Documenting Software Architecture (according to the SEI) (with a little added context)

resources:

http://www.sei.cmu.edu/library/abstracts/podcasts/dsapodcast.cfm

http://www.sei.cmu.edu/library/assets/DSA_podcast.mp3

http://www.sei.cmu.edu/reports/04tr008.pdf (component/connector views)

https://wiki.sei.cmu.edu/sad/index.php/High_Level_Module_View

http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/

(this topic is in Part 2 Chapter 9 “Documenting Software Architectures”)

https://sites.google.com/site/softwarearchitectureinpractice/9-documenting-software-architecture/a-intro

http://users.ece.utexas.edu/~perry/work/papers/swa-sen.pdf

Software Engineering Institute

Introduction to Paul Clements

http://www.sei.cmu.edu/library/abstracts/podcasts/dsapodcast.cfm

Documentation is Important

- You need to be able to **communicate** your software architecture
- if you can’t, your architecture is useless, because people won’t be able to understand it

Why document?

- S/A serves as the blueprint for the system
- Also, as the blueprint for the *project that develops the system*
- Defines work assignment
- Primary carrier of quality attributes
- Blueprint needs to be understood to be carried out.

Documenting helps Development

- As you write the documentation for the architecture, the architecture itself takes shape
- It serves as a point of communication between developers, to iterate towards a final concept for the architecture.
- Capturing as we create it makes for higher quality architecture and helps architect

How to document?

- View based approach: multi-dimensional construct
- Too complex to be seen all at once.
- Systems composed of many structures simultaneously. (modules/component/process/deployment structures)
- Views allow separate capture of various dimensions
- Views are representations of structures
- Used to manage complexity to separate concerns

Basic Principle

- Documenting software architecture involves:
  - documenting the relevant **views**
  - adding information that applies to more than one view (**across views**, or beyond views)

Views: Pedigree (Not a New Thing)

- 1974 - Parnas “Software is composed of structures”
- 1992 - Dewayne Perry and Alexander Wolf “Architecture is like buildings, and require different views” http://users.ece.utexas.edu/~perry/work/papers/swa-sen.pdf
- 1995 - Philippe Krutchen proposed 4+1 views
- 2000 - IEEE adopted 1471 standard - View-based approach to S/A documentation

What is a “view”?

- A view is a representation of a particular structure
- A representation of a set of system elements and the relationships between them.
- Not all elements - just certain kinds, at certain times.

View, in a nutshell

Beyond primary presentation of box and line

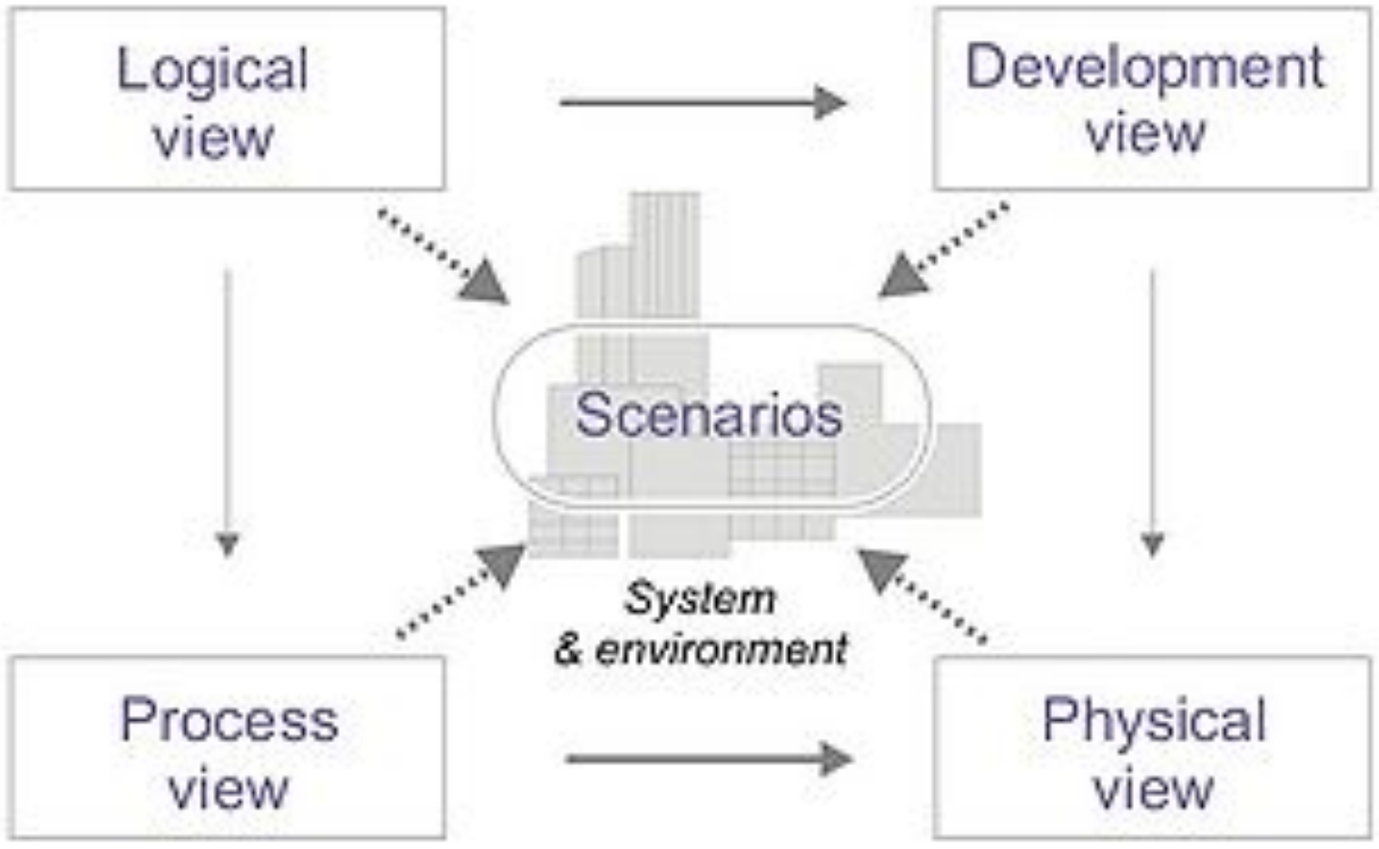
- Initially ask: what do the boxes mean, what do the lines mean?
- Note: Software Architectural Views have no prescribed syntax -- must establish through communication or clear definition within the artefact
- Must establish what elements and what connections types are you showing.

