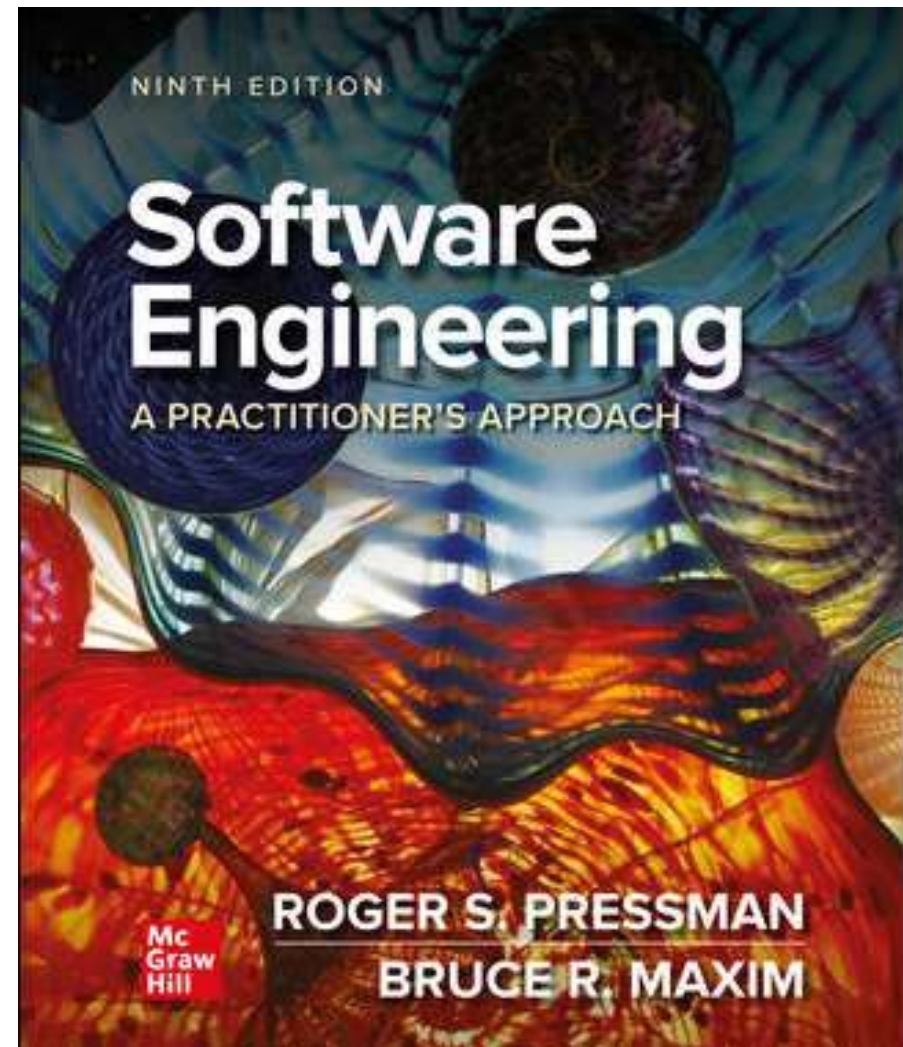


Chapter 7

Understanding Requirements

Part Two - Modeling



Requirements Engineering ¹

- Inception - establish a basic understanding of the problem, the people who want a solution, and the nature of the solution that is desired, important to establish effective customer and developer communication.
- Elicitation - elicit requirements and business goals from all stakeholders.
- Elaboration - focuses on developing a refined requirements model that identifies aspects of software function, behavior, and information.

Requirements Engineering ²

- Negotiation—agree on the scope of a deliverable system that is realistic for developers and customers.
- Specification—can be any or all of the following: written documents, graphical models, mathematical models, usage scenarios, prototypes.
- Validation—Requirements engineering work products produced during requirements engineering are assessed for quality and consistency.
- Requirements management – set of traceability activities to help the project team identify, control, and track requirements and their changes to requirements as the project proceeds.

Non-functional Requirements

Non-Functional Requirement (NFR) – quality attribute, performance attribute, security attribute, or general system constraint.

A two-phase process is used to determine which NFR's are compatible:

- The first phase is to create a matrix using each NFR as a column heading and the system SE guidelines a row labels.
- The second phase is for the team to prioritize each NFR using a set of decision rules to decide which to implement by classifying each NFR and guideline pair as complementary, overlapping, conflicting, or independent.

Establishing the Groundwork

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?

Collaborative Requirements Gathering

- Meetings (real or virtual) are conducted and attended by both software engineers and other stakeholders.
- Rules for preparation and participation are established.
- Agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- A “facilitator” (customer, developer, or outsider) controls the meeting.
- A “definition mechanism” (worksheets, flip charts, wall stickers or virtual forum) is used.
- Goal is to identify the problem, propose solution elements, and negotiate different approaches.

