UNIT-III ARCHITECTURAL VIEWS

- Introduction
- Standard Definitions for views
- Structures and views
- Representing views
- available notations
- Standard views
- 4+1 view of RUP
- Siemens 4 views
- SEI's perspectives and views
- Case studies

1.INTRODUCTION

- In software, the different drawings are called views.
- A view represents the system as composed of some types of elements and relationships between them.
- Which elements are used by a view, depends on what the view wants to highlight.
- Different views expose different properties and attributes, thereby allowing the stakeholders and analysts to properly evaluate those attributes for the system.
- A view describes a structure of the system.

- Two concepts— views and structures—used interchangeably.
- Use the term architectural view to refer to a view.
- Many types of views have been proposed.
- Most of the proposed views generally belong to one of these three types.
    - Module
    - Component and connector
    - Allocation

ARCHITECTURAL VIEW

- It is not possible for us to describe everything related to systems architecture in a single architectural model

- Divide the system architecture in different architectural views with each view representing a different perspective of the system.

- An architectural view is a way to portray those aspects or elements of the architecture that are relevant to the concern the view intends to address—and, by implication, the stakeholders for whom those concerns are important.

- In 1995 Phillipe Kruchten of the Rational Corporation published his widely accepted written description of viewpoints, **Architectural Blueprints—The 4+1 View Model of Software Architecture .**

- This suggested four different views of a system and the use of a set of scenarios (use cases) to check their correctness.

- More recently, IEEE Standard 1471 has formalized these concepts and brought some welcome standardization of terminology.

- Definition of a view is based on and extends the one from the IEEE standard.

2. STANDARD DEFINITIONS FOR VIEWS

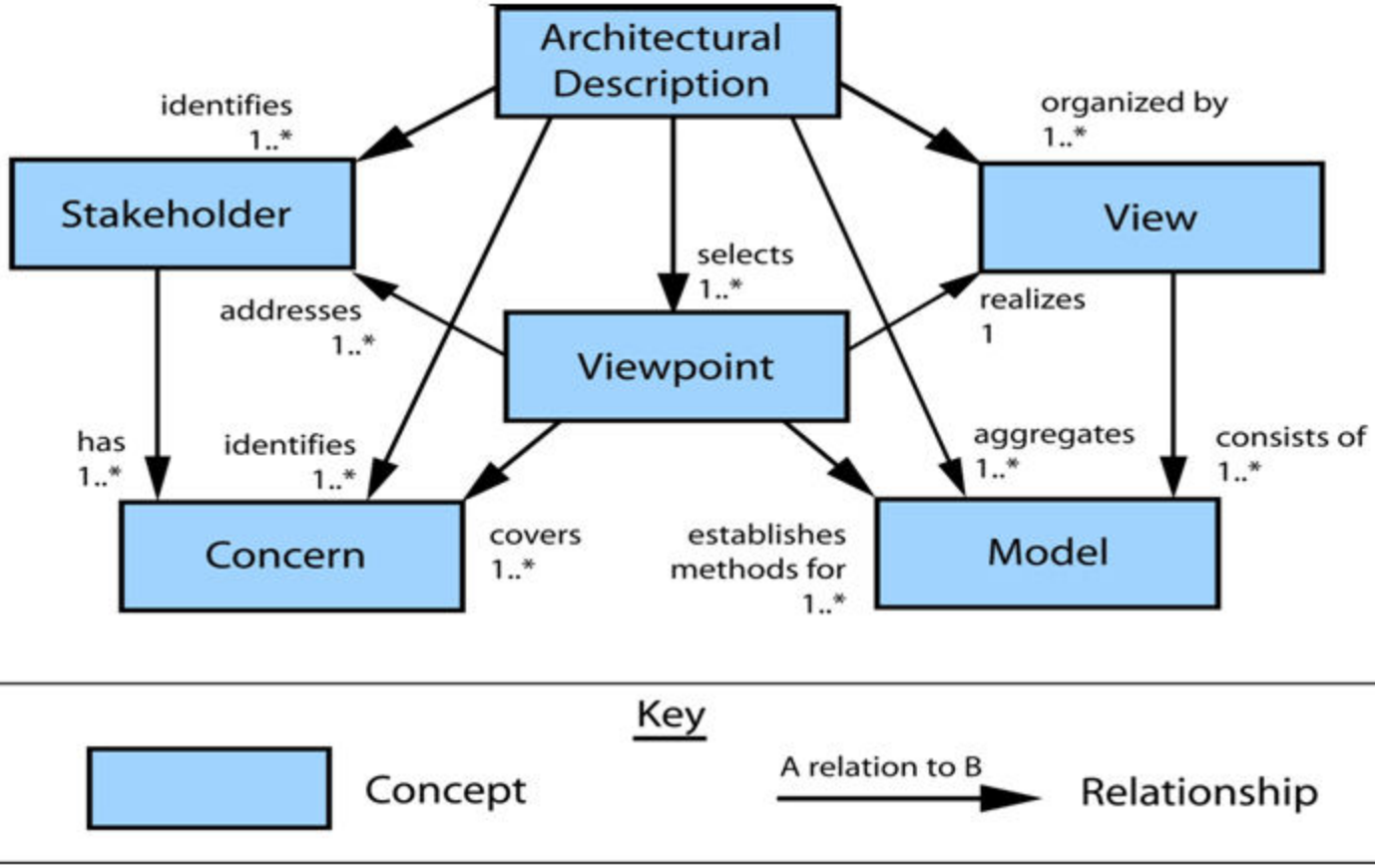
IEEE Standard 1471

A **viewpoint** is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views.

A view is a representation of a set of system elements and relations among them – not all system elements, but those of a particular type.

For example, a layered view of a system would show elements of type "layer" – that is, it would show the system's decomposition into layers – and the relations among those layers.

IEEE Standard 1471-2000



When deciding what to include in a view, ask yourself the following questions.

- What class(es) of stakeholder is the view aimed at?
- How much technical understanding do these stakeholders have?
- What stakeholder concerns is the view intended to address?
- How much do the stakeholders know about the architectural context and background to these concerns?
- How much do these stakeholders need to know about this aspect of the architecture?
- For non technical stakeholders such as users, how competent are they in understanding its technical details?

- Representing software architecture in multiple **architectural views**.

- Each architectural view addresses some specific set of concerns, specific to stakeholders in the development process: end users, designers, managers, system engineers, maintainers, and so on.

- The views capture the major structural design decisions by showing how the software architecture is decomposed into components, and how components are connected by **connectors** to produce useful **forms.**

- These design choices must be tied to the requirements, functional, and supplementary, and other constraints.

3.STRUCTURES AND VIEWS

- Structures is a set of coherent(logical) elements and the relations among them.

- For each structure these we can specify:
  - Types of elements
  - Types of relations
  - A set of syntactic constraints
  - Semantics of the diagram
  - Rationale(basis), principles, and guidelines
  - For what purposes it is useful

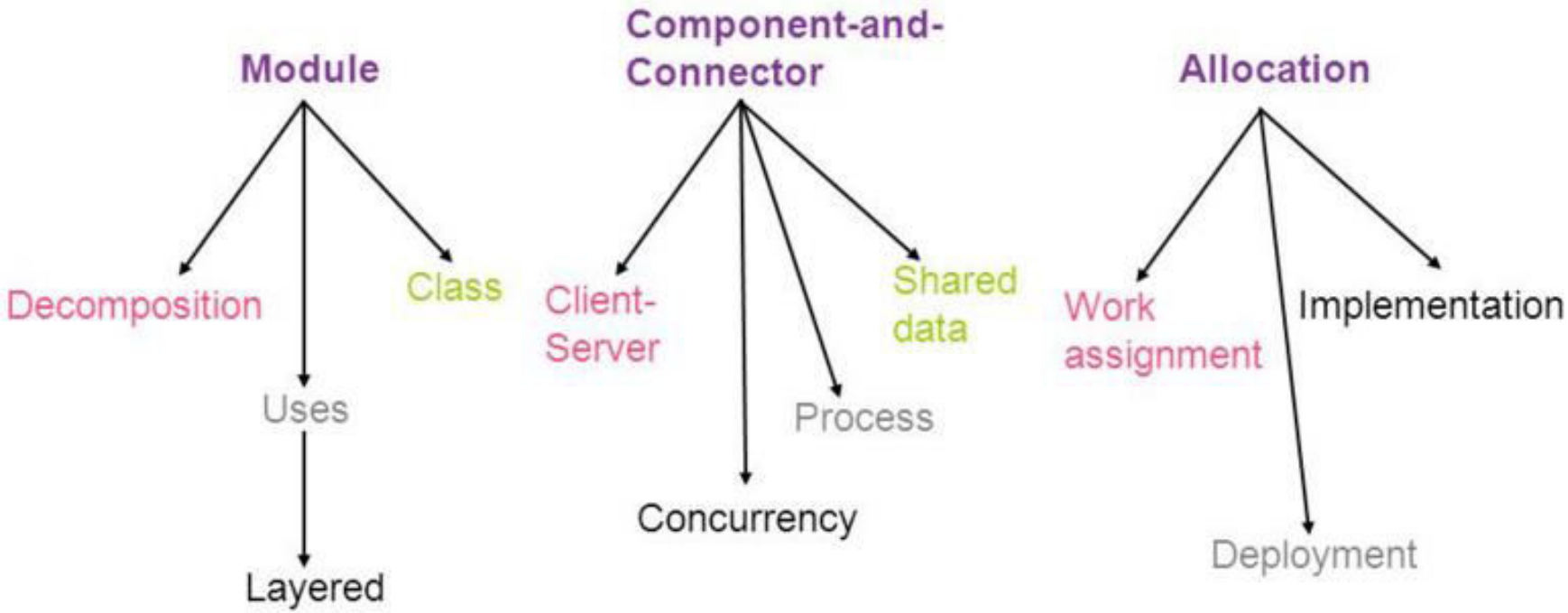
- View is a representation of software architecture based on an structure as written by the architect and read by stakeholders (an instance(example) of the structure)

- SA is documented by a number of views.

Categorization of Structures

1. Module Structures
2. Component and Connector Structures
3. Allocation Structures

- **View ::** representation of a coherent set of architectural elements, as written by and read by system stakeholders
  - **Composed of:** Elements and relation among them
- **Structure ::** set of elements itself
  - Module {code}
  - Component-and-connector
  - Allocation
- Ex: module structure x module view


Module
Decomposition
Class
Uses
Layered
Component-and-Connector
Client-Server
Shared data
Process
Concurrency
Allocation
Work assignment
Implementation
Deployment

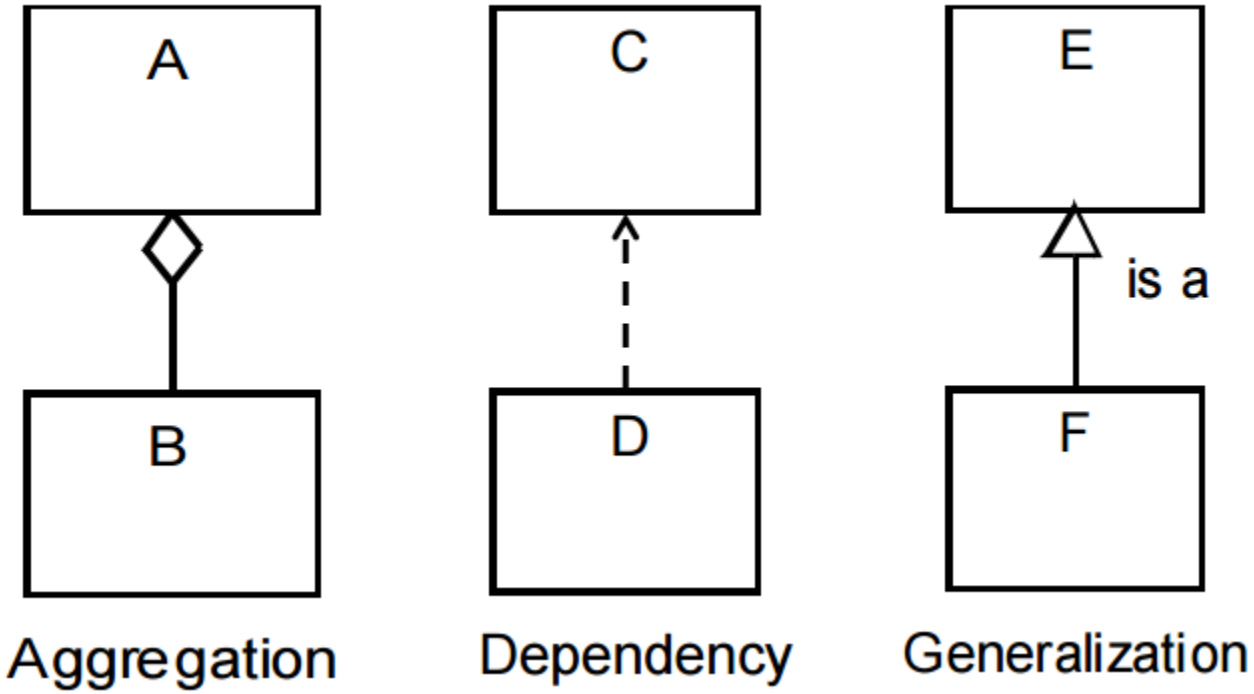
one system, many views

- an architect examines the system in three ways
  - how is it structured as a set of code units?
    - module viewtype
  - how is it structured as a set of elements that have run-time behavior and interactions?
    - component & connector viewtype
  - how does it relate to non-software structures, such as hardware and development teams?
    - allocation viewtype

