Advanced Software Architecture

Lecture #1 - Introduction

Professor Jörgen Hansson
Department of Computer Science and Engineering
Chalmers University of Technology
jorgen.hansson@chalmers.se

Brief Intro of Me

Work Experience

2010 - Professor at Chalmers in the area of software engineering

2005-2010 Senior Member of Technical Staff
Software Engineering Institute, Carnegie Mellon Univ., USA

– Focus on architectural descriptions and validation using the industry standard AADL, which was created by the SEI.

2000-2007 Professor at Linköping University in the area of real-time systems

– Focus on management of real-time data in embedded real-time systems, including QoS/QoD, real-time component models, database systems

Brief Intro of Me

Domain experience:

- Governmental experience: Dept of Defense (DoD), Dept of Energy (DoE), Dept of Interior (DoI), Veterans Administration (VA), National Security Agency (NSA), U.S. Nuclear Regulatory Commission (NRC)

Industrial experience:

- Aviation: AVSI, including Boeing, Airbus, BAE, Rockwell-Collins
- Automotive: Toyota Research (Tokyo), Volvo CE, Saab Automobile, Fiat/GM powertrain, Mecel

Other:

- Worked as a consultant and advisor in the areas of embedded real-time systems, networked systems, and software engineering
- Started a spin-off company in the area of real-time data management

Outline

- What is an Architecture
- What is the rationale and purpose of architecting
- I.e., what are the problems architecting aims to address
- Designing and Architecting next generation aircraft
- Architectural Assessment
- The SAE AADL architecture description language – An overview

CNN

Full List | High Pay | Job Growth | Quality of Life | Sectors

1. Software Architect

Recommend 1K 1 of 100 Next

Top 100 rank: 1
Sector: Information Technology

What they do: Like architects who design buildings, they create the blueprints for software engineers to follow -- and pitch in with programming too. Plus, architects are often called on to work with customers and product managers, and they serve as a link between a company's tech and business staffs.

What's to like: The job is creatively challenging, and engineers with good people skills are liberated from their screens. Salaries are generally higher than for programmers, and a typical day has more variety.

"Some days I'll focus on product strategy, and other days I'll be coding down in the guts of the system," says David Chaiken, 46, of Yahoo in Sunnyvale, Calif., whose current projects include helping the web giant customize content for its 600 million users. Even though programming jobs

PHOTO: DAVID LAURIDSEN

Chaiken, a software engineer for more than two decades, relishes the more

What are the problems?

And what do they have to do with architecture?

Late Discovery of System Problems

- Mismatched assumptions
  - Units, range, delta, base value (Ariane)
- False promises of time partitioning
  - DMA impact across partitions (JSF)
- Unmanaged resource sharing
  - Overload of device bus (Daimler)
- Unexpected Latency variation
  - Unexpected latency jitter (F16)
- Trusting scheduling analysis
  - Detection of priority inversion (Mars Rover)

System Level Fault Root Causes

Violation of data stream assumptions
- Stream miss rates, mismatched data representation, latency jitter & age

Partitions as Isolation Regions
- Space, time, and bandwidth partitioning
- Isolation not guaranteed due to undocumented resource sharing
- Fault containment, security levels, safety levels, distribution

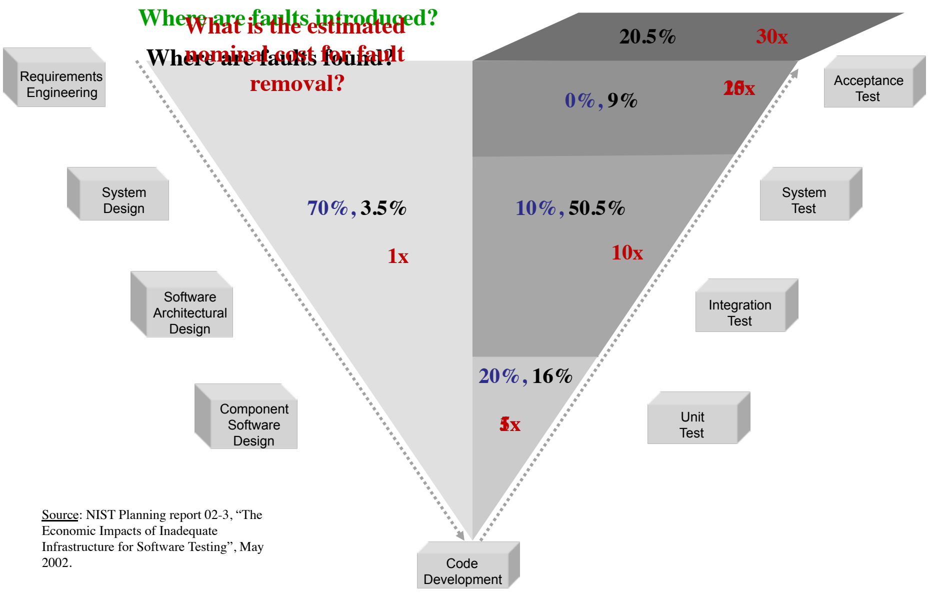
Virtualization of time & resources
- Logical vs. physical redundancy
- Time stamping of data & asynchronous systems

•Inconsistent System States & Interactions
- Modal systems with modal components
- Concurrency & redundancy management
- Application level interaction protocols

Observations and Facts

- Systems outlive their anticipated life expectancy
- Costly faults due to mismatch of assumptions between components and systems
- Tiny proportion of failures due to bugs *
- Largest proportion due to eliciting, recording, and analysis of requirements*
- Scientific evaluation of software failures hard due to lack of reliable data*
- Certification regimes and standards reliance on testing, not enough for high dependability*
- Result:
  - system integration – high risk; evolvability – very expensive
  - life cycle support – very expensive; leads to rapidly outdated components

* Software for Dependable Systems: Sufficient Evidence? By Daniel Jackson et al.

