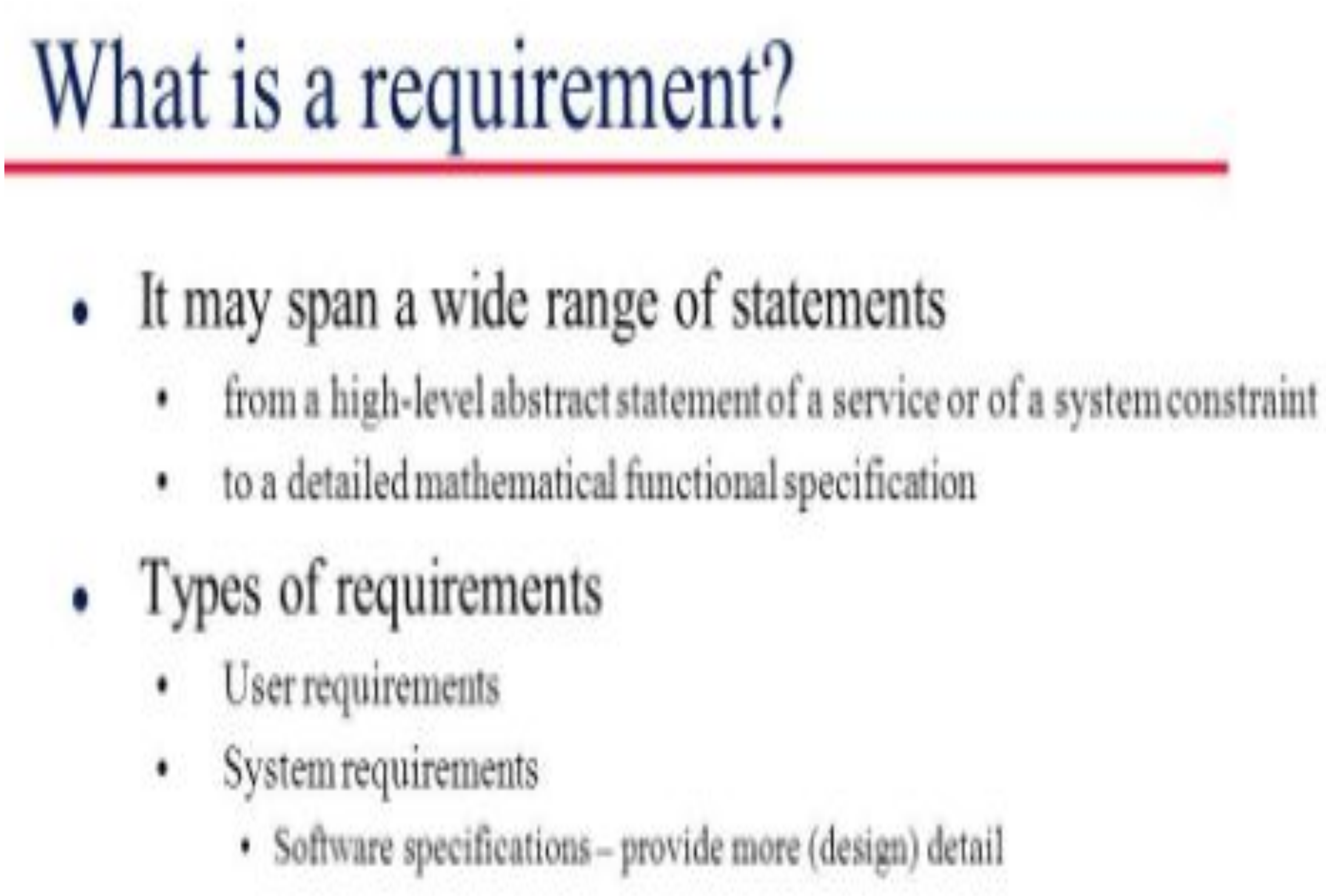
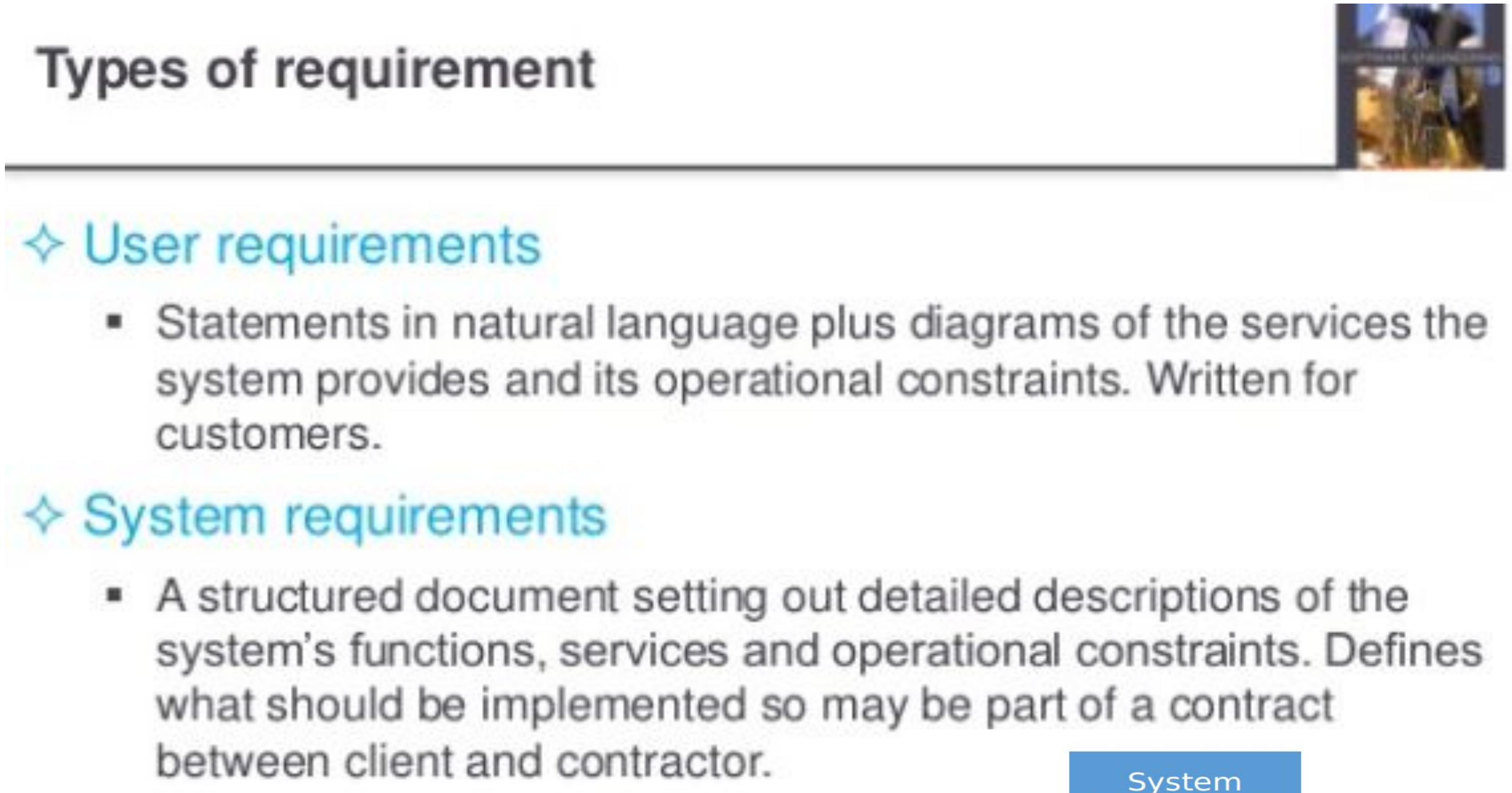
UNIT II



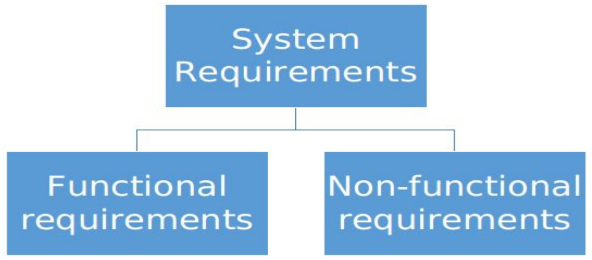
What is a requirement?

