Software Architecture

Jesper Andersson

Linnæus University

Software Design

- ✓ Characteristics
  - Complex systems
  - Invisible
  - Group activity
  - Always the “first time”
- ✓ Problems
  - Align work
  - Decompose for decentralization
  - Coherence
  - Conformance

Design

"To form a plan or scheme of, to arrange or conceive in the mind…for later execution."

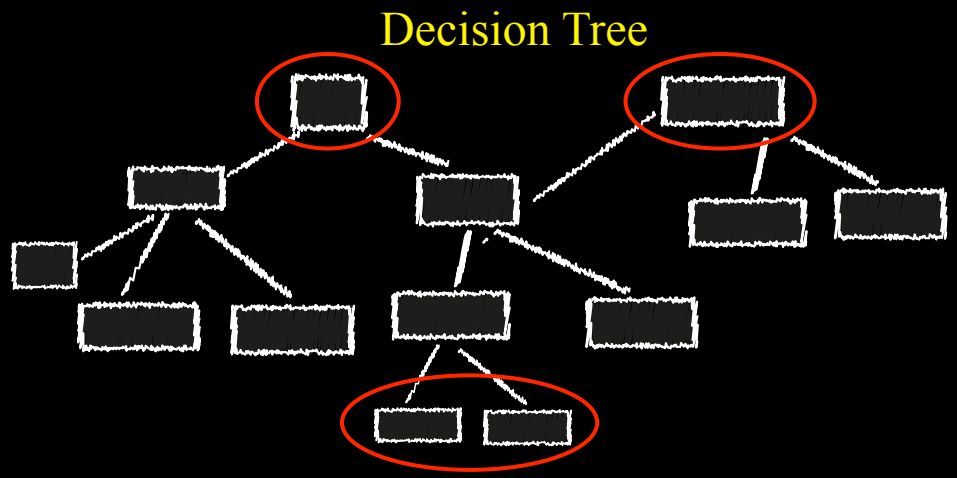
Oxford English Dictionary

Software Design

- ✓ Design Decisions
- ✓ Some Decisions are **Global**
- ✓ Some are **Local**

- ✓ Global Decisions impact Local
- ✓ We must decide on Global first.

- ✓ We need something to Reason about Global Decisions!



Software Architecture

- ✓ System decomposition
  - How is the system divided into subsystems?
  - Do we have all the parts?
  - Will the parts fit together?
- ✓ System concerns
  - Functionality
  - System characteristics and qualities
  - System quality trade-offs
- ✓ Engineering Practices
- ✓ **System Integrity**

Architecture is not a phase of development!

- ✓ The **principal designs decisions** regarding its design
- ✓ The **key abstractions** that characterize the application
- ✓ The **structuring** of these key abstractions and principal **interaction mechanisms**

So…

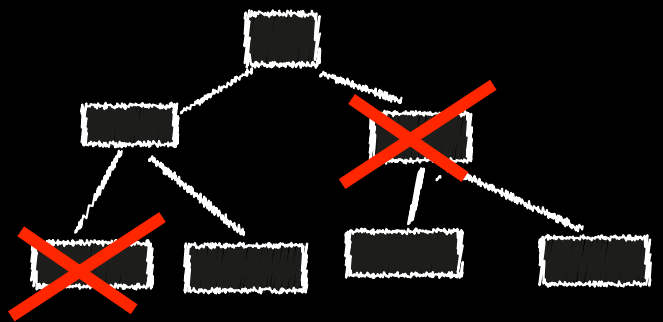
- ✓ During development we will face a number of strategic decisions

strategic *or* **strategical** (strəˈtiːdʒɪk) — ***adj***

Important or essential in relation to a plan of action
Highly important to an intended objective

- ✓ To create a system with the expected
  - Functionality
  - Quality
- ✓ Thus **we need** sufficient tooling to be able to discuss and reason about these complex matters

Architecture - Decomposition



- ✓ A system consists of subsystems, which contain subsystems, which can be further decomposed in subsystems …
- ✓ This is true for all systems, biological as well as artificial
- ✓ So problem solving is all about
  1. Finding the best "pieces"
  2. Join them together

Or?

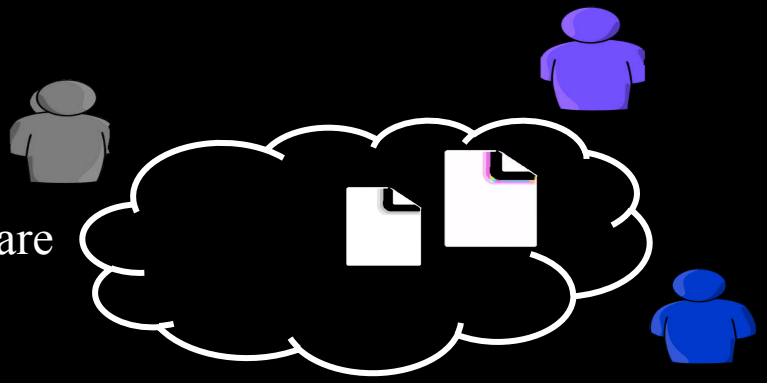
- ✓ Will the pieces fit?

System integrity

in·teg·ri·ty noun \in-ˈte-grə-tē\
: the quality of being honest and fair
: the state of being complete or whole
1: firm adherence to a code of especially moral or artistic values : incorruptibility
2: the quality or state of being complete or undivided : completeness

- ✓ The most important (critical) system characteristics first.
- ✓ We can not achieve a cohesive system bottom up
  - Perform required trade offs during decomposition to make sure the system satisfies its goals.
  - Define the architectural mechanisms that cross-cut the system and manage the systems characteristics, internal as well as external.

An example



- ✓ Let's study an application where "users share documents"
- ✓ What is most important?
- ✓ Requirements
  - *Functional*, "share files"
  - *Quality*, performance, security, reliability

Some Decision Examples

- ✓ What does the system's module decomposition look like?
- ✓ Is there a reference application architecture we may use?
- ✓ How will the system be distributed?
- ✓ How will the system address concerns X and Y?

