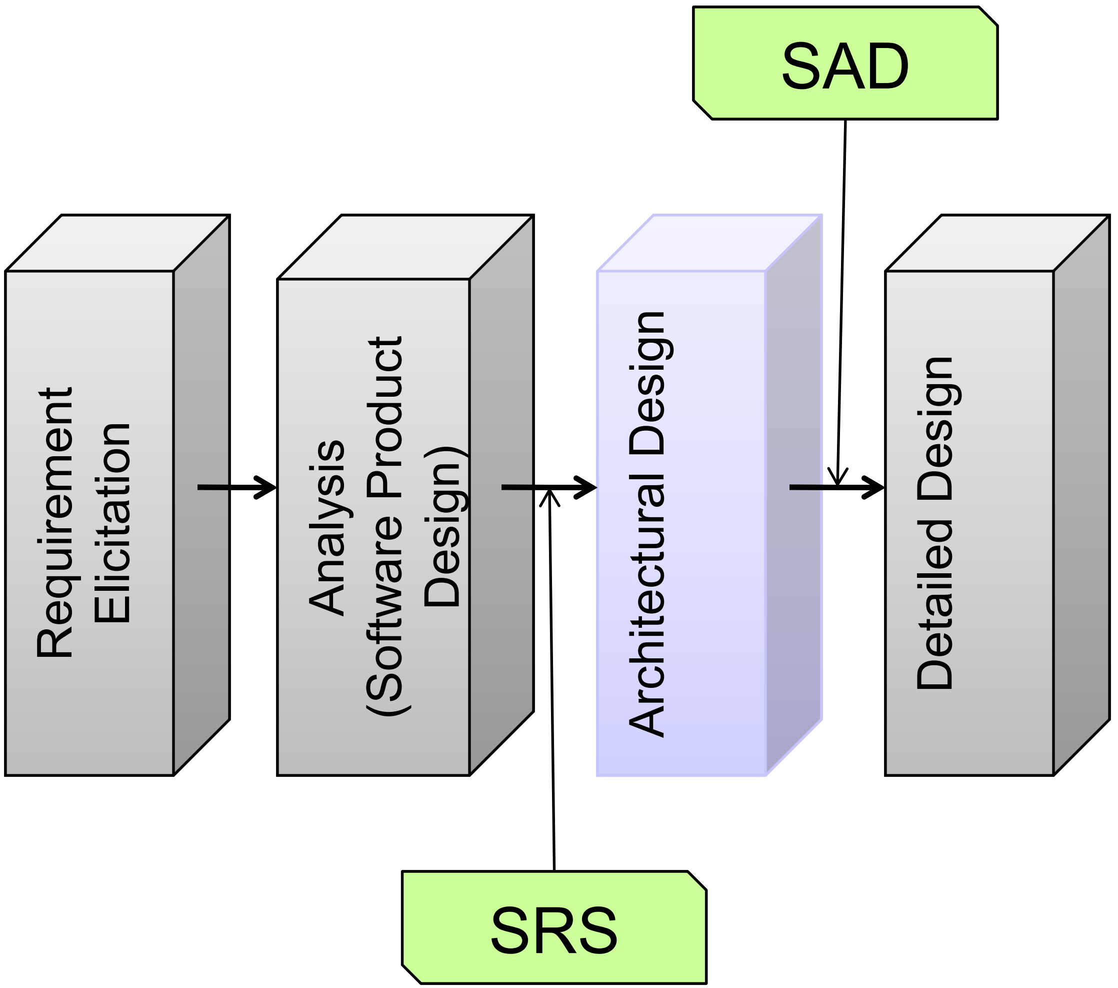
Introduction to Software Engineering

ECSE-321

Unit 9 – Architectural Design Approaches

Architectural Design

Software Architecture Document



Why Architectural Design?

Architecture is often needed to

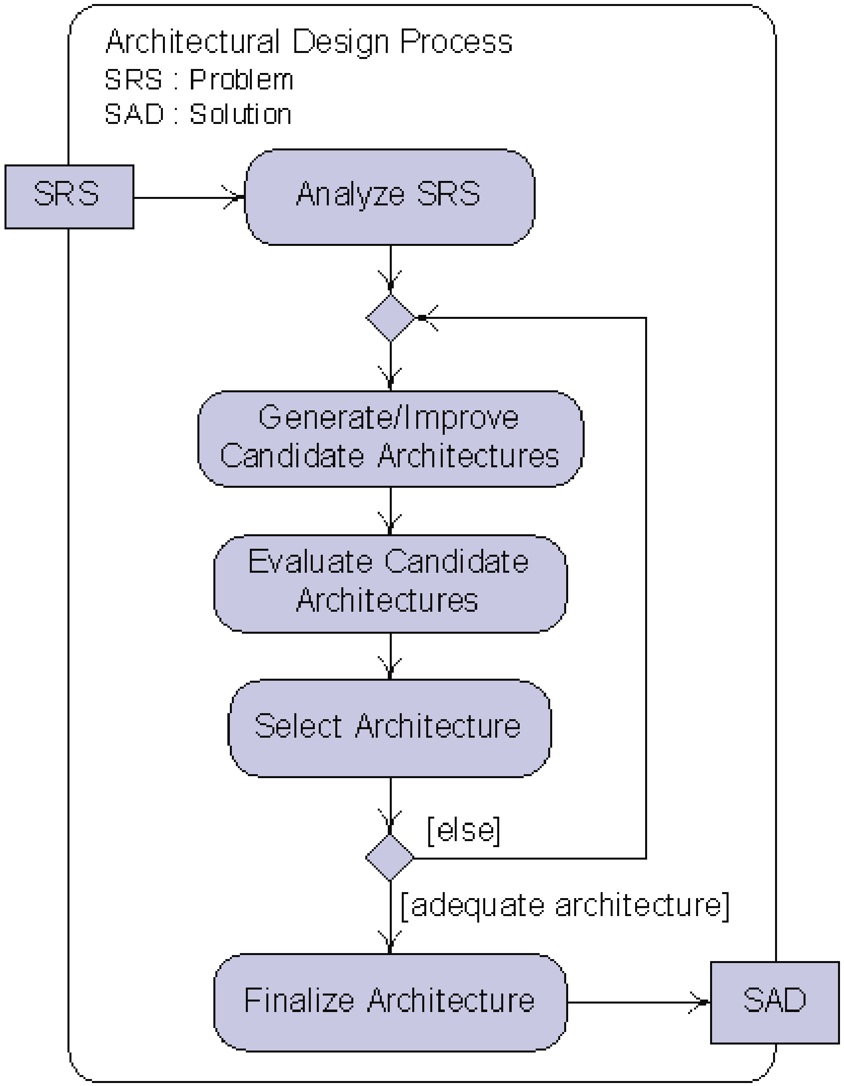
- Judge feasibility
- Convince stakeholders their needs can be met
- Conduct tradeoff analyses
- Plan the project