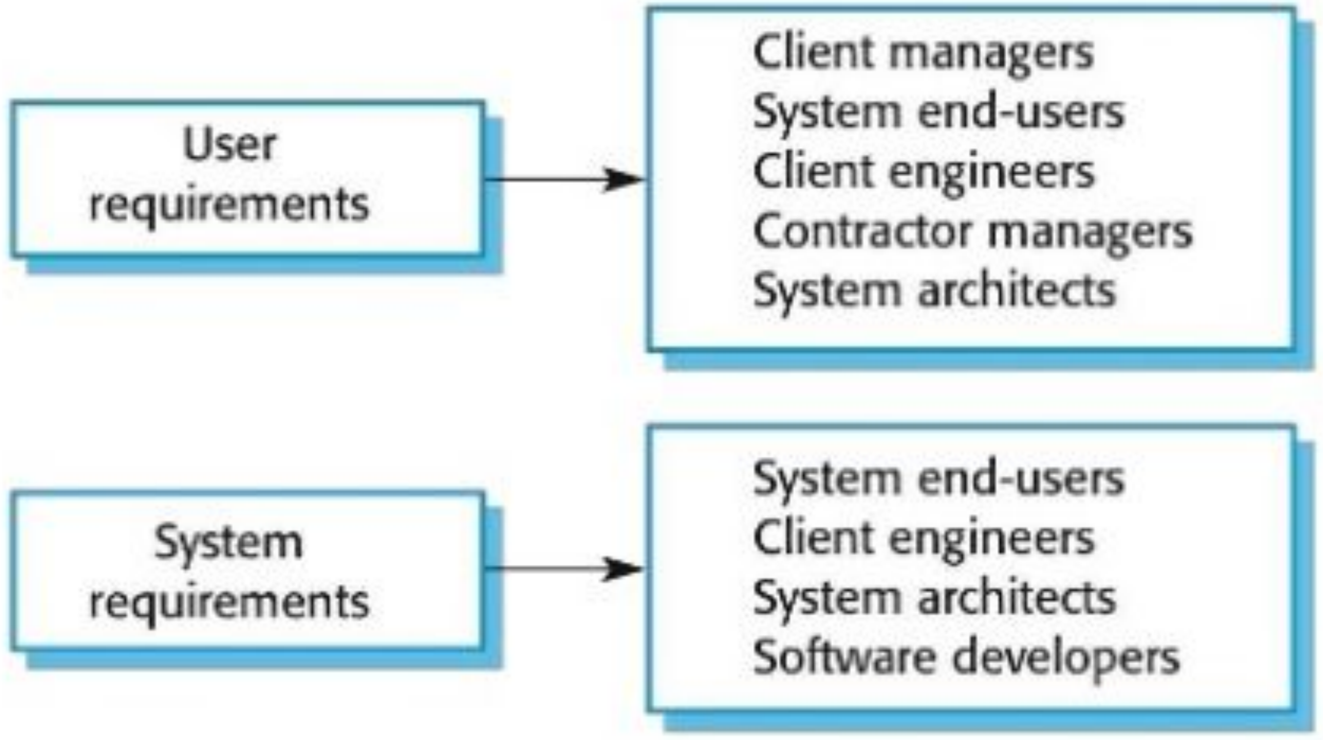
- It may span a wide range of statements
  - from a high-level abstract statement of a service or of a system constraint
  - to a detailed mathematical functional specification
- Types of requirements
  - User requirements
  - System requirements
    - Software specifications – provide more (design) detail



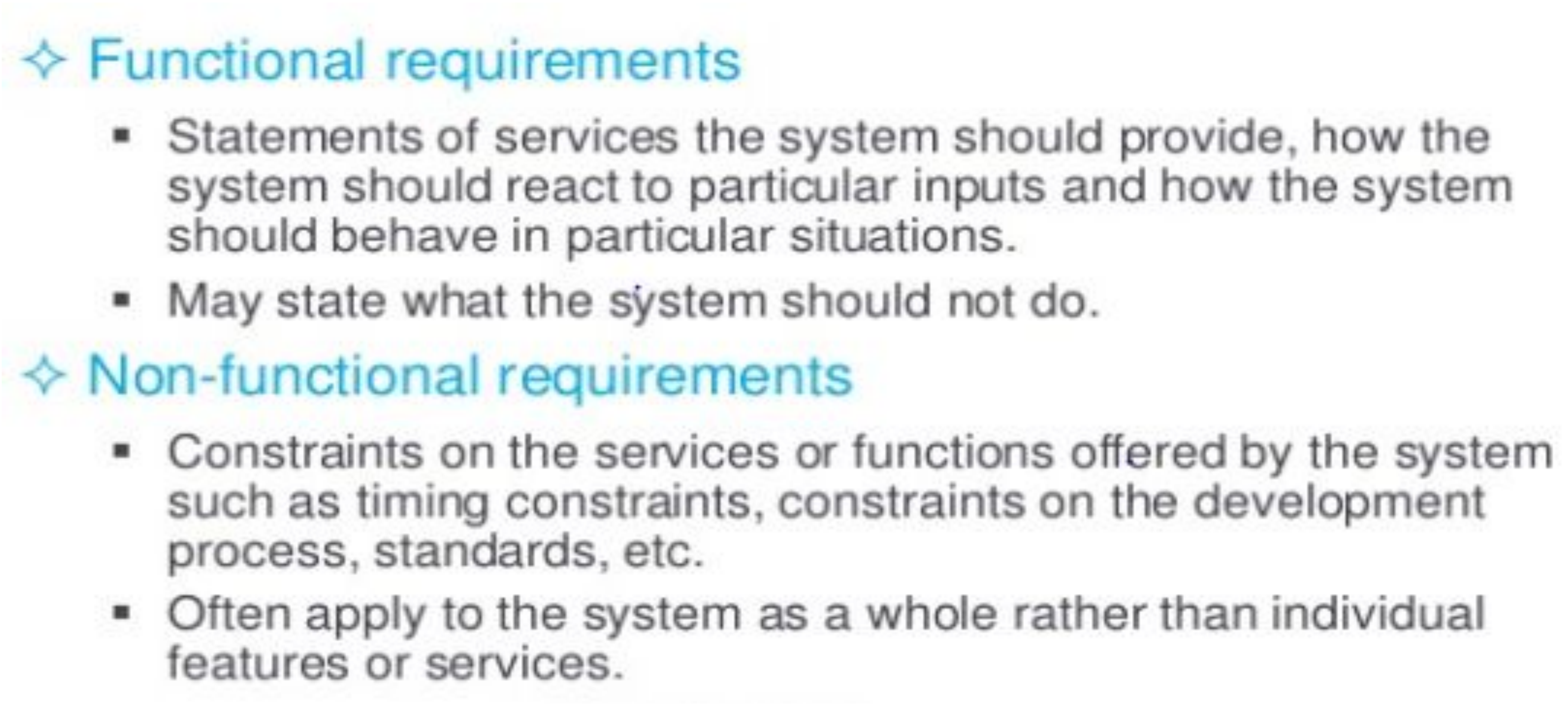
Types of requirement

- ✧ User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- ✧ System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.




