Components Based Design and Development

Computer Engineering Studies

Universidad Carlos III de Madrid

Unit 3: Software Design Quick Overview

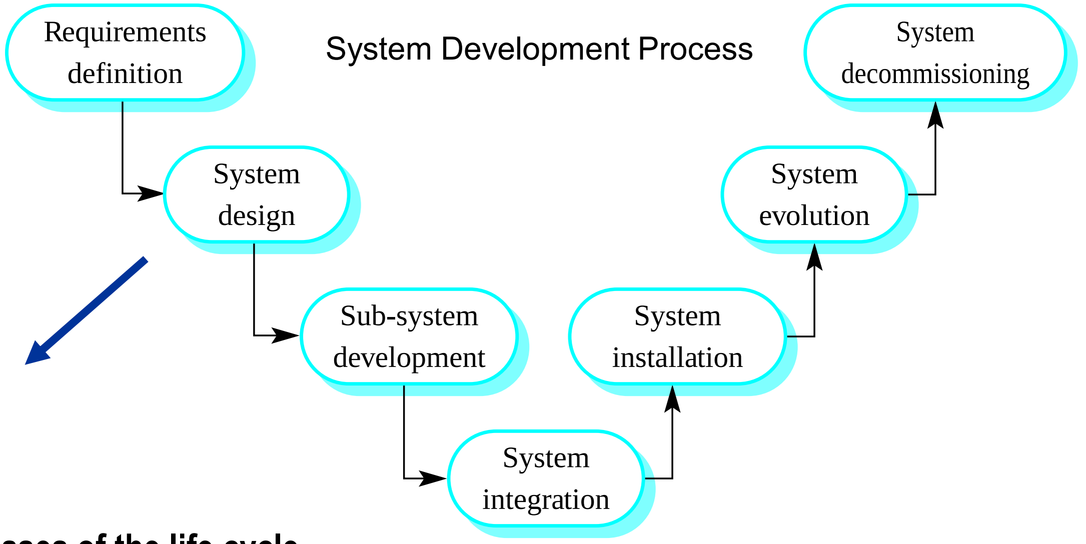
Juan Llorens

Högskolan på Åland – Finland / Universidad Carlos III de Madrid - Spain

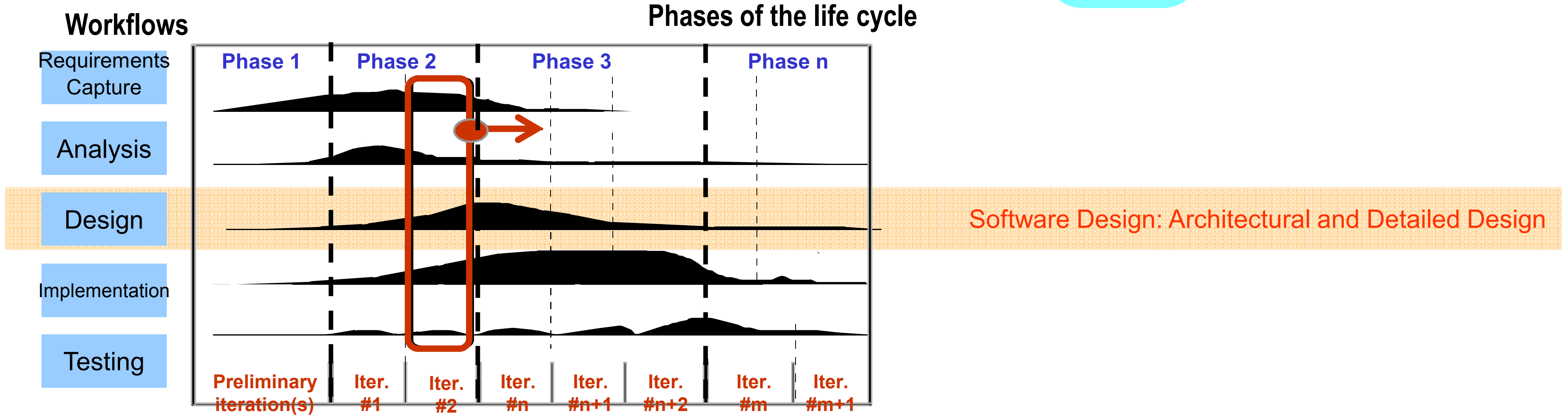
Juan.llorens@uc3m.es

Information Engineering HÖGSKOLAN PÅ ÅLAND

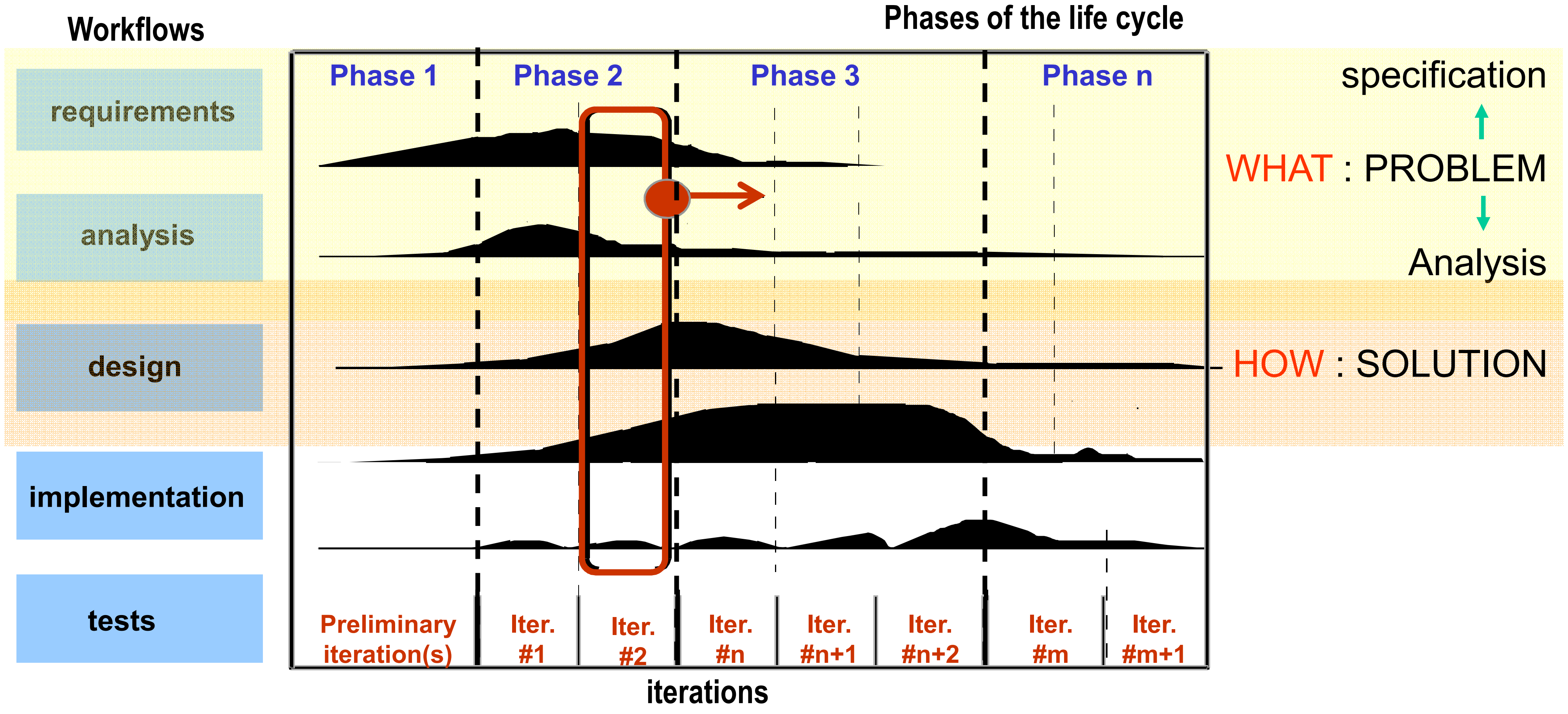
Where is the Software Design activity located



