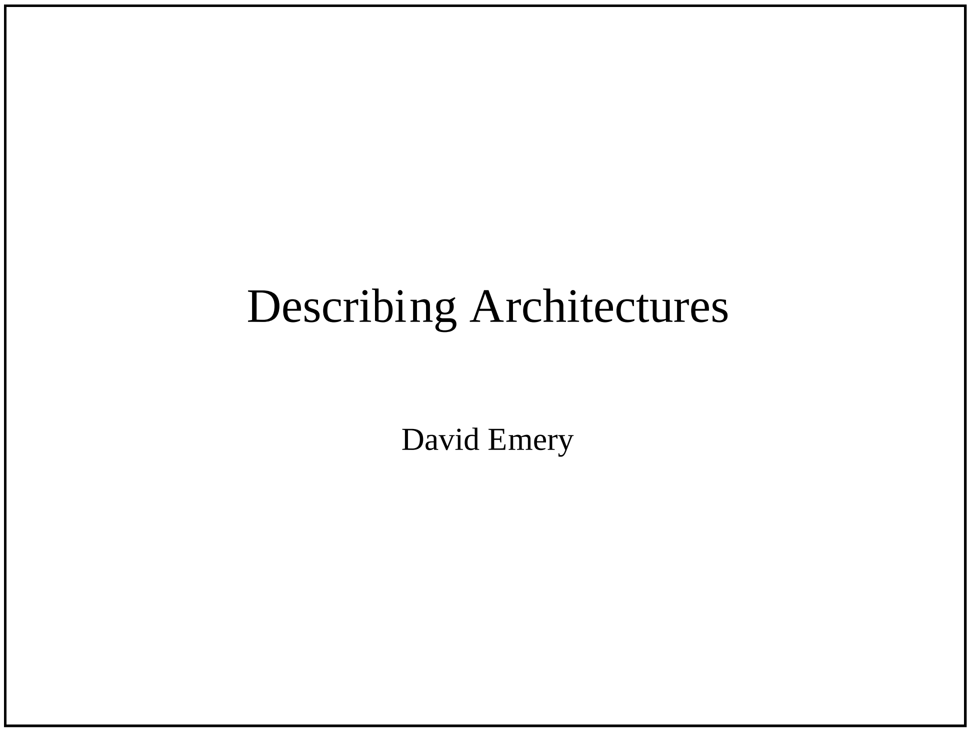


Describing Architectures

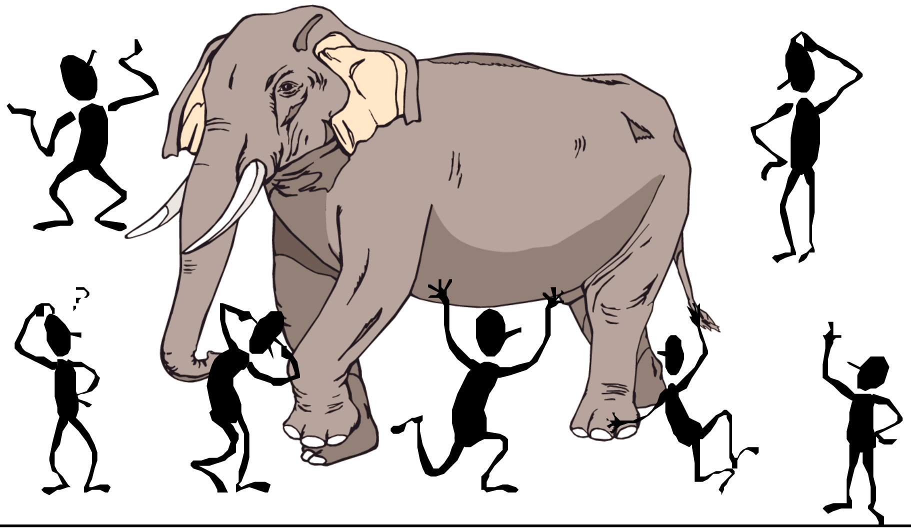
David Emery

This talk is presented in 'annotated briefing' format, since it depends heavily on the specific graphics being presented.

The term "architecture" has come into wide usage across our industry. For this talk, we are concentrating on the notion of "architecture" applied to a system as a whole. This can include the architecture of a software-only system, or of a software-intensive system.

Within the IEEE 1471 effort (more on that later), defining the term 'architecture' was the single most contentious issue. There was much more consensus that we were really less interested in "architecture" than in -describing- architectures, and working with descriptions of architectures, to document, explain, compare and even verify architectures. So this talk will concentrate on my experiences with various forms of architecture descriptions. After providing examples, I will introduce IEEE Std 1471:1990 Recommended Practice for Architectural Descriptions of Software-Intensive Systems, and use IEEE 1471 to unify thoughts on the problem of 'architectural descriptions'.

The 7 Blind Men and the Elephant

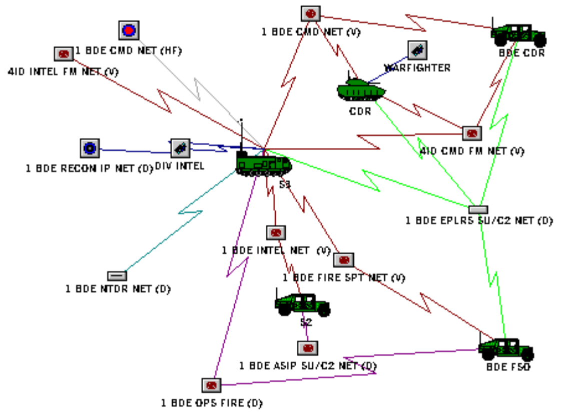


A wise king sent his 7 wisest men, who happened to be blind, to determine the architecture of an elephant and report back. One wise man felt the elephant's trunk, and said "An elephant is like a snake." Another felt the elephant's leg and said "An elephant is like a tree," etc. <footnote>.

This is the current state of describing architectures. We have lots of experiences describing architectures from various specific viewpoints.

What follows are some example representations of architectures.

An Architecture is a Picture...

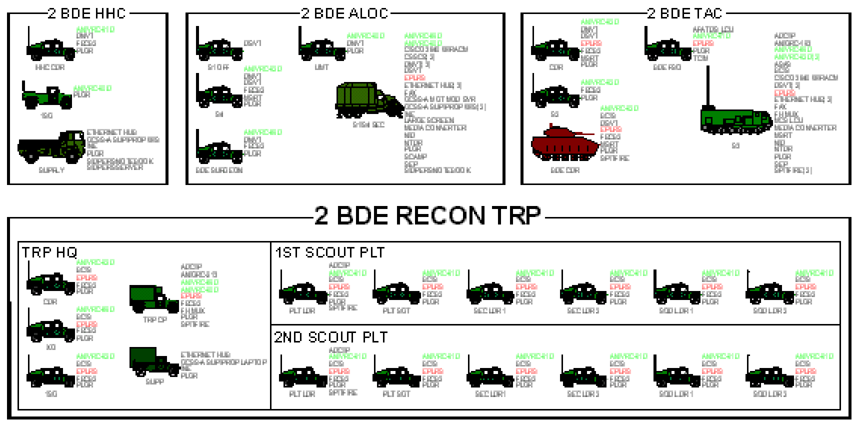


Army System Architecture: Brigade Tactical Operations Center

Here's a description of an architecture. This comes from the Army's First Digital Force System Architecture (1DFSA) project. It shows the vehicles that make up a Brigade Tactical Operations Center, the radios on those vehicles, and the nets that those radios participate in.

Pictures and diagrams are very common representations of architectures. Many notions of architecture concentrate on 'connectivity,' of software modules, of communications nodes, etc. One of the problems with such pictures is that it captures a single instance of connectivity, where the real intent is that there are a large number of potentially valid configurations of nodes and connections.