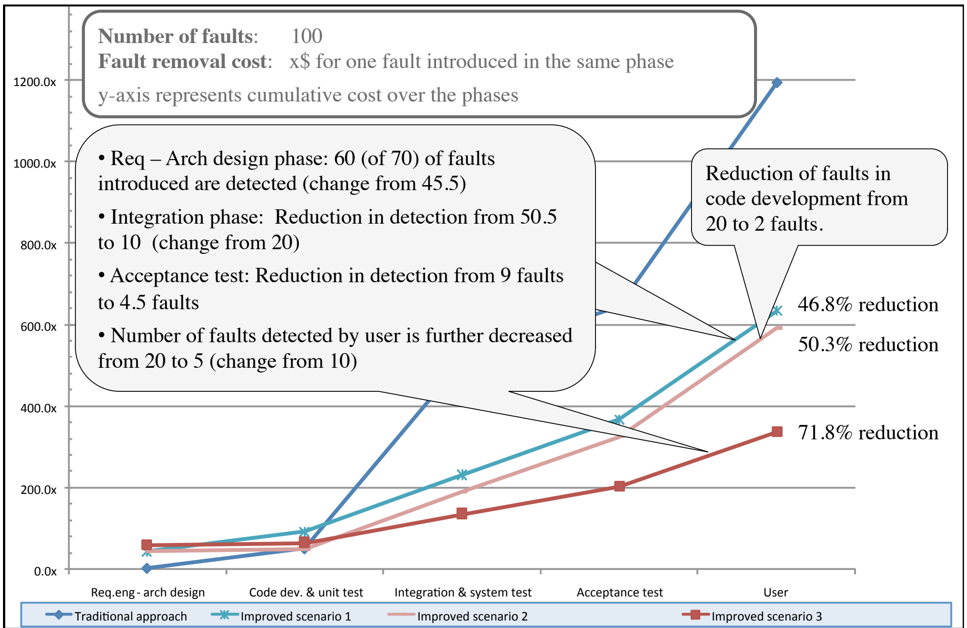


Source: NIST Planning report 02-3, "The Economic Impacts of Inadequate Infrastructure for Software Testing", May 2002.

Defect Economics

Phase	Defects originating in phase (%)	Relative defect removal cost of each phase of origin				
		Req's	Design	Unit test	Integration	Documentation
Requirements	15%	1				
Design	35%	2.5	1			
Unit coding	30%	6.5	2.5	1		
Integration	10%	16	6.4	2.5	1	
Documentation	10%					1
System/Acceptance test	-	40	16	6.2	2.5	2.5
Operation	N/A	110	44	17	6.9	6.8

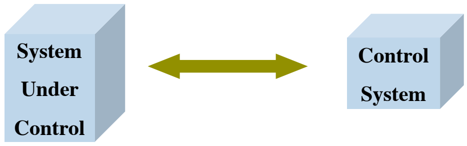
Source: D. Galin, "Software Quality Assurance: From Theory to Implementation", Pearson/Addison-Wesley (2004) & B.W. Boehm, "Software Engineering Economics", Prentice Hall (1981)



