

Chapter 8

■ Understanding Requirements

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

Slides copyright © 1996, 2001, 2005, 2009 by Roger S. Pressman

For non-profit educational use only

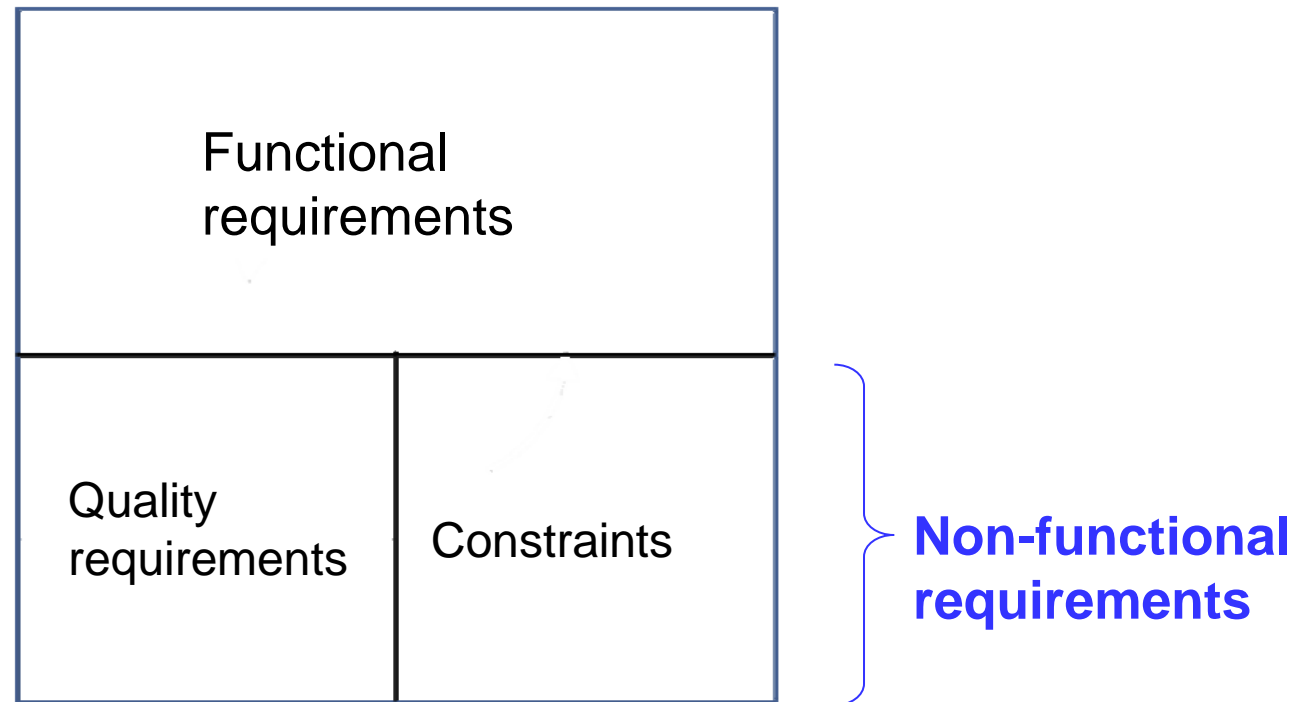
May be reproduced ONLY for student use at the university level when used in conjunction with *Software Engineering: A Practitioner's Approach, 7/e*. Any other reproduction or use is prohibited without the express written permission of the author.

All copyright information MUST appear if these slides are posted on a website for student use.

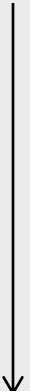
Table of Contents

- 8.1 Requirements Engineering
- 8.2 Establishing the Groundwork
- 8.3 Eliciting Requirements
- 8.4 Developing Use Cases
- 8.5 Building the Analysis Model
- 8.6 Negotiating Requirements
- 8.7 Requirements Monitoring
- 8.8 Validating Requirements


Taxonomy of System Requirements



8.1 Requirements Engineering-I

- 
- **Inception**—ask a set of 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**—elicit requirements from all stakeholders
 - **Elaboration**—create an analysis model that identifies data, function and behavioral requirements
 - **Negotiation**—agree on a deliverable system that is realistic for developers and customers

8.1 Requirements Engineering-II

- 
- **Specification**—can be any one (or more) of the following:
 - A written document
 - A set of models
 - A formal mathematical
 - A collection of user scenarios (use-cases)
 - A prototype
 - **Validation**—a 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.
 - **Requirements management**
 - Activities to identify, control and track requirements and changes to them

8.2 Establishing the Groundwork- Inception

- Identify stakeholders
 - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- 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?

