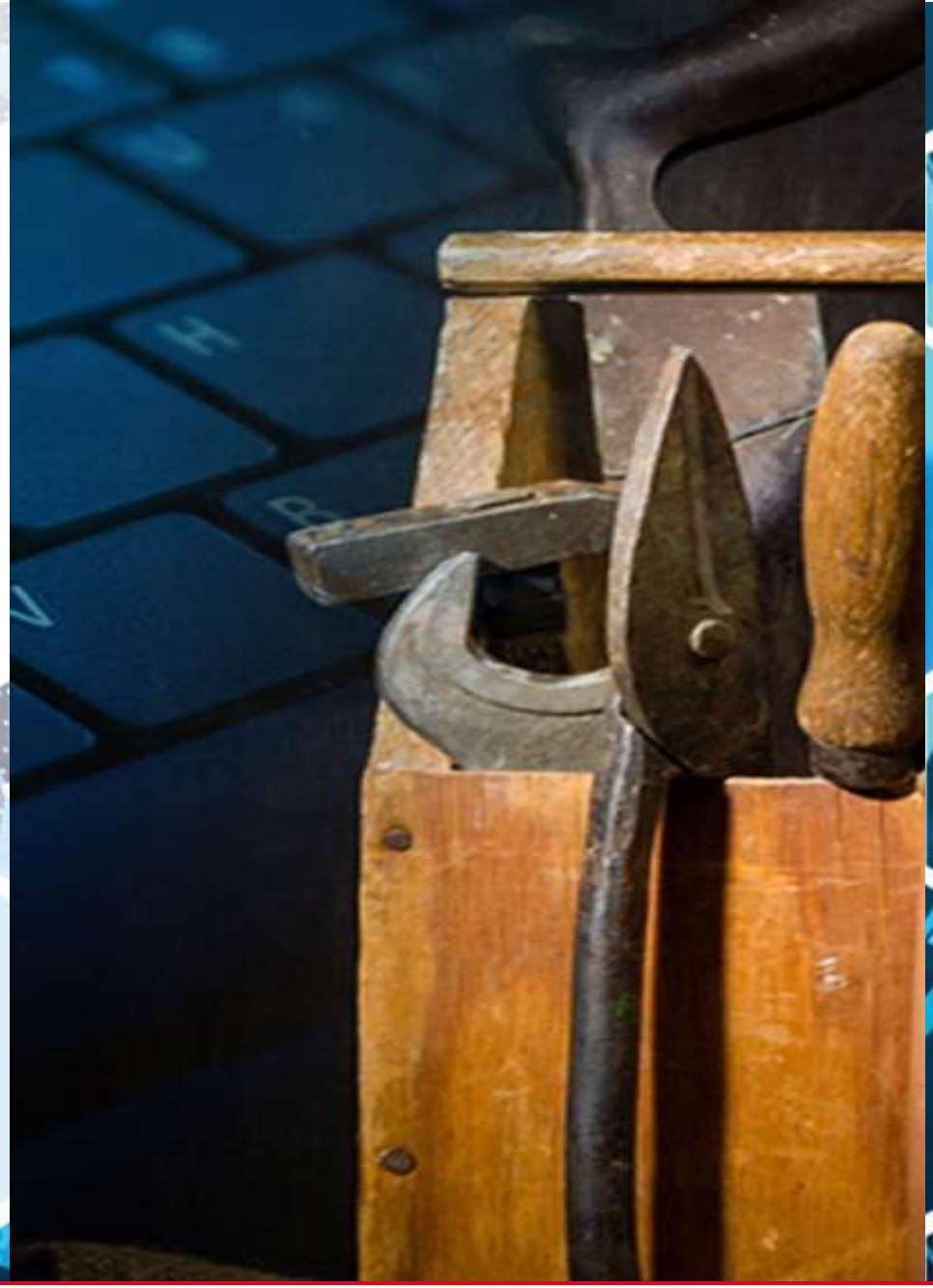
SEI
Training
Course

Excerpt: Modeling System Architectures Using the Architecture Analysis and Design Language (AADL)

Overview of Model-Base Software Engineering and Architecture-Centric Virtual Integration

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



This is the “Modeling Guideline” module (Module 10) from the **SEI Modeling System Architectures Using the Architecture Analysis and Design Language (AADL)** eLearning course.

This module consists of two parts:

- Part 1: Overview of Model-base Software Systems Engineering (~ 9 mins)
  - Key concepts of software architecture modeling and Architecture-centric Virtual Integration Practice (ACVIP)
- Part 2: Phases of a structured architecture centric approach to embedded systems modeling, integration and analysis (~25 mins)
  - Concepts of model focus & scope, model building, component integration, analysis

Sources of additional information:

- The eLearning course: https://www.sei.cmu.edu/education-outreach/courses/course.cfm?courseCode=V40
- About AADL & Open Source AADL Tool Environment (OSATE): aadl.info -> Learn About AADL and OSATE
- ACVIP overview: aadl.info -> Learn about ACVIP


David Gluch: According to Miles Davis, who was
a jazz trumpeter, band leader and composer.
00:08
08:54
CC
HD
1x

Outline

Model-Based Software Systems Engineering

Architecture-Centric Virtual Integration Practices

AADL Architecture Abstractions

A Structured Architecture-Centric Approach

Summary

Model-Based Software Systems Engineering

Model-based software systems engineering aligns software systems development with practices in other engineering disciplines.

