Universitas Esa Unggul

Smart, Creative and Entrepreneurial

CPL230-PENGEMBANGAN PERANGKAT LUNAK (PERTEMUAN-6)

www.esaunggul.ac.id

Dosen Pengampu :

5165-Kundang K Juman

Prodi Teknik Informatika Fakultas Ilmu Komputer

Requirements Analysis and Specification

Based on presentations by G. Mussbacher, G.V Bochmann, N. Niu, with material from: Wiegers: Software Requirements, Amyot 2008-2009, Somé 2008

RE process model

(suggested by Bray)

Again, this diagram shows

- RE activities (elicitation, analysis, specification, HMI design)
- subsequent design activity (internal design)
- RE documents (requirements, specification, HMI specification)

Important point:

Distinction between

- Problem domain (described by requ. doc.)
- System (to be built) (described by spec. doc.)

Note: one has to distinguish between current (problematic) version of the problem domain, and the projected future version which includes the system to be built.

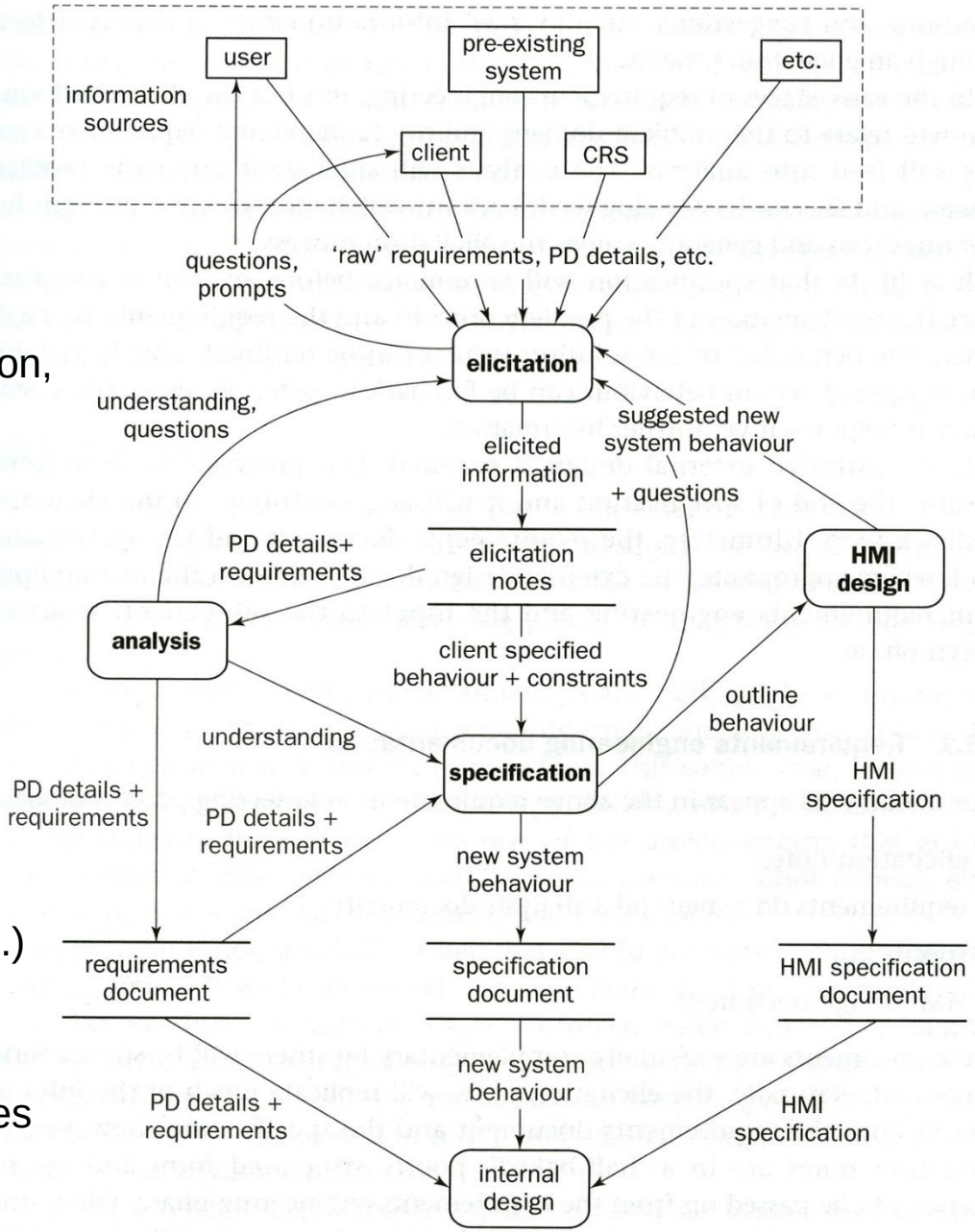


Figure 2.1
Note: CRS – client requirements specification

What is Requirements Analysis

The process of studying and analyzing the customer and the user needs to arrive at a definition of the problem domain and system requirements

Objectives

- Discover the boundaries of the new system (or software) and how it must interact with its environment within the new problem domain
- Detect and resolve conflicts between (user) requirements
- Negotiate priorities of stakeholders
- Prioritize and triage requirements
- Elaborate system requirements, defined in the requirement specification document, such that managers can give realistic project estimates and developers can design, implement, and test