Traditional Embedded System Engineering

System Engineer

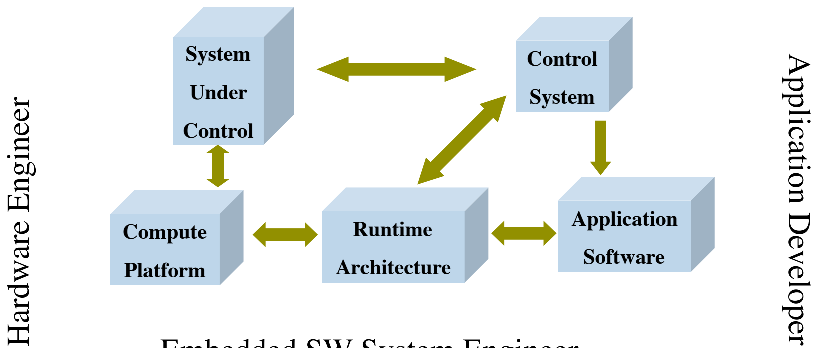
Control Engineer



Software-Intensive Embedded Systems

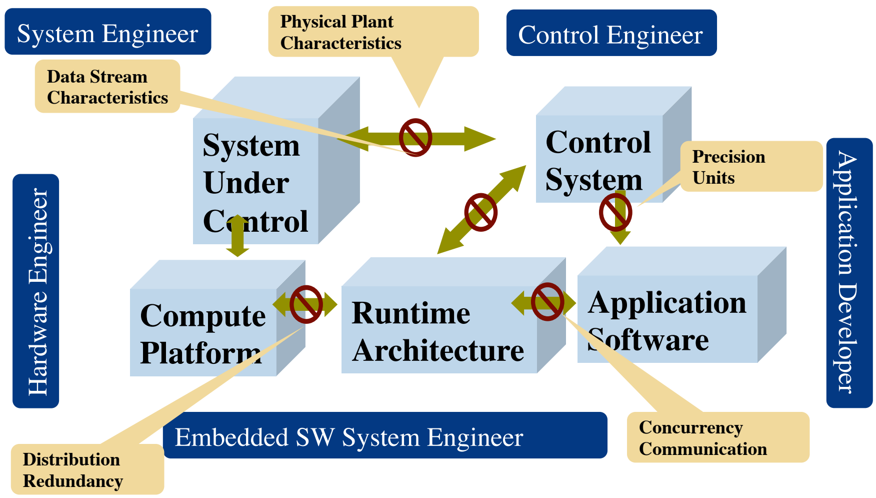
System Engineer

Control Engineer



Embedded SW System Engineer

Mismatched Assumptions



About Time to Discuss What an Architecture is…

Group Discussion

Scenario: "Play it everywhere".

You work at the company Macrohard and you are tasked to develop a networked computer game. The theme for the game is heavily inspired by the latest movie Spider-man 2.0 which was just released. In order to maximize outreach, it is important that the computer game can run on several platforms (personal computers and some mobile phones).

Your task is to architecture the system.

Q1: Who do you believe your stakeholders are?

Q2: What do you imagine their expectations of the system are, or what the system requirements are?

"Modern" Definitions

ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems

- Architecture is defined by the recommended practice as *the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.*
- This definition is intended to encompass a variety of uses of the term architecture by recognizing their underlying common elements. Principal among these is the need to understand and control those elements of system design that capture the **system's utility**, **cost**, and **risk**. In some cases, these elements are the physical components of the system and their relationships. In other cases, these elements are not physical, but instead, logical components. In still other cases, these elements are enduring principles or patterns that create enduring organizational structures.

"Modern" Definition of a Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which **comprise software components**, the **externally visible properties** of those components, and the **relationships** between them.

- "Externally visible" properties: refers to those assumptions other elements can make of an element, e.g., such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

The term also refers to documentation of a system's software architecture. Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects.

Bass, Len; Paul Clements, Rick Kazman (2003). *Software Architecture In Practice, Second Edition*. Boston: Addison-Wesley. pp. 21–24. ISBN 0-321-15495-9.

Implications of Previous Definition

Implication #1: *Architecture defines elements*.

- The architecture embodies information about how the elements relate to each other. This means that architecture specifically *omits* certain information about elements that does not pertain to their interaction.
- Thus, an architecture is foremost an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements.

- In nearly all modern systems, elements interact with each other by means of interfaces that partition details about an element into public and private parts. Architecture is concerned with the public side of this division; private details of elements—details having to do solely with internal implementation—are not architectural

Implications of Previous Definition

Implication #2: *Systems can and do comprise more than one structure* .

• No single structure holds the irrefutable claim to being *the* architecture.

Comment: For example, all non-trivial projects are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Implications of Previous Definition

Implication #3: *Every software system has an architecture* because every system can be shown to be composed of elements and relations among them.

Comment: In the most trivial case, a system is itself a single element (monolith)—an uninteresting and probably non-useful architecture, but an architecture nevertheless. Even though every system has an architecture, it does not necessarily follow that the architecture is known to anyone. Unfortunately, an architecture can exist independently of its description or specification, which raises the importance of *architecture documentation* and *architecture reconstruction*.

Implications of Previous Definition

Implication #4: *The behavior of each element is part of the architecture*

• Behavior (fully or partially) can be observed or discerned from the point of view of another element.

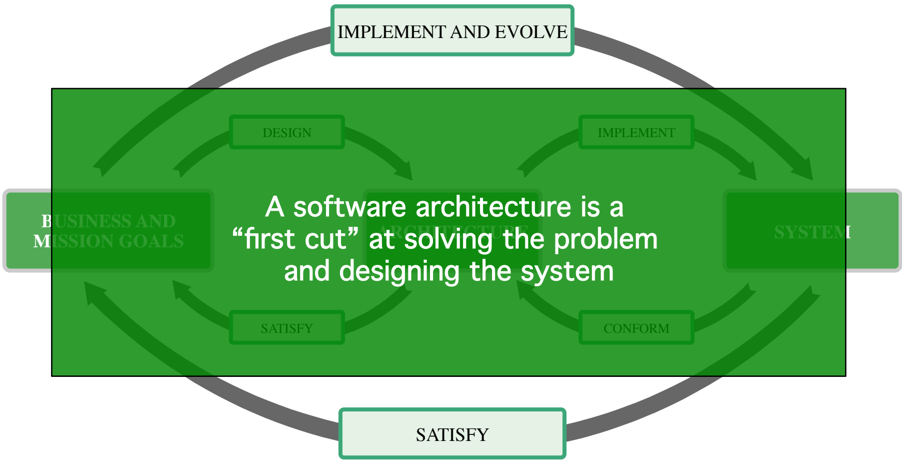
Comment: This behavior is what allows elements to interact with each other, which is clearly part of the architecture. This does not mean that the exact behavior and performance of every element must be documented in all circumstances; but to the extent that an element's behavior influences how another element must be written to interact with it or influences the acceptability of the system as a whole, this behavior is part of the software architecture.

