# Documenting Software Architectures

# Topics

- The purpose for and audience of architecture documentation
- Templates to capture the key documentation contents
  - Views, behavior, and interfaces
- [Architecture Description Languages]
  - Including UML (in your reading)

# Documenting Software Architecture

- Recall the definition of Software Architecture
  - "The software architecture … **structures** … comprise software **elements**… **properties** … **relationships** among them."
- To be useful, this needs to be **communicated**
  - In **documentation**
  - Leverage the concept of architecture **views**
- Write from perspective of the reader (stakeholder)
  - Potentially more than one document for different stakeholders

# Uses of Architecture Documentation

| | |
|---|---|
| Analysts | Validate qualities , tradeoff negotiation |
| Software engineers | Design and implementation |
| Testers | Test development |
| Maintainers | Change impact analysis |
| Quality assurance | Validation of system as built |
| Users | System usability assessment |
| Managers | Work organization and planning |
| New personnel | Training |
| Follow-on architects | Design rationale |

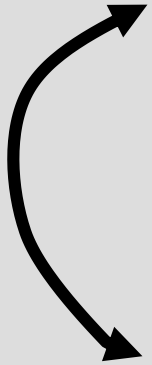
- **Prescriptive** – what should be true by placing **constraints on future decisions** (e.g., implementer)
- **Descriptive** – what is true by communicating **decisions already made** (e.g., project manager)

# Stakeholders and Views

| | Module Views | | | | C&C Views | Allocation Views | |
|---|---|---|---|---|---|---|---|
| **Stakeholder** | **Decomposition** | **Uses** | **Class** | **Layer** | **Various** | **Deployment** | **Implementation** |
| Project Manager | s | s | | s | | d | |
| Member of Development Team | d | d | d | d | d | s | s |
| Testers and Integrators | | d | d | | s | s | s |
| Maintainers | d | d | d | d | d | s | s |
| Product Line Application Builder | | d | s | o | s | s | s |
| Customer | | | | | s | o | |
| End User | | | | | s | s | |
| Analyst | d | d | s | d | s | d | |
| Infrastructure Support | s | s | | s | | s | d |
| New Stakeholder | x | x | x | x | x | x | x |
| Current and Future Architect | d | d | d | d | d | d | s |

*Key:* d = detailed information, s = some details, o = overview information, x = anything

# Notations



- **Informal**
  - Views depicted using **general-purpose diagramming** tools
  - **Natural language** semantics
  - Cannot be formally analyzed
- **Semiformal**
  - Views depicted using **standardized notation**
  - **Rudimentary semantic analysis** can be applied
  - **UML** is a semiformal notation in this sense.
- **Formal**
  - Views are described in a notation that has a **precise** (usually mathematically based) **semantics**.
  - **Formal analysis** of both syntax and semantics is possible.
  - **Architecture description languages** (ADLs)
  - Support automation through associated tools.

# The Architecture Document

1. Views
2. Interfaces
3. The “big picture”

**Reference template**

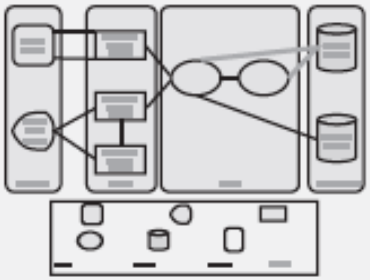
# 1. Views

- **Choose** relevant **views** based on **stakeholders and uses**
  - Module, component-connector, allocation, data
  - Prioritize – top-down, **breadth first** is best
- Document each **view** with a **consistent template**

# A View Document Template

## Template for a View

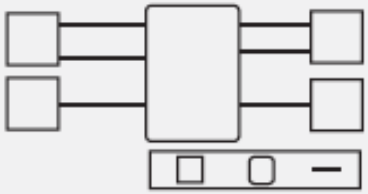
**Section 1. Primary Presentation**



**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. Rationale**

* Optional

# Documenting Behavior

- Structural information does not capture sequences of interactions (e.g., real time dependencies, concurrency)
- Relevant UML diagrams
  - Sequence diagrams
  - Communication diagrams
  - Activity diagrams
  - State charts