For Software subsystems



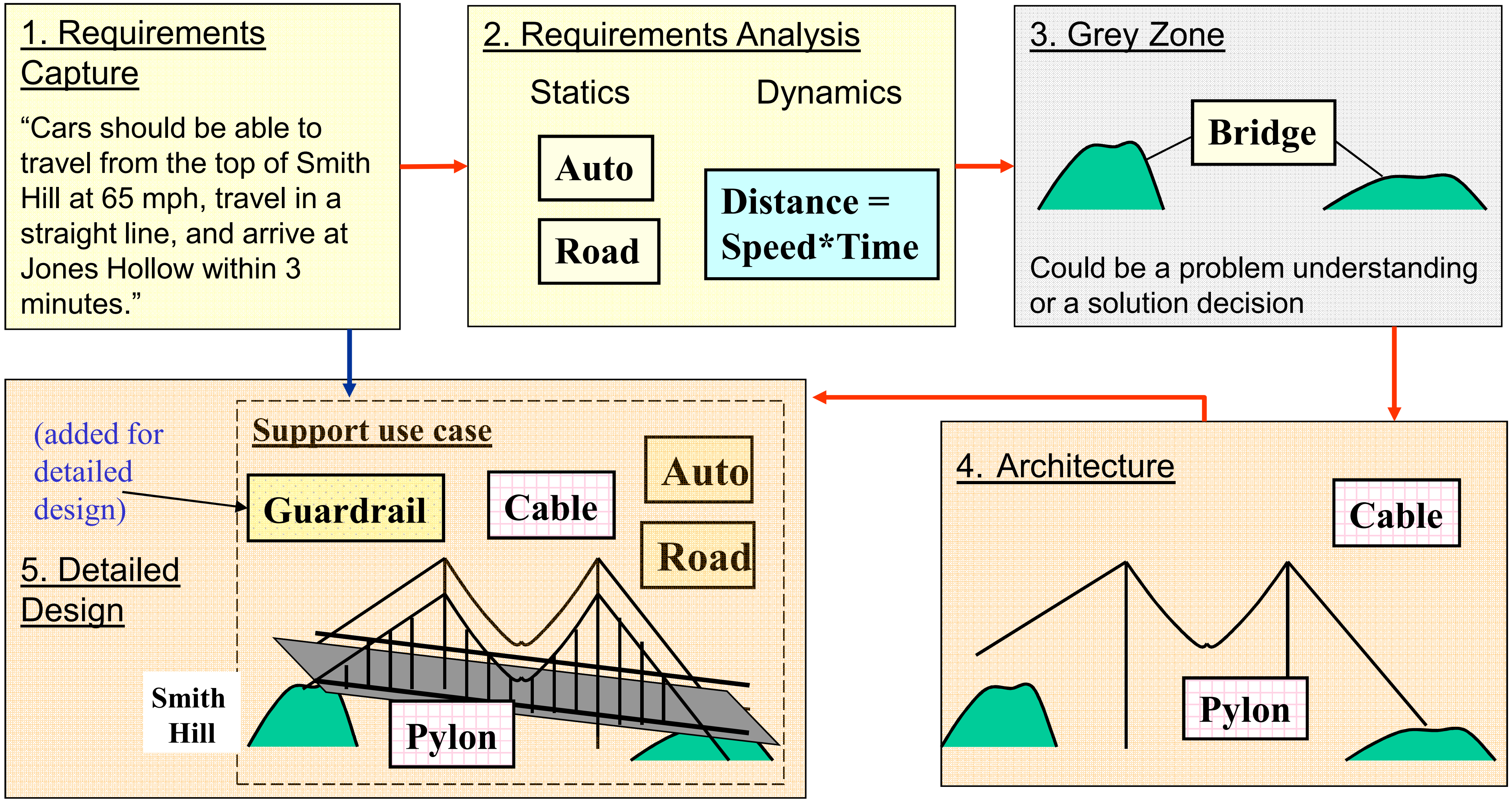
Transition from Analysis to Design



Transition from Analysis to Design (II)

- The Analysis activity pretends to model an understanding of the system to be developed => [Application] Problem Domain
- The Design activity pretends to model a possible solution to a particular [Application] problem => [Application] Solution Domain
- Grey Zone : Where is the border between both activities?

Transition from analysis to Design : Example


1. Requirements Capture

"Cars should be able to travel from the top of Smith Hill at 65 mph, travel in a straight line, and arrive at Jones Hollow within 3 minutes."

2. Requirements Analysis

3. Grey Zone

Could be a problem understanding or a solution decision

4. Architecture

5. Detailed Design

(added for detailed design)

Based on E. Braude, *Software Engineering: An Object-Oriented Perspective*

Software design

- The process of converting the system specification into computational model ready to be coded.
- Software design
  - Design a software structure that realises the specification;
- And.. Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

Based on Ian Sommerville 2004 - Software Engineering, 7th edition. Chapter 4

So… What is Software Design

- To develop Computational Models representing a complete Solution to a proposed software need (Problem).
  - Design implies to find computational solutions to problems representing functional and not functional needs
  - Design implies to perform Modeling

Design => Solve problems using models

What are the design instruments?

- 1- A Process guidance
  - A Design process defining what activities to follow
- 2- Techniques
  - 2.1 High Level: Problem solving techniques
  - 2.2 Low Level: Modeling techniques
- 3- Rules
  - Constraints applied to system models;
- 4- Tools
  - Whatever conceptual device that can be applied to solve a particular problem
- 5- Recommendations
  - Advice on good design practice;
- 6- Documentation guidelines
  - Based on document templates

Based on Ian Sommerville 2004 - Software Engineering, 7th edition. Chapter 1

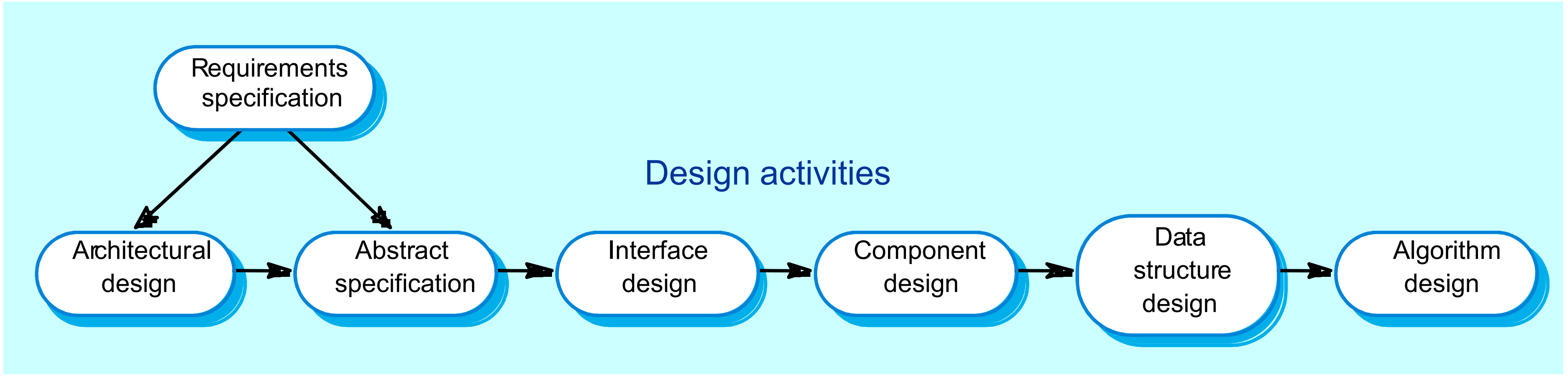
1- Design process model: process activities