Elicitation Work Products

- Statement of need and feasibility.
- Bounded statement of scope for the system or product.
- List of customers, users, and other stakeholders who participated in requirements elicitation,
- Description of the system's technical environment.
- List of requirements (preferably organized by function) and the domain constraints that apply to each.
- Set of usage scenarios (written in stakeholders' own words) that provide insight into the use of the system or product under different operating conditions.

Use Case Definition

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:

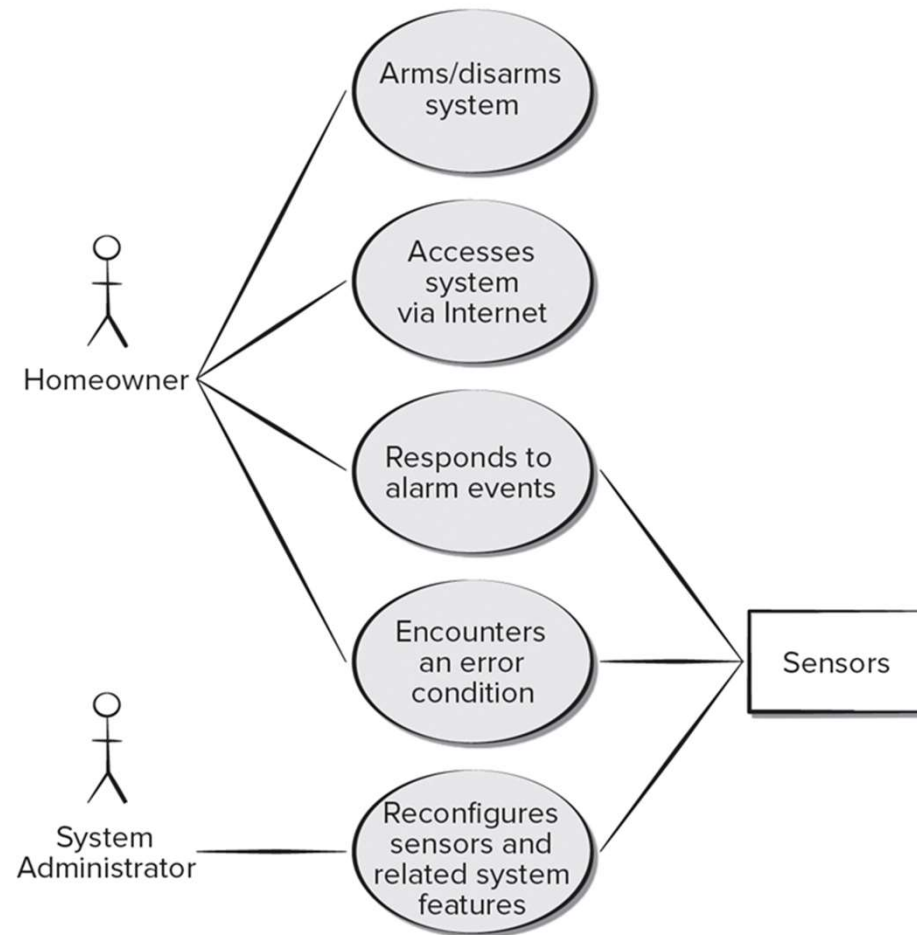
- Who is the primary actor, the secondary actor (s)?
- What are the actor’s goals?
- What preconditions should exist before the story begins?
- What main tasks or functions are performed by the actor?
- What extensions might be considered as the story is described?
- What variations in the actor’s interaction 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?

Analysis Model Elements

- *Analysis model* provides a description of the required informational, functional, and behavioral domains for a computer-based system.
- **Scenario-based elements** – functional descriptions are expressed in the customer's own words and user stories and as interactions of actors with the system expressed using UML use case diagrams.
- **Class-based elements** – collections of attributes and behaviors implied by the user stories and expressed using UML class diagrams (information domain).
- **Behavioral elements** – may be expressed using UML state diagrams as inputs causing state changes.

