Requirements Analysis

- Requirements analysis
  - specifies software’s operational characteristics
  - indicates software’s interface with other system elements
  - establishes constraints that software must meet

- Requirements analysis allows the software engineer (called an analyst or modeler in this role) to:
  - elaborate on basic requirements established during earlier requirement engineering tasks
  - build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.
Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non-functional models until design.
- Minimize coupling throughout the system.
- Be certain that the analysis model provides value to all stakeholders.
- Keep the model as simple as it can be.

Data Modeling

- Examines data objects independently of processing
- Focuses attention on the data domain
- Creates a model at the customer’s level of abstraction
- Indicates how data objects relate to one another

What is a Data Object?

Object — something that is described by a set of attributes (data items) and that will be manipulated within the software (system)

- Each instance of an object (e.g., a book) can be identified uniquely (e.g., ISBN #)
- Each plays a necessary role in the system i.e., the system could not function without access to instances of the object
- Each is described by attributes that are themselves data items
Typical Objects

- **external entities** (printer, user, sensor)
- **things** (e.g., reports, displays, signals)
- **occurrences or events** (e.g., interrupt, alarm)
- **roles** (e.g., manager, engineer, salesperson)
- **organizational units** (e.g., division, team)
- **places** (e.g., manufacturing floor)
- **structures** (e.g., employee record)

Data Objects and Attributes

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object.

**Object:** automobile

**Attributes:**
- make
- model
- body type
- price
- options code

What is a Relationship?

**relationship** — indicates "connectedness"; a "fact" that must be "remembered" by the system and cannot or is not computed or derived mechanically.

- several instances of a relationship can exist
- objects can be related in many different ways
Building an ERD

- Level 1—model all data objects (entities) and their "connections" to one another
- Level 2—model all entities and relationships
- Level 3—model all entities, relationships, and the attributes that provide further depth

The ERD: An Example

Object-Oriented Concepts

- Must be understood to apply class-based elements of the analysis model
- Key concepts:
  - Classes and objects
  - Attributes and operations
  - Encapsulation and instantiation
  - Inheritance
Classes

- object-oriented thinking begins with the definition of a class, often defined as:
  - template
  - generalized description
  - “blueprint”... describing a collection of similar items
- a metaclass (also called a superclass) establishes a hierarchy of classes
- once a class of items is defined, a specific instance of the class can be identified

Building a Class

Encapsulation/Hiding

The object encapsulates both data and the logical procedures required to manipulate the data

Achieves “information hiding”
These courseware materials are to be used in conjunction with Software Engineering: A Practitioner’s Approach, 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005.

Class Hierarchy

Instance of Chair

Methods
(a.k.a. Operations, Services)

An executable procedure that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class. A method is invoked via message passing.

Scenario-Based Modeling

"[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)," Ivar Jacobson.

1. What should we write about?
2. How much should we write about it?
3. How detailed should we make our description?
4. How should we organize the description?
Use-Cases

- a scenario that describes a “thread of usage” for a system
- actors represent roles people or devices play as the system functions
- users can play a number of different roles for a given scenario

Developing a Use-Case

- What are the main tasks or functions that are performed by the actor?
- 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

[Diagram showing a Use-Case Diagram related to SafeHome system, including interactions between actors and system components like configuring system parameters and setting an alarm.]
Activity Diagram
Supplements the use-case by providing a diagrammatic representation of procedural flow.

Swimlane Diagrams
Allows the modeler to represent the flow of activities described by the use-case and at the same time indicate which actor (if there are multiple actors involved in a specific use-case) or analysis class has responsibility for the action described by an activity rectangle.

Flow-Oriented Modeling
Represents how data objects are transformed as they move through the system.
A data flow diagram (DFD) is the diagrammatic form that is used.
Considered by many to be an ‘old school’ approach, flow-oriented modeling continues to provide a view of the system that is unique—it should be used to supplement other analysis model elements.
The Flow Model

Every computer-based system is an information transform...

---

Flow Modeling Notation

- external entity
- process
- data flow
- data store

---

External Entity

A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

Data must always originate somewhere and must always be sent to something
**Process**

A data transformer (changes input to output)

Examples: compute taxes, determine area, format report, display graph

*Data must always be processed in some way to achieve system function.

---

**Data Flow**

Data flows through a system, beginning as input and be transformed into output.

- base
- compute triangle area
- height
- area

---

**Data Stores**

Data is often stored for later use.

- sensor #, type, location, age
- report required
- look-up sensor data
- sensor number
- type, location, age
- sensor data
**Data Flow Diagramming: Guidelines**

- All icons must be labeled with meaningful names.
- The DFD evolves through a number of levels of detail.
- Always begin with a context level diagram (also called level 0).
- Always show external entities at level 0.
- Always label data flow arrows.
- Do not represent procedural logic.

