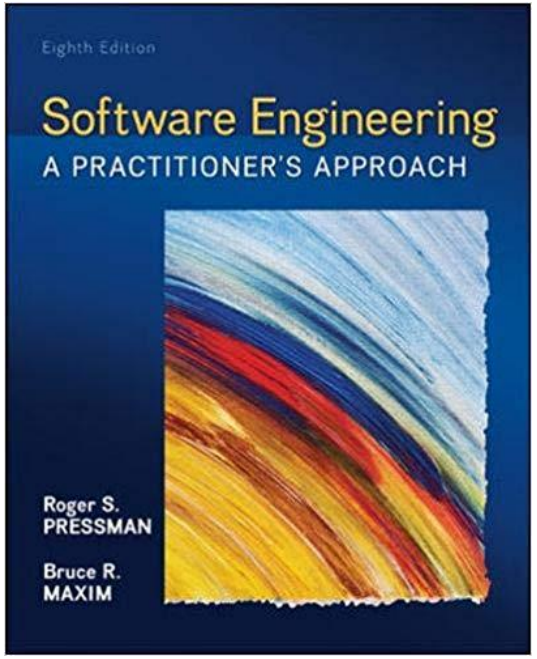


Requirement Analysis

CS46 Software Engineering

Dr. Shilpa chaudhari

Department of Computer Science and Engineering
RIT, Bangalore

Outline of Session-2

- **Understanding Requirements**: Requirements Engineering, Eliciting Requirements, Developing Use Cases, Building the Analysis Model, Negotiating Requirements, Requirements Monitoring, Validating Requirements, Avoiding Common Mistakes.
  - 8.1, 8.3, 8.4, 8.5 8.6, 8.9
- **Scenario-Based Requirements Modeling**: Requirements Analysis, Scenario-Based Modeling, UML Models That Supplement the Use Case.
  - 9.1, 9.2, 9.3
- **Requirements Modeling for Web and Mobile Apps**
  - 11.5
- Applying requirement engineering on the same case study of Unit-1.

Understanding Requirements

The Problems with our Requirements Practices

- We have trouble understanding the requirements that we do acquire from the customer
- We often record requirements in a disorganized manner
- We spend far too little time verifying what we do record
- We allow change to control us, rather than establishing mechanisms to control change
- Most importantly, we fail to establish a solid foundation for the system or software that the user wants built

The Problems with our Requirements Practices

- Many software developers argue that
  - Building software is so compelling that we want to jump right in (before having a clear understanding of what is needed)
  - Things will become clear as we build the software
  - Project stakeholders will be able to better understand what they need only after examining early iterations of the software
  - Things change so rapidly that requirements engineering is a waste of time
  - The bottom line is producing a working program and that all else is secondary
- All of these arguments contain some truth, especially for small projects that take less than one month to complete
- However, as software grows in size and complexity, these arguments begin to break down and can lead to a failed software project

A Solution: Requirements Engineering

- Begins during the communication activity and continues into the modeling activity
- Allows the requirements engineer to examine
  - the context of the software work to be performed
  - the specific needs that design and construction must address
  - the priorities that guide the order in which work is to be completed
  - the information, function, and behavior that will have a profound impact on the resultant design

- ***The requirements engineering process can be described in seven distinct steps:***
  - Inception
  - Elicitation
  - Elaboration
  - Negotiation
  - Specification
  - Validation
  - Requirements Management

Requirements Engineering Tasks

1. **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
2. **Elicitation**—elicit requirements from all stakeholders
3. **Elaboration**—create an analysis model that identifies data, function and behavioral requirements
4. **Negotiation**—agree on a deliverable system that is realistic for developers and customers

Requirements Engineering Tasks

5. **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
6. **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.
7. **Requirements management**

Requirements Engineering Tasks

- Some of these tasks may occur in parallel and all are adapted to the needs of the project
- All strive to define what the customer wants
- All serve to establish a solid foundation for the design and construction of the software

Establishing the Groundwork

- During inception, the requirements engineer asks a set of questions to establish…
  - A basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired
  - The effectiveness of preliminary communication and collaboration between the customer and the developer
- Through these questions, the requirements engineer needs to…
  - Identify the stakeholders
  - Recognize multiple viewpoints
  - Work toward collaboration
  - Break the ice and initiate the communication –ask Questions
  - Nonfunctional requirements
  - traceability

Establishing the Groundwork

- **Identify the stakeholders** - business operations managers, product managers, marketing people, internal and external customers, end users consultants, product engineers, software engineers, support and maintenance engineers
  - Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail
- **Recognizing Multiple Viewpoints -** requirements of the system will be explored from many different points of view each stakeholder - emerging requirements may be inconsistent or may conflict with one another
  - You should categorize all stakeholder information in a way that will allow decision makers to choose an internally consistent set of requirements for the system.

Establishing the Groundwork

- **Working toward Collaboration:** Customers must collaborate among themselves and with software engineering practitioners if a successful system is to result
  - The job of a requirements engineer is to identify areas of commonality and areas of conflict or inconsistency
  - Collaboration does not necessarily mean that requirements are defined by committee
    - In many cases, stakeholders collaborate by providing their view of requirements, but a strong “project champion” may make the final decision about which requirements make the cut

Establishing the Groundwork

The First Set of Questions - focus on the customer, other stakeholders, the overall goals, and the benefits

- 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?

Establishing the Groundwork

The Next Set of Questions - enable the requirements engineer to gain a better understanding of the problem and allow the customer to voice his or her perceptions about a solution

- How would you characterize "good" output that would be generated by a successful solution?
- What problem(s) will this solution address?
- Can you show me (or describe) the business environment in which the solution will be used?
- Will special performance issues or constraints affect the way the solution is approached?

Establishing the Groundwork

The Final Set of Questions - focus on the effectiveness of the communication activity itself

- Are you the right person to answer these questions? Are your answers "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

These questions will help to “break the ice” and initiate the communication that is essential to successful elicitation

Establishing the Groundwork

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

- **Traceability** - refers to documented links between software engineering work products
  - A traceability matrix allows a requirements engineer to represent the relationship between requirements and other software engineering work products (Rows - requirement names while columns - software engineering work product)

Eliciting Requirements

- Requirements elicitation combines elements of problem solving, elaboration, negotiation, and specification
- Elicitation may be accomplished through two activities
  - Collaborative requirements gathering
  - Quality function deployment

Eliciting Requirements - **Collaborative Requirements Gathering**

- Meetings are conducted and attended by both software engineers, customers, and other interested stakeholders
- Rules for preparation and participation are established
- An 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" is used such as work sheets, flip charts, wall stickers, electronic bulletin board, chat room, or some other virtual forum
- 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 - Collaborative Requirements Gathering

- A one- or two-page “product request” is generated during inception
- A meeting place, time, and date are selected
- a facilitator is chosen; and attendees from the software team and other stakeholder organizations are invited to participate
- The product request is distributed to all attendees before the meeting date

Case Study : *SafeHome*

- Product request written by a marketing person involved in the *SafeHome*

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, can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected. *e project*

Case Study : *SafeHome*

- While reviewing the product request in the days before the meeting, each attendee is asked to make a list of objects that are part of the environment that surrounds the system, other objects that are to be produced by the system, and objects that are used by the system to perform its functions

SafeHome

Conducting a Requirements-Gathering Meeting

The scene: A meeting room. The first requirements-gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator (pointing at whiteboard): So that's the current list of objects and services for the home security function.