- Classify requirements information into various categories and allocate requirements to sub-systems
- Evaluate requirements for desirable qualities

Questions

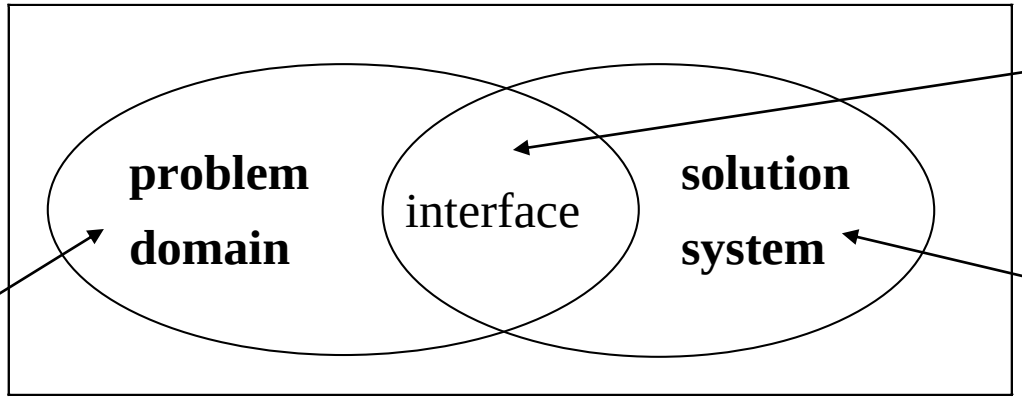
- We have seen how to specify requirements in terms of structure, standards, and writing rules, but:
  - How to identify the real problems to solve in the elicitation results?
  - How to detect conflicting aspects?
  - How to negotiate to resolve conflicts?
  - How to decide what is important and a priority?
  - How to ensure that nothing is forgotten?
  - How to validate that the findings of the analysis are good?
  - How to use models in this context?

How to Find the Real Problems?

- Ask: Why?
  - Root cause analysis
  - Determine (recursively) the factors that contribute to the problem(s) found by stakeholders
- The causes do not all have the same impact nor the same weight
  - Some may perhaps not deserve to be corrected, at least for the moment
- Goal-oriented modeling can help understand these causes and their relationships
- This analysis identifies problems that need to be solved

What is Requirements Specification?

- The invention and definition of the behavior of a new system (solution domain) such that it will produce the required effects in the problem domain
- Start from a knowledge of the problem domain and the required effects determined by elicitation and analysis
- Generally involves modeling
- Results in the specification document
  - Precise expression of requirements, may include models