- A general process model for Components Based Development (CBD):

– Architectural design	: The overall organization of the system
– Abstract specification	: The problem formal specification
– Interface design	: The definition of communications
– [Detailed] Components design	: The definition of components
– Data structure design	: The definition of information
– Algorithm design	: The definition of behaviour

Based on Ian Sommerville 2004 - Software Engineering, 7th edition. Chapter 4

Sommerville’s software design process



- ***Architectural design:*** Identify sub-systems.
- ***Abstract specification:*** Specify sub-systems.
- ***Interface design:*** Describe sub-system interfaces.
- ***Component design:*** Decompose sub-systems into components.
- ***Data structure design:*** Design data structures to hold problem data.
- ***Algorithm design:*** Design algorithms for problem functions.

©Ian Sommerville 2004 - Software Engineering, 7th edition. Chapter 4

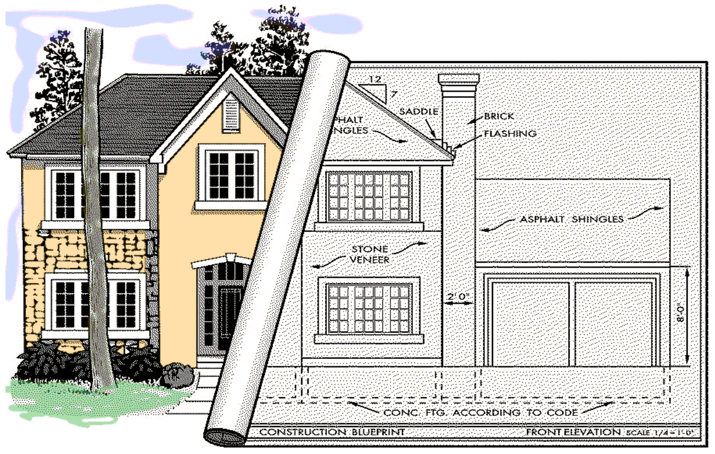
2.1- Problem Solving techniques for Software Design

- Apply existing design solutions to already solved design problems that occur in my project
- Be able to find and Reuse previous designs (solutions) for the problems of my project
- Build in advance general solutions to be applied to my present and future problems
- If nothing from the previous help => model a solution form scratch

Based on Ian Sommerville 2004 - Software Engineering, 7th edition. Chapter 1

2.2 - Modeling: Analogy with architecture (for buildings)

- Does it make sense to lay the bricks before making the plans?
- the model - the plans help to master the **complexity of the project**
- What is the **language adequate** for representing the plans?
- Direct engineering and reverse engineering: a house, a car, a virus...



Direct engineering

reverse engineering

What is a model?

- **Abstraction or simplification of the reality**: divide and conquer
- Various types of models:
  - structure, electricity, sanitation, …
  - static, dynamic...

    ➔**how are they related between themselves?**
- Formal models and informal models
  - **Informal models**: *ad hoc*, without a common language
  - **Formal models**: universal language, precision, rigor, coherence
- Modelling and language
  - language is a vehicle for your thoughts: helps to **think clearly**
  - modelling is an **essential element** of the process of software development
  - modelling requires an **adequate language**
- To model is not (just) to make diagrams but to **think with diagrams**