User requirements
Client managers
System end-users
Client engineers
Contractor managers
System architects
System requirements
System end-users
Client engineers
System architects
Software developers

Software System Requirements:

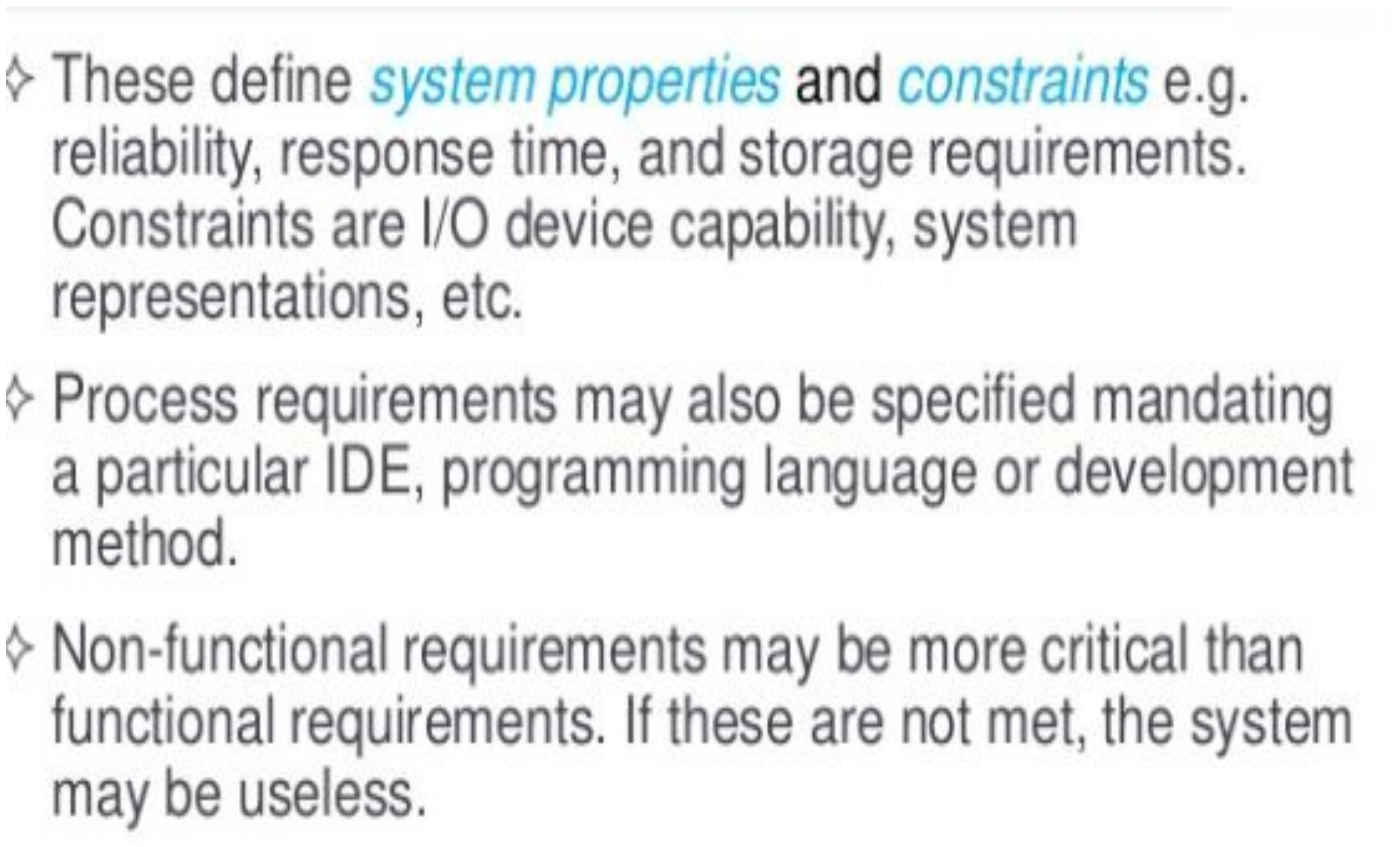


- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.

Functional Requirements

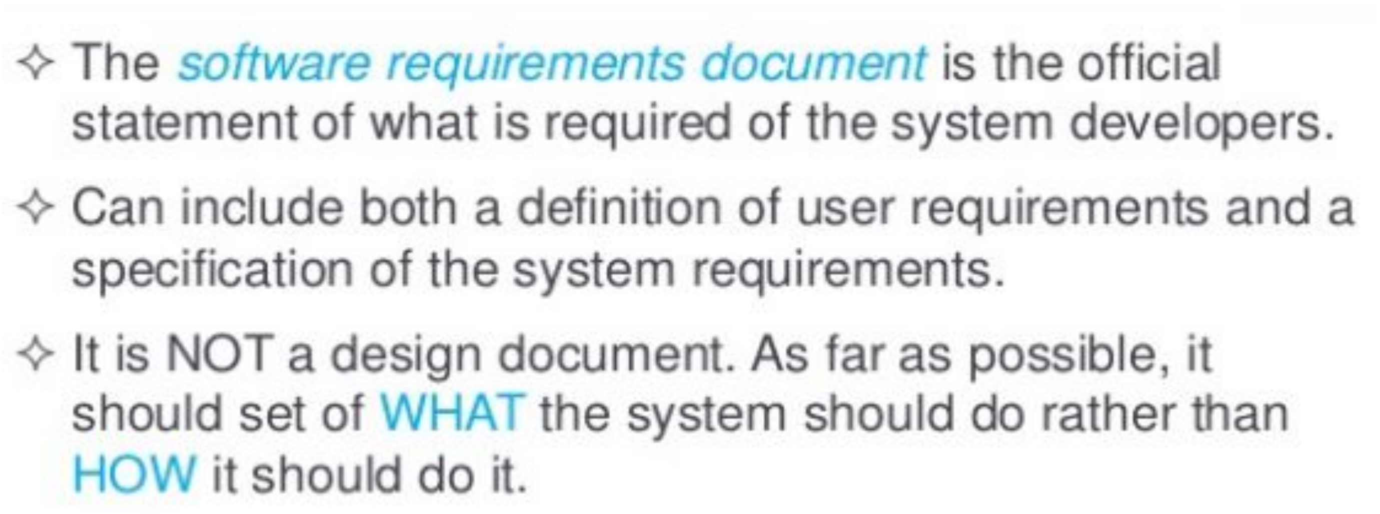
- Describe ***functionality*** or system ***services***.
- Depend on the type of software, expected users and the type of system where the software is used.
- **Functional user requirements** may be high-level statements of **what** the system should do.
- **Functional system requirements** should describe the system services in detail.