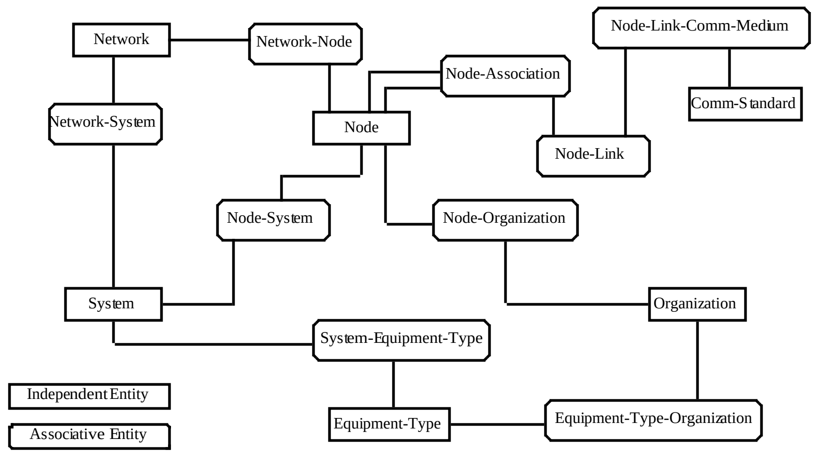
Another Architectural Picture...



Army System Architecture: Brigade Headquarters Company

Here's another representation of the same architecture. Where the previous representation concentrated on representing communications connectivity, this view shows the equipment issued to the organization, including a listing of major Command Control and Communications equipment in each vehicle. The primary audience for this picture are the people who are concerned about counts, such as the unit that has to store the vehicles, and the acquisition agent who has to purchase enough radios to fit into each required vehicle.

An Architecture is a Database...

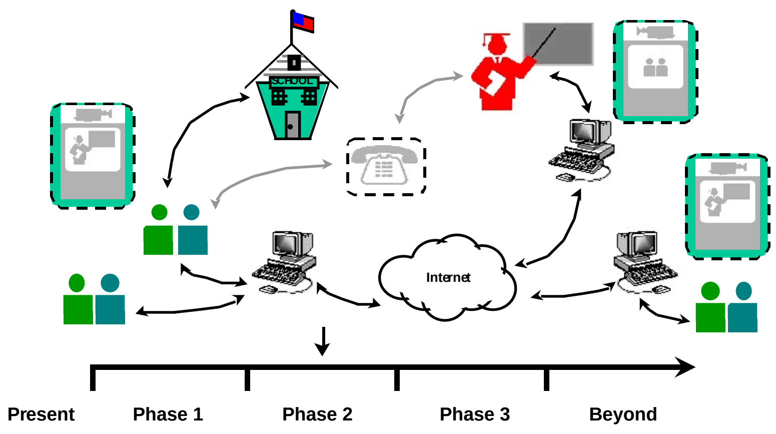


C4ISR Architecture Data Model extract

Many claim that an architecture is a database. The real intent of an architecture, they assert, is to completely capture the data that describes the system. This is an extract of the C4ISR Architecture Framework Data Model (CADM). It captures the data represented in a node-connectivity diagram. If you add information about the icons and their location on the screen, you can use this data to reproduce the radio nets picture shown previously. In fact, the Army has integrated its graphical views and the underlying database, so that they can create the pictures from the database, -and- use the graphical tool (NetVIZ) to load information into the architecture database.

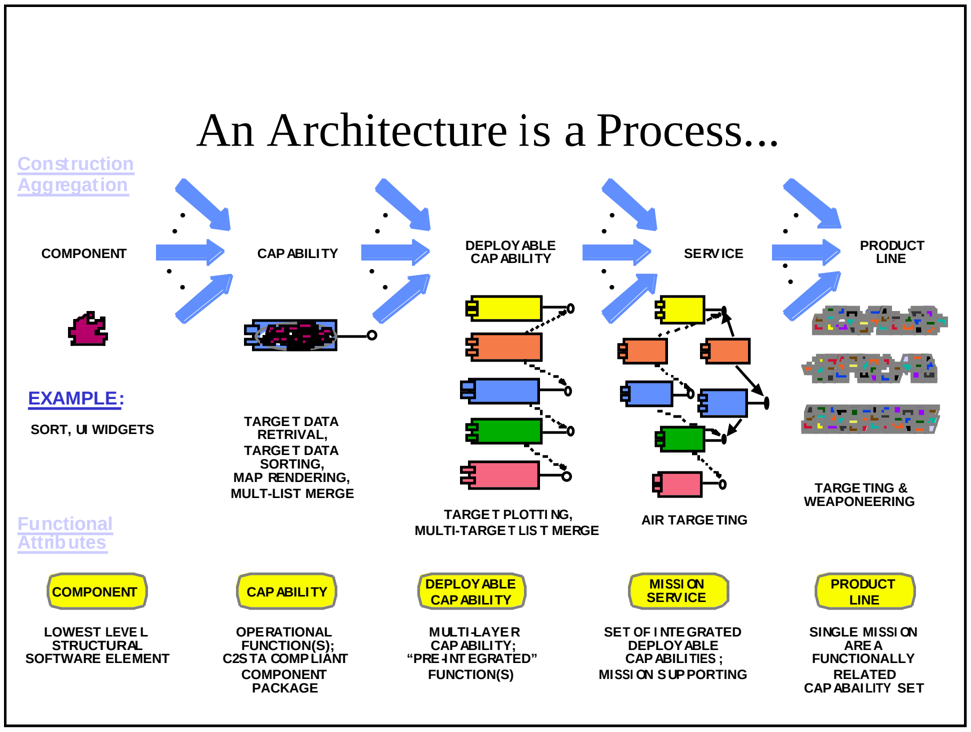
The representation used for the CADM is IDEF-1X. It's also possible to use UML to capture this kind of database schema information.

An Architecture is a Use Case...



Distance Learning via Computer-Based Instruction and Collaborative/Groupware

Another approach to architectures says that the architecture can be expressed as a series of Use Cases. Philippe Kruchten's "4+1" model for architectures places particular emphasis on Use Cases as part of the architecture description. (It's the "+1" part of "4+1".) This example is one of a series of use cases that we used to capture the Total Army Distance Learning Program's architecture. One of the key concerns was to understand how instruction was delivered to the student through a variety of sources/media. In this example, we have students in a local classroom, communicating via the computer to students in other classrooms and to the instructor. The "computer" here includes interactive video. Students also have the option of phoning questions directly to the instructor. Thus this Use Case shows some of the issues with 'distributed learning', including synchronizing video displays, mechanisms for remote students to ask questions of the instructor, and also the various technologies (computers, internet, phone lines, video cameras, "NetMeeting", etc) used to deliver this example of distributed distance learning.



An Architecture is a Process...

COMPONENT	CAPABILITY	DEPLOYABLE CAPABILITY	MISSION SERVICE	PRODUCT LINE
LOWEST LEVEL STRUCTURAL SOFTWARE ELEMENT	OPERATIONAL FUNCTION(S); C2STA COMPLIANT COMPONENT PACKAGE	MULTI-LAYER CAPABILITY; “PRE-INTEGRATED” FUNCTION(S)	SET OF INTEGRATED DEPLOYABLE CAPABILITIES; MISSION SUPPORTING	SINGLE MISSION AREA FUNCTIONALLY RELATED CAPABAILITY SET

Some of the previous examples have demonstrated an architecture by showing one instance of a set of configurations. Implicit in those representations are the process by which a valid instance of the system can be constructed.

This representation says that the real architecture is the process by which valid instances are constructed. Rather than showing the architecture as one or more examples and making the user "derive" the rules, this approach captures the rules or process explicity.

This example comes from the US Air Force C2 Architecture. It shows the process by which components are assembled into systems. The USAF C2 Architecture is used to provide an umbrella over a large number of interoperable C2 programs within the Air Force. The ultimate vision is that you can configure a conforming system "on the fly" by integrating the necessary components.

An Architecture is a Set of Rules...

No actor at a given classification level shall be able to read data classified at a higher level.

No actor at a given classification level shall be able to write data classified at a lower level.

Each session shall allow a maximum of 3 login attempts. After the third failed attempt, the user account shall be frozen.

A related approach to describing architectures is to capture the rules and constraints defined by the architecture. The difference between this approach and the "process" approach is that this approach does not specify any particular process, but only specifies the constraints. The "process" approach tells you both what to do and also how to do it.

The first two rules are an expression of the Bell-LaPadula multi-level security rules. The third rule is a common rule for managing interactive logins. A consequence of this rule is that there must be some facility to handle the situation where a terminal gets frozen after 3 login attempts. The mechanisms for doing this are not captured in the rules for the system.

An Architecture is a Set of Patterns...