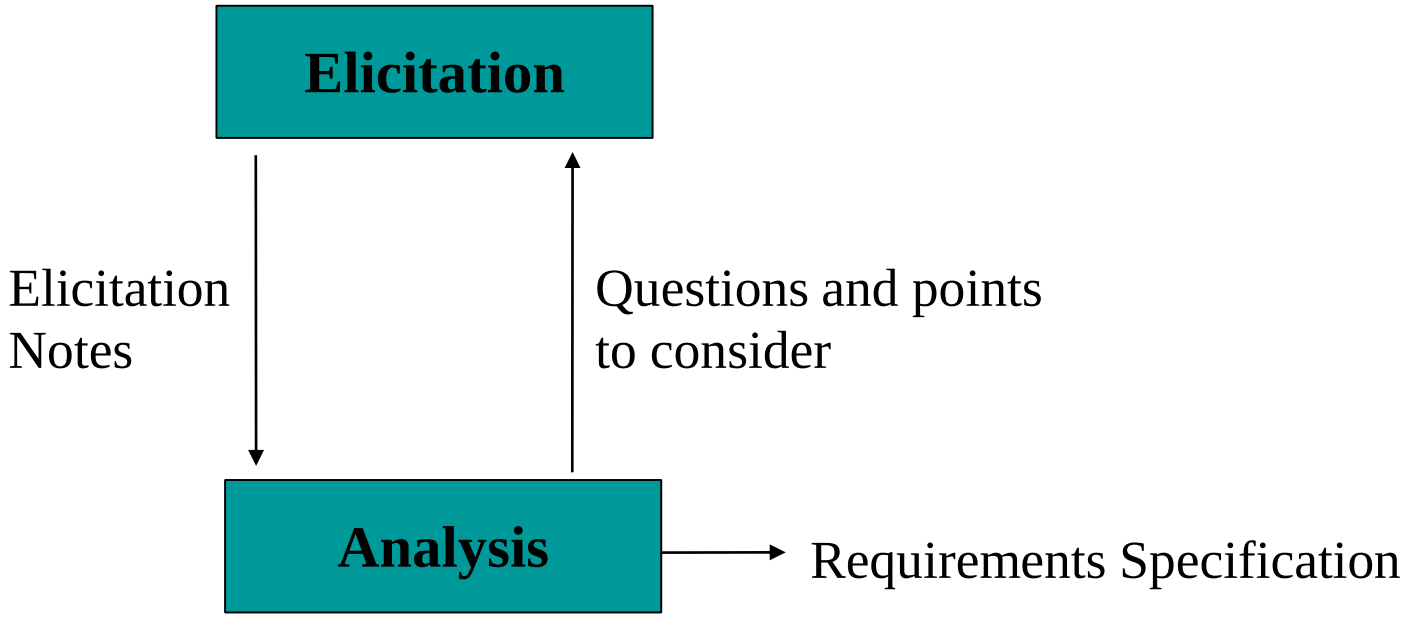
- Specification
  - description of Solution System needed to satisfy Problem Domain