Architectural design decisions

- ✓ Architectural design is an intellectual creative process.
- ✓ Most architectural design processes are generic. No guidance on which decisions you should make
- ✓ Experience and domain based

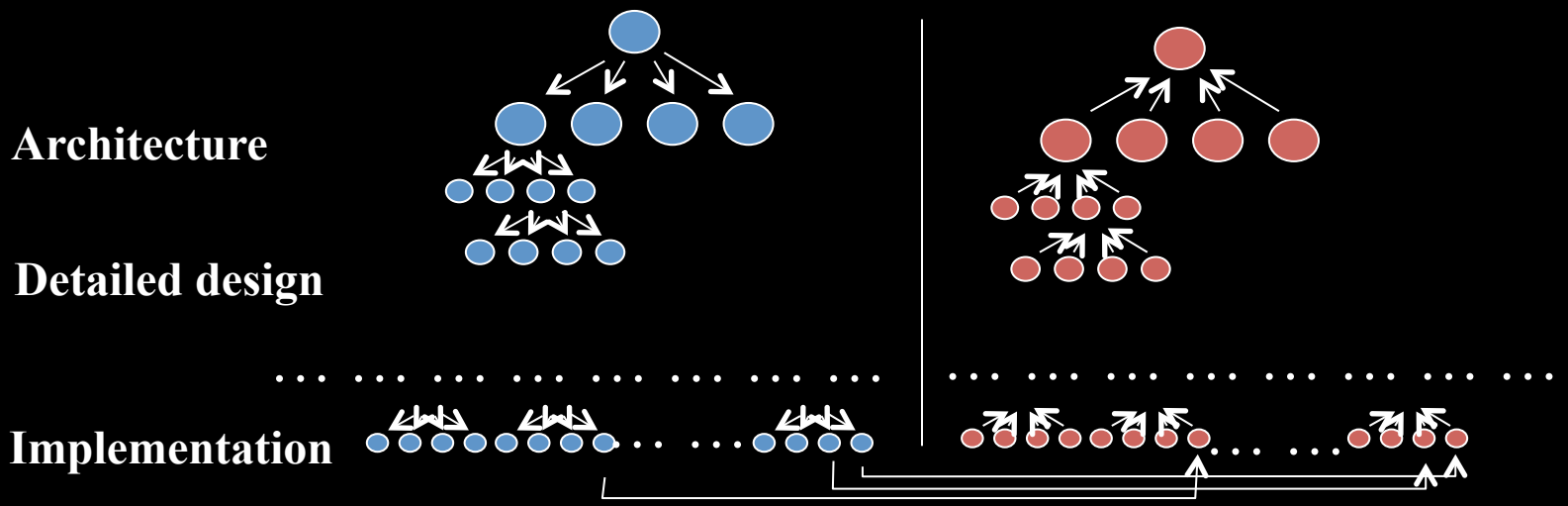
- ✓ **It is important to document your decisions and their rationale**

Design @ different levels

- ✓ Software Design – “To form a plan or scheme of, to arrange or conceive in the mind…for later execution.”

- ✓ Design decision: An explicit refinement of a program design. Each decision reflects the elaboration of a design or programming plan into a more concrete realization.
  - Locality (impact)
  - Generality (abstraction level)

Design @ different levels – Divide n' Conquer



A Decision's Impact on System – Locality

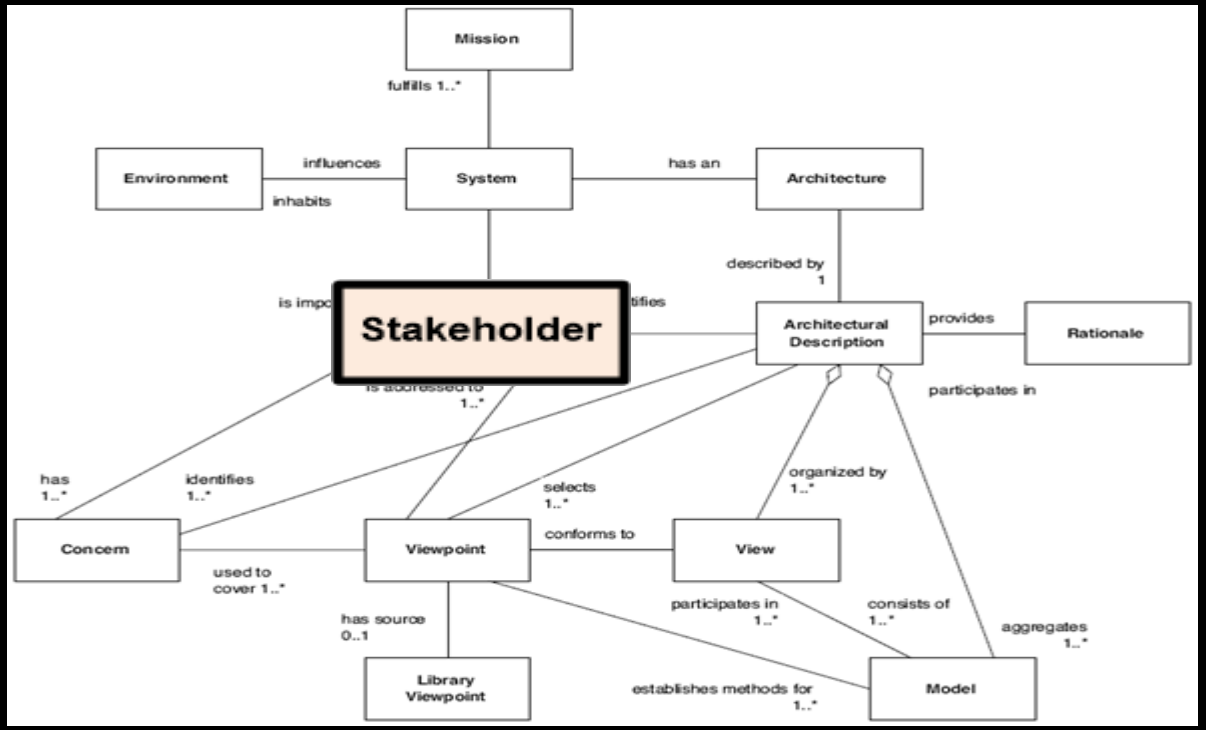
- ✓ What is the impact of a decision on the system?
  - Global – affects the entire or a large part of a system
  - Local – affects a single entity or small part of a system

- ✓ Can be difficult to define when **Local** becomes **Global** and vice versa.