Clients-Servers

Object Factories

Data Push/Data Pull

Pipes and Filters

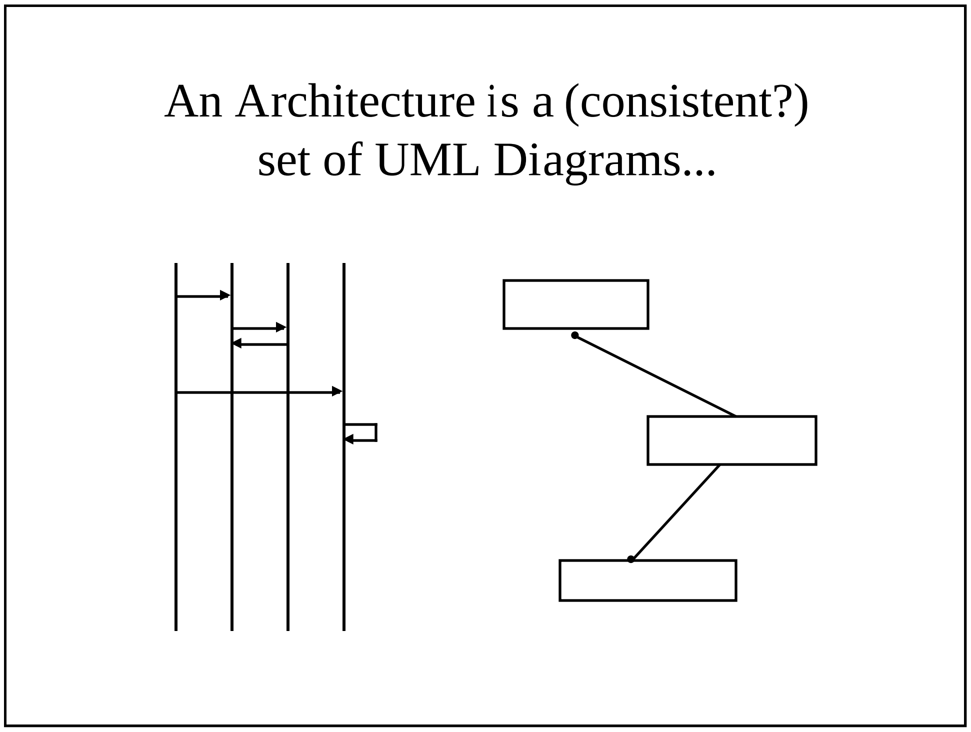
Blackboards/Collaboration

Seven-Layer Protocol Stacks

Model-View-Controller

An approach that has been defined in the literature is to capture architecture as a set of patterns or styles. Thus you can construct a "client-server" style system, and much of the architecture is pre-defined by what it means to be "client-server". Most of the patterns on this list come from the book "Object-Oriented Software Architectures: A System of Patterns." The work done at CMU and SEI on "architectural styles" also follows this approach.

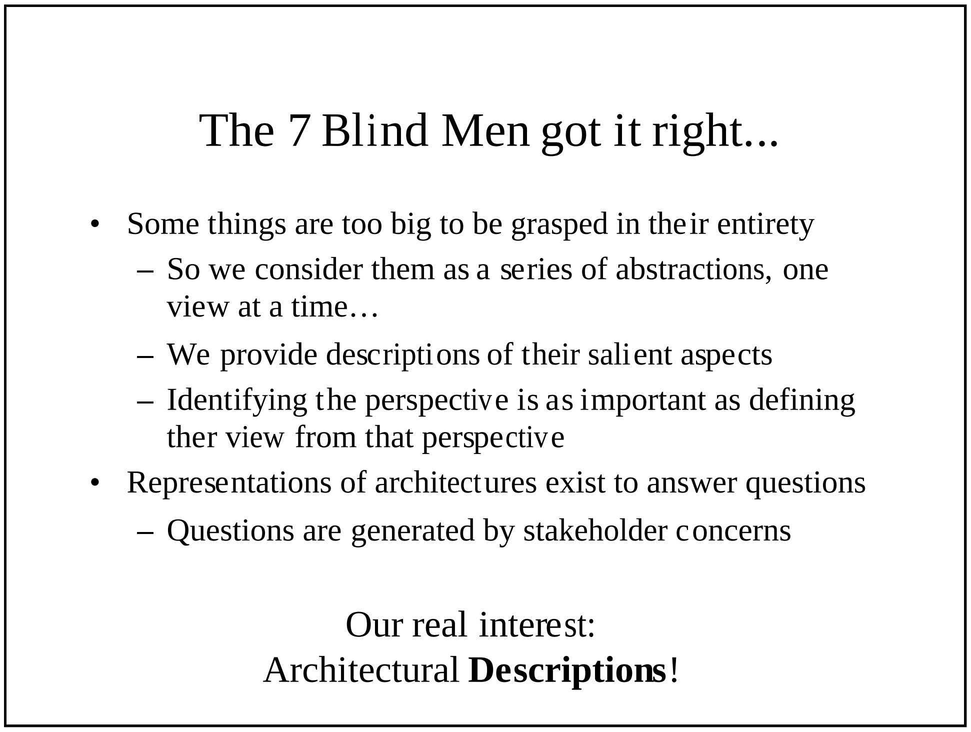
The key point of this approach is that it focuses on the structure of a system. "Client-Server" describes how the system can be built, but doesn't tell you very much about how the system actually performs its role in the organization.



An Architecture is a (consistent?) set of UML Diagrams...

Many have asserted that UML contains everything you need to describe architectures. Thus an architecture is a UML notations, and presumably one of the advantages of using UML is that UML helps keep these representations consistent.

It is not clear, though, that the existing UML models capture the right kinds of architectural information. An open issue is how UML can be extended/modified to support architectural concerns (particularly concerns not associated with the structure of the system, such as performance or security or maintainability.)



The 7 Blind Men got it right...

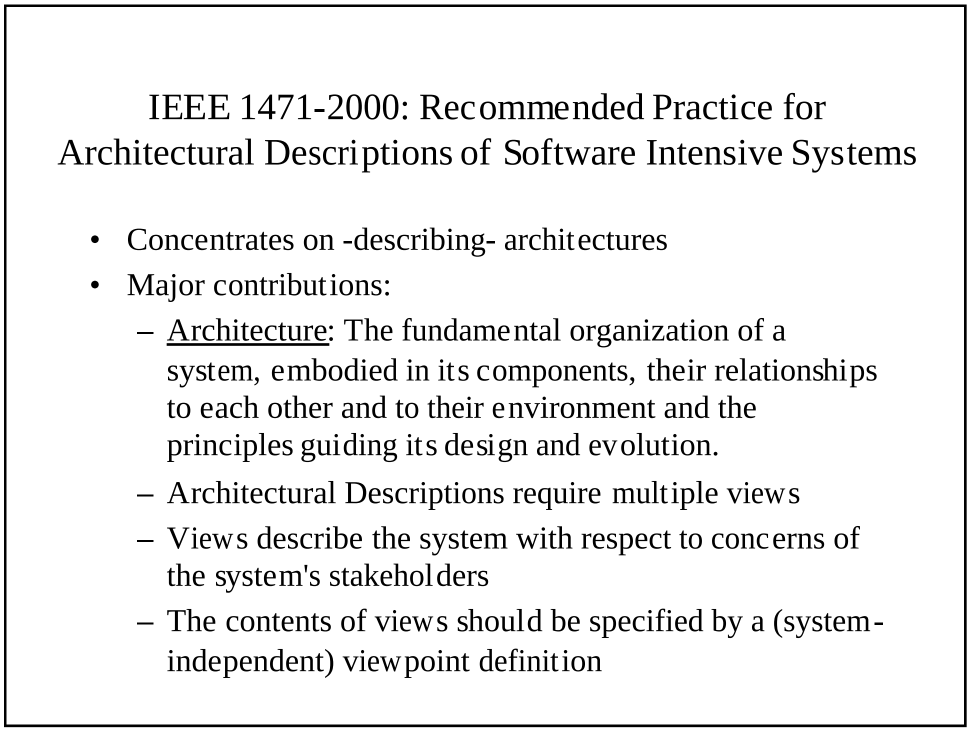
- Some things are too big to be grasped in their entirety
  - So we consider them as a series of abstractions, one view at a time…
  - We provide descriptions of their salient aspects
  - Identifying the perspective is as important as defining ther view from that perspective
- Representations of architectures exist to answer questions
  - Questions are generated by stakeholder concerns

Our real interest:
Architectural **Descriptions**!