- Domain properties
- Requirements

- Hardware
- Software

Requirements Analysis

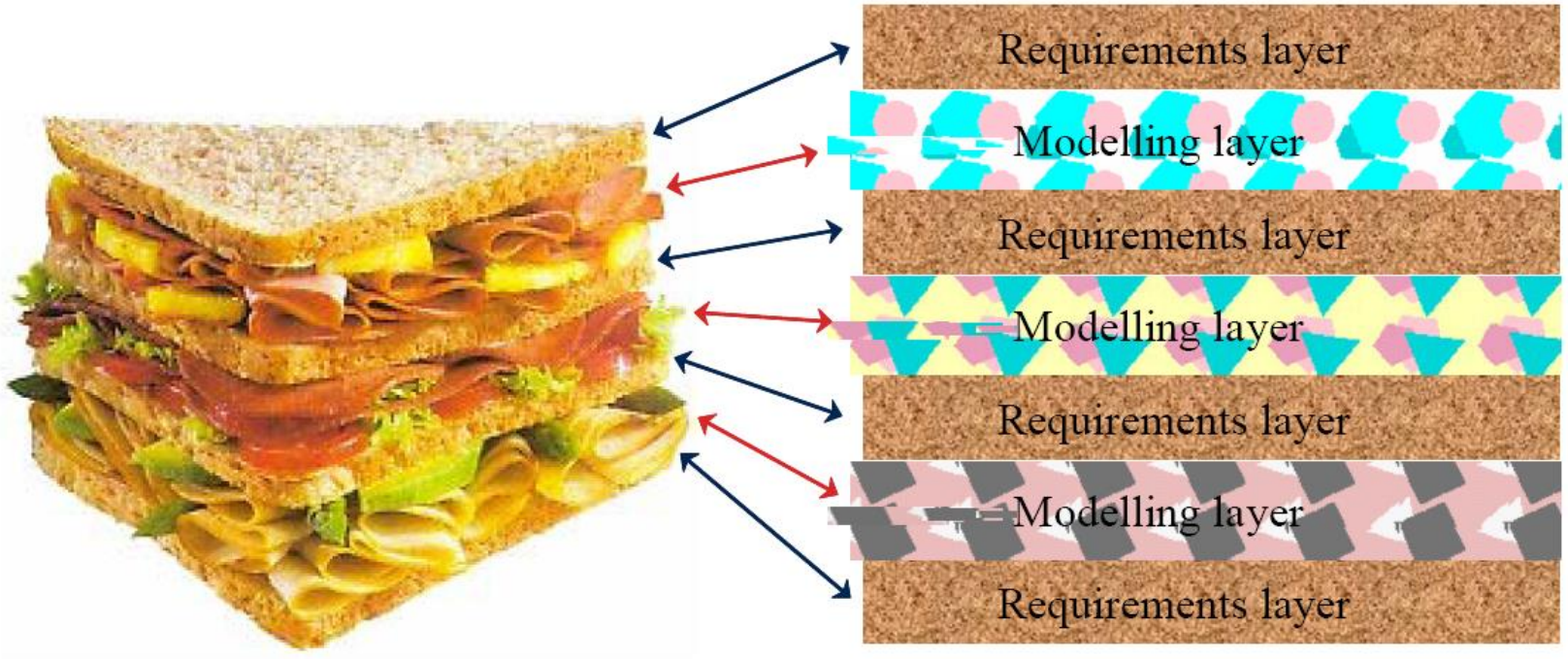
- Problem analysis
  - Development of product vision and project scope
- Analysis and elicitation feed each other



- Analysis goes hand-in-hand with modeling

Requirements Modeling

- Elicitation/analysis and modeling are intermixed



Source: http://www.telelogic.com/download/paper/SystemsEngineeringSandwich.pdf

Requirements Modeling

This is an essential task in specifying requirements

- Map elements obtained by elicitation to a more precise form
- Help better understand the problem
- Help find what is missing or needs further discussion
- Different modeling languages
  - Informal:
    natural language
  - Goal-oriented modeling (GRL)
  - Functional modeling:
    UML (Unified Modeling Notation)
    SDL (Specification and Description Language)
    Logic, e.g. Z, VDM (Vienna Development Method)
    UCM (Use Case Maps)
    ...

Requirements Verification and Validation

- Need to be performed at every stage during the (requirements) process
  - Elicitation
    - Checking back with the elicitation sources
    - “So, are you saying that . . . . . ?”
  - Analysis
    - Checking that the domain description and requirements are correct
  - Specification
    - Checking that the defined system requirement will meet the user requirements under the assumptions of the domain/environment
    - Checking conformity to well-formedness rules, standards…

Structuring requirements

Requirements Classification

In order to better understand and manage the large number of requirements, it is important to organize them in logical clusters

- It is possible to classify the requirements by the following categories (or any other clustering that appears to be convenient)
  - Features
  - Use cases
  - Mode of operation
  - User class
  - Responsible subsystem
- This makes it easier to understand the intended capabilities of the product
- And more effective to manage and prioritize large groups rather than single requirements

Requirements Classification – Features

- A Feature is
  - a set of logically related (functional) requirements that provides a capability to the user and enables the satisfaction of a business objective
- The description of a feature should include[1]
  - Name of feature (e.g. Spell check)
  - Description and Priority
  - Stimulus/response sequences
  - List of associated functional requirements

[1] Source: K.E. Wiegers

Requirements Classification – Feature Example (1)

- 3.1 Order Meals
  - 3.1.1 Description and Priority
    - A cafeteria Patron whose identity has been verified may order meals either to be delivered to a specified company location or to be picked up in the cafeteria. A Patron may cancel or change a meal order if it has not yet been prepared.
    - Priority = High.
  - 3.1.2 Stimulus/Response Sequences
    - Stimulus: Patron requests to place an order for one or more meals.
    - Response: System queries Patron for details of meal(s), payment, and delivery instructions.
    - Stimulus: Patron requests to change a meal order.
    - Response: If status is “Accepted,” system allows user to edit a previous meal order.