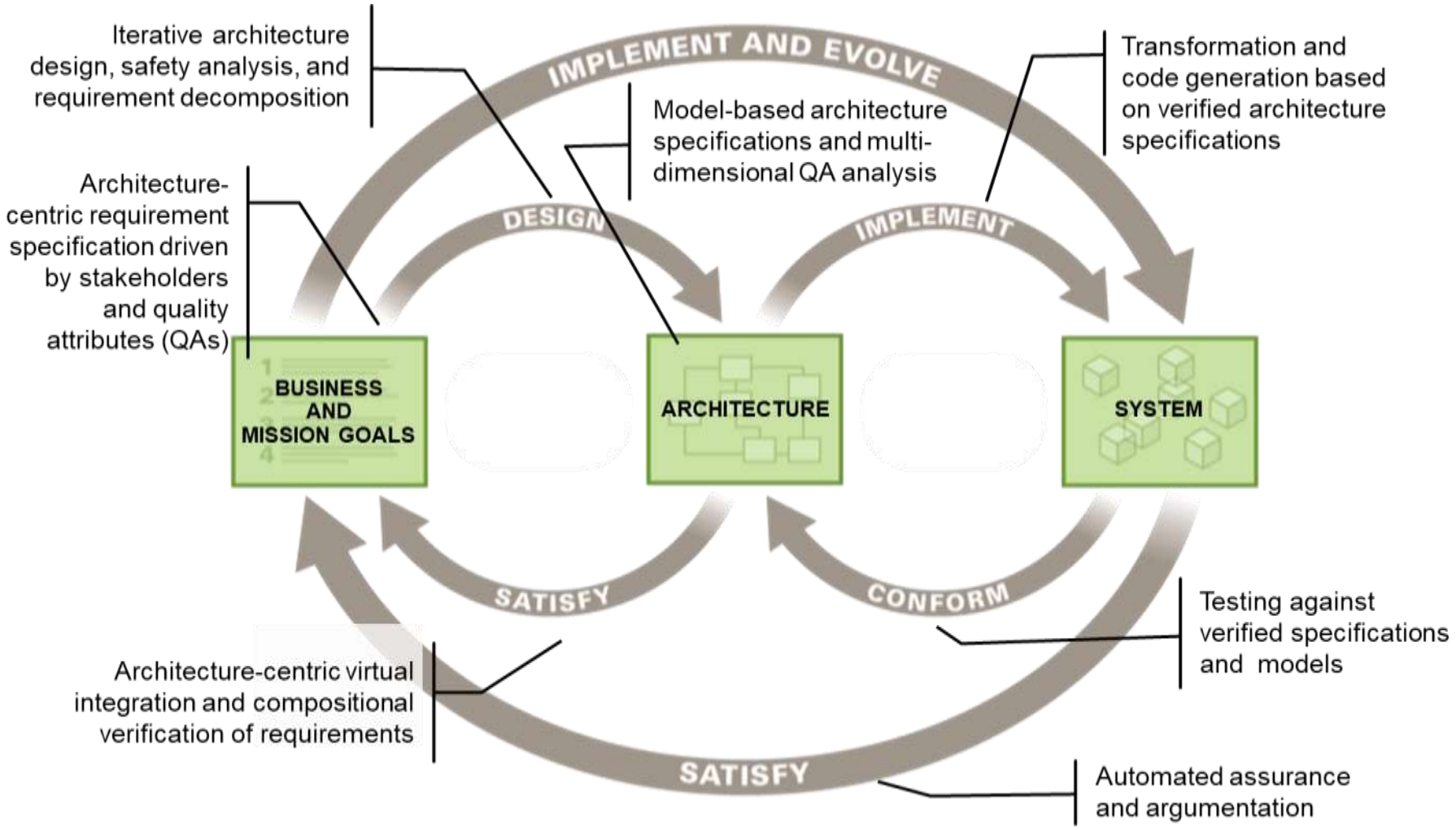
In engineering disciplines, engineers

- iteratively model and analyze before building
- develop models to answer specific concerns
- develop models to effectively explore design alternatives
- use models as part of testing and verification
- use models to specify the design throughout the lifecycle

A comprehensive architectural model within a virtual system integration practice that is supported by design and analysis tools enables