UML Use Case Diagram

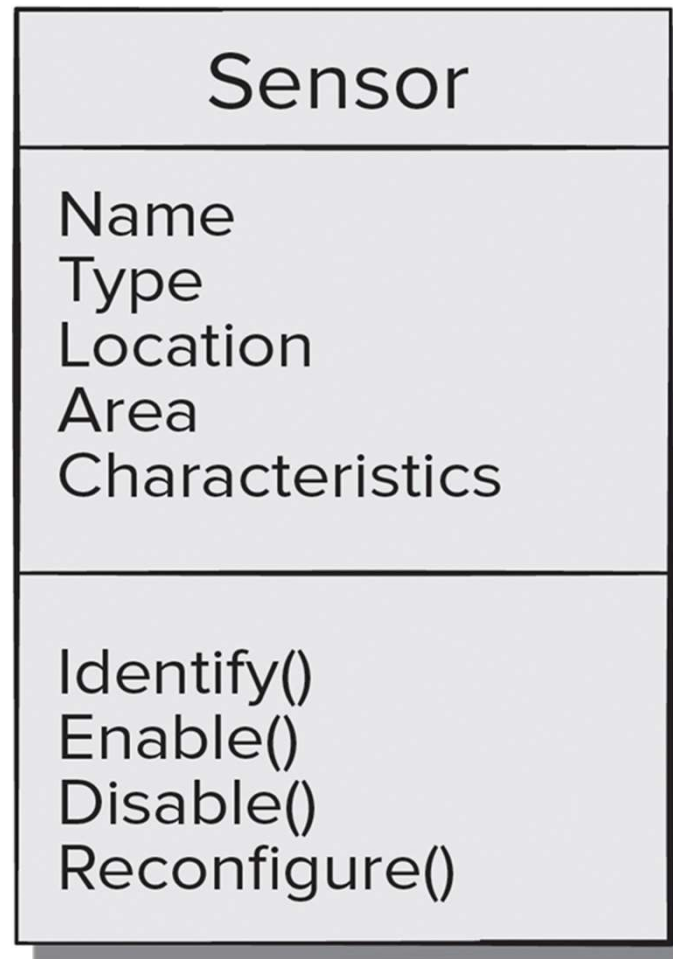
Copyright © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.



[Access the text alternative for slide images.](#)

UML Class Diagram

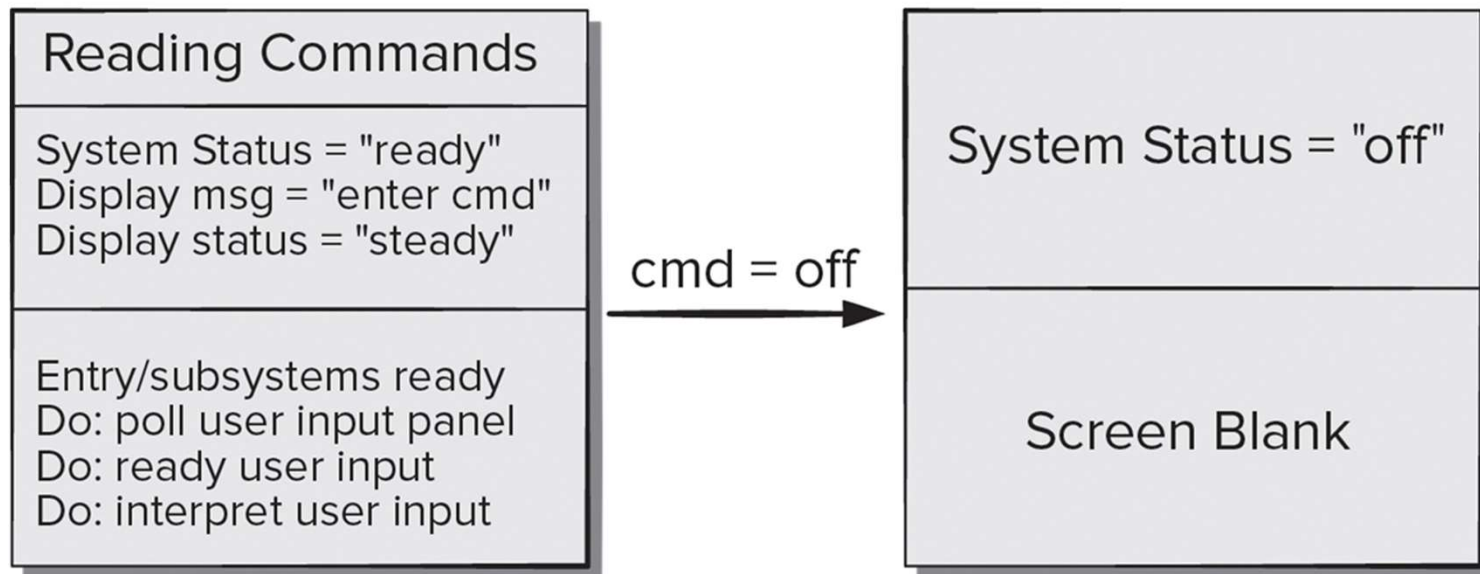
Copyright © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.



[Access the text alternative for slide images.](#)

UML State Diagram

Copyright © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.



[Access the text alternative for slide images.](#)

Analysis Patterns

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 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.

Negotiating Requirements

Negotiations strive for a “win-win” result, stakeholders win by getting a product satisfying most of their needs and developers win by getting achievable deadlines.

Handshaking is one-way to achieve “win-win”.

- Developers propose solutions to requirements, describe their impact, and communicate their intentions to the customers.
- Customer review the proposed solutions, focusing on missing features and seeking clarification of novel requirements.
- Requirements are determined to be *good enough* if the customers accept the proposed solutions.

Handshaking tends to improve identification, analysis, and selection of variants.

Requirements Monitoring

Useful for incremental development includes:

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

Validating Requirements ₁

- 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?

Validating Requirements ²

- 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 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?



Because learning changes everything.®

www.mheducation.com