Software Architecture: Introduction

2II45
Fall 2009

Background

- Tom Verhoeff, Mark van den Brand, Alexander Serebrenik, Lou Somers
- SET = Software Engineering & Technology
- LaQuSo
- www.win.tue.nl/set
- www.win.tue.nl/~wstomv/edu/2ii45

LaQuSo Laboratory for Quality Software

You Are Expected to:

- Read literature (see last slides)
- Do small homework assignment(s)
- Write essay (more information next week)
  - in couples
- Take written exam (1.5h in January)
  - There is also 1.5h retry of Block 1 in Jan.

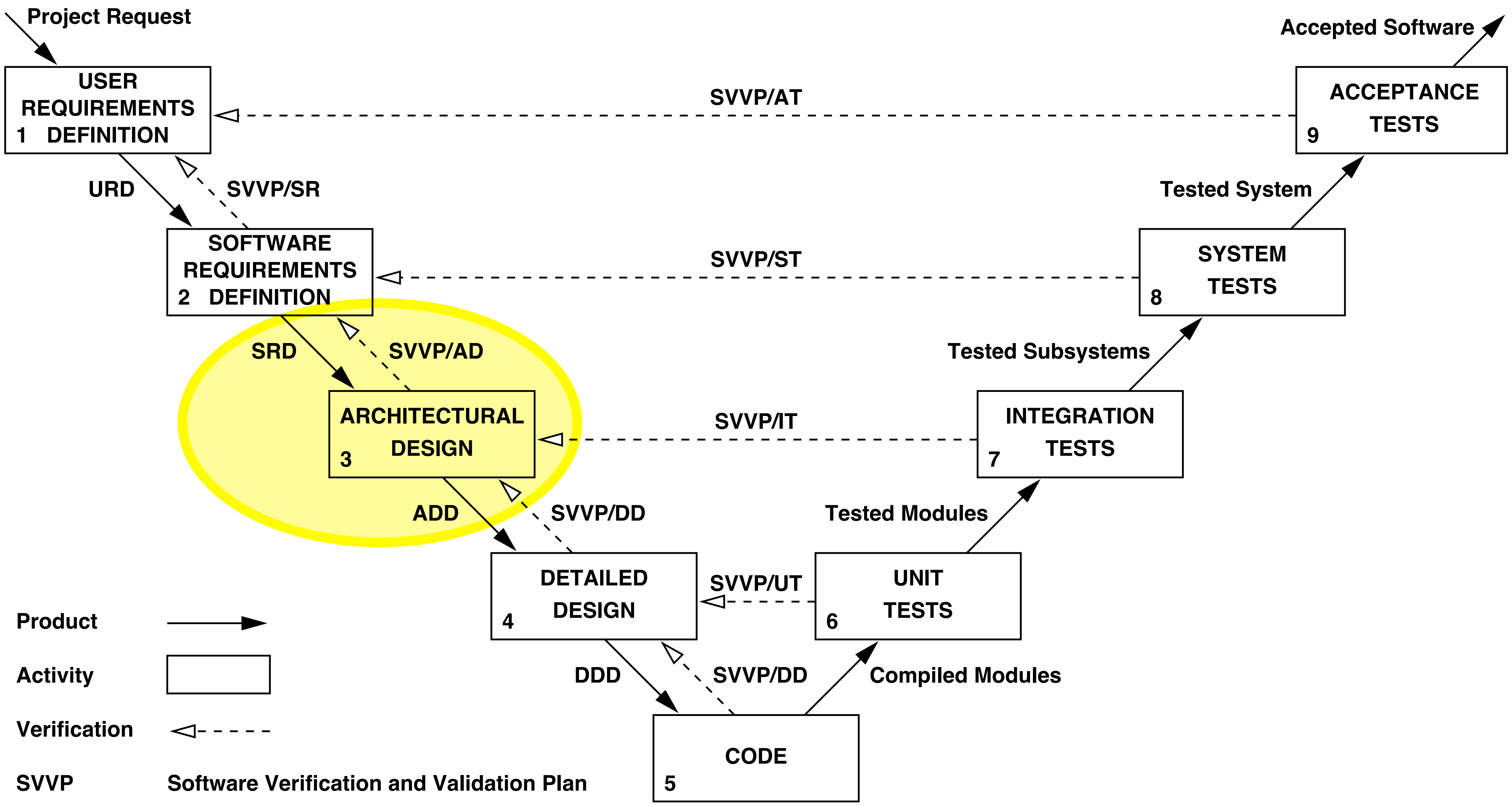
The Big Picture

- Software Engineering, *and Architecture in particular*, is all about managing complexity
- Divide and Conquer
- Abstraction (deciding what to ignore when)

```
HW = 'Hello World!';
document.writeln(HW);
document.writeln(HW);
```

```
function twice(s) {
  document.writeln(s);
  document.writeln(s);
}
twice('Hello World!');
```

Context of Software Architecture



ESA Software Engineering Standards: Life Cycle Verification Approach

System Engineering

From: M.J. Christensen, R.H. Thayer. *The Project Manager's Guide to Software Engineering's Best Practices.* Wiley, 2002

