# ASD(R&E) NCCP/FABRIC

2018-01-11

## Introduction to Software Architecture

AFRL, ARL, CERDEC, SPAWAR

GTRI, IDA, LLNL, MIT-LL, SEI







Carnegie Mellon University
Software Engineering Institute




This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

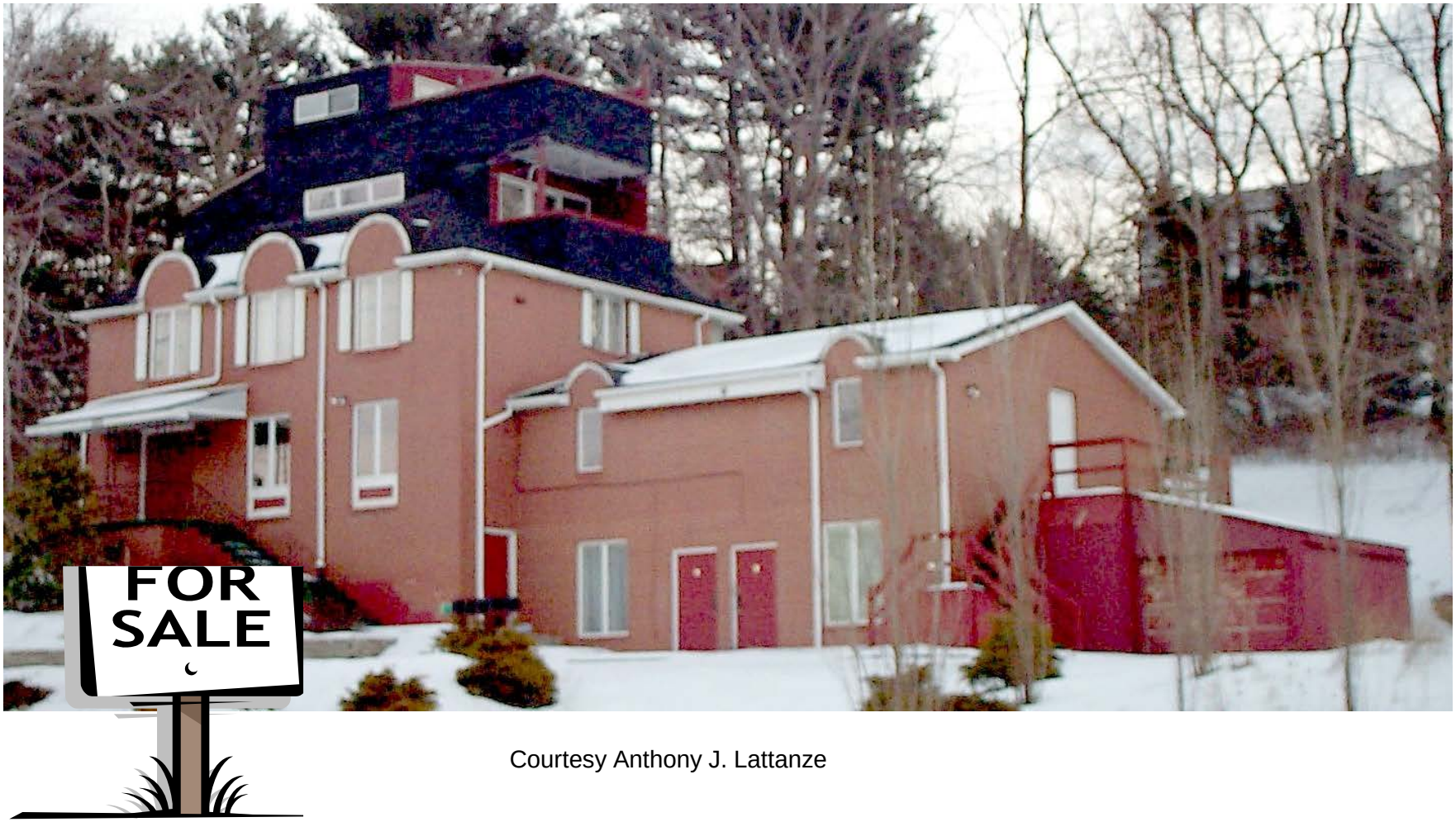


DM18-0007

# Agenda

- Motivation & Goals
- Context
- Quality Attributes
- Views
- Structures
- Styles
- Tactics