# 2. Documenting Interfaces: Interface Specification Template

| | |
|---|---|
| Identity | Meaningful name |
| Syntax | The signature – arguments, data types |
| Semantics (usage guide) | Pre and post conditions, return values, exceptions, call orchestration, event call backs, atomic/interruptible |
| Variability | Configuration, binding |
| Quality attributes | Quality implications such as performance, security |
| Resource requirements | Computational and memory usage |
| Rationale | Why is it designed this way? |

**Architecturally significant component interfaces are best candidates**

# 3. Document Across Views – Big Picture

- Information that applies to the architecture and architecture documentation **package as a whole**

**Template for Documentation Beyond Views**

Section 1: Documentation Roadmap
Section 2: How a View is Documented

Section 3: System Overview
Section 4: Mapping Between Views
Section 5: Rationale
Section 6: Glossary, Acronym List

**All systems are hierarchical**

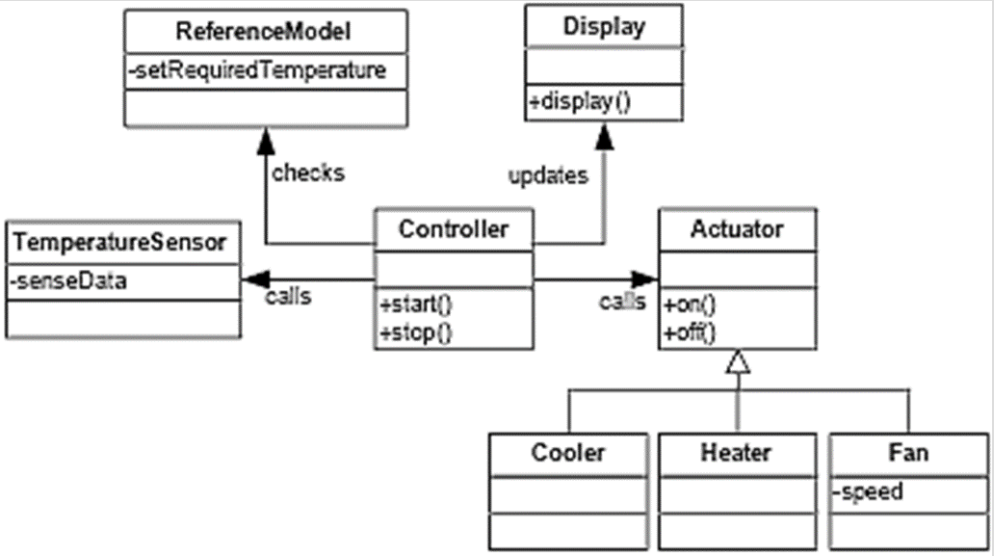
# Document Across Views – Big Picture

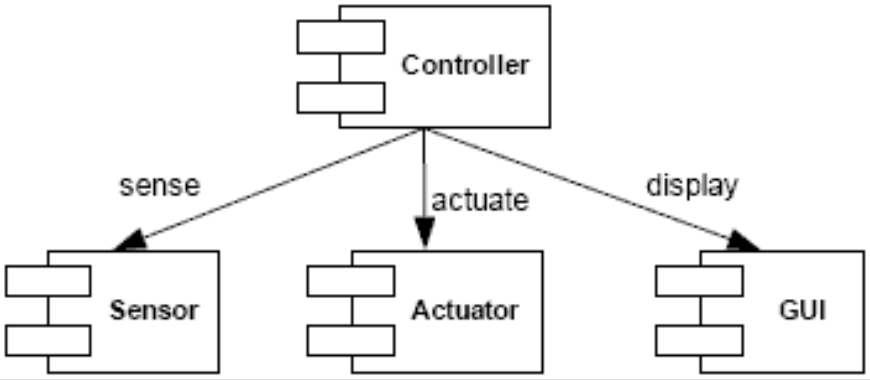
- **Roadmap –** background scope and summary, document and view overview
- **View template** description
- **System overview –** give users a **consistent mental model** of the **system** and **purpose**
  - High level **system context diagram** in relationship to external systems
  - Short **description** of the **system's function** and **users**
- **Mapping between views**
- **Rationale –** explain how design pattern and tactics decisions satisfy architecturally significant requirements

