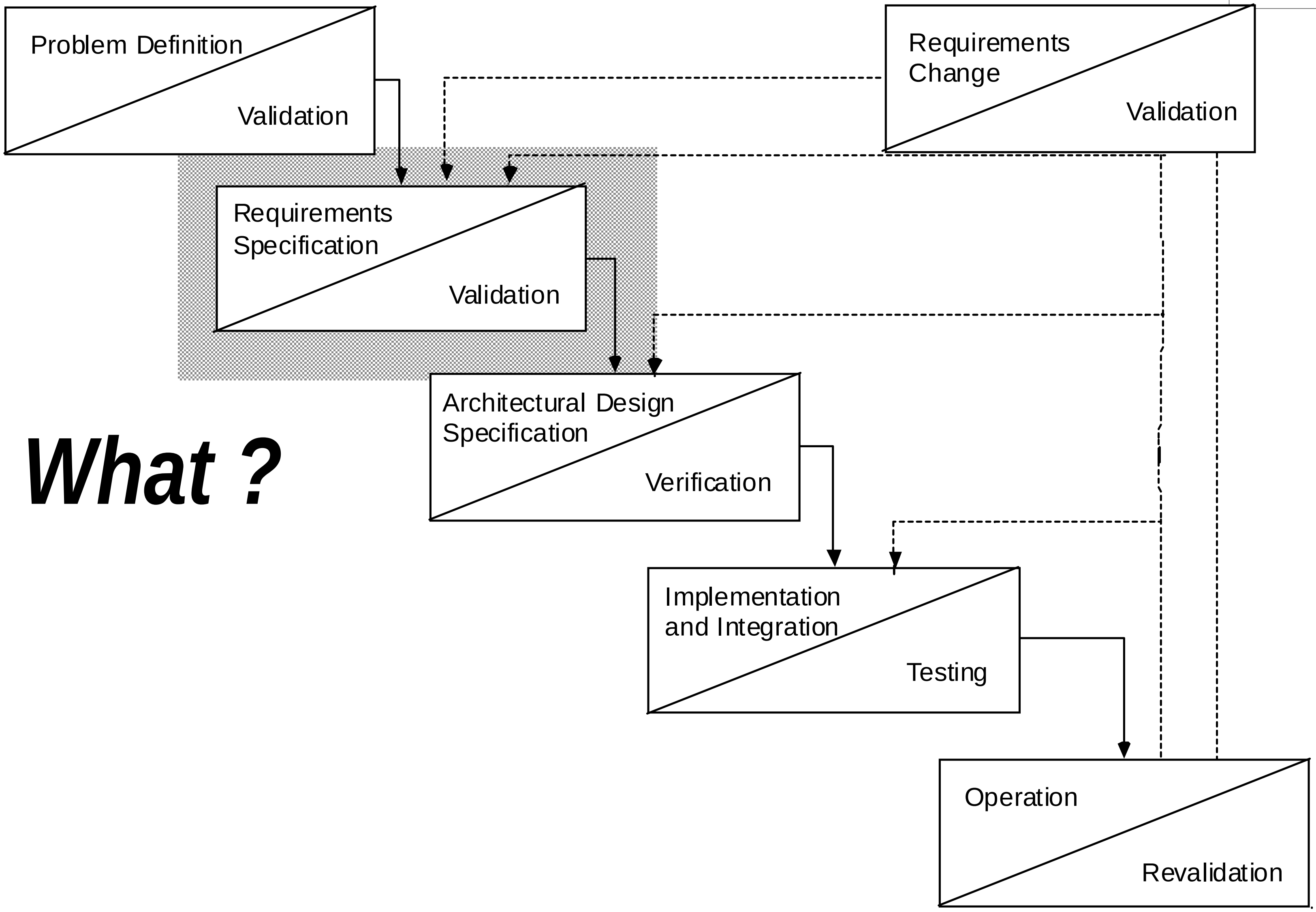
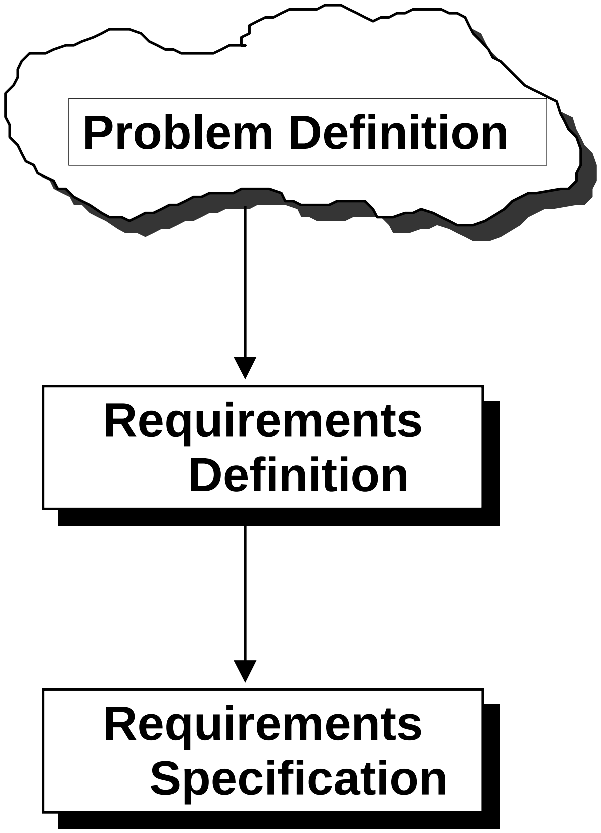
Requirements in the Software Lifecycle