So what are some “views” to use?

- Lots to choose!!!
- Some authors prescribe a particular set of views
    - Philippe Krutchen’s 4+1
    - Siemens 4-view approach
    - Software cost reduction project (US Navy) suggests 3 views
    - Phillips has 5 views (customer, application, functional, conceptual, realization)

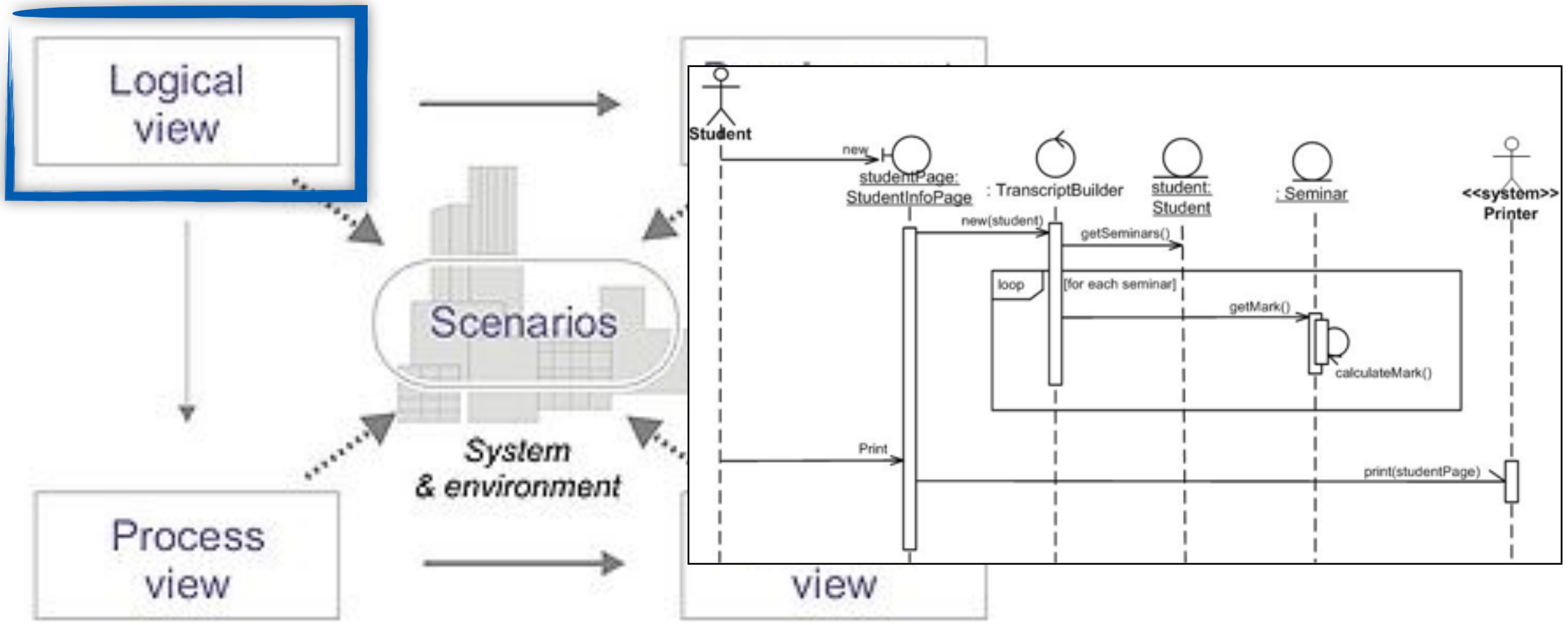
View Paradigm Example:

4+1 by Philippe Krutchen



http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

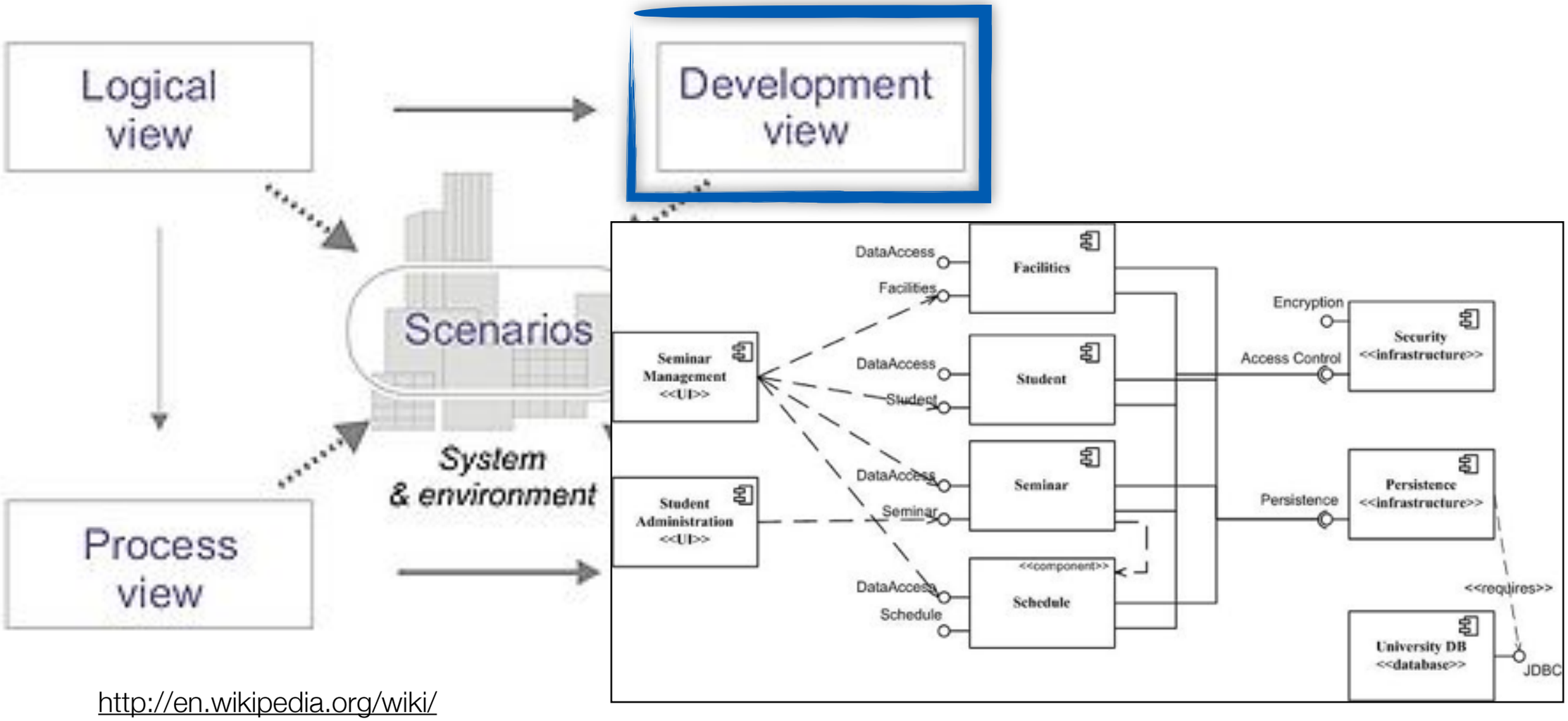
View Paradigm: 4+1



http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

functionality provided to the customer. Includes UML diagrams, such as class diagram, sequence diagram, and communication diagram.