# Is this a good architecture?




Courtesy Anthony J. Lattanze

# Why do we do this?

- Ultimately we want to realize a system that meets the requirements of the system.
- The goal of software architecture (a big-picture approach) is to turn requirements into a design that can be implemented.
- To be successful we need to
  - Partition responsibilities into different *elements*
  - Define *relationships* between elements
  - Reason about a system's properties
  - Make reasoned decisions about the architecture
  - Communicate all this to others
  - Answer pragmatic questions about schedule and work allocation

# What do we want out of it?

- A series of diagrams, prose and models that describe
  - The needs of the system as the architects understand them, placed in the context of architecture
  - The key decisions made during the process
    - This includes decisions to NOT do something.
  - The proposed architecture(s)
    - Alternate architectures provide insight into decisions.
  - Design decisions and rationale
    - The best architects also explain why certain “obvious” solutions aren’t chosen.
  - Guidance for implementers
    - This guidance usually comes from prototypes developed.

**Essentially we produce a set of constraints with rationale and guidance.**

# Formalisms

“Writing is nature’s way of letting you know how sloppy your thinking is.” - Richard Guindon

# Formalisms

"Writing is nature's way of letting you know how sloppy your thinking is." - Richard Guindon

"**Math** is nature's way of letting you know how sloppy your **writing** is."- Leslie Lamport

# Formalisms

"Writing is nature's way of letting you know how sloppy your thinking is." - Richard Guindon

"**Math** is nature's way of letting you know how sloppy your **writing** is."- Leslie Lamport

"I didn't have time to write a short letter, so I wrote a long one instead." - Mark Twain

# Formalisms

"Writing is nature's way of letting you know how sloppy your thinking is." - Richard Guindon

"**Math** is nature's way of letting you know how sloppy your **writing** is."- Leslie Lamport

"I didn't have time to write a short letter, so I wrote a long one instead." - Mark Twain

"Software is iterative refinement from ideas to the ultimate formalism of code." - Andrew

# What it does and doesn't do!

- Architecture does NOT guarantee anything.
- It promotes or inhibits the properties that we considered or *failed to consider* during the architecture process.
  - The architecture will be consumed by other architects, designers, developers, or deployers.
  - The implementation may not adhere to the architecture.

# Architecture – Definitions

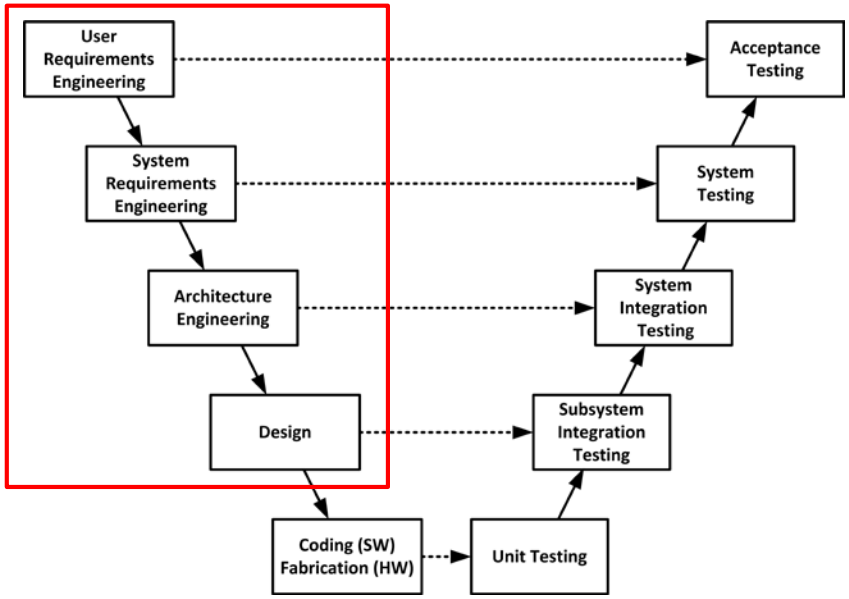
- Extensive list of definitions at http://www.sei.cmu.edu/architecture/definitions.html
- "The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time."
  - Garlan, Perry. "Introduction to the Special Issue on Software Architecture," *IEEE Transactions on Software Engineering*, April 1995.
- "The software architecture of a system is the ***set of structures*** needed to ***reason about the system***, which comprise software ***elements***, ***relations*** among them, and ***properties*** of both."
  - Clements, P., et al. *Documenting Software Architectures: Views and Beyond (2nd edition)*, Addison-Wesley, 2010.