So what can we learn from all of these representations of "architecture"? I think that the 7 blind men had the right idea about describing architectures. Each of the blind men approached the problem from a given perspective, and each captured knowledge gained from looking at the problem from that perspective.

This slide shows what I've learned about "architectures" over the last 10 years. Fundamentally, the key problem is not "what is an architecture?" but rather "how do we describe architectures?" Without descriptions, architectures are just an amorphous concept, and we have no way to analyze or compare architectures unless we can describe them.

What we are trying to capture with architectural descriptions are answers to concerns about the system. These concerns derive from the stakeholders of the system. There are lots of "stakeholders" for systems, including users, developers, acquisition agents, oversight agents, maintainers, testers, etc. Each has concerns about the system. Each needs to understand the whole system. Thus architectures should be described via a set of views, where each view covers the entire system and address some set of stakeholder concerns. The Architecture is represented by the sum of these views.



IEEE 1471-2000: Recommended Practice for Architectural Descriptions of Software Intensive Systems

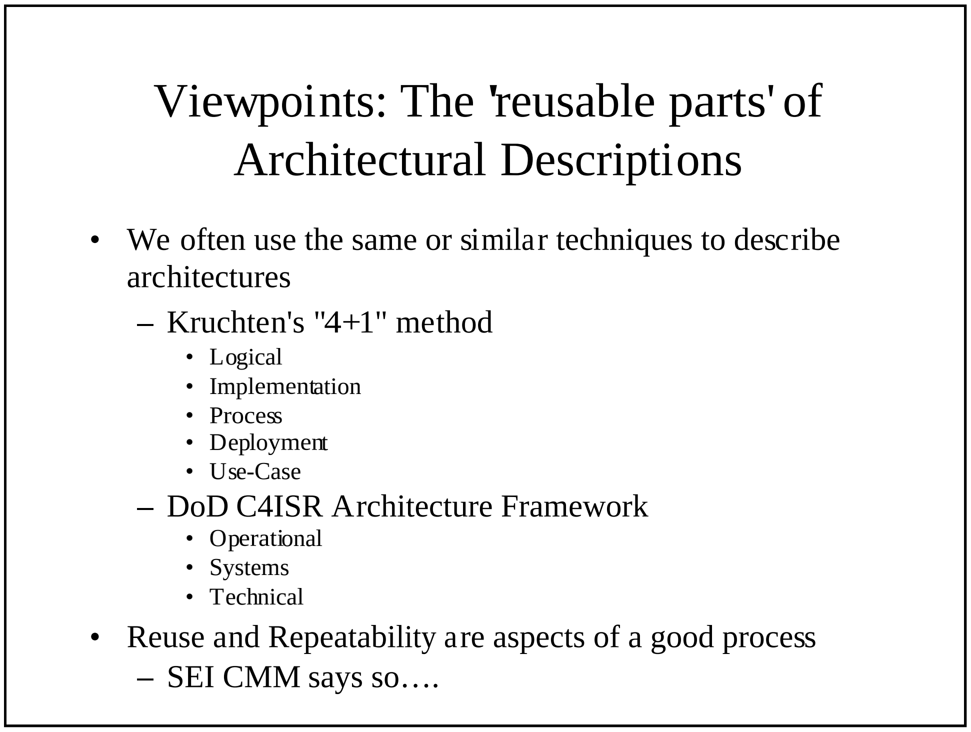
- Concentrates on -describing- architectures
- Major contributions:
  - Architecture: The fundamental organization of a system, embodied in its components, their relationships to each other and to their environment and the principles guiding its design and evolution.
  - Architectural Descriptions require multiple views
  - Views describe the system with respect to concerns of the system's stakeholders
  - The contents of views should be specified by a (system-independent) viewpoint definition

The IEEE recently approved a new 'standard'. It's a recommended practice (the 'lowest level' of standard issued by the IEEE) and it is focused on architectural descriptions. IEEE 1471 tells you how to describe an architecture, but not how to generate the description. Thus it specifies content, but not process.

This talk should help motivate and justify the main contributions/points of IEEE 1471.

It turns out that the single most contentious issue in the entire standardization effort was to get to an agreed-upon definition for "architecture". But the definition of "architecture" is secondary to IEEE 1471's real intent, which is how to describe an 'architecture', whatever you think an architecture is.

Our IEEE reviewers titled the standard "Recommended Practice for Architectural Descriptions of Software Intensive Systems." This was to ensure that its focus was clear to people seeing this standard listed in the IEEE Standard Association's catalog. IEEE 1471 is clearly applicable to software-intensive systems, and we think that it might well have applicability in other domains, also. (I presented a draft of the standard to a group of EE/ME students working on intelligent transportation systems and robots, and their reaction was that it was very useful for the kinds of things they were trying to do in hardware.)



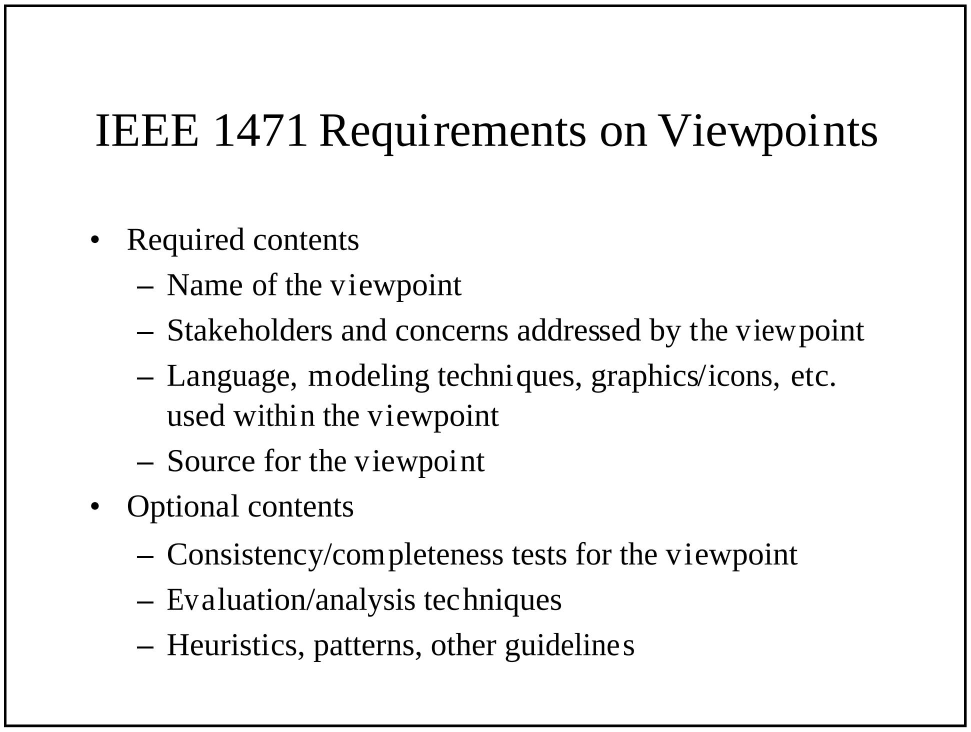
Viewpoints: The 'reusable parts' of Architectural Descriptions

- We often use the same or similar techniques to describe architectures
  - Kruchten's "4+1" method
    - Logical
    - Implementation
    - Process
    - Deployment
    - Use-Case
  - DoD C4ISR Architecture Framework
    - Operational
    - Systems
    - Technical
- Reuse and Repeatability are aspects of a good process
  - SEI CMM says so….

The key new contribution of IEEE 1471 is the recognition of "viewpoints" as the reusable parts of architectural descriptions. In Ada terms, viewpoints are generics, that you have to instantiate to use.

Kruchten's "4+1" was one of the first attempts to specify the viewpoints for architectural descriptions. ISO Reference Model for Open Distributed Computing (RM ODP) is another example. The US DoD C4ISR Architecture Framework also serves as a 'viewpoint specification'. Depending on how you read the document, it specifies 3 viewpoints with a total of 47 representations, or it specifies 47 different viewpoints.

We know from our SEI CMM that reuse and repeatability are key parts to any good process, so having viewpoint definitions that we can apply for doing architectural descriptions contribute to good processes. But IEEE 1471 does not specify how viewpoints are discovered, cataloged or used, only that the description document should identify a viewpoint definition for each view in the description.