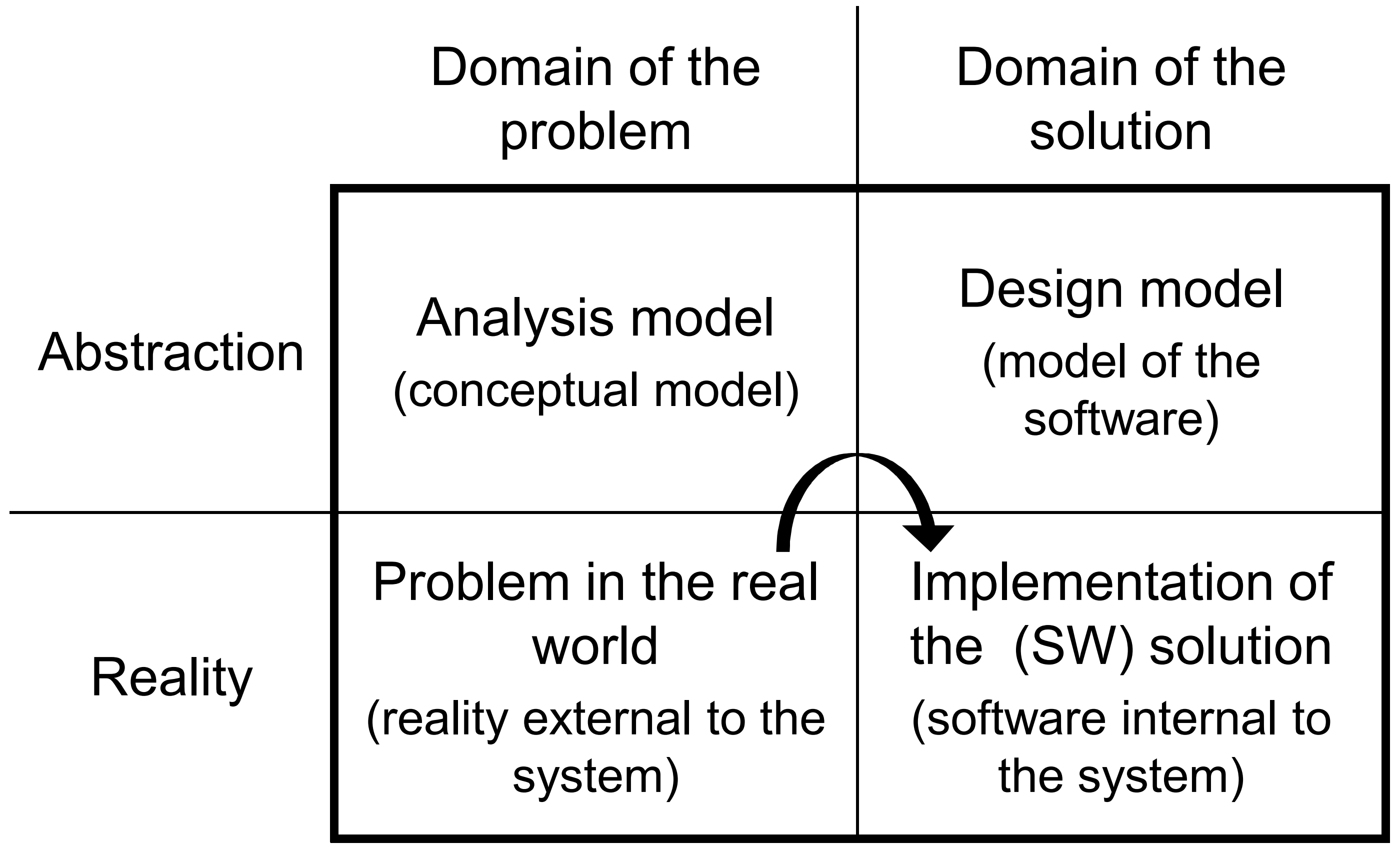
System, model, diagram

- a **system** is a collection of elements organized to fulfil a concrete **finality**
  - a system can be divided into subsystems
- a **model** is an **abstraction** of a system, a simplification (complete and consistent) of the real system, which is used to better understand it
  - Temporarily, a model may be incomplete (missing elements) or inconsistent (containing contradictions)
  - a system can be modelled from different complementary viewpoints, according to what is considered relevant in each case
- a **diagram** is the graphical representation of a set of interconnected elements, a partial view of a model
  - a model is not just a collection of diagrams
  - a model may contain elements not represented in a diagram
  - a model may contain textual specifications which are essential

Analysis models and design models

- Essential parts of the **documentation** of any project
- Despite the use of a similar notation, the models of analysis and design are **abstractions** (models) **of different things**
  - **analysis** (conceptual model): abstraction of the problem, the **real world** as it is, or will be, in which the proposed system will be built
  - **design** (model of the software): abstraction of the internal construction of the proposed **system**, rather than the solution to the problem posed in the real world
- Therefore, they have a different **purpose**
  - **problem**: to study the requirements without making decisions of implementation
  - **solution**: to establish how to build the system before actually building it
- There is no **strict temporal ordering** between analysis and design
  - Analysis precedes design but only within an iteration
  - They are interdependent models; they can evolve in parallel
- **Transition** is neither simple, immediate nor automatic (problem → solution)
  - A good design is not attained simply by "adding details" to the analysis
  - the analysis model cannot be transformed simply in a part of the design model: they represent different realities

Analysis and design represent different realities



	Domain of the problem	Domain of the solution
Abstraction	Analysis model (conceptual model)	Design model (model of the software)
Reality	Problem in the real world (reality external to the system)	Implementation of the (SW) solution (software internal to the system)

H. Kaindl. “Difficulties in the Transition from OO Analysis to Design”. IEEE Software, 16(5), 1999.

Differences between the models of analysis and design

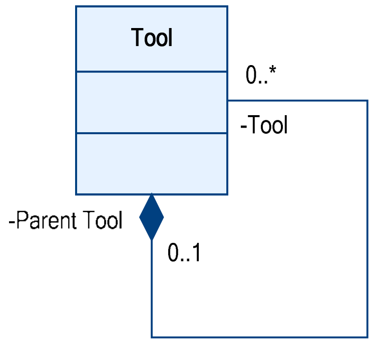
- analysis: creation of a specification of the problem and of the requirements
  - Exploration and clarification of the **requirements** on the system
  - construction of a **model of the real world** based on the concepts of the domain (conceptual model)
  - **What must** (or *does*: reverse engineering) the system **do**, but **not how**
  - Do not introduce design or implementation artefacts

- design: definition of a software solution that satisfies the requirements
  - **Technological solutions** for implementing the requirements on the system
  - construction of a **model of the system** before effectively building it (model of the software)
  - Includes **aspects of implementation**: design patterns, class libraries, components, persistence mechanisms, etc.
  - Introduces **new artefacts**: an object of analysis may be implemented by a group of design objects
  - Takes into account the **implementation platform** (machines, operating systems, languages, etc.), as well as considerations of efficiency, throughput, optimization of resources, fault-tolerance, etc.

- Difficult to determine where the analysis ends and where the design begins

4 – Tools (Not only SW tools)

- The importance for a SW Engineer to understand their existence
- To be a professional Software engineer you must be skilled, trained and experienced in the way the existing tools can be applied within a Software Design.
- Tools must be organized. The Tool-Set Bag.
  - Build a tool-set bag during our course.

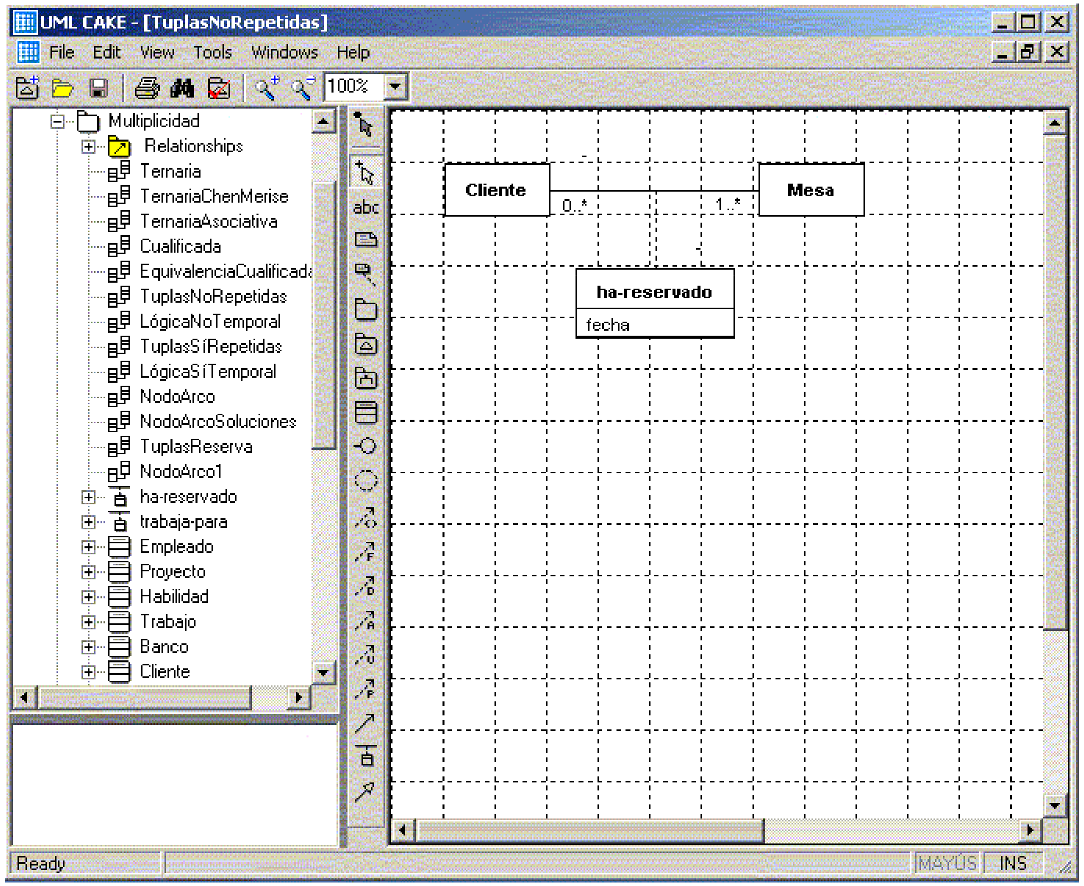


Lets build a personal Tool-Set Bag

Type	Tool	Type	Tool	Type	Tool
General Conceptual	Abstraction	Computational	Object Orientation	Engineering	Modeling
General Conceptual	Encapsulation			Engineering	Diagramming
Organizational	Record Specifications in video				
SW	CASE				
SW	Lower CASE				
SW	Higher CASE				