Marketing person: That about covers it from our point of view.

Vinod: Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

Marketing person: Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

Facilitator: Does that also add some constraints?

Jamie: It does, both technical and legal.

Production rep: Meaning?

Jamie: We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part.

Doug: Very true.

Marketing: But we still need that . . . just be sure to stop an outsider from getting in.

Ed: That's easier said than done and . . .

Facilitator (interrupting): I don't want to debate this issue now. Let's note it as an action item and proceed.

(Doug, serving as the recorder for the meeting, makes an appropriate note.)

Facilitator: I have a feeling there's still more to consider here.

(The group spends the next 20 minutes refining and expanding the details of the home security function.)

Eliciting Requirements - Quality Function Deployment

- This is a technique that translates the needs of the customer into technical requirements for software
- It emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process through functions, information, and tasks
- It identifies three types of requirements
  - Normal requirements: These requirements are the objectives and goals stated for a product or system during meetings with the customer
  - Expected requirements: These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them
  - Exciting requirements: These requirements are for features that go beyond the customer's expectations and prove to be very satisfying when present

Eliciting Requirements - **Usage Scenarios**

- how these functions and features will be used by different classes of end users?
  - developers and users can create a set of scenarios (*use cases)* that identify a thread of usage for the system to be constructed

SafeHome

Developing a Preliminary User Scenario

The scene: A meeting room, continuing the first requirements gathering meeting.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator: We've been talking about security for access to *SafeHome* functionality that will be accessible via the Internet. I'd like to try something. Let's develop a usage scenario for access to the home security function.

Jamie: How?

Facilitator: We can do it a couple of different ways, but for now, I'd like to keep things really informal. Tell us (he points at a marketing person) how you envision accessing the system.

Marketing person: Um . . . well, this is the kind of thing I'd do if I was away from home and I had to let someone into the house, say a housekeeper or repair guy, who didn't have the security code.

Facilitator (smiling): That's the reason you'd do it . . . tell me how you'd actually do this.

Marketing person: Um . . . the first thing I'd need is a PC. I'd log on to a website we'd maintain for all users of *SafeHome*. I'd provide my user ID and . . .

Vinod (interrupting): The Web page would have to be secure, encrypted, to guarantee that we're safe and . . .

Facilitator (interrupting): That's good information, Vinod, but it's technical. Let's just focus on how the end user will use this capability. OK?

Vinod: No problem.

Marketing person: So as I was saying, I'd log on to a website and provide my user ID and two levels of passwords.

Jamie: What if I forget my password?

Facilitator (interrupting): Good point, Jamie, but let's not address that now. We'll make a note of that and call it an *exception*. I'm sure there'll be others.

Marketing person: After I enter the passwords, a screen representing all *SafeHome* functions will appear. I'd select the home security function. The system might request that I verify who I am, say, by asking for my address or phone number or something. It would then display a picture of the security system control panel along with a list of functions that I can perform—arm the system, disarm the system, disarm one or more sensors. I suppose it might also allow me to reconfigure security zones and other things like that, but I'm not sure.

(As the marketing person continues talking, Doug takes copious notes; these form the basis for the first informal usage scenario. Alternatively, the marketing person could have been asked to write the scenario, but this would be done outside the meeting.)

Eliciting Requirements - Work Products

Vary depending on the size of the system or product to be built but should include one or more of the following items

- 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.

Eliciting Requirements - Agile Requirements Elicitation

- All stakeholders asked to create *user stories that describes a simple system* requirement written from the user's perspective
  - Often lacks overall business goals and nonfunctional requirements
  - rework is required to accommodate performance and security issues
  - user stories may not provide a sufficient basis for system evolution over time

Eliciting Requirements - Service-Oriented Methods

- views a system as an aggregation of services
- service design methods emphasize understanding the customer, thinking creatively, and building solutions quickly
- focuses on the definition of services to be rendered by an application

Elaboration Task

- During elaboration, the software engineer takes the information obtained during inception and elicitation and begins to expand and refine it
- Elaboration focuses on developing a refined technical model of software functions, features, and constraints
- It is an **analysis modeling task**
  - Use cases are developed
  - Domain classes are identified along with their attributes and relationships
  - State machine diagrams are used to capture the life on an object
- The end result is an analysis model that defines the functional, informational, and behavioral domains of the problem

Developing Use Cases

- A Use case represents the functionality of the system.
  - 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
- Step One – Define the set of actors that will be involved in the story
  - Actors are people, devices, or other systems that use the system or product within the context of the function and behavior that is to be described
  - Actors are anything that communicate with the system or product and that are external to the system itself
  - Primary actors interact to achieve required system function and derive the intended benefit from the system - They work directly and frequently with the software
  - Secondary actors support the system so that primary actors can do their work.
- Step Two – Develop use cases, where each one answers a set of questions

Developing Use Cases

- 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?

Example: SafeHome

SafeHome - Actor

Four actors:

- **Homeowner** (a user)
- **setup manager (likely the same person as homeowner, but playing a** different role)
- **sensors (devices attached to the system)**
- **the monitoring and response subsystem (the central station that monitors the *SafeHome home security*** function).

SafeHome - homeowner Actor

(1) enters a password to allow all other interactions,

(2) inquires about the status of a security zone,

(3) inquires about the status of a sensor,

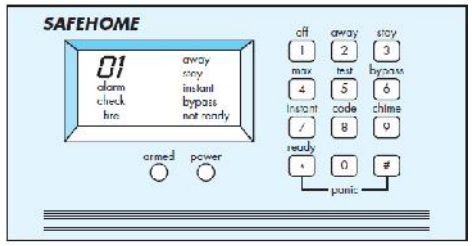
(4) presses the panic button in an emergency,

(5) activates/deactivates the security system.

- basic use case presents a high-level story that describes the interaction between the actor and the system

SafeHome - homeowner uses the control panel

- basic use case for system activation follows



1. The homeowner observes the *SafeHome* control panel (Figure 8.1) 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 8.1) 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.

SafeHome - homeowner uses the control panel

Use case: *InitiateMonitoring*

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, that is, 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 *SafeHome* has been armed

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.