# Practitioner Definitions

- “All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where *significant* is measured by cost of change.”
  - Grady Booch
- “Software architecture is the set of decisions which, if made incorrectly, may cause your project to be cancelled.”
  - Eoin Woods
- “Decomposition of the problem in a way that allows your development organization to efficiently solve it, considering constraints like organizational structure, team locations, individual skills, and existing assets.”
  - John Klein

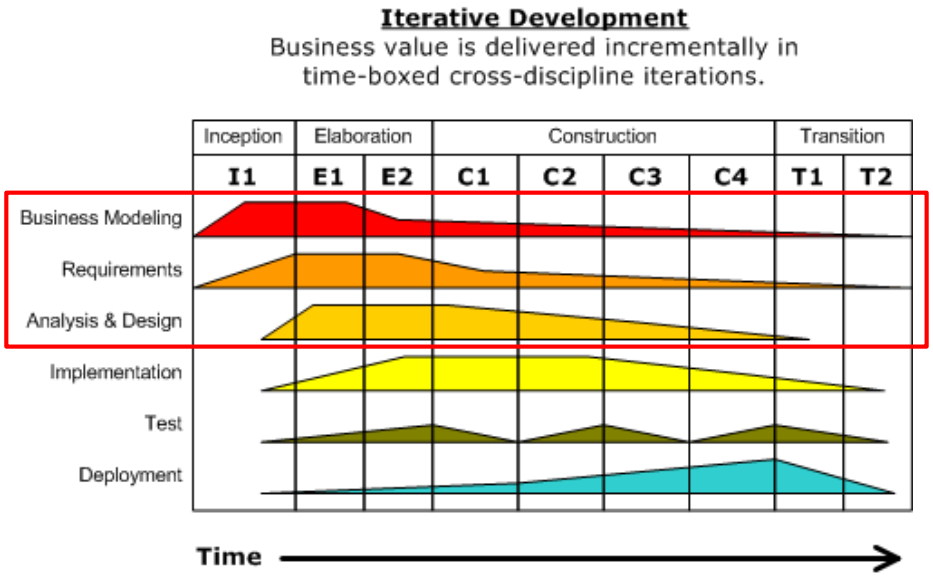
# When do we do architecture?

