Chapter 7

Understanding Requirements

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 8/e

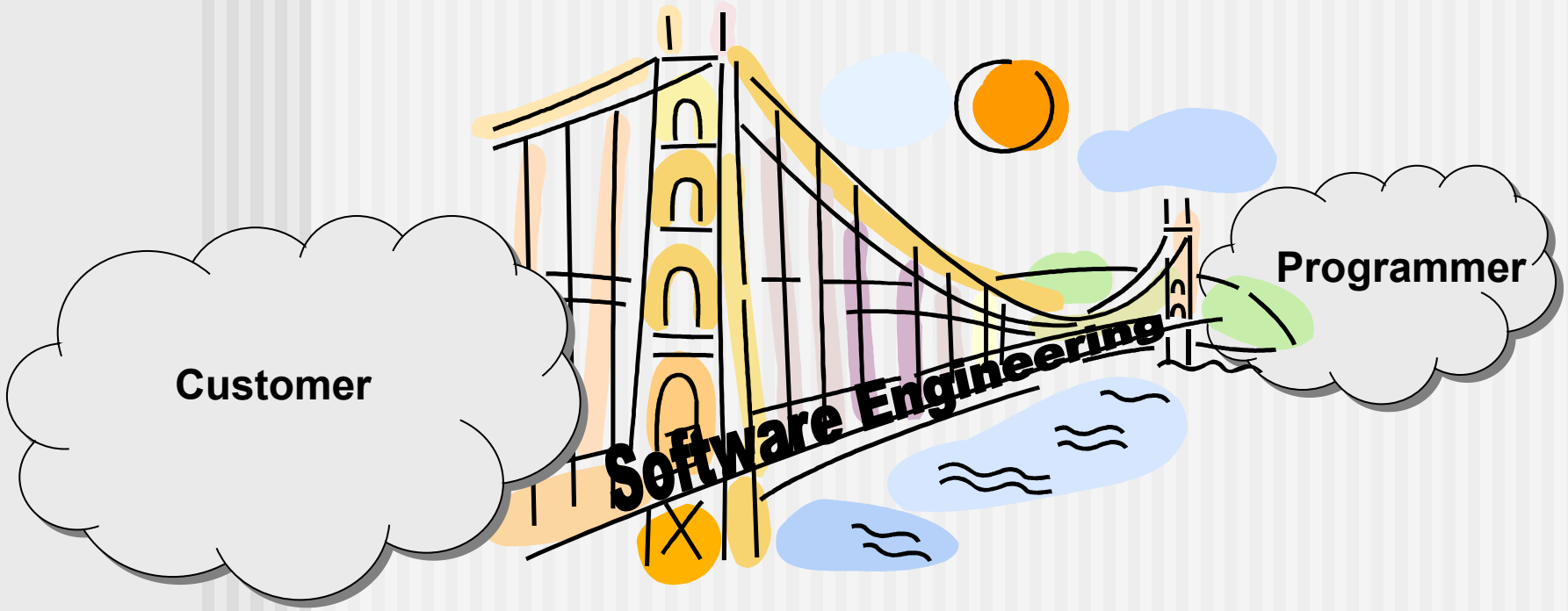
by Roger S. Pressman and Bruce R. Maxim

Requirements Engineering

- The broad spectrum of tasks and techniques that lead to an understanding of requirements is called requirements engineering.
  - Doesn't the customer know what is required?
  - Shouldn't the end users have a good understanding of the features and functions that will provide benefit?
  - Surprisingly, in many instances the answer to these questions is "No".
  - Even if customers and end users are explicit in their needs, those needs will change throughout the project.
- Requirements engineering builds a bridge to design and construction.
  - Where does the bridge originate?
  - From stakeholders? or
  - From a broader system definition?
- The fundamental problem remains the same, getting timely, accurate, and stable **stakeholder input**. Requirements engineering establishes a solid base for design and construction.