3.1 MODULE VIETYPE

- Module view type describes the code structure
- Elements are modules
- Code unit that implements a set of functionalities
- Relations among modules include
- A is part of B
  - defines a part-whole relation
- A depends on B
  - defines a functional dependency relation
- A is a B
  - defines specialization and generalization

- UML class diagrams:



- Module view type is used for code construction and budgeting

Construction

- module views are the blueprints for the code
- modules are assigned to teams for implementation
- modules are often the unit for refining the design (e.g., module interfaces)

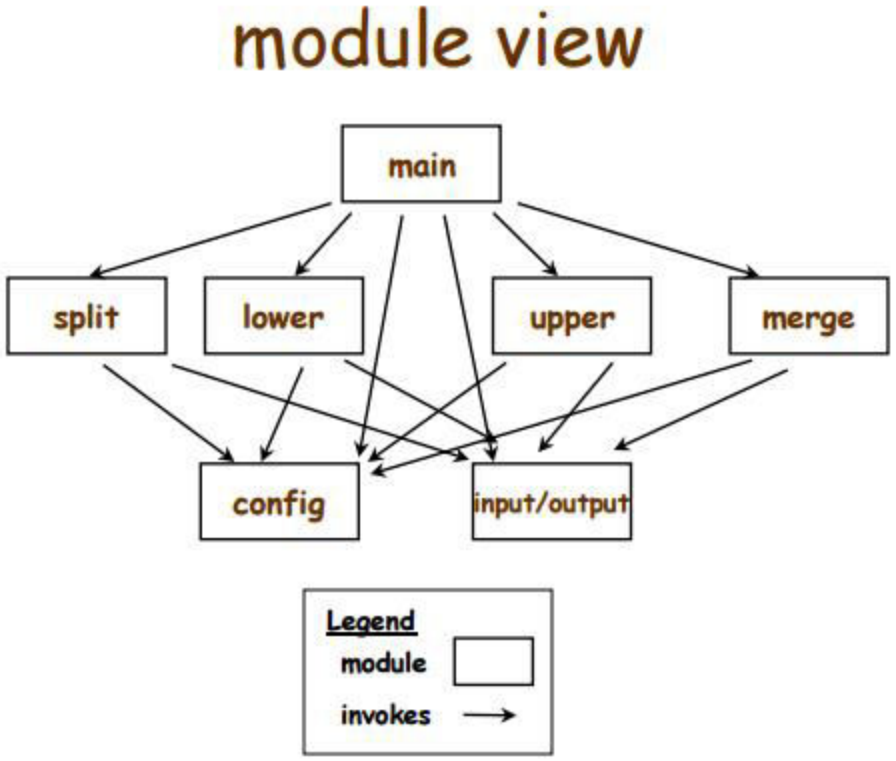
Analysis

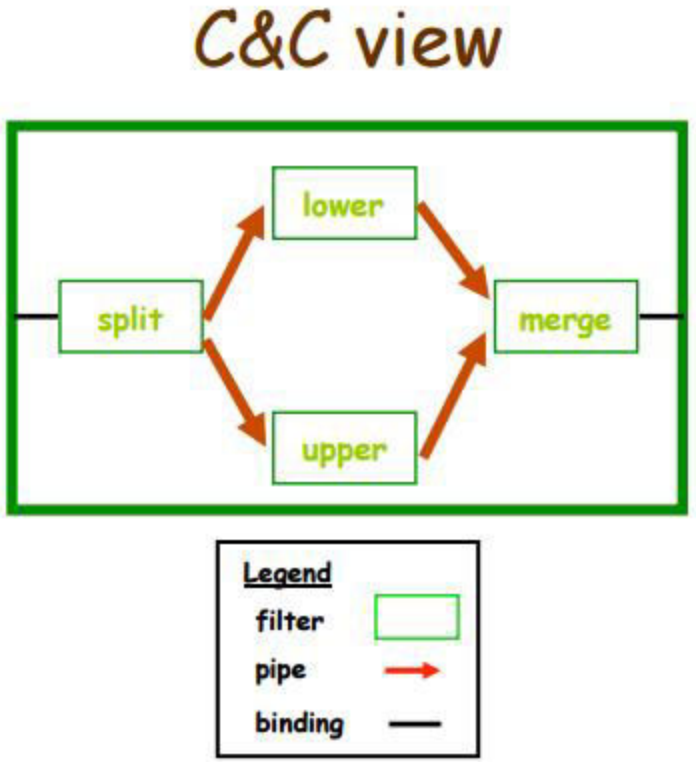
- Traceability and impact analysis
- Budgeting, project management: planning and tracking

Example program:

- Produce alternating case of characters in a stream

MODULE AND C&C SHOW DIFFERENT ASPECTS





3.2 C&C VIEWTYPE

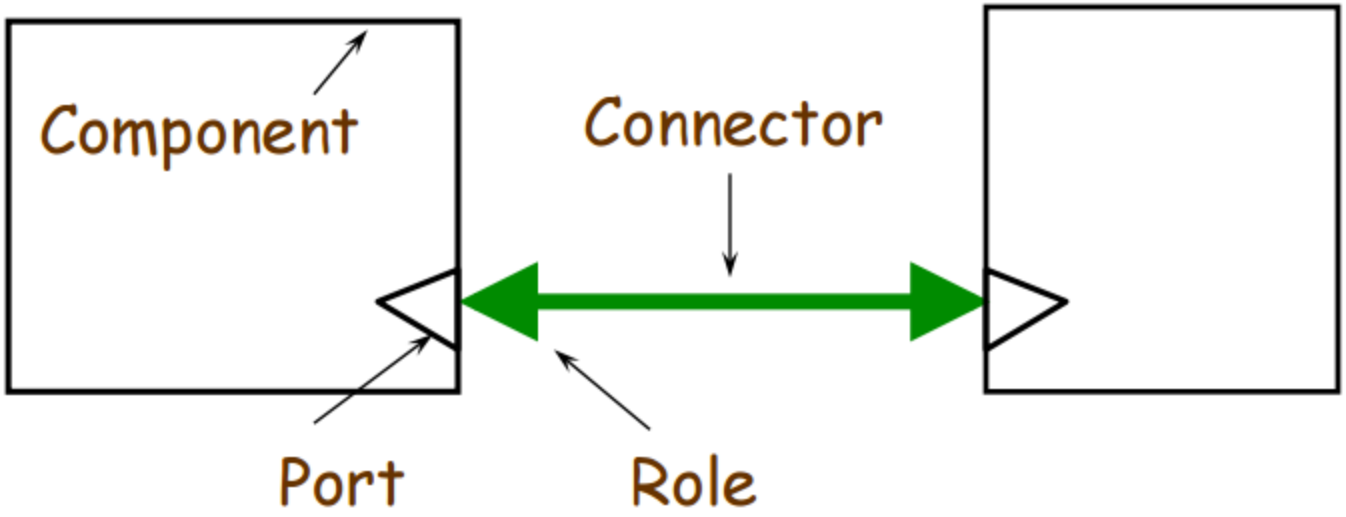
- Describes how the system works

Elements

- components (boxes)
  - principal units of run-time computation and data stores
- connectors (lines)
  - interaction mechanisms – identity and behavior of their own
- Relations
  - attachment of components to connectors
- Properties
  - information for construction & analysis quality attributes others, depending on style

Different notations exist for C&C views

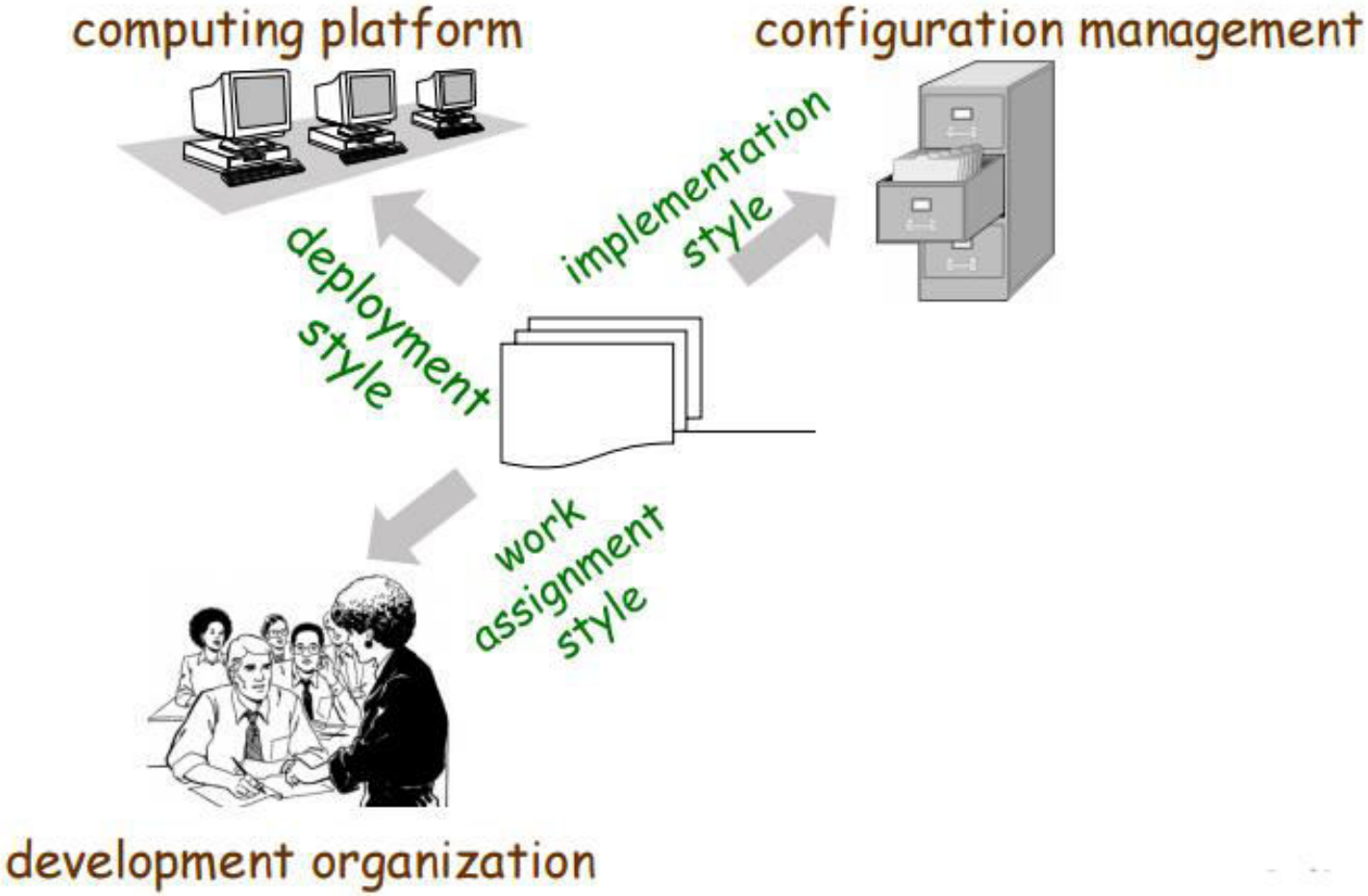
- ACME diagrams
- Notations (normally box-and-line)



- C&C view type used for behavior and QoS analysis
- Construction
  - how the system will appear at run time
  - what kind of behavior must be built in
  - pathways of interaction and communication mechanisms
- Analysis of runtime properties
  - availability
  - performance
  - security
  - reliability…

3.3 ALLOCATION VIEW TYPE

- Elements
  - software elements as defined in module or C&C views
  - Environment elements such as hardware and development teams
- Relations
  - allocated-to
- Notations
  - for allocation views depend on the style


computing platform
configuration management
implementation style
deployment style
work assignment style
development organization

4.REPRESENTING VIEWS

- Viewpoint models utilize perspectives to separate the concerns.
- Perspectives are variously called views, view types, and viewpoints.
- Different viewpoint models focus on the different uses of documentation.
  - Communication.
  - Design.
  - Re-use

- A view is a subset of a model that abstracts a specific, relevant perspective.