## V-Model




https://insights.sei.cmu.edu/assets/content/F1 - Traditional V Model.jpg

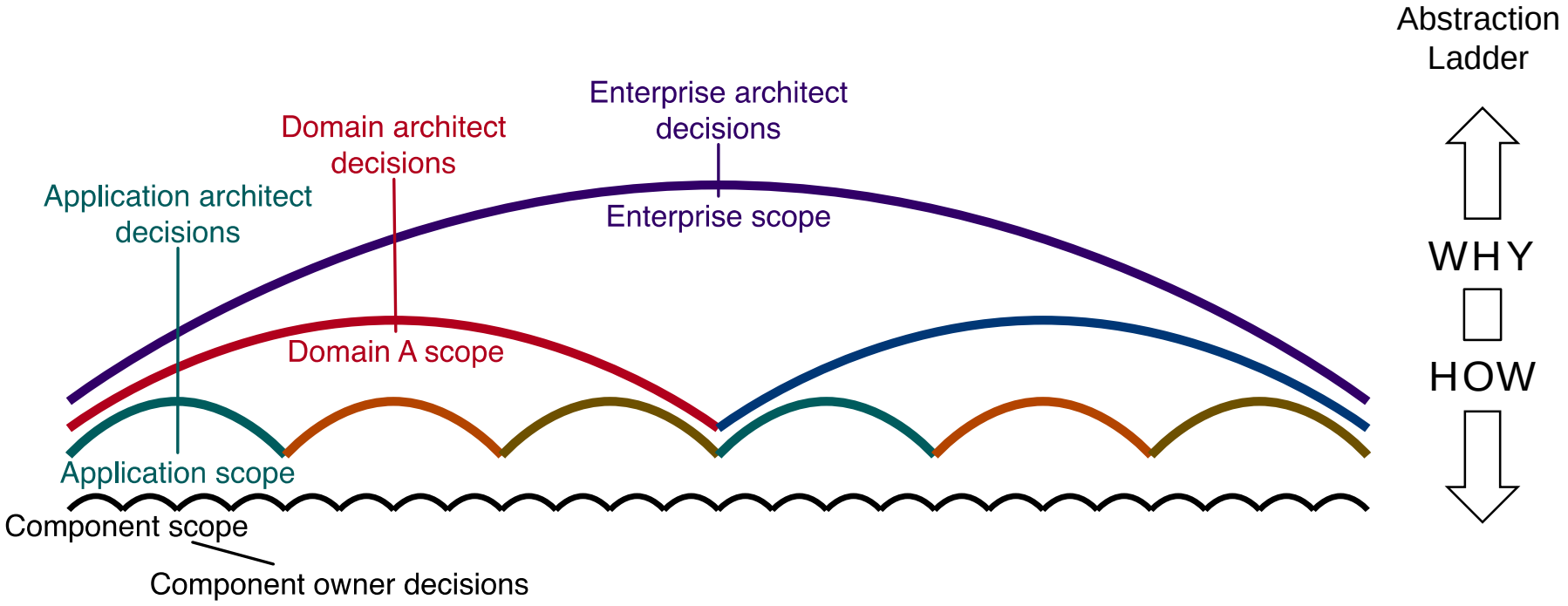
## RUP Model




https://en.wikipedia.org/wiki/Rational_Unified_Process

- Requirements, architecture, modeling and design tasks start early.
- However, they require effort through the entirety of the project.
- We should expect and plan for changes!

# One architect's decisions become another's constraints.




from Malan & Bredemeyer, "Less is More with Minimalist Architecture", *IT Pro*, Sept/Oct 2002, p. 48.

- The balance is between under and over specification.
- The balance varies based on the skills of the consumer.
- Any decision NOT made at one level is made at the lower one.
- Architecture and design hoist the decisions out of the developer's hands.

# Architectural Drivers

- Key functional requirements: Not all functional requirements impact the architecture.
  - Requirements documents
  - Use cases and scenarios
  - Mission thread vignettes
- Business Constraints
  - Examples: targeted markets, compliance, product lines, rollout
- Technical Constraints
  - Examples: legacy code, platforms, languages, protocols
- Quality Attributes
  - Non-functional requirements, “ilities”
  - Most difficult to identify, quantify, and document
  - Usually end up becoming the biggest challenge

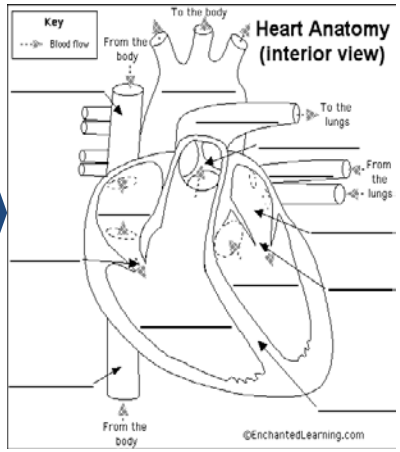
# Quality Attributes

- SEI Core Attributes:
  - Availability
  - Interoperability
  - Modifiability
  - Performance
  - Security
  - Testability
  - Usability
- Other groups (ISO) emphasize different lists.

- Six part scenario for defining quality attribute scenarios
  - Stimulus
  - Source of stimulus
  - Environment
  - Artifact stimulated
  - Response
  - Response measure

# Architectural Structures – 1

A human body comprises multiple *structures*.