Outcome: Product Request

written by marketing person involved in the *SafeHome* project

Our research indicates that the market for home management systems is growing at a rate of 40 percent per year. The first *SafeHome* function we bring to market should be the home security function. Most people are familiar with “alarm systems” so this would be an easy sell.

The home security function would protect against and/or recognize a variety of undesirable “situations” such as illegal entry, fire, flooding, carbon monoxide levels, and others. It’ll use our wireless sensors to detect each situation. It can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

8.3 Eliciting Requirements (= Requirements Gathering)

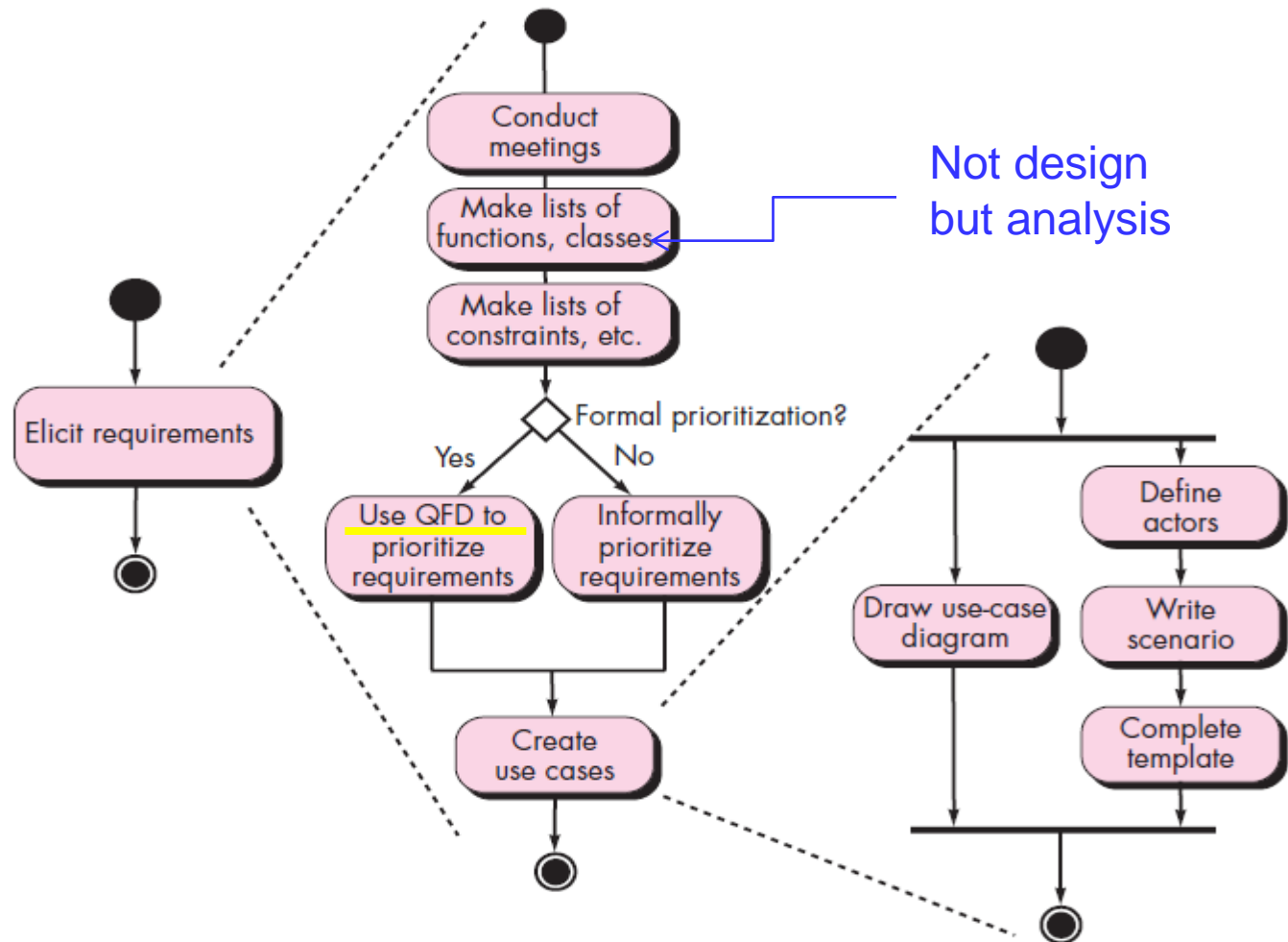
8.3.1 Collaborative Requirements Gathering