View Paradigm: 4+1

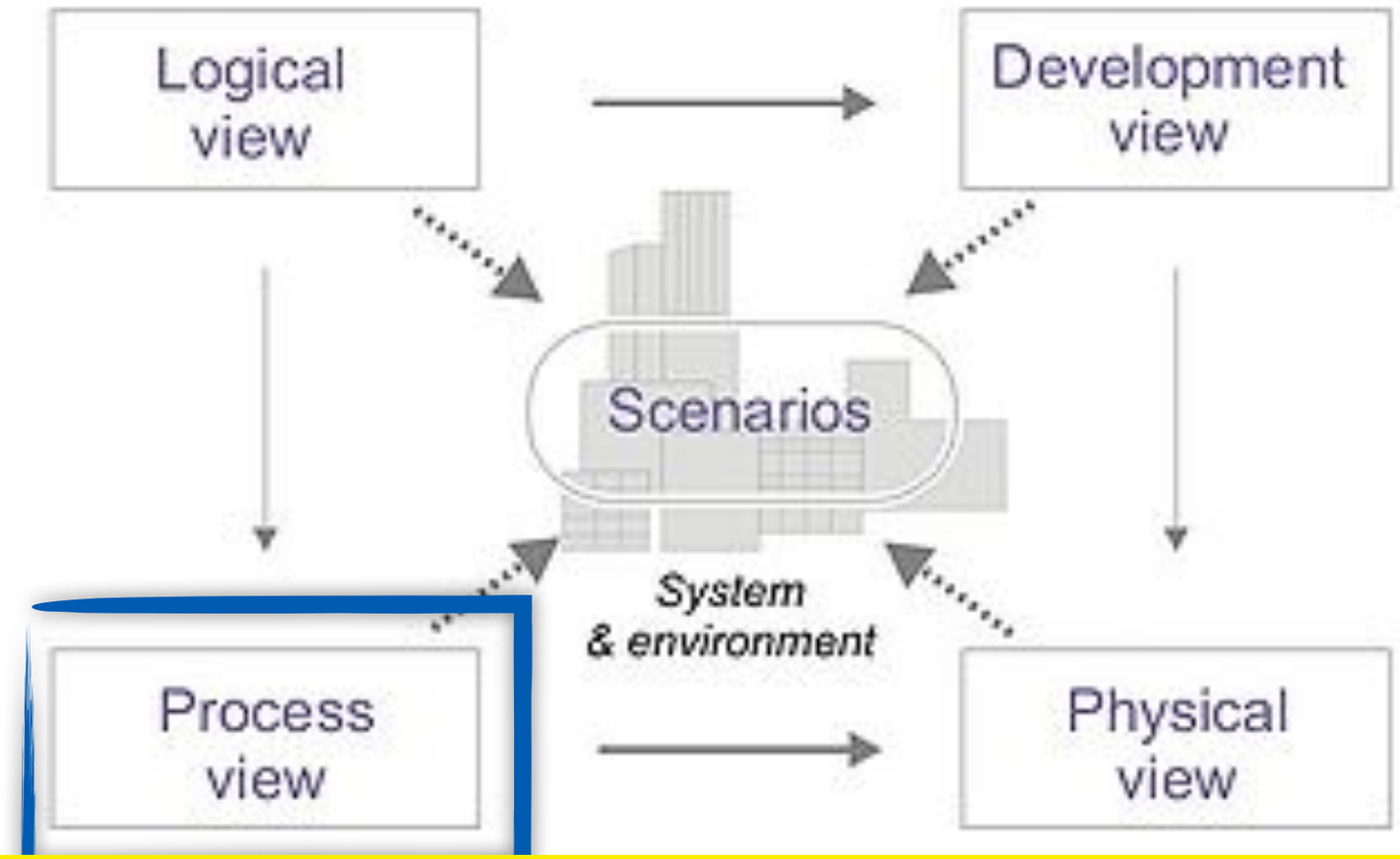


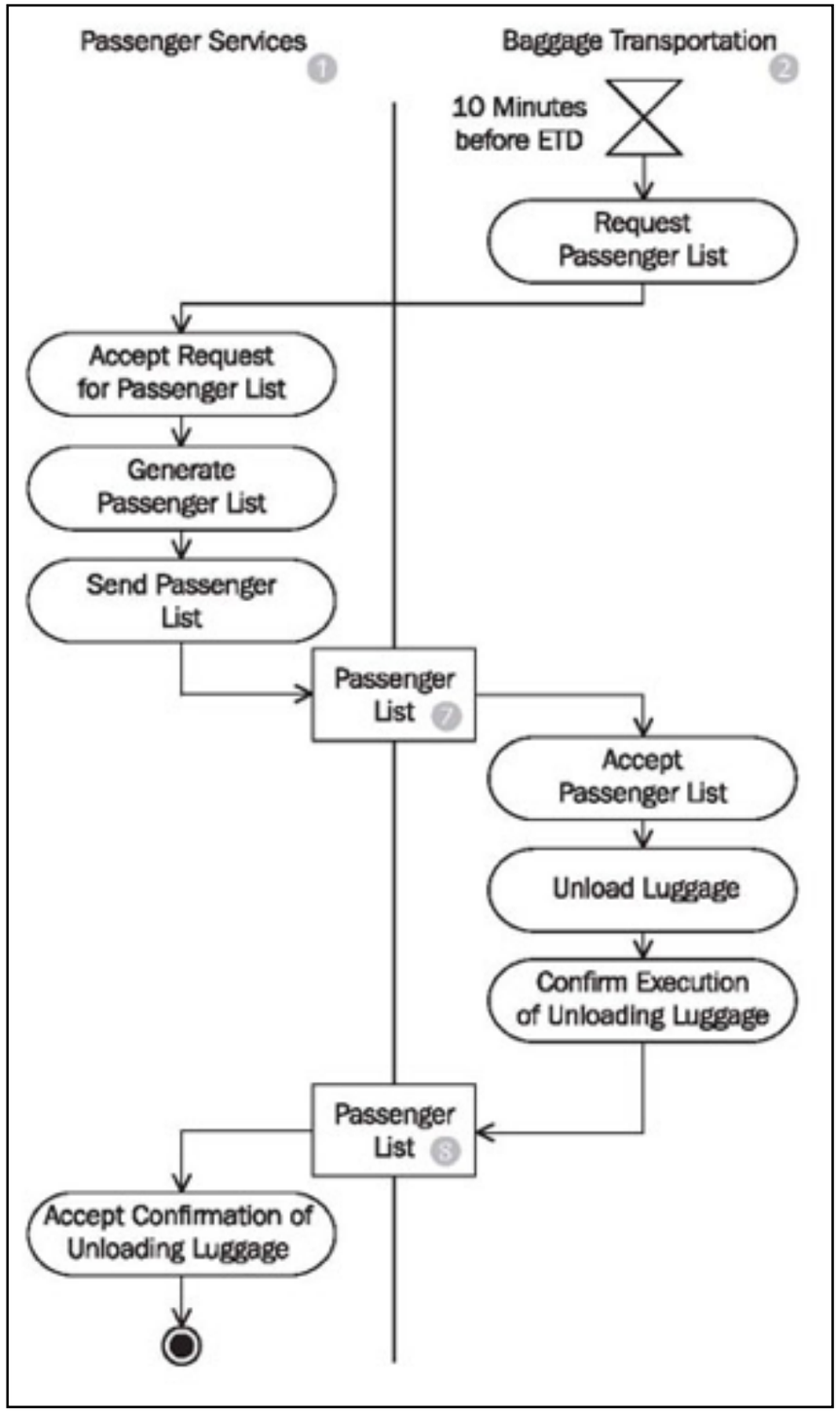
http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