Implications of Previous Definition

Implication #5: *Definition is indifferent as to whether the architecture for a system is a good one or a bad one*,

- Architecture will allow or prevent the system from meeting its behavioral, performance, and life-cycle requirements.
- Architecture Evaluation is important
  - Assuming that we do not accept trial and error as the best way to choose an architecture for a system—that is, picking an architecture at random, building the system from it, and hoping for the best—this raises the importance of ***architecture evaluation*****.**

Architecture Design and Analysis



CHALMERS Chalmers University of Technology

Software Processes and Architecture Business Cycle

- Creating the business case for the system
- Understanding the requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing or evaluating the architecture
- Implementing the system based on the architecture
- Ensuring that the implementation conforms to the architecture

"If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development."
-- Barry Boehm, 1995

The quality and longevity of a software system is determined by its architecture!!

Why is an Architecture Important
The Technical Perspective

- Communication among stakeholders
  - Software architecture represents a common abstraction of a system that most if no all of the system's stakeholders can use as a basis for mutual understanding, negotiation, consensus, and communication
- Early design decisions
  - Software architecture manifests the earliest design decisions about a system, and these early bindings carry weigh far out of proportion to their individual gravity with respect to the system's remaining development, its deployment, and its maintenance life.
- Transferable abstraction of a system
  - Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its elements work together.
  - The model is transferable across systems, e.g., it can be applied to other systems exhibiting similar quality attribute and functional requirements, thus promoting large-scale reuse.

Architecture Stakeholders

- An architecture is the result of **business** and **technical** decisions among stakeholders
- Architectures are influenced by
  - System stake holders include: Customer, end users, project manager, maintainers, system owners, marketers (e.g., think of cloud, iPhone, SOA, WWW, etc)
  - Developing organization
  - Background and experience of architects
  - Technical environment (e.g., WWW, Middleware, SOA)

Functional vs. Non-Functional behavior

(i) Functional behavior

(ii) Non-functional behavior (aka quality attributes, extra-functional behavior)

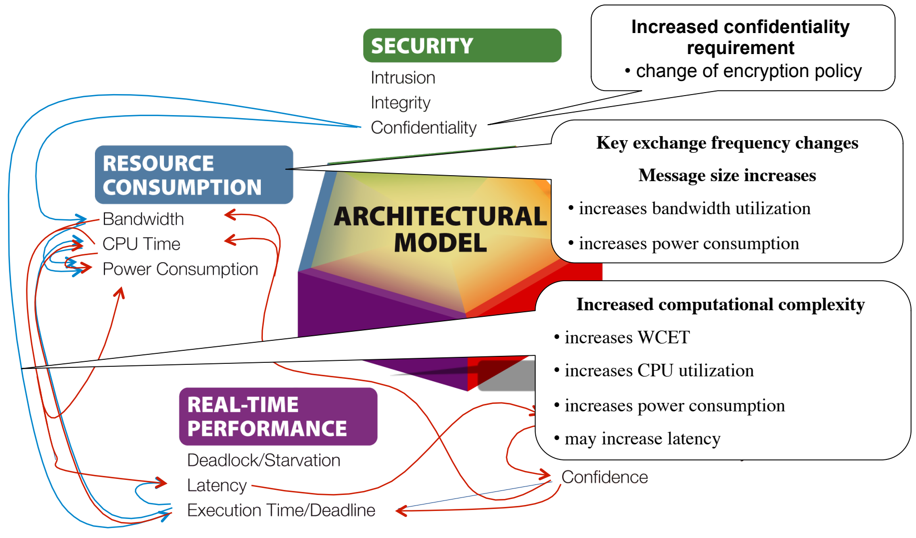
- Performance: real-time
- Security
- Reliability
- Availability
- Maintainability
- Evolvability
- X-ility....

Observation #1: Functional behavior gives the "uniqueness" of the software/ system…. Non-functional behavior drives the perceived quality of the software/ system

Observation #2: Many quality attributes are system attributes, i.e., it involves software and hardware.

Observation #3: Quality attributes are not independent…

Multi-Dimensional Analysis



Architecture Manifests Earliest Set of Design Decisions

Architecture

- defines constraints on implementation
- dictates organizational structure
- inhibits or enables a system’s quality attributes
- is analyzable and a vehicle for predicting system qualities
- makes it easier to reason about and manage change
- helps in evolutionary prototyping
- enables more accurate cost and schedule estimates

Process recommendations

- Architecture should be the product of a single architect or a small group of architects with an identified leader
- Architect team should have functional requirements for the system and an articulated prioritized list of quality attributes that the architecture is expected to satisfy
- Architecture should be well documented, and circulated and reviewed by system stakeholders
- Architecture should be analyzed for applicable quantitative measures and formally evaluated for quality attributes before it is too late to make changes to it.
- Architecture should lend itself to incremental refinement and implementation

Architectural Patterns

- Blackboard
- Client-server (2-tier, n-tier, peer-to-peer, Cloud Computing all use this model)
- Database-centric architecture (broad division can be made for programs which have database at its center and applications which don't have to rely on databases, E.g. desktop application programs, utility programs etc.)
- Distributed computing
- Event Driven Architecture
- Front-end and back-end
- Implicit invocation
- Monolithic application
- Peer-to-peer
- Pipes and filters
- Plugin
- Representational State Transfer
- Rule evaluation
- Search-oriented architecture (A pure SOA implements a service for every data access point)
- Service-oriented architecture
- Shared nothing architecture
- Software componentry (strictly module-based, usually object-oriented programming within modules, slightly less monolithic)
- Space based architecture
- Structured (module-based but usually monolithic within modules)
- Three-tier model (An architecture

Outline

- What is an Architecture
- What is the rationale and purpose of architecting
- I.e., what are the problems architecting aims to address
- **Designing and Architecting next generation aircraft**
- Architectural Assessment
- The SAE AADL architecture description language – An overview

Does Model-Based Development Scale?