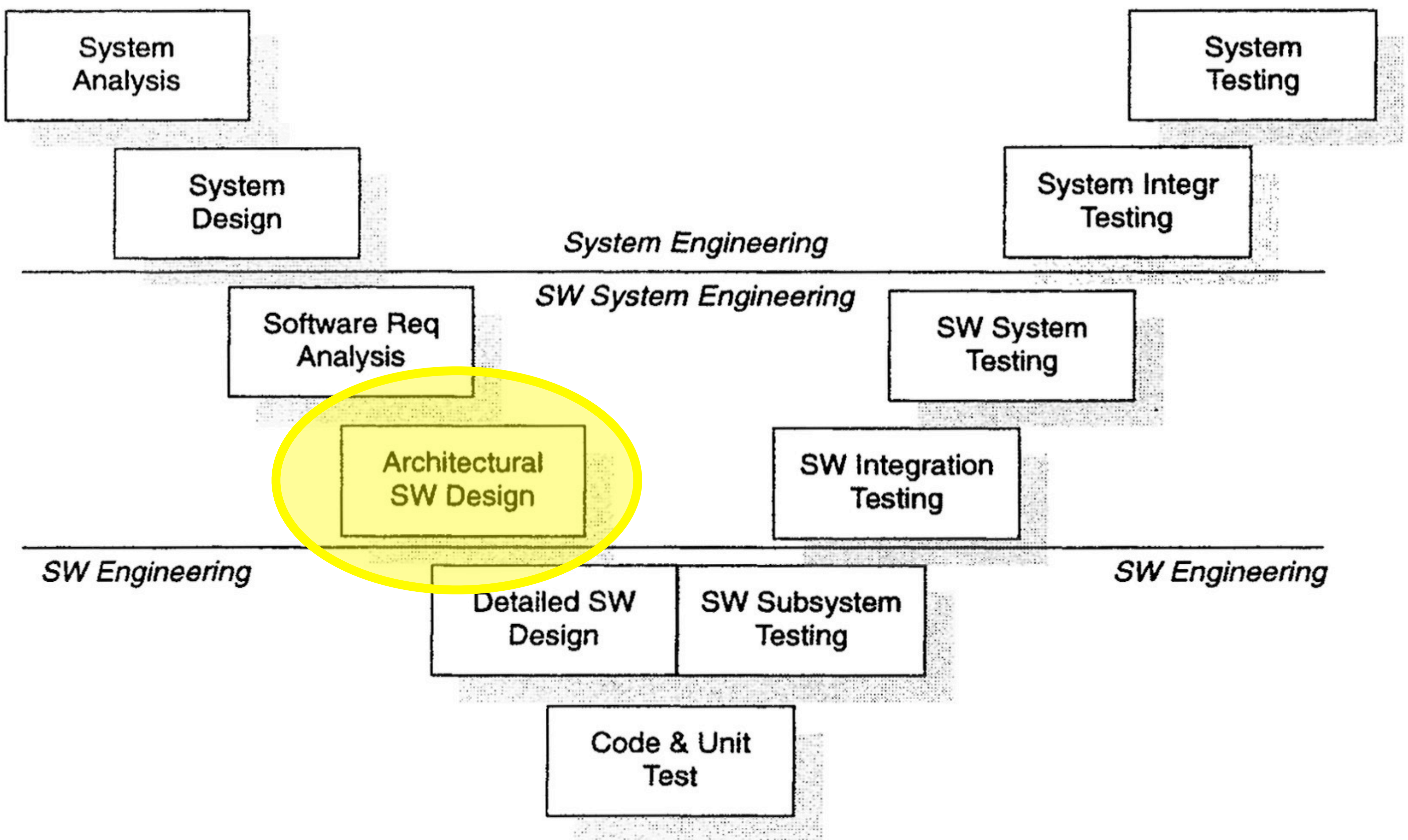
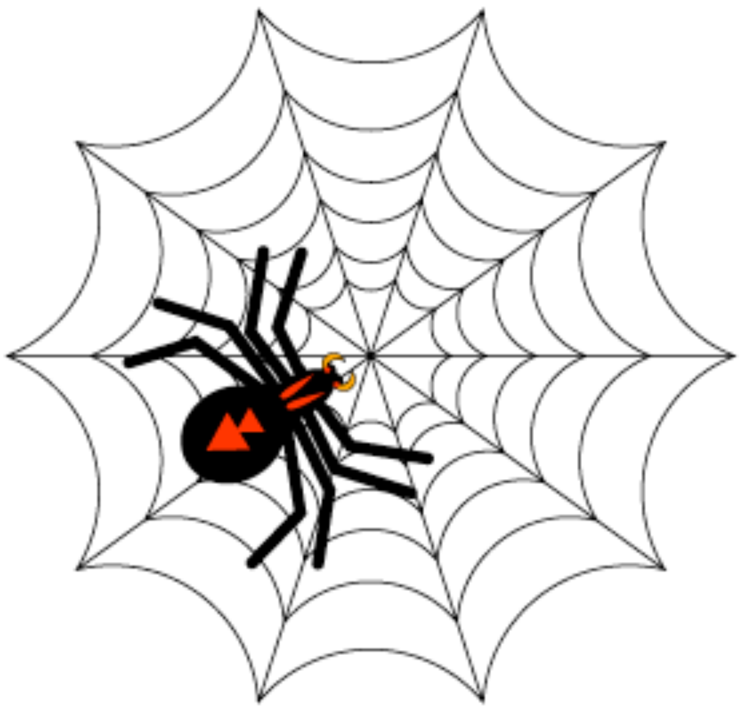


Figure 1.4: Engineering Activities and Product Flow

Who Are You (Going to Be)?

- Software Architect
- Requirements Engineer, Systems Engineer
- Software Engineer
- Test Engineer
- Project Manager
- Quality Engineer
- (Academic) Researcher
- Independent Consultant, Auditor



On What Side of the Table Are You?

- Candidate in job interview (architect-to-be)
- Director of start-up, hiring staff
- Looking for a contractor to do architectural design for your project
- Architect negotiating requirements
- Architect leading a design team
- Assistant in a project review or audit

Range of Project Sizes

- Small: one-person, one-month effort
- Large: >100 M€, >100 persons, >10 yrs
- Single-platform versus multi-platform, etc.
- Requires (very) different approaches
- “People problems” play a role

Existing Industrial Architectural Frameworks

- IBM
- Oracle
- Microsoft
- Sun

Architecture Tooling

- Architecture Description Languages (ADLs)
  - openArchitectureWare (in Eclipse)
  - Acme (CMU)
  - AADL
  - ...
- Lattix Architecture Management System

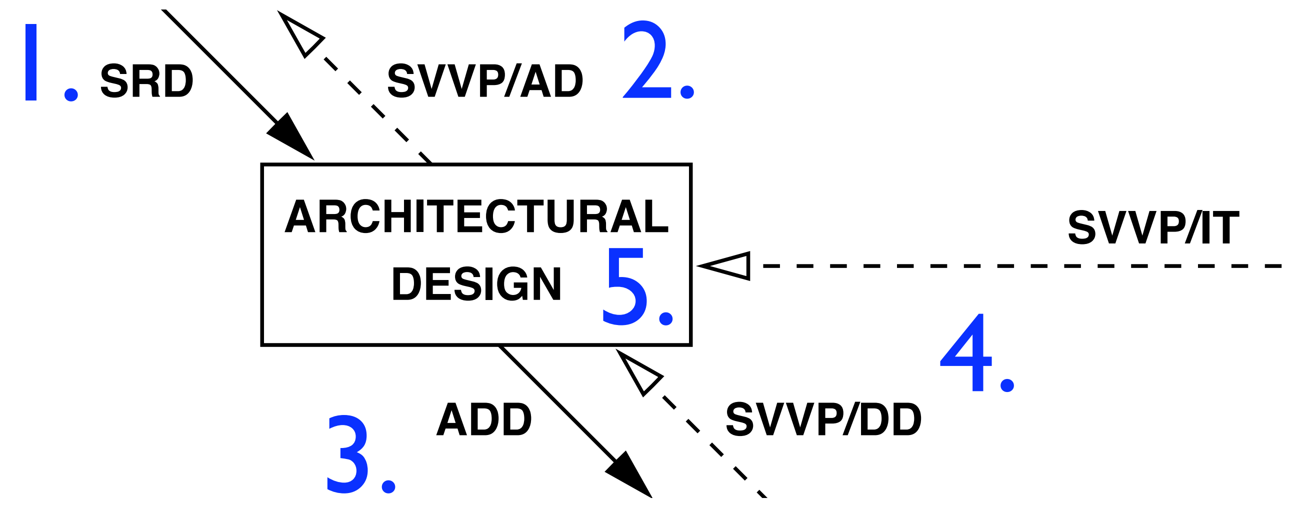
Course Goals

- Know the fundamental concepts in context
- Awareness of issues, approaches, and future trends
- Ability to find and read relevant literature
- Ability to critically assess
- A quantitative, scientific/engineering attitude
- NOT: Make you an architecture designer