Source: K.E. Wiegers

Requirements Classification – Feature Example (2)

- Stimulus: Patron requests to cancel a meal order.
- Response: If status is “Accepted, ”system cancels a meal order.

– 3.1.3 Functional Requirements
  - 3.1.3.1. The system shall let a Patron who is logged into the Cafeteria Ordering System place an order for one or more meals.
  - 3.1.3.2. The system shall confirm that the Patron is registered for payroll deduction to place an order.
  - .....

Source: K.E. Wiegers

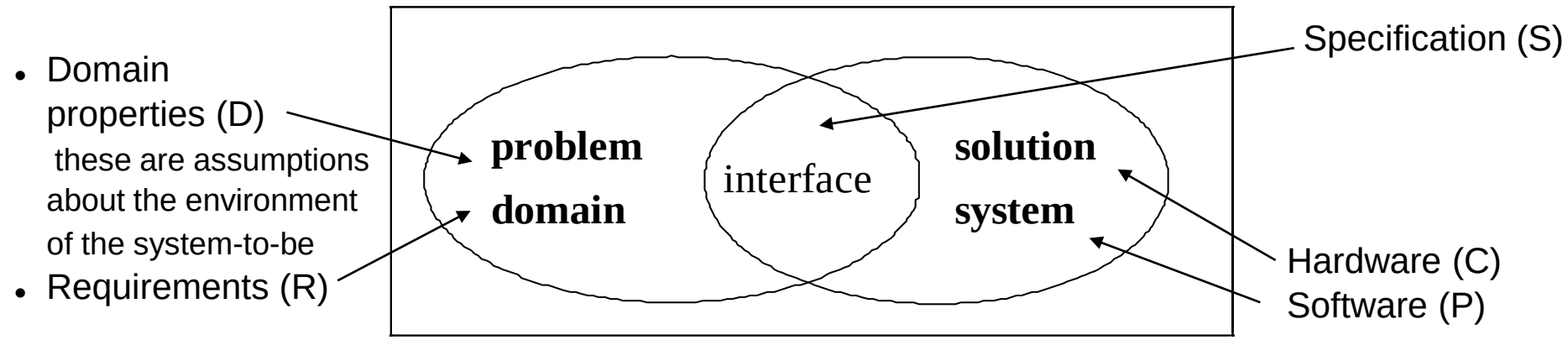
External design

Requirements Specification = External Design

- Requirements Specification is «The invention and definition of the behavior of a new system (solution domain) such that it will produce the required effects in the problem domain »
- During Requirements Analysis, one finds the existing properties of the problem domain, as well as the requirements that should be satisfied in the domain-to-be. We assume that the domain-to-be will include a new system-to-be-built and other aspects of the domain may also be changed.
- The determination of this domain-to-be, including the system-to-be) is a typical design process. It is called external design because the system-to-be is considered during this process as a black-box and the external environment of it is designed (in a white-box manner) – the domain-to-be.

The World and the Machine[1]

(or the problem domain and the system)



- **Validation question** (do we build the right system?) : if the domain-to-be (excluding the system-to-be) has the properties D, and the system-to-be has the properties S, then the requirements R will be satisfied.

$$D \text{ and } S \Rightarrow R$$

- **Verification question** (do we build the system right?) : if the hardware has the properties H, and the software has the properties P, then the system requirements S will be satisfied.