The Role of Software Engg. (1)

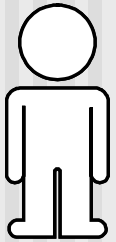
A bridge from customer needs to programming implementation



First law of software engineering

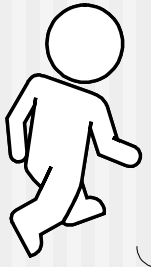
Software engineer is willing to learn the problem domain
(problem cannot be solved without understanding it first)

The Role of Software Engg. (2)

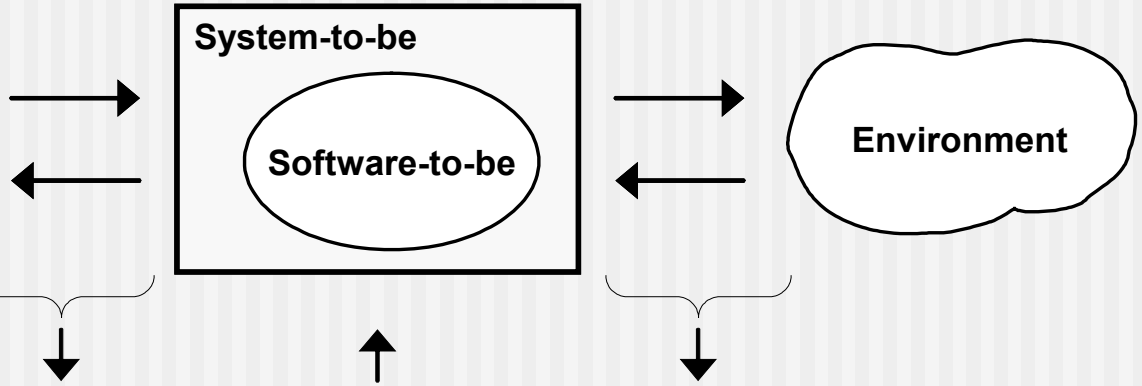


Customer:

Requires a computer system to *achieve some business goals* by user interaction or interaction with the environment in a specified manner



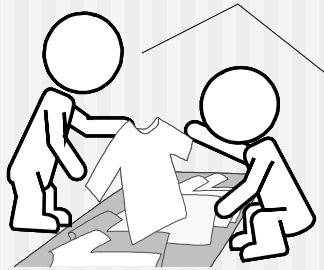
User



Software Engineer's task:

To ***understand how*** the system-to-be needs to interact with the user or the environment so that customer's requirement is met and ***design*** the software-to-be

May be the same person



Programmer's task:

To ***implement*** the software-to-be designed by the software engineer

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
  - The important part is to establish business goals.
  - Problems of scope
  - Problems of understanding
  - Problems of volatility
- Elaboration—create an analysis model that identifies data, function and behavioral requirements
  - Create and refine the user scenarios that describe how the end users interact with the system.
  - Parse each user scenario to extract analysis classes (business domain entities that are visible to the end user).

Requirements Engineering-II

- Recall the Unified Process (Chapter 4) defines a more comprehensive “inception phase”.
- The inception phase encompasses both customer communication and planning activities.
  - Fundamental business requirements --- use cases
  - Architecture --- a tentative outline of major subsystems, functions and features.
  - Planning identifies resources, assesses major risks, defines a schedule.
  - Establishing a basis for the phases that are to be applied as the software increment is developed.

Requirements Engineering-II

- The elaboration phase encompasses the communication and modeling activities of the generic process model.
  - Refining and expanding the preliminary use cases that were developed as part of the inception phase.
  - Expanding the architectural representation to include 5 different views of the software, there are use case model, analysis model, design model, implementation model, deployment model.
  - Executable architectural baseline.

Requirements Engineering-III

- Negotiation—agree on a deliverable system that is realistic for developers and customers
- Specification—can be any one (or more) of the following:
  - A written document、A set of models、A formal mathematical model
  - A collection of user scenarios (use-cases)、A prototype

  EXAMPLE: SRS Template

  Textbook Software Engineering a practitioner's approach. 8th Edition

  Page 136, Software requirement specification template.

Requirements Engineering-III

- 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.

  EXAMPLE: Validation Checklist

  Page 106, Requirements validation checklist
- Requirements management

Establishing the groundwork

- Identify stakeholders
  - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- Example: Using “Planning Poker”, Priority List

Establishing the groundwork

- 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?