Key Questions

- What to know? (Fundamentals vs. state of the art)
- What to do?
- How to do it?
- What to deliver?
- Who does what when?
- Creating a Software Architecture is not an atomic action, but involves various activities and kinds of persons. You can't do everything alone at once.
- (Un)fortunately: (too) many answers

Topics in Block 2



1. From Req. to Arch.: Doing Design
2. From Arch. to Req.: Doing Evaluation
3. From Arch. to Code: Doing Implementation, code generation, infrastructure for testing, code configuration managment
4. From Code to Arch.: Monitoring impl. work, Reverse Engineering, Integration
5. Process, Documentation, Tools, Standards

With a Focus on Evaluation

Tentative Schedule

9. Introduction
10. Architecture & Implementation
11. Architecture & Requirements
12. Architecture Evaluation
13. Component-Based Architecture
14. Reverse Engineering an Architecture
15. Model-Driven Engineering/Architecture
16. Guest Lecture

Architecture (IEEE def.)

- The fundamental organization of a system
- embodied in its components,
- their relationships to each other and
- to the environment, and
- principles guiding its design and evolution.

Alternative definition: Set of high-level design decisions

Architectural Description of Sw-Intensive Systems: IEEE Std 1471-2000

a) Expression of the system and its evolution

b) Communication among the system stakeholders

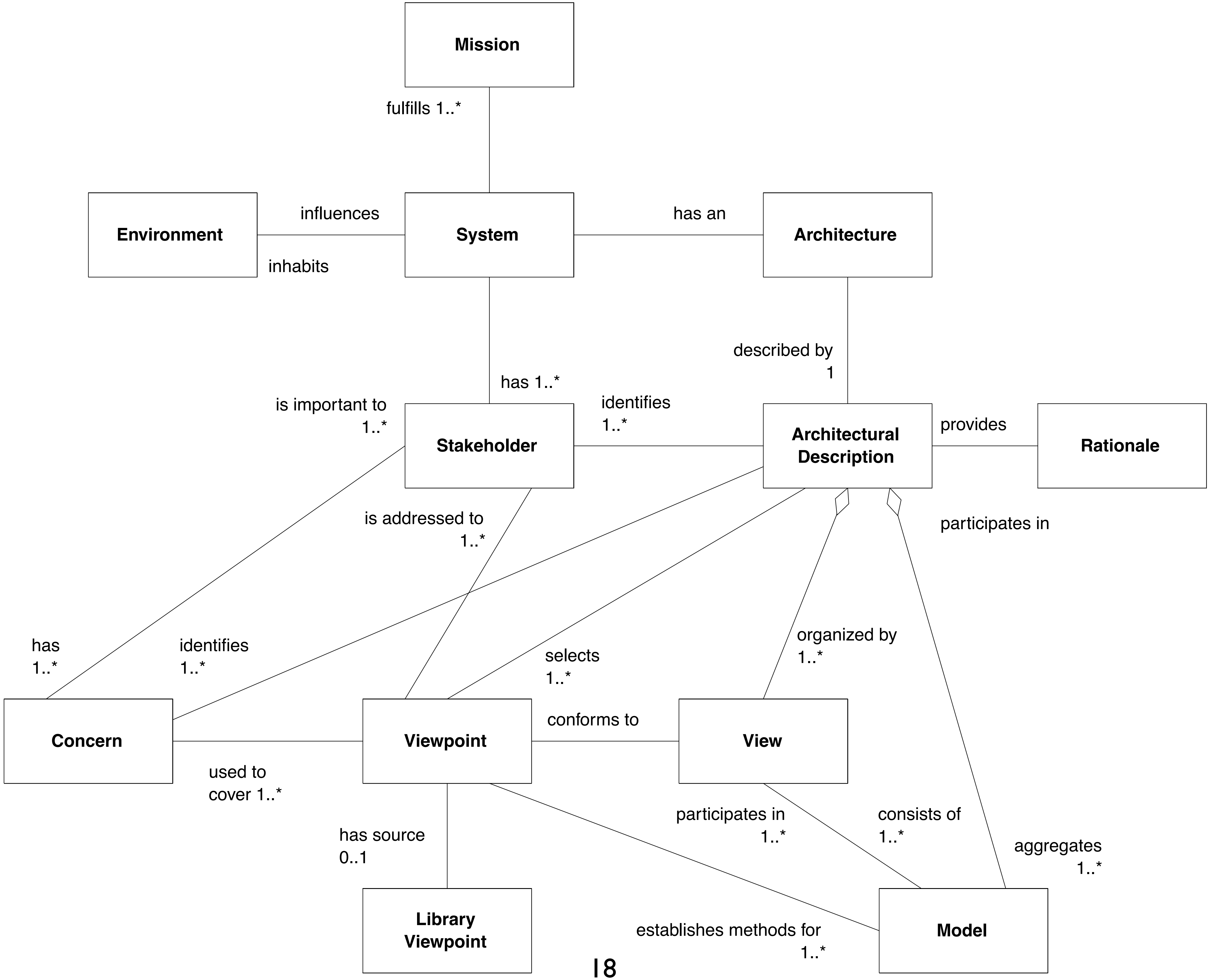
c) Evaluation and comparison of architectures in a consistent manner

d) Planning, managing, and executing the activities of system development

e) Expression of the persistent characteristics and supporting principles of a system to guide acceptable change

f) Verification of a system implementation's compliance with an architectural description

Conceptual model of architectural description



Architectural Description

b) Identification of the system stakeholders and their concerns judged to be relevant to the architecture

c) Specifications of each viewpoint that has been selected to organize the representation of the architecture and the rationale for those selections