Non- Functional Requirements



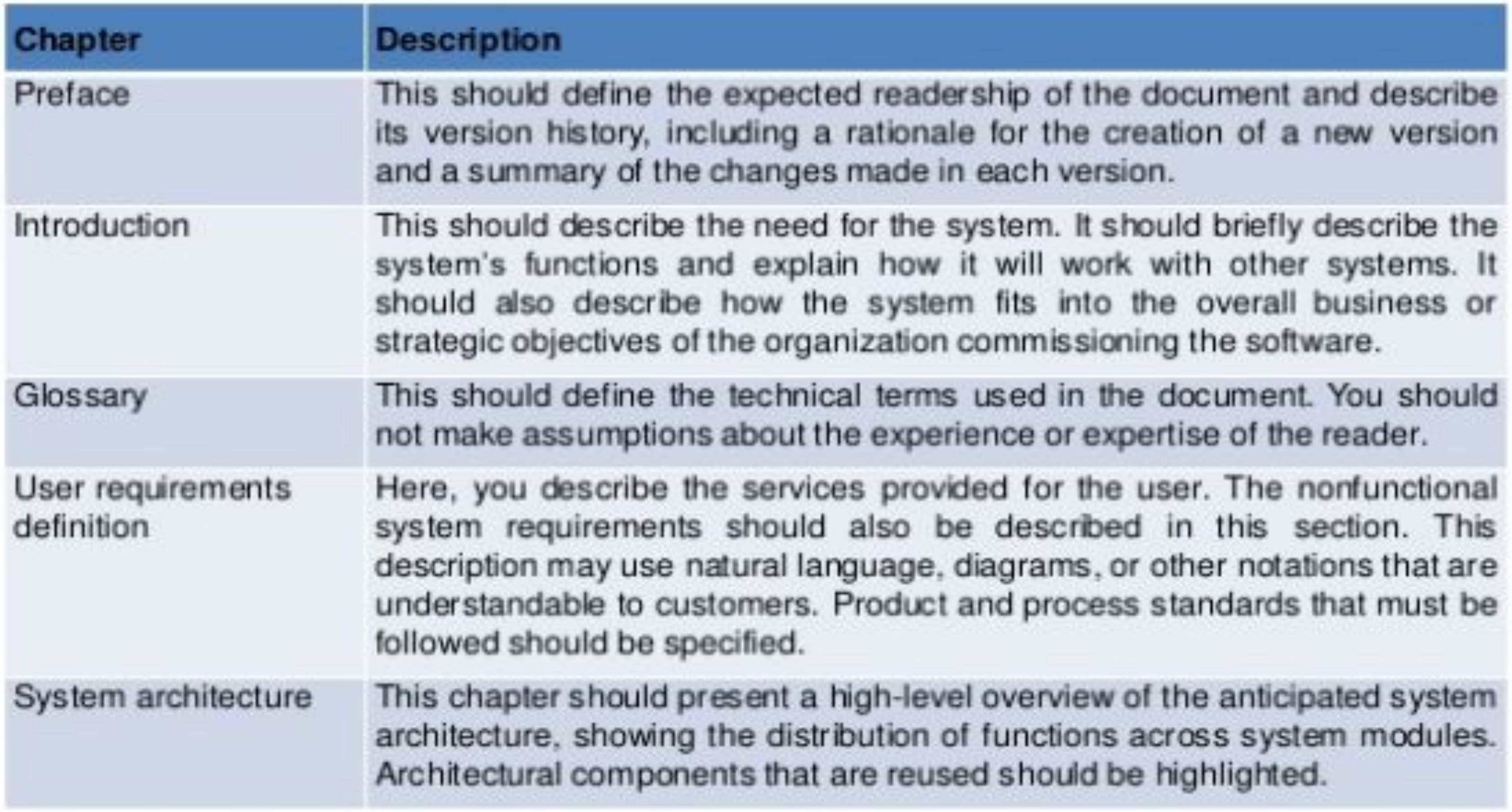
- These define *system properties* and *constraints* e.g. reliability, response time, and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Software Requirement Specification Document



- ✧ The *software requirements document* is the official statement of what is required of the system developers.
- ✧ Can include both a definition of user requirements and a specification of the system requirements.
- ✧ It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

The structure of a requirements document



Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

Requirements Engineering

- Requirement: A function, constraint or other things that the system must provide to fill the needs of the user(s)
- Engineering: implies that systematic and repeatable techniques should be used
- Requirement Engineering means that requirements for a product are defined, managed and tested systematically

Requirements Engineering

- It is essential that the software engineering team understand the requirements of a problem before the team tries to solve the problem.
- R.E is software engineering actions that start with communication activity and continues into the modeling activity.
- R.E establishes a solid base for design and construction. Without it ,it’s not possible to meet the customer needs.

7.2 Requirements Engineering Tasks

- Inception —Establish a basic understanding of the problem and the nature of the solution.
- Elicitation —Draw out the requirements from stakeholders.
- Elaboration (Highly structured)—Create an analysis model that represents information, functional, and behavioral aspects of the requirements.
- Negotiation—Agree on a deliverable system that is realistic for developers and customers.
- Specification—Describe the requirements formally or informally.
- . Validation — Review the requirement specification for errors, ambiguities, omissions, and conflicts
- Requirements management —Manage changing requirements.

Inception

- Inception— Ask “context-free” questions that establish …
  - Basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired, and
  - The effectiveness of preliminary communication and collaboration between the customer and the developer

Elicitation

- Elicitation - elicit requirements from customers, users and others.
  - Find out from customers, users and others what the product objectives are
  - what is to be done
  - how the product fits into business needs, and
  - how the product is used on a day to day basis and how many modules need to develop

Why Requirement elicitation is difficult?

- Problems of scope:
  - The boundary of the system is ill-defined.
  - Customers/users specify unnecessary technical detail that may confuse rather than clarify objectives.
- Problem of understanding:
  - Customers are not completely sure of what is needed.
  - Customers have a poor understanding of the capabilities and limitations of the computing environment.
  - Customers don’t have a full understanding of their problem domain.
  - Customers have trouble communicating needs to the system engineer.
  - Customers specify requirements that are ambiguous or not able to test.
- Problems of volatility:
  - Requirement change over time.

Elaboration

- Focuses on developing a refined technical model of software functions, features, and constraints using the information obtained during inception and elicitation
- Create an analysis model that identifies data, function and behavioral requirements.
- It describe how the end-user will interact with the system.
- End result defines informational, functional and behavioral domain of the problem

Negotiation

- Negotiation - agree on a deliverable system that is realistic for developers and customers
  - Requirements are categorized and organized into subsets
  - Relations among requirements identified
  - Requirements reviewed for correctness
  - Requirements prioritized based on customer needs
  - Negotiation about requirements, project cost and project timeline.

Specification

- Specification – Different things to different people.
- It can be –
  - Written Document
  - A set of graphical models,
  - A formal mathematical models
  - Collection of usage scenario.
  - A prototype
  - Combination of above.
- For large systems, written document, language descriptions, and graphical models may be the best approach.
- For small systems or products, usage scenarios

Validation

- Requirements Validation - formal technical review mechanism that looks for
  - Errors in content or interpretation
  - Areas where clarification may be required
  - Missing information
  - Inconsistencies (a major problem when large products or systems are engineered)
  - Conflicting or unrealistic (unachievable) requirements.

Requirement Management

– Set of activities that help project team to identify, control, and track requirements and changes as project proceeds

–Requirements begin with identification. Each requirement is assigned a unique identifier. Once requirement have been identified, traceability table are developed.

Traceability Table:

– **Features traceability table** - shows how requirements relate to customer observable features

–**Source traceability table** - identifies source of each requirement

–**Dependency traceability table** - indicate relations among requirements

–**Subsystem traceability table** - requirements categorized by subsystem

7.3 Initiating Requirements Engineering Process

- **Identify stakeholders**
  - Stakeholder can be "anyone who benefits in a direct or indirect way from the system which is being developed"

  Ex. Business manager, project manager, marketing people, software engineer, support engineer, end-users, internal-external customers, consultants, maintenance engineer.
  - Each one of them has different view of the system.
- **Recognize multiple points of view**
  - Marketing group concern about feature and function to excite potential market. To sell easily in the market.
  - Business manager concern about feature built within budget and will be ready to meet market.
  - End user – Easy to learn and use.
  - SE – product functioning at various infrastructure support.
  - Support engineer – Maintainability of software.

Role of RE is to categorize all stakeholder information in a way that there could be no inconsistent or conflict requirement with one another

- **Work toward collaboration**
  - RE identify areas of commonality (i.e. Agreed requirement) and areas of conflict or inconsistency.
  - It does not mean requirement defined by committee. It may happened they providing just view of their requirement.
  - Business manager or senior technologist may make final decision.
- **Asking the first questions**
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution
  - Is there another source for the solution that you need?

7.4 Eliciting Requirement

Approach for eliciting requirement:

- Collaborative Requirement Gathering
- Quality Function Deployment
- User Scenarios
- Elicitation Work Products

Collaborative Requirement Gathering

Guidelines for the meeting:

- Meetings are attended by all interested stakeholders.
- Rules established for preparation and participation.
- Agenda should be formal enough to cover all important points, but informal enough to encourage the free flow of ideas.
- A facilitator controls the meeting.
- A definition mechanism (blackboard, flip charts, etc.) is used.

- During the meeting:
  - The problem is identified.
  - Elements of the solution are proposed.
  - Different approaches are negotiated.
  - A preliminary set of solution requirements are obtained.
  - The atmosphere is collaborative and non-threatening.
- Flow of event – Outline the sequence of events occurs
  - Requirement gathering meeting (initial meeting)
  - During meeting
  - Follow the meeting.