Why Architectural Design?

- Architectural design also influences the choices for:
  - Code libraries and other assets
  - Organizational structure
  - Knowledge and experience of designers
- Architectures influence people and organizations too
  - Team that works on the project
  - Organizations participating in outsourced projects

Architectural Design Process

Software requirement doc.



Software architecture document

Software Architecture Document

- *Product Overview*—Product vision, stakeholders, target market, etc.
- *Architectural Models*—Specification using various models, both static and dynamic
  - DeSCRIPTR
- *Mapping Between Models*—Tables and text relating models
- *Architectural Design Rationale*—Explanation of difficult, crucial, puzzling, and hard-to-change design decisions

Quality Attributes

A **quality attribute** is a characteristic or property of a software product independent of its function that is important in satisfying stakeholder needs.

- Non-functional requirements
- Architectures have a big influence on quality attributes
- *Development* or *operational* attributes

Development Attributes

- *Maintainability*—Ease with which a product can be corrected, improved, or ported
  - Often subdivided
- *Reusability*—Degree to which a product's parts can be reused in another product
- Others

Operational Attributes

- *Performance*—Ability to accomplish product functions within time or resource limits
- *Availability*—Readiness for use
- *Reliability*—Ability to behave in accord with requirements under normal operating conditions
- *Security*—Ability to resist being harmed or causing harm by hostile acts or influences
- Others

Notations for Architectural Specifications

Type of Specification	Useful Notations
Decomposition	Box-and-line diagrams, class diagrams, package diagrams, component diagrams, deployment diagrams
States	State diagrams
Collaborations	Sequence and communication diagrams, activity diagrams, box-and-line diagrams, use case models
Responsibilities	Text, box-and-line diagrams, class diagrams
Interfaces	Text, class diagrams
Properties	Text
Transitions	State diagrams
Relationships	Box-and-line diagrams, component diagrams, class diagrams, deployment diagrams, text

Interfaces

An **interface** is a communications boundary between entities.

An **interface specification** describes the mechanism that an entity uses to communicate with its environment.

Interface Specifications

- *Syntax*—Elements of the communications medium and how they are combined to form messages
- *Semantics*—The meanings of messages
- *Pragmatics*—How messages are used in context to accomplish tasks
- Interface specifications should cover the syntax, semantics, and pragmatics of the communication between a module and its environment.

Interface Specification Template

1. Services Provided
   For each service provided specify its
      a) Syntax
      b) Semantics
      c) Pragmatics
2. Services Required
   Specify each required service by name.
   A service description may be included.
3. Usage Guide
4. Design Rationale

Semantic Specification

- A **precondition** is an assertion that must be true at the start of an activity or operation.
- A **postcondition** is an assertion that must be true at the completion of an activity or operation.
- Pre- and postconditions can together specify what happens when an operation executes, thus explaining its semantics.