Modelling tools

- What can offer a CASE tool for UML?
  - Drawing
  - Syntactical verification
  - Consistency between diagrams
  - Integration with other applications
  - Group work
  - Reutilization
  - Code generation...
- Examples
  - Rational Rose
  - Visual UML
  - SWReuser



6 - Documentation Guidelines

ESA PSS-05-04 Issue 1 Revision 1 (March 1995) -THE ARCHITECTURAL DESIGN DOCUMENT

Service Information:
- a - Abstract
- b - Table of Contents
- c - Document Status Sheet
- d - Document Change Records made since last issue

1 Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms and abbreviations
- 1.4 References
- 1.5 Overview

2 System Overview

3 System Context
- 3.n External interface definition

4 System Design
- 4.1 Design method
- 4.2 Decomposition description

5 Component Description
- 5.n [Component identifier]
- 5.n.1 Type
- 5.n.2 Purpose
- 5.n.3 Function
- 5.n.4 Subordinates
- 5.n.5 Dependencies
- 5.n.6 Interfaces
- 5.n.7 Resources
- 5.n.8 References
- 5.n.9 Processing
- 5.n.10 Data

6 Feasibility and Resource Estimates

7 Software Requirements vs Components Traceability matrix

Documentation Guidelines

ESA PSS-05-05 Issue 1 Revision 1 (March 1995) 75 THE SOFTWARE USER MANUAL

Service Information:
- a - Abstract
- b - Table of Contents
- c - Document Status Sheet
- d - Document Change Records made since last issue.

1 INTRODUCTION
- 1.1 Intended readership
- 1.2 Applicability statement
- 1.3 Purpose
- 1.4 How to use this document
- 1.5 Related documents (including applicable documents)
- 1.6 Conventions
- 1.7 Problem reporting instructions

2 [OVERVIEW SECTION]

(The section ought to give the user a general understanding of what parts of software provide the capabilities needed)

3 [INSTRUCTION SECTION]

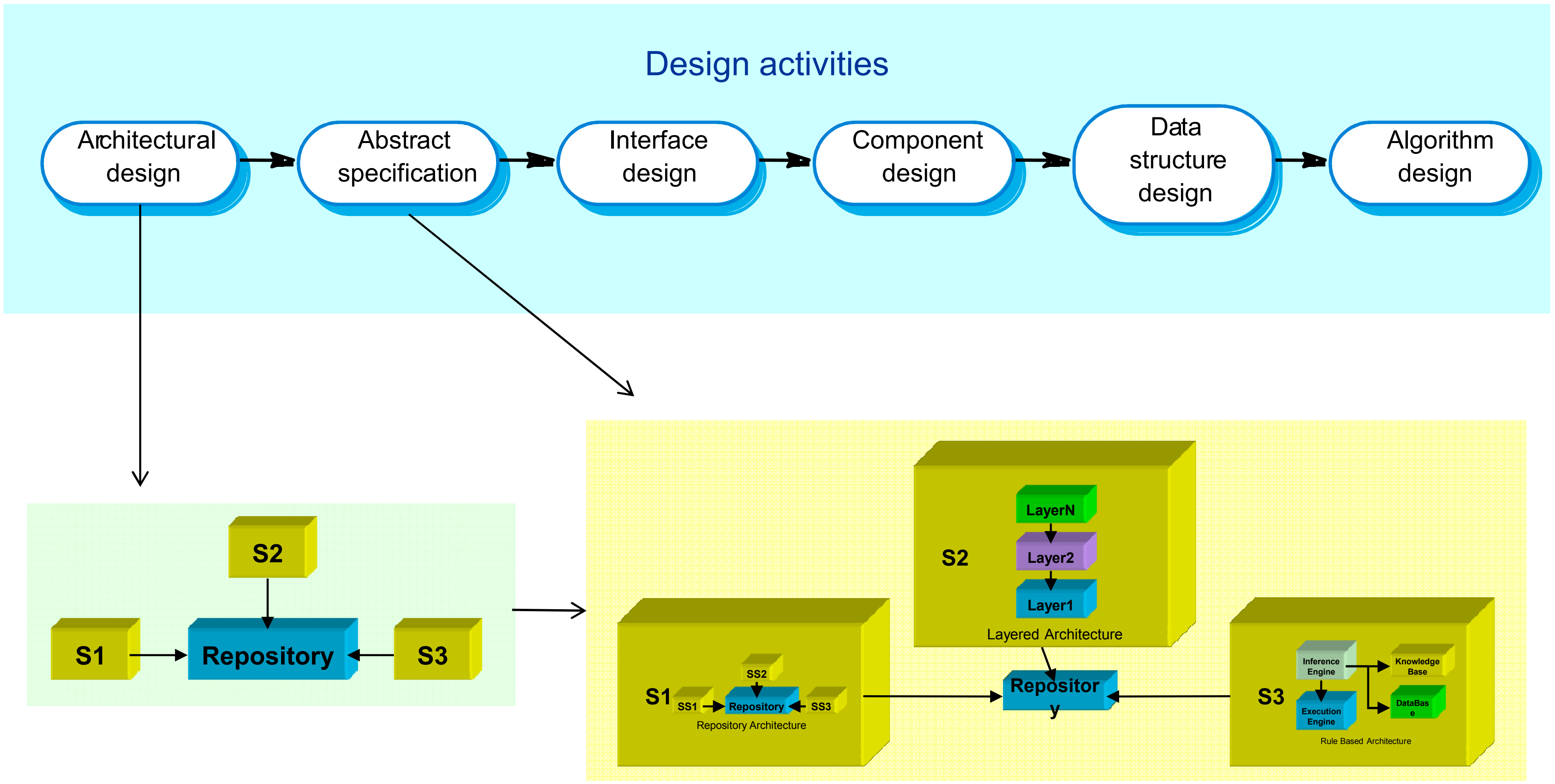
(For each operation, provide...
- (a) Functional description
- (b) Cautions and warnings
- (c) Procedures, including,
  - Set-up and initialisation
  - Input operations
  - What results to expect
- (d) Probable errors and possible causes)

4 [REFERENCE SECTION]