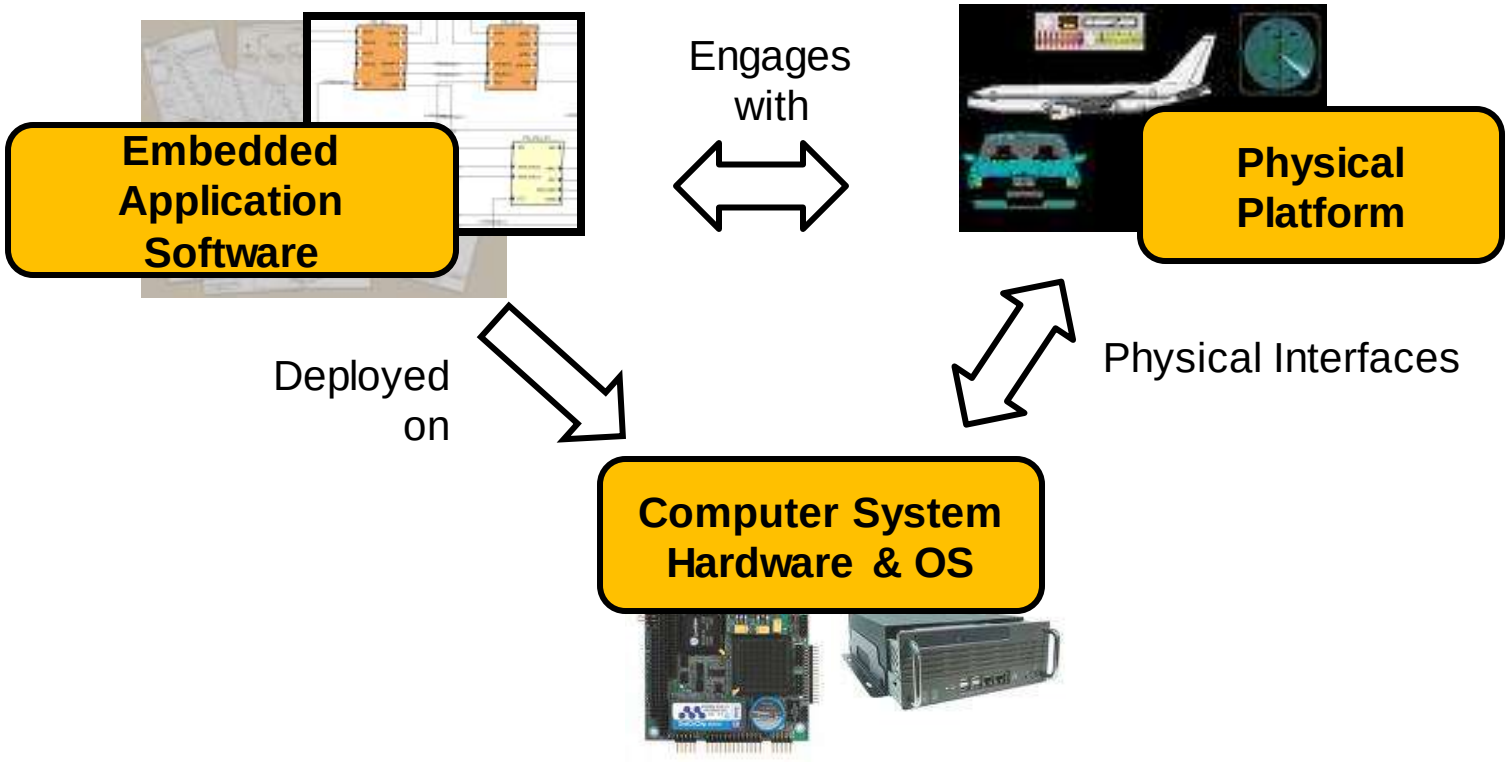
- virtual integration of a system and analysis of critical system aspects (e.g., timing, utilization)
- early detection of errors
- assurance throughout the life cycle (assurance cases)
- assessment of broad system considerations (e.g., safety, reliability)

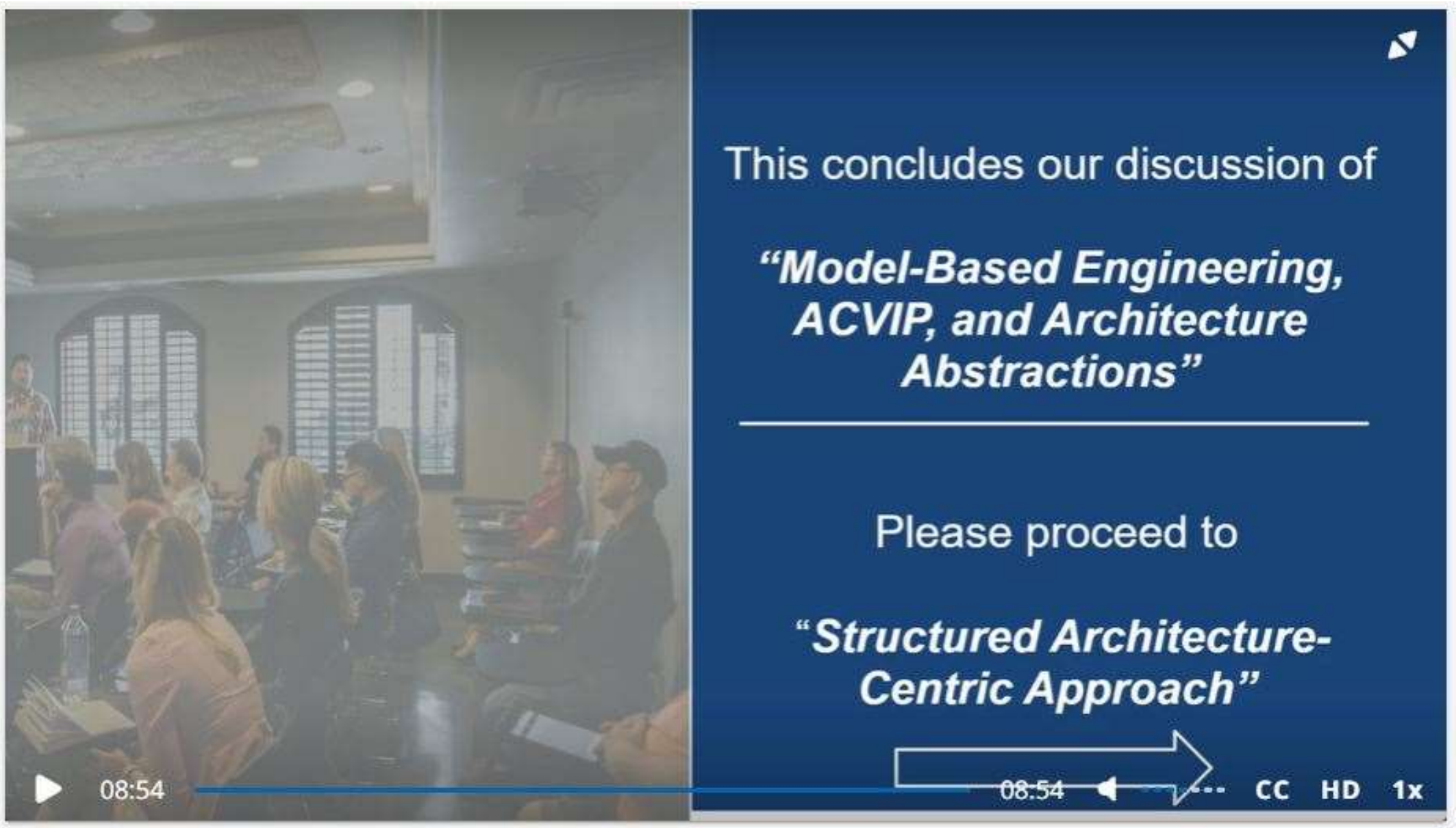
Architecture-Centric Virtual Integration Practices (ACVIP)