d) One or more architectural views

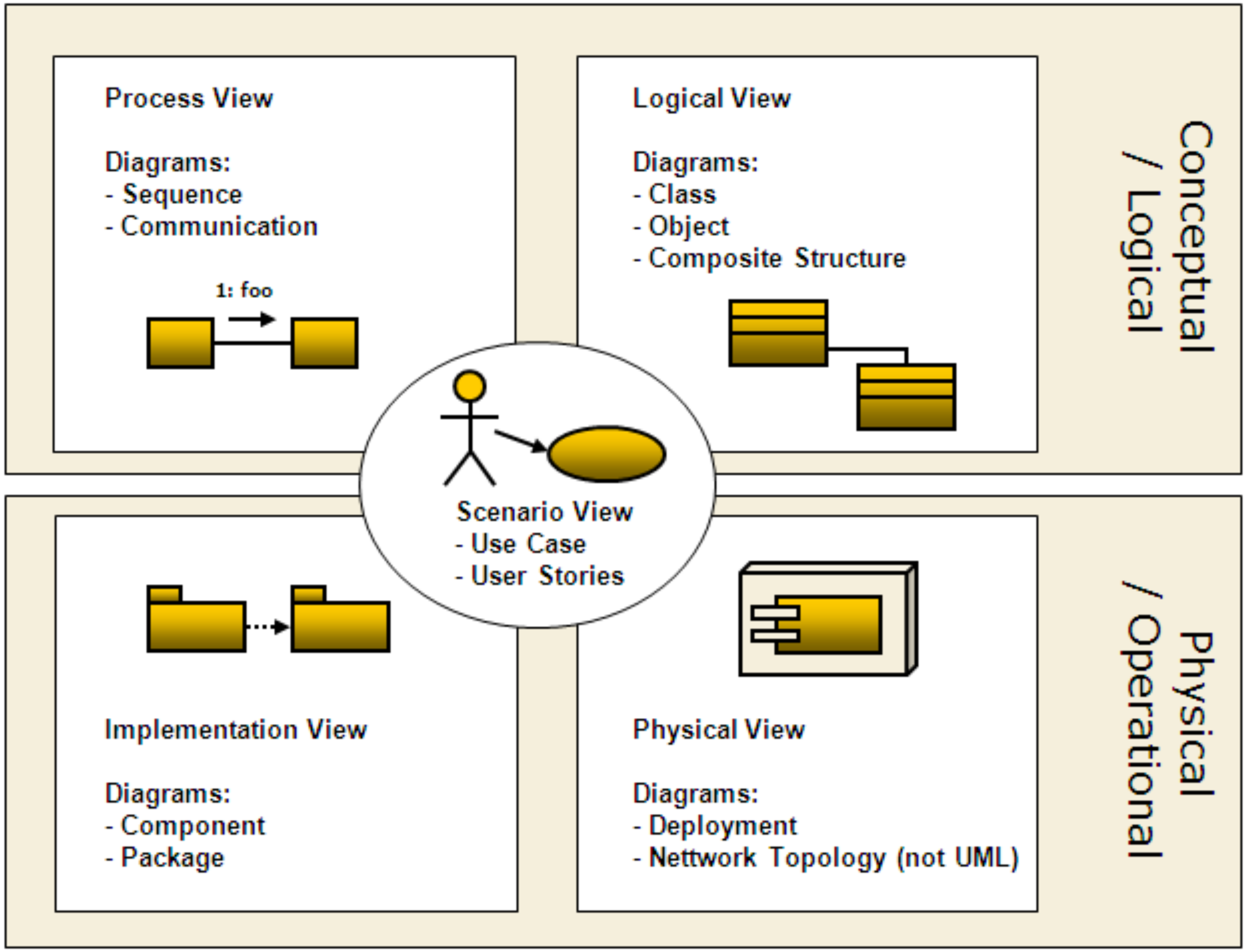
e) A record of all known inconsistencies among the architectural description's required constituents

f) A rationale for selection of the architecture

Example Viewpoints

- Structural viewpoints
- Behavioral viewpoints
- Physical interconnect viewpoint
- Link bit error rate viewpoint
- Decomposition and allocation, Enterprise, Information, Computational, Engineering, Technology

Kruchten's 4+1 Views



Implementation View = Development View   Physical View = Deployment View

Why Architecture?

- Organizes communication about *solution* domain.
- Facilitates parallel construction by a team.
- Improves ability to plan work, track progress.
- Improves verifiability (facilitates getting it to work):
  - Allows early review of design.
  - Allows unit testing of separate components.
  - Allows stepwise integration (no "big bang").
- Improves maintainability: changes affect few components.
- Improves possibilities for reuse.

Economy of Defects

- The longer a defect is undiscovered, the higher its cost: cost grows exponentially in amount of time between injection and removal of a defect.
- Defects decrease the predictability of a project. Cost and time of defect localization and repair are extremely variable.
- Defects concern risks (uncertainty); product could be defect-free at once, but defects are likely.
- The likelihood of defects increases rapidly with higher system complexity.

Quality Chain

- Product-in-use qualities: Car gets end-user how quickly/reliably from A to B? …
- External product qualities: Max. speed of car? Garage bills …
- Internal product/design qualities: Engine specs, choice of materials, …
- Process qualities: Factory organization …

Lack-of-Quality Chain

- Product-in-use: failures
- Product itself (before use): defects, faults
- Product Design: defects, faults
- Process: (human) mistakes
- Read: Ariane 5 Failure Report

Modularization: Divide and Conquer

- Define subsystems/components/modules and their interfaces
- How to decide what goes where
- How to describe: IEEE Std 1016-1998
- Programming languages offer facilities for modularization, but these are often unsuitable for describing an architecture

Sw Design Description

- IEEE Std 1016-1998
- Recommended Practice for SDD
- SDD describes structure of Sw solution
- Design entities & attributes
- Necessary, intrinsic attributes

Design Entity Attributes

- Identification (unique name, for reference)
- Type (nature of the component, e.g. library)
- Purpose (why, traced to requirements)
- Function or data type (what it does/stores)
- Subordinates (constituting components of composite entities)

Design Entity Attributes (2)

- Dependencies (relation to other entities: uses, requires)
- Interfaces (provided to other entities, incl. protocols)
- Resources (used from outside design)
- Processing (algorithmic details of function)
- Data (stored/maintained inside entity)