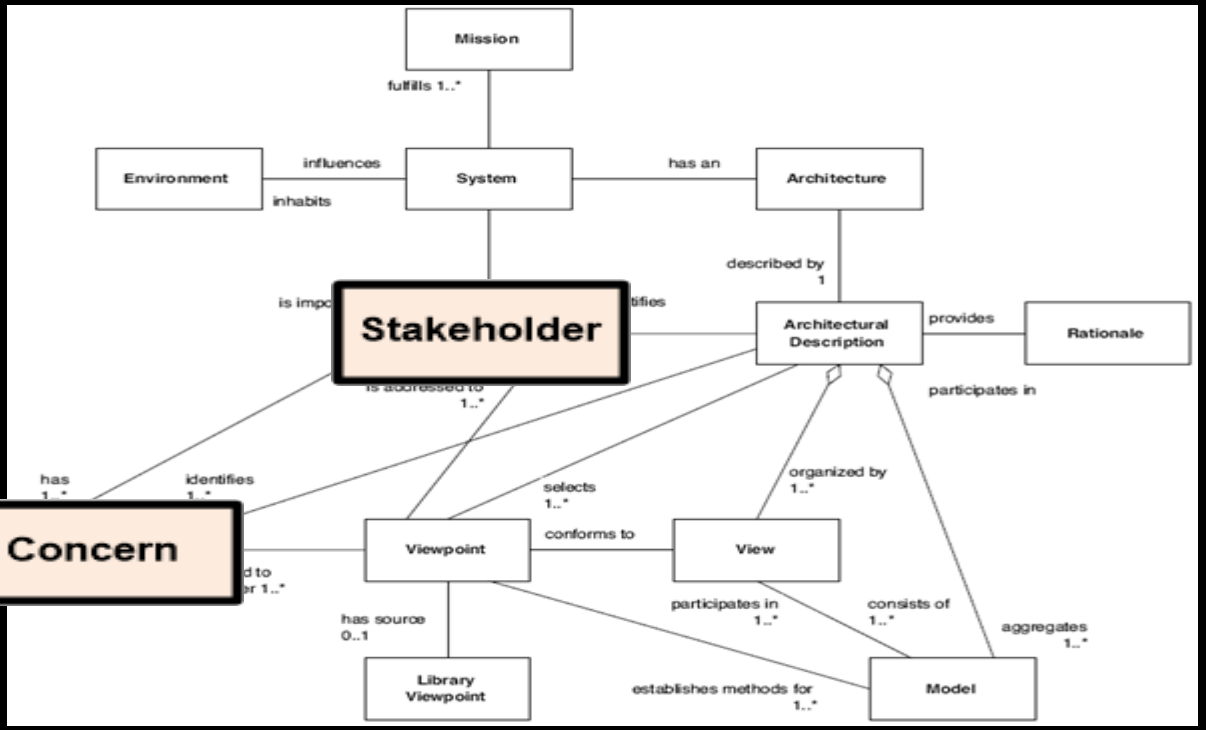
Example Global – System-wide Decisions

- ✓ Implementation technology – Prescribes an architecture structure
- ✓ Decomposition into components
- ✓ Component organization – Architecture patterns
- ✓ Principles for Quality Concerns,
  - e.g., **security ➔**
    - Authentication
    - Session management
    - User management

ISO 42010


Mission
fulfills 1..*
Environment
influences
inhabits
System
has an
Architecture
described by
1
Stakeholder
Architectural
Description
provides
Rationale
participates in
has
1..*
identifies
1..*
selects
1..*
organized by
1..*
Concern
Viewpoint
conforms to
View
used to
cover 1..*
has source
0..1
participates in
1..*
consists of
1..*
aggregates
1..*
Library
Viewpoint
establishes methods for
1..*
Model

ISO 42010


Mission
fulfills 1..*
Environment
influences
inhabits
System
has an
Architecture
described by
1
Stakeholder
Architectural
Description
provides
Rationale
participates in
has
identifies
selects
1..*
organized by
1..*
Concern
Viewpoint
conforms to
View
has source
0..1
participates in
1..*
consists of
1..*
aggregates
1..*
Library
Viewpoint
establishes methods for
1..*
Model

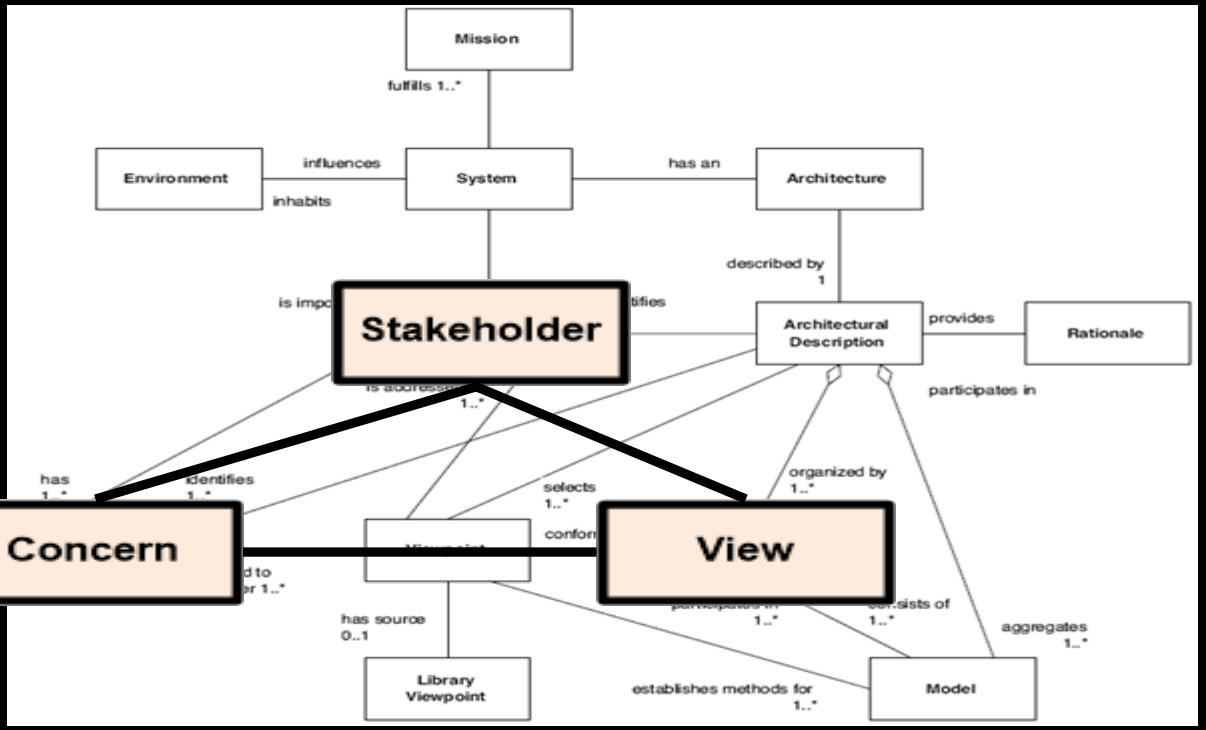
ISO 42010 – Stakeholders and Concerns

- ✓ Stakeholders of a system have concerns with respect to the system-of-interest considered in relation to its environment.
- ✓ A concern could be held by one or more stakeholders.
- ✓ Concerns arise throughout the life cycle from system needs and requirements, from design choices and from implementation and operating considerations.
- ✓ A concern could be manifest in many forms, such as in relation to one or more stakeholder needs, goals, expectations, responsibilities, requirements, design constraints, assumptions, dependencies, quality attributes, architecture decisions, risks or other issues pertaining to the system.

Concerns – Examples from the ISO 42010

functionality, feasibility, usage, system purposes, system features, system properties, known limitations, structure, behavior, performance, resource utilization, reliability, security, information assurance, complexity, evolvability, openness, concurrency, autonomy, cost, schedule, quality of service, flexibility, agility, modifiability, modularity, control, inter-process communication, deadlock, state change, subsystem integration, data accessibility, privacy, compliance to regulation, assurance, business goals and strategies, customer experience, maintainability, affordability and disposability.