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established
- an agenda is suggested
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- the goal is
 - to identify the problem
 - propose elements of the solution
 - negotiate different approaches, and
 - specify a preliminary set of solution requirements

Eliciting Requirements

FIGURE 5.3

UML activity diagrams for eliciting requirements



8.3.2 Quality Function Deployment (1/2)

- Quality Function Deployment (= QFD)
- Originally developed by Dr. Yoji Akao in 1966.
- A quality management technique that translates the (non-measurable) **needs of the customer** into (measurable) **technical requirements for software**.
 - 1) Customer requirements
e.g.) "how it looks, how it feels, durability, etc."
 - 2) Technical specifications
e.g.) "oven temperature, mould diameter, etc."

8.3.2 Quality Function Deployment (2/2)

- QFD emphasizes an **understanding of what is valuable to the customers** in order to eventually deploy the values.
- Three types of requirements:
 - **Normal:** Normally stated for a system
E.g.) Graphical displays, defined levels of performance...
 - **Expected:** So fundamental that the customer does not explicitly state them
E.g.) Usability, correctness, ...
 - **Exciting:** Beyond the customer's expectation
E.g.) multi-touch screen, visual voice mail, ...

8.3.3 Usage Scenarios

- To move into more technical software engineering activities
- To understand how these functions and features will be used by different classes of end users.
- Can create a set of scenarios that identify a thread of usage for the system to be constructed.
- The scenarios, often called use cases [Jac 92], provide a description of how the system will be used.
- To be discussed in greater detail in Section 5.4.

8.3.4 Elicitation Work Products

- a statement of need and feasibility.
- a bounded statement of scope for the system or product.
- a list of customers, users, and other stakeholders who participated in requirements elicitation
- a description of the system's technical environment.
- a list of requirements (preferably organized by function) and the domain constraints that apply to each.
- a set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- any prototypes developed to better define requirements.

8.4 Developing Use-Cases(Elaboration)

- 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?
- (See example in Section 8.5.1 A)

8.5 Building the Analysis Model(Elaboration)

8.5.1 Elements of the analysis model

Opposite
viewpoints

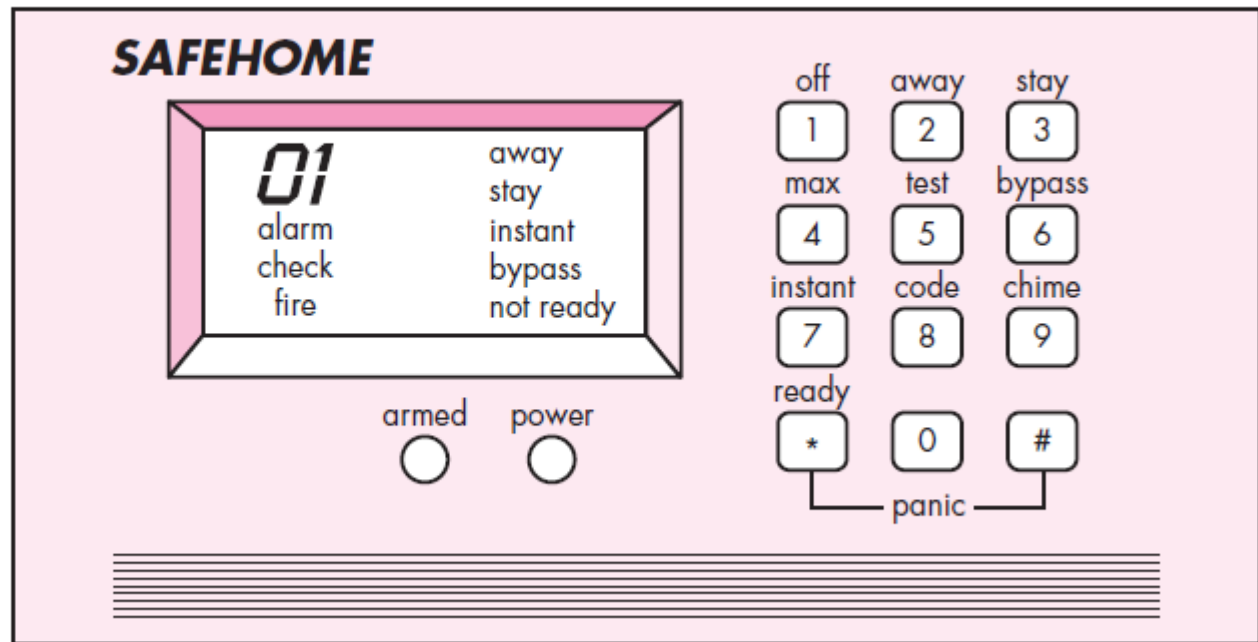
- A. **Scenario-based elements**
 - Functional—processing narratives for software functions
 - Use-case—descriptions of the interaction between an “actor” and the system
- B. **Class-based elements**
 - Implied by scenarios
- C. **Behavioral elements**
 - State (transition) diagram