Airbus A380

Length	239 ft 6 in
Wingspan	261 ft 10 in
Maximum Takeoff Weight	1,235,000 lbs
Passengers	Up to 840
Range	9,383 miles

Systems Developed Using MBD

- Flight Control
- Auto Pilot
- Flight Warning
- Cockpit Display
- Fuel Management
- Landing Gear
- Braking
- Steering
- Anti-Icing
- Electrical Load Management

The Problem

- Airbus and Boeing have data that support Systems doubling in size and complexity every 2 years.
- Growing use of Integrated Modular Electronics and COTS
- Requirements continue to be refined throughout product lifecycle
- Integration costs increasing as systems do not function as "specified"
  - Unanticipated Interactions (Emergent behavior)
- Desire to design a new airplane every year or 2 instead of every 10.
- Desire to substitute subsystems on the airplane from different vendors. (Airline can make choice as is currently done on Engines)
- Desire to support incremental certification

Aerospace Vehicle Systems Institute

System and Software Integration Verification

Texas Engineering Experiment Station

Participants

- Active – BAE, Boeing, DoD (Army, Navy), FAA, GE Aerospace (Smiths), Honeywell, Lockheed Martin, Rockwell Collins, Airbus, Dassault-Aviation, JPL/NASA
- General Dynamics, Raytheon, Thales
- Software Engineering Institute, Carnegie Mellon

Project Overview

- Overall Concept of Operations
  - Design and production based on early and continuous integration (virtual => physical)
  - Integrate, then build
- Objective
  - Shift architecting, design, and production activities to explicitly address integration issues early, reducing program execution risks, cycle time and cost
- Approach
  - Adopt/develop “integration-based” software and system development processes with emphasis on integrating component-based, model-based and proof-based development

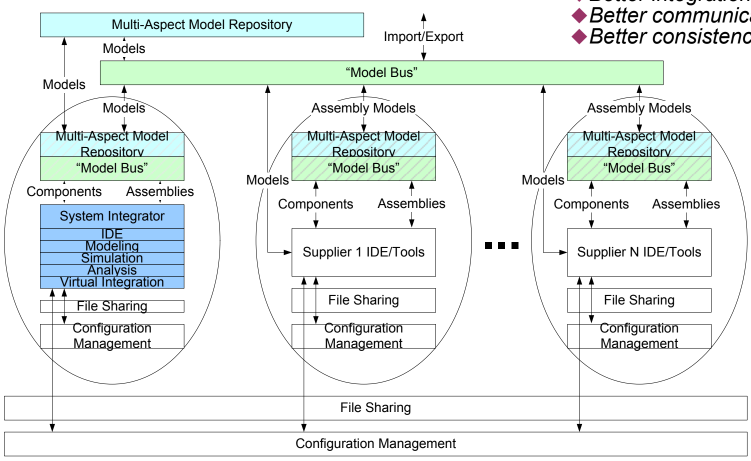
Expanded Objectives

- Integrate system, software, and hardware integration models in one framework
  - Support component-based system assurance through analysis of functionality, performance, safety and security
  - Increase the degree of standardization and commonality for technical data exchanged between airframers, suppliers, and regulatory authorities
- Integrate – then build
  - Predict system behavior through analysis to ensure it is acceptable
  - Build to the requirements determined through the analysis
- Reduce the cost of developing avionic systems
  - Maintain or improve existing levels of safety and security
- Start with the aerospace industry
  - Leverage capabilities developed in related domains
  - Coordinate with related domains when advantageous
- Foster U.S. Government and Aerospace industry Cooperation
  - Complement the large, government/industry funded European R&D efforts

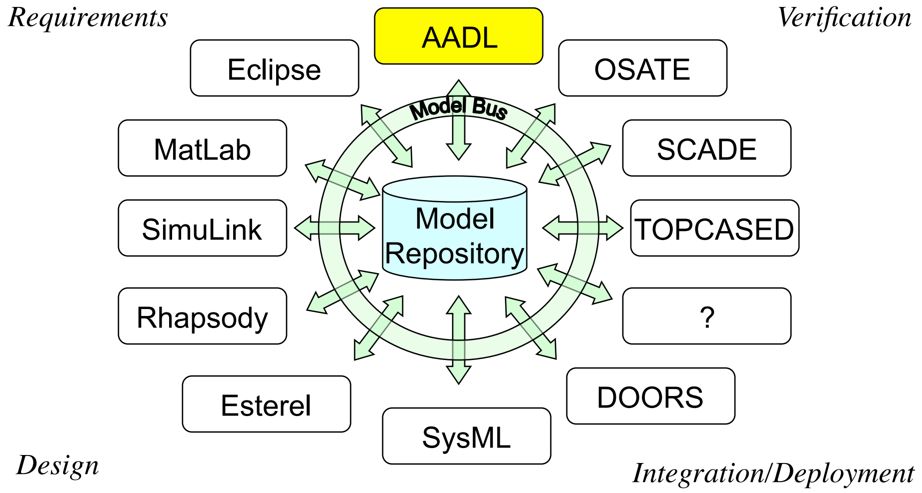
Single Information and Relationships Repository

- Integrate information and relationships in a single repository with a “model bus”