Collaborative requirement gathering (contd.)

- In initial meeting, distribute “Product request” (defined by stakeholder) to all attendee.
- Based on product request, each attendee is asked to make
  - List of objects (Internal or external system objects)
  - List of services(Processes or functions)
  - List of constraints (cost, size, business rules) and performance criteria(speed, accuracy) are developed.
- Collect lists from everyone and combined.
- Combined list eliminates redundant entries, add new ideas , but does not delete anything.
- Objective is to develop a consensus list in each topic area (objects, services, constraints and performance).
- Based on lists, team is divided into smaller sub-teams : each works to develop mini-specification for one or more entries on each of the lists.

Collaborative requirement gathering (Contd.)

- Each sub-team the presents its mini-specification to all attendees for discussion. Addition, deletion and further elaboration are made.
- Now each team makes a list of validation criteria for the product and present to team.
- Finally, one or more participants is assigned the task of writing a complete draft specification.

Quality Function Deployment

- It is a technique that translate the needs of the customer into technical requirement for software.
- Concentrates on maximizing customer satisfaction.
- QFD emphasizes – what is valuable to the customer and then deploys these values throughout the engineering process.

Three types of requirement:

1. Normal Requirements – reflect objectives and goals stated for product. If requirement are present in final products, customer is satisfied.
2. Expected Requirements – customer does not explicitly state them. Customer assumes it is implicitly available with the system.
3. Exciting Requirements- Features that go beyond the customer's expectation.

During meeting with customer –

Function deployment determines the "value" of each function required of the system.

Information deployment identifies data objects and events and also tied with functions.

Task deployment examines the behavior of the system.

Value analysis determines the priority of requirements during these 3 deployments.

User Scenario

- It is difficult to move into more software engineering activities until s/w team understands how these functions and features will be used by diff. end-users.
- Developers and users create a set of usage threads for the system to be constructed
- A use-case scenario is a story about how someone or something external to the software (known as an actor) interacts with the system.
- Describe how the system will be used
- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way

Elicitation Work Products

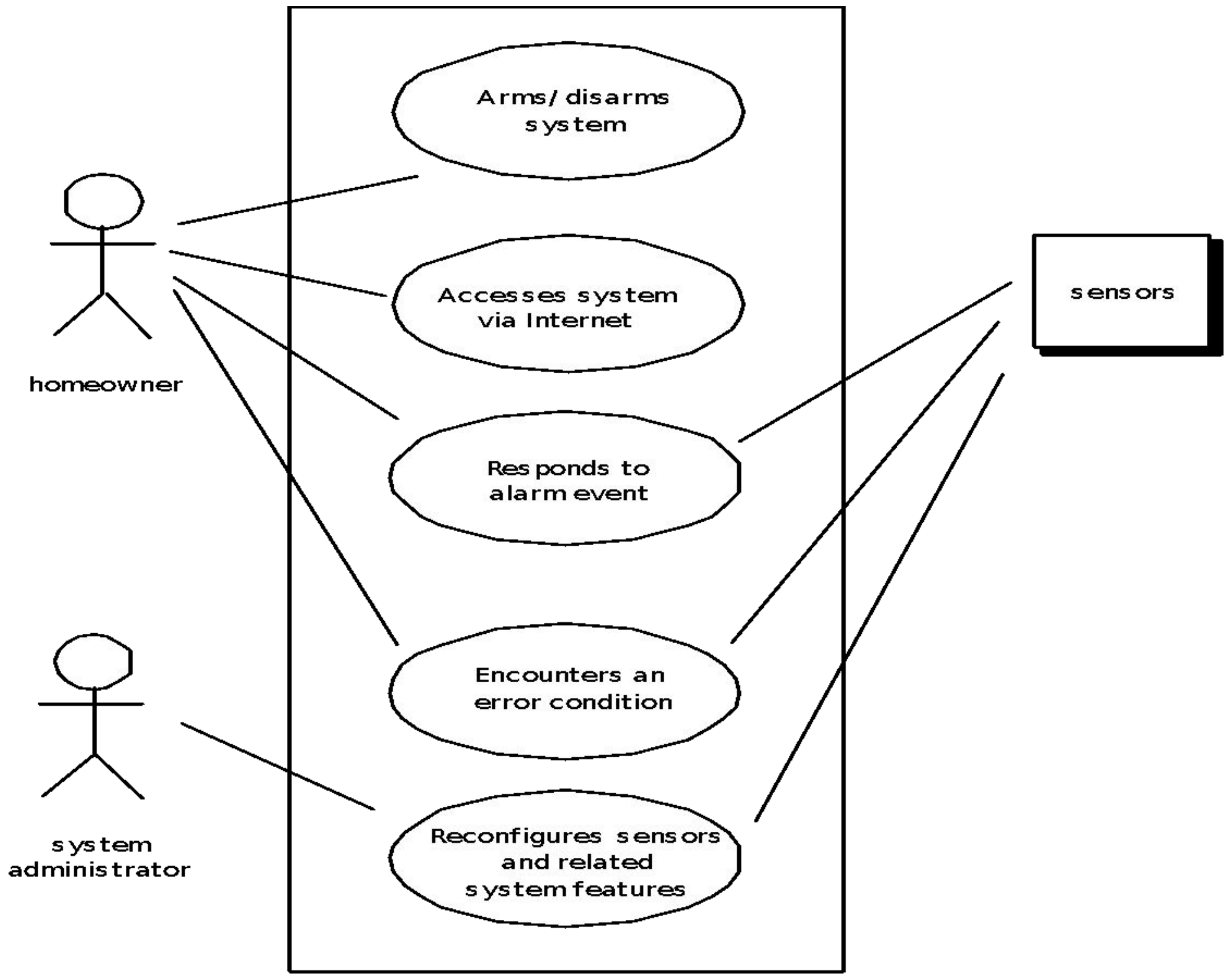
Elicitation work product will vary depending upon the size of the system or product to be built.

- Statement of need and feasibility.
- Statement of scope.
- List of participants in requirements elicitation.
- Description of the system’s technical environment.
- List of requirements and associated domain constraints.
- List of usage scenarios.
- Any prototypes developed to refine requirements.

7.5 Developing Use-Cases

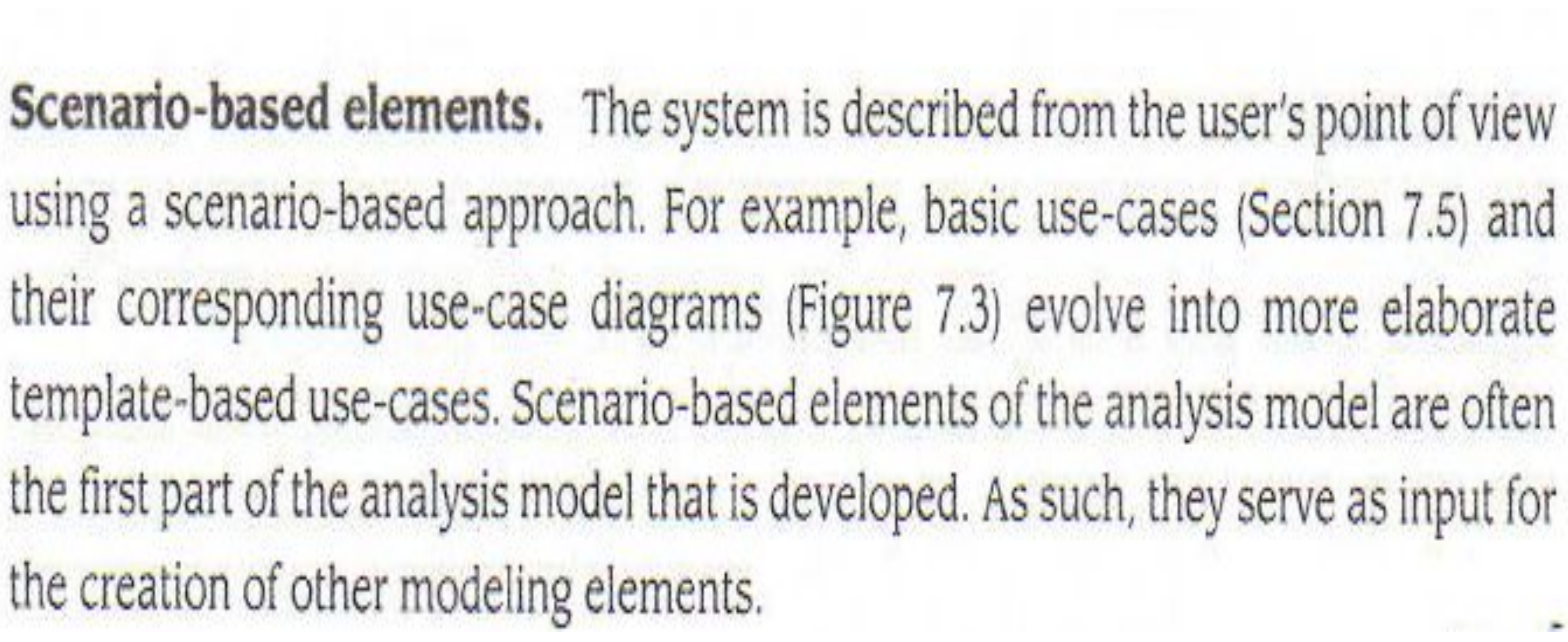
- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor’s goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor’s interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