ISO 42010

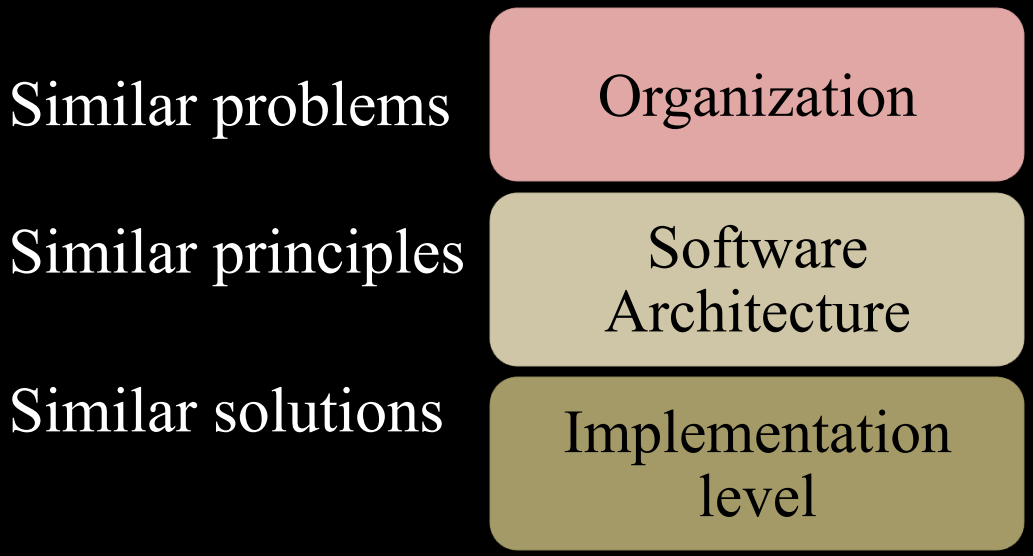

Mission
fulfills 1..*
Environment
influences
inhabits
System
has an
Architecture
described by
1
Stakeholder
Architectural Description
provides
Rationale
participates in
has
identifies
selects
1..*
organized by
1..*
Concern
View
has source
0..1
Library Viewpoint
aggregates
1..*
establishes methods for
1..*
Model

Have a closer look at the Concerns

functionality, feasibility, usage, system purposes, system features, system properties, known limitations, structure, behavior, performance, resource utilization, reliability, security, **information assurance**, complexity, evolvability, openness, concurrency, autonomy, cost, schedule, quality of service, flexibility, agility, modifiability, modularity, control, inter-process communication, **deadlock**, state change, subsystem integration, data accessibility, privacy, compliance to regulation, assurance, **business goals and strategies**, customer experience, maintainability, affordability and disposability.

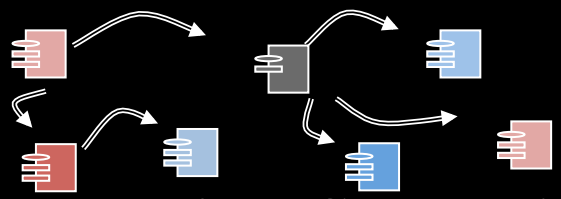
They seem to be all over the place!!!

System and Software Architecture



Programming level

- ✓ Procedures, functions, objects provide functionality used by other procedures, functions and objects
- ✓ Example Object Oriented Languages
  - Objects, concept that creates logically grouped data and behavior
  - Messages, concept that is an abstraction of behavior usage



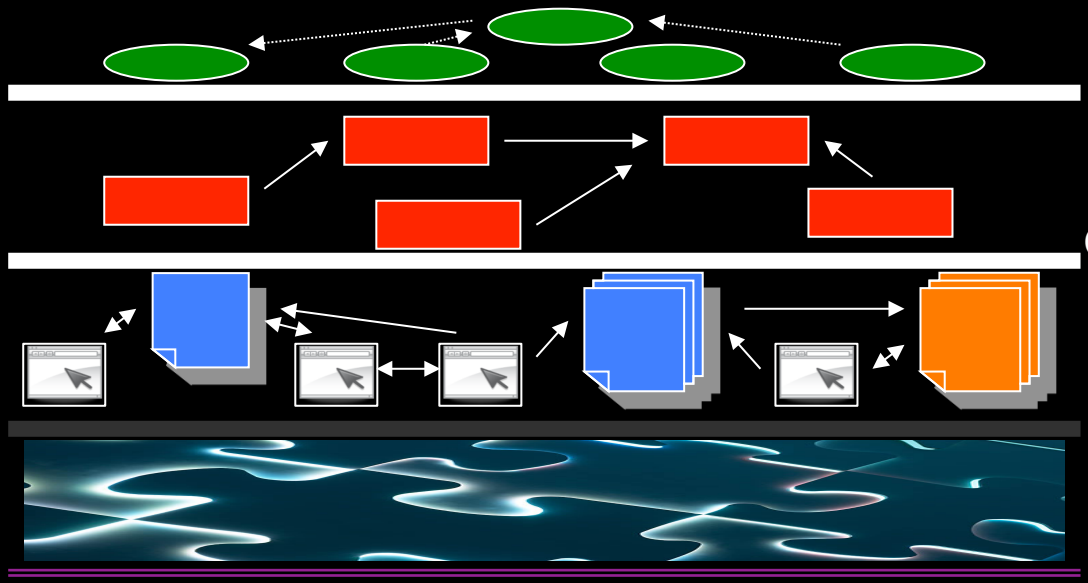
It's a System!

Organization level

"An organization is a collectivity with a relatively identifiable boundary, a normative order (rules), ranks of authority (hierarchy), communications system, and membership coordinating system (procedures); this collectivity exists, on a relatively continuous basis in an environment, and engages in activities that are usually related to a set of goals; the activities have outcomes for organizational members, the organization itself, and for society."

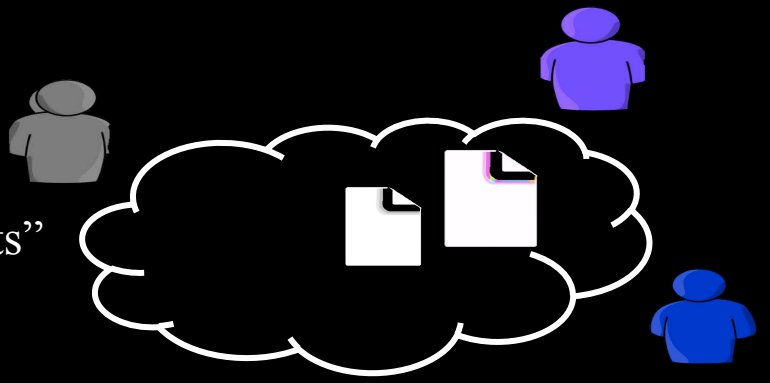
It's a System!

Enterprise Architecture!



Decisions mitigate Coordination Risks!