- uses cases are further elaborated to provide considerably more detail about the interaction

SafeHome - homeowner uses the control panel

Priority: Essential, must be implemented

When available: First increment

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

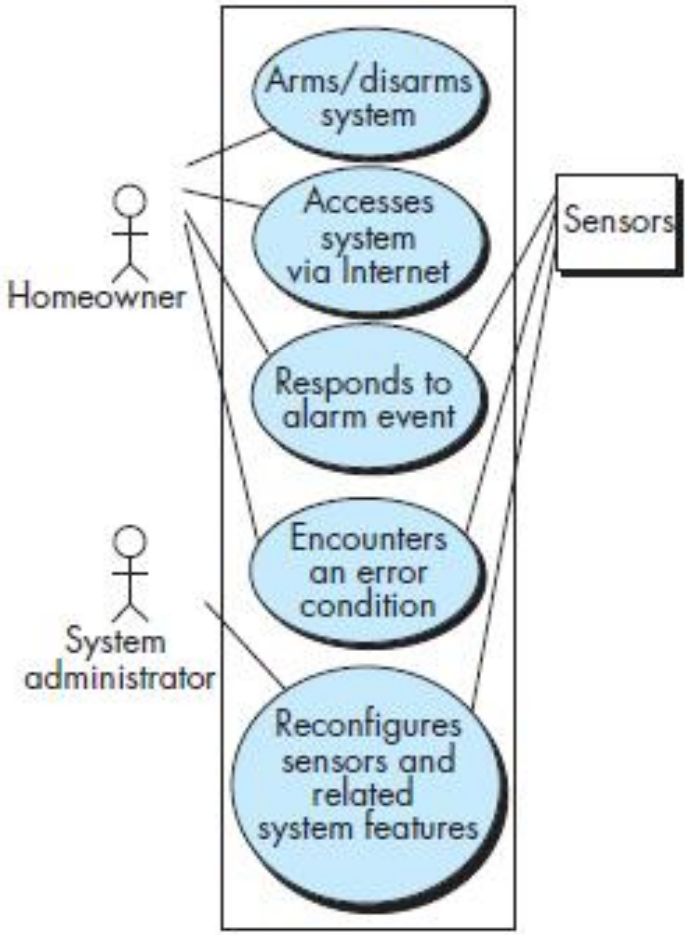
Open issues:

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?

- Use cases for other homeowner interactions would be developed in a similar manner

Use Case Diagram for *SafeHome*

- **UML use case diagram for *SafeHome* home security function**



Building the Analysis Model

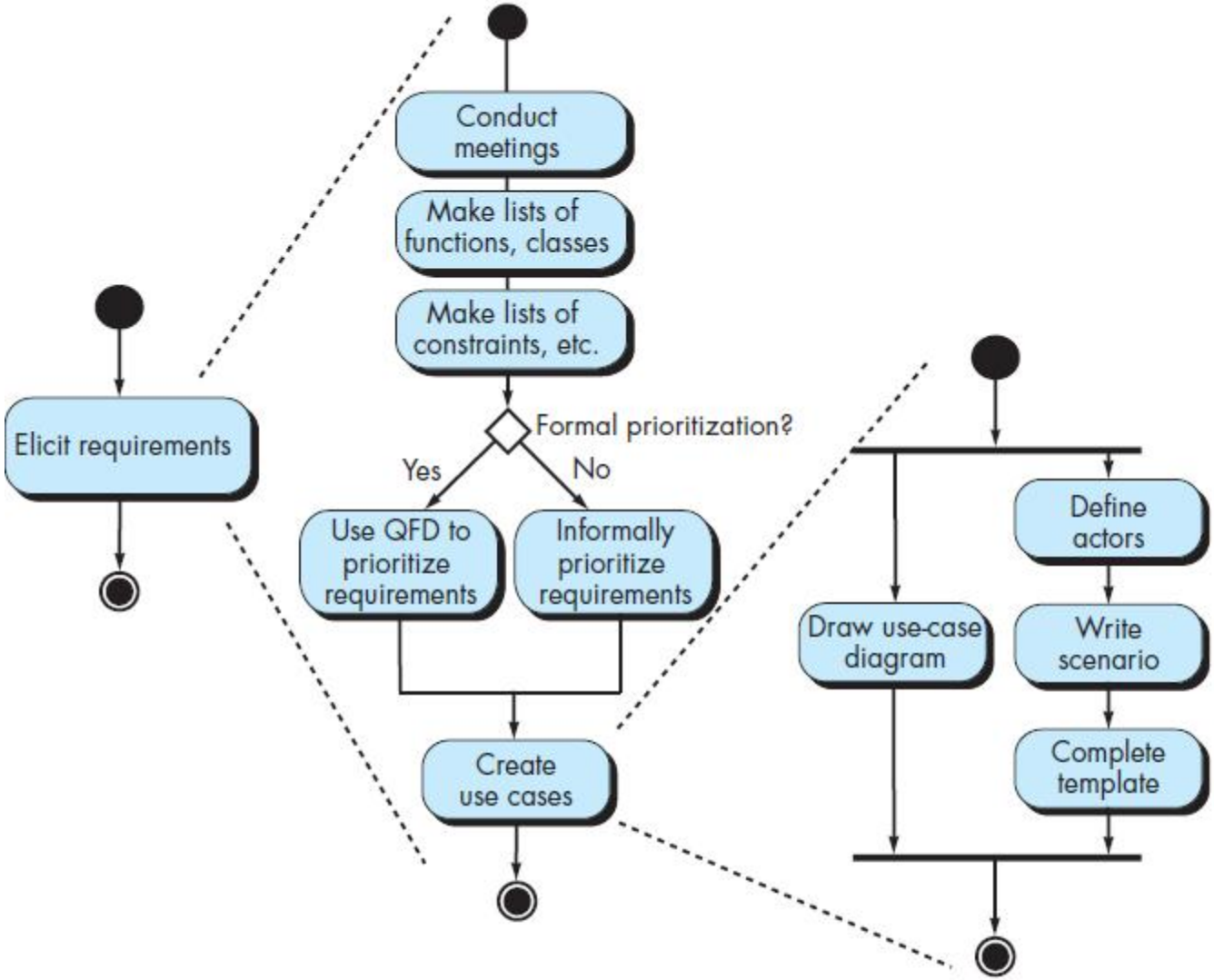
- Analysis model provides a description of the required informational, functional, and behavioral domains for a computer-based system
- approach to build analysis model should have a higher probability of uncovering omissions, inconsistencies, and ambiguity
- Description of analysis model and the methods that are used to build it includes
  - Elements of the Analysis Model
  - Analysis Patterns
  - Agile Requirements Engineering
  - Requirements for Self-Adaptive Systems

Elements of the Analysis Model

- Scenario-based elements- Describe the system from the user's point of view using scenarios that are depicted in use cases and activity diagrams
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an “actor” and the system
- Class-based elements- Implied by scenarios
    - Identify the domain classes for the objects manipulated by the actors, the attributes of these classes, and how they interact with one another; they utilize class diagrams to do this
- Behavioral elements - Use state diagrams to represent the state of the system, the events that cause the system to change state, and the actions that are taken as a result of a particular event; can also be applied to each class in the system
- Flow-oriented elements - Use data flow diagrams to show the input data that comes into a system, what functions are applied to that data to do transformations, and what resulting output data are produced

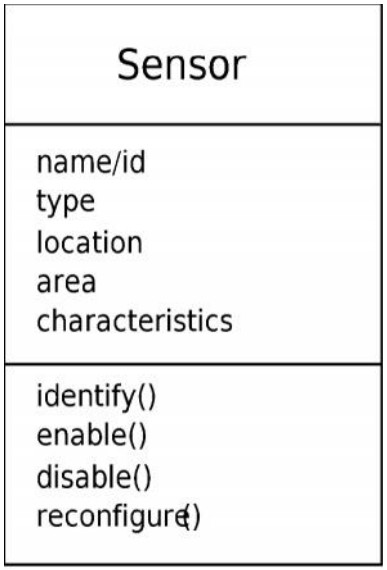
Scenario-based elements

- serve as input for the creation of other modeling elements
- UML activity diagrams for eliciting requirements and representing them using use cases.