What ?

Requirements Analysis and Specification



Problem Definition

informal statement of need for system

Requirements Definition

natural language statement of what system is to provide

Requirements Specification

notational and/or formal description software system

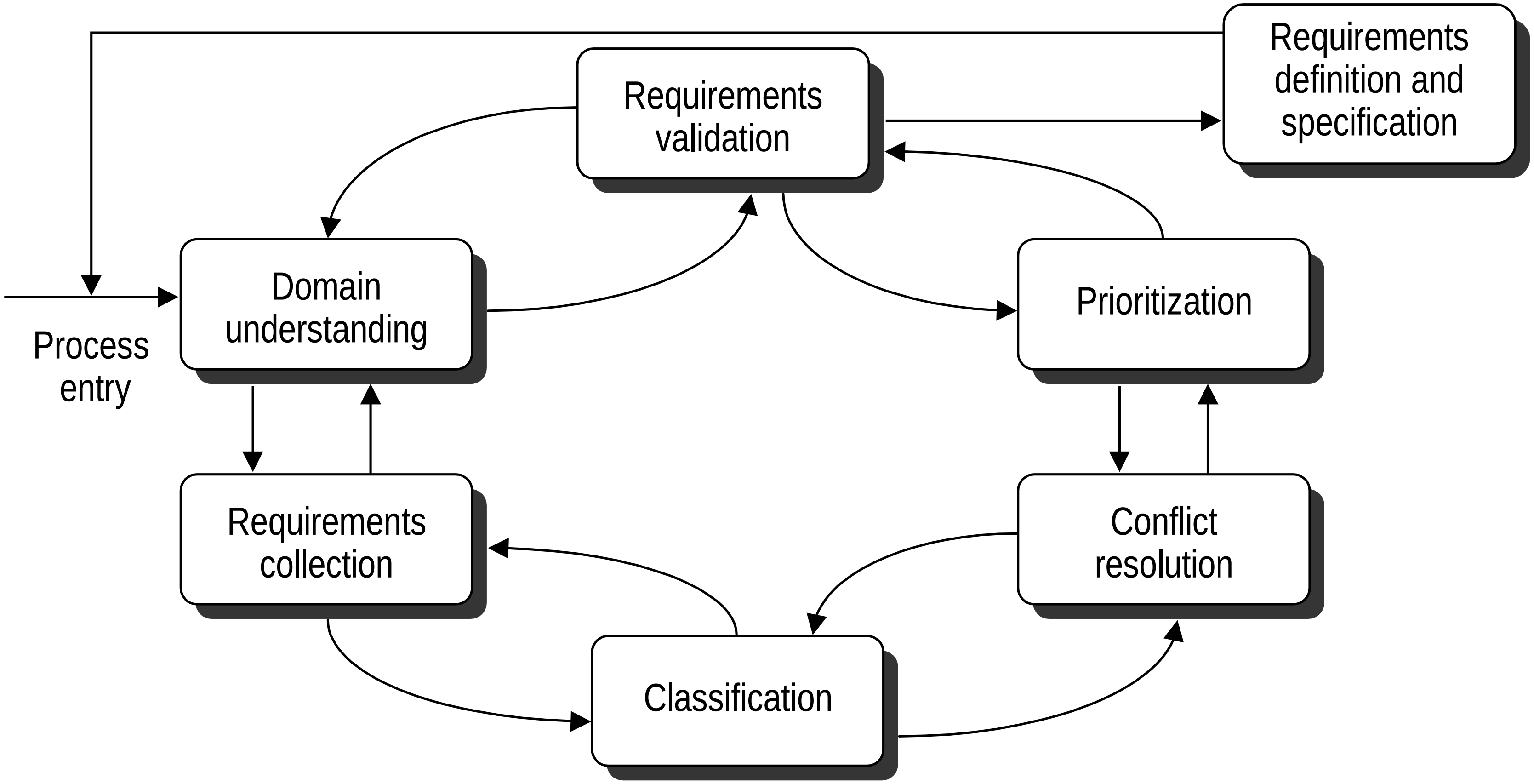
Goals and Objectives

- Understand and specify requirements from customers’ needs
- Document customers' needs before plunging into design
  - customer best knows what is wanted but usually doesn't know what can be achieved
- Determine functional requirements and non-functional constraints to meet needs