AKA Implementation view. System from programmers perspective. Shows components and packages.

View Paradigm: 4+1

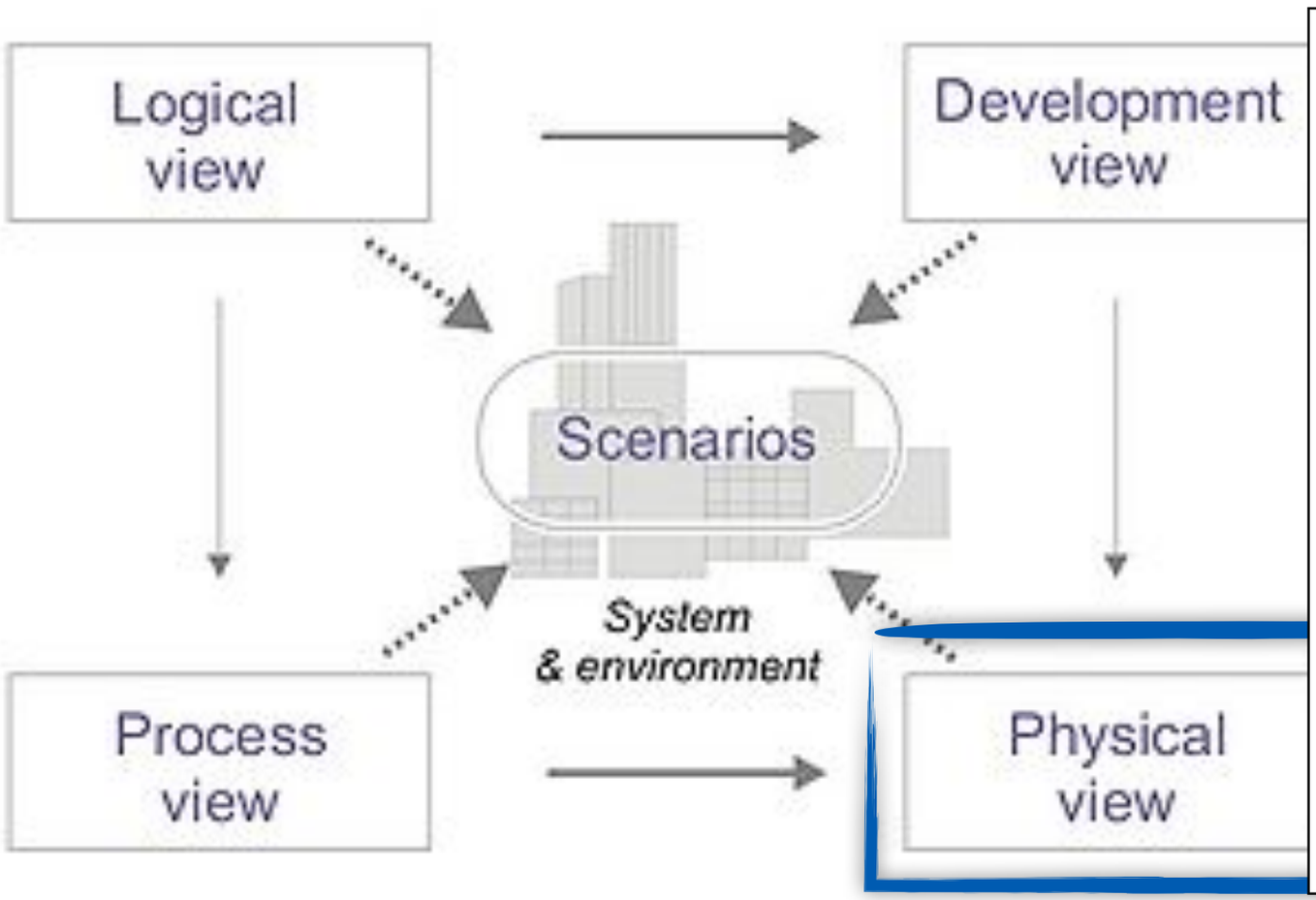
http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