The Control Panel of *Safehome*

FIGURE 5.1

SafeHome
control panel

User
interface
provides a
high-level
view of the
system
services.



Actors

- Homeowner (= a user)
- Setup manager(= system administrator)
- Sensors(= devices attached to the system)
- Monitoring and response subsystem
(= the central station that monitors the *SafeHome*)

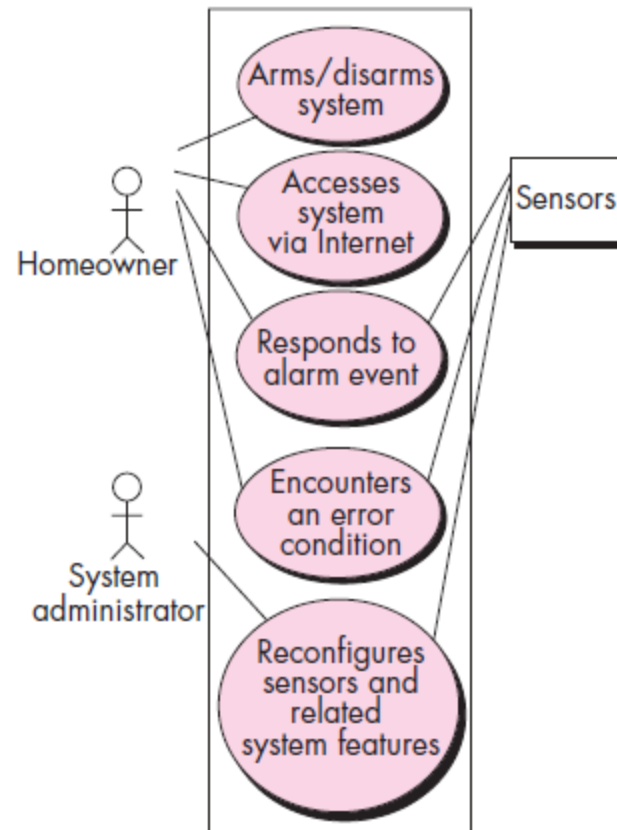
A. The basic use case for system activation

1. The homeowner observes the *SafeHome* control panel (Figure 5.1) to determine if the system is ready for input. If the system is not ready, a *not ready* message is displayed on the LCD display, and the homeowner must physically close windows or doors so that the *not ready* message disappears. [A *not ready* message implies that a sensor is open; i.e., that a door or window is open.]
2. The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.
3. The homeowner selects and keys in *stay* or *away* (see Figure 5.1) to activate the system. *Stay* activates only perimeter sensors (inside motion detecting sensors are deactivated). *Away* activates all sensors.
4. When activation occurs, a red alarm light can be observed by the homeowner.

Use-Case Diagram

FIGURE 5.2

UML use case diagram for *SafeHome* home security function



- Context Diagram
- Actors

More detailed use case description (1/3)

Use case:	<i>InitiateMonitoring</i>
Primary actor:	Homeowner.
Goal in context:	To set the system to monitor sensors when the homeowner leaves the house or remains inside.
Preconditions:	System has been programmed for a password and to recognize various sensors.
Trigger:	The homeowner decides to "set" the system, i.e., to turn on the alarm functions.
Scenario:	
	1. Homeowner: observes control panel
	2. Homeowner: enters password
	3. Homeowner: selects "stay" or "away"
	4. Homeowner: observes read alarm light to indicate that <i>SafeHome</i> has been armed

More detailed use case description (2/3)

Exceptions:

1. Control panel is *not ready*: homeowner checks all sensors to determine which are open; closes them.
2. Password is incorrect (control panel beeps once): homeowner reenters correct password.
3. Password not recognized: monitoring and response subsystem must be contacted to reprogram password.
4. *Stay* is selected: control panel beeps twice and a *stay* light is lit; perimeter sensors are activated.
5. *Away* is selected: control panel beeps three times and an *away* light is lit; all sensors are activated.