◆ *Better requirements*
◆ *Better integration*
◆ *Better communication*
◆ *Better consistency*

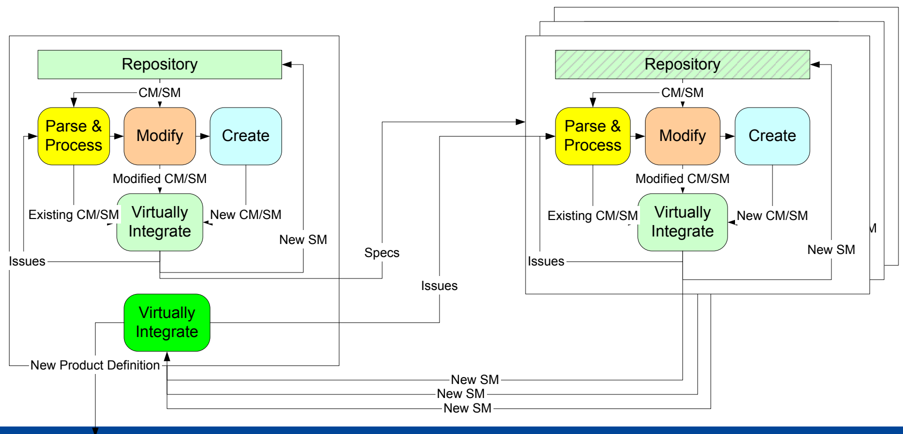


Overview of Multi-Aspect Model Repository & Model Bus



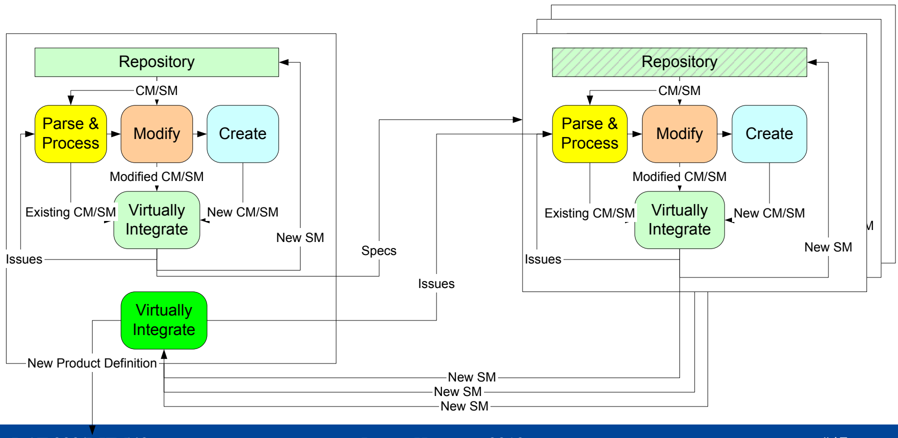
Modified Business Model

- System Integrator defines a new product using internal repository of virtual “parts”
- Specifications for virtual subcomponents sent to suppliers

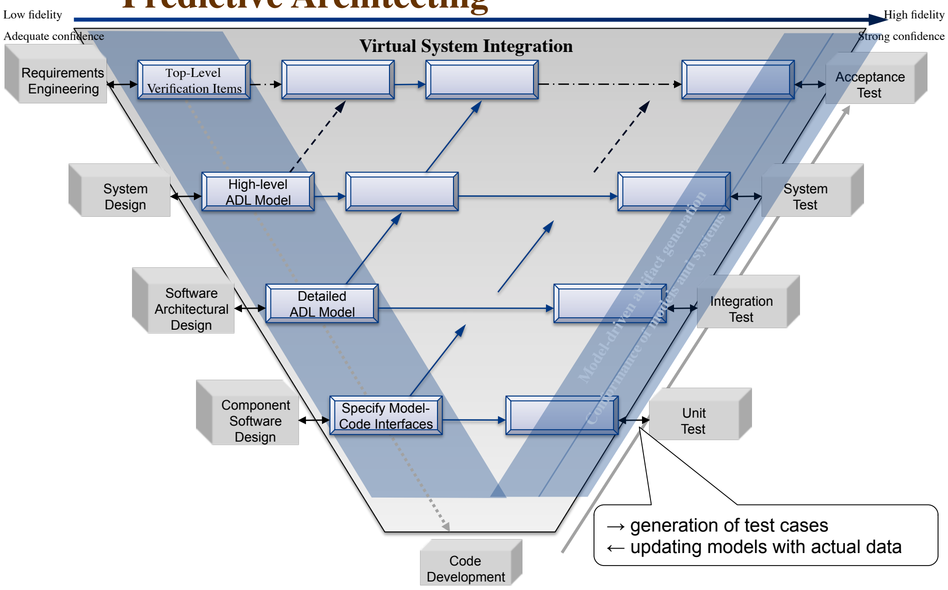


Modified Business Model (continued)

- Virtual parts returned for virtual integration into a virtual product
  - Cost savings realized by finding problems early on virtual parts
- Once the virtual product is satisfactory, the actual product is developed
  - Cycle-time reduction realized since re-work on physical parts virtually eliminated