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

Traceability

	Req ID1	Req ID2	Req ID3	Req ID4	Req ID5
Req ID1					
Req ID2					
Req ID3	X				
Req ID4			X		
Req ID5	X	X			

(a)

	DE ID1	DE ID2	DE ID3	DE ID 4
Req ID1	X			
Req ID2			X	
Req ID3		X		
Req ID4		X		
Req ID5				X

(b)

FIGURE 2. Simple example of traceability matrices: (a) shows which requirements depend upon which others—that is, Req ID3 depends upon Req ID1—and (b) shows which design elements satisfy which requirements—that is, Req ID3 is satisfied by DE ID2.

Eliciting Requirements

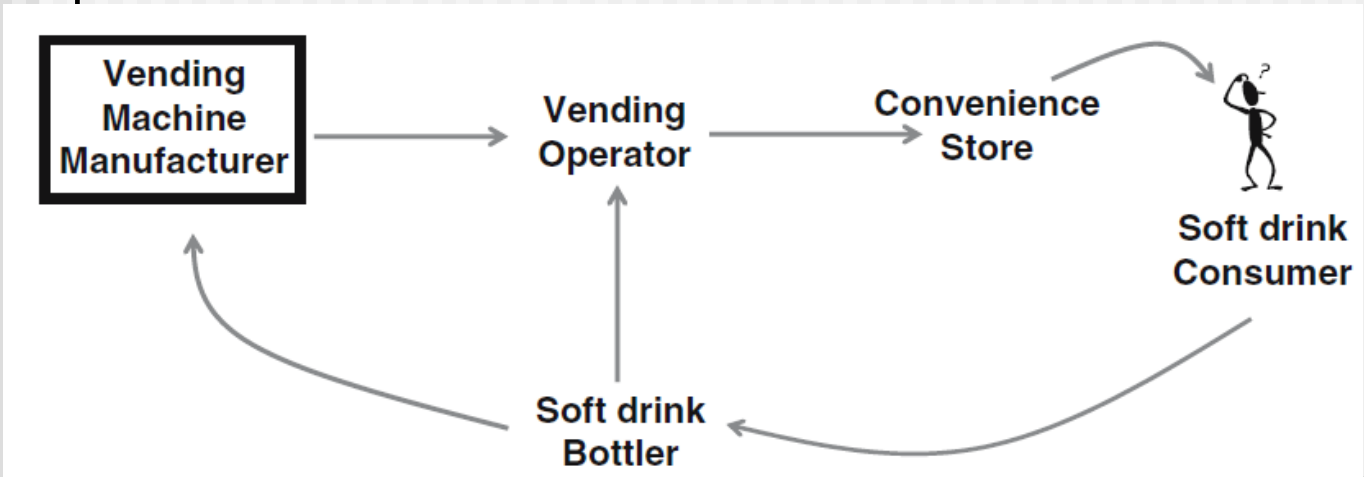
- meetings are conducted and attended by both software engineers and customers.
- rules for preparation and participation are established
- an agenda is suggested.
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting.
- Example: SafeHome project (page110)
- CaseStudy: SafeHome (page111)

Eliciting Requirements

- 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

Quality Function Deployment(**QFD**)

- Function deployment determines the “value” (as perceived by the customer) of each function required of the system.
- Information deployment identifies data objects and events.
- Task deployment examines the behavior of the system.
- Value analysis determines the relative priority of requirements.



Eliciting Requirements

- Usage Scenarios
- CaseStudy: SafeHome (page112)