Use-Case Diagram

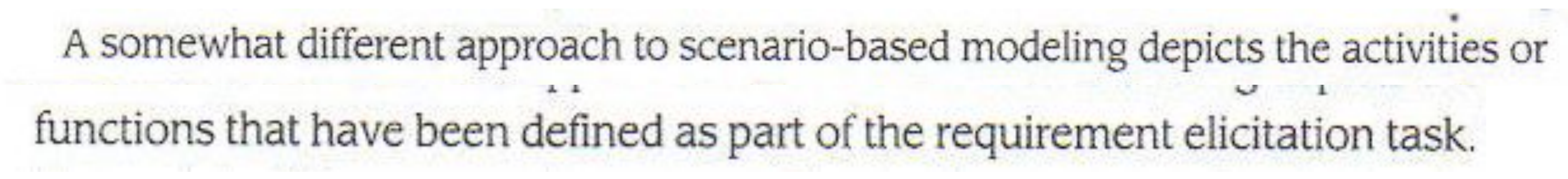

Arms/ disarms system
Accesses system via Internet
Responds to alarm event
Encounters an error condition
Reconfigures sensors and related system features
homeowner
system administrator
sensors

7.6 Building the Analysis Model

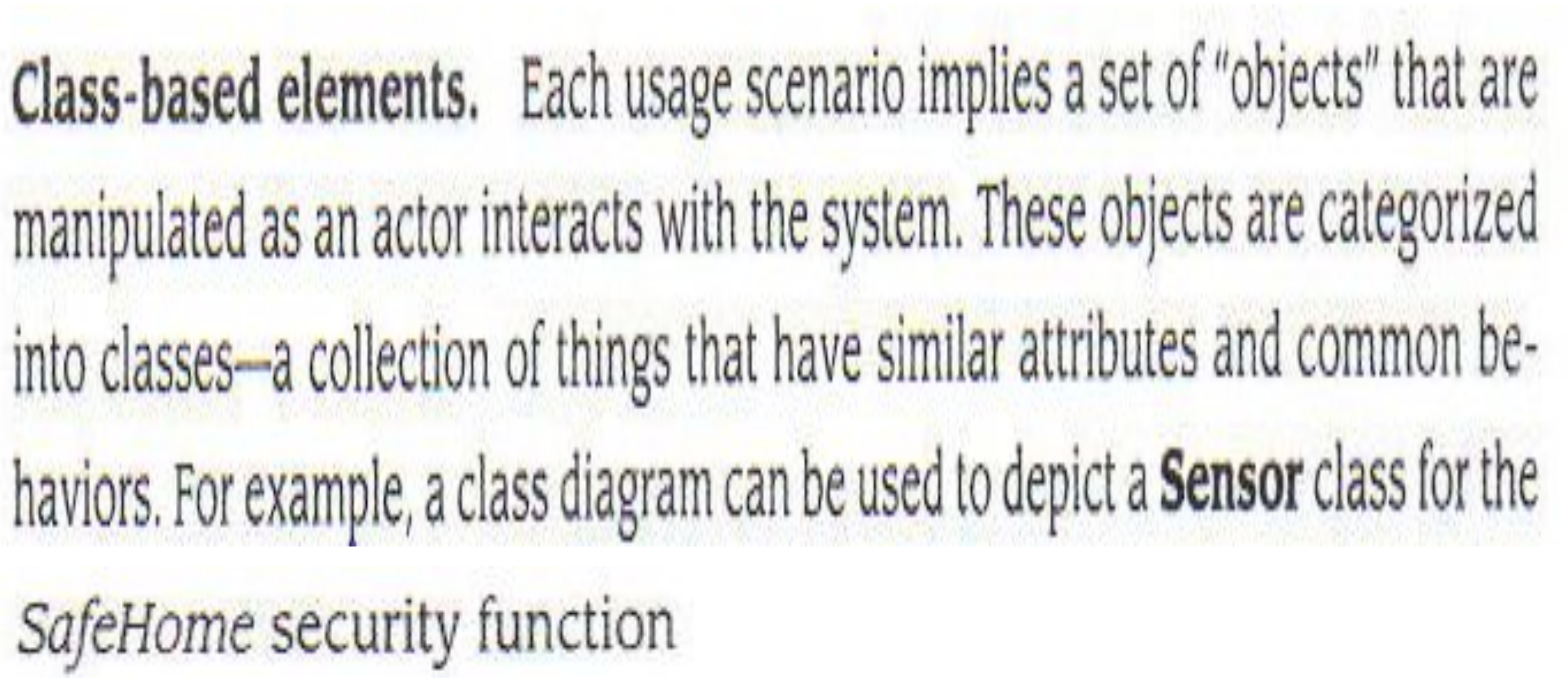
- 7.6.1 Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an “actor” and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram



Scenario-based elements. The system is described from the user's point of view using a scenario-based approach. For example, basic use-cases (Section 7.5) and their corresponding use-case diagrams (Figure 7.3) evolve into more elaborate template-based use-cases. Scenario-based elements of the analysis model are often the first part of the analysis model that is developed. As such, they serve as input for the creation of other modeling elements.



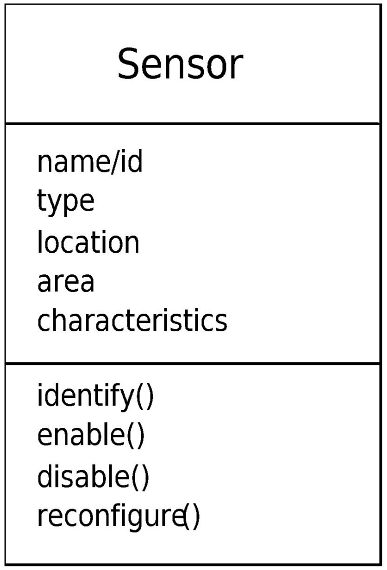
A somewhat different approach to scenario-based modeling depicts the activities or functions that have been defined as part of the requirement elicitation task.