- Architecture is represented by a number of different architectural views, which in their essence are extracts illustrating the "architecturally significant" elements of the models.

- In November 1995, while working as Lead software architect at Hughes Aircraft Of Canada Philippe Kruchten published a paper

- It is entitled: "Architectural Blueprints—The “4+1” View Model of Software Architecture".

- The intent was to come up with a mechanism to separate the different aspects of a software system into different *views* of the system.

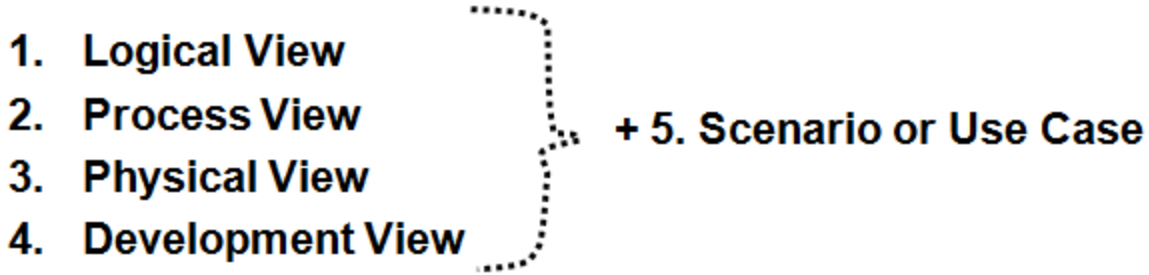
- Why? Because different stakeholders always have different interest in a software system.

- Some aspects of a system are relevant to the Developers; others are relevant to System administrators.

- The Developers want to know about things like classes; System administrators want to know about deployment, hardware and network configurations and don't care about classes.

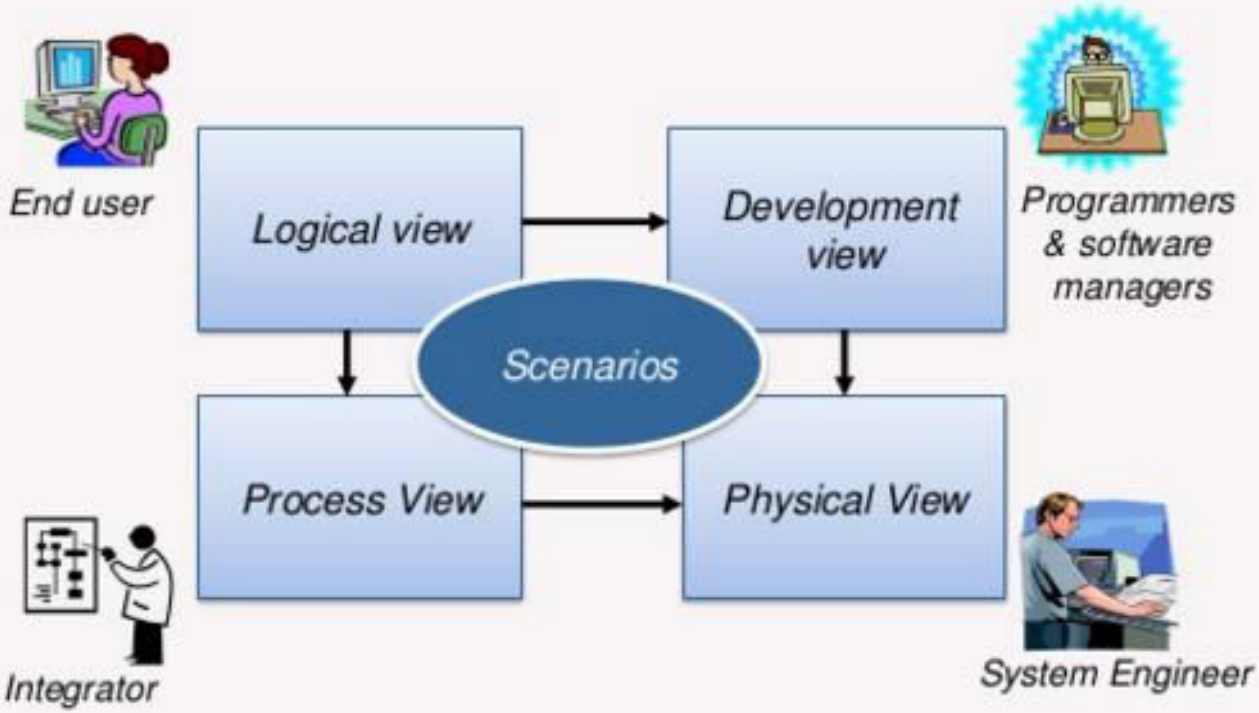
- Similar points can be made for Testers, Project Managers and Customers.

- Kruchten thought it made sense to decompose architecture into distinct views so stakeholders could get what they wanted.
- In total there were 5 views in his approach but he decided to call it 4 + 1.
- why it's called 4 + 1?



In addition selected use cases or scenarios are utilized to illustrate the architecture serving as the **'plus one'** view.

The 4+1 View model



- **Describes software architecture using five concurrent views.**

- Framework
  - based on IEEE 1471-2000, IEEE (2000).
- Viewpoint Models:
  - “4+1” View Model, Kruchten, P. (1997).
  - SEI View Model, Clements, P. et al. (2002b).
  - ISO RM-ODP, ISO (1994).
  - Siemens Four View Model, Soni, D. et al. (1995).
  - Rational ADS, Norris, D. (2004).

Models selected are general viewpoint models:

1) Kruchten's "4+1" view model.

Incorporated into Rational ADS.

Includes a **iterative design process**.

2) Software Engineering Institute of Carnegie Mellon University.

Relatively **independent viewpoints**.

Focus on documentation for various stakeholders.

3) International Standards Organization.

Reference Model of Open Distributed Processing.

Focus on **Interoperability** of systems.

Explicit mention of only **Developers** and **Standards writers**.

No defined translations between viewpoints.

4) Siemens Corporation.

Focus on **design process**.

Explicitly for **Architects** to **design** software architecture.

5.NOTATIONS

- There are many notations to specify software architectures, which address a wide range of formality and completeness.

- Completely formal notations produce accurate and analyzable software architecture specifications

- Types of Notations
  - Informal notations
  - Semiformal notations (e.g. UML)
  - Formal notations (e.g. Architecture Description Language (ADL))

INFORMAL NOTATIONS.

- Views are depicted (often graphically) using general-purpose diagramming and editing tools and visual convention(rule) chosen for the system at hand.

- The semantics of the description are characterized in natural language and cannot be formally analyzed.

- Natural (human) languages, such as English, are expressive, but ambiguous, non-formal, and non-rigorous(exact).

- They cannot be effectively processed and understood by machines, and so can only be inspected by humans.

- Natural languages are easily accessible to stakeholders and can be handled using common word-processing tools.

Advantages

- Highly expressive
- Accessible to all stakeholders
- Good for capturing non-rigorous or informal architectural elements like rationale(basis) and non-functional requirements
- Plentiful tools available (word processors and other text editors)

Disadvantages

- Ambiguous, non-rigorous, non-formal
- Often verbose(wordy)
- Cannot be effectively processed or analyzed by machines/software

SEMIFORMAL NOTATIONS

- Views are expressed in a standardized notation that prescribes graphical elements and rules of construction

- It does not provide a complete semantic treatment of the meaning of those elements.

- Rudimentary(simple) analysis can be applied to determine if a description satisfies syntactic properties.

- Unified Modeling Language (UML) is a semiformal notation in this sense.

UML

- 13 loosely-interconnected notations called diagrams that capture static and dynamic aspects of software-intensive systems
- UML (Booch, Rumbaugh, and Jacobson 2005) combines concepts from earlier notations: Booch diagrams, OMT, OOSE, and Statecharts.
- UML provides a wide variety of modeling constructs and viewpoints.
- There are many tools that provide varying degrees of support for UML (from diagram construction to generation of code from UML models).
- UML is an extensive notation that traditionally employs graphical symbols with textual annotations to represent the system from thirteen different viewpoints (as of UML 2.0).
- Early versions of UML focused on detailed design modeling. UML 2.0 has added support for higher-level architectural constructs.

Advantages

- Support for a diverse array of viewpoints focused on many common software engineering concerns.
- Ubiquity(present everywhere) improves clarity.
- Extensive documentation and tool support from many vendors.

Disadvantages

- Needs customization through profiles to reduce ambiguity.
- Difficult to assess consistency(uniformity) among views.
- Difficult to capture foreign concepts or views.

FORMAL NOTATIONS

- Views are described in a notation that has a precise (usually mathematically based) semantics.
- Formal analysis of both syntax and semantics is possible.
- There are a variety of formal notations for software architecture available, although none of them can be said to be in widespread use.
- Architecture Description Languages (ADLs)

- There is no single definition of what constitutes an architecture description language.
- The uncertainty (doubt) regarding this issue turn on disagreement on what should and should not be included in the concept of software architecture.
- Looking at architecture as a set of principal design decisions about a system, there are several languages that fall into the category of ADLs.
- The early ADLs did not go past the research project stage and are not used in practice.
- However, they are an important part of architectural modeling development.
- A brief feature overview is provided below for three early ADLs: Darwin, Rapide, and Wright.

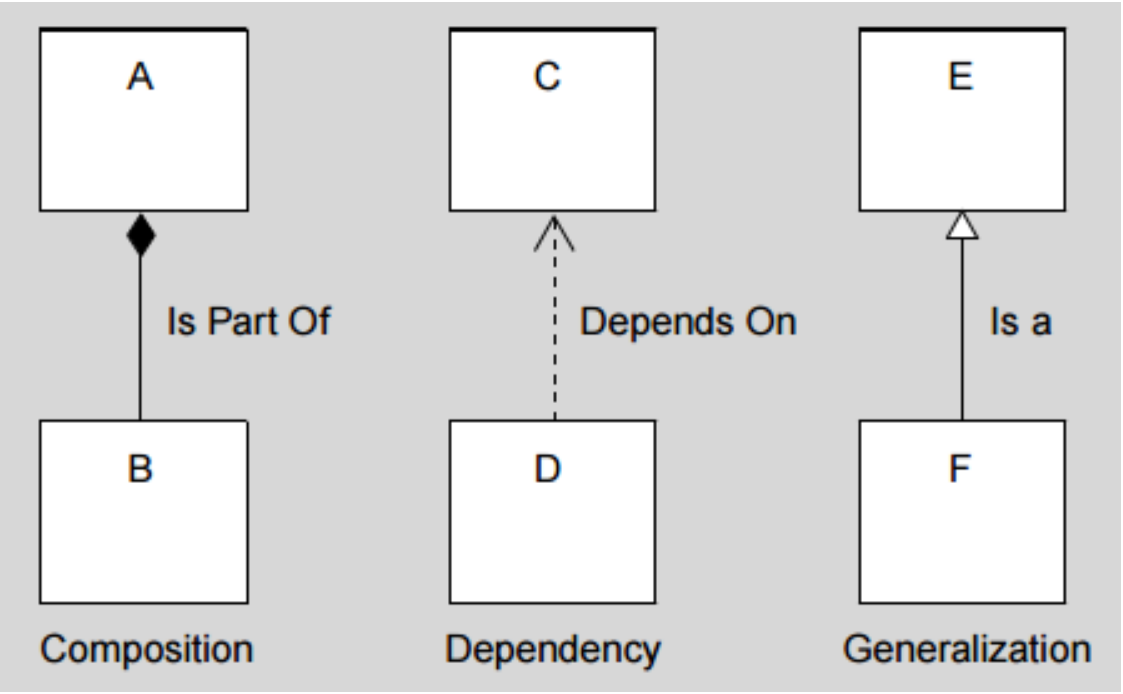
5.1 NOTATION FOR MODULE VIEW TYPE

Informal:

- box-and-line;
- nesting can represent “is part of” relations

Semi-formal:

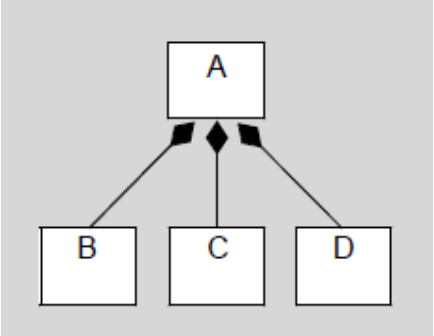
- UML, class diagrams



STYLES IN THE MODULE VIEWTYPE

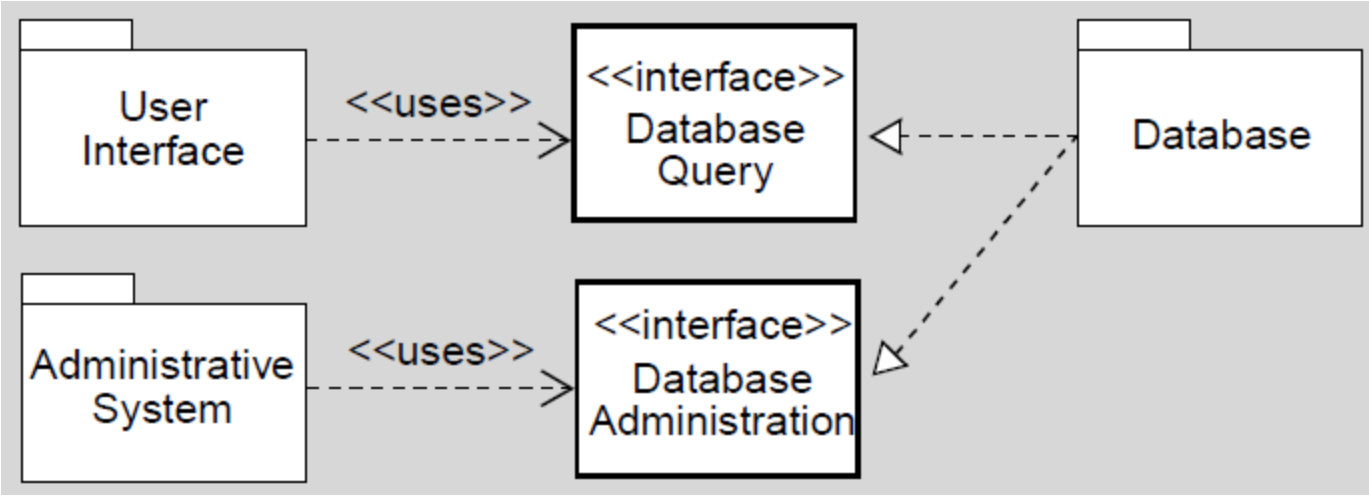
Decomposition style: **Documents how system** responsibilities are partitioned across modules and how those modules are decomposed into sub modules

Relations: **is part of**



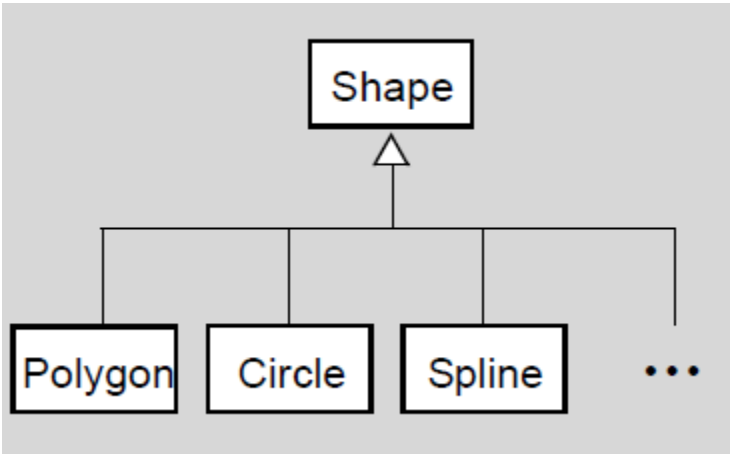
- **Uses style:** **Tells the developer** what other modules must exist for this portion of the system to work correctly

Relations: uses, a refined form of "depends on" relation



- **Generalization style:** Documents the **"*is a*"** relations among elements of the system

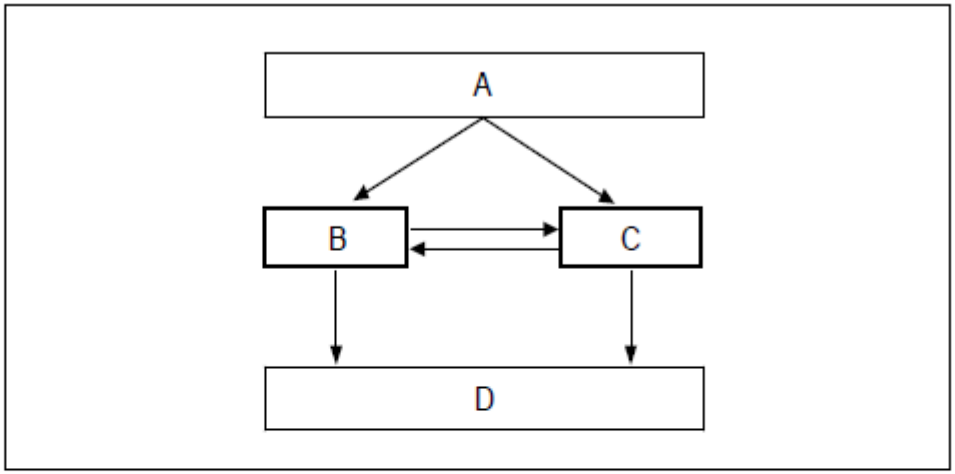
  Relations: generalization, an "is a" relation



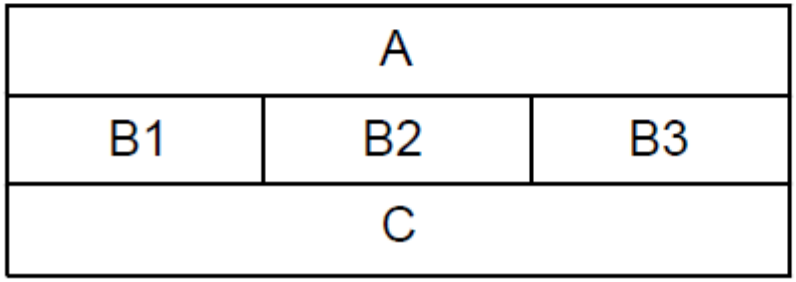
- **Layered style:** Documents the **"allowed to use"** relations among elements of the system

  Relations: "allowed to use," a specialization of the **"depends on"** relation

boxes and arrows



stacks of boxes



What do these diagrams mean?

5.2 NOTATION FOR C&C VIEW TYPE

Elements:

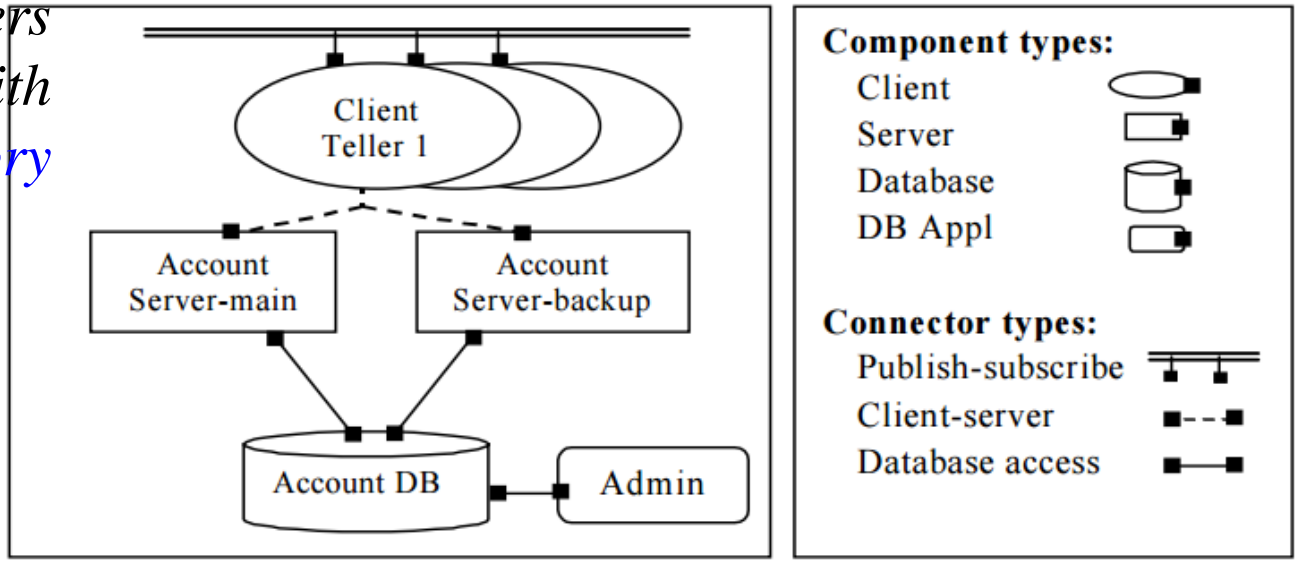
- components: principal units of runtime interaction and data stores
- connectors: interaction mechanisms

Relations:

- attachment of components' *ports to connectors' roles* (interfaces with protocols)

System contains a shared repository accessed by servers and an administrative component.

A set of client tellers can interact with account repository servers



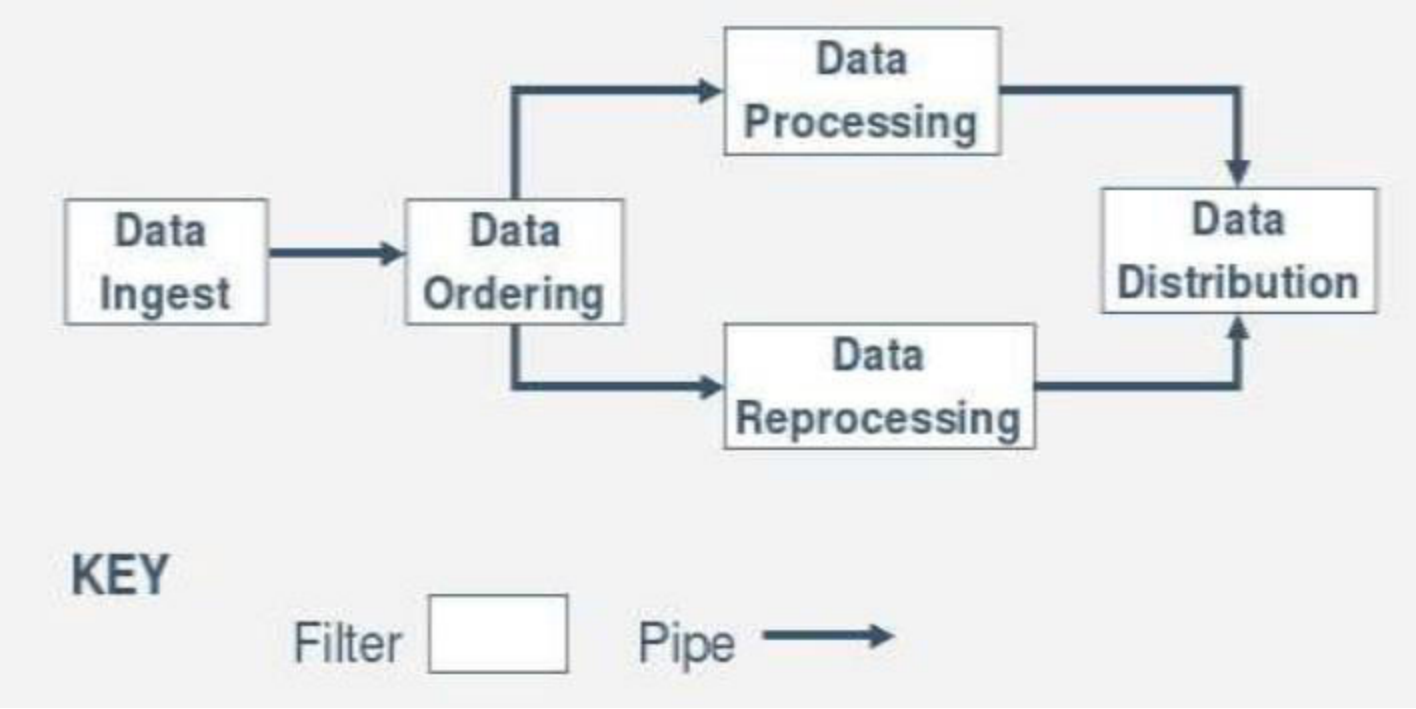
Communication through a publish-subscribe connector.