AADL for Embedded System Architectures

Based on architectural abstractions, AADL focuses on the interactions between the three major elements of an embedded software-dependent system




This concludes our discussion of
"Model-Based Engineering, ACVIP, and Architecture Abstractions"
Please proceed to
"Structured Architecture-Centric Approach"
08:54
08:54
CC
HD
1x

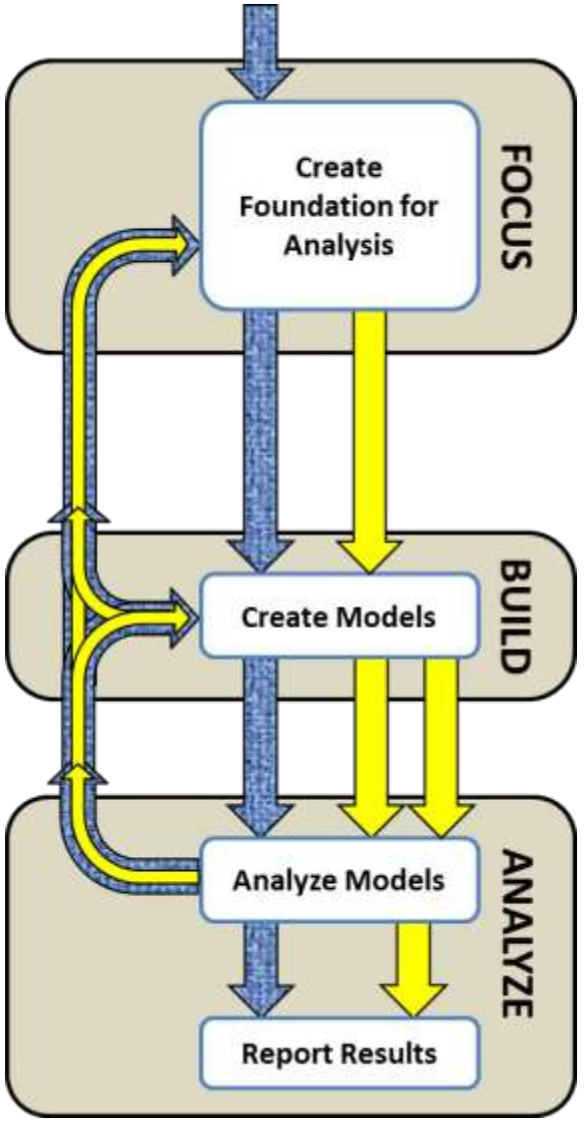
A Structured Architecture-Centric Approach

A representative framework for developing and analyzing a software-dependent system architecture

An architecture-centric, model-based process partitioned into three phases:

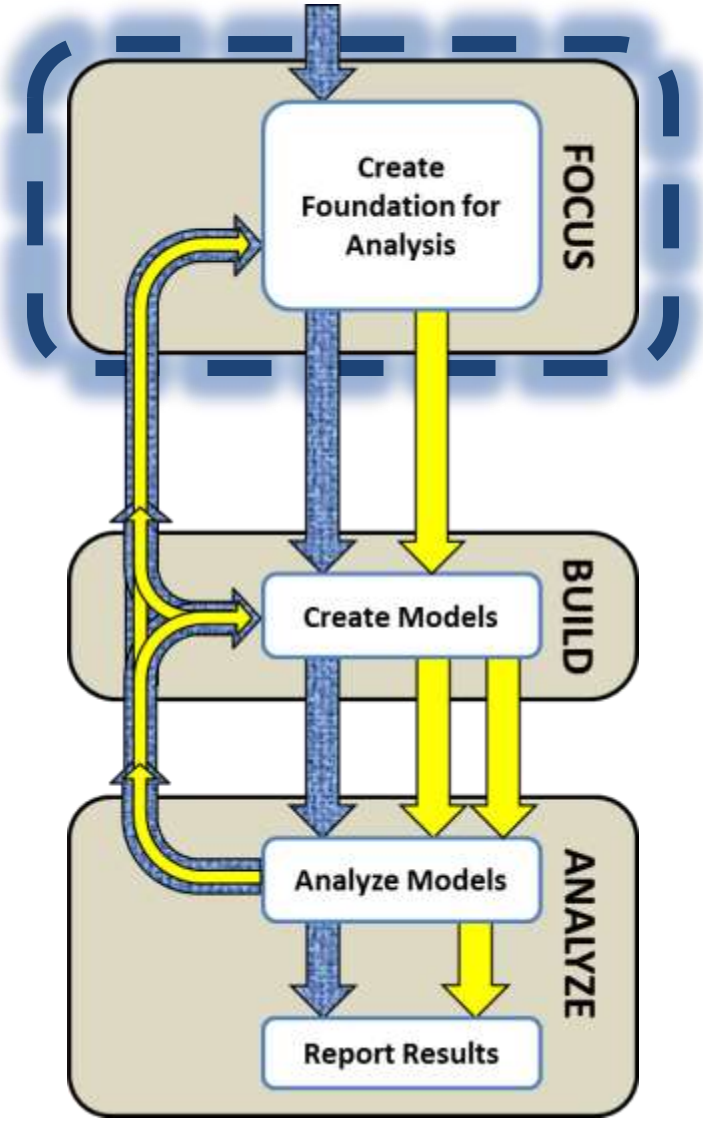
- Focus the effort
- Develop architecture models
- Analyze systems and models