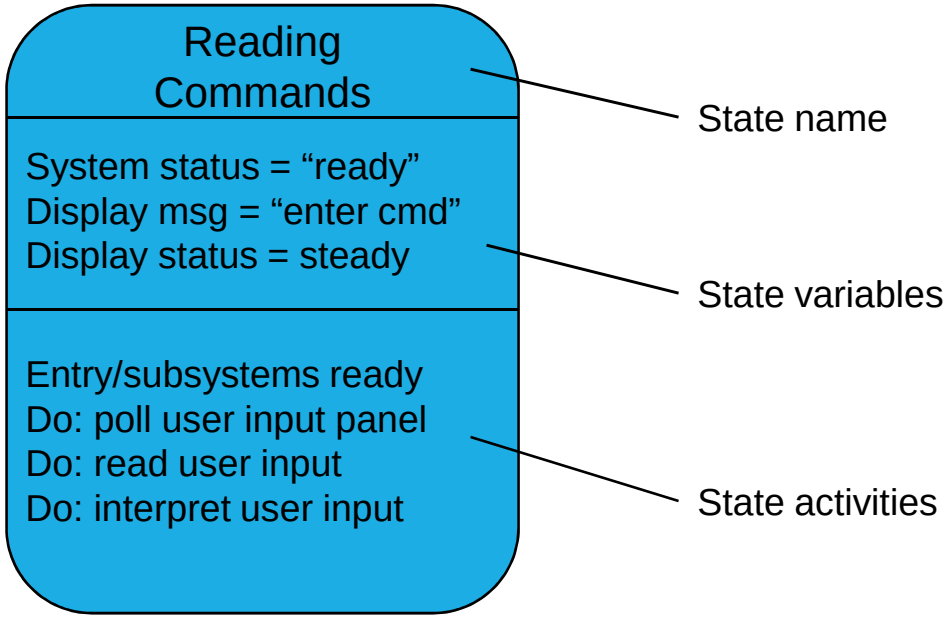
Class-based elements

- depict the manner in which classes collaborate with one another and the relationships and interactions between classes
- **From the *SafeHome* system …**

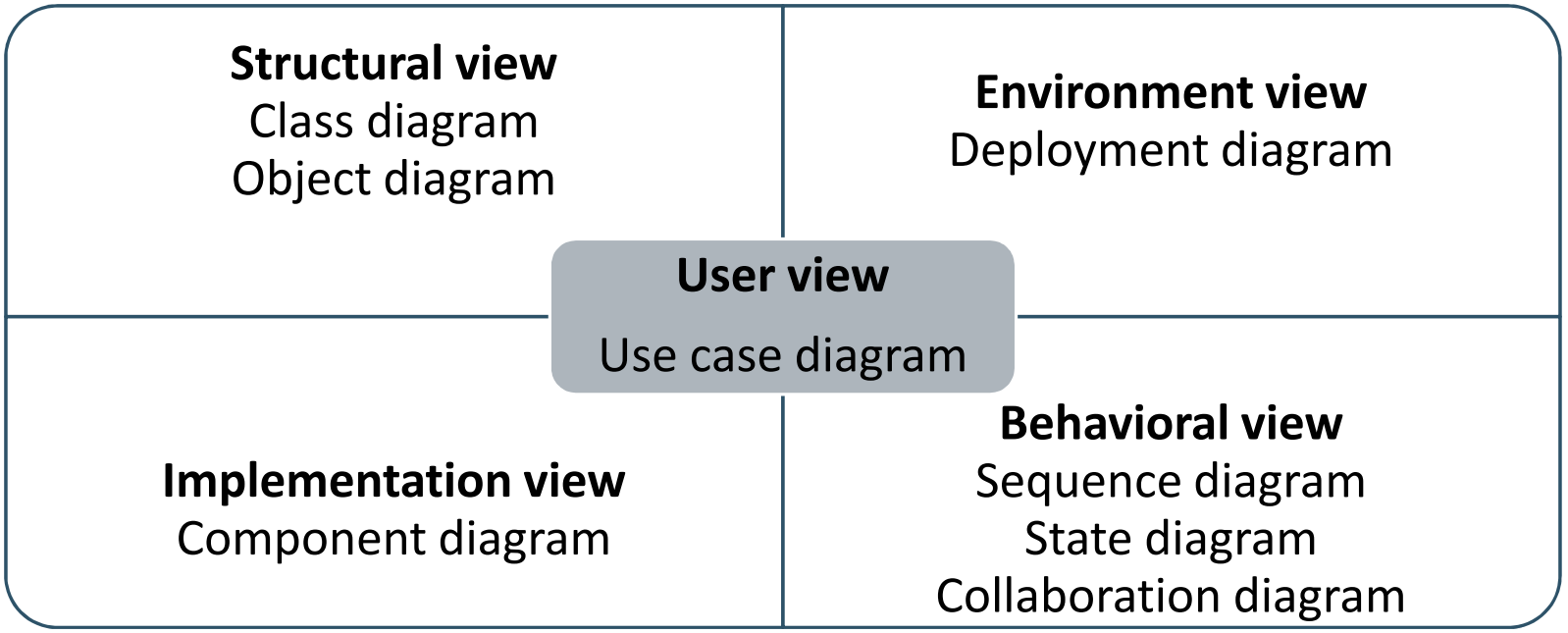


Behavioral elements

- behavioral representations of the system as a whole
- Behavior of individual classes
- Simple UML state diagram



Different mode of representation



Analysis Patterns

- *If you want to obtain solutions to customer requirements more rapidly and provide your team with proven approaches, use analysis patterns.*
- *Benefits*
  - speed up the development of abstract analysis models
  - facilitate the transformation of the analysis model into a design model by suggesting design patterns and reliable solutions for common problems

Agile Requirements Engineering

- transfer ideas from stakeholders to the software team rather than create extensive analysis work products
- allows the early creation and testing of working prototypes
- addresses important issues
  - high requirements volatility
  - incomplete knowledge of development technology
  - customers not able to articulate their visions until they see a working prototype

Requirements for Self-Adaptive Systems

- *Self-adaptive systems can*
  - *reconfigure themselves*
  - *augment their functionality*
  - protect themselves
  - recover from failure
  - accomplish all of this while hiding most of their internal complexity from their users
- document the variability needed for self-adaptive systems include
  - timing uncertainty
  - user profile differences (e.g., end users versus systems administrators)
  - behavior changes based on problem domain (e.g., commercial or educational)
  - or pre-defined behaviors exploiting system assets
  - More variability increases complexity