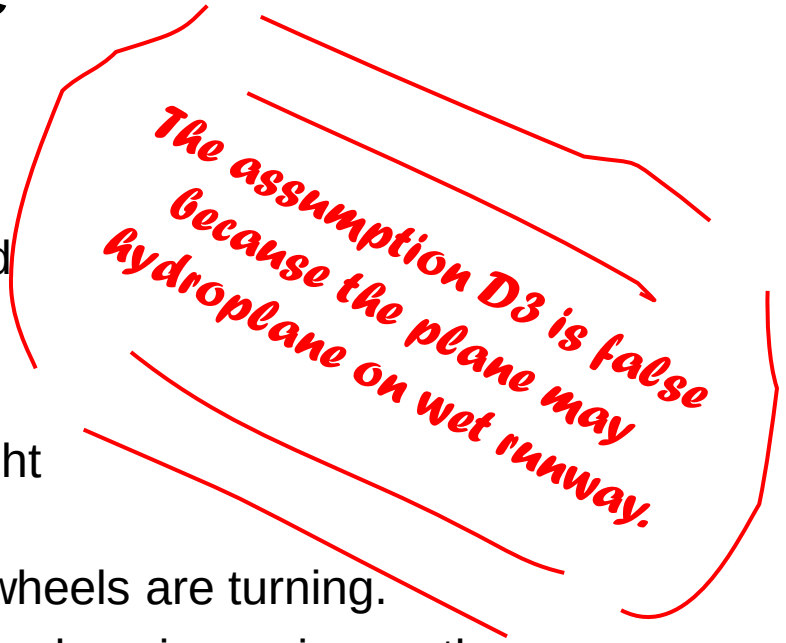
$$C \text{ and } P \Rightarrow S$$

- Conclusion:

$$D \text{ and } C \text{ and } P \Rightarrow R$$

[1] M. Jackson, 1995

Example



- Requirement
  - (R) Reverse thrust shall only be enabled when the aircraft is moving on runway.
- Domain Properties
  - (D1) Deploying reverse thrust in mid-flight has catastrophic effects.
  - (D2) Wheel pulses are on if and only if wheels are turning.
  - (D3) Wheels are turning if and only if the plane is moving on the runway.
- System specification
  - (S) The system shall allow reverse thrust to be enabled if and only if wheel pulses are on.
- Does D1 and D2 and D3 and S $\Rightarrow$ R?
  - Are the domain assumptions (D) right? Are the requirement (R) or specification (S) what is really needed?

based on P. Heymans, 2005

Requirement specifications including assumptions

- Often the requirements for a system-to-be include assumptions about the environment of the system.
- The system specification S, then, has the form:

  $S = A \Rightarrow G$

  where A are the assumptions about the environment and G are the guarantees that the system will provide as long as A hold.
- If these assumptions (A) are implied by the known properties of the domain (D), that is $D \Rightarrow A$, and we can check that the domain properties (D) and the system guarantees (G) imply the requirements (R), that is D and $G \Rightarrow R$, then the “validation condition” D and $S \Rightarrow R$ is satisfied.

Specification with assumptions and guarantees (example)

Example: A power utility provides electricity to a client. The problem is that the monthly invoice is not related to the electricity consumption, because there is no information about this consumption.

- Idea of a solution: introduce an electricity counter.
- **Specification of the electricity counter**
  - **Inputs and outputs**
    - input power from utility (voltage, current) – voltage supplied by utility
    - output power to client (voltage, current) – current used by client
    - Reset button (input)
    - consumption (output - watt-hours of electricity consumption)

Example (suite)

- **Assumptions**
  - Input voltage < 500 Volts (determined by utility)
  - Output current < 20 Amps (determined by client)
- **Guarantees**
  - Output voltage = input voltage
  - Input current = output current
  - Consumption output shall indicate the consumption since the last reset operation, that is, the integral of (output voltage x output current) over the time period from the occurrence of the last reset operation to the current time instant.

- **Software example**
- Specification of a method providing the interface “List search(Criteria c. **Assumption:** c is a data structure satisfying the Criteria class properties. **Guarantee:** the returned result is a list satisfying the List class properties and includes all items from the database that satisfy c.