Pipe-and-Filter style

Elements:

- component type: filter, which transforms data
- connector type: pipe, a unidirectional data means that preserves the order and value of data

Relations: attachment of pipes to filters



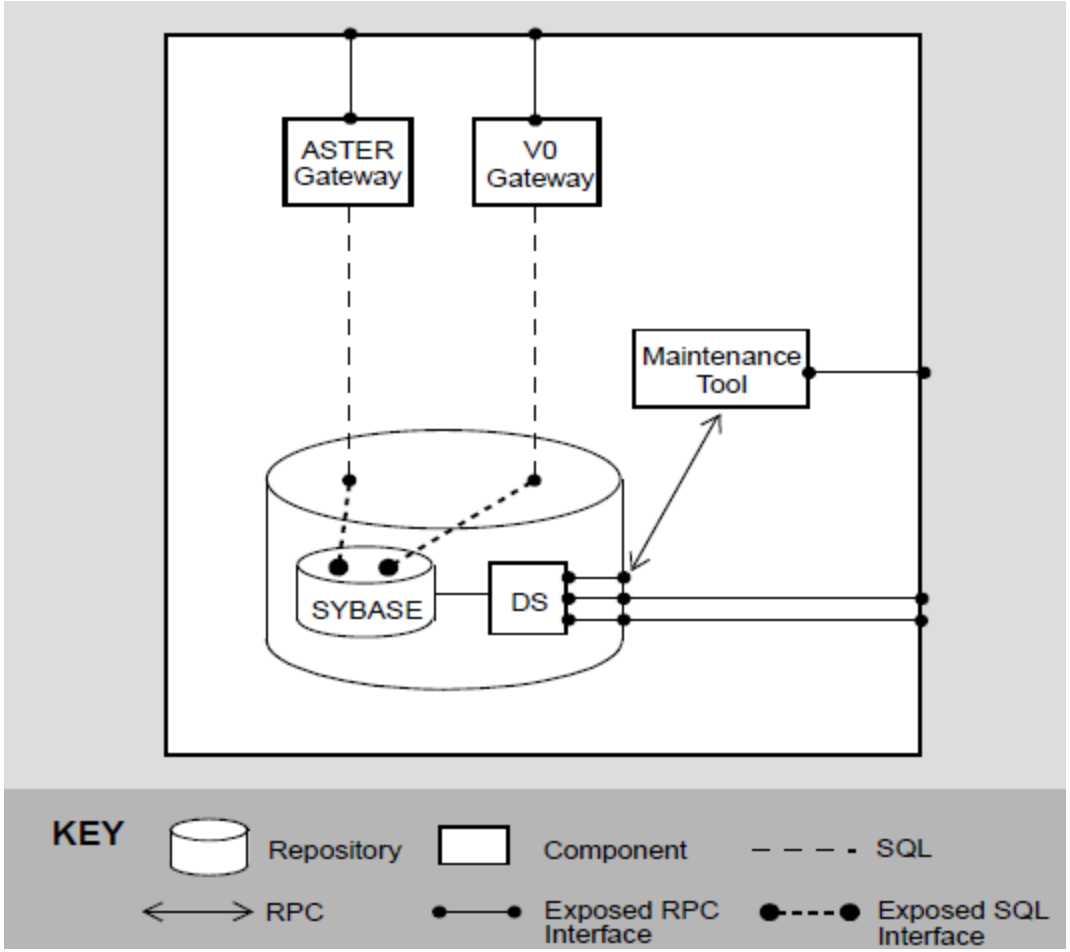
Shared Data Style

Elements:

- component types: data stores and accessors
- connector types: data reading and writing

Relations: attachment

Sybase is a computer software company that develops and sells database management system (DBMS) and middleware products.



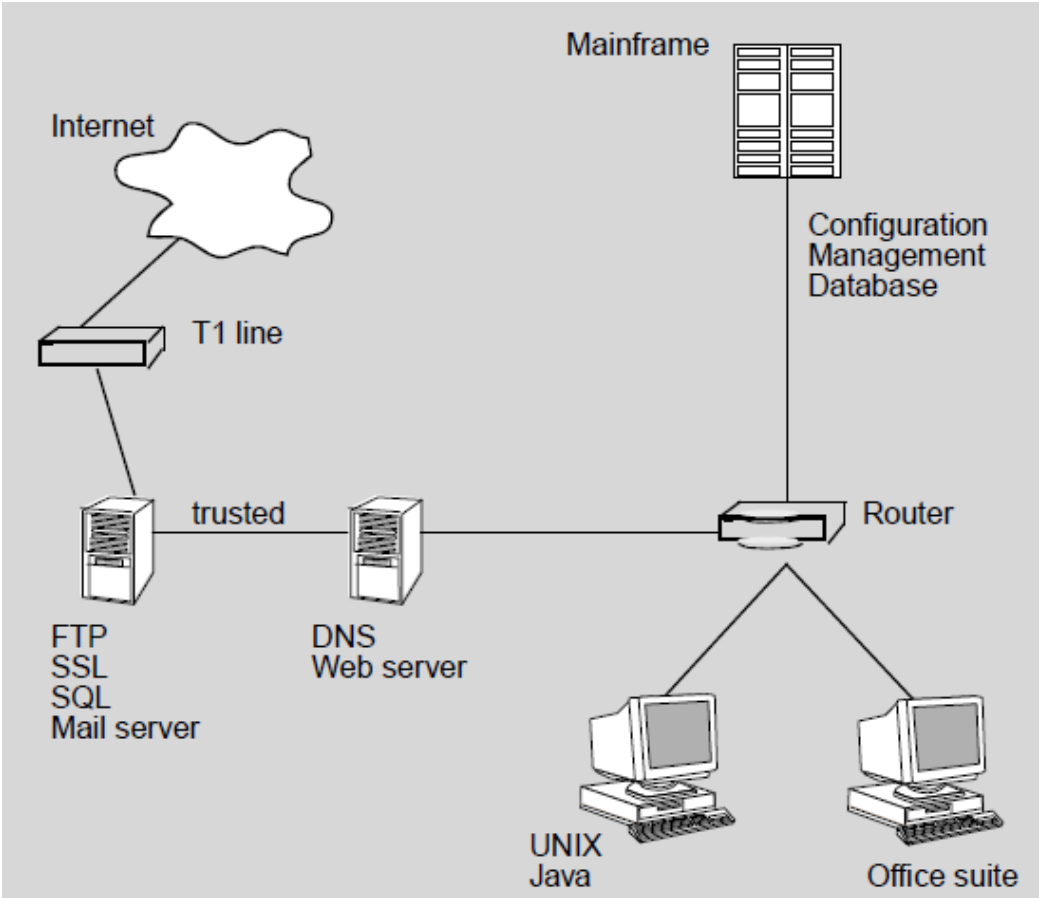
5.3 ALLOCATION VIEW TYPE

Elements:

- Software elements (as defined in module or C&C styles)
- Environment elements

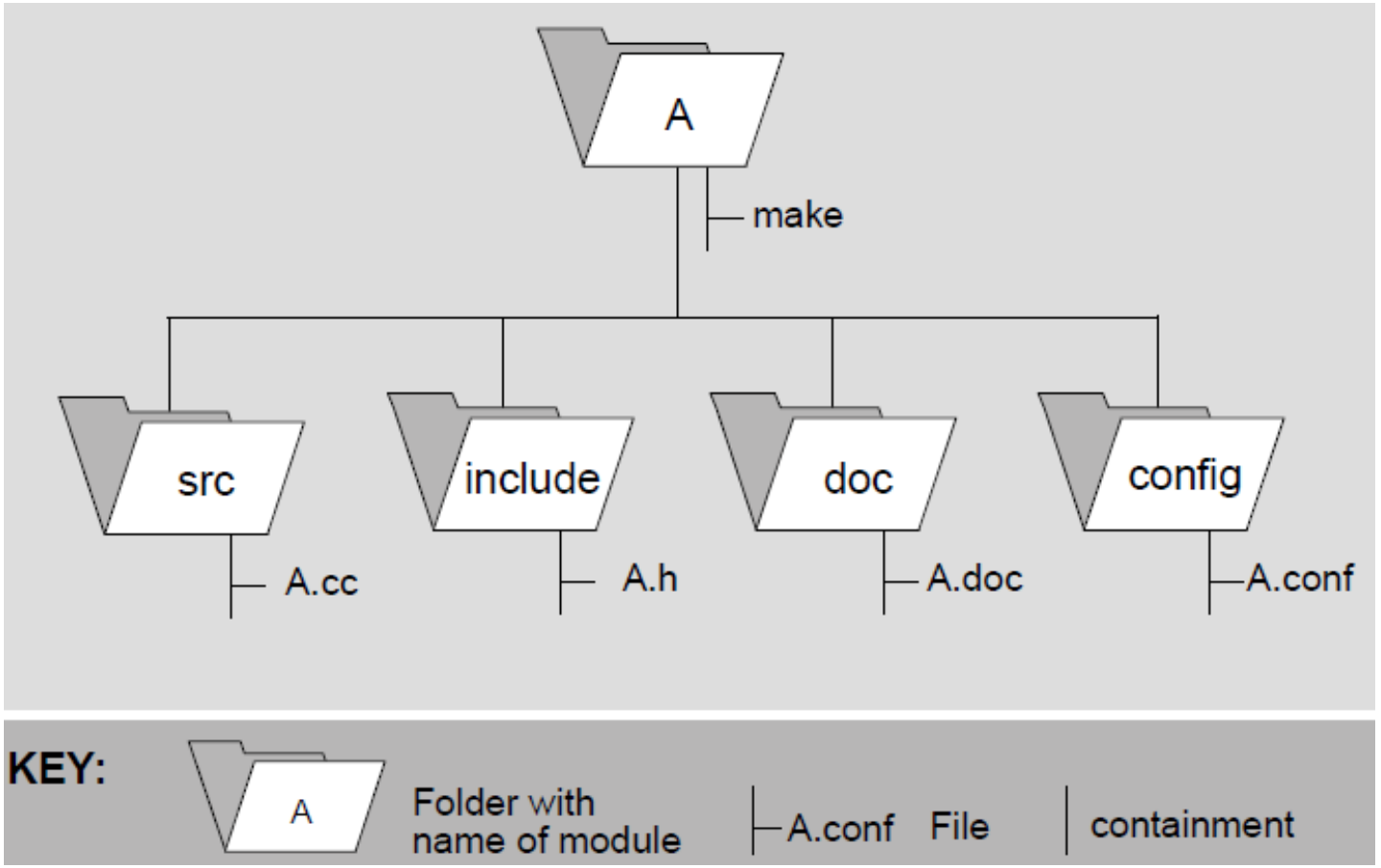
Relations: **“allocated to”**

Deployment style: Allocates software elements to processing and communication nodes



Implementation style:

Allocates software elements to structures in the development environment's file systems