Uses the AADL as the principal specification and modeling language



The Focus Phase

1. Identify key mission drivers
   - Identify the drivers
   - Agree on top driver(s)
2. Identify key elements and aspects of architecture in terms of
   - Mission functionality
   - Computer platform
   - Mission platform
3. Define key quality attributes into a utility tree
   - Define quality attributes
   - Identify operational quality attributes
   - Prioritize in terms of importance and difficulty
4. Develop a prioritized set of scenarios
   - Include use case, growth, and exploratory
   - Focus on key quality attributes



Identify Business/Mission Drivers

Describe the system's business/mission drivers, including the

- business context for the system
- high-level functional requirements
- high-level quality attribute requirements
  - architectural drivers: quality attributes that "shape" the architecture
  - critical requirements: quality attributes that are most central to the system's success

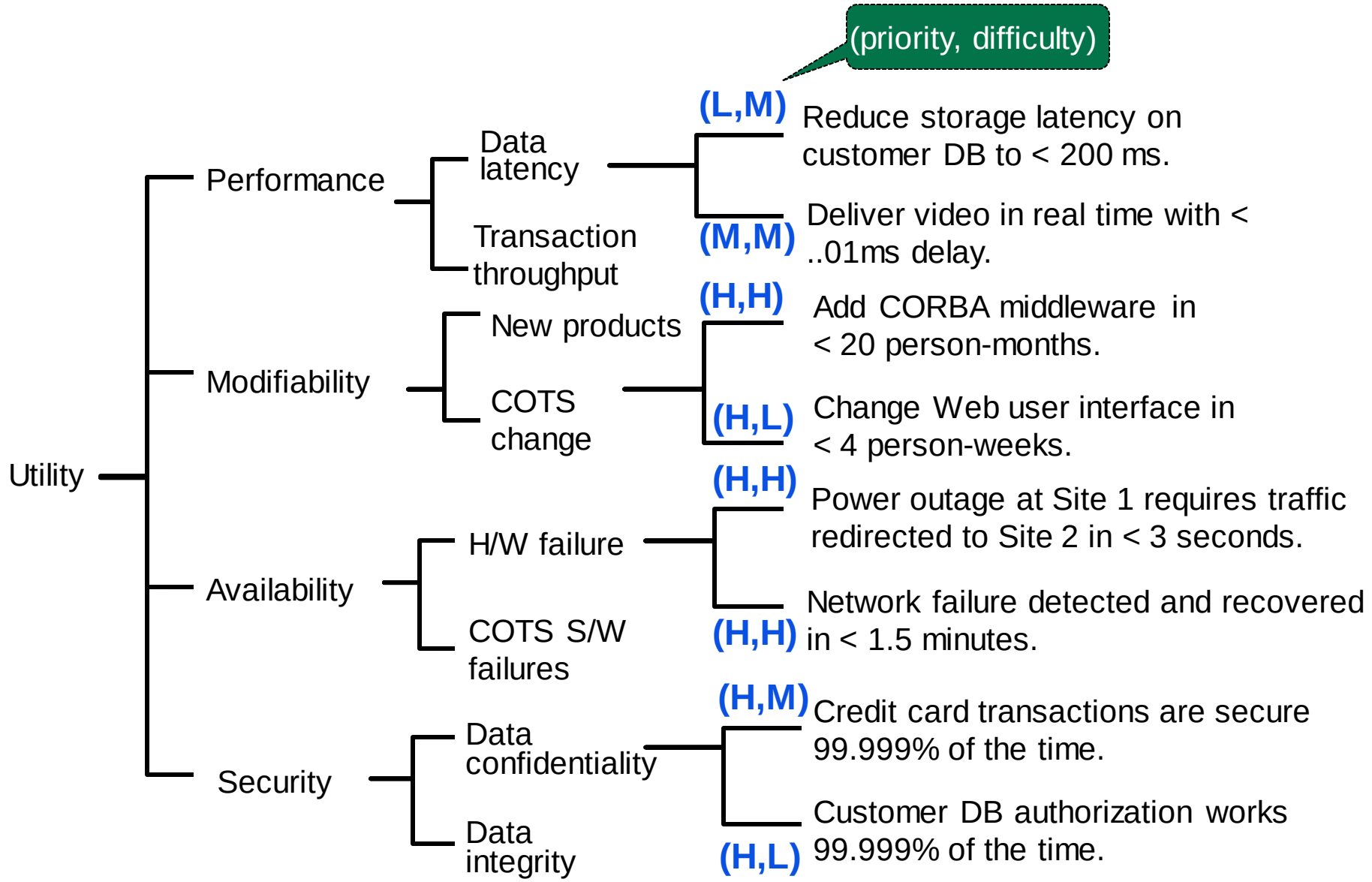
Generate Quality Attribute Utility Tree

Identify, prioritize, and refine the most important quality attribute goals by building a *utility* tree.