Non-Intrinsic Attributes

- Designer names
- Design status
- Revision history

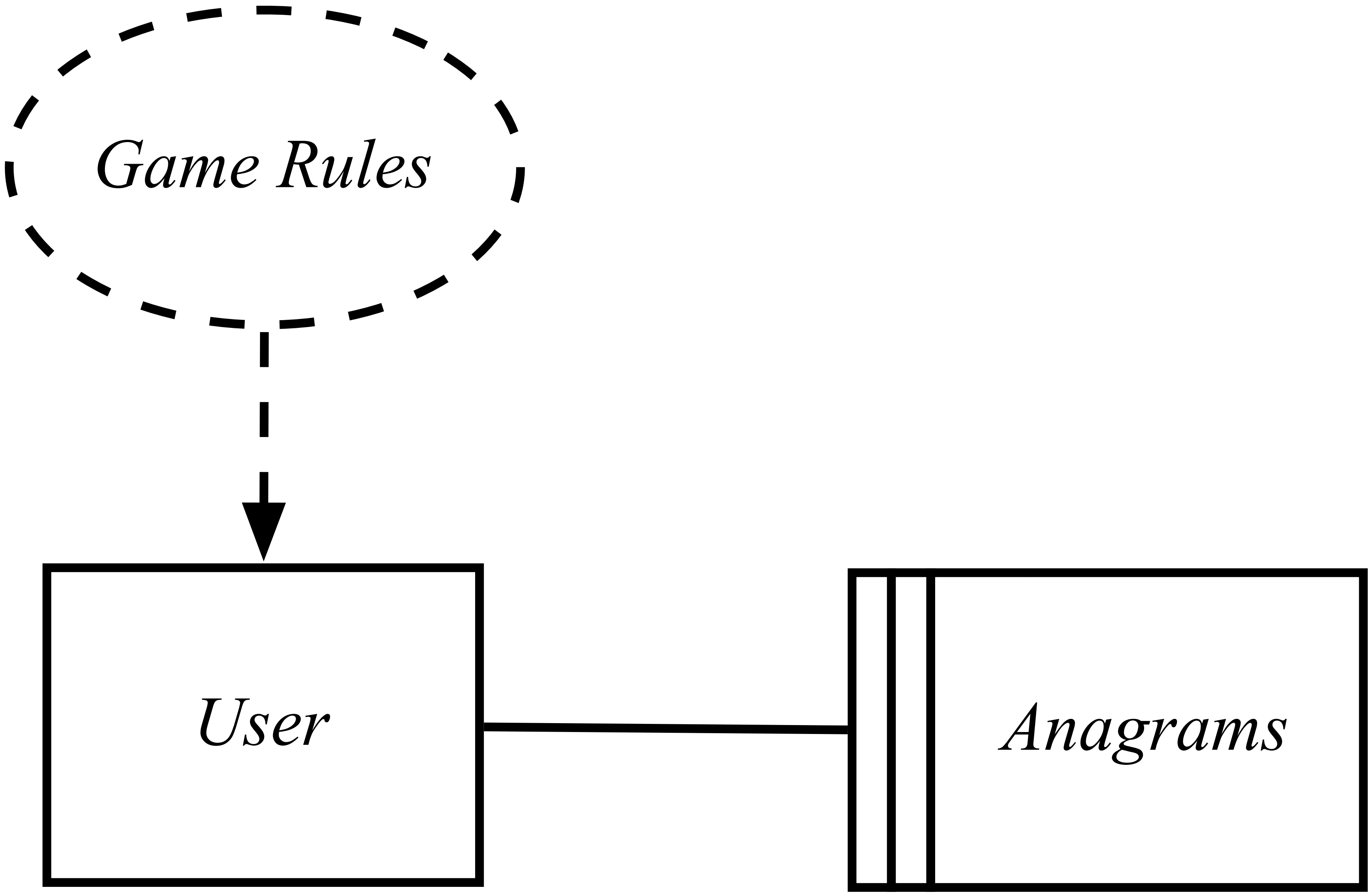
Design View: Subset of design entity attribute information

Table 1—Recommended design views

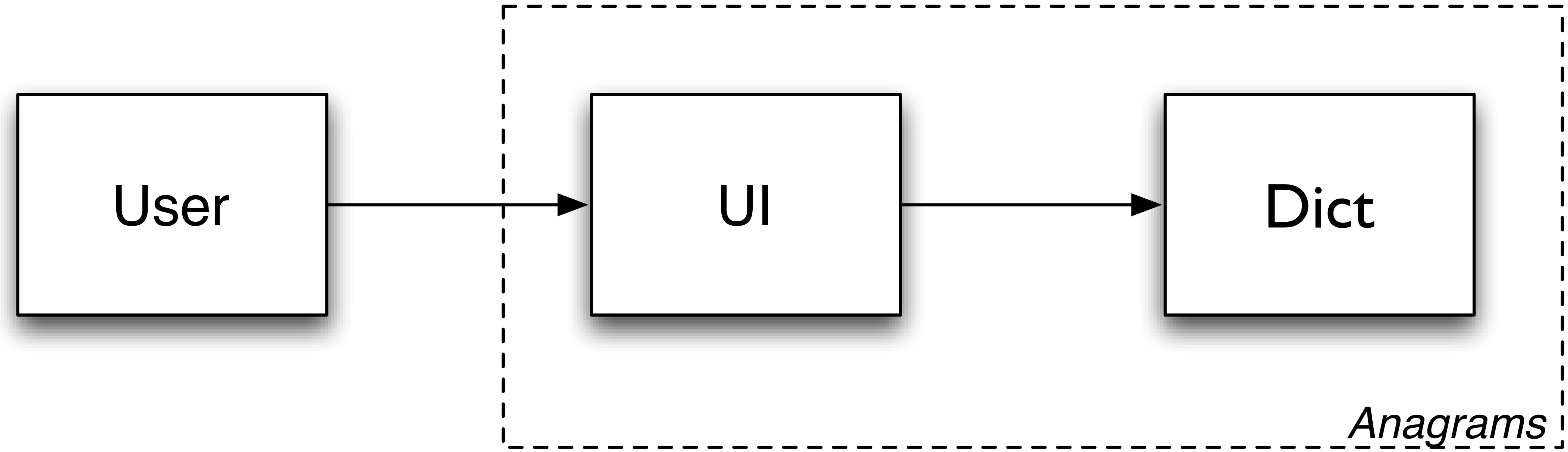
Design view	Scope	Entity attributes	Example representations
Decomposition description	Partition of the system into design entities	Identification, type, purpose, function, subordinates	Hierarchical decomposition diagram, natural language
Dependency description	Description of the relationships among entities and system resources	Identification, type, purpose, dependencies, resources	Structure charts, data flow diagrams, transaction diagrams
Interface description	List of everything a designer, programmer, or tester needs to know to use the design entitites that make up the system	Identification, function, interfaces	Interface files, parameter tables
Detail description	Description of the internal design details of an entity	Identification, processing, data	Flowcharts, N-S charts, PDL

There are *various* logical/development (sub)views

Mini Example: *Anagrams* Requirements as Problem Frame (Context Diagram)



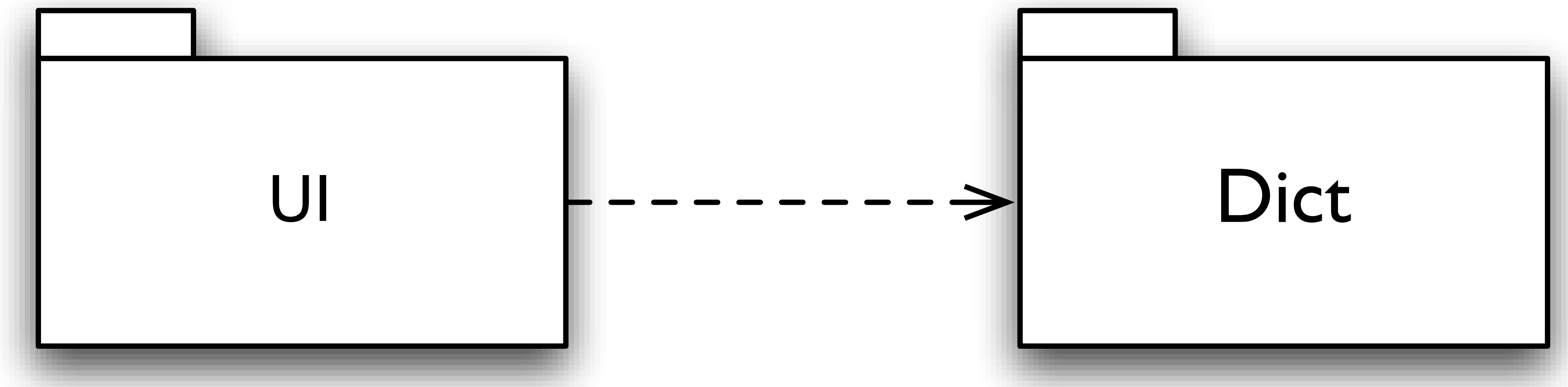
Architecture: Logical View (Decomposition)



What is the most (?) important information conveyed in this diagram?