Negotiating Requirements

- develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team
- set of negotiation activities
  - 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”

Negotiating Requirements

- Handshaking - tends to improve identification, analysis, and selection of variants and promotes win-win negotiation
  - the software team
    - proposes solutions to requirements
    - describes their impact
    - communicates their intentions to customer representatives.
  - customer representatives
    - review the proposed solutions, focusing on missing features
    - seeking clarification of novel requirements

Guidelines to negotiate effectively

- *Recognize that it's not a competition. To be successful,* both parties have to feel they've won or achieved something. Both will have to compromise.
- *Map out a strategy. Decide what you'd like to* achieve, what the other party wants to achieve, and how you'll go about making both happen.
- *Listen actively. Don't work on formulating your* response while the other party is talking. Listen to her. It's likely you'll gain knowledge that will help you to better negotiate your position.
- *Focus on the other party's interests. Don't take hard* positions if you want to avoid conflict.
- *Don't let it get personal. Focus on the problem that* needs to be solved.
- *Be creative. Don't be afraid to think out of the box* if you're at an impasse.
- *Be ready to commit. Once an agreement has been* reached, don't waffle; commit to it and move on.

Requirements Monitoring

Especially needes in incremental development that encompasses five tasks

- *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.

Validating Requirements

- A review of the requirements model addresses the following questions: to ensure that the requirements model is an accurate reflection of stakeholder needs and that it provides a solid foundation for design
- 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?

Validating Requirements

- 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?

Avoiding Common Mistakes

- Featuritis - the practice of trading functional coverage for overall system quality
- Flexibilitis - happens when software engineers overload product with adaptation and configuration facilities
- Performitis - occurs when software developers become overly focused on system performance at the expense of quality attributes like maintainability, reliability, or security

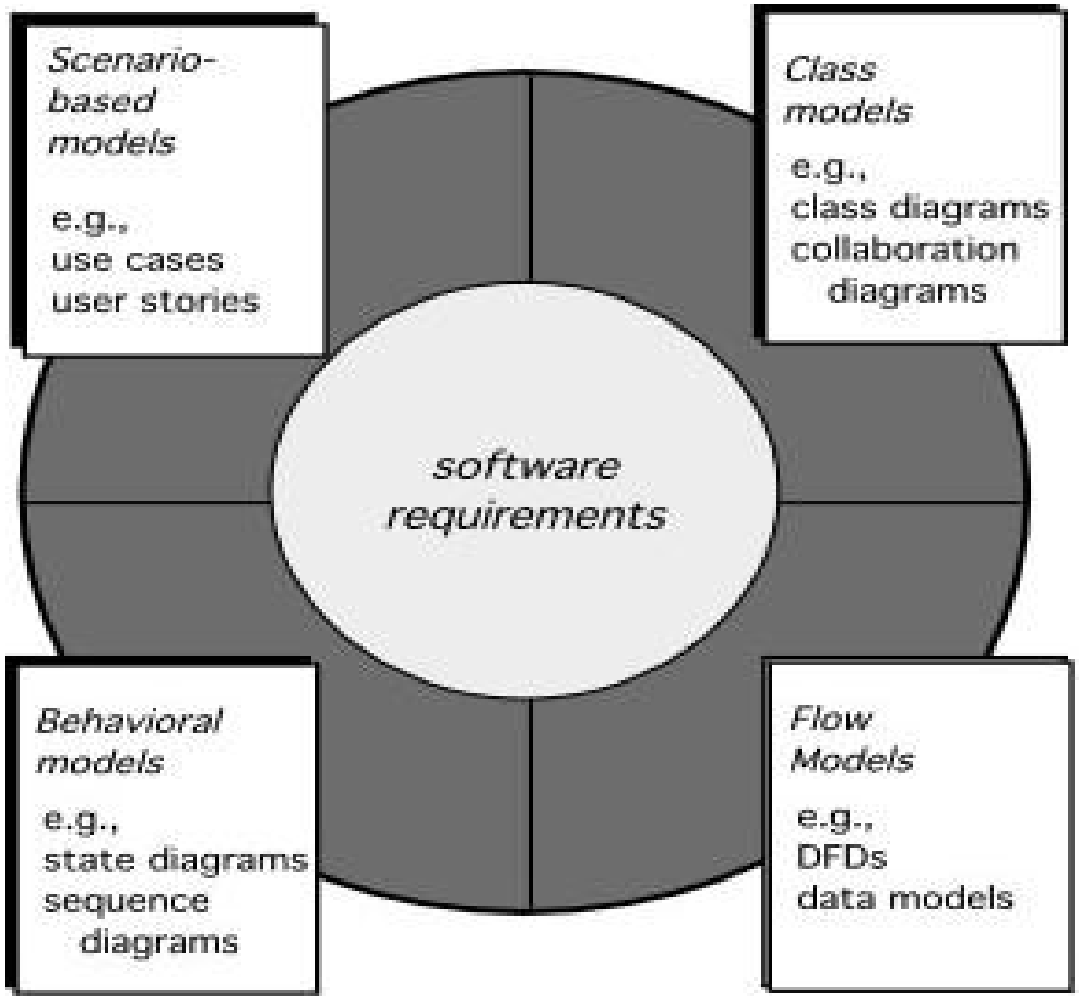
Scenario-Based Requirements Modeling

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.

57

Elements of Requirements Analysis



- Scenario-based - system from the user's point of view
- Data - shows how data are transformed inside the system
- Class-oriented - defines objects, attributes, and relationships
- Flow-oriented - shows how data are transformed inside the system
- Behavioral - show the impact of events on the system states

58

Requirement Analysis -Overall Objectives and Philosophy

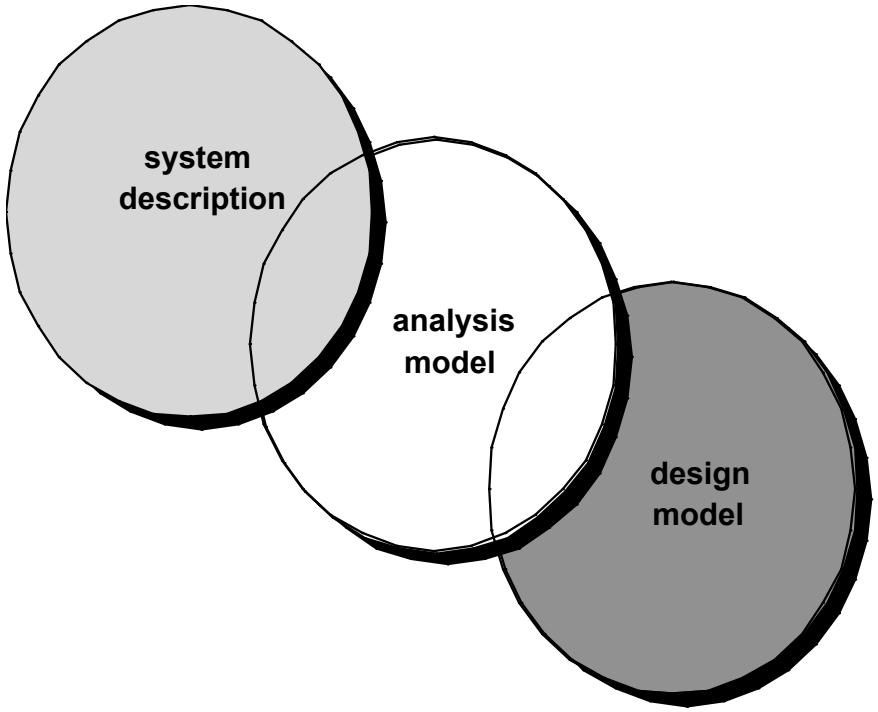
- Primary focus is on *what, not how*
  - What user interaction occurs in a particular circumstance
  - what objects does the system manipulate
  - what functions must the system perform
  - what behaviors does the system exhibit
  - what interfaces are defined
  - what constraints apply?
- Requirements model must achieve three primary objectives:

(1) to describe what the customer requires

(2) to establish a basis for the creation of a software design, and

(3) to define a set of requirements that can be validated once the software is built.

Analysis Model - A Bridge



Analysis model bridges the gap between

1. a system- level description that describes overall system or business functionality as it is achieved by applying software, hardware, data, human, and other system elements
2. a software design that describes the software's application architecture, user interface, and component-level structure

60

Analysis Model - 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.

61

Analysis Model - Domain Analysis

- Goal - to find or create the analysis classes and/or analysis patterns that are broadly applicable so that they may be reused
- May be viewed as an umbrella activity for the software process

Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain . . . [Object-oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks . . .

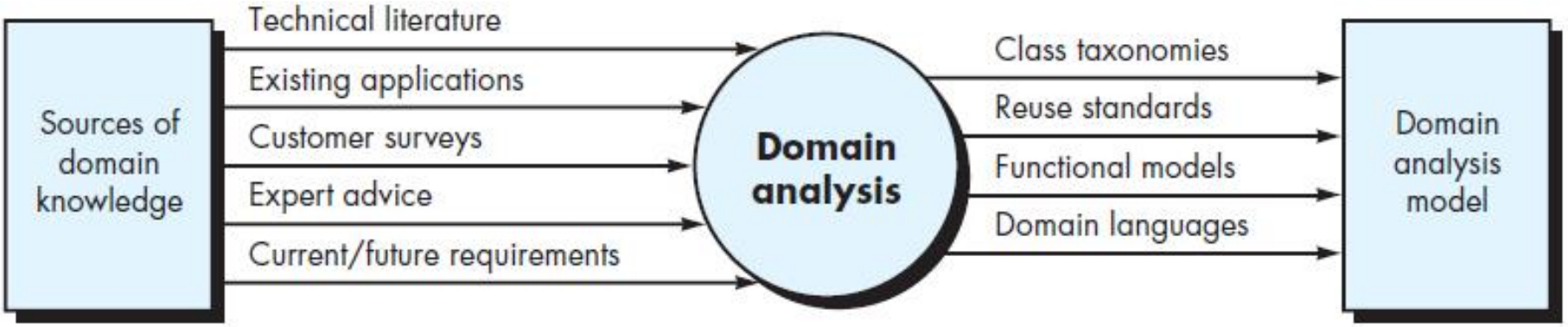
Donald Firesmith

Domain Analysis

Role of the domain analyst -

- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.
- Analyze each application in the sample.
- Develop an analysis model for the objects

Input and output for domain analysis

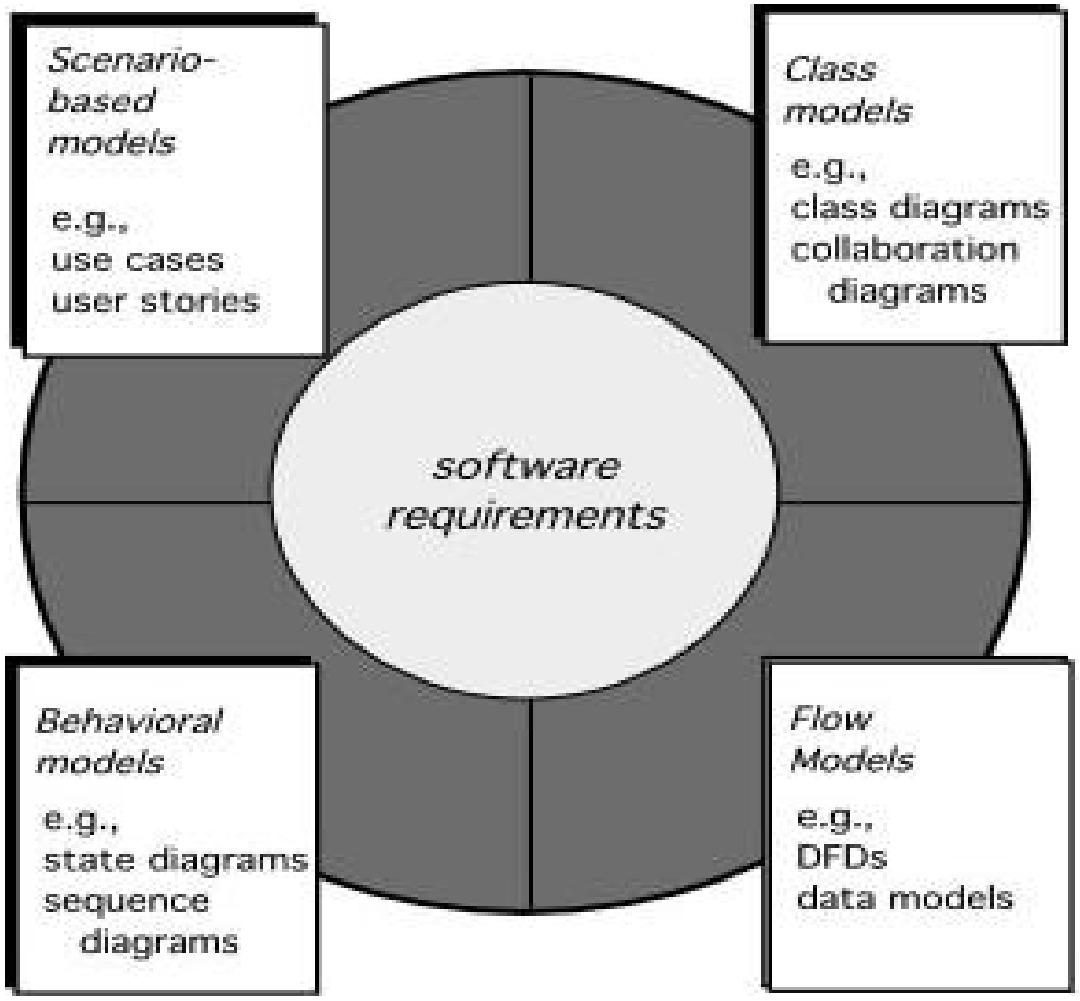


63

Requirements Modeling Approaches

- *structured analysis*
  - considers data and the processes that transform the data as separate entities
    - Data objects are modeled in a way that defines their attributes and relationships.
    - Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- *object-oriented analysis*
  - focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements.
  - UML and the Unified Process are predominantly object oriented

Element of the requirements model



- presents the problem from a different point of view

Scenario-Based Modeling

- How to measure success of a computer-based system or product?
  - Many ways, user satisfaction resides at the top of the list
    - requirements modeling with UML can be used to understand user satisfaction that begins with the creation of scenarios in the form of use cases, activity diagrams, and swimlane diagrams
- We noted that a use case describes a specific usage scenario in straightforward language from the point of view of a defined actor.
  - Each use case is represented by an oval.
- But how do you know (1) what to write about, (2) how much to write about it, (3) how detailed to make your description, and (4) how to organize the description?