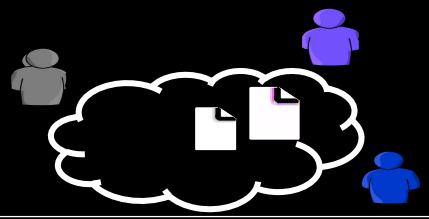
Back to the Example



- ✓ An application where ”users share documents”
- ✓ Requirements
  - *Functional*, “share files”
  - *Quality*, performance, security, reliability

How should we decompose?

- ✓ System decomposition
  - How should it be decomposed?
  - Are all required parts defined?
  - Will they fit?
- ✓ System concerns
  - System properties and qualities
  - System quality trade offs
- ✓ System integrity!



Functionality

1. Identify subsystems and **responsibilities**
2. Identify subsystem **interfaces**

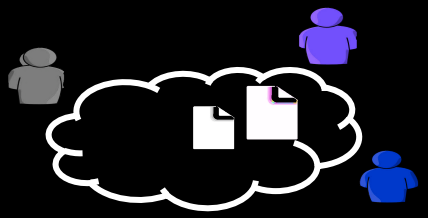
Quality

1. Identify **architectural mechanisms**
2. Allocate **responsibilities** to subsystems and interfaces

Decision Making – Architectural Reasoning

- ✓ Architectural Reasoning
- ✓ Engineering practices
  - If **alternatives** exist, engineers **compare** design alternatives and **choose** the solution that **best** matches the requirements.
  - Engineering should be **predictive**, i.e., engineers attempt to predict how well their designs will perform to their specifications prior to full-scale production.

Functionality

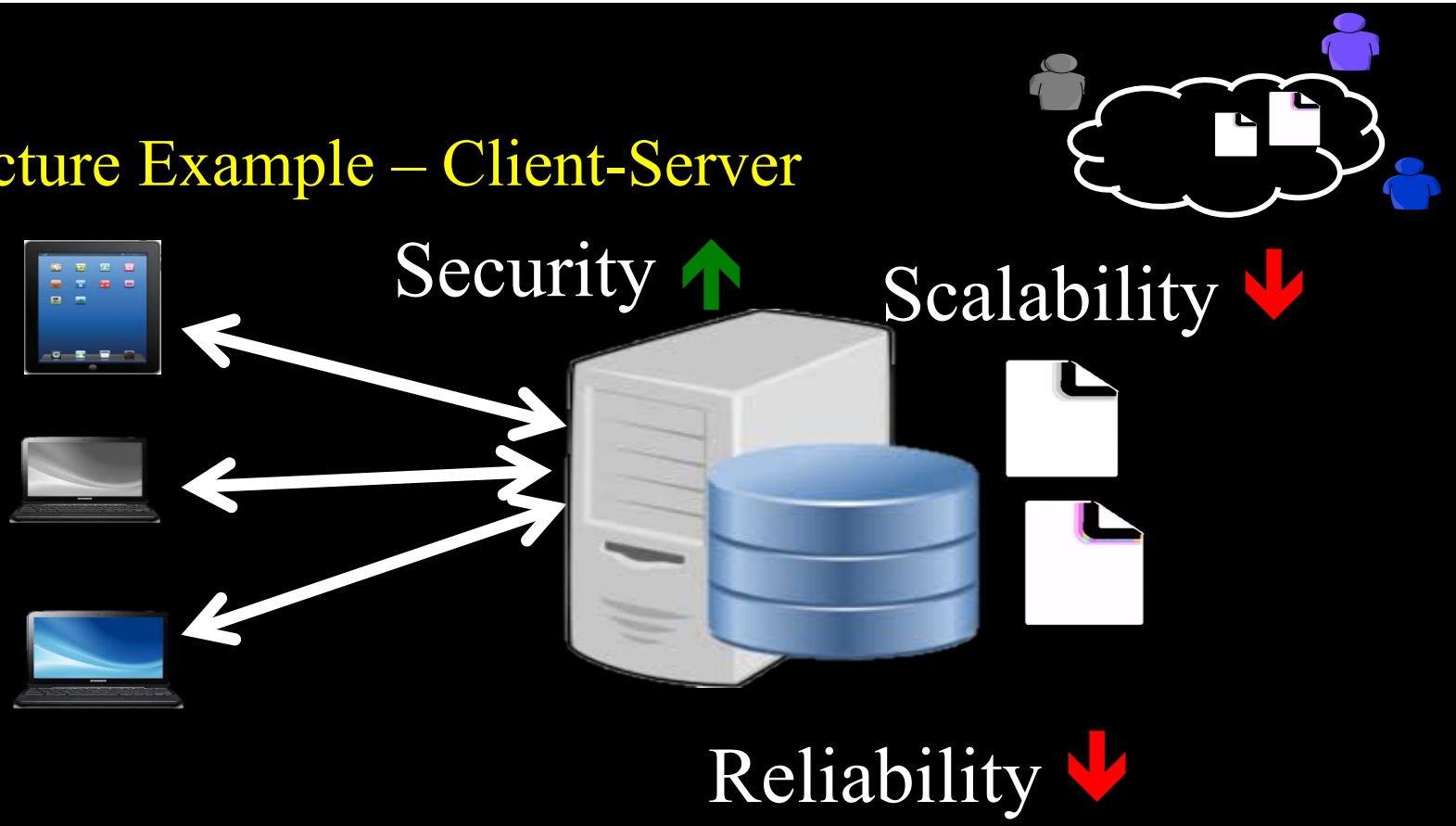


- ✓ The use cases give us the functional requirements
- ✓ Purpose:
  - Identify subsystems
  - Allocate sufficient responsibility to the subsystems so that they can realize the functionality as described in the use case.
- ✓ Use scenarios to verify that sufficient responsibilities have been allocated.