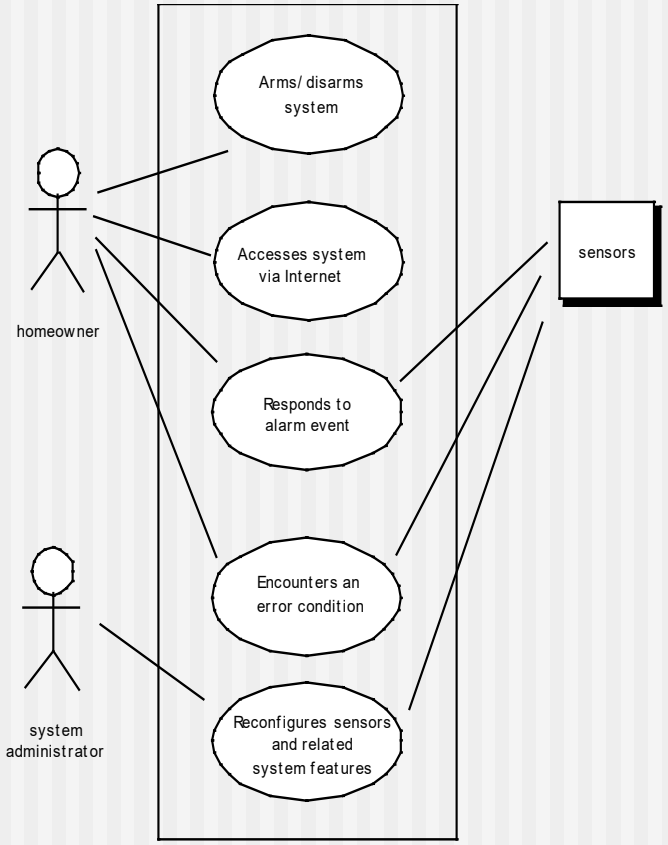
Elicitation Work Products

- 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.

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?