- Develop a contract between customer and developer
- Provide basis for definitive testing and verification

Goals and Objectives - 2

- Identify functional capabilities to be provided
- Identify desired responses to undesired events
- Identify non-functional and environmental constraints to be satisfied
- Avoid specifying how needs should be met
- Serve as guide to developers, testers, maintainers

Requirements Anlaysis Process



Products

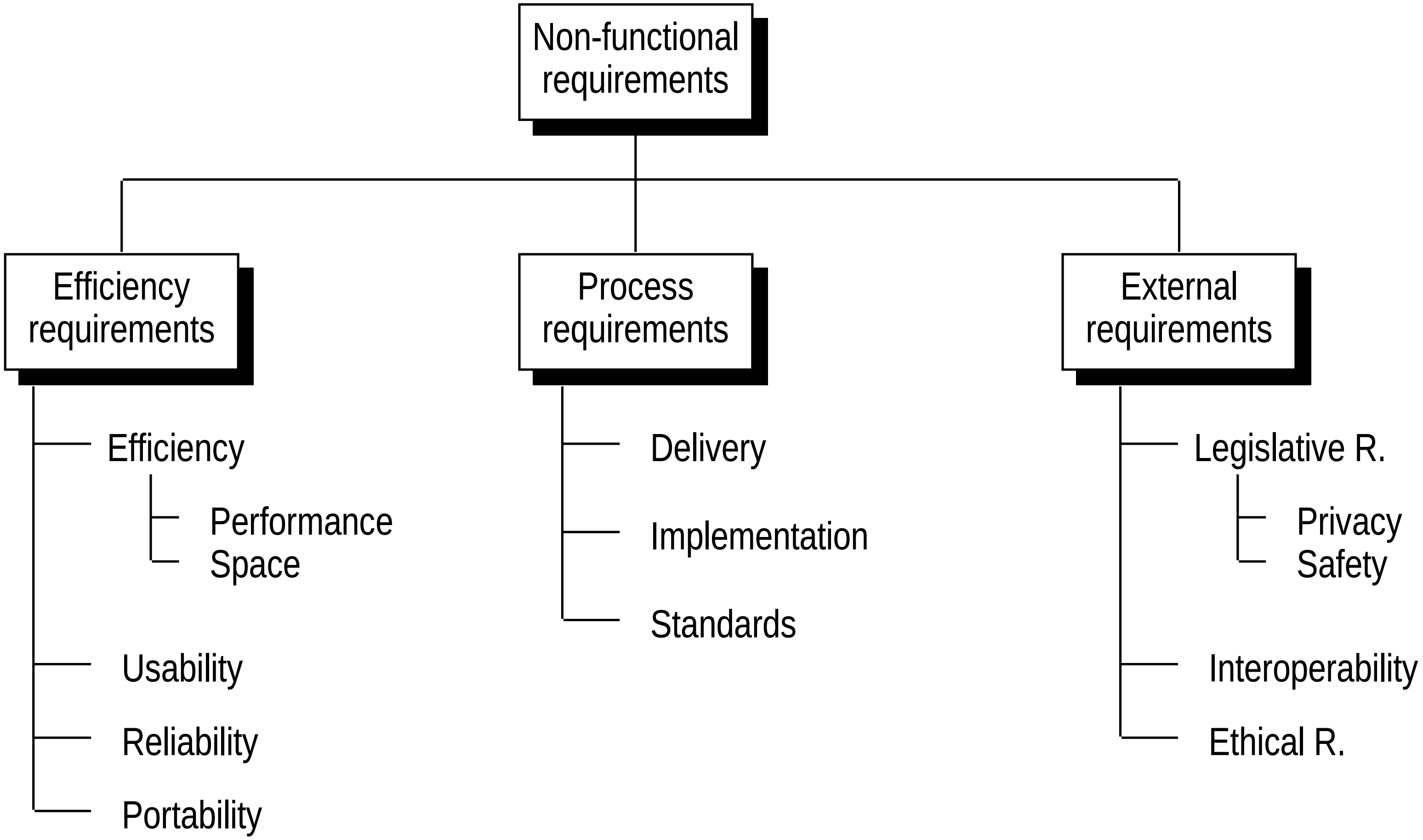
- Refinement of customer needs
- Documentation of all requirements and constraints
  - functional
  - nonfunctional
- Lifecycle considerations
- Acceptance Test Plan