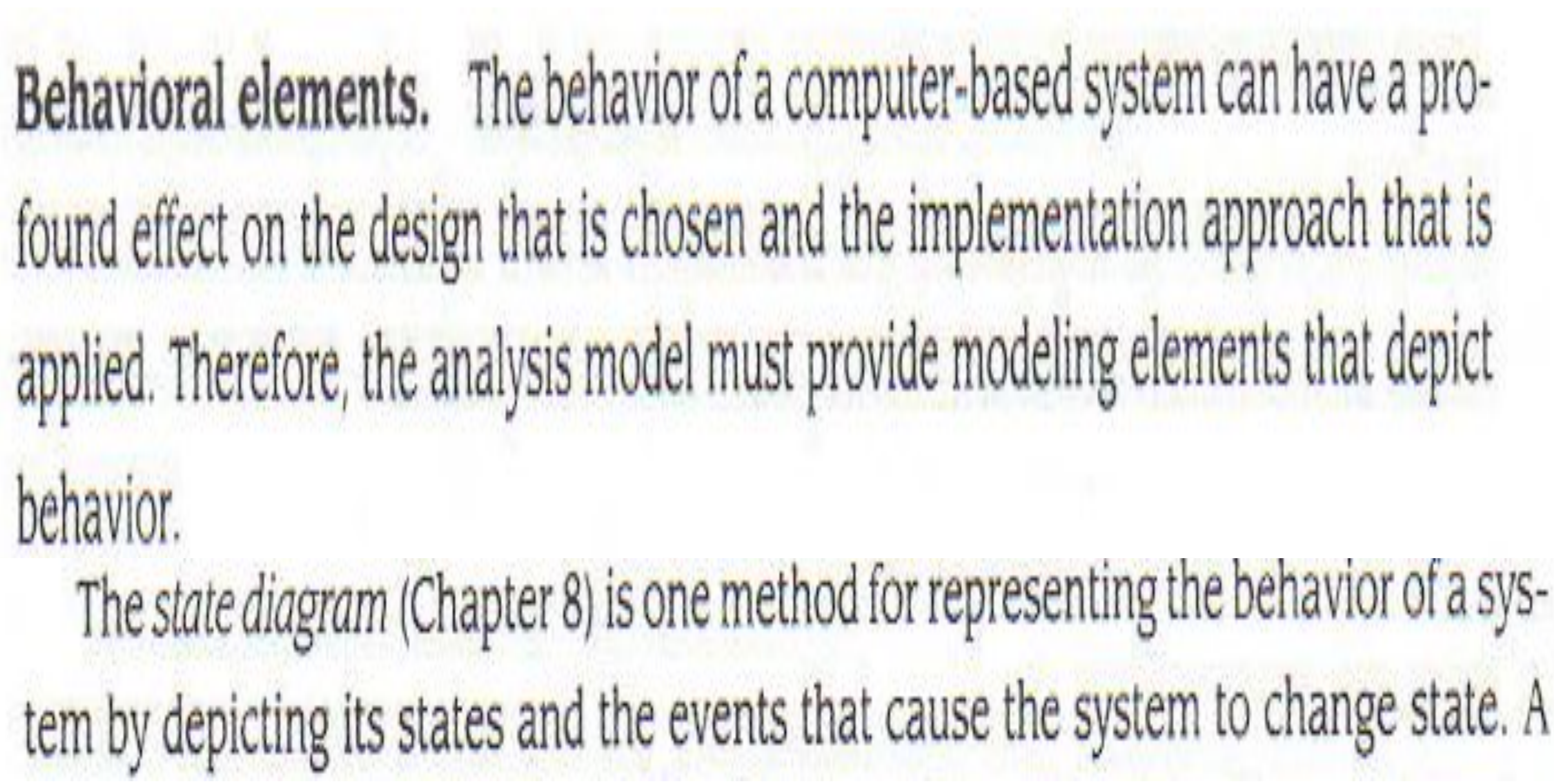


Class-based elements. Each usage scenario implies a set of "objects" that are manipulated as an actor interacts with the system. These objects are categorized into classes—a collection of things that have similar attributes and common behaviors. For example, a class diagram can be used to depict a **Sensor** class for the *SafeHome* security function

Class Diagram

From the *SafeHome* system …

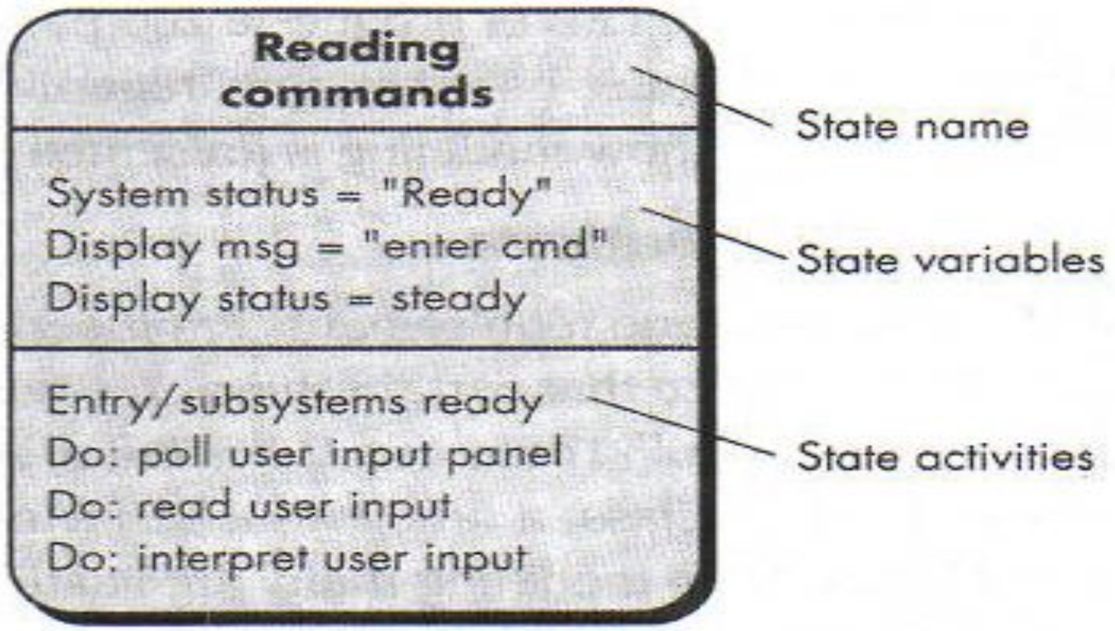


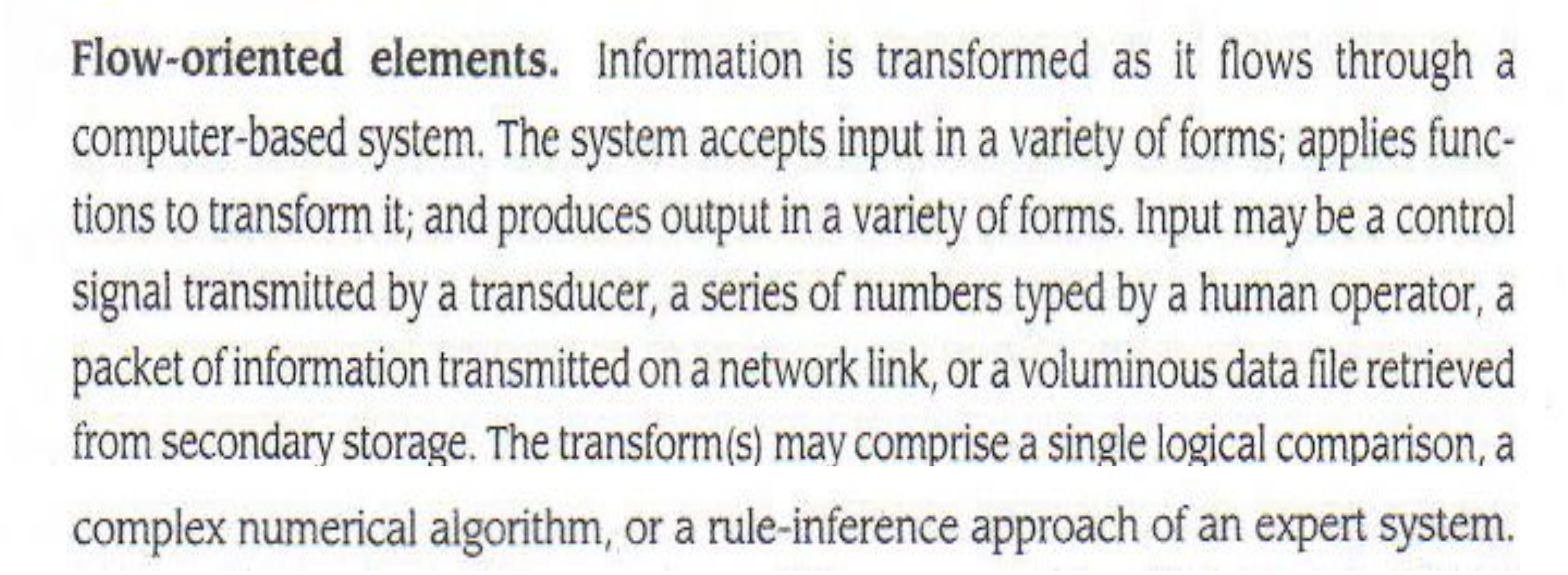


Behavioral elements. The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied. Therefore, the analysis model must provide modeling elements that depict behavior.