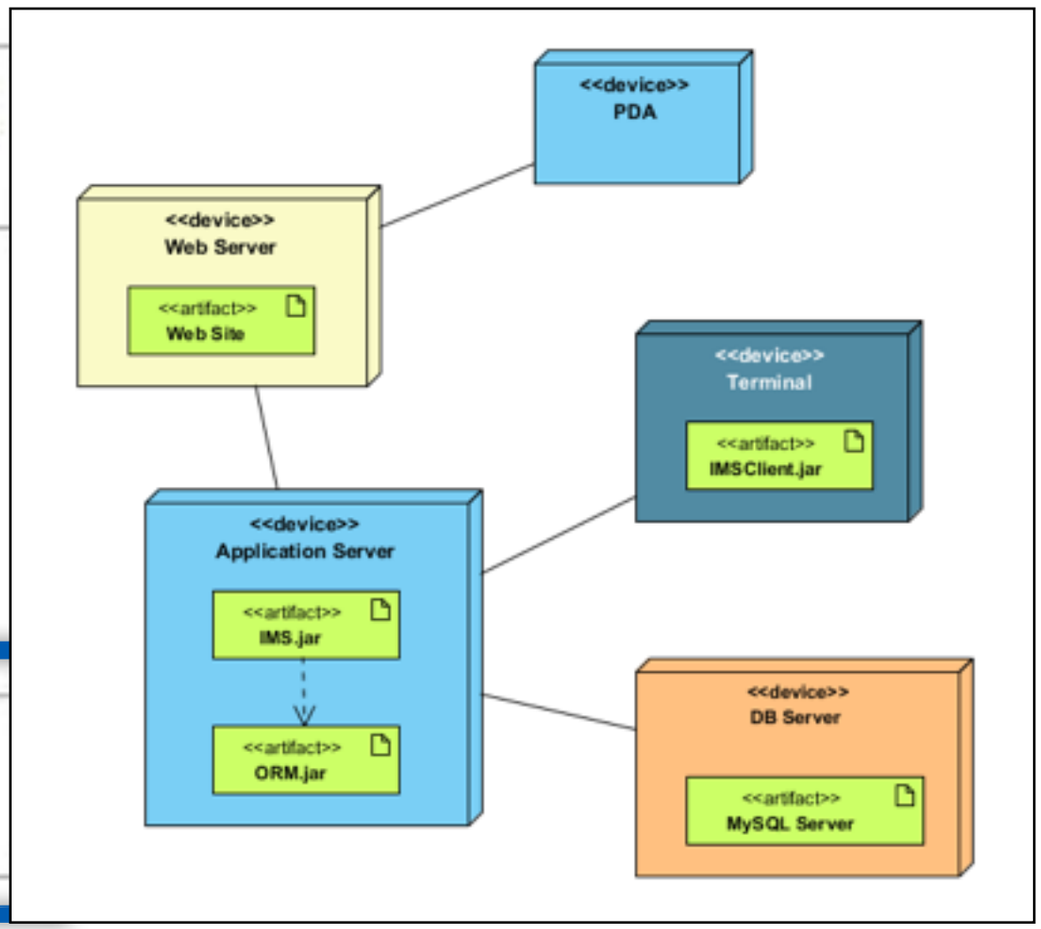




AKA Activity Diagram. Focuses on runtime behaviour of the system. How components communicate. Addresses concurrency, scalability, performance, etc. Helps capture business process if that is relevant.

View Paradigm: 4+1

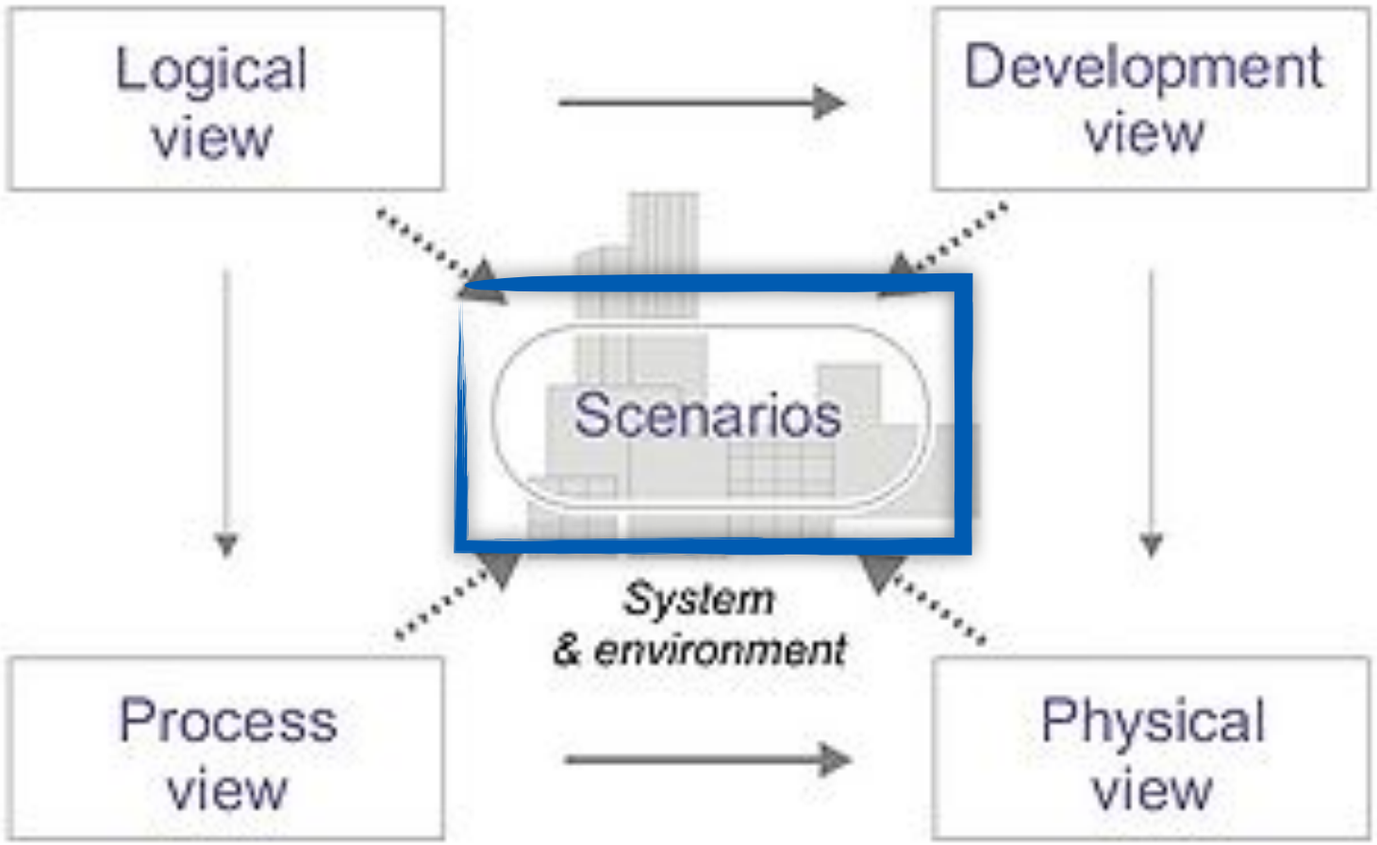




http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

AKA Deployment view. System engineers point of view. Topology of software components, physical connections between those components.