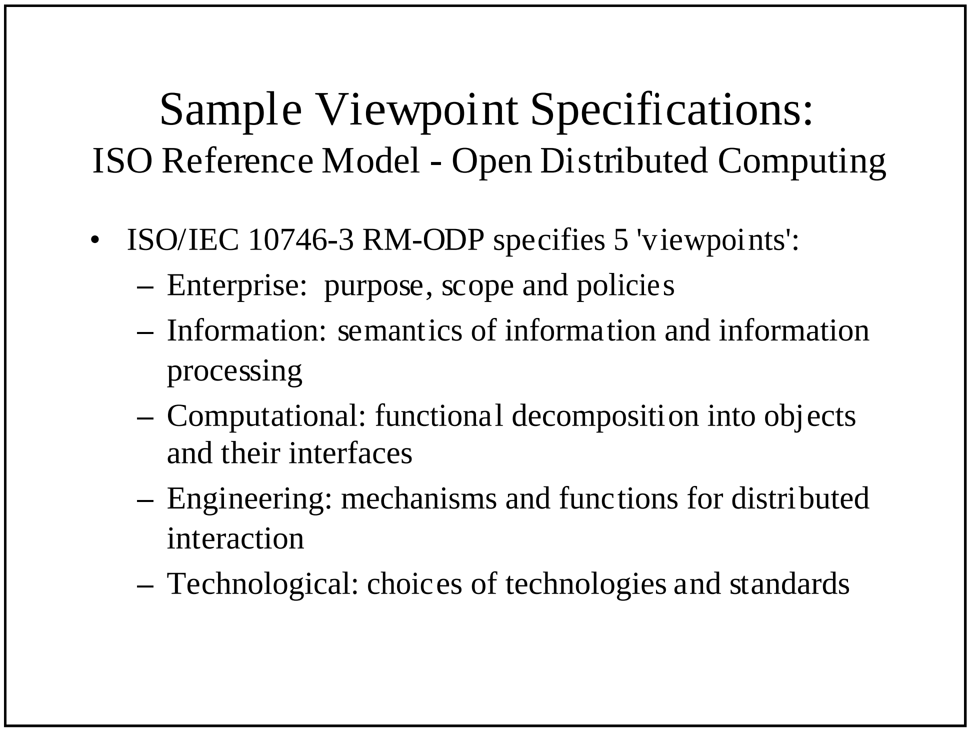


IEEE 1471 Requirements on Viewpoints

- Required contents
  - Name of the viewpoint
  - Stakeholders and concerns addressed by the viewpoint
  - Language, modeling techniques, graphics/icons, etc. used within the viewpoint
  - Source for the viewpoint
- Optional contents
  - Consistency/completeness tests for the viewpoint
  - Evaluation/analysis techniques
  - Heuristics, patterns, other guidelines

The idea of having separate architectural descriptions is relatively new, and we are still gaining experience and building consensus about what works. IEEE 1471 has a relatively small set of normative requirements. This slide lists the requirements for a viewpoint specification. Views also have the same requirements.

A viewpoint must tell you how to describe the view. It can provide optional guidance on how to analyze or evaluate the view, and it can provide hints on useful techniques for such views.



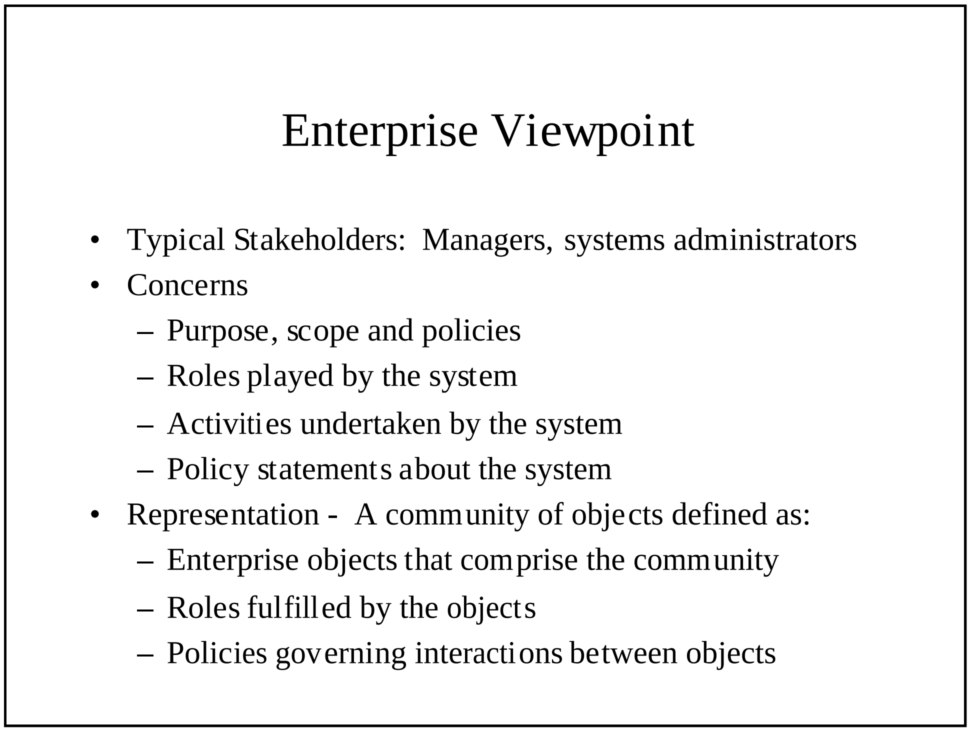
Sample Viewpoint Specifications:

ISO Reference Model - Open Distributed Computing

- ISO/IEC 10746-3 RM-ODP specifies 5 'viewpoints':
  - Enterprise: purpose, scope and policies
  - Information: semantics of information and information processing
  - Computational: functional decomposition into objects and their interfaces
  - Engineering: mechanisms and functions for distributed interaction
  - Technological: choices of technologies and standards

We have many existing examples of viewpoints. I thought it would be useful to apply the IEEE 1471 model to some examples of viewpoint specifications.

ISO RM ODP specifies 5 viewpoints, as shown. We'll describe each one using the IEEE 1471 viewpoint specification requirements, based on words from the RM ODP specification.

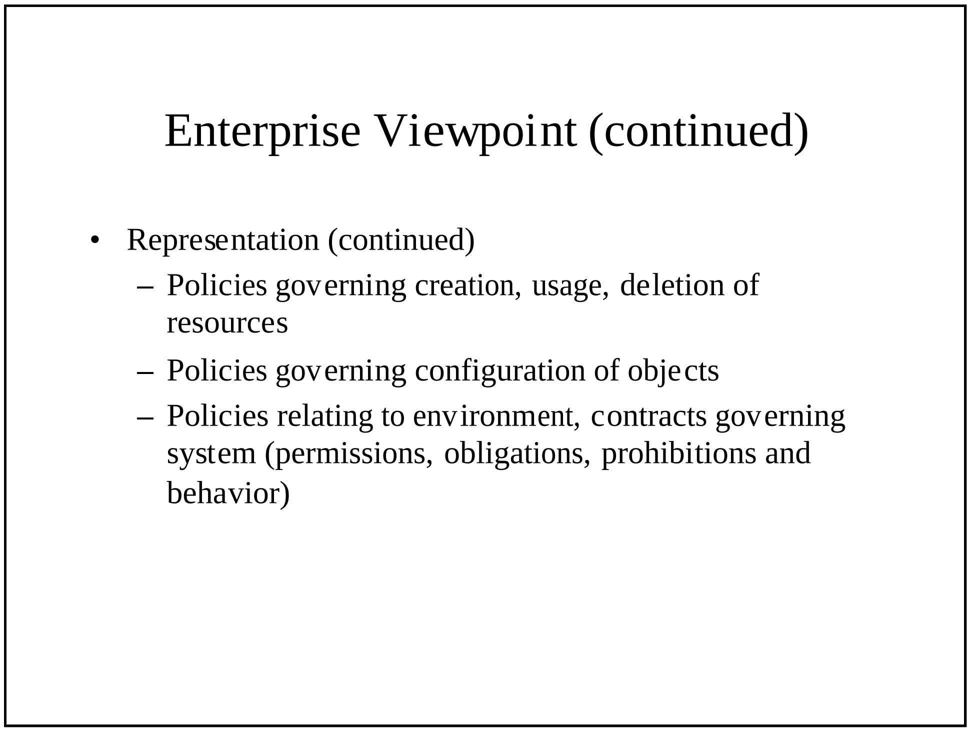


Enterprise Viewpoint

- Typical Stakeholders: Managers, systems administrators
- Concerns
  - Purpose, scope and policies
  - Roles played by the system
  - Activities undertaken by the system
  - Policy statements about the system
- Representation - A community of objects defined as:
  - Enterprise objects that comprise the community
  - Roles fulfilled by the objects
  - Policies governing interactions between objects

The Enterprise Viewpoint exists to capture the role of the system within the enterprise. This is an important part of 'architecture', since it's the architect's job to ensure that the system is useful for its intended purpose. Thinking back to the building industry, the role of the architect in designing a church is to fit the structure onto the lot, to ensure that it has the right ambience of a church (including specific religious requirements), and to make sure that people "fit" in the building. The engineer assumes that the architect has designed a structure that is useful/suitable as a church, and concentrates on building the structure that the architect has specified.