a *static view* of one human *structure*

a *dynamic view* of that *structure*

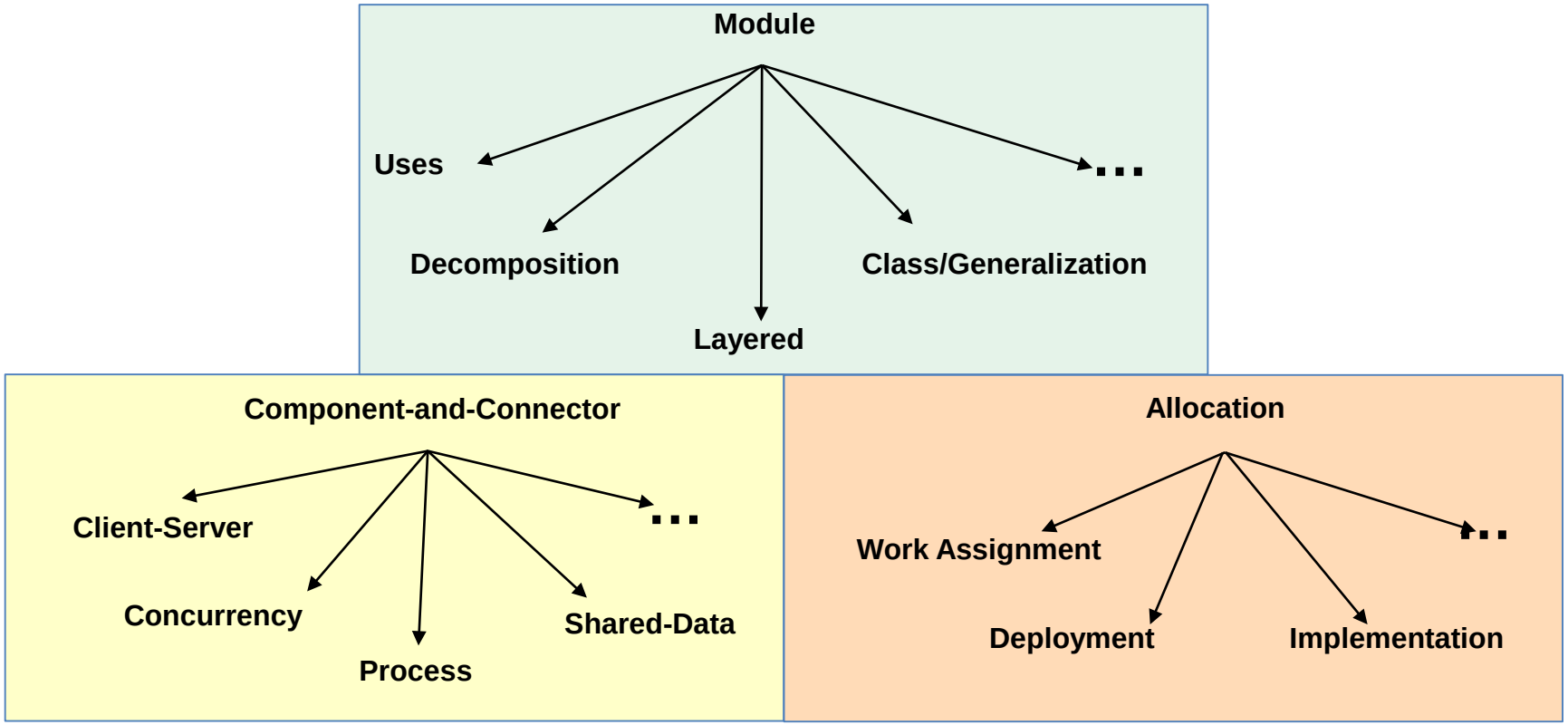
- One body has many structures with different views of each.
- Software is too complex to grasp all at once.
- Different *perspectives* show different types of properties and support different types of analysis.
- Even within one perspective, different diagrams (i.e., views) express difference details.
- Different stakeholders will prefer different views.

# Architectural Structures - 2

Architectural structures for software systems can be divided into three types:

- **Module structures** (Static) – consisting of elements that are units of implementation called modules and the relationships among them
- **Component-and-Connector structures** (Dynamic) – consisting of runtime components (units of computation) and the connectors (communication paths) between them
- **Allocation structures** (Physical) – consisting of software elements and their relationships to elements in external environments in which the software is created and executed

# Architectural Structures Summary




**Architects must focus on whatever structures will provide them with the most leverage in achieving the desired quality attributes of a system.**