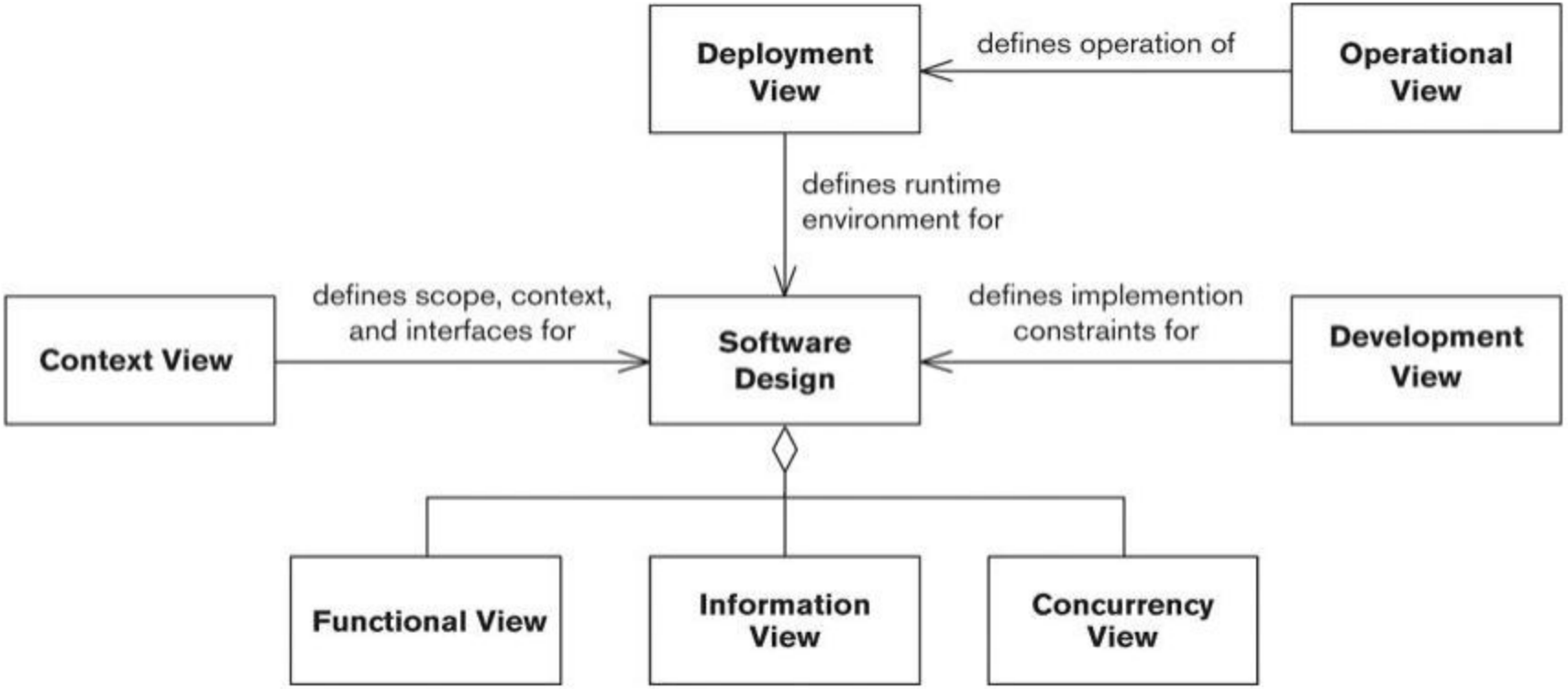


- **Work assignment style:** Allocates software elements to organizational work units

ECS Element (Module)		Organizational unit
Segment	Subsystem	
Science Data Processing Segment (SDPS)	Client	Science team
	Interoperability	Prime contractor team 1
	Ingest	Prime contractor team 2
	Data Management	Data team
	Data Processing	Data team
	Data Server	Data team
	Planning	Orbital vehicle team
Flight Operations Segment (FOS)	Planning and Scheduling	Orbital vehicle team
	Data Management	Database team
	User Interface	User Interface Team
…	…	…

6. STANDARD VIEWS

- Six core viewpoints for Software architecture: the Functional, Information, Concurrency, Development, Deployment, and Operational viewpoints.



Context:

- Describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts).

- Many architecture descriptions focus on views that model the system's internal structures, data elements, interactions, and operation.

- Architects tend to assume that the "outward-facing" information — the system's runtime context, its scope and requirements

- Including a definition of the system's context as part of your architectural description.

Functional:

- Describes the system's functional elements, their responsibilities, interfaces, and primary interactions.

- It is often the first part of the description that stakeholders try to read.

- It drives the shape of other system structures such as the information structure, concurrency structure, deployment structure, and so on.

- It also has a significant impact on the system's quality properties such as its ability to change, its ability to be secured, and its runtime performance.

Information:

- Describes the way that the architecture stores, manipulates, manages, and distributes information.

- The ultimate purpose is to manipulate information in some form, and this viewpoint develops a complete but high-level view of static data structure and information flow.

- The objective of this analysis is to answer the big questions around content, structure, ownership, latency, references, and data migration.

Concurrency:

- Describes the concurrency structure of the system
- Maps functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled.
- This lead to the creation of models that show the process and thread structures that the system will use
- It shows the inter process communication mechanisms used to coordinate their operation.

Development:

- Describes the architecture that supports the software development process.

- Development views communicate the aspects of the architecture of interest to those stakeholders involved in building, testing, maintaining, and enhancing the system.

Deployment:

- Describes the environment into which the system will be deployed

- This view captures the hardware environment that your system needs the technical environment requirements for each element, and the mapping of the software elements to the runtime environment that will execute them.

Operational:

- Describes how the system will be operated, administered, and supported when it is running in its production environment.
- Installing, managing, and operating the system is a significant task that must be considered and planned at design time.
- The aim of the Operational viewpoint is to identify system-wide approaches for addressing the operational concerns of the system's stakeholders and to identify solutions that address these.

7] 4+1 VIEW OF RUP

- The Rational Unified Process is a software engineering process, which is an extension to Unified Modeling Language (UML)

- A guide to the effective use of the UML for modeling.

- It organizes software projects in terms of workflows and phases, each consisting of one or more iterations, in the way that testing and validating design ideas

- Decreasing risks is possible in the early steps of a lifecycle

- Philippe Kruchten
  - Over 16 years of experience as the leader of RUP development team in Rational corp. (now owned by IBM)
  - Valuable experiences in industry (Telecom, Air traffic control system) which he used them for confirmation of his model
- The "4+1 view model" paper:
  - 60 citations according to ACM portal site

Problem

- Arch. documents over-emphasize an aspect of development (i.e. team organization) or do not address the concerns of all stakeholders

- Various stakeholders of software system: end-user, developers, system engineers, project managers

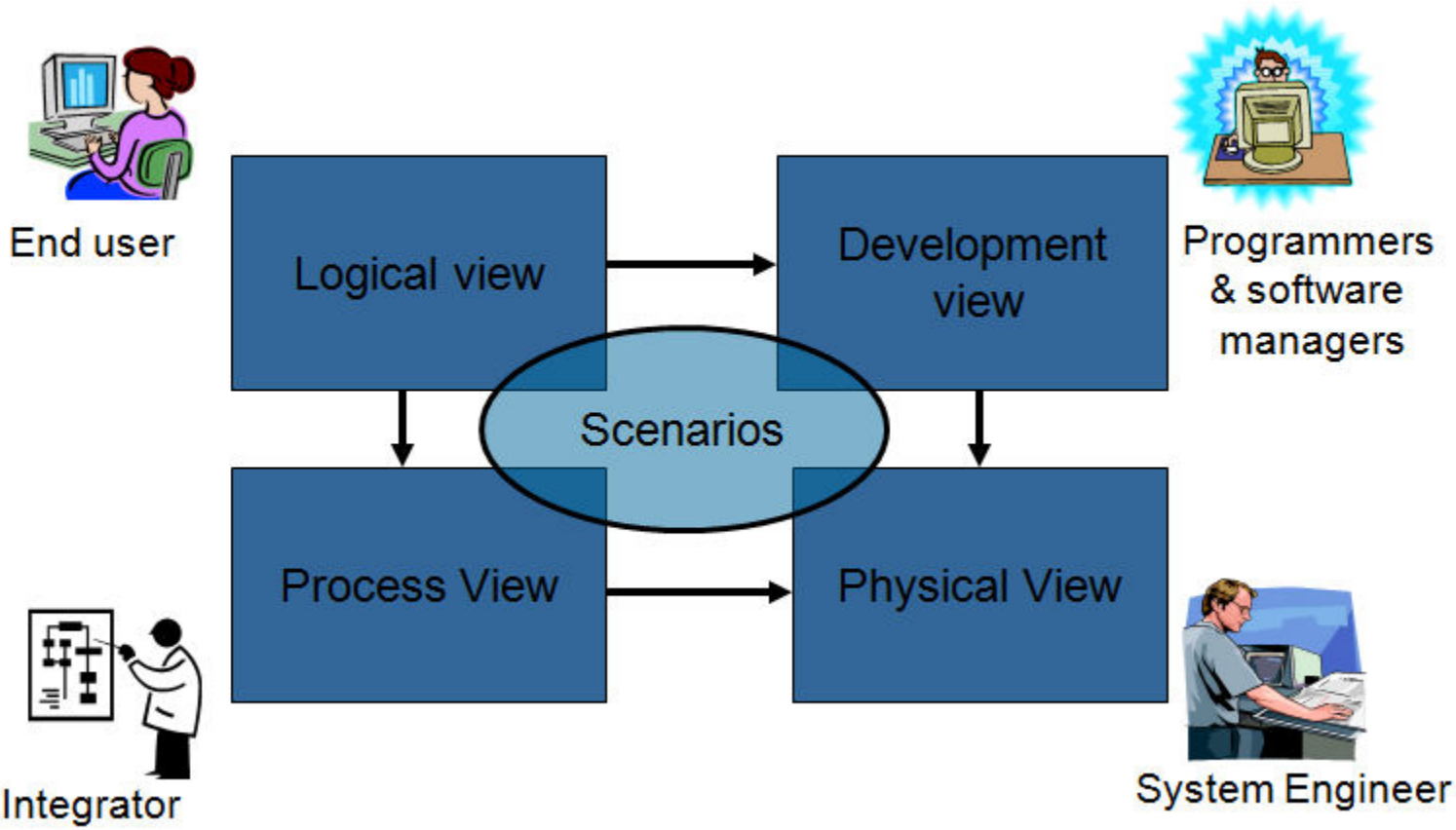
- Software engineers struggled to represent more on one blueprint, and so arch. documents contain complex diagrams

Solution

- Using several concurrent *views* or *perspectives*, with different notations each one addressing one specific set for concerns
- "4+1" view model presented to address large and challenging architectures
- It's a way to show key viewpoints of an architecture.
- The four views are: logical, process, development, and physical.
- The +1 is the scenarios. The scenarios are at the center of the model.

- The logical view is a view of the important classes and relationships.
- The process view is the runtime and execution view.
- The development view shows the layers and how classes are organized, as well as dependencies.
- The physical view shows the deployment scenario and associated hardware.

4+1 View Model of Architecture



Logical view

(Object-oriented Decomposition)

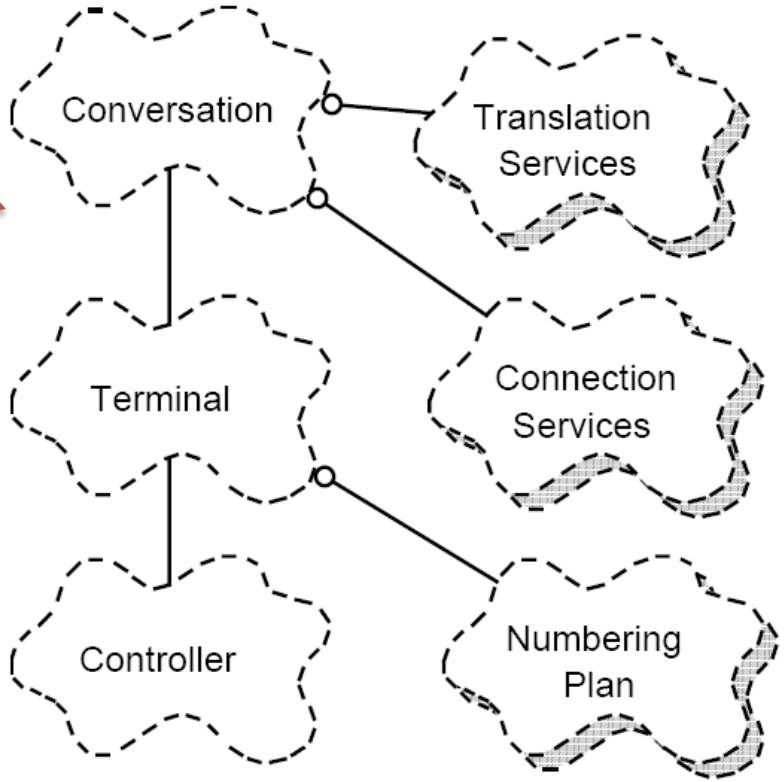
Viewer: End-user

considers: Functional requirements- What the system should provide in terms of services to its users.