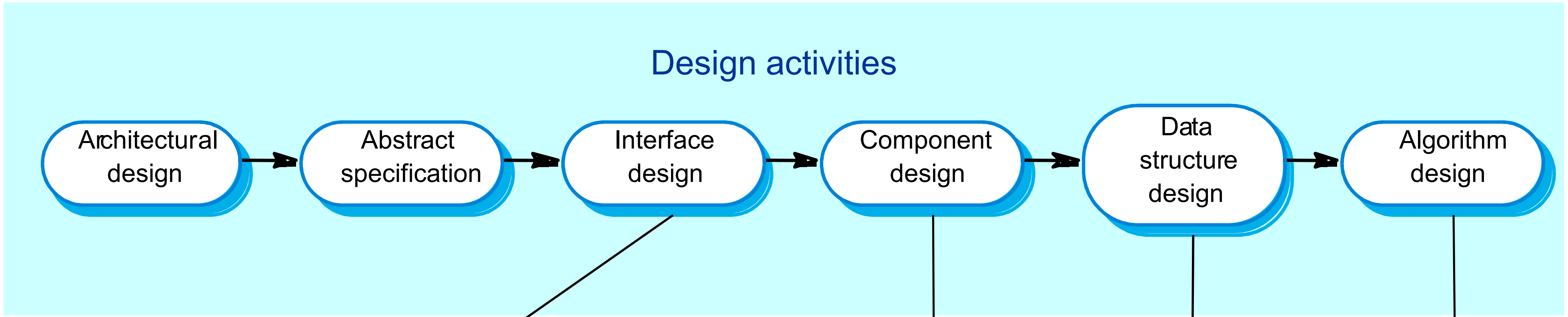
(Describe each operation, including:
- (a) Functional description
- (b) Cautions and warnings
- (c) Formal description, including as appropriate:
  - required parameters
  - optional parameters
  - default options
  - order and syntax
- (d) Examples
- (e) Possible error messages and causes
- (f) Cross references to other operations)

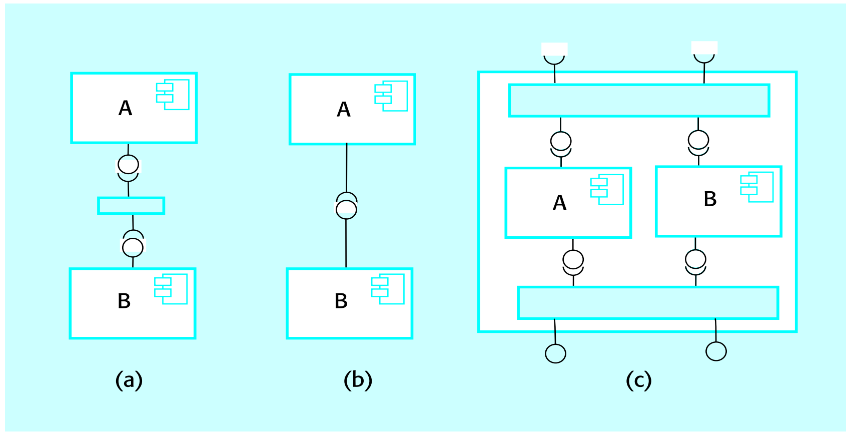
Appendix A Error messages and recovery procedures
Appendix B Glossary
Appendix C Index (for manuals of 40 pages or more)

