITECTURAL MODEL, REFERENCE MODEL & REFERENCE ARCHITECTURE

ARCHITECTURAL MODEL(PATTERN)

- An architectural pattern is a description of element and relation types together with a set of constraints on how they may be used.
- A pattern can be thought of as a set of constraints on an architecture-on the element types and their patterns of interaction-and these constraints define a set or family of architectures that satisfy them.
- For example, ***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.

- Use of the term client-server implies only that multiple clients exist; the clients themselves are not identified, and there is no discussion of what functionality, other than implementation of the protocols, has been assigned to any of the clients or to the server.

- Countless architecture is of the client-server pattern under this (informal) definition, but they are different from each other.

- An architectural pattern is not an architecture, then, but it still conveys a useful image of the system-it imposes useful constraints on the architecture and, in turn, on the system.

- One of the most useful aspects of patterns is that they exhibit(show) known quality attributes.

- This is why the architect chooses a particular pattern and not one at random.

- Some patterns represent known solutions to performance problems, others lend themselves well to high-security systems, still others have been used successfully in high-availability systems.

- Choosing an architectural pattern is often the architect's first major design choice.

- The term *architectural style* has also been widely used to describe the same concept.

REFERENCE MODEL

- 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.

- Reference models are a characteristic of mature domains, E.g. Compiler, DBMS

REFERENCE ARCHITECTURE

- 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.

- The mapping may be, but by no means necessarily is, one to one. A software element may implement part of a function or several functions.

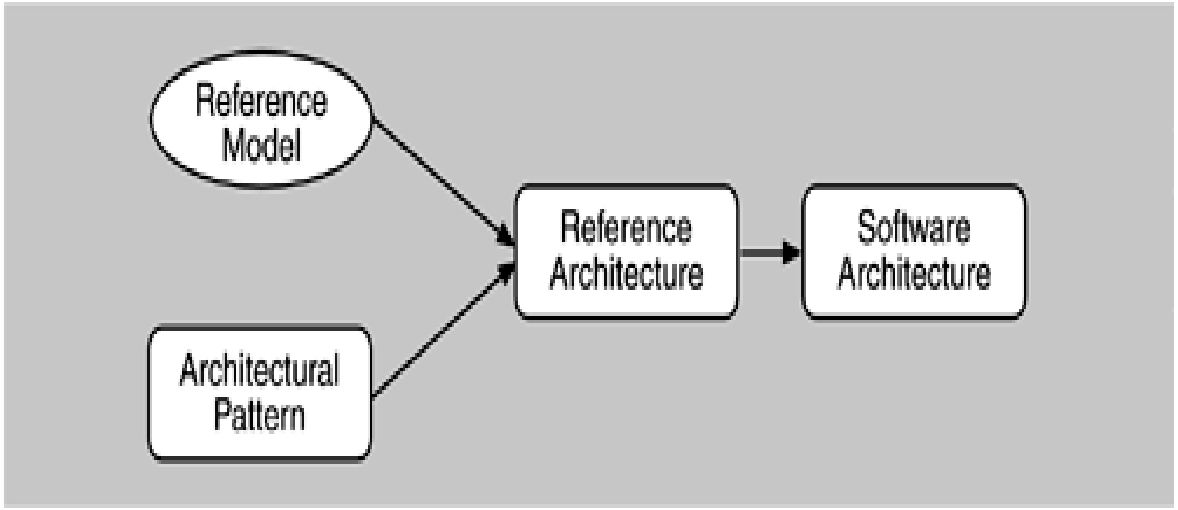
RELATIONSHIPS

Reference model + architectural pattern => reference architecture
=> software architecture

- 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 below.



10. WHY SOFTWARE ARCHITECTURE IS IMPORTANT

- Communication among stakeholders
- Early design decisions
- Transferable abstraction(concept) of a system
- Less is more: It pays to restrict the vocabulary of design alternatives
- An architecture permits template-based development
- An architecture can be the basis for training

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(agreement), and communication.

Early design decisions

- Software architecture manifests the earliest design decisions about a system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system's remaining development, its deployment, and its maintenance life.

- It is also the earliest point at which design decisions governing the system to be built can be analyzed.
  - The architecture defines constraints(limitation) on implementation
  - The architecture inhibits or enables a system's quality attributes
  - Predicting system qualities by studying the architecture
  - The architecture makes it easier to reason about and manage change
  - The architecture helps in evolutionary prototyping
  - The architecture enables more accurate cost and schedule estimates

Transferable abstraction of a system

- Software architecture constitutes a relatively small, intellectually graspable(logical) model for how a system is structured and how its elements work together, and this model is transferable across systems.
- In particular, it can be applied to other systems exhibiting similar quality attribute and functional requirements and can promote large-scale re-use.
  - Software product lines share a common architecture
  - Systems can be built using large, externally developed elements

11. HYBERTSSON'S THREE VIEWS FOR SOFTWARE ARCHITECTURE WITH AN EXAMPLE

Three things that Hybertsson could have used:

- Case studies of successful architectures crafted(ability) to satisfy demanding requirements, so as to help set the technical playing field of the day.

- Methods to assess(judge) an architecture before any system is built from it, so as to mitigate(reduce) the risks associated with launching unprecedented designs.

- Techniques for incremental architecture-based development, so as to uncover(find out) design flaws before it is too late to correct them.

THE WAR SHIP VASA