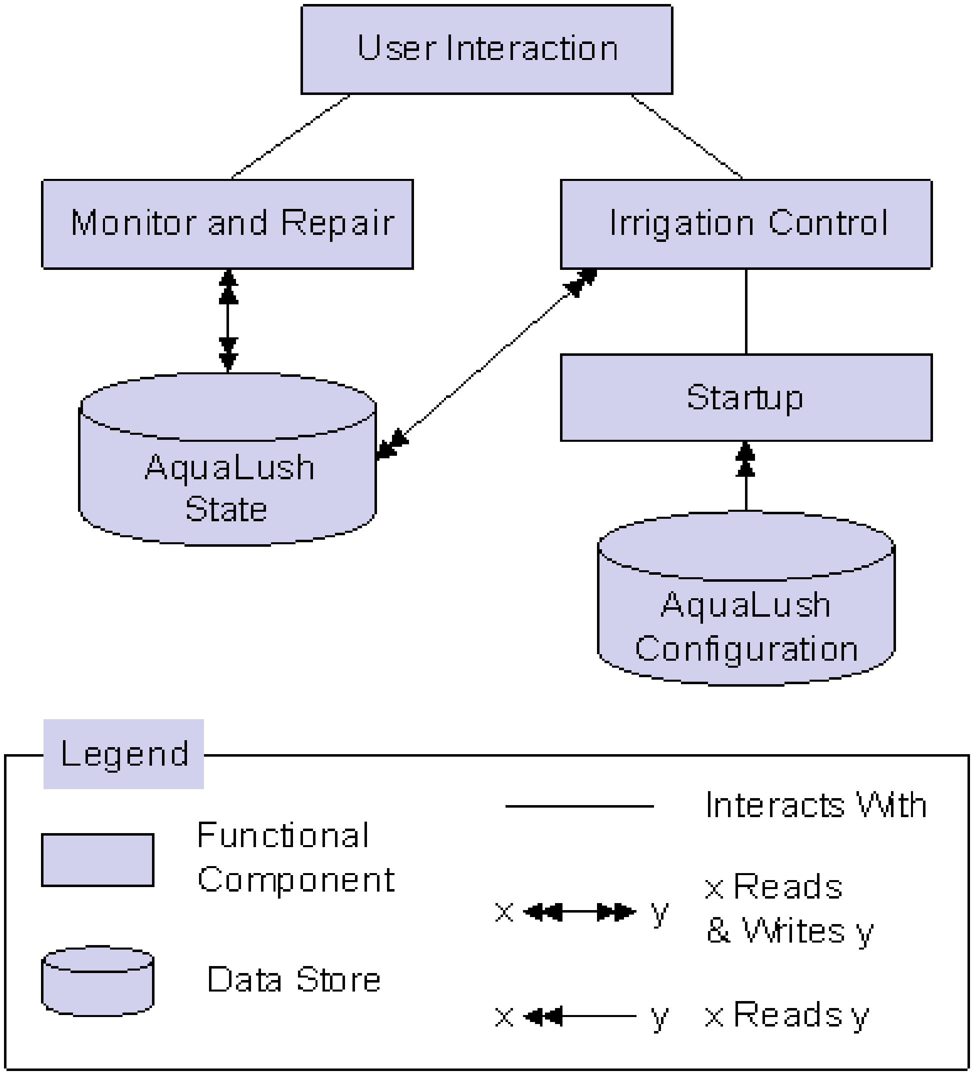
Architectural Modeling Notations

- Several notations for architectural modeling
  - Box-and-line diagrams
  - UML package diagrams
  - UML component diagrams
  - UML deployment diagrams

Box-and-Line Diagrams

- Icons (boxes) connected with lines
- No rules governing formation
- Used for both static and dynamic modeling
- Good idea to include a legend

Box-and-Line Diagram Example



Box-and-Line Diagram Heuristics

- *Make box-and-line diagrams only when no standard notation is adequate*.
- Keep the boxes and lines simple.
- Make symbols for different things look different.
- Use symbols consistently in different diagrams.
- Use grammatical conventions to name elements (noun phrases for things and verb phrases for actions)

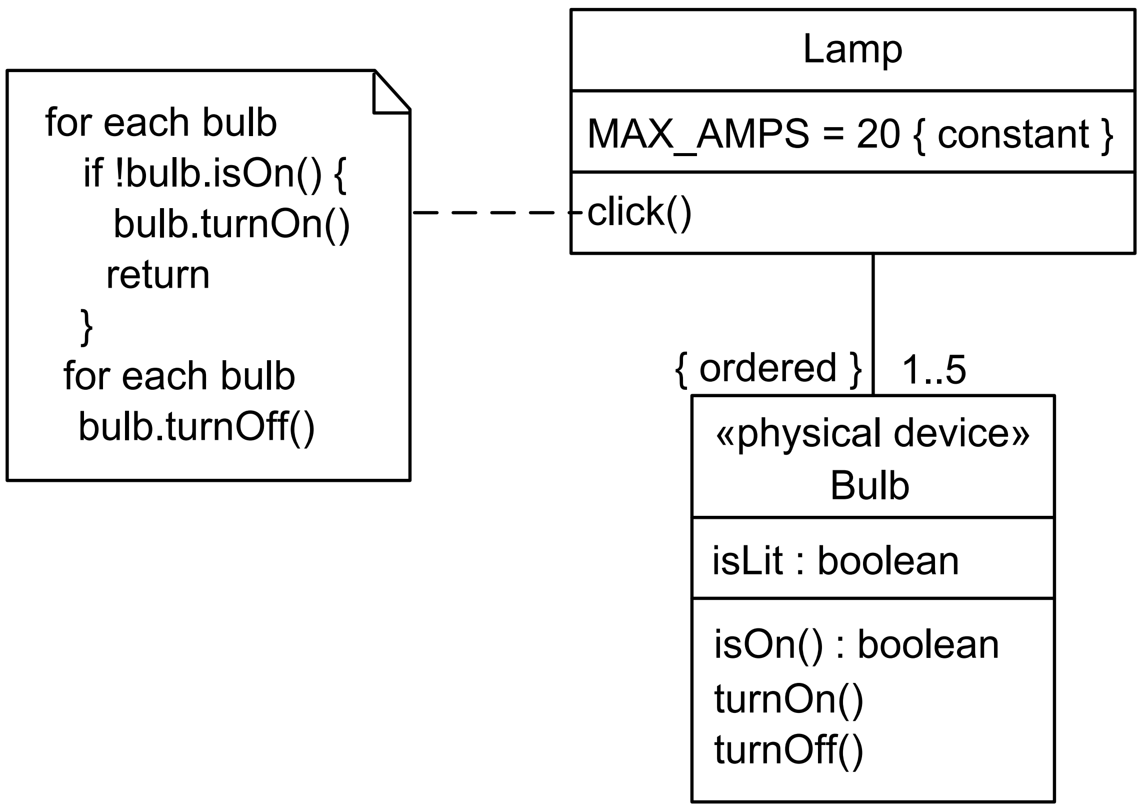
UML Notes and Constraints

- *Note*—A dog-eared box connected to model elements by a dashed line
  - May contain arbitrary text
  - Used for comments and specifications
- *Constraint*—A statement that must be true of entities designated by model elements
  - Written inside curly brackets
  - Beside single model elements
  - Beside a dashed line connecting several model elements