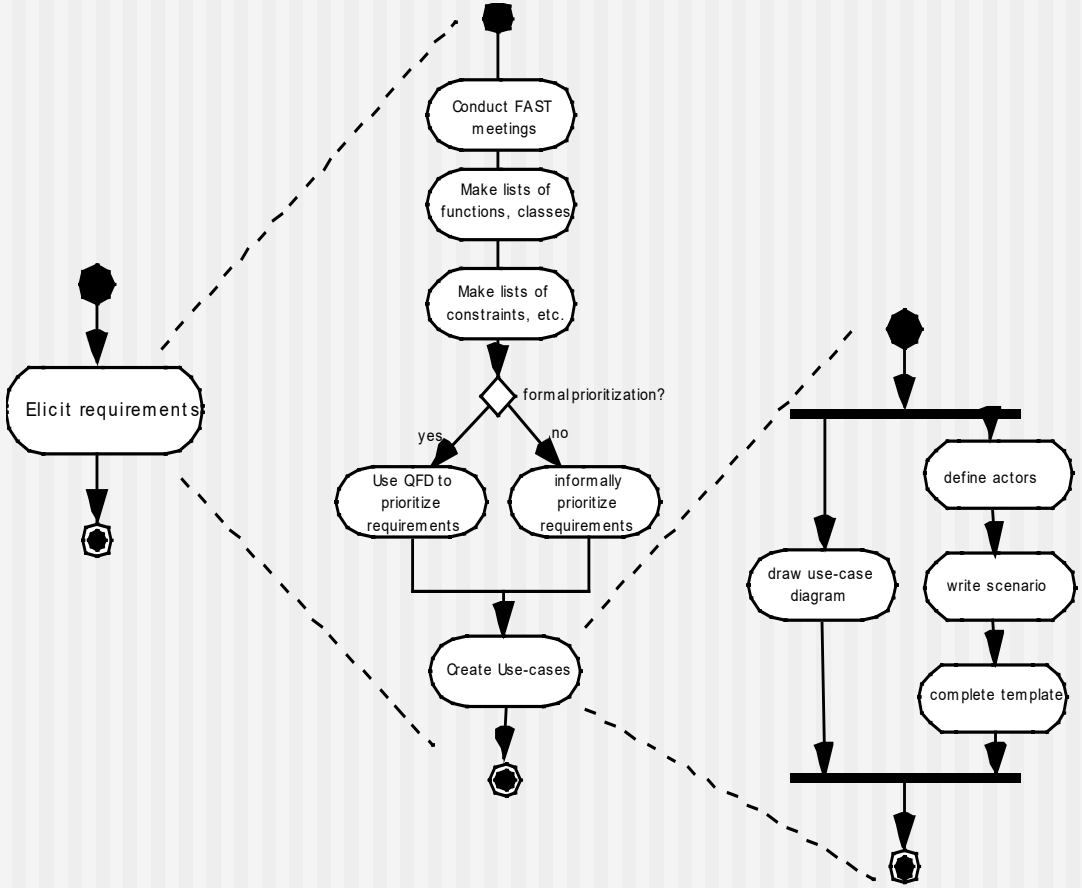
Case: Use-Case Diagram



Building the Analysis Model

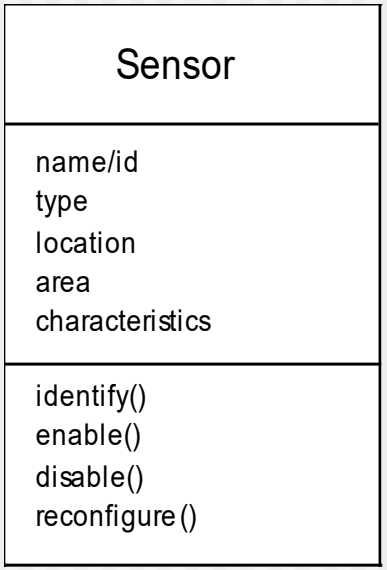
- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an "actor" and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
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

Case: Eliciting Requirements

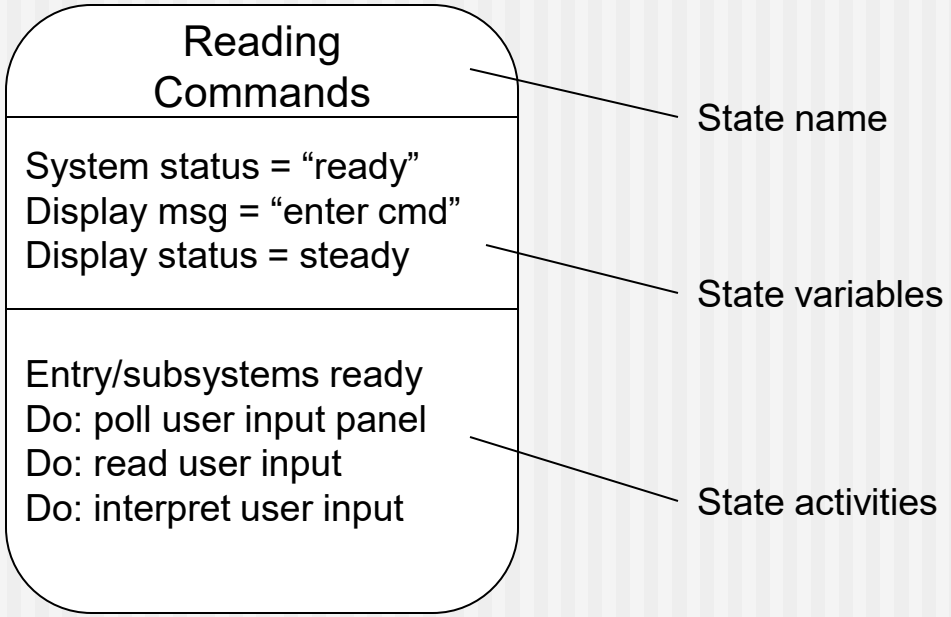


Case: Class Diagram

From the *SafeHome* system …



Case: State Diagram



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 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.

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"

Example: Art of Negotiation

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 co-design* – provides information to stakeholders as the system evolves.