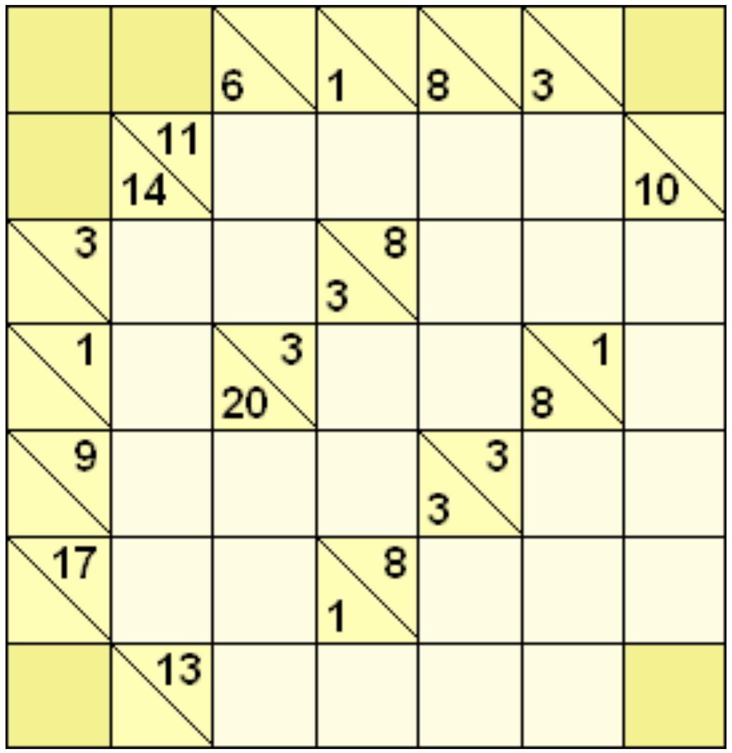
That User is not directly related to Dict