UML Properties and Stereotypes

- *Property*—Characteristic of an entity designated by a model element
  - List of tagged values in curly brackets
  - Tagged value: *tag* = *value*
  - Boolean properties that are true may drop the value and equals sign
- Stereotype—A model element given more specific meaning
  - Shown with icons, colors, graphics
  - Stereotype keywords between guillemots, for example «interface»

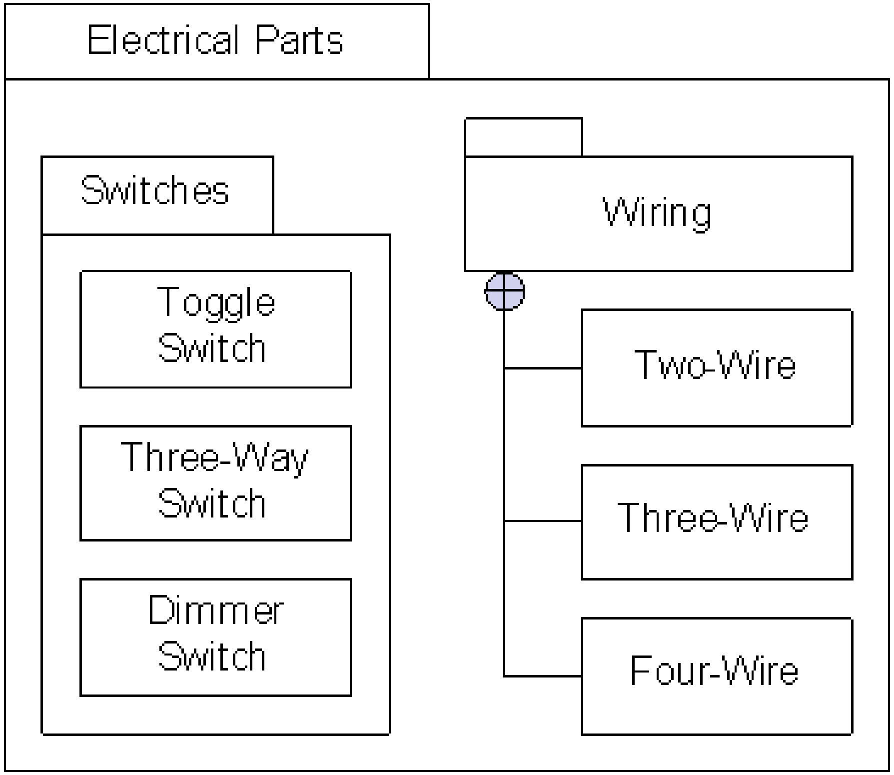
Common Elements Example



UML Packages

- A UML *package* is a collection of model elements, called *package members*.
- The package symbol is a file folder
  - Package name in tab if body is occupied, otherwise in the body
  - Members shown in body or using a containment symbol (circled plus sign)

Package Diagram Example



Software Components

- A **software component** is a reusable, replaceable piece of software.
- **Component-based development** is an approach in which products are designed and built using commercially available or custom-built software components.