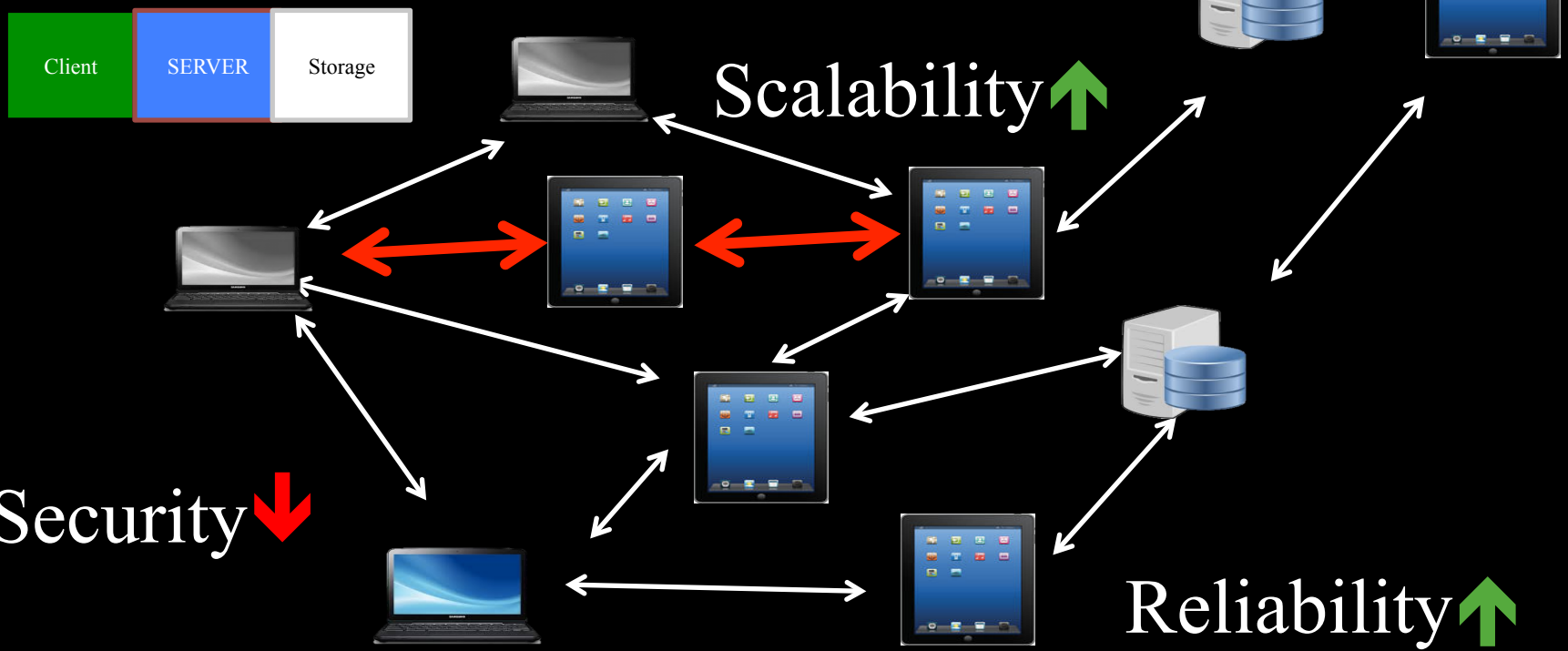
Quality

- ✓ “Cross-cutting”, a system quality is everywhere in a system not in specific parts.
- ✓ You can use two strategies for your architecture mechanisms, that realize system quality
  1. ***Structure***, some structural principles (patters) give specific characteristics.
  2. ***Behavior***, extra functionality added to the system

Structure Example – Client-Server

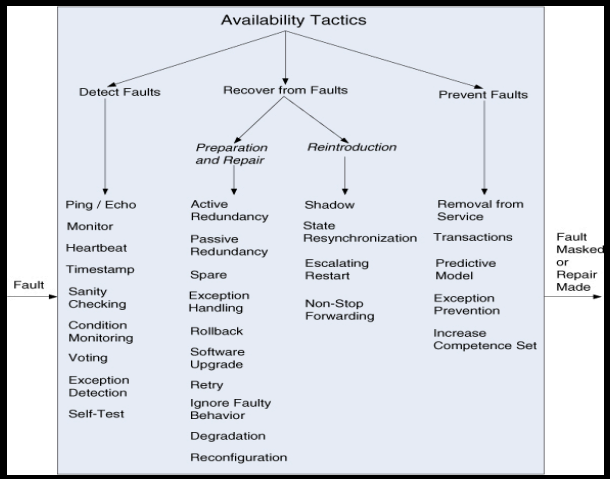


Structure example – Peer-2-Peer



Behavior Example – Tactics

- ✓ Quality as **functionality** in the software
- ✓ Examples
  - Performance through load balancing
  - Reliability by fault recovery
- ✓ We also have tactics that affect the software structure
- ✓ Examples
  - Increase modifiability
  - Increase reusability



Jed’s Rental - Example

Our client is Jed's RentAll, Inc., a local equipment rental company. Jed Braun, the owner, has two stores with an inventory of over 6000 rental items, including everything from picnic tables to portable generators. Jed rents to individuals and also has a lot of repeat business from contractors and other companies.

Jed wants a combined rental agreement, inventory tracking and sales/rental management system. Jed has a number of employees that work on the phone or counter to arrange rentals. They also do a small amount of equipment sales. Each inventory item has a rental rate by the hour, day, week and month. Generally, rates are determined by equipment type, but the sales staff can over-ride this part of the contract, for good customers and variations in the situation.

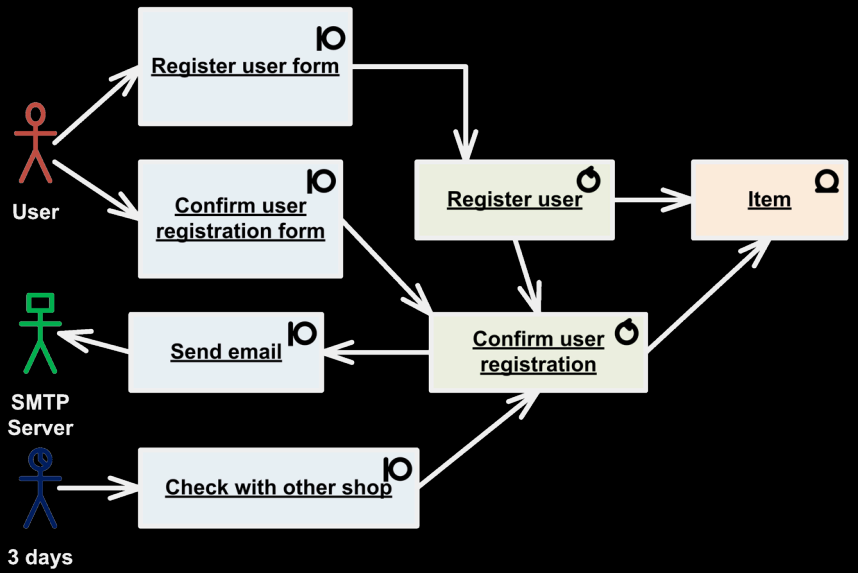
What type of Decisions

- ✓ Functional decomposition
- ✓ Security
- ✓ Persistency
- ✓ Distribution
- ✓ Deployment targets
- ✓ Robustness
- ✓ Security