- A utility tree is a top-down vehicle for characterizing and prioritizing the "driving" attribute-specific requirements.
- The driving quality attributes are the high-level nodes (typically performance, modifiability, security, and availability).
- Scenarios are the leaves of the utility tree.

Output: a characterization and a prioritization of specific quality attribute requirements along with supporting scenarios for assessing them

Utility Tree Construction



Scenarios

Scenarios are used to

- represent *stakeholders'* interests
- understand quality attribute requirements

Scenarios should cover a range of information:

- use case scenarios: anticipated uses of the system
- growth scenarios: anticipated changes to the system
- exploratory scenarios: unanticipated stresses to the system

Scenarios should be specific.

Example Scenarios

Use case scenario

Pilot presses update button requesting an updated situational display including updated sensor values and that updated information is displayed in less than 50 milliseconds.

Growth scenario

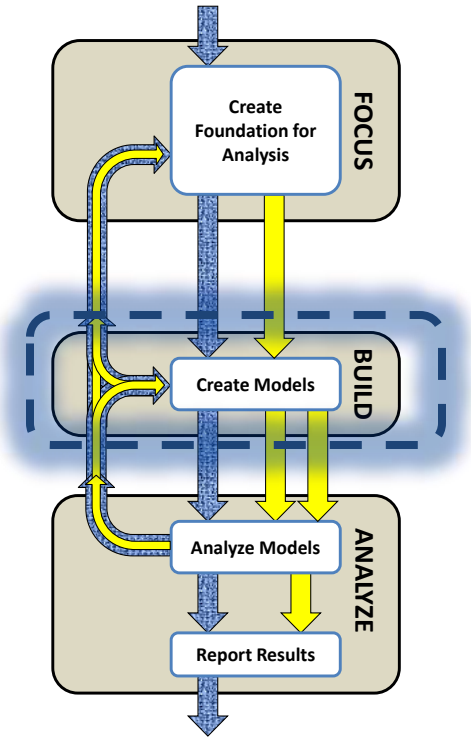
Add a new data server to the ground tracking system to reduce data latency in Scenario 1 to 2.5 seconds, within 1 person-week.

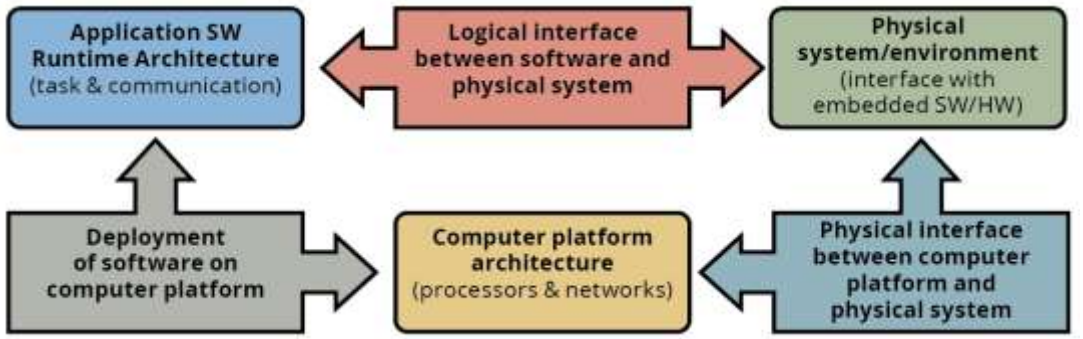
Exploratory scenario

Three out of the eight system temperature sensors in the sensor array fail during normal operation without affecting overall system availability.