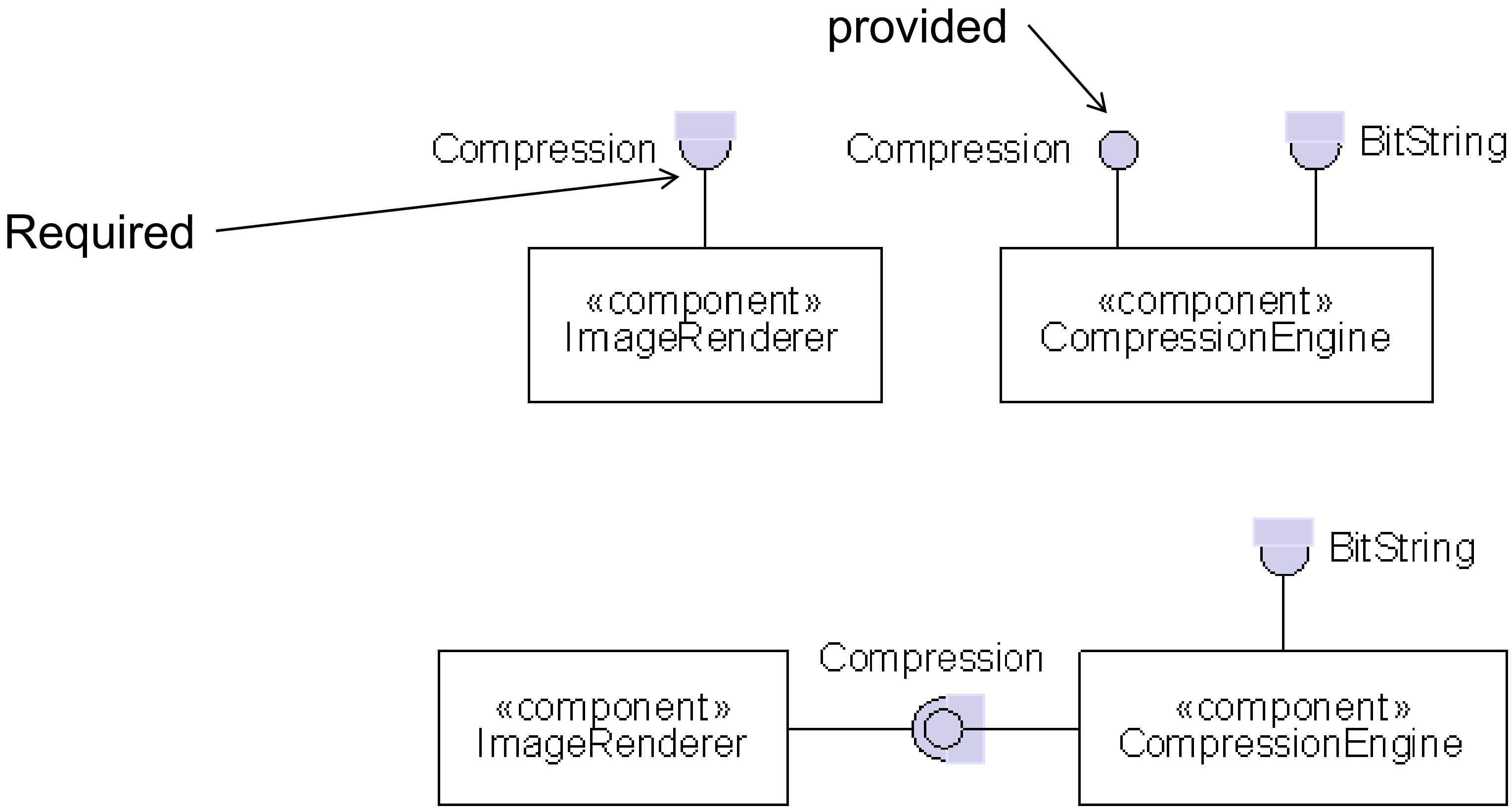
UML Component Diagrams

- A UML **component** is a modular, replaceable unit with well-defined interfaces.
  - Component symbols are rectangles containing names
  - Stereotyped «component»
- A UML **component diagram** shows components, their relationships to their environment, and their internal structure.

UML Interfaces

- A UML **interface** is a named collection of public attributes and abstract operations.
  - Represented by special ball and socket symbols
- *Provided interface*—Realized by a class or component
  - Represented by a ball or lollipop symbol
- *Required interface*—Needed by a class or component
  - Represented by a socket symbol

Interface Symbols Example



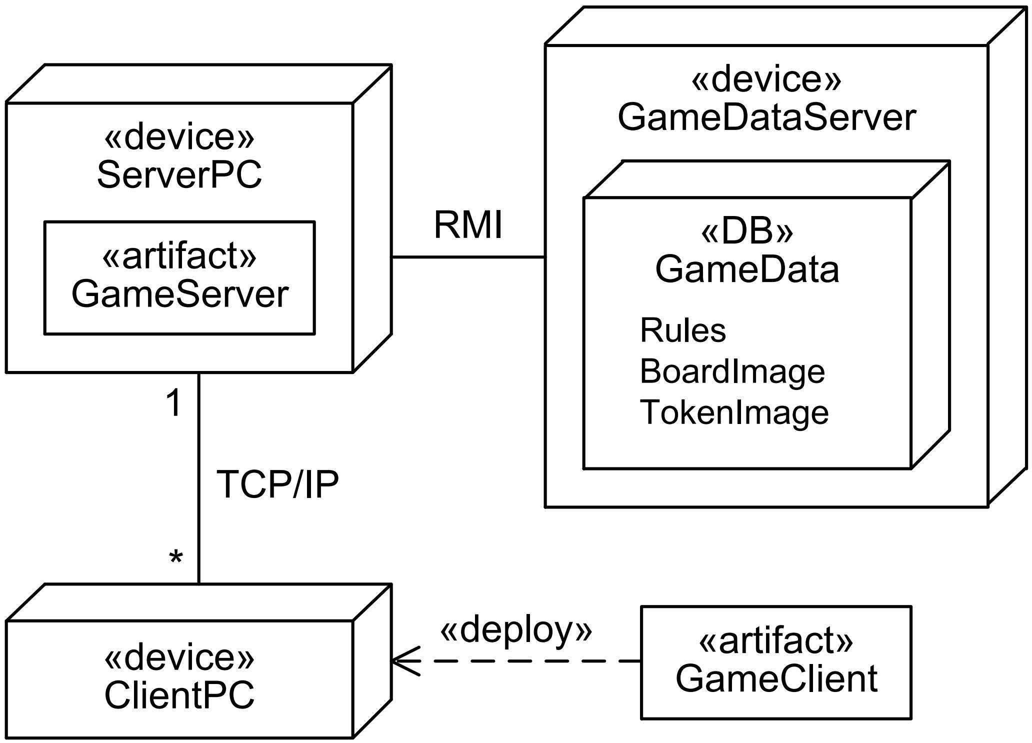
Deployment Diagrams

- A UML **deployment diagram** models computational resources, communication paths among them, and artifacts that reside and execute on them.
- Used to show
  - Real and virtual machines used in a system
  - Communication paths between machines
  - Program and data files realizing the system
    - Residence
    - Execution

Deployment Diagram Rules

- Computational resources are nodes
- Communication paths are solid lines between nodes
  - May be labeled
  - May have multiplicities and role names
- Artifact symbols may
  - Appear within node symbols
  - Be listed within node symbols
  - Be connected to node symbols by dependency arrows stereotyped with «deploy»