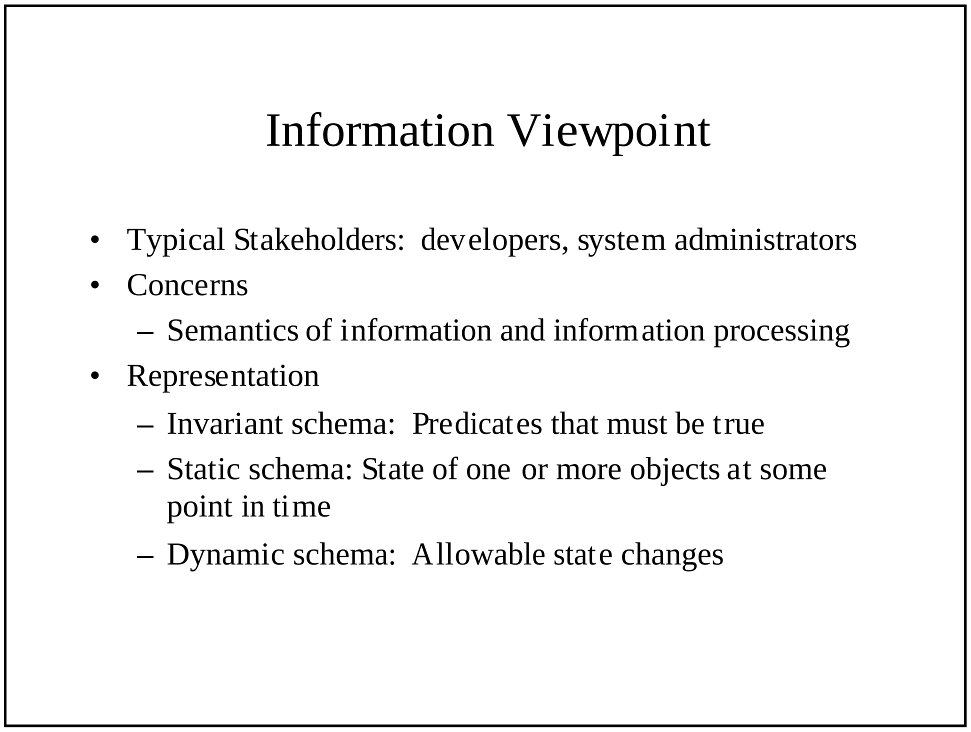
Thus the Enterprise Viewpoint in RM ODP focuses on these kinds of issues of 'suitability for intended use' of the system within its using enterprise.



Enterprise Viewpoint (continued)

- Representation (continued)
  - Policies governing creation, usage, deletion of resources
  - Policies governing configuration of objects
  - Policies relating to environment, contracts governing system (permissions, obligations, prohibitions and behavior)

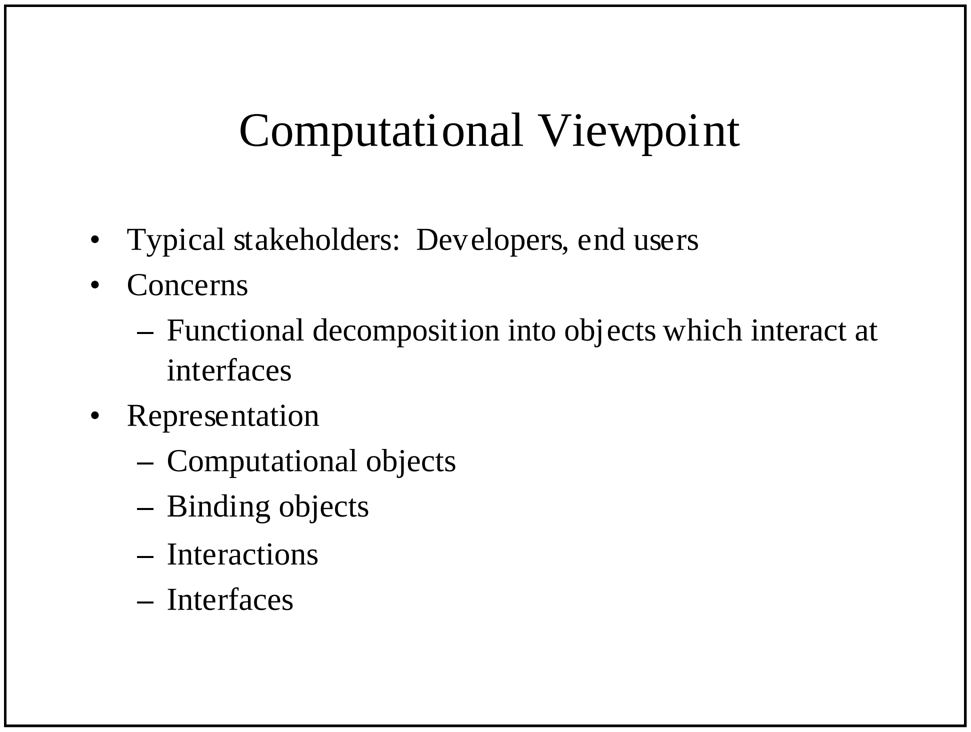
Much of the RM ODP Enterprise Viewpoint focuses on policies and similar non-pictorial kinds of information about the system.



Information Viewpoint

- Typical Stakeholders: developers, system administrators
- Concerns
  – Semantics of information and information processing
- Representation
  – Invariant schema: Predicates that must be true
  – Static schema: State of one or more objects at some point in time
  – Dynamic schema: Allowable state changes

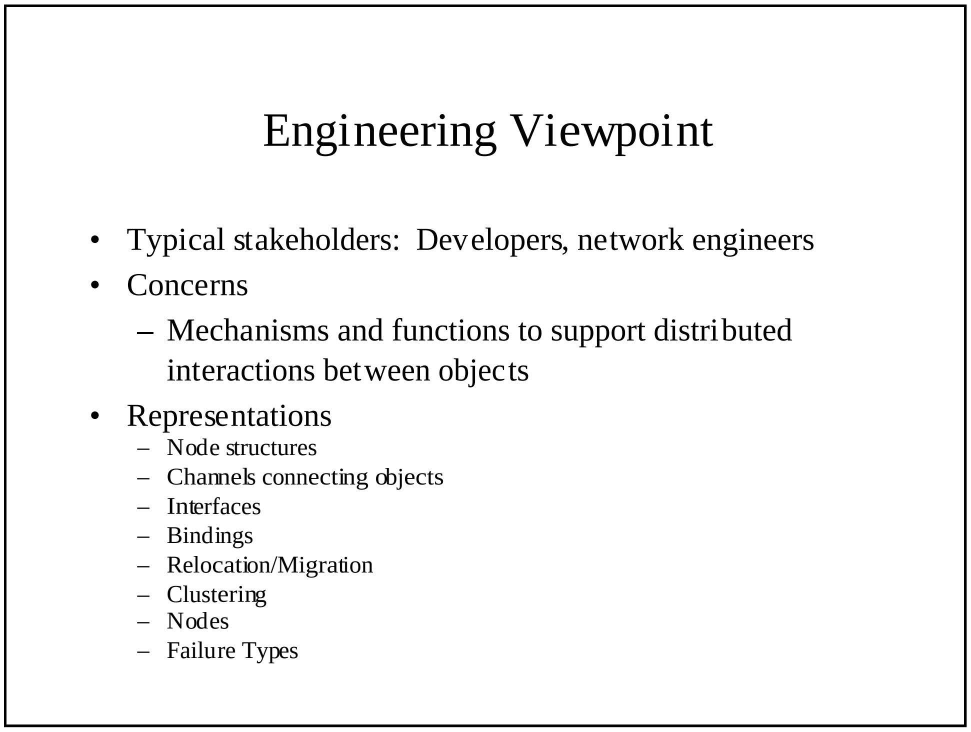
The Information Viewpoint exists to capture information/data content of the system. Note that the viewpoint specifies 3 different kinds of schemas. The collection of schemas try to handle the dynamics of the system, as well as the traditional database schema information.