View Paradigm: 4+1



http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

View #5! Use case view, User stories, Scenario diagrams. Helps with architectural validation (almost a start of a test suite at a high level).

How to choose?

- Choose the views that are appropriate for your situation.
- Depends on:
    - structures inherent in the software architecture. Example: if uniprocessor, then “deployment view” is not interesting because all software will run on the same box. However, if you’re building a layered system, then a deployment view would be helpful.
    - depends on stakeholders and what they need to know. What they’re going to use them for.

Views are to be *Engineered*

- Views typically centre on quality attributes (performance, modifiability)
- By engineering the view, you achieve those quality attributes
- And then the resultant view shows how those quality attributes were achieved.

Who are potential stakeholders for views?

- **Who is involved in the system, and what do they need to know**
    - Management might want to track progress
    - Performance analyst to see if system is going to meet its deadlines
    - Need to document the views reflecting the structures in the system that are meaningful to those stakeholders.
    - Developers might need to know how their components interact with the rest of the system, or what the responsibilities are for their feature.

Important: Separation of Concerns.

- SEI suggests 3 kinds of views:

  Module | **Component/Connector** | **Allocation**

- Blurring these concerns (intentionally or unintentionally) causes confusion, and makes diagrams unclear or difficult to understand.

- Maintaining the 3-way categorisations helps maintain clarity.

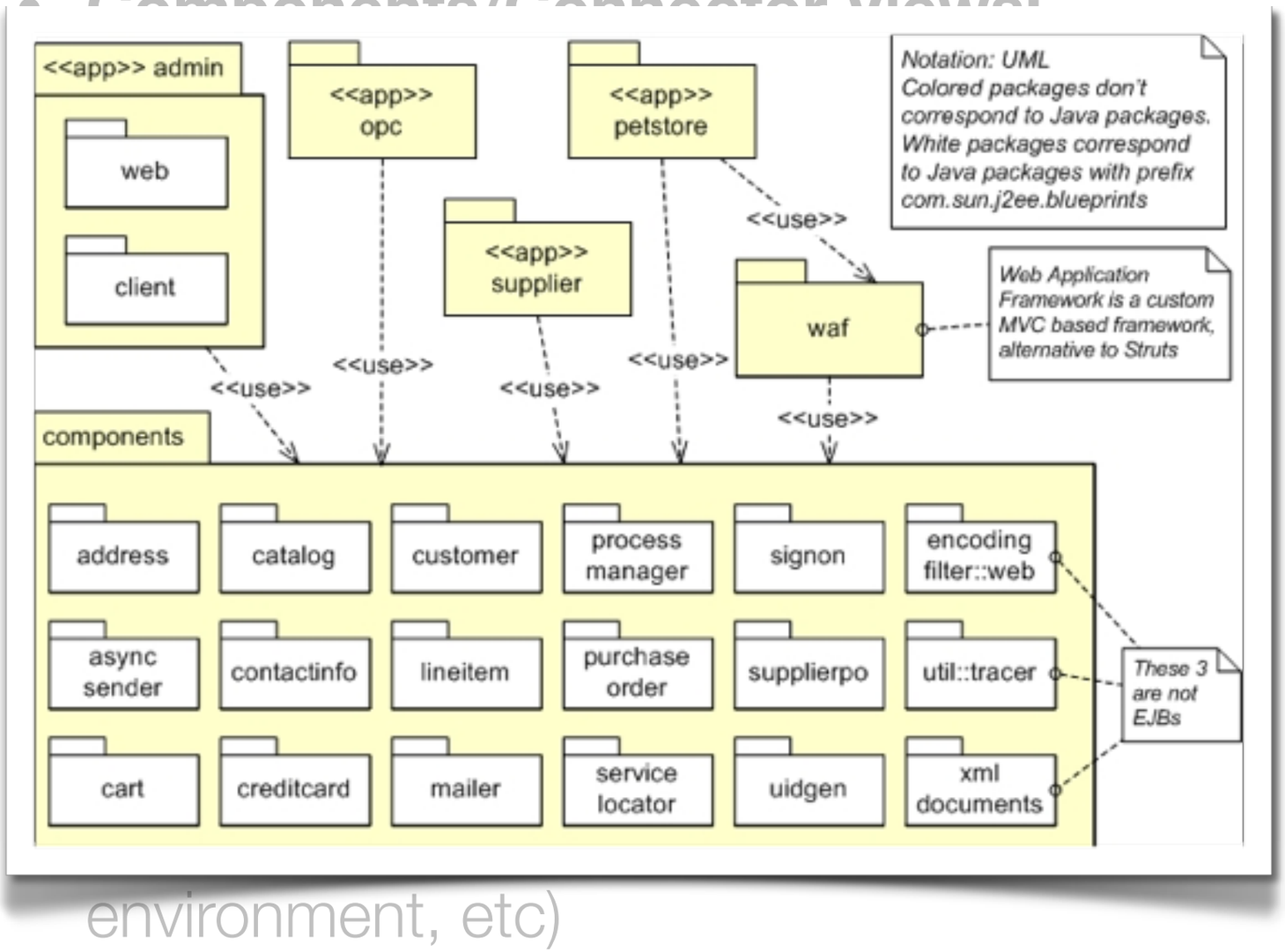
Kinds of views...