Deployment Diagram Example



Summary

- So far – for architectural design
  - concepts involved in the design (last lecture)
  - notations
- Next –
  - generation
  - evaluation
  - improvement and selection of software architectures
  - Finalizing - Reviews

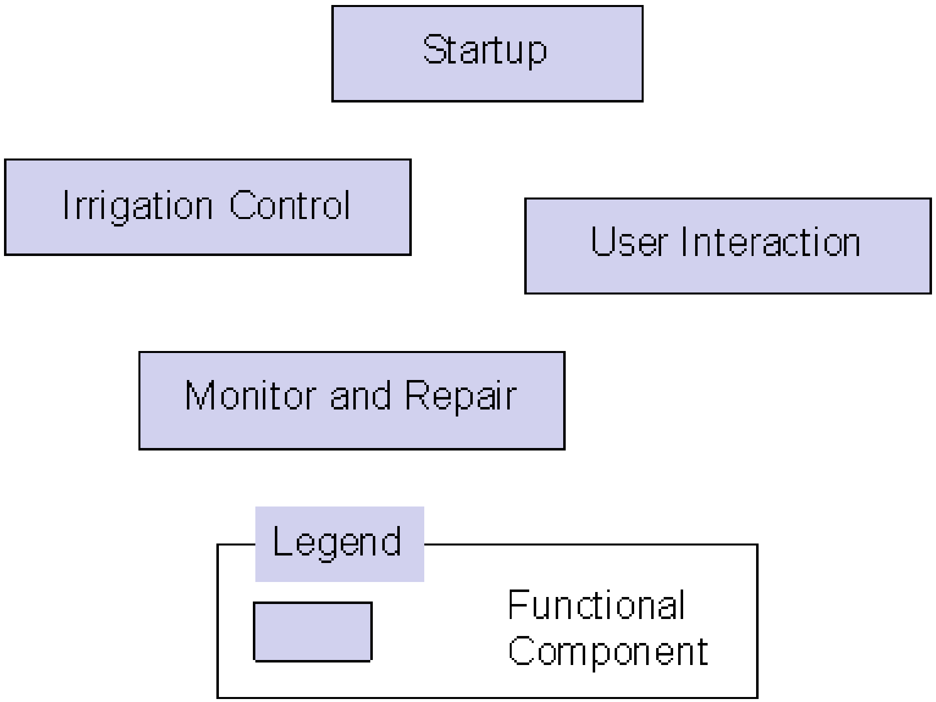
Architectural Design - Generation

- *Determine Functional Components*—Create components responsible for realizing coherent collections of functional and data requirements.
- *Determine Components Based on Quality Attributes*—Form components to meet non-functional requirements, then add components to fill functional and data requirements gaps.

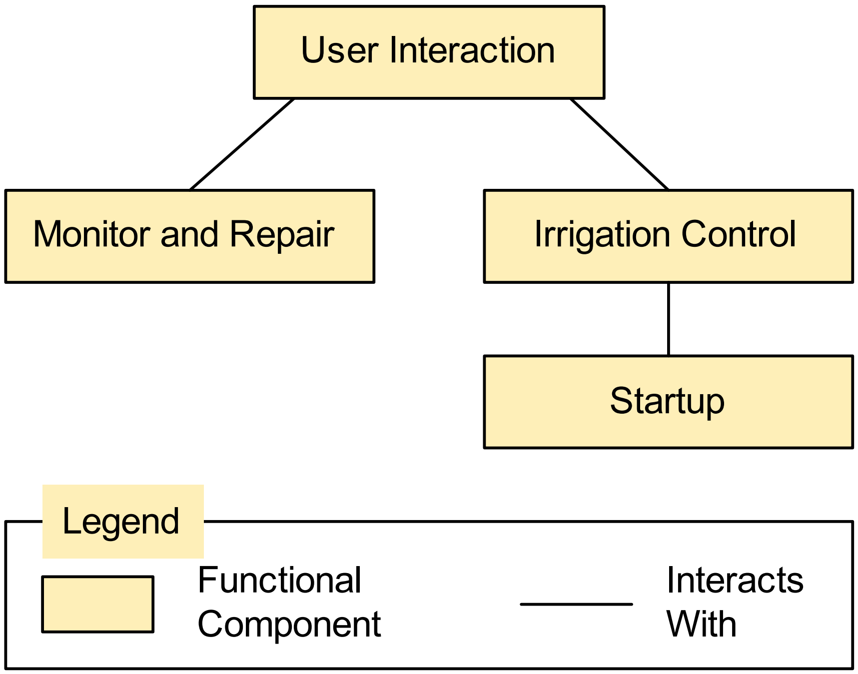
Architectural Design – Generation..

- *Modify an Existing Architecture*—Alter an architecture for a similar program.
- *Elaborate an Architectural Style*—An **architectural style** is a paradigm of program or system constituent types and their interactions (more on this later). Elaborate a style to form an architecture.
- *Transform a Conceptual Model*—Modify a conceptual model from a problem to a solution description.