Should not begin a project without a

GOOD CONTRACT

that completely describes customer expectations

Non-Functional Requirements



Desirable Characteristics of a Requirements Specification

- Abstract (one model, many realizations)
- Complete (to the extent required)
- Consistent (no contradictions)
- Unambiguous (any system that satisfies it is acceptable)
- Precise (uniquely interpretable)
- Feasible (can be satisfied within constraints)
- Even (entire document at same level of detail)
- Modifiable (living document)
- Reference Tool (readable by customer, developer, maintainer)

Desirable Characteristics of a Requirements Specification - 2

- **Concise (no extraneous details)**
- **Appropriate (not more than is needed)**
- **Verifiable (testable)**
- **No implementation bias**
  - **premature details can unduly constrain designers**
  - **take users' point of view**
    - **external, not internal, perspective**
    - **specifics have their place only when user requires it (such as if algorithm is only potential view)**

Requirements specify
***what* is to be provided**
NOT *how* it is to be provided

Common Problems

- Incompleteness
  - customer may be unavailable or inaccessible
  - customer asks for too little
  - customer doesn’t think of *everything*
  - the world changes
  - (sometimes incompleteness is okay)
- Inconsistency
  - customer may be a group that disagrees
  - different people may negotiate different parts
- Ambiguity
  - customer may be a group where noone sees the whole picture
  - difficult to spot ambiguity in large, complex applications

Common Problems - 2

- **Imprecision**
  - **customer may be a group with a different vocabulary**
  - **precision easiest in mature application areas (accounting, numerical analysis)**
  - **precision difficult in new disciplines**
- **Infeasibility**
  - **customer asks for *too much***
    - **no cenceivable algorithm**
    - **unrealistic requests**
- **Uneveness**
  - **different sources of information**
  - **different people write different parts**
  - **different parts of specification are more difficult than others**

Caution

- Shortchanging requirements phase
- Emphasizing design
- Substituting test plans for requirements

these are DEADLY

to later development

Method-based requirements analysis

- **Most widely used approach to requirements analysis**
  - Depends on the application of some structured methods to understand the system
  - Results are expressed in a set of system models
  - Methods have different emphases: some are focused exclusively on requirements elicitation/analysis, others are very close to design
- **Structured methods usually include:**
  - Process model (dataflow analysis, control scenario identification)
  - System modelling notations (diagrammatic, form-based, linguistic)
  - Rules applied to the system model
  - Design guidelines
  - Report templates

Basic Techniques

- Customer leads while developer learns, organizes, disciplines
  - helps surface ambiguity, inconsistency, incompleteness
- Interviews, investigations, questionnaires
  - state questions before answering them
  - don't let available information prejudice
  - separate concerns
- Develop glossaries to aid communication
- Describe in a (semi-)formal notation (possibly just formatted)
- Hierarchical decomposition
- System modeling (Dataflow Diagrams, Entity-Relationship Diagrams, Petri nets, State charts, etc.)