1st level Decomposition

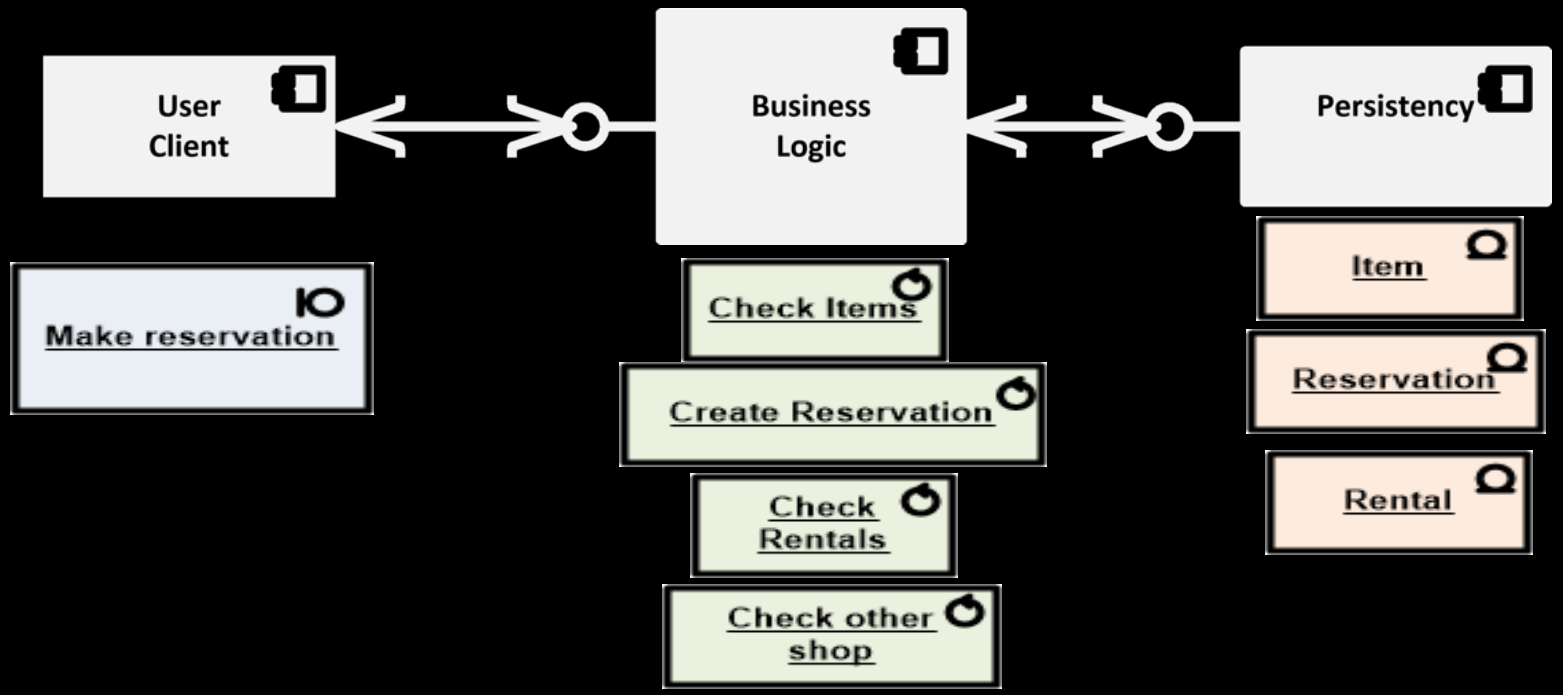
- ✓ Top-Level
- ✓ Define the Interfaces
- ✓ The work on each Component Separately
- ✓ Problems (Concerns)
  - Distribution
  - Authentication
  - Robustness

1st level Decomposition – Functional Responsibilities

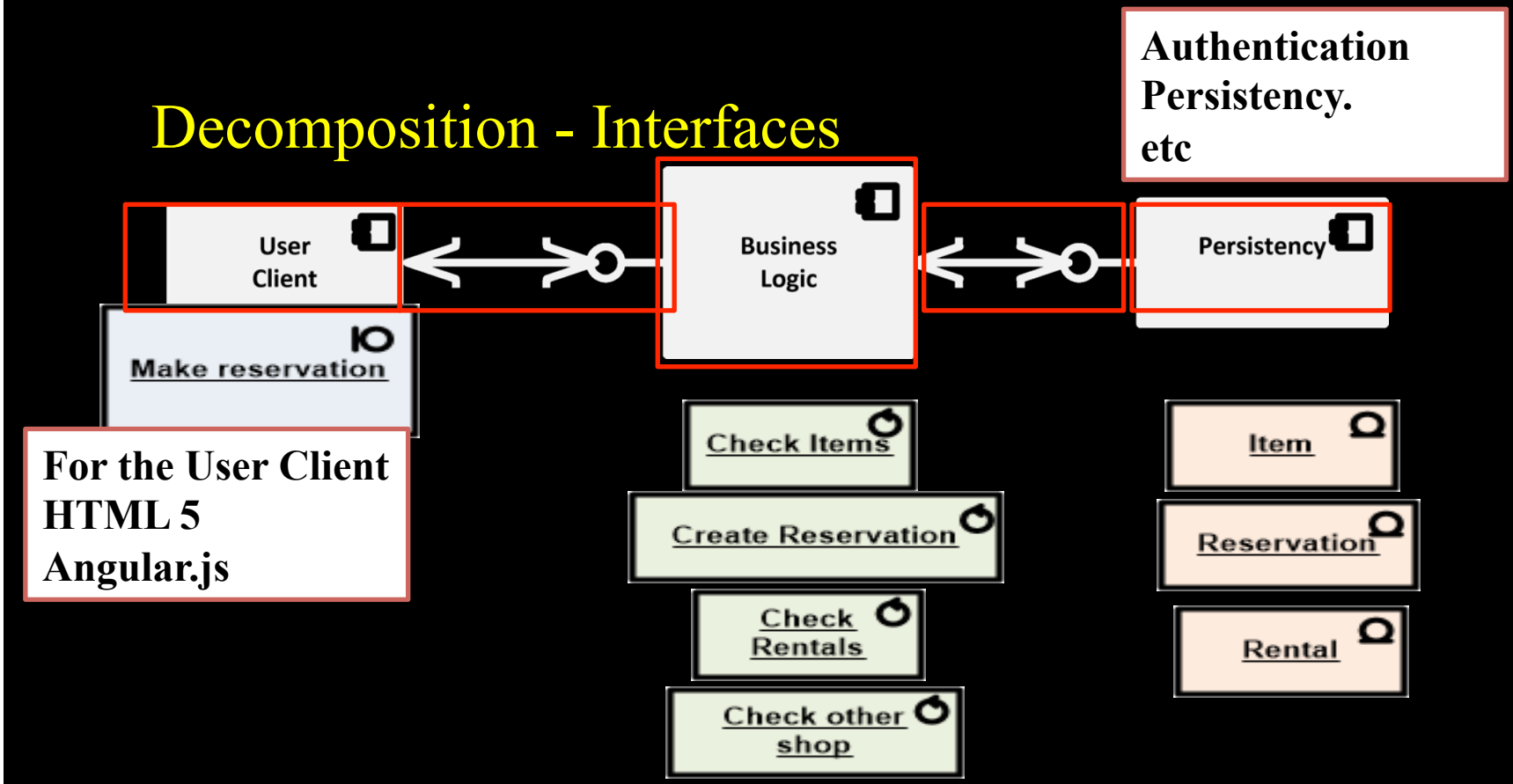


- ✓ Robustness Diagrams to Analyze Functional Requirements
- ✓ Additional specs, standards, design constraints, etc.