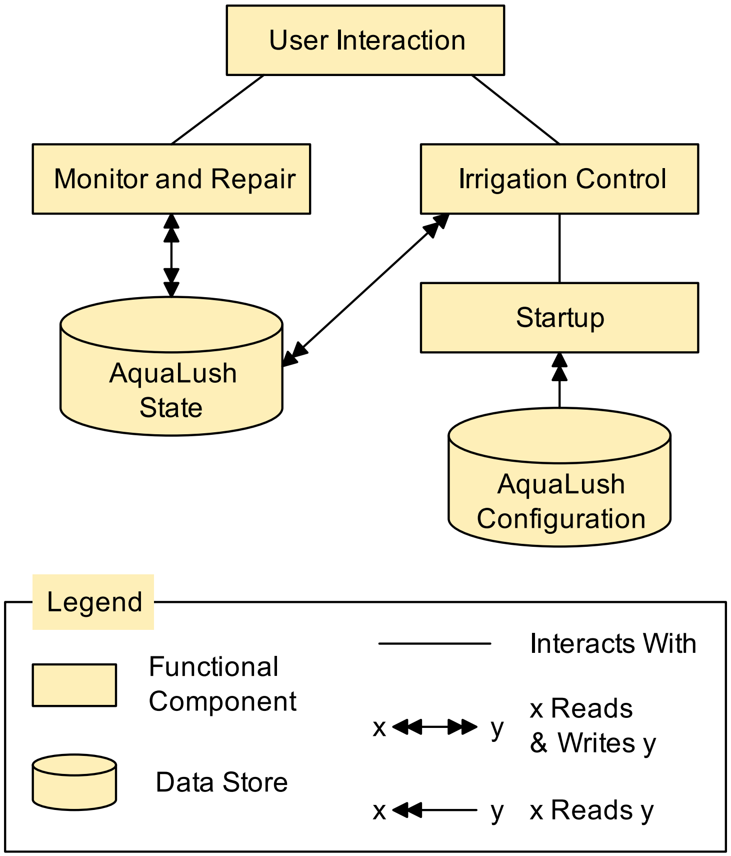
Example - Functional Decomposition (Irrigator) (Draft 1)



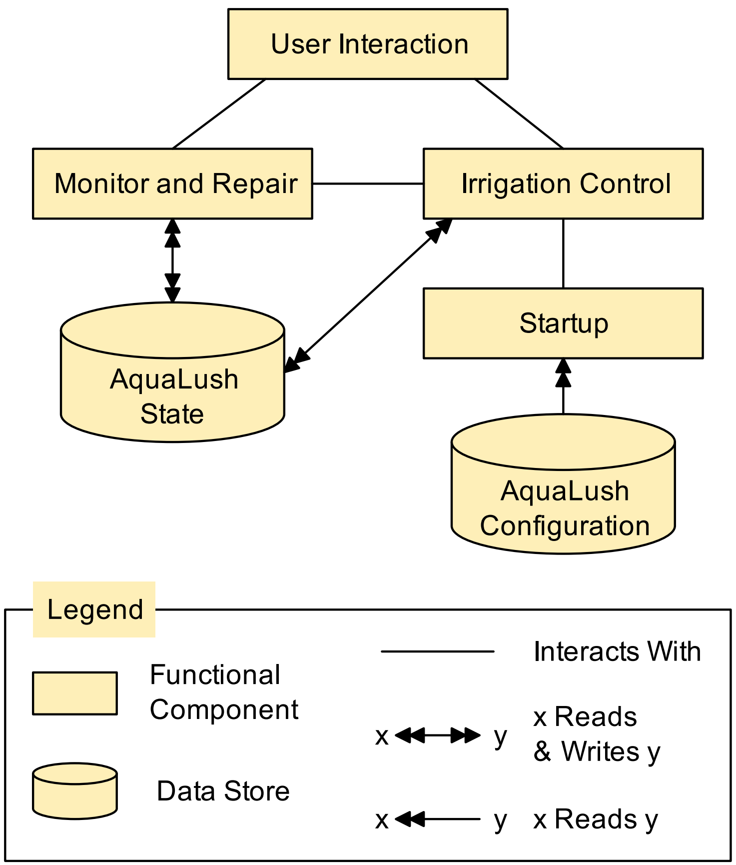
Example - Functional Decomposition (Draft 2)



Example - Functional Decomposition (Draft 3)



Example - Functional Decomposition (Draft 4)



Improving Alternatives

- *Combine Alternatives*—Combine the best features of two or more alternatives
- *Impose an Architectural Style*—Modify an architecture that almost fits a style so that it does fit the style
- *Apply Design Patterns*—Modify an architecture to take advantage of known design patterns

Evaluating Alternatives

- How can designers determine whether a program built to an architectural specification will satisfy its requirements before the program is built?
- No one knows how to guarantee this, but several techniques make it more likely.
- We examine the use of scenarios and prototypes for evaluation.

Scenarios

A **scenario** is an interaction between a product and particular individuals.

- Use case instances are interactions between and a product and actors
- Broader view because now we consider interactions between a product and any individual

Profiles

A **profile** is a set of scenarios used to evaluate whether a product is likely to meet a set of requirements.

- Examples: usage profile, reliability profile
- Scenarios in profiles should have weights
- Profiles are formed by choosing 3 to 10 representative scenarios from all those that fit a profile

Creating Profiles and Scenarios

- A **utility tree** is a tree whose sub-trees are profiles and whose leaves are scenarios.
  - Label the root "utility."
  - Add children with profile names that reflect product requirements.
  - Fill in scenarios for each profile.
    - Brainstorm scenarios
    - Rationalize the list
    - Weight each scenario
    - Eliminate low-weight scenarios until each profile has 3 to 10 leaves
  - Write scenario descriptions.