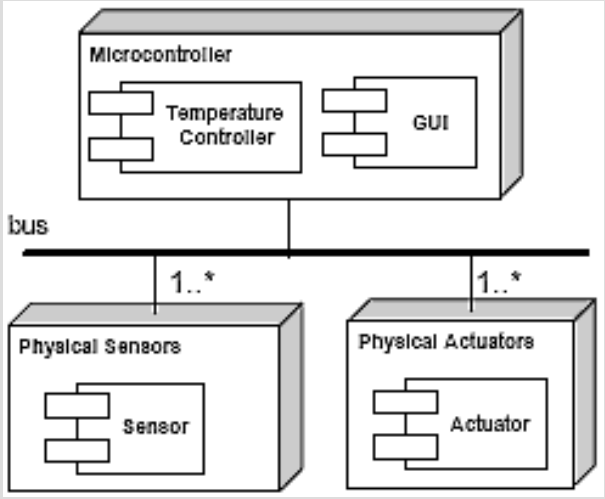
# Mapping Between Views

- **How does the architecture work as a unified whole**?
- Element **associations** across views are **many-to-many**.
- Capture view-to-view associations as **tables**.
  - Name the correspondence between elements across views.
  - Examples
    - "**is implemented by**" for mapping from a component-and-connector view to a module view
    - "i**mplements**" for mapping from a module view to a component-and-connector view
    - "**included in**" for mapping from a decomposition view to a layered view

# Vehicle HVAC System



# Example View Association Table

| Module | C-C Controller | C-C Sensor | C-C Actuator | C-C Display |
|---|---|---|---|---|
| Controller | X | | | |
| TempSensor | | X | | |
| Ref Model | X | | | |
| Display | | | | X |
| Actuator | | | X | |
| Cooler | | | X | |
| Heater | | | X | |
| Fan | | | X | |

**Module to Component-Connector “Implements”**

# Practitioner Guidelines for Software Architecture Documents

- 50 pages or less
- Use a “standard” **template**
- Describe the architecture **top down**, **abstract to concrete**
- **Requirements traceability** is evident
- **Glossary** and **references** to other documents
- Version control
- **Good naming** – contextual meaning, avoid jargon, consistent; good names end up in code

Hitchhiker's Guide to Software Architecture and Everything Else - by Michael Stal

# Documenting Architecture in an Agile Development Project

- Some architecture necessary as a **project roadmap** and to capture **design decisions**
- Adopt a **template or standard organization** to document
- **Document views if** (but only if) a strongly identified **stakeholder constituency**.
  - **Fill in view** template sections when **information** becomes **available**
- **Document** just **enough** design to **enable coding**

# Supplemental Material
# Architecture Description Languages

# Architecture Description Languages (ADLs)

- **Documenting elements and relationships** "demands" a **graphical notation** (sophisticated, formal box-and-arrow diagrams)
- An ADLs "is a computer language used to describe and represent software architectures"
- ISO/IEC Standard 42010, "Systems and Software Engineering – Architecture Description"
  - "... form of expression used for the description of architectures"
  - Specifies minimum requirements for ADLs
  - Based on IEEE Standard 1471 "Architecture Description"

# ADL Positives

- Provide a **formal** way of representing architecture
- Intended to be **human and machine readable**
- Support describing a system at a higher level than previously possible
- Permit **analysis** of architectures – completeness, consistency, ambiguity, performance, etc.
- Can support **automatic generation** of software systems

# ADL Negatives

- No **universal agreement** on what ADLs should represent, particularly as regards the behavior of the architecture
- Representations currently in use are relatively **difficult to parse** and are **not supported** by **commercial tools**
- Most ADL work today has been undertaken with **academic** rather than commercial goals in mind
- Most ADLs tend to be very **vertically optimized** toward a particular kind of analysis

# Some ADLs

- Leading candidates
  - ABACUS
  - ACME (CMU/USC)
  - Rapide (Stanford)
  - Wright (CMU)
  - Unicon (CMU)
- Secondary candidates
  - Aesop (CMU)
  - MetaH (Honeywell)
  - C2 SADL (UCI)
  - SADL (SRI)

- *Bass, Clements, and Kazman have adopted UML as an ADL*
- *A key goal of defining UML 2.0 was to enable better architecture description*