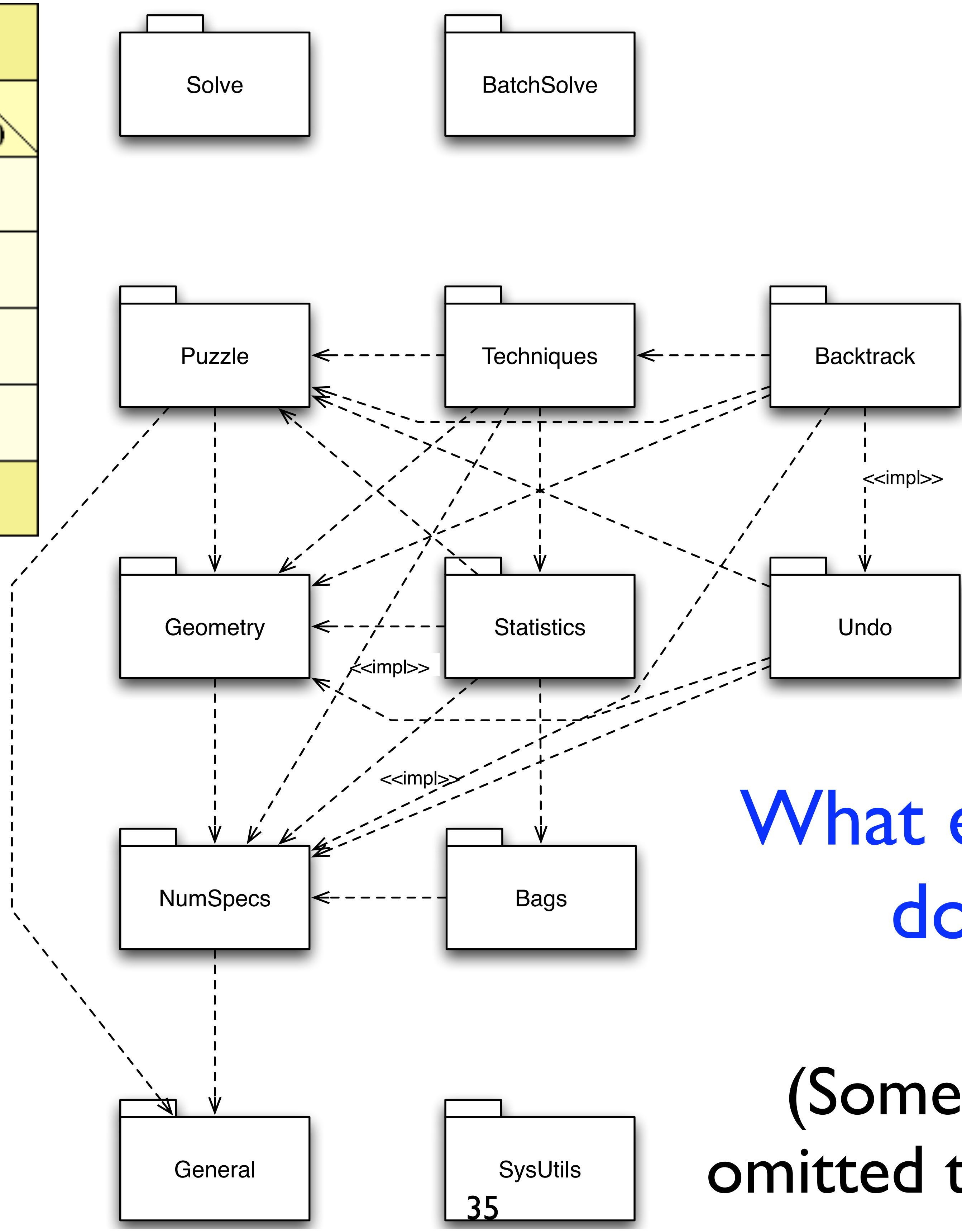
Package Dependencies: Development View



Keep It Simple, Stupid (KISS):
Development view can mimic logical view

Kakuro Architecture



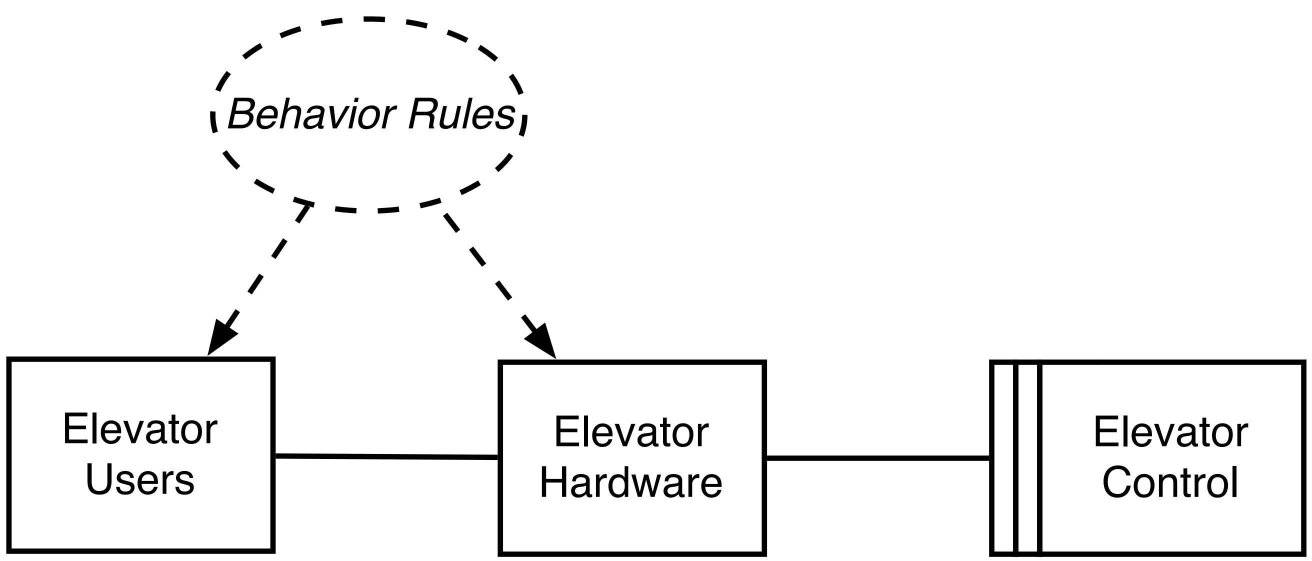


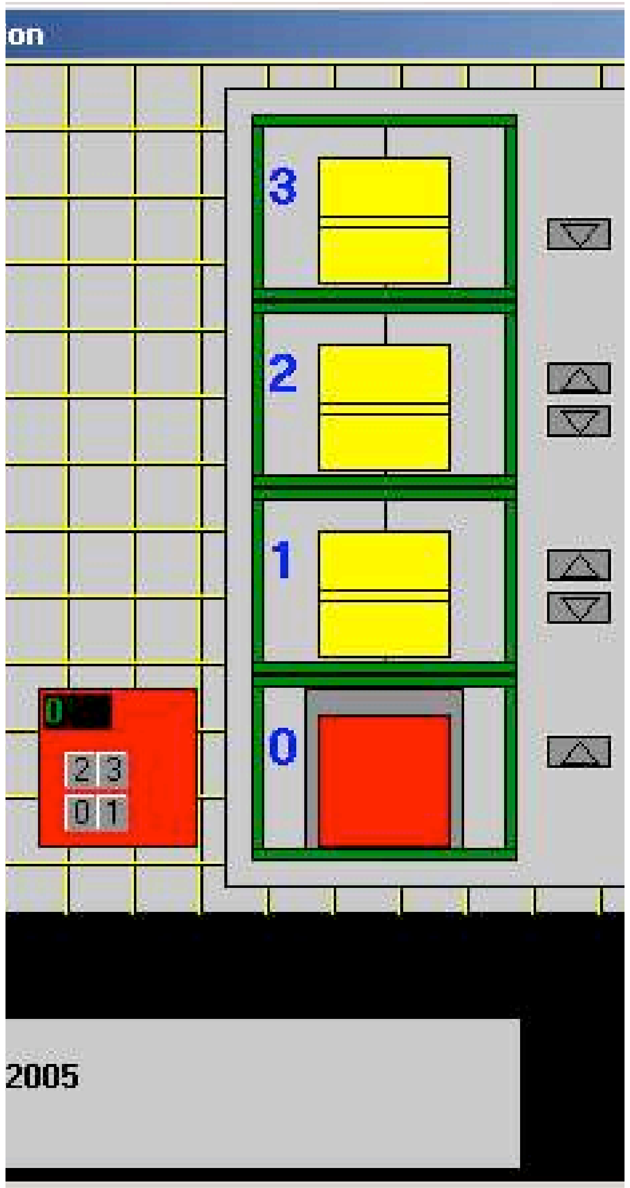
What evolution can do to you!

(Some arrows were omitted to avoid clutter!)

Elevator Control

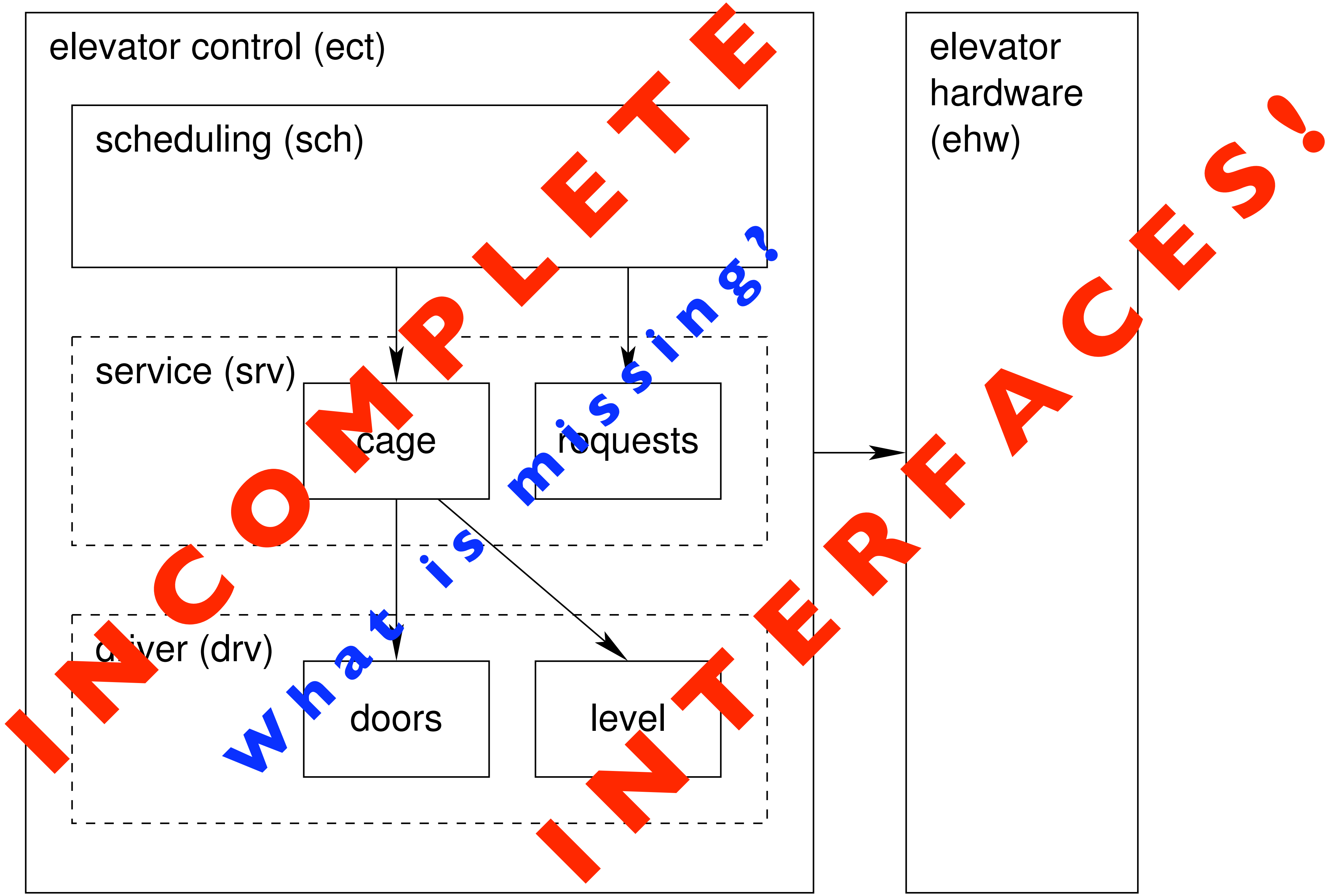
- Single-cage four-floor elevator
- Separate cage doors and floor doors
- Cage and floor buttons with lights
- Display in cage