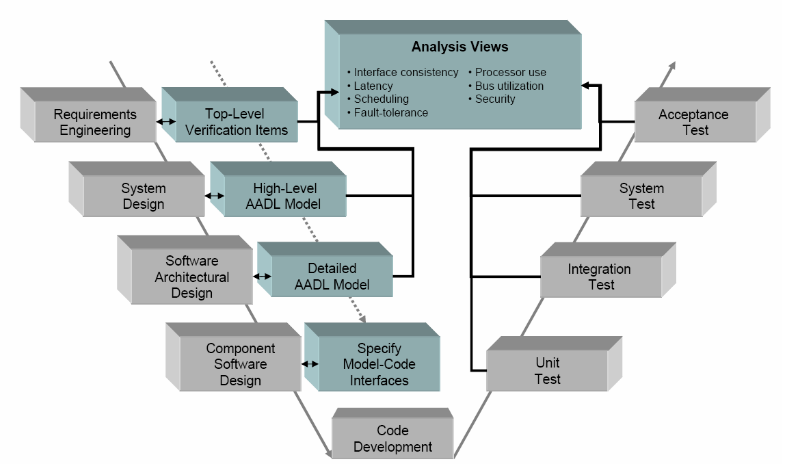
Predictive Architecting



Predictable System Integration Through Model-Based Engineering

- Reduce the risks
  - Analyze system early and throughout life cycle
  - Understand system wide impact
  - Validate assumptions across system
- Increase the confidence
  - Validate models to complement integration testing
  - Validate model assumptions in operational system
  - Evolve system models in multiple fidelity
- Reduce the cost
  - Fewer system integration problems
  - Tool-based engineering support
  - Simplified life cycle support

AADL in the Life Cycle



System modeling and analysis with AADL in the context of a development life cycle

Model-Based Engineering Benefits

Analyzable models drive development
Prediction of runtime characteristics at different fidelity
Bridge between control & software engineer
Prediction early and throughout lifecycle
Reduced integration & maintenance effort

- Benefits of modeling and architecture standards

Common modeling notation across organizations
Single architecture model augmented with properties
Interchange & integration of architecture models
Tool interoperability & integrated engineering environments

Outline

- What is an Architecture
- What is the rationale and purpose of architecting
- I.e., what are the problems architecting aims to address
- Designing and Architecting next generation aircraft
- **The SAE AADL architecture description language – An overview**

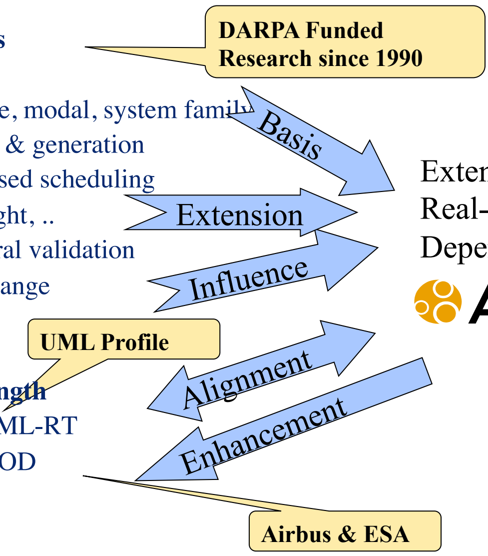
SAE Architecture Analysis & Design Language (AADL) Standard

- Notation for specification of task and communication architectures of Real-time, Embedded, Fault-tolerant, Secure, Safety-critical, Software-intensive systems, of hardware platforms, and deployment
- Fields of application: Avionics, Automotive, Aerospace, Autonomous systems, …
- Based on 15 Years of DARPA funded technologies
- Standard approved & published Nov 2004
- www.aadl.info

AADL in Context

Research ADLs
- MetaH
  - Real-time, modal, system family
  - Analysis & generation
  - RMA based scheduling
- Rapide, Wright, ..
  - Behavioral validation
- ADL Interchange
  - ACME

Industrial Strength
- UML 2.0, UML-RT
- HOOD/STOOD
- SDL



Extensible
Real-time
Dependable
AADL

Key Elements of SAE AADL Standard

- Core AADL language standard
  - Textual & graphical, precise semantics, extensible
- AADL Meta model & XMI/XML standard
  - Model interchange & tool interoperability
- Error Model Annex as standardized extension
  - Fault/reliability modeling, hazard analysis
- UML 2.0 profile for AADL
  - Transition path for UML practitioner community

http://www.aadl.info

AADL: The Language

- Precise execution semantics for components & interactions
  - Thread, process, data, subprogram, system, processor, memory, bus, device
- Continuous control & event response processing
  - Data and event flow, synchronous call/return, shared access
  - End-to-End flow specifications
- Operational modes & fault tolerant configurations
  - Modes & mode transition
- Modeling of large-scale systems
  - Component variants, packaging of AADL models
- Accommodation of diverse analysis needs
  - Extension mechanism, standardized extensions

Focus Of SAE AADL

- **Component View**
  - Model of system composition & hierarchy
  - Software, execution platform, and physical components
  - Well-defined component interfaces
- **Concurrency & Interaction View**
  - Time ordering of data, messages, and events
  - Dynamic operational behavior
  - Explicit interaction paths & protocols
- **Deployment view**
  - Execution platform as resources
  - Binding of application software
  - Specification & analysis of runtime properties
    - timeliness, throughput, reliability, graceful degradation, …

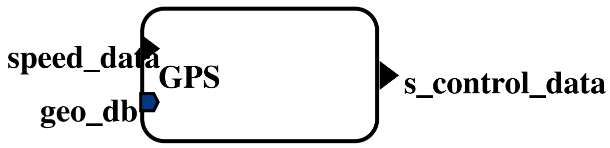
Predictable System Integration Through Model-Based Engineering