Priority: Essential, must be implemented

When available: First increment

More detailed use case description (3/3)

Frequency of use: Many times per day

Channel to actor: Via control panel interface

Secondary actors: Support technician, sensors

Channels to secondary actors:

Support technician: phone line

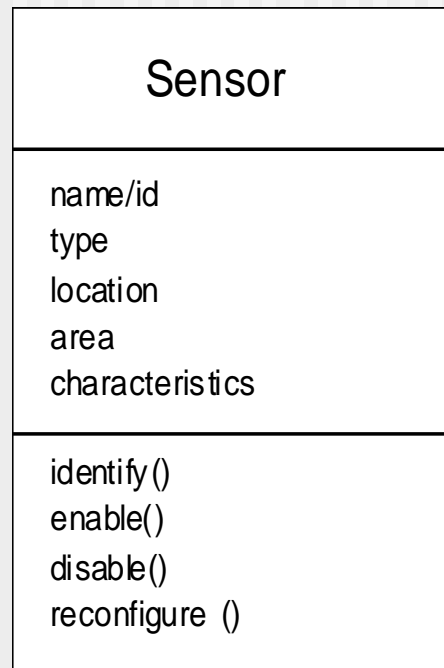
Sensors: hardwired and radio frequency interfaces

Open issues:

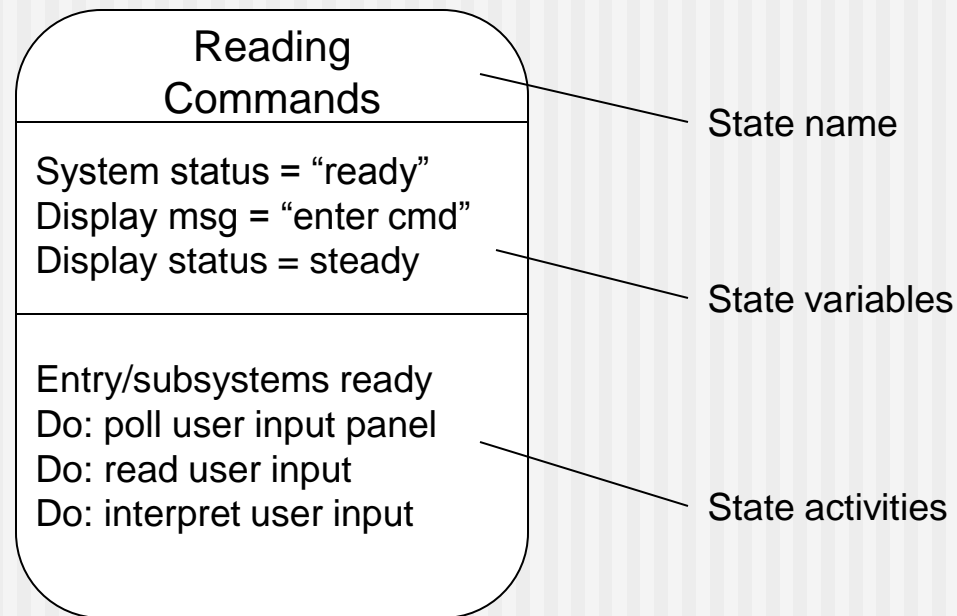
1. Should there be a way to activate the system without the use of a password or with an abbreviated password?
2. Should the control panel display additional text messages?
3. How much time does the homeowner have to enter the password from the time the first key is pressed?
4. Is there a way to deactivate the system before it actually activates?

B. Class Diagram

From the *SafeHome* system ...



C. State Diagram



8.5.2 Analysis Patterns – Template

Analysis
Pattern –
description of
a recurring
problem.

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: One 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.

8.6 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”

Is it possible for a requirement to be a loss to one stakeholder and at the same time a win to another?

=> Yes!

8.7 Requirements Monitoring

Especially needs in incremental development

- *Distributed debugging* – uncovers errors and determines their cause.
- *Run-time verification* – determines whether software matches its specification.
- *Run-time validation* – assesses whether evolving software meets user goals.
- *Business activity monitoring* – evaluates whether a system satisfies business goals.
- *Evolution and codesign* – provides information to stakeholders as the system evolves.

8.8 Validating Requirements - I

- 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?

8.8 Validating Requirements - II

- 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?