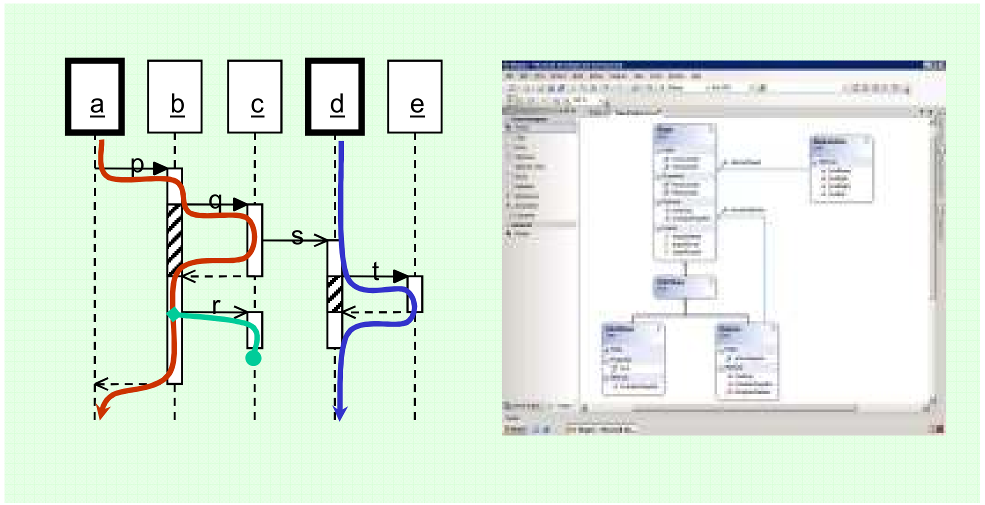
How to perform CB software design



How to perform CB software design

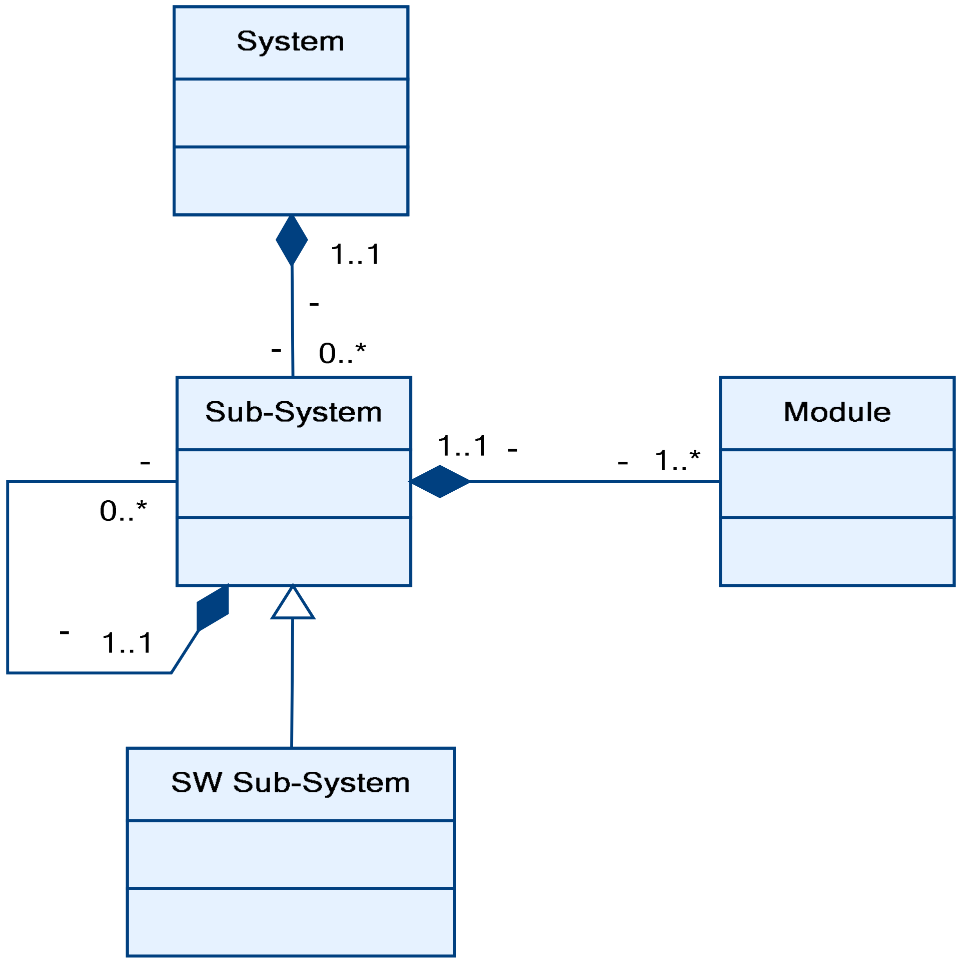






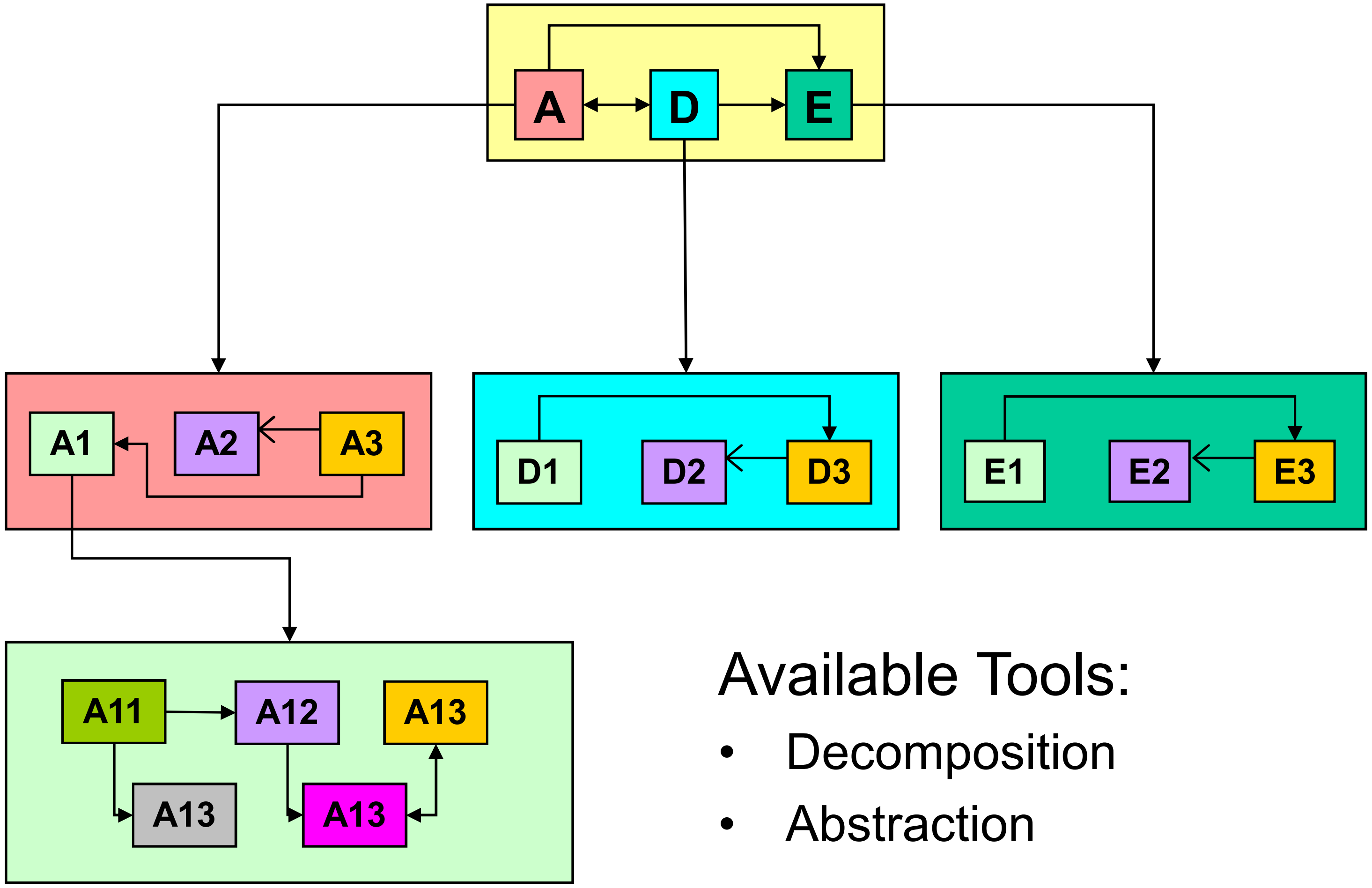
How to perform Architectural Design

- Software Architectural Design is concerned with decomposing a software sub-system into further interacting sub-systems.



- Sw Architectural Design = sub-systems engineering

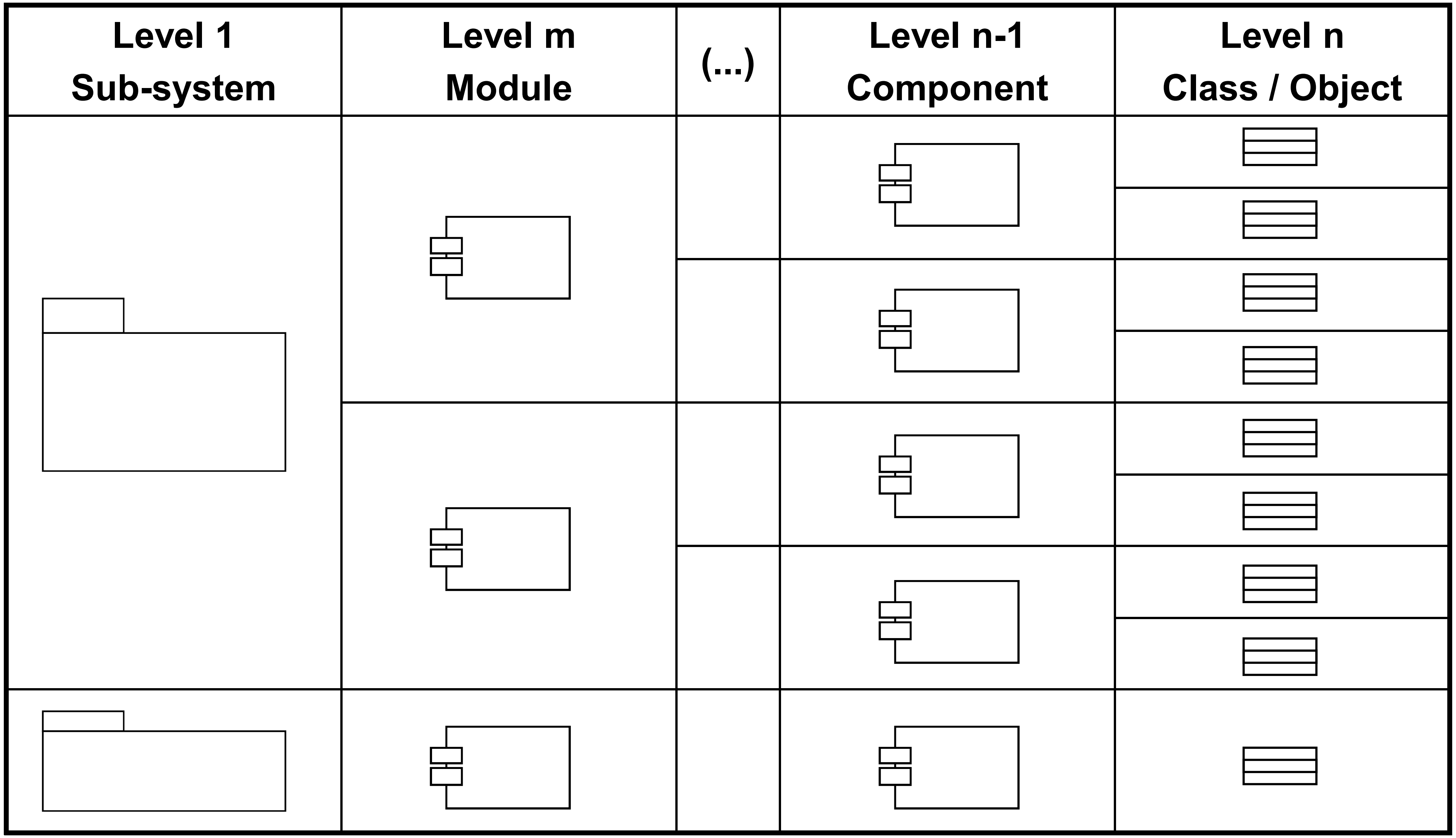
How to perform Architectural Design: Sub-system / module Decomposition