Contents

- Description
- Functionality
- Data
- Environment
- Robustness
- Security
- Safety
- Performance
- Resources

Contents (121)

- Title Page
- Summary
- Use Cases
- Deliverables
- Delivery Platform
- Tests

Testing: System Test Plan

- **Developed as part of requirements analysis and specification**
- **Basic goal is to test behavior of each specified software feature**
- **Non-functional testing of other behavioral features, qualities stated in requirements specification**
  - **load/stress testing**
  - **performance testing**
  - **reliability testing**
  - **robustness/recovery testing**
  - **storage testing**
  - **configuration testing**
  - security testing
  - safety testing
  - real-time response testing
  - documentation testing
  - usability testing
  - compatibility testing
  - installability testing

Testing: Acceptance Test Plan

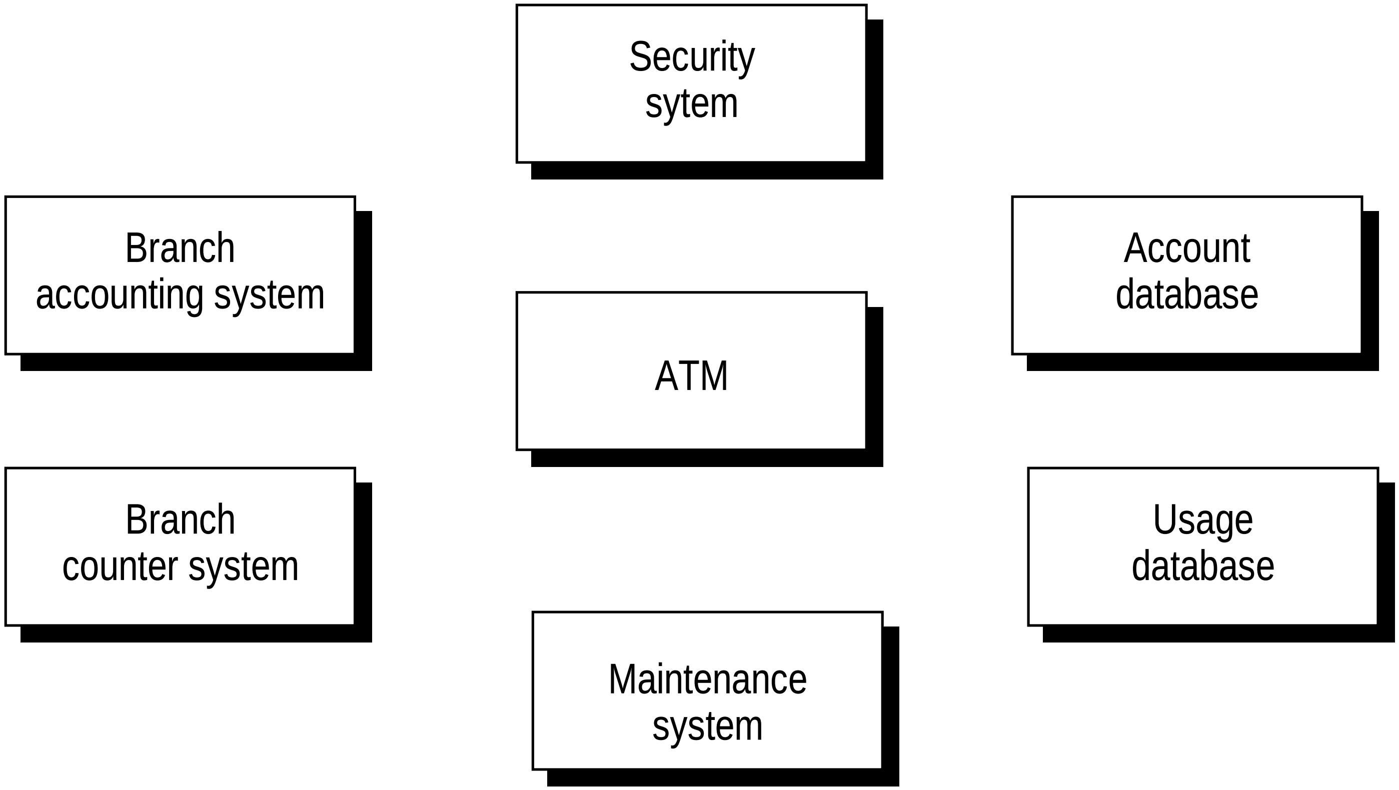
- **An operational way of determining consistency between the requirements document and the delivered system**

If the system passes the tests demanded by this plan,
then the user has ***less*** basis for complaint