- **Module Views :** How system is structured as a set of implementation units. Things people need to go off and build, or go and buy.

https://wiki.sei.cmu.edu/sad/index.php/High_Level_Module_View

https://sites.google.com/site/softwarearchitectureinpractice/9-documenting-software-architecture/b-module-views

- Components/Connector Views:



environment, etc)

- **decomposition** view (looks at module decomposition)
- **uses** view (looks at how modules use one another)
- **generalisation** view (is-a relationships)

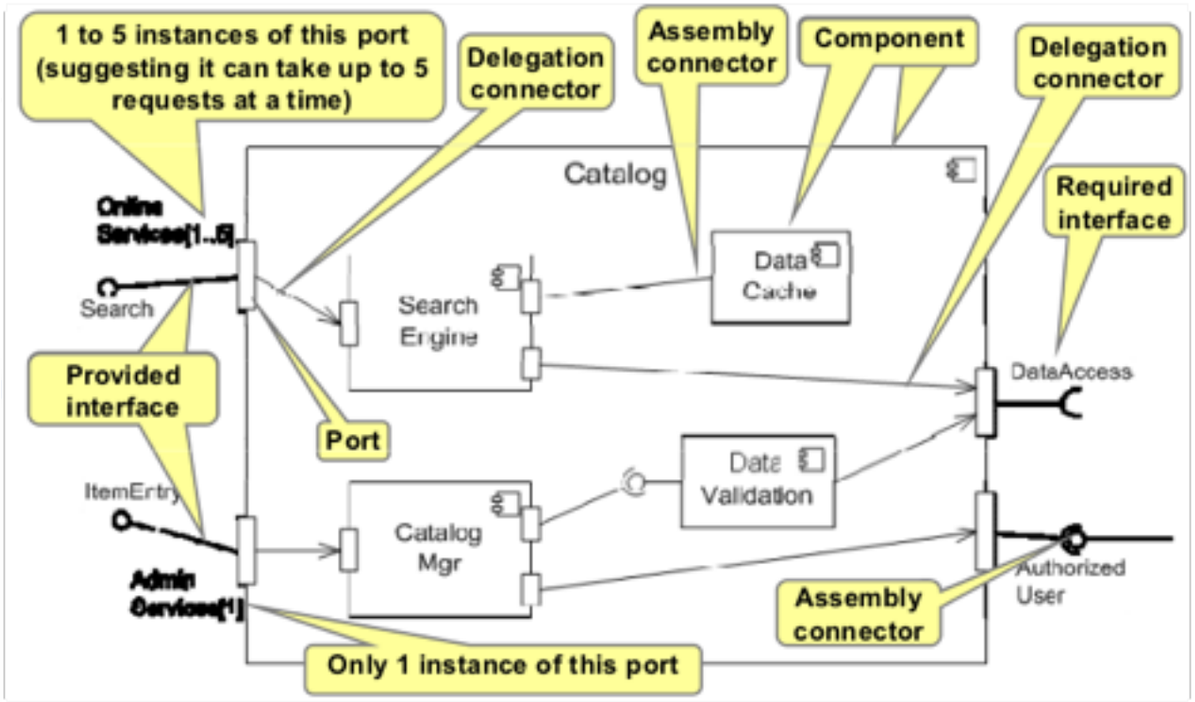
Kinds of views...

https://sites.google.com/site/softwarearchitectureinpractice/9-documenting-software-architecture/c-component-and-connector-c-c-views

- **Module Views :** How system is structured as a set of implementatio units. Things people need to go off and build, or go and buy.

- **Components/Connector Views:** how software is structured as a set of elements that have runtime behaviour and interactions. What does the system *do* and what are the pieces involved in making it do those things?

- **Allocation Views:** how does it relate to non software structures in its environment (execution environment, its development environment, etc)



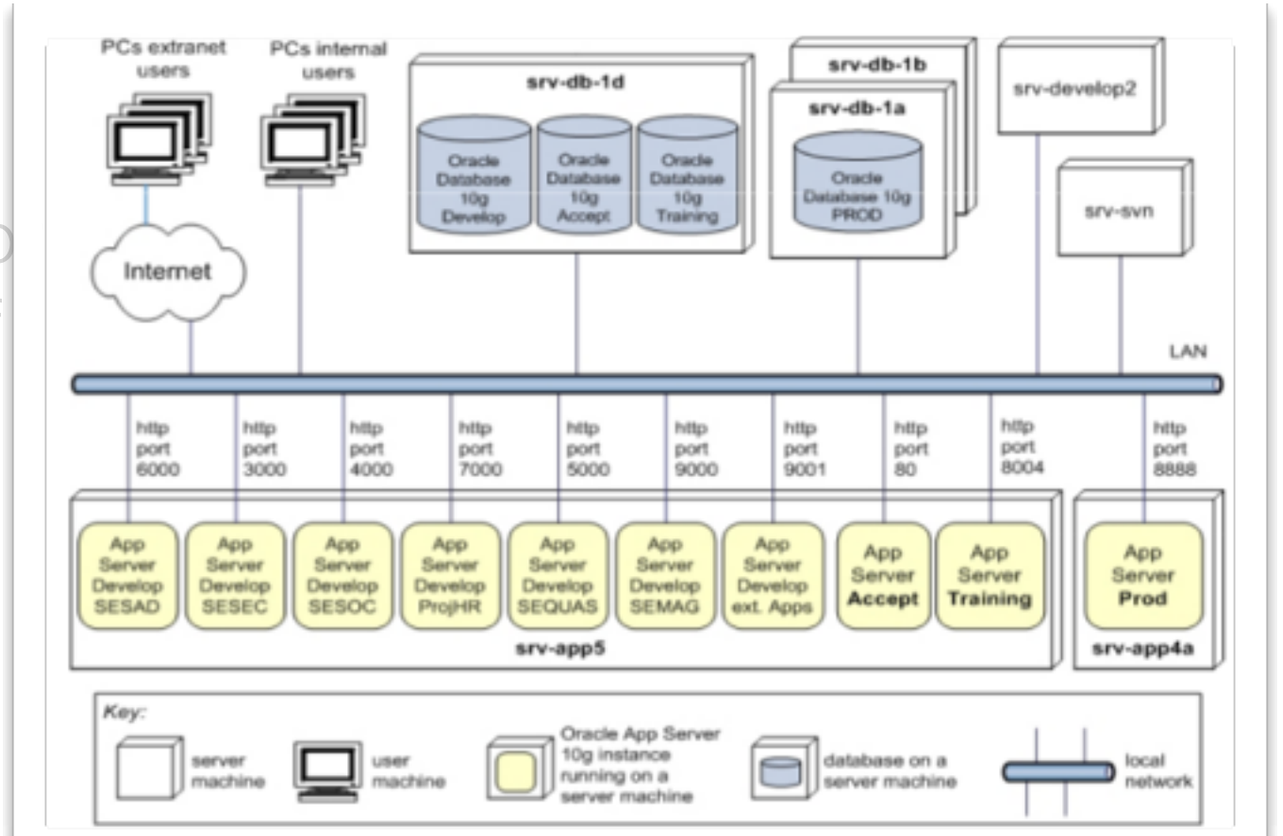
- **shared data** view
- **service-oriented architecture** view
- **pipe and filter** view