**Constructing a DFD—I**

- Review the data model to isolate data objects and use a grammatical parse to determine "operations".
- Determine external entities (producers and consumers of data).
- Create a level 0 DFD.

**Level 0 DFD Example**

[Diagram showing a level 0 DFD example with entities and data flows labeled.]
Constructing a DFD—II

- write a narrative describing the transform
- parse to determine next level transforms
- "balance" the flow to maintain data flow continuity
- develop a level 1 DFD
- use a 1:5 (approx.) expansion ratio

The Data Flow Hierarchy

Flow Modeling Notes

- each bubble is refined until it does just one thing
- the expansion ratio decreases as the number of levels increase
- most systems require between 3 and 7 levels for an adequate flow model
- a single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)
Process Specification (PSPEC)

- narrative
- pseudocode (PDL)
- equations
- tables
- diagrams and/or charts

DFDs: A Look Ahead

Maps into

Class-Based Modeling

- Identify analysis classes by examining the problem statement
- Use a "grammatical parse" to isolate potential classes
- Identify the attributes of each class
- Identify operations that manipulate the attributes
Analysis Classes

- External entities (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
- Things (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
- Occurrences or events (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.
- Roles (e.g., manager, engineer, salesperson) played by people who interact with the system.
- Organizational units (e.g., division, group, team) that are relevant to an application.
- Places (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
- Structures (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.

Selecting Classes—Criteria

- retained information
- needed services
- multiple attributes
- common attributes
- common operations
- essential requirements

Class Diagram
Behavioral Modeling

- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the following steps:
  - Evaluate all use-cases to fully understand the sequence of interaction within the system.
  - Identify events that drive the interaction sequence and understand how these events relate to specific objects.
  - Create a sequence for each use-case.
  - Build a state diagram for the system.
  - Review the behavioral model to verify accuracy and consistency.

State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function and the state of the system as observed from the outside as the system performs its function
  - The state of a class takes on both passive and active characteristics [CHA93].
    - A passive state is simply the current status of all of an object's attributes.
    - The active state of an object indicates the current status of the object as it undergoes a continuing transformation or processing.
State Diagram for the ControlPanel Class

The States of a System
- state—a set of observable circumstances that characterizes the behavior of a system at a given time
- state transition—the movement from one state to another
- event—an occurrence that causes the system to exhibit some predictable form of behavior
- action—process that occurs as a consequence of making a transition

Behavioral Modeling
- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another (How does the system change state?)
- indicate event
- indicate action
- draw a state diagram or a sequence diagram
These courseware materials are to be used in conjunction with Software Engineering: A Practitioner’s Approach, 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005.

### Sequence Diagram

![Sequence Diagram](image)

**Figure 8.27** Sequence diagram (partial) for SafeHome security function

### Writing the Software Specification

![Writing the Software Specification](image)

Everyone knew exactly what had to be done until someone wrote it down!

### Specification Guidelines

- use a layered format that provides increasing detail as the "layers" deepen
- use consistent graphical notation and apply textual terms consistently (stay away from aliases)
- be sure to define all acronyms
- be sure to include a table of contents; ideally, include an index and/or a glossary
- write in a simple, unambiguous style (see "editing suggestions" on the following pages)
- always put yourself in the reader’s position, “Would I be able to understand this if I wasn’t intimately familiar with the system?”
Specification Guidelines

Be on the lookout for persuasive connectors, ask why?
- keys: certainly, therefore, clear, obviously, it follows that ...

Watch out for vague terms
- keys: some, sometimes, often, usually, ordinarily, most, mostly ...

When lists are given, but not completed, be sure all items are understood
- keys: etc., and so forth, and so on, such as

Be sure stated ranges don’t contain unstated assumptions
- e.g., valid codes range from 10 to 100. Integer? Real? Hex?

Beware of vague verbs such as handled, rejected, processed, ...

Beware "passive voice" statements
- e.g., the parameters are initialized. By what?

Beware "dangling" pronouns
- e.g., the I/O module communicated with the data validation module and its control flag is set. Whose control flag?

Specification Guidelines

When a term is explicitly defined in one place, try substituting the definition for other occurrences of the term

When a structure is described in words, draw a picture

When a structure is described with a picture, try to redraw the picture to emphasize different elements of the structure

When symbolic equations are used, try expressing their meaning in words

When a calculation is specified, work at least two examples

Look for statements that imply certainty, then ask for proof
- keys: always, ever, all, none, never

Search behind certainty statements—be sure restrictions or limitations are realistic.