- **Develop a plan for conducting tests to examine:**
  - **functional requirements**
  - **non-functional constraints**
  - **subsets**

System Contexts

- Early in the analysis process, the boundaries of the system have to be defined
- **Example:**



System Models

- **A system model is an *abstract* description of the system to be developed**
- **Particular requirements analysis methods choose a set of system models as part of the method**
  - **Different system models contribute in different ways to the understanding of the system (there is no ideal system model, nor is there an ideal method to develop such models !)**
- **Different system models are based on different approaches to abstraction (functional, data)**
- **Typical kinds of system models:**
  - **data-processing model**
  - **composition model**
  - **classification model**
  - **stimulus-response model**

System Models - 2

- **Widely used system models:**
  - ***Data-flow models:***
    - **Show how data is processed by a system**
    - **Data-flow models are basic system models of Structured Systems Analysis [DeMarco,1978]**
  - ***Semantic data models*:**
    - **Identify the data entities, their attributes, and relationships between them**
    - **Examples: Entity-Relationship Modeling [Chen,1976]**
      SDM [Hammer/McLeod,1981]
      RM/T (extension of the relational model) [Codd,1979]
  - ***Object models*:**
    - **Represent data and its processing (together with structure of the data)**
    - **Various notations: [Booch,1994]**
      [Coad/Yourdon,1990]
      [Rumbaugh et al.,1991]
      [Coleman et al.,1994]
      [Fowler et al. 1997]

Requirements Specification

- **Structured natural language**
  - **Extended, more detailed form of textual requirements definition**
  - **Advantage:**
    - **Uses expressiveness and understandability of natural language**
  - **Problems:**
    - **Inherent ambiguity of natural language**
    - **Requirements are not partitioned effectively by the language itself (it´s difficult to find related requirements)**
  - **Example:** **Usage of standard forms**

Function	Add node
Description	Adds a node to an existing design. The user selects the type of node, and its position. [....]
Inputs	Node type, Node position, Design identifier
[........]	

Requirements Specification - 2

- **Requirements/Program description languages**
  - **A PDL is a language derived from a programming language (e.g. Ada). It may contain additional, more abstract constructs to increase its expressive power**
  - **Various special-purpose requirements specification languages have been designed, e.g. PSL/PSA [Teichrow/Hershey,1977], RSL [Alford,1977]**

Example: A PDL description of ATM operation

```
procedure ATM is

begin
  loop
    Get_card (Acc_no, PIN, Valid_card);
    if Valid_card then
      Validate_PIN (PIN, Valid_PIN);
     if Valid_PIN then
      Get_account (Acc_no, Balance); Get_service (Service);
      while a service is selected loop
          Deliver_selected_service; Get_service (Service);
      end loop;
      Return_card;
     endif;
    endif;
  end loop;
end ATM;
```

Requirements Specification - 3

- **Semi-formal/Graphical notations**
  - **Graphical notations have a loose semantics associated with the structure**
  - **Widely used, e.g. SADT [Ross,1977], SSA [DeMarco,1978] [Gane/Sarsen,1979] [Yourdon/Constantine,1979]**

- **Formal/Mathematical notations**
  - **Formal specifications base on a formal semantics (mathematical concept)**
  - **Specifications are unambiguous (reduce the arguments between customer and contractor about system functionality)**
  - **Difficult to understand for customer**
  - **Examples: Finite State Machines, Petri Nets [Peterson,1977],
    Algebraic Specifications, Z [Hayes,1987], VDM [Jones,1980]**

Prototyping

- **Mockup of software product**
  - explanation to user
  - technical exploration
  - specification development and assessment
- **Addresses problems of understanding user needs**
  - adequacy of user services
  - usability of user interface
  - incomplete and/or inconsistent requirements specification
  - system feasibility
  - analysis of alternative design decisions

Means of Requirements Acquisition

Prototyping Techniques

- Executable specification languages
  - Animation of a formal system specification to provide a system prototype
  - Problems:
    - GUI cannot be prototyped
    - Executable system usually slow and inefficient
    - Executable specifications only test functional requirements
- Very high level languages
  - Programming languages which include powerful data management facilities (simplifies program/prototype development)
  - Examples: Lisp, Prolog, Smalltalk, APL
- Fourth-generation languages
  - Powerful languages, especially in the business system domain
  - Examples: Database Query Languages (including report generator, etc.)