Example Utility Tree

- Utility
  - Usage
    - Irrigate Automatically (H)
    - Irrigate Manually (M)
    - Repair During Irrigation (L)
    - Configure at Startup (H)
    - Simulate AquaLush (M)
  - Hardware Adaptability
    - Add a Valve Type (H)
    - Replace the Keypad (L)
  - Modifiability
    - Add Timer-Based Irrigation (M)
    - Modify Irrigation Process (M)
  - Reusability
    - In an Agricultural Product (L)
    - In a Consumer Product (M)
  - Reliability
    - Power Fails (H)
    - A Sensor Fails (H)

Evaluating and Selecting with Scenarios

- Walk through each scenario.
  - Judge how well a design alternative supports the scenario.
  - Record a judgment for each scenario.
- Use a selection technique to choose an alternative.
  - Pros and cons
  - Multi-dimensional ranking
    - Scenarios weights are normalized
    - Judgments are quantified

Evaluating and Selecting with Prototypes

- Prototypes may be built to test out design alternatives.
- Scenario walkthroughs may give rise to a need for prototyping.
- Prototypes provide the factual basis for selection using
  - Pros and cons;
  - Multi-dimensional ranking.

Summary

- Several complimentary techniques can be used to generate and improve architectural alternatives.
- Building profiles consisting of weighted scenarios and walking through them is a solid technique for evaluating architectural alternatives.
- Prototypes can also supply data for architectural evaluation.

Finalizing Architectural Design - Reviews

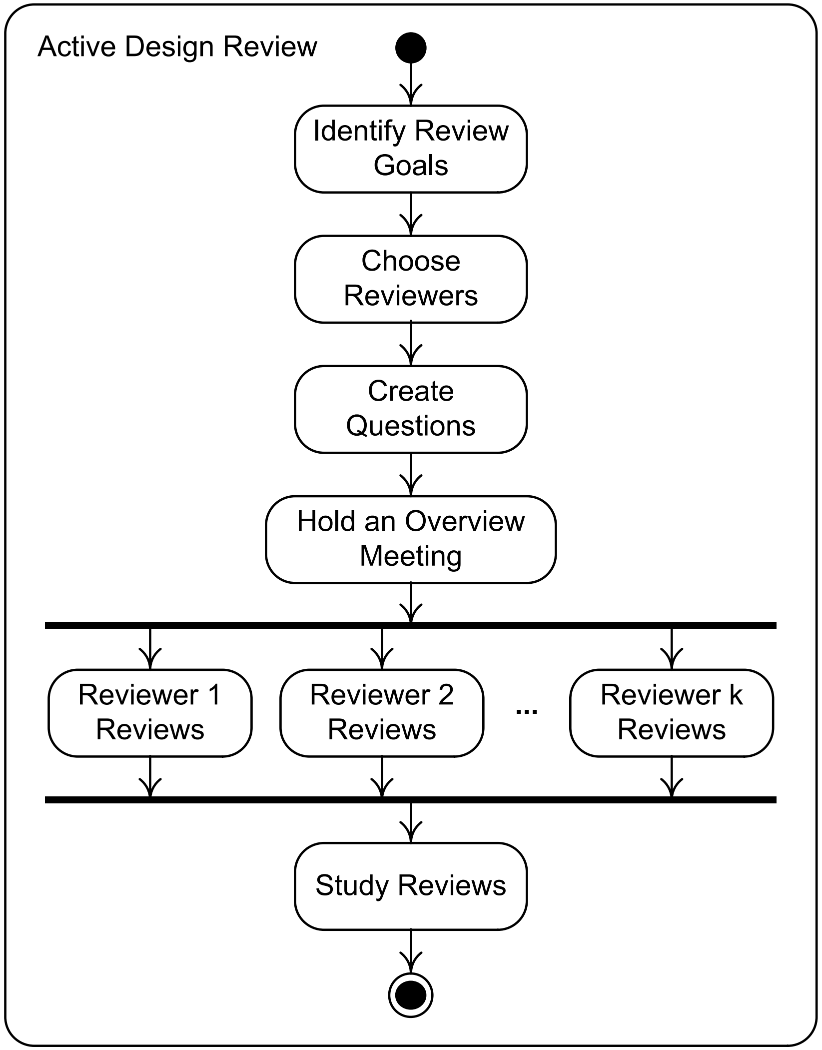
A **review** is an examination and evaluation of a work product or process by a team of qualified individuals.

- *Desk Check*—An assessment of a design by the designer
- *Walkthrough*—An informal presentation to a team of designers
- *Inspection*—A formal review by a trained inspection team
- *Audit*—A review conducted by experts from outside the design team
- *Active Review*—An examination by experts who answer specific questions about the design

Active Design Reviews

- Remedies problems with traditional reviews
  - Lack of expertise
  - Cursory reviews
- Forces reviewers to engage the document in their areas of expertise by asking them to answer specific questions about design details

Active Design Review Process



Review Preparation

- *Identify Review Goals*—Designers choose aspects of the design they want checked.
- *Choose Reviewers*—Designers identify two to four qualified reviewers and obtain their consent to do the review.
- *Create Questions*—Designers formulate questions to be answered by reviewers.
  - Force reviewers to understand the design
  - Ask reviewers to solve problems, explain something, etc.

Review Performance

- *Hold an Overview Meeting*—Designers sketch the architecture, explain the process, set deadlines, etc.
- *Do Reviews*—The reviewers do their reviews on their own.
  - May meet with designers are send emails to get clarification, explanations, etc.
  - Deliver their results when complete

Review Completion

- *Study Reviews*—Designers study the review results.
  - May meet with reviewers or email questions