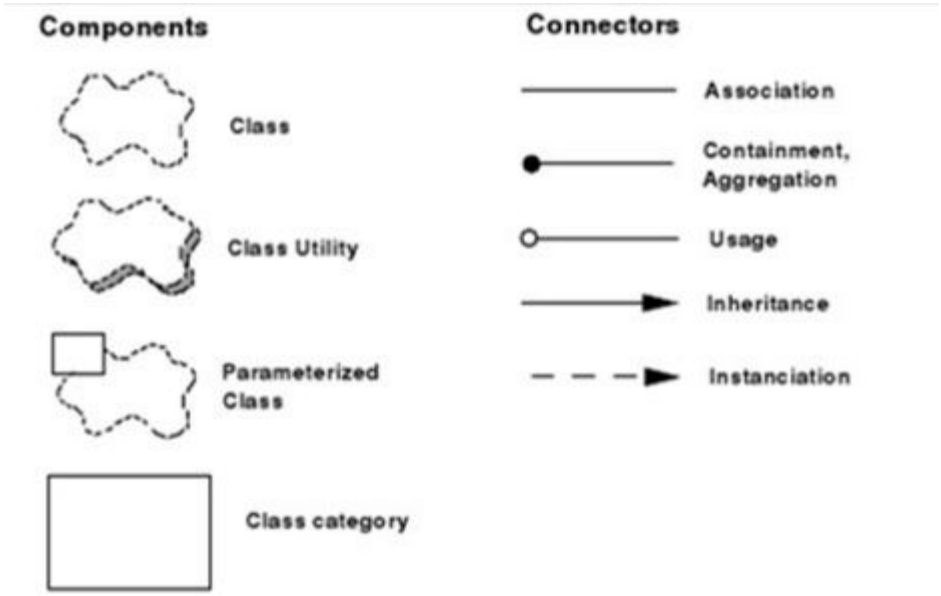
Notation: The Booch notation (OMT) (object and dynamic models)

Tool: Rational Rose

Example



Notation for Logical View



Process View

(The process decomposition)

viewer: Integrators

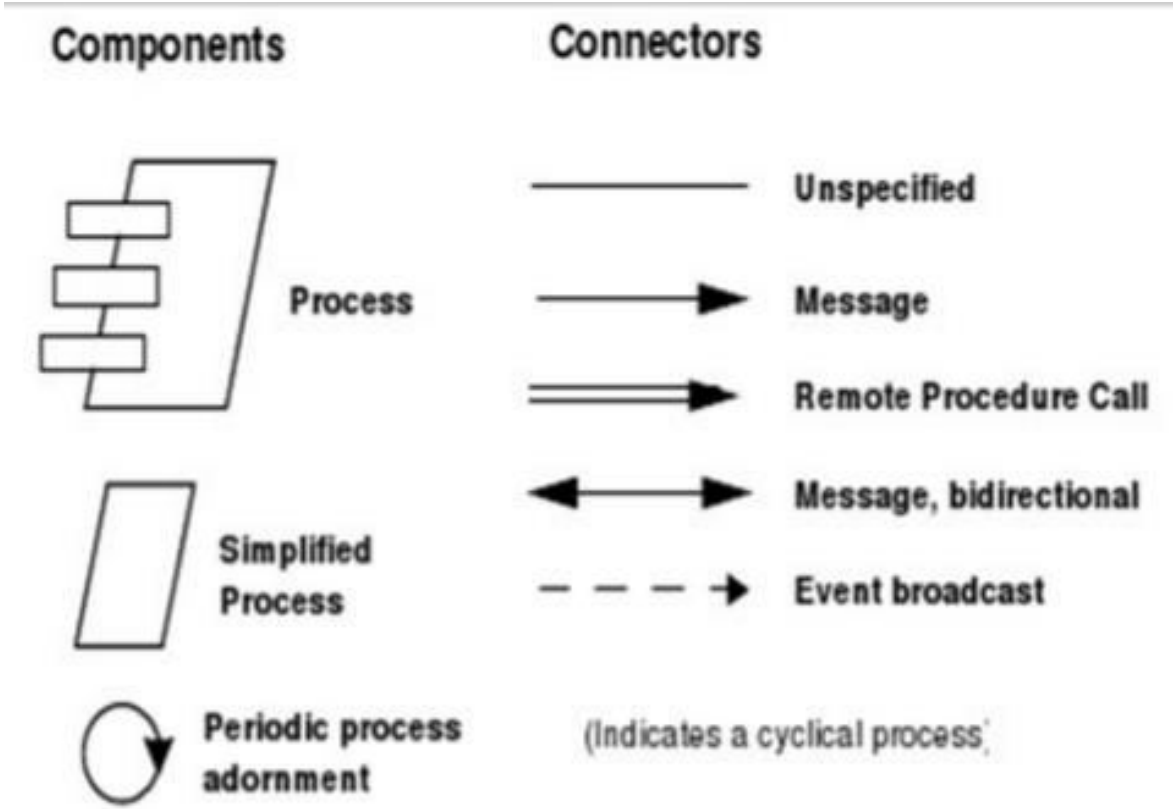
considers: Non - functional requirements (concurrency, performance, scalability)

style: Several styles would fit in this view (Garlan and Shaw ‘s Architecture styles)

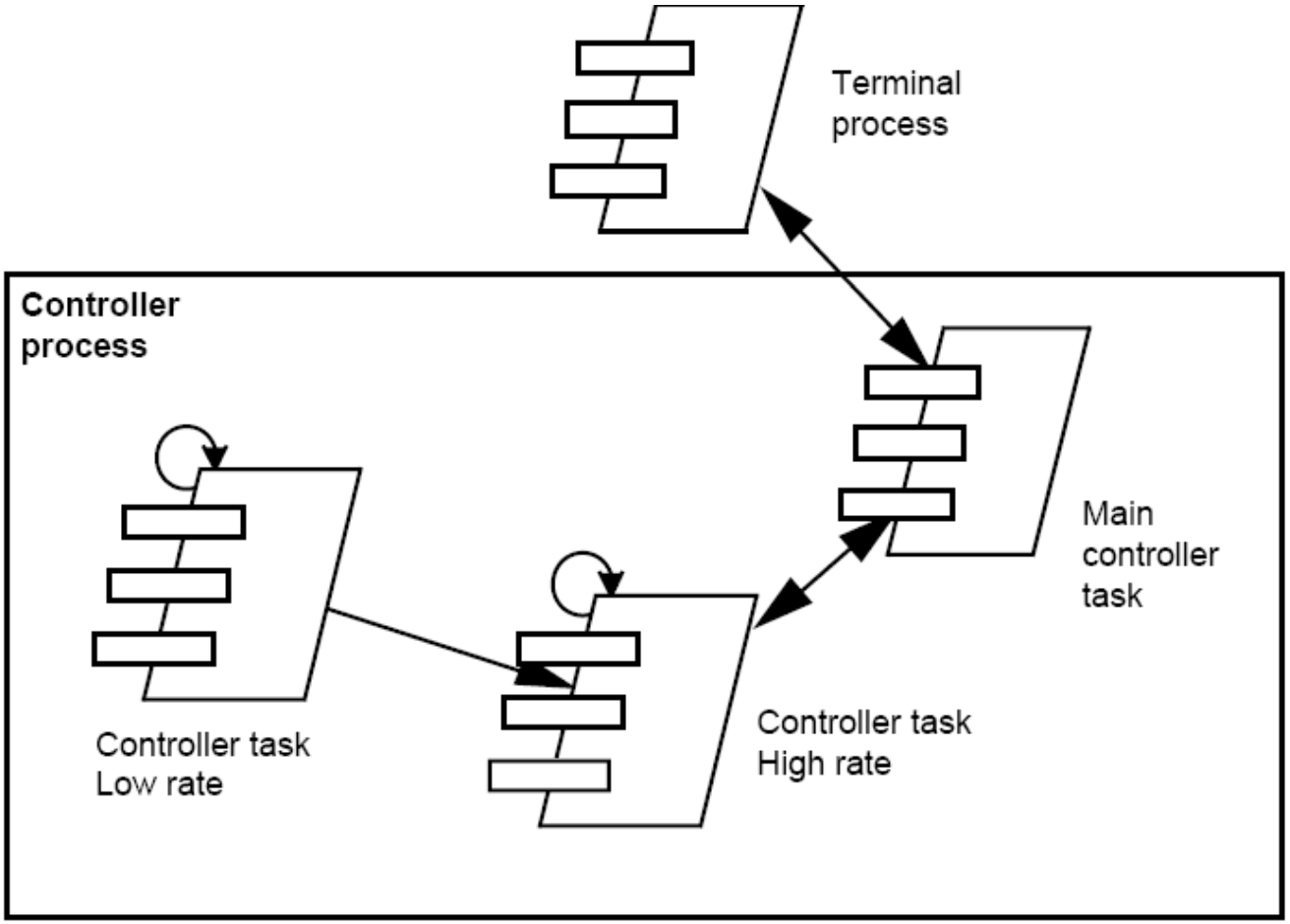
- Uses multiple levels of abstractions, a logical network of processes at the highest level

- A process is a grouping of tasks that form an executable unit:
  - Major Tasks: Arch. relevant tasks
  - Minor Tasks: Helper tasks. (Buffering)

Notation for Process View



Example for Process view



Development View

(Subsystem decomposition)

Viewer: Programmers and Software Managers

considers: software module organization (Hierarchy of layers, software management, reuse, constraints of tools)

Style: layered style

Notation: the Booch notation (module, subsystem, layer)

Physical View

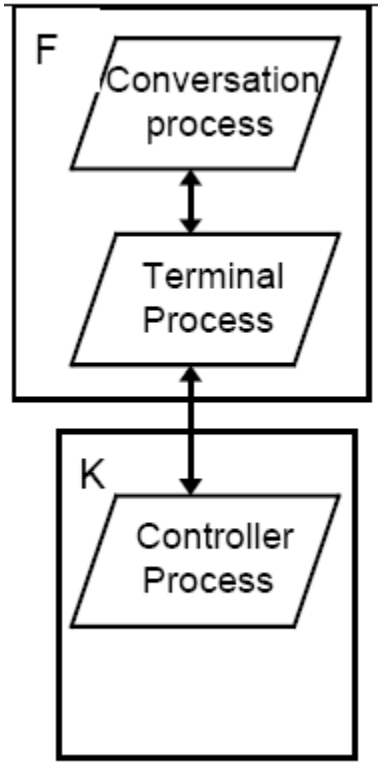
(Mapping the software to the Hardware)

Viewer: System Engineers

Considers: Non-functional req. regarding to underlying hardware (Topology, Communication)

Notation: May have several forms and may Tightly connected to the process view

- There may be two architecture:
  - Test and development
  - deployment



SCENARIOS

(Putting it all together)

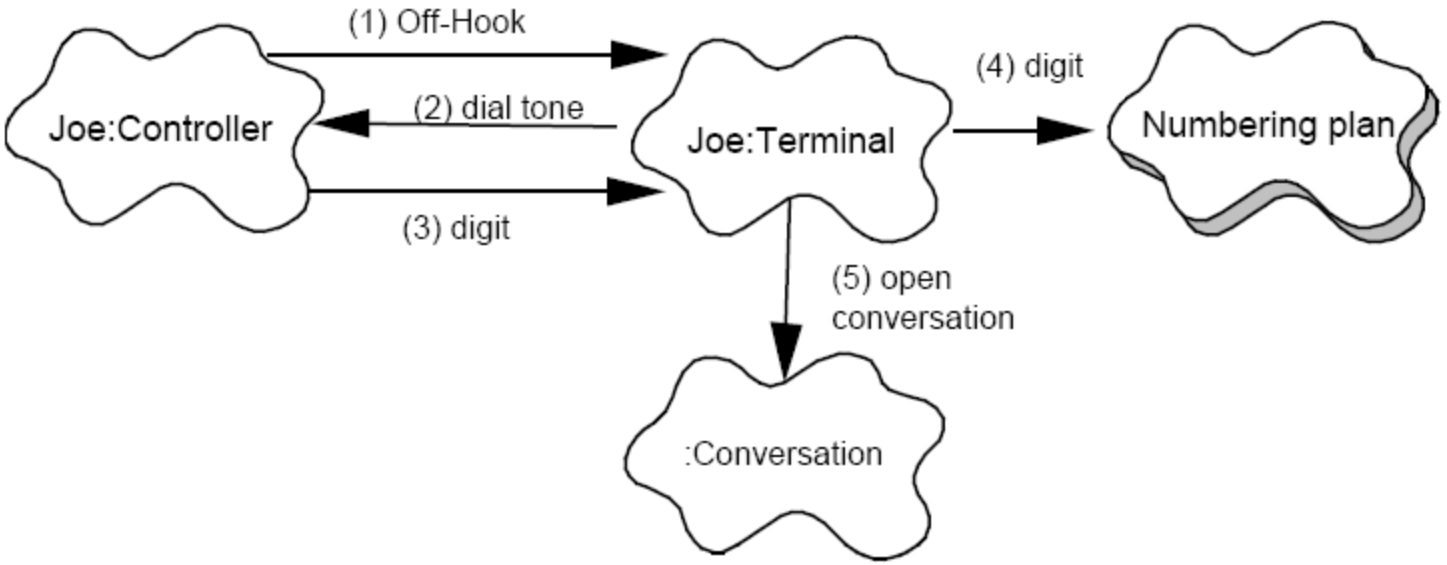
Viewer: All users of other views and Evaluators.