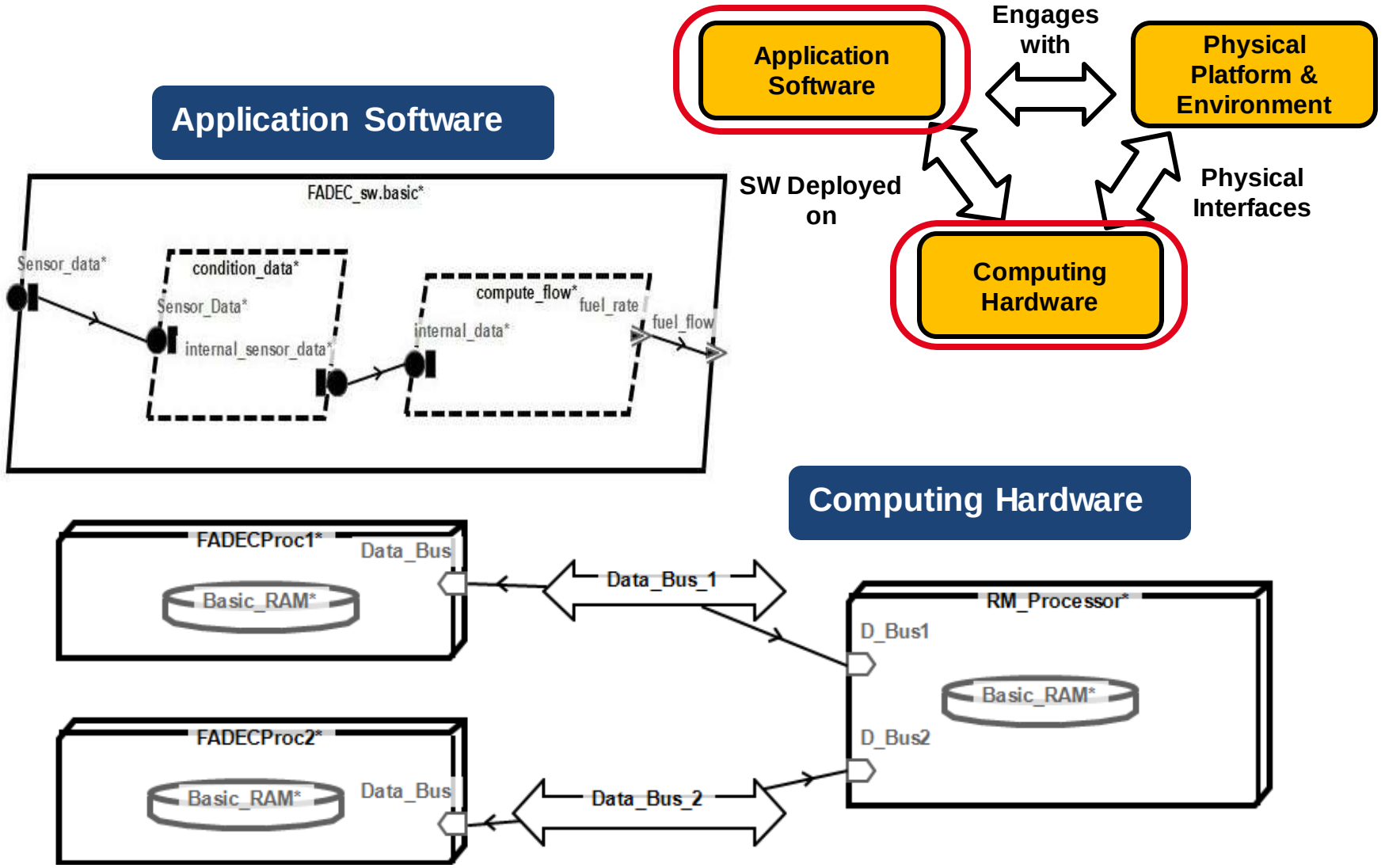
The Build Phase (Modeling) with the AADL

1. Identify relevant architecture patterns
   - Rely on architecture views
   - Use application patterns and task/communication patterns
2. Create library of building blocks
   - Identify components at the relevant level of abstraction
   - Organize into application software, computer hardware, and mission platform components
3. Create model of system
   - Create sub-architectures and combine
4. Annotate model with relevant data for use scenarios
   - Use pre-declared AADL properties
   - Add properties as needed to record important characteristics and assumptions
5. Reflect scenarios in model
   - Use end-to-end flows (use cases, growth)
   - Exercise variants of the model (exploratory, growth)

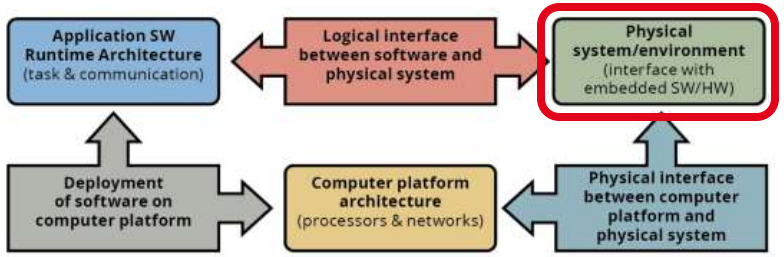




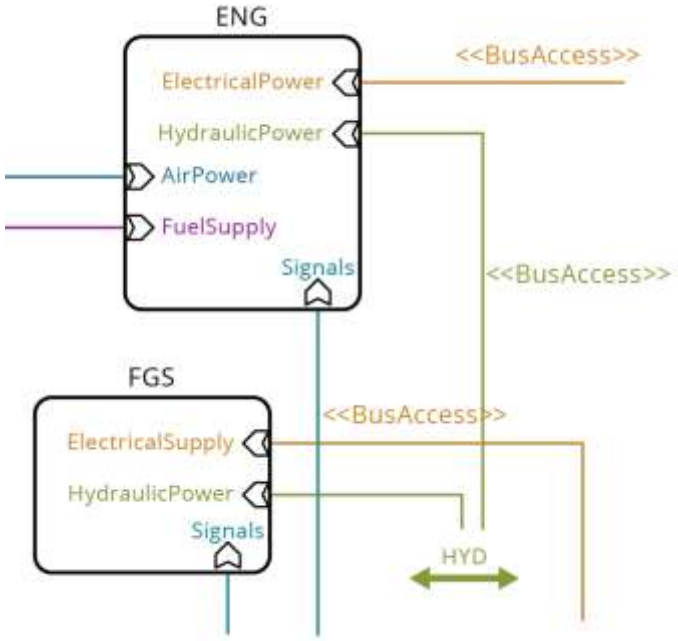
Separation of Concerns: Platform and Application