66

What to Write About?

- Inception and elicitation—provide you with the information you'll need to begin writing use cases.
- Requirements gathering meetings, quality function deployment (QFD), and other requirements engineering mechanisms are used to
  - identify stakeholders
  - define the scope of the problem
  - specify overall operational goals
  - establish priorities
  - outline all known functional requirements, and
  - describe the things (objects) that will be manipulated by the system.
- To begin developing a set of use cases, list the functions or activities performed by a specific actor.

67

Developing Another Preliminary User Scenario

The scene: A meeting room, during the second requirements-gathering meeting.

The players: Jamie Lazar, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator: It's time that we begin talking about the *SafeHome* surveillance function. Let's develop a user scenario for access to the surveillance function.

Jamie: Who plays the role of the actor on this?

Facilitator: I think Meredith (a marketing person) has been working on that functionality. Why don't you play the role?

Meredith: You want to do it the same way we did it last time, right?

Facilitator: Right . . . same way.

Meredith: Well, obviously the reason for surveillance is to allow the homeowner to check out the house while he or she is away, to record and play back video that is captured . . . that sort of thing.

Ed: Will we use compression to store the video?

Facilitator: Good question, Ed, but let's postpone implementation issues for now. Meredith?

Meredith: Okay, so basically there are two parts to the surveillance function . . . the first configures the system including laying out a floor plan—we have to have tools to help the homeowner do this—and the second part is the actual surveillance function itself. Since the layout is part of the configuration activity, I'll focus on the surveillance function.

Facilitator (smiling): Took the words right out of my mouth.

Meredith: Um . . . I want to gain access to the surveillance function either via the PC or via the Internet. My feeling is that the Internet access would be more frequently used. Anyway, I want to be able to display camera views on a PC and control pan and zoom for a specific camera. I specify the camera by selecting it from the house floor plan. I want to selectively record camera output and replay camera output. I also want to be able to block access to one or more cameras with a specific password. I also want the option of seeing small windows that show views from all cameras and then be able to pick the one I want enlarged.

Jamie: Those are called thumbnail views.

Meredith: Okay, then I want thumbnail views of all the cameras. I also want the interface for the surveillance function to have the same look and feel as all other *SafeHome* interfaces. I want it to be intuitive, meaning I don't want to have to read a manual to use it.

Facilitator: Good job. Now, let's go into this function in a bit more detail . . .

What to Write About?

- *SafeHome home surveillance function* identifies the following functions that are performed by the **homeowner actor**
  - Select camera to view.
  - Request thumbnails from all cameras.
  - Display camera views in a PC window.
  - Control pan and zoom for a specifi c camera.
  - Selectively record camera output.
  - Replay camera output.
  - Access camera surveillance via the Internet.
- develop use cases for each of the functions noted in informal narrative fashion or in structured format.

What to Write About?

- stakeholder who takes on the role of the homeowner actor might write the following narrative for the function *access camera surveillance via the Internet—display camera views (ACS-DCV)*

Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)

Actor: homeowner

If I'm at a remote location, I can use any PC with appropriate browser software to log on to the *SafeHome Products* website. I enter my user ID and two levels of passwords and once I'm validated, I have access to all functionality for my installed *SafeHome* system. To access a specific camera view, I select "surveillance" from the major function buttons displayed. I then select "pick a camera" and the floor plan of the house is displayed. I then select the camera that I'm interested in. Alternatively, I can look at thumbnail snapshots from all cameras simultaneously by selecting "all cameras" as my viewing choice. Once I choose a camera, I select "view" and a one-frame-per-second view appears in a viewing window that is identified by the camera ID. If I want to switch cameras, I select "pick a camera" and the original viewing window disappears and the floor plan of the house is displayed again. I then select the camera that I'm interested in. A new viewing window appears.

What to Write About?

- Revisiting the **ACS-DCV function** as an ordered sequence of user actions

1. The homeowner logs onto the *SafeHome Products website.*
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the “surveillance” from the major function buttons.
6. The homeowner selects “pick a camera.”
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the “view” button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

How Much to Write About?

- As further conversations with the stakeholders progress, the requirements gathering team develops use cases for each of the functions noted.
- In general, use cases are written first in an informal narrative fashion.
- If more formality is required, the same use case is rewritten using a structured format similar to the one proposed.

72

Refining a Preliminary Use Case

- Use-cases are written first in narrative form and mapped to a template if formality is needed
- Each primary scenario should be reviewed and refined to see if alternative interactions are possible
  - Can the actor take some other action at this point?
  - Is it possible that the actor will encounter an error condition at some point? If so, what?
  - Is it possible that the actor will encounter some other behavior at some point? If so, what?
- Each of the answer can be characterized as a use case exception.
  - An *exception describes a situation (either a failure condition* or an alternative chosen by the actor) that causes the system to exhibit somewhat different behavior

73

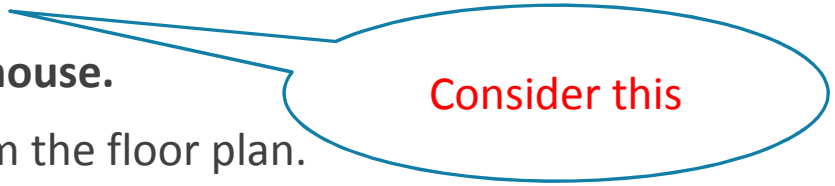
Refining a Preliminary Use Case

- Explore following issues through “brainstorming” session to derive a reasonably complete set of exceptions for each use case
  - *Are there cases in which some “validation function” occurs during this use case?*
    - Eg .Occurrence of potential error condition
  - *Are there cases in which a supporting function (or actor) will fail to respond appropriately?*
    - *Eg. Timeout for user action*
  - *Can poor system performance result in unexpected or improper user actions?*
    - Eg. multiple selects on a processing button queue the requests inappropriately and ultimately generate an error condition

Refining a Preliminary Use Case- ACS-DCV

- Revisiting the **ACS-DCV function** as an ordered sequence of user actions

1. The homeowner logs onto the *SafeHome Products website.*
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the “surveillance” from the major function buttons.
6. **The homeowner selects “pick a camera.”**
7. **The system displays the floor plan of the house.**
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the “view” button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.