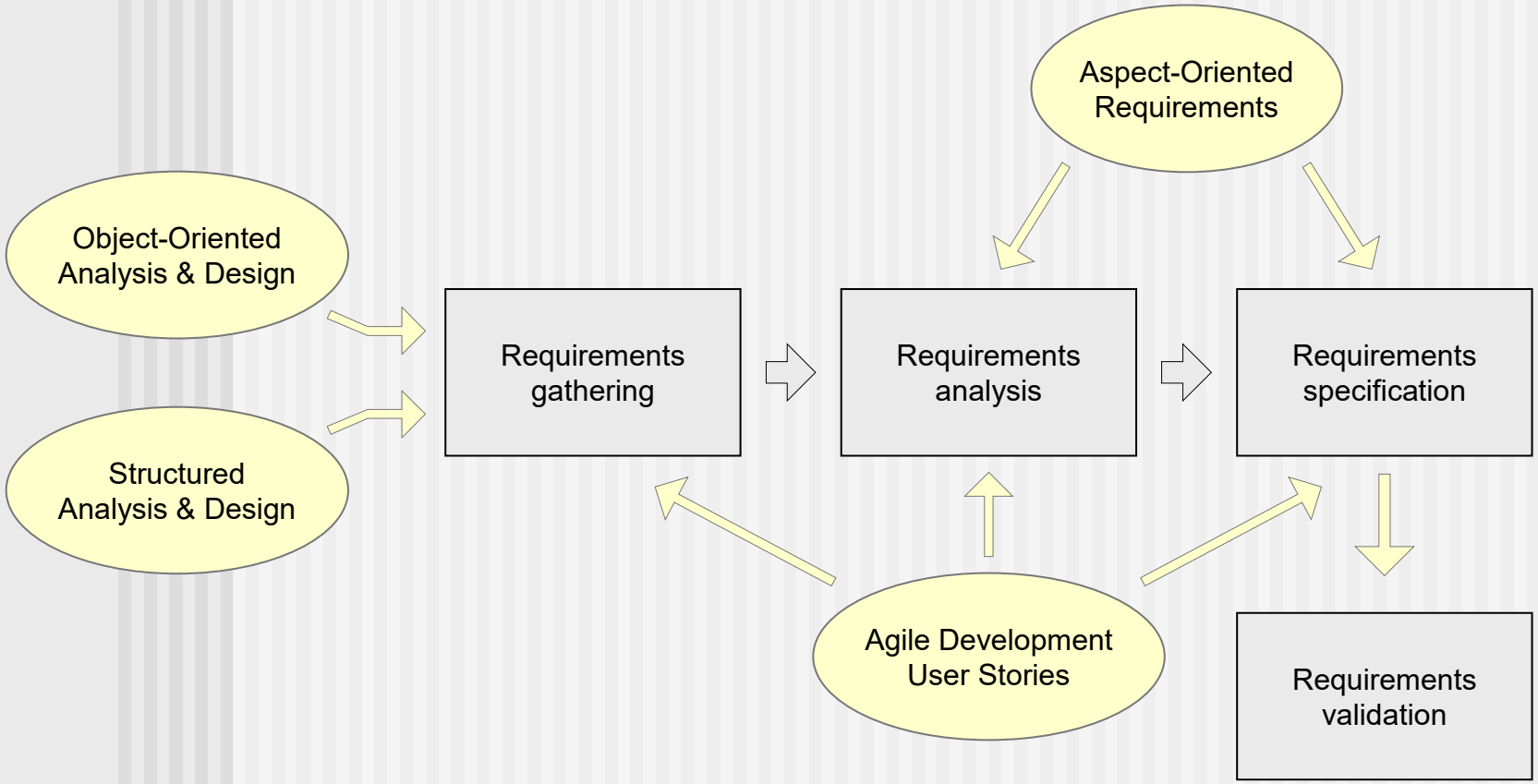
Validating Requirements - I

- 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?

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?

Conclusion: Requirements Process



Summary

- Requirements engineering helps software engineers better understand the problems they are trying to solve. Building an elegant computer solution that ignores the customer's needs helps no one.
- It is very important to understand the customer's wants and needs before you begin designing or building a computer-based solution.
- The requirements engineering process begins with inception, moves on to elicitation, negotiation, problem specification, and ends with review or validation of the specification.
- The intent of requirements engineering is to produce a written understanding of the customer's problem.
- Several different work products might be used to communicate this understanding (user scenarios, function and feature lists, analysis models, or specifications).

Assignment

Please choose one of the following activities as your assignment.

From textbook《Software Engineering》9Edition page124.

7.5 Develop a complete use case for one of the following activities:

a) Making a withdrawal at an ATM.

b) Using your charge card for a meal at a restaurant.

c) Buying a stock using an online brokerage account.

- Preview

《Software Engineering》(8th Edition)

Chapter 9 Requirements Modeling: Scenario-based methods

by R.S.Pressman