The *state diagram* (Chapter 8) is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state. A

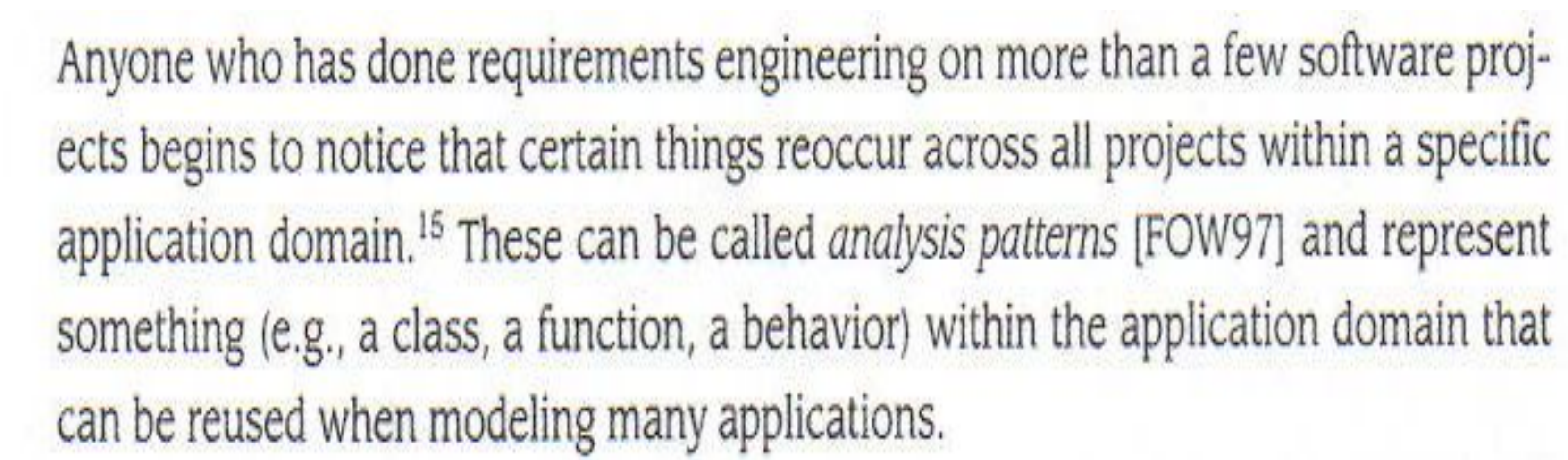
To illustrate a state diagram, consider a *reading commands* state for an office photocopier. UML state diagram notation is shown in Figure 7.6. A rounded rectangle represents a state. The rectangle is divided into three areas: (1) the state name (e.g., Reading commands), (2) *state variables* that indicate how the state manifests itself to the outside world, and (3) *state activities* that indicate how the state is entered (**entry/**) and actions (**do:**) that are invoked while in the state.


Reading commands
System status = "Ready"
Display msg = "enter cmd"
Display status = steady
Entry/subsystems ready
Do: poll user input panel
Do: read user input
Do: interpret user input
State name
State variables
State activities

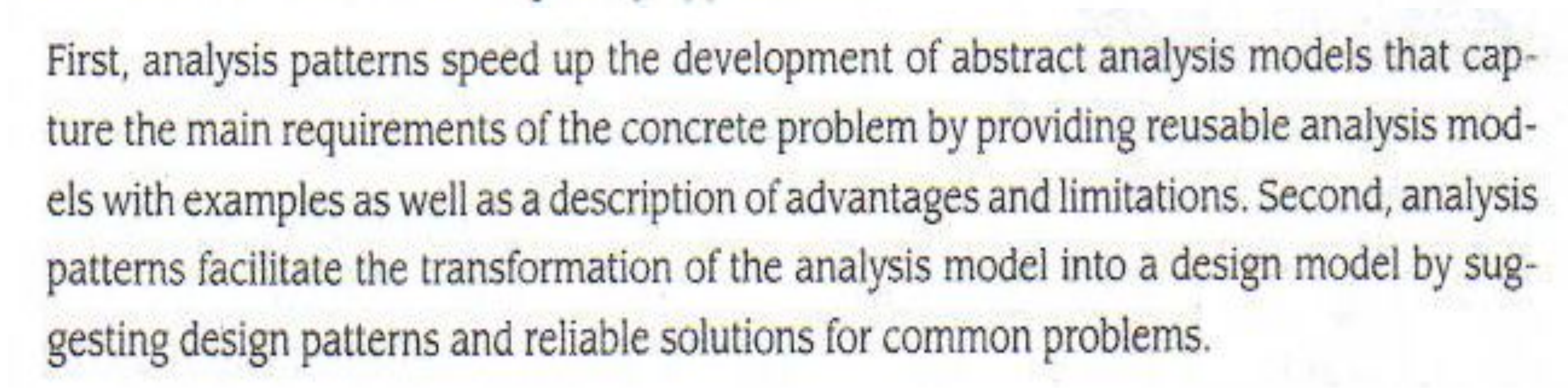


Flow-oriented elements. Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies functions to transform it; and produces output in a variety of forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.

7.6.2. Analysis Patterns



Anyone who has done requirements engineering on more than a few software projects begins to notice that certain things reoccur across all projects within a specific application domain.[15] These can be called *analysis patterns* [FOW97] and represent something (e.g., a class, a function, a behavior) within the application domain that can be reused when modeling many applications.



First, analysis patterns speed up the development of abstract analysis models that capture the main requirements of the concrete problem by providing reusable analysis models with examples as well as a description of advantages and limitations. Second, analysis patterns facilitate the transformation of the analysis model into a design model by suggesting design patterns and reliable solutions for common problems.

Pattern name: A descriptor that captures the essence of the pattern.

Intent: Describes what the pattern accomplishes or represents

Motivation: A scenario that illustrates how the pattern can be used to address the problem.

Forces and context: A description of external issues (forces) that can affect how the pattern is used and also the external issues that will be resolved when the pattern is applied.

Solution: A description of how the pattern is applied to solve the problem with an emphasis on structural and behavioral issues.

Consequences: Addresses what happens when the pattern is applied and what trade-offs exist during its application.

Design: Discusses how the analysis pattern can be achieved through the use of known design patterns.

Known uses: Examples of uses within actual systems.

Related patterns: On e or more analysis patterns that are related to the named pattern because (1) it is commonly used with the named pattern; (2) it is structurally similar to the named pattern; (3) it is a variation of the named pattern.

7.7 Negotiating Requirements

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders "win conditions"
  - Win conditions are not always obvious
- Negotiate
  - Work toward a set of requirements that lead to "win-win"

7.8 Validating Requirements

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?

- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

rid	a1	a2	a3	a4	
r1	✔yes		✔yes		
r2		✔yes		✔yes	
R3	✔yes	✔yes		✔yes	
r4	✔yes		✔yes		