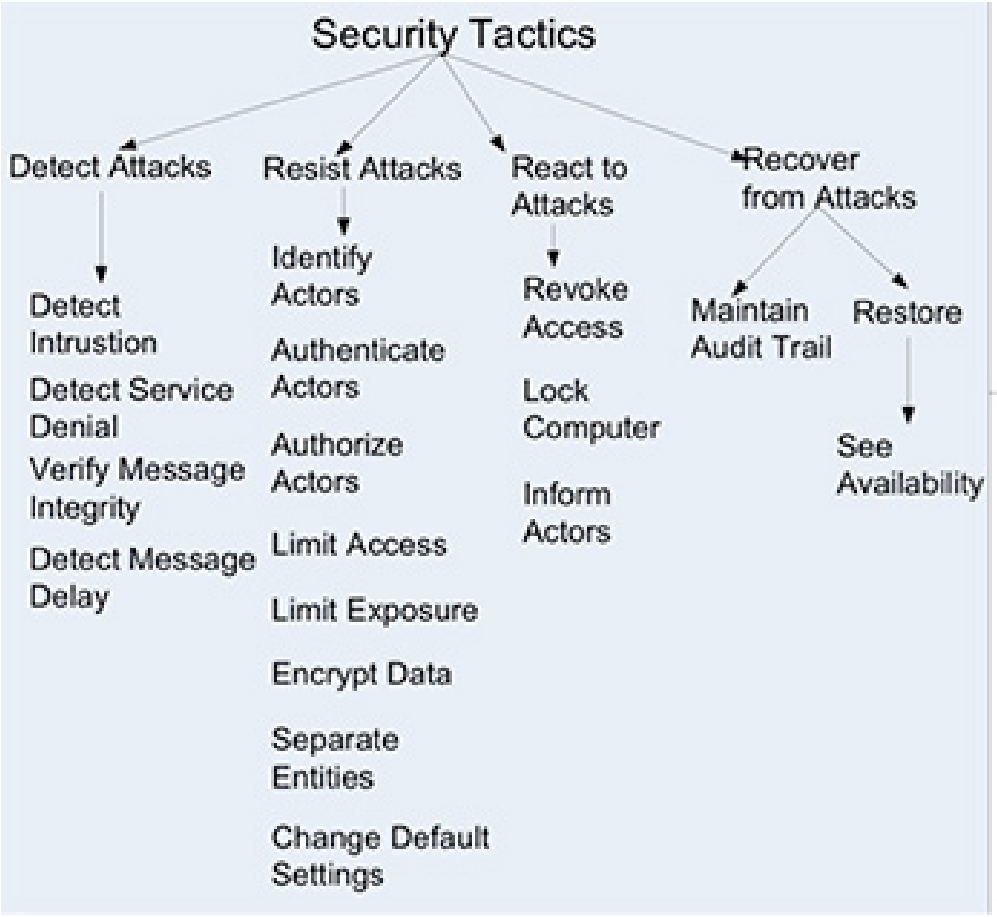
# Architectural Styles

- Architectural Styles and Patterns
  - Common ways of solving problems
  - Possess known gross properties
  - Provide a vocabulary for communication
  - Styles are usually modified and blended
  - Styles are associated with a perspective
- Example Styles:
  - Module Styles
    - Layered, Uses, Generalization, Aspects, Data Model
  - Component and Connector Styles
    - Data Flow: Pipe and Filter, Batch Sequential
    - Call-Return: Client Server, Peer to Peer, Service Oriented
    - Event Based: Pub Sub
    - Repository: Shared-Data Style
  - Allocation Styles
    - Deployment, Install, Work Assignment

# Tactics

Each quality attribute has *tactics* (guidelines) that can be applied to promote that attribute, while potentially inhibiting others.

Tactics can be combined and blended and generally consist of finer details than a style or pattern.



# Examples

- Quality Attribute Workshop (QAW)
  - "…provide a method for identifying a system's architecture-critical quality attributes…"
- Architecture Tradeoff Analysis Method (ATAM)
  - "…is a method for evaluating software architectures relative to quality attribute goals."
- Mission Thread Workshop (MTW)
  - "…is a facilitated process that brings together SoS stakeholders to augment existing mission threads with quality attribute considerations that will shape the SoS architecture and to identify SoS architectural challenges."
- Architecture Analysis and Design Language (AADL)
  - "… is an architecture description language standardized by SAE."
- Attribute Driven Design (ADD)
  - "…s a systematic step-by-step method for designing the software architecture of a software-intensive system."
- Architecture Centric Design Method (ACDM)
  - "…is a scaleable *method* for *designing* the *architecture* of a software intensive system with a product focus that uses the *architecture* to complement organizational processes and implementation activities."
- The Open Group Architecture Framework (ToGAF)
  - "… is a framework for enterprise architecture that provides an approach for designing, planning, implementing, and governing an enterprise information technology architecture." – This is a *process* framework.
- DoD Architecture Framework (DoDAF)
  - "…is an architecture framework for the United States Department of Defense (DoD) that provides visualization infrastructure for specific stakeholders concerns through viewpoints organized by various views." – This is a *documentation* framework.
- And many more!

# Some Resources

- Architectures for Software Systems – Carnegie Mellon University course 17-655 offered as part of Software Engineering Master's degree programs
- *Software Architecture in Practice (3rd Edition)* - Len Bass, Paul Clements, and Rick Kazman. 2012. Addison-Wesley Professional.
- *Documenting Software Architectures: Views and Beyond* (2nd ed.) - David Garlan, Felix Bachmann, James Ivers, Judith Stafford, Len Bass, Paul Clements, and Paulo Merson. 2010. Addison-Wesley Professional.
- *Architecting Software Intensive Systems: A Practitioners Guide* (1st ed.) - Anthony J. Lattanze. 2008. Auerbach Publications, Boston, MA, USA.
- And many more!