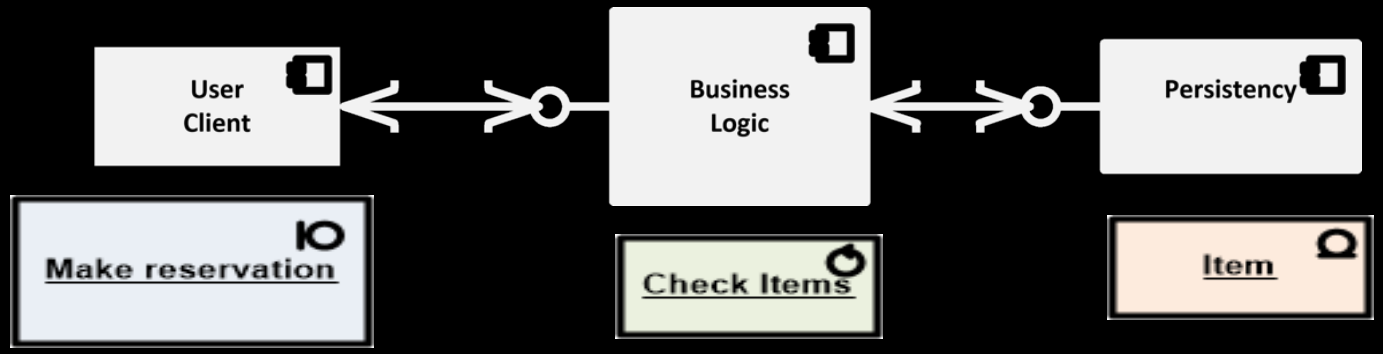
1st level Decomposition - Interfaces



Decomposition - Interfaces

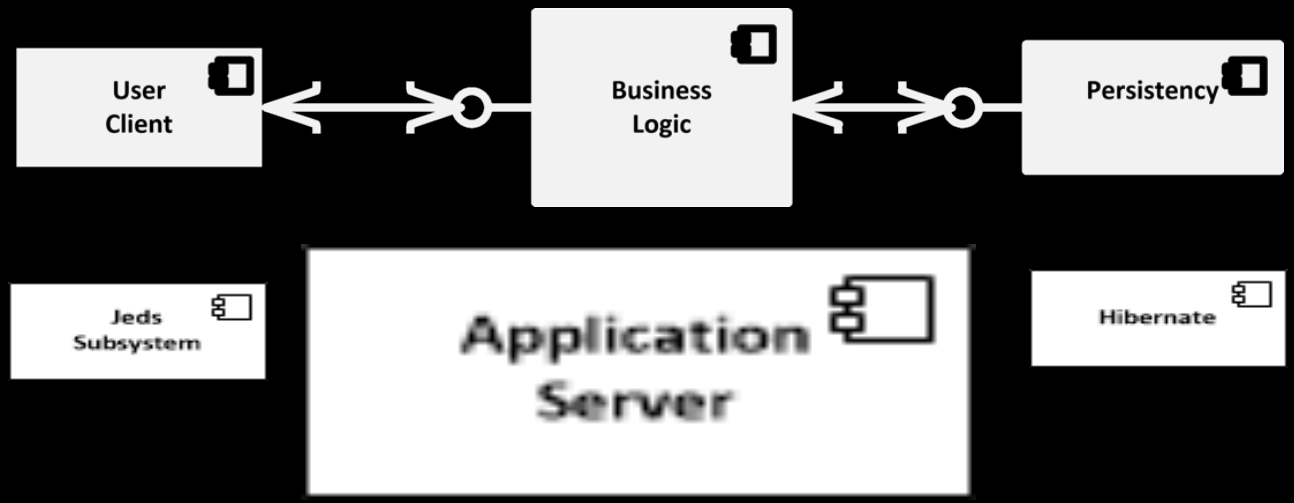


Decomposition – Design for a UC/Scenario

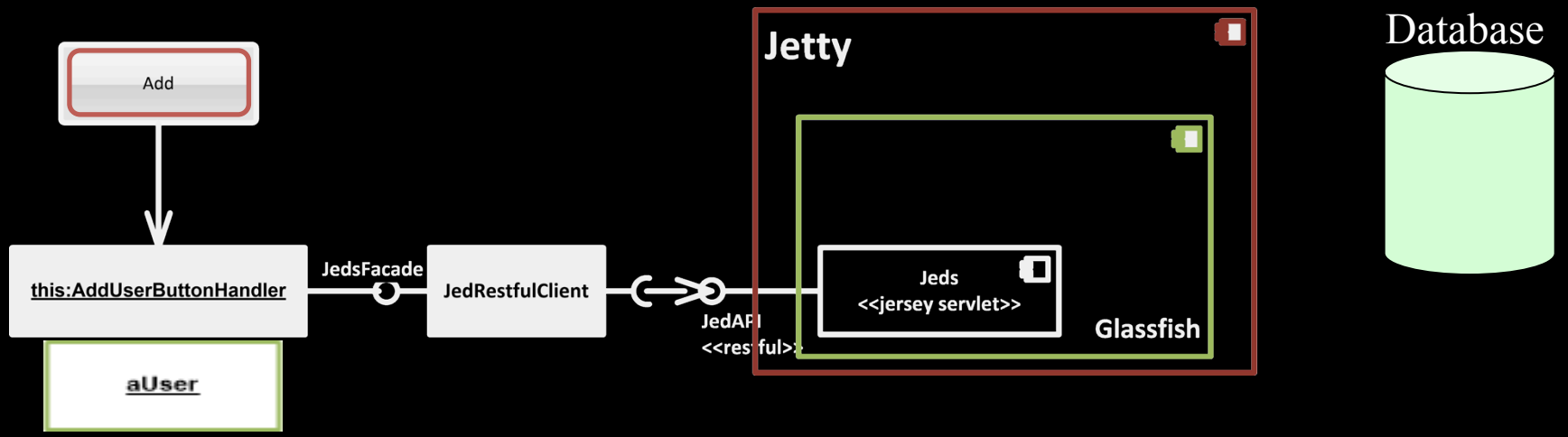


- ✓ Pick a UC/Scenario/Set of Scenarios Design Interfaces Boundary, Control and Entity classes
- ✓ Architectural mechanisms (authentication, persistency, etc.)
- ✓ Considering implementation technology.

Subsystems


User Client
Business Logic
Persistency
Jeds Subsystem
Application Server
Hibernate

How this will work



Today's takeaways Software Architecture

- ✓ What
  - The *infrastructure* of an application
  - Hosts functionality, *distribution of functional responsibilities*.
  - Promotes quality, structuring principles, patterns
- ✓ Why
  - Reasoning about system properties
  - Early design decisions, cross-cutting

Today's Take-away