Computational Viewpoint

- Typical stakeholders: Developers, end users
- Concerns
  - Functional decomposition into objects which interact at interfaces
- Representation
  - Computational objects
  - Binding objects
  - Interactions
  - Interfaces

The Computational Viewpoint tends to resemble the "functional view" of a system, capturing various kinds of objects and the interfaces that they provide. The RM ODP specification provides definitions for "computational objects" and "binding objects."

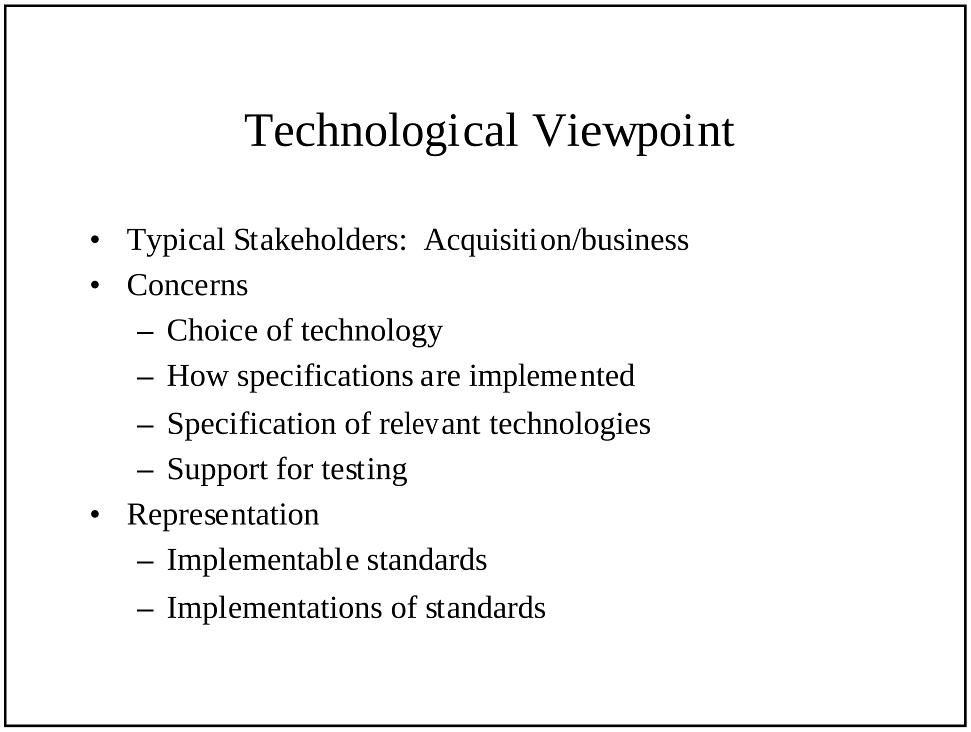


Engineering Viewpoint

- Typical stakeholders: Developers, network engineers
- Concerns
  - Mechanisms and functions to support distributed interactions between objects
- Representations
  - Node structures
  - Channels connecting objects
  - Interfaces
  - Bindings
  - Relocation/Migration
  - Clustering
  - Nodes
  - Failure Types

The Engineering Viewpoint shows how the system is constructed and behaves. It captures the 'nodes and connections' kind of information that we saw in some of the example representations. It also captures rules about how the system itself is constructed/engineered, akin to the "process" we showed earlier.

Again, the RM ODP specification defines the meanings of the various representations.



Technological Viewpoint

- Typical Stakeholders: Acquisition/business
- Concerns
  - Choice of technology
  - How specifications are implemented
  - Specification of relevant technologies
  - Support for testing
- Representation
  - Implementable standards
  - Implementations of standards

The Technological Viewpoint is very interesting. It captures not only specific technologies and standards of interest for the system, but also discusses how those standards and technologies can evolve and affect the system over time. It distinguishes between the standards of interest and implementations of those standards.

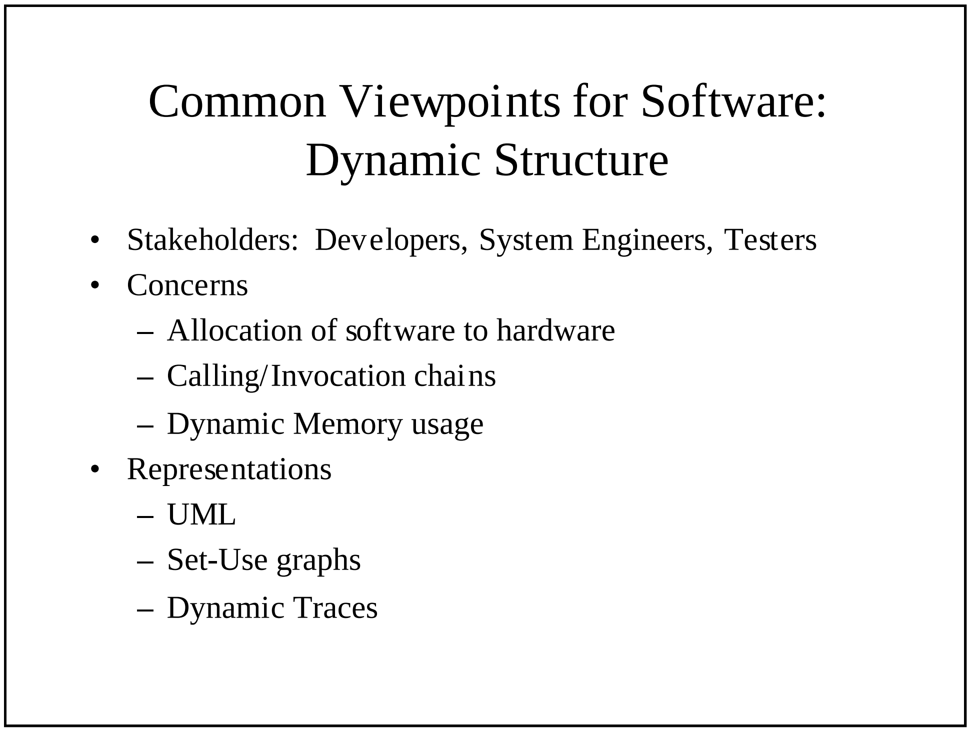
Common Viewpoints for Software: "Static Structure"

- Stakeholders: Developers, Testers, Integrators, Managers
- Concerns
  - Allocation of requirements
  - Work-breakdown structure
- Representations
  - UML
  - Directory/filesystem structure

Much of the work on software architectures has been focused on the structure of the software. (See, for example, the CMU/SEI publications on software architecture.) In fact, some claim that "software architecture" is explicitly about (only) the structure of the software. (I do not hold to this view. I see structure as only 1 part of the "holistic" nature of architecture. Just as a church or office building is not completely described by its framing, I don't believe that the architecture of a software system is completely described by its structure. How do you handle such issues as "security," "performance" or "maintainability" with only a structural perspective on the system?)

This viewpoint captures the general concerns of a "Structural viewpoint". Note that it deals with both the layout of the software during its construction (e.g. work breakdown structures, directory structures) and with the software when inserted into the system (e.g. UML representations of the system showing the relationships between the software components.)