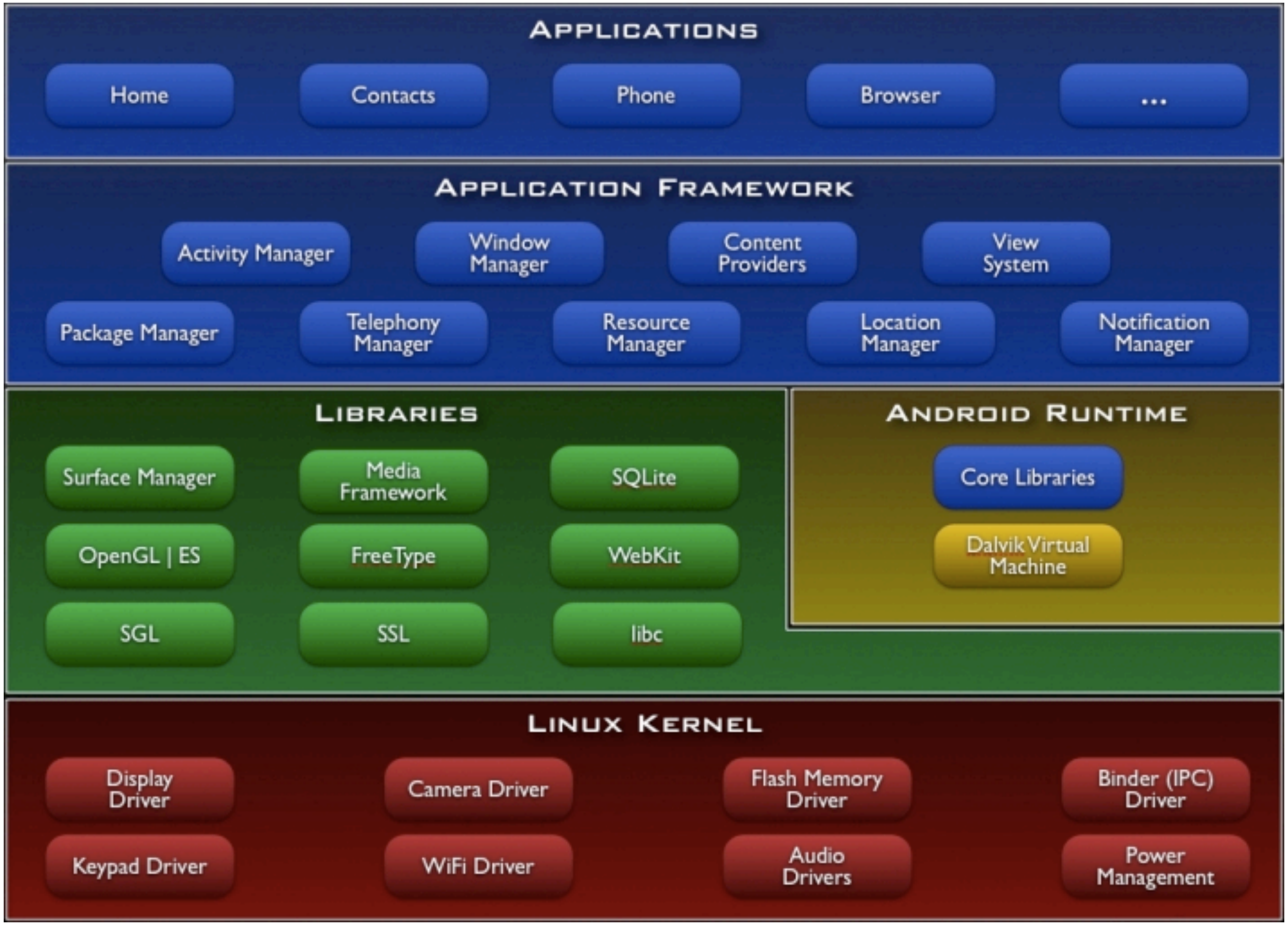


Elevator Users do not interact directly with Elevator Control!

Elevator Control Architecture: Logical View



Android: Development View



Evaluate Modularization

- Number and size of components
- Number of relations (less is better)
- Coupling: how components depend on others
- Cohesion: similar items in same component
- Complexity/nature of interfaces
- Fan-in, fan-out

Kinds of Cohesion

- Coincidental cohesion (worst)
- Logical cohesion (e.g. input module)
- Temporal cohesion (e.g. initialization)
- Procedural cohesion (e.g. batch processes)
- Communicational cohesion
- Sequential (output-to-input) cohesion
- Fuctional cohesion (best)

Kinds of Coupling

- Content: via internals, not using specified interfaces (high/bad)
- Common (via global variables)
- External (via a file format, common protocol)
- Control (via command parameter)
- Stamp (passing too much information)
- Message coupling (low)
- Routine call, call-back
- Type use
- Inclusion/import
- No coupling (lowest)

Homework Assignment 6

- About coupling & cohesion (will be made available on webpage and in peach)

Main Book for Part 2

- L. Bass, P. Clements, R. Kazman. *Software Architecture in Practice (2nd Ed.).* Addison-Wesley, 2007.

Supplementary (more recent) textbook:

- R.N. Taylor, N. Medvidovic, E.M. Dashofy. *Software Architecture: Foundations, Theory, Practice.* Wiley, 2010.

Reading Material

- M.J. Christensen, R.H. Thayer. *The Project Manager's Guide to Software Engineering's Best Practices.* Wiley, 2002. Chapter 1.
- *ARIANE 5: Flight 501 Failure.* Report by the Inquiry Board. July 1996.
- *IEEE Recommended Practice for Architectural Description of Software Intensive Systems*. Std 1471-2000.
- M.W. Maler, D. Emery, and R. Hilliard. Software Architecture: Introducing IEEE Standard 1471, *IEEE Computer*, April 2001.

Reading Material (2)

- D.L. Parnas. On Criteria To Be Used in Decomposing Systems into Modules. *CACM* **15**(12), Dec. 1972.
- [Optional] E. Yourdon and L.L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and System Design.* Prentice-Hall, 1979.
- [Optional] *IEEE Recommended Practice for Software Design Descriptions.* Std 1016-1998.