Considers: System consistency, validity

Notation: almost similar to logical view

Tool: Rational Rose

- Help illustrate and validate the document
- Help Architect during the architecture design



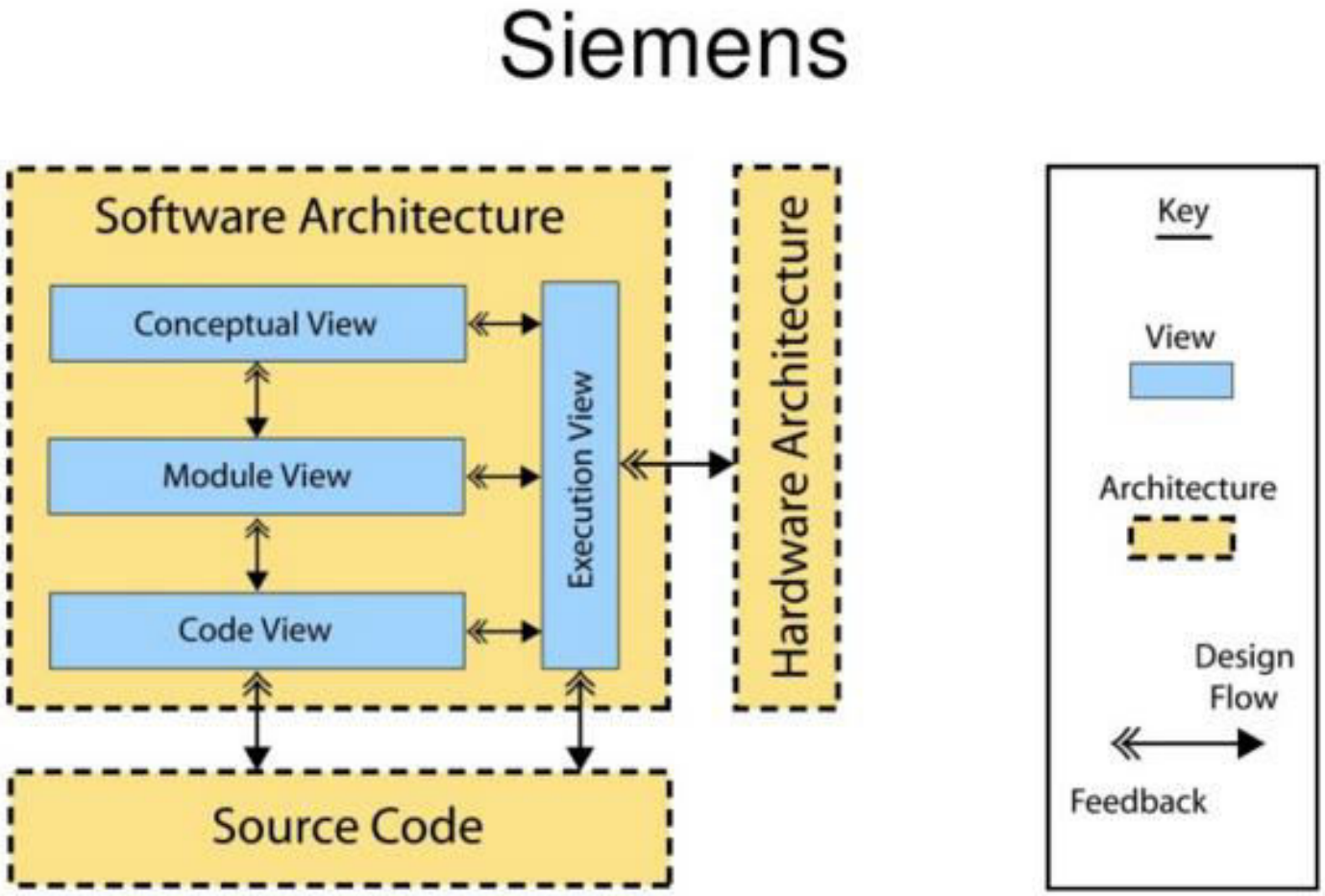
View	*Logical*	*Process*	*Development*	*Physical*	*Scenarios*
Components	Class	Task	Module, Subsystem	Node	Step, Scripts
Connectors	association, inheritance, containment	Rendez-vous, Message, broadcast, RPC, etc.	compilation dependency, "with" clause, "include"	Communica-tion medium, LAN, WAN, bus, etc.	
Containers	Class category	Process	Subsystem (library)	Physical subsystem	Web
Stakeholders	End-user	System designer, integrator	Developer, manager	System designer	End-user, developer
Concerns	Functionality	Performance, availability, S/W fault-tolerance, integrity	Organization, reuse, portability, line-of-product	Scalability, performance,av ailability	Understand-ability
Tool support	Rose	UNAS/SALE DADS	Apex, SoDA	UNAS, Openview DADS	Rose

Table 1 — Summary of the "4+1" view model

8.SIEMENS 4 VIEWS

- Developed at Siemens Corporate Research
- This is based on best architecture practices for industrial systems
- The four views are conceptual, execution, module and code architecture views.
- These four views separate different engineering concerns, thus reducing the complexity of the architecture design task.

Siemens Four View Model



Describe each view

Conceptual view: Major design elements of the problem domain.

Module view: Functional decomposition of systems and sub-systems.

Code view: Dynamic structures of the system.

Execution view: Language level modules.

1. Conceptual view:
   - The primary engineering concerns in this view are to address how the system fulfills the requirements. The functional requirements are a central concern.
   - The conceptual architecture view explains how the system's functionality is mapped to components and connectors.

2. Module view:
   - Modules are organized into two orthogonal structures:
   - decomposition and layers.
   - The decomposition structure captures how the system is logically decomposed into subsystems and modules.
   - A module can be assigned to a layer, which then constrains its dependencies on other modules.
   - This view are to minimize dependencies between modules, maximize reuse of modules, and support testing.
   - Documenting the module architecture view can be done by using the module viewtype.

3. Execution architecture view:

- The execution architecture view explains how the system's functionality is mapped to runtime platform elements, such as processes and shared libraries.

- This view describes the system's structure in terms of its runtime platform elements.

- Runtime properties of the system, such as performance, safety, and replication are also addressed here.

4. Code architecture view:

- This is concerned with the organization of the software artifact

- The engineering concerns of this view are to make support product versions and releases, minimize effort for product upgrades, minimize build time, and support integration and testing.

- To use views prescribed by the Siemens Four Views approach, do the following list:

To Achieve This Siemens Four Views View	Use This Approach
Conceptual architecture	One or more styles in the C&C viewtype
Module architecture	One or more styles in the module viewtype
Execution architecture	Deployment style in the allocation viewtype; for processes, communicating-processes style in the C&C viewtype
Code architecture	Implementation style in the allocation viewtype

In Siemens View Global Analysis involves

- Identifying influencing(weighted) factors
- Analyzing them to identify their importance to the architecture, flexibility, and changeability
- Identifying key issues or problems that arise from a set of factors

S4V splits evaluation into two types

  - global evaluation, done by the architect as the design progresses
  - architecture evaluation, led by a team of external reviewers, and done at major checkpoints

9. SEI'S PERSPECTIVES AND VIEWS

- Described by Clements et al., of the Carnegie Mellon

- University's Software Engineering Institute, in 2002.

Describe each view;

- Module view type:

  Static structure of the system.

- C & view type:

  Dynamic structure.

- Allocation view type:

  Relationship to external environments.

Viewpoint Groups

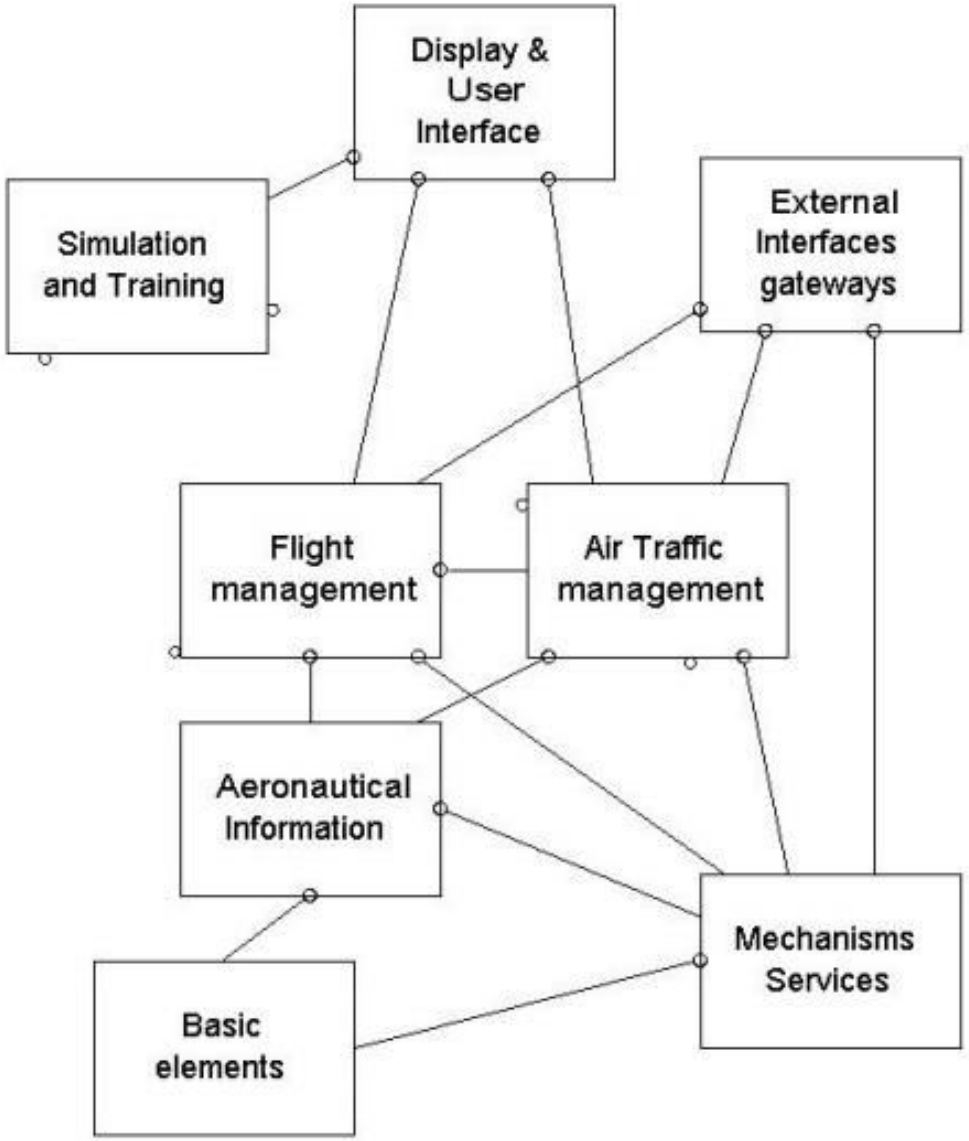
	"4+1" model	SEI model	Siemens model
Functional	Logical view	Module viewtype	Module view
Behavioural	Process view	C&C viewtype	Execution view
External	Development view	Allocation viewtype	Code view

CASE STUDIES

Key Word In Context (KWIC)

- Search index
  - searching for keywords with context sensitive display
  - provides the user with more information

Blueprint for an Air Traffic Control System



LOGICAL VIEW