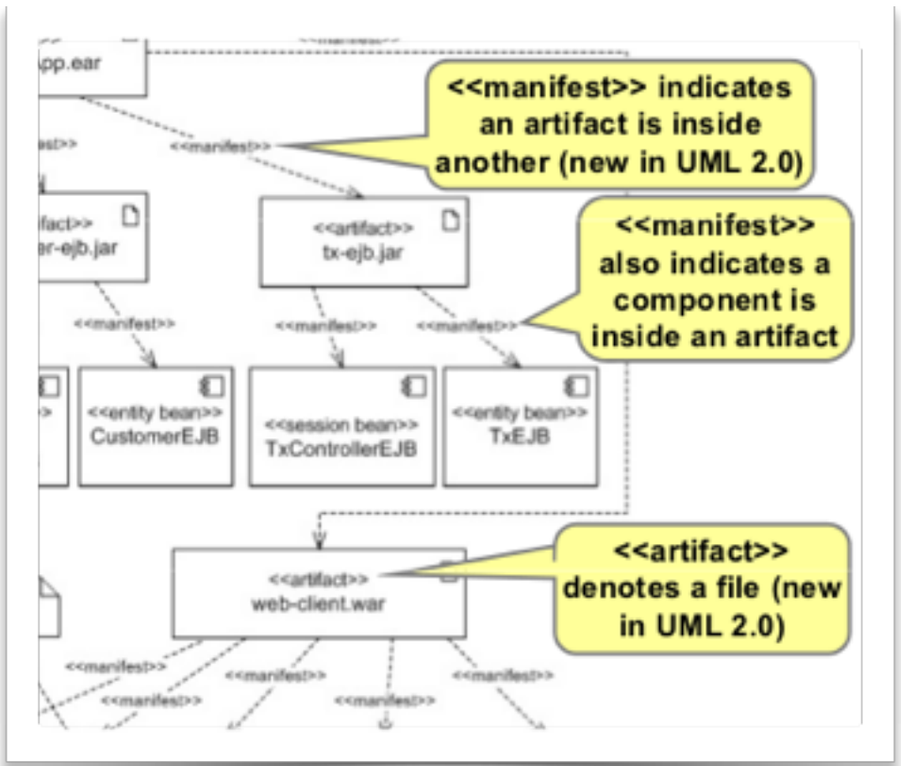
Kinds of views...

- **Module Views :** How system is structured as a set of implementatio units. Things people need to go off and build, or go and buy
    - **deployment** view (how machines and other non-software entities are organised wrt each other)
    - **implementation** view ("is contained in"). View of files and their function.
- **Co** s:
hov et
of e
beh
doe
the
tho

- **Allocation Views:** how does it relate to non software structures in its environment (execution environment, its development environment, etc)

https://sites.google.com/site/softwarearchitectureinpractice/9-documenting-software-architecture/d-allocation-views





So to documentation...

- Figure out how to document a single view.
- Figure out how to document information that applies to more than one view.
- SEI provides a (detailed!) template for this:

  http://www.sei.cmu.edu/downloads/sad/SAD_template_05Feb2006.dot

Documentation for a single view

- **Primary presentation**: Box and Line diagram
    - often this is all you get -- but this is not enough
- **Element catalogue** (name, responsibility, properties)
- **Context diagram** (how it relates to environment)
- **Variability guide** (how variations can be made in the architecture)
- **Rationale** (why we made the decisions we made, and what alternatives we explored that were dead ends)

Documentation Beyond Views

- **Roadmap**: readers visit this first. Discusses the stakeholders, and which views certain stakeholders might want to look at.
- **View Template**: explaining the views themselves, and the way in which those views are documented.
- **System overviews**
- **Mapping between views**: Understanding how the views are related to each other allows real insights into the system. This is usually done pairwise, but not for every pair.
- **Cross-View Rationale**

So, made simple....

- Step 1: Make a table like this:

	View	**View**	**View**	**View**
Stakeholder	*how much info stakeholder needs from this view*			
Stakeholder				
Stakeholder				
Stakeholder				

So, made simple....

- Step 2: Reduce Views to Manageable Size:

	View	**View**	**View**	**View**
Stakeholder	*how much info stakeholder needs from this view*			
Stakeholder				
Stakeholder				
Stakeholder				

Combine where possible
Cut views that are not sufficiently required

So, made simple....

- Step 3: Prioritise view creation:

	View #2		View #1	
Stakeholder	*how much info stakeholder needs from this view*			
Stakeholder				
Stakeholder				
Stakeholder				

Create most needed views first, based on stakeholder requirements (when the view is needed, how much information is needed)

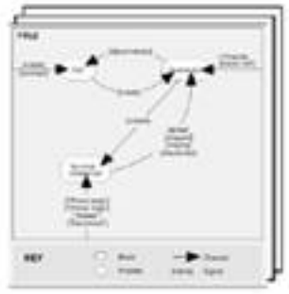
So, made simple....

- Step 4: Document the views:

	View	Vi
Stakeholder	*how much info stakeholder needs from this view*	
Stakeholder		
Stakeholder		
Stakeholder		

Views

Section 1. Primary Presentation of the View



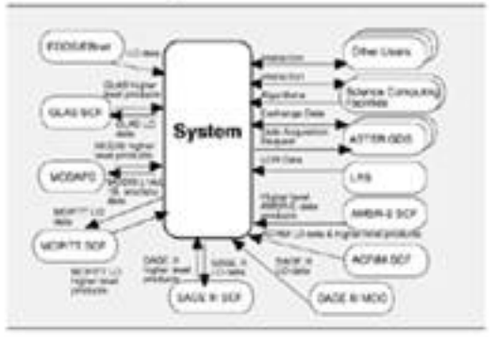
OR

Textual version of the primary presentation

Section 2. Element Catalog

Section 2.A Elements and their properties
Section 2.B Relations and their properties
Section 2.C Element interfaces
Section 2.D Element behavior

Section 3. Context Diagram



Section 4. Variability Guide

Section 5. Architecture Background

Section 5.A Design rationale
Section 5.B Analysis of results
Section 5.C Assumptions

Section 6. Glossary of Terms

Section 7. Other Information

Source: Adapted from [Clements 03].

http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/Part+Two+Creating+an+Architecture/Chapter+9.+Documenting+Software+Architectures/9.4+Documenting+a+View/

So, made simple....

- Step 5: Get holistic:
    - Mapping between views:

	View	**View**
View	*How elements in the views relate*	
View		

- System overview
- Cross-view Rationale (system-wide rationale)

http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/Part+Two+Creating+an+Architecture/Chapter+9.+Documenting+Software+Architectures/9.4+Documenting+a+View/