- Reduce the risks
  - Analyze system early and throughout life cycle
  - Understand system wide impact
  - Validate assumptions across system
- Increase the confidence
  - Validate models to complement integration testing
  - Validate model assumptions in operational system
  - Evolve system models in multiple fidelity
- Reduce the cost
  - Fewer system integration problems
  - Tool-based engineering support
  - Simplified life cycle support

System Type

System

```
system GPS
features
  speed_data: in data port metric_speed
     {SEI::BaseType => UInt16;};
  geo_db: requires data access real_time_geoDB;
  s_control_data: out data port state_control;
flows
  speed_control: flow path
     speed_data -> s_control_data;
properties SEI::redundancy => Dual;
end GPS;
```

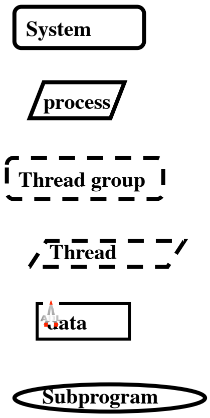


System Implementation

```
system implementation GPS.secure
subcomponents
  decoder: system PGP_decoder.basic;
  encoder: system PGP_encoder.basic;
  receiver: system GPS_receiver.basic;
connections
  c1: data port speed_data -> decoder.in;
  c2: data port decoder.out -> receiver.in;
  c3: data port receiver.out -> encoder.in;
  c4: data port encoder.out -> s_control_data;
flows
 speed_control: flow path speed_data -> c1 -> decoder.fs1
            -> c2 -> receiver.fs1 -> c3 -> encoder.fs1
            -> c4 -> s_control_data;
modes none;
properties SEI::redundancy_scheme => Primary_Backup;
end GPS.secure;
```

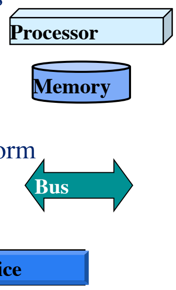
Application Components

- System: hierarchical organization of components
- Process: protected address space
- Thread group: organization of threads in processes
- Thread: a schedulable unit of concurrent execution
- Data: potentially sharable data
- Subprogram: callable unit of sequential code



Execution Platform Components

- Processor – provides thread scheduling and execution services
- Memory – provides storage for data and source code
- Bus – provides physical connectivity between execution platform components
- Device – interface to external environment



Some Standard Properties

Thread

- Dispatch_Protocol => Periodic;
- Period => 100 ms;
- Compute_Deadline => value (Period);
- Compute_Execution_Time => 10 ms .. 20 ms;
- Compute_Entrypoint => “speed_control”;
- Source_Text => “waypoint.java”;
- Source_Code_Size => 12 KB;

Dispatch execution properties

Code to be executed on dispatch

File containing the application code

Processor

- Thread_Swap_Execution_Time => 5 us.. 10 us;
- Clock_Jitter => 5 ps;

Bus

- Allowed_Message_Size => 1 KB;
- Propagation_Delay => 1ps .. 2ps;
- bus_properties::Protocols => CSMA;

Protocols is a user defined property

Component Interactions & Modes

Completely defined interfaces & interactions
- Port-based flows
  - State data, events, messages
  - Flow specifications & connections
  - End-to-end flows
- Synchronous call/return
- Shared access

Modal & dynamically configurable systems
- Modeling of operational modes
- Modeling of fault tolerant configurations
- Modeling of different levels of service

AADL Language Extensions

- Model annotation through properties and sublanguages
- New properties defined through property sets
- Standard compliant sublanguage syntax in annex subclauses
- Project-specific language extensions
- Language extensions as approved SAE AADL standard annexes
- Examples
  - Error Model
  - Concurrency Behavior
  - System partitions (e.g., ARINC 653)

Airbus Annex Extension

```
THREAD t
 FEATURES
  sem1 : DATA ACCESS semaphore;
  sem2 : DATA ACCESS semaphore;
END t;

THREAD IMPLEMENTATION t.t1
 PROPERTIES
  Period => 13.96ms;
  cotre::Priority => 1;
  cotre::Phase => 0.0ms;
  Dispatch_Protocol => Periodic;
```

```
ANNEX cotre.behavior {**
  STATES
   s0, s1, s2, s3, s4, s5, s6, s7, s8 : STATE;
   s0 : INITIAL STATE;
  TRANSITIONS
   s0 -[ ]-> s1 { PERIODIC_WAIT };
   s1 -[ ]-> s2 { COMPUTATION(1.9ms, 1.9ms) };
   s2 -[ sem1.wait ! (-1.0ms) ]-> s3;
   s3 -[ ]-> s4 { COMPUTATION(0.1ms, 0.1ms) };
   s4 -[ sem2.wait ! (-1.0ms) ]-> s5;
   s5 -[ ]-> s6 { COMPUTATION(2.5ms, 2.5ms) };
   s6 -[ sem2.release ! ]-> s7;
   s7 -[ ]-> s8 { COMPUTATION(1.5ms, 1.5ms) };
   s8 -[ sem1.release !]-> s0;
 **};
END t.t1;
```

COTRE thread properties

COTRE behavioral annex

Courtesy of

Summary

- What is an Architecture ✔
- What is the rationale and purpose of architecting ✔
- I.e., what are the problems architecting aims to address ✔
- Designing and Architecting next generation aircraft ✔
- Architectural Assessment ✔
- The SAE AADL architecture description language ✔