Formal Verification and Validation

- Evaluating the satisfaction of “D and S $\Rightarrow$ R” is difficult with natural language
  - Descriptions are verbose, informal, ambiguous, incomplete...
  - This represents a risk for the development and organization
- Verification of this “validation question” is more effective with formal methods (see below)
  - Based on mathematically formal syntax and semantics
  - Proving can be tool-supported
- We can also reduce this risk by using semi-formal modeling

The Use of Models

Models

- According to B. Selic, a model is a reduced representation (simplified, abstract) of (one aspect of) a system used to:
  - Help understand complex problems and / or solutions
  - Communicate information about the problem / solution
  - Direct implementation
- Qualities of a good model
  - Abstract
  - Understandable
  - Accurate
  - Predictive
  - Inexpensive

Modeling Notations

Natural language (English)

- \+ No special training required
- \- Ambiguous, verbose, vague, obscure ...
- \- No automation

Ad hoc notation (bubbles and arrows)

- \+ No special training required
- \- No syntax formally defined
- \- meaning not clear, ambiguous
- \- No automation

Semi-formal notation (URN, UML...)

- \+ Syntax (graphics) well defined
- \+ Partial common understanding, reasonably easy to learn
- \+ Partial automation
- \- Meaning only defined informally
- \- Still a risk of ambiguities

Formal notation (Logic, SDL, Petri nets, FSM ...)

- \+ Syntax & semantics defined
- \+ Great automation (analysis and transformations)
- \- More difficult to learn & understand

Modeling notations (2)

- Informal language is better understood by all stakeholders
  - Good for user requirements, contract
  - But, language lacks precision
  - Possibility for ambiguities
  - Lack of tool support
- Formal languages are more precise
  - Fewer possibilities for ambiguities
  - Offer tool support (e.g. automated verification and transformation)
  - Intended for developers

Source (for decision table): Easterbrook and Callahan, 1997

Modeling Structure

Concepts of Entities and their Relationships. Use one of the following notations:

- ERD (Entity Relationship Diagram – the traditional version)
- UML class diagrams
- Relational tables

- Can be used for the following
  - Model of the problem domain (called “domain model”)
    - The two versions: existing and to-be
  - Model of input and output data structures of system-to-be
  - Model of the stored data (database)
    - not necessarily an image of the domain data
    - Additional data is introduced (e.g. user preferences)
  - Architectural design of the system-to-be

Modeling inputs and outputs

- Nature of inputs and outputs:
  - IO related to problem (problem data)
  - Additional data related to solution (solution data)
    - E.g., prompts, user options, error messages…
- Collected in Data Dictionary using
  - Plain text (natural language)
  - EBNF
  - Code-like notations
  - Logic (e.g., VDM)
  - Structure charts
  - …
- Graphical output (screens, forms)
  - Iconic (representational) drawings, prototype screens or forms, printouts produced by operational prototype

Modeling Dynamic Behavior

- Behavior modeling techniques
  - Text (plain, function statements, use cases)
  - Decision tables
  - Activity Diagrams / Use Case Maps
  - Finite state machines
    - Simple state machines (FSM) : use state diagrams or transition table notation
    - Extended state machines (e.g. UML State Machines – including SDL)
    - Harel's State Charts (concepts included in UML notation)
    - Petri nets (allows for flexible concurrency, e.g. for data flow, similar to Activitity Diagrams)
  - Logic (e.g. Z, VDM) for describing input-output assertions and possibly relationship to internal object state that is updated by operations)
- It is important to chose what best suits the problem

Model Analysis

- By construction
  - We learn by questioning and describing the system
- By inspection
  - Execute/analyze the model in our minds
  - Reliable?
- By formal analysis
  - Requires formal semantics (mathematical) and tools
  - Reliable (in theory), but expensive (for certain modeling approaches)
- By testing
  - Execution, simulation, animation, test...
  - Requires well-defined semantics and execution/simulation tools
  - More reliable than inspection for certain aspects
  - Possible to interact directly with the model (prototype)

Typical Modeling Approaches

- Many approaches involve modeling to get a global picture of the requirements
  - Structured Analysis (1970)
  - Object-Oriented Analysis (1990)
  - Problem Frames (1995)
  - State Machine-Based Analysis
  - Conflict Analysis
    - E.g. with mis-use cases or with GRL/UCM models and strategies/scenarios
- It is important to distinguish between
  - Notation used for defining the model
  - Process defining a sequence of activities leading to a desired model
- Note: Analysis can be on individual requirements as well
  - Remember tips and tricks on how to write better requirements