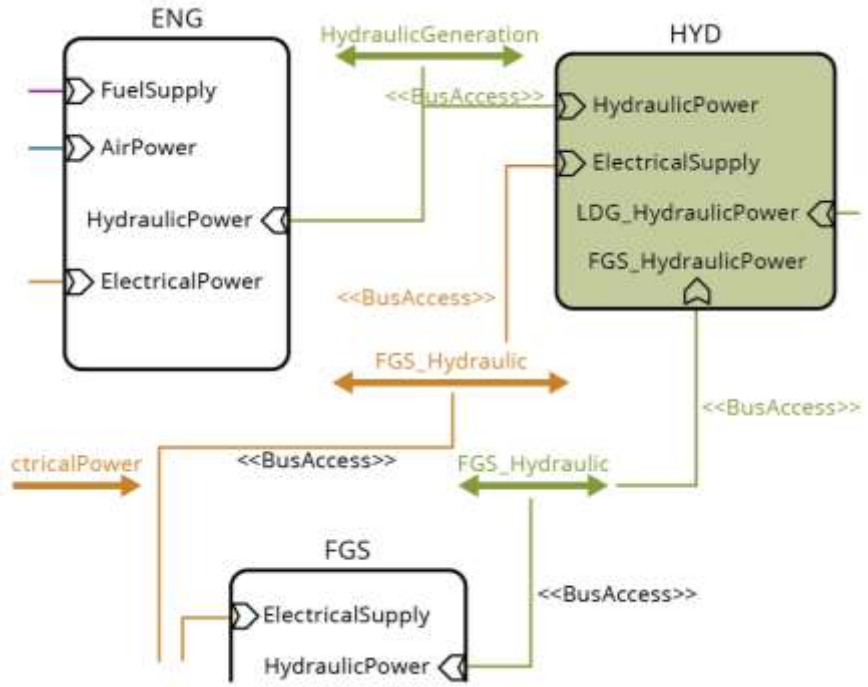
Representing Physical System Resources



Bus to represent resource such as hydraulics

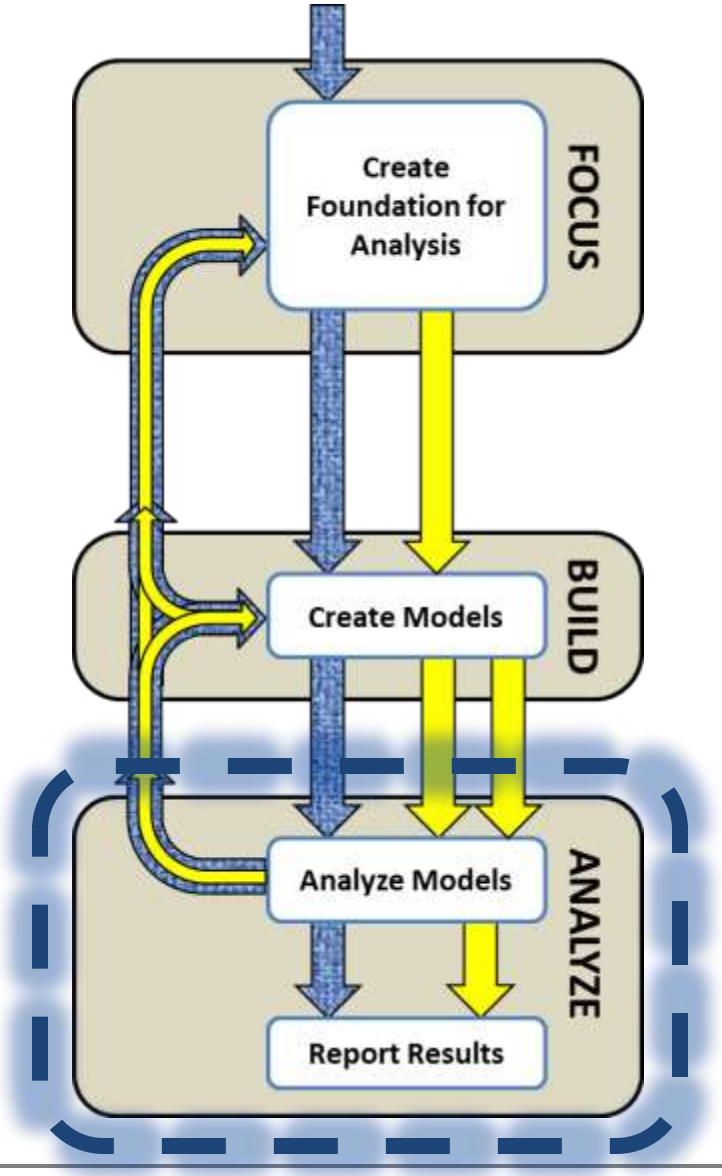


Bus to represent resource connections such as hydraulic lines



The Analysis Phase

1. Compile issues identified during model creation
   - Decisions made when recording system elements as AADL model elements (e.g., subprogram executed every 20 ms as logical thread)
2. Record inconsistencies and their resolution
   - Consistency checkers identify potential issues
   - Record of resolution may provide insight to some root cause issues
3. Perform analysis and record results
   - Run appropriate analysis as needed for each scenario
   - Record and interpret result
4. Conduct what-if analysis to explore alternatives
   - Utilize exploratory scenarios to investigate stresses
   - Virtually investigate options to resolve issues



Summary

Software engineering is an engineering discipline

Architecture models are central to the engineering practices

This approach employs virtual integration and system analysis

AADL is a standard that supports engineering practices for architecture-centric software systems.

This structured approach to evaluating architectures

- is architecture centric and model based
- has three phases: focus, build, analyze
- establishes and prioritizes the most important quality attributes
- helps validate architecture and refine requirements through scenarios