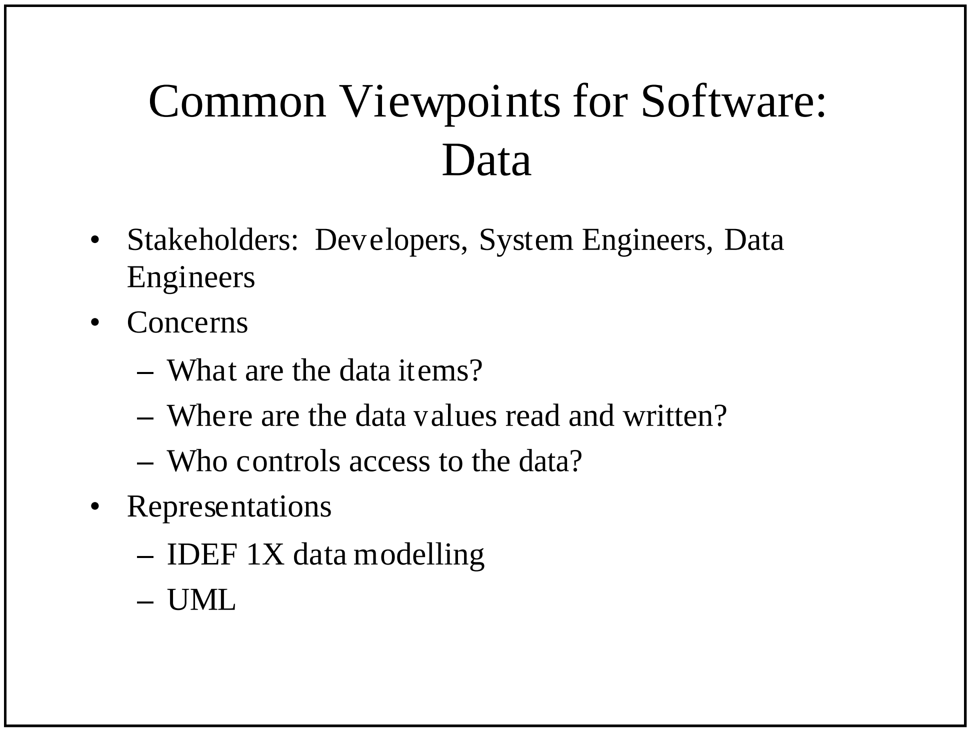
Even within the notion of 'structure,' we need two separate views. This view concentrates on things that are determinable from the source code itself, such as the "with" dependencies. The next view talks about the structure of an executing system.



Common Viewpoints for Software: Dynamic Structure

- Stakeholders: Developers, System Engineers, Testers
- Concerns
  - Allocation of software to hardware
  - Calling/Invocation chains
  - Dynamic Memory usage
- Representations
  - UML
  - Set-Use graphs
  - Dynamic Traces

To fully capture the software structure, you need two views. One captures the static software layout, and the other captures the dynamics of a running system. Consider a system using Ada tasking. The static structure establishes the packages where the task types are declared, the units that can create task instances, etc. The dynamic structure shows, for some point in the execution of the program, the tasks that are active at that instant. Other issues addressed by dynamic structure include resource usage (e.g. dynamic memory) and performance issues.



Common Viewpoints for Software: Data

- Stakeholders: Developers, System Engineers, Data Engineers
- Concerns
  – What are the data items?
  – Where are the data values read and written?
  – Who controls access to the data?
- Representations
  – IDEF 1X data modelling
  – UML

Most software systems also have a "data view," as captured in this viewpoint. Traditional data modeling captures the definitions of data items. It is quite common to extend this to cover issues such as data ownership, readers and writers, access control, etc. IDEF-1X and UML are very common representations used in data views.

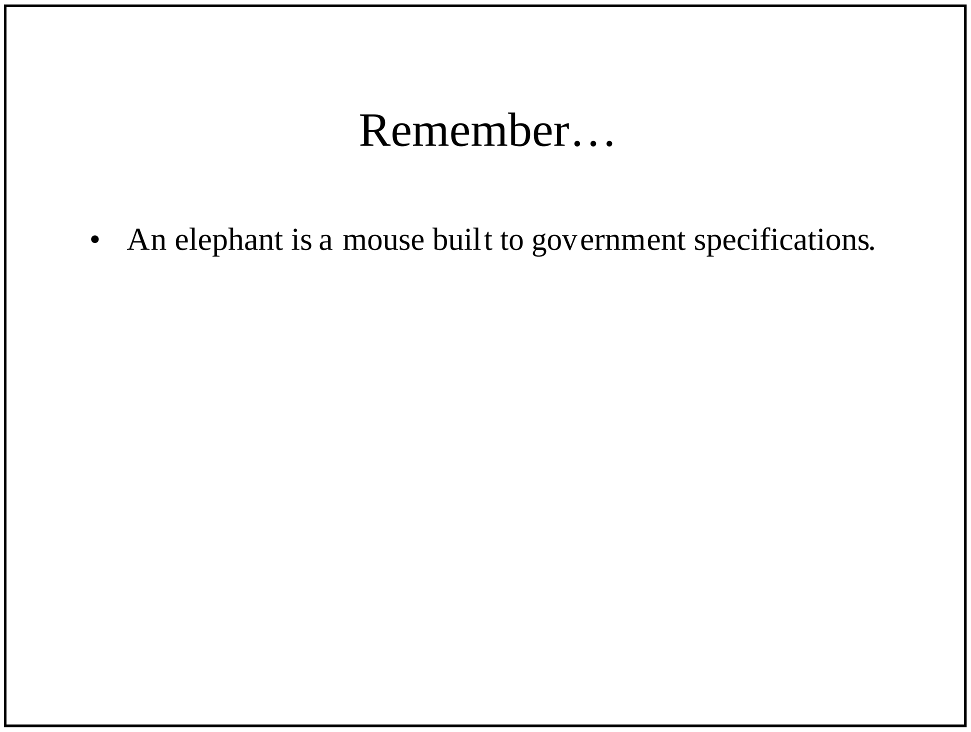
In summary...

- Multiple views are needed to fully describe an architecture
  - Each view brings a different perspective on the architecture
  - Views exist to answer questions about the architecture
- Reusable viewpoints provide a starting point for each architectural description
  - Consistency across architectural descriptions
  - Reduced training costs for architects and clients
- IEEE 1471 provides a framework for architectural descriptions
  - You have to provide the process

The intent of this talk is to motivate the concept that multiple views are required to adequately describe architectures. I started by showing examples of views, comparing them to the blind men and the elephant. Each view captures the system, but only from the perspective of the view's stakeholders and concerns. The concentration here is not on the abstract notion of "architecture" but on the concrete representations of architectures, that we can use for comprehension, comparison and analysis.

IEEE 1471 captures the multi-view concept of architectural descriptions. It introduces the notion of a 'viewpoint,' a 'generic' that can be 'instantiated' to define the contents of views for a given system architectural description. By sharing/reusing viewpoints, we can gain repeatability in the architectural process, and enhance interoperability and understanding among software/system architects.

IEEE 1471 defines requirements on the software architecture description product, but does not define or mandate any process for producing architectural descriptions. Future work in the IEEE's Architecture Working Group includes a guidebook that recommends a process and techniques for producing architectural descriptions. My SIGAda 2000 tutorial describes one such process.



Remember…

- An elephant is a mouse built to government specifications.

Finally, let's remember that one of the ideas behind architecture is to get our hands around the entire system, to ensure it's suitable for use. Viewing the blind men's description of the architecture of an elephant should help you realize this is not a mouse...

References:

Hilliard, R. "Using the UML for Architectural Description", *Proceedings of <<UML>> '99*, Lecture Notes in Computer Science 1723, Springer Verlag.

IEEE (Architecture Working Group) *IEEE Std 1471-2000 Recommended Practice for Architectural Description of Software Intensive Systems*, IEEE, Sept 2000.

ISO *ISO/IEC 10746-3:1996 Open Distributed Processing Reference Model - Architecture*, International Standards Organization, 1996.

U.S. DoD. *C4ISR Architecture Framework, Version 2.0*, <date?>

Walker, R. <paper on CADM>

Kruchten, P. *The "4+1" View Model of Softw.are Architecture*, in *IEEE Software*, 12(6)

Buschmann, F, et.al. *A System of Patterns: Pattern-Oriented Software Architecture*, Wiley, 1996.

Shaw, M. and Garlan, D, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996

Rechtin, E, and Maier, M, *The Art of Systems Architecting*, CRC Press, 1997. (Revised version just released by CRC Press.)

Emery, D. <Santander Paper>