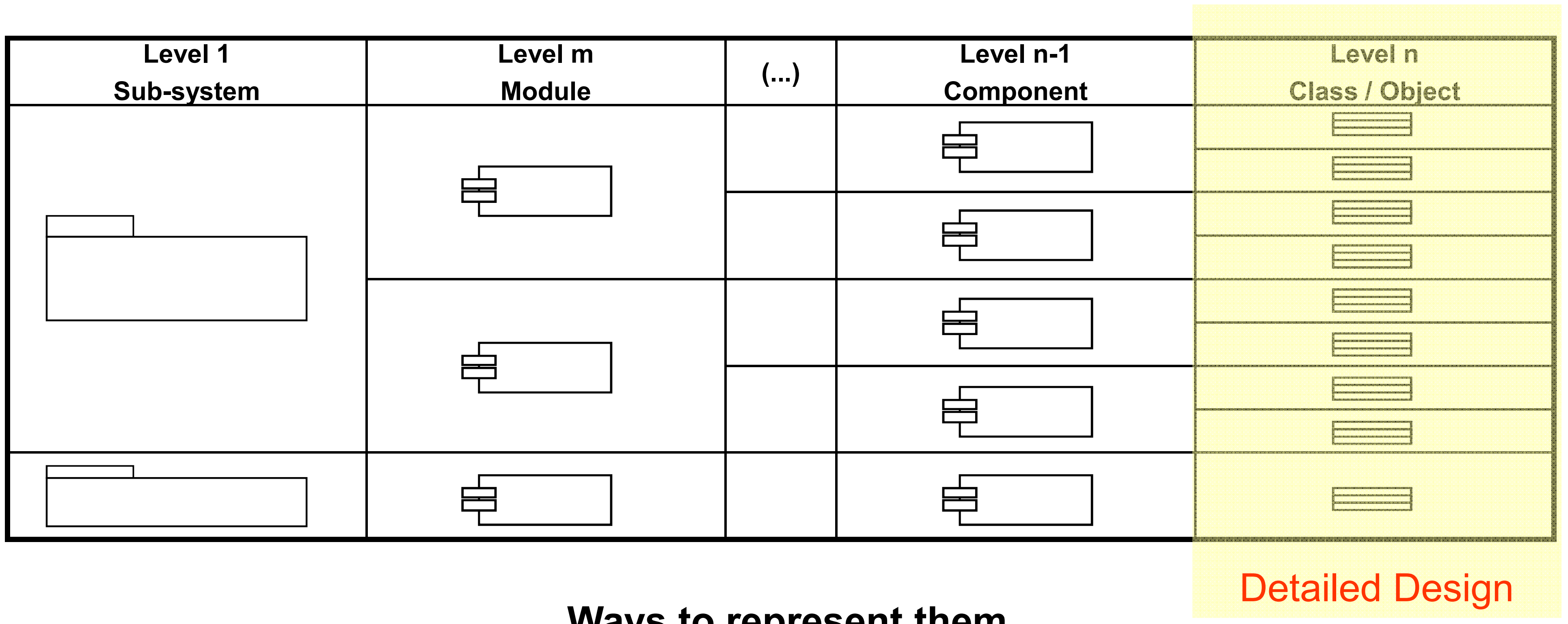
Available Tools:

- Decomposition
- Abstraction

System Decomposition Layers for OO methods



System Decomposition Layers for OO methods



Ways to represent them

Boxes Boxes Boxes | -

Box and Lines Diagrams Box and Lines Diagrams | UML Classes

UML Packages | UML Components UML Components | UML Objects

Diagrams with Packages | UMLComponents Diagrams | UML Class Diagrams

How to perform Object Oriented Detailed Design

- Detailed Design is the process of decomposing a module/system into objects.
- This decomposition can be affected by:
  - Encapsulation, granularity, dependency, flexibility, performance, evolution, reusability, etc.

    E.Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. Elements of Reusable Object Oriented Software
- The Inputs to OO Detailed Design must be:
  - The Requirements affecting design
  - The Domain Objects and Classes found in Requirements Analysis (RA)
  - The Domain Objects and Classes collaborations defined in RA
  - The SW Architecture (including Architectural Objects and classes when they exist)
  - The Modules/sub-systems solution principles if they were defined in the Architectural Design

How to perform Object Oriented Detailed Design

- The Outputs to be produced in an OO Detailed Design must be
  - The objects structure
  - Their corresponding classes structure (interfaces, Attributes, Operations)
  - The collaborations between the objects (messages, signals,..)
  - The definition of the possible states of the objects
  - The organization of the objects in components and nodes
  - The way the structures, states and collaborations can be tested

How to perform Object Oriented Detailed Design

- Therefore, the problem in detailed design can be located in

“*how to organize, integrate and adapt the domain and architectural objects, as well as their structure, interfaces, operative, collaborations and states*”.

- Many problems, specific to Object Orientation, may oblige to include new specific objects (Detailed Design Objects) to solve them.

- Many of these problems have already been solved and the objects/classes structures are available for us designers..

They are forming the Design Patterns

Identified problems in Object Oriented Detailed Design

- Not setting the focus on the collaboration
- Finding Appropriate Objects
- Determining Object Granularity
- Specifying Object Implementations
  - Class versus type: Specifying object interfaces
  - Programming to an Interface / Programming to an Implementation
  - The creation of objects
  - Inheritance versus Composition
  - Delegation
  - Inheritances versus Parameterized Types
  - The structures of Objects
  - The behavior of Objects
  - Coupling
- Relating Run-Time and Compile-Time structures
- Designing for Change

E.Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. Elements of Reusable Object Oriented Software

Find Appropriate Objects

- Many objects in a design come from the analysis model. But object-oriented designs often end up with classes that have no counterparts in the real world. Some of these are low-level classes like arrays. Others are much higher-level.
  - By using Design Patterns, for example, the Composite pattern introduces an abstraction for treating objects uniformly that doesn't have a physical counterpart. Strict modeling of the real world leads to a system that reflects today's realities but not necessarily tomorrow's. The abstractions that emerge during design are key to making a design flexible.

E.Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. Elements of Reusable Object Oriented Software

Determining Object Granularity

- Objects can vary tremendously in size and number. They can represent everythink down to the hardware or all the way up to entire applications
  - Design patterns help you identify less-obvious abstractions and the objects that can capture them. For example, objects that represent a process or algorithm don't occur in nature, yet they are a crucial part of flexible designs. The Strategy (315) pattern describes how to implement interchangeable families of algorithms. The State (305) pattern represents each state of an entity as an object. These objects are seldom found during analysis or even the early stages of design; they're discovered later in the course of making a design more flexible and reusable.

E.Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. Elements of Reusable Object Oriented Software