Refining a Preliminary Use Case- ACS-DCV

- *Can the actor take some other action at this point?*
  - *Yes* –the actor may choose to view thumbnail snapshots of all cameras simultaneously
    - Secondary scenario might be "View thumbnail snapshots for all cameras."
- *Is it possible that the actor will encounter some error condition at this point?*
  - Floor plan with camera icons may have never been configured
    - Selecting "pick a camera" results in an error condition
      - Secondary Scenario - "No floor plan configured for this house."
- *Is it possible that the actor will encounter some other behavior at this point?*
  - Alarm condition would result in the system displaying a special alarm notification (type, location, system action) and providing the actor with a number of options relevant to the nature of the alarm
    - occur at any time for virtually all interactions
      - a separate use case— **Alarm condition encountered —would** be developed and referenced from other use cases as required

Writing a Formal Use Case

Use Case Template for Surveillance

Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)

Iteration: 2, last modification: January 14 by V. Raman.

Primary actor: Homeowner.

Goal in context: To view output of camera placed throughout the house from any remote location via the Internet.

Preconditions: System must be fully configured; appropriate user ID and passwords must be obtained.

Trigger: The homeowner decides to take a look inside the house while away.

Scenario:

1. The homeowner logs onto the *SafeHome Products* website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the "surveillance" from the major function buttons.
6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the "view" button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Exceptions:

1. ID or passwords are incorrect or not recognized—see use case **Validate ID and passwords.**
2. Surveillance function not configured for this system—system displays appropriate error message; see use case **Configure surveillance function.**
3. Homeowner selects "View thumbnail snapshots for all camera"—see use case **View thumbnail snapshots for all cameras.**
4. A floor plan is not available or has not been configured—display appropriate error message and see use case **Configure floor plan.**
5. An alarm condition is encountered—see use case **alarm condition encountered.**

Priority: Moderate priority, to be implemented after basic functions.

When available: Third increment.

Frequency of use: Infrequent.

Channel to actor: Via PC-based browser and Internet connection.

Secondary actors: System administrator, cameras.

Channels to secondary actors:

1. System administrator: PC-based system.
2. Cameras: wireless connectivity.

Open issues:

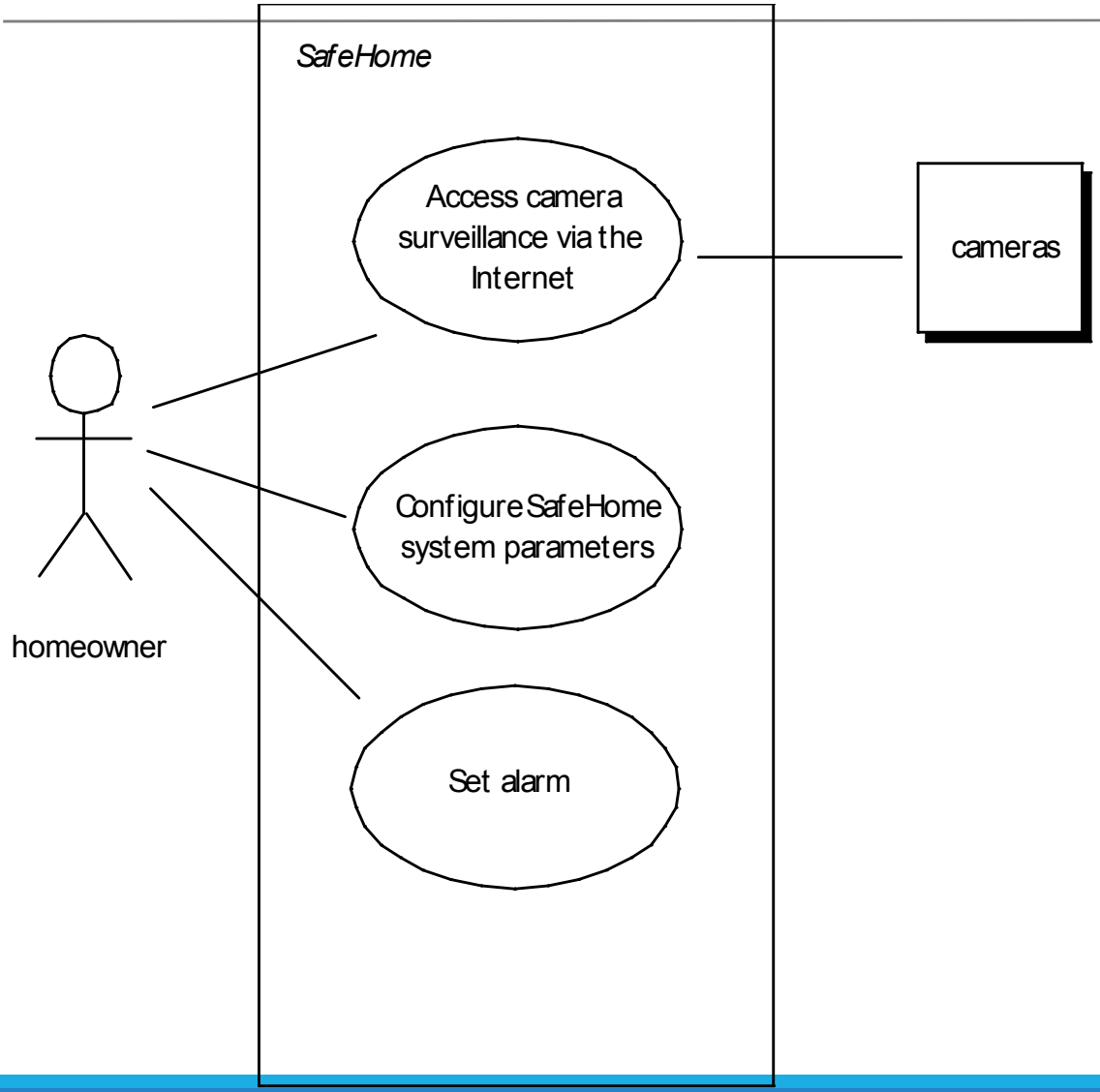
1. What mechanisms protect unauthorized use of this capability by employees of *SafeHome Products*?
2. Is security sufficient? Hacking into this feature would represent a major invasion of privacy.
3. Will system response via the Internet be acceptable given the bandwidth required for camera views?
4. Will we develop a capability to provide video at a higher frames-per-second rate when high-bandwidth connections are available?

- Desired when a use case involves a critical activity or describes a complex set of steps with a significant number of exceptions

Writing a Formal Use Case

- ***goal*** *in context identifies the overall scope of the use case*
- ***precondition*** *describes what is known to be true before the use case is initiated.*
- ***trigger*** *identifies the event or condition that "gets the use case started"*
- ***scenario*** *lists the specific actions that are required by the actor and* the appropriate system responses
- ***Exceptions*** *identify the situations uncovered* as the preliminary use case is refined
- Additional headings may or may not be included and are reasonably self-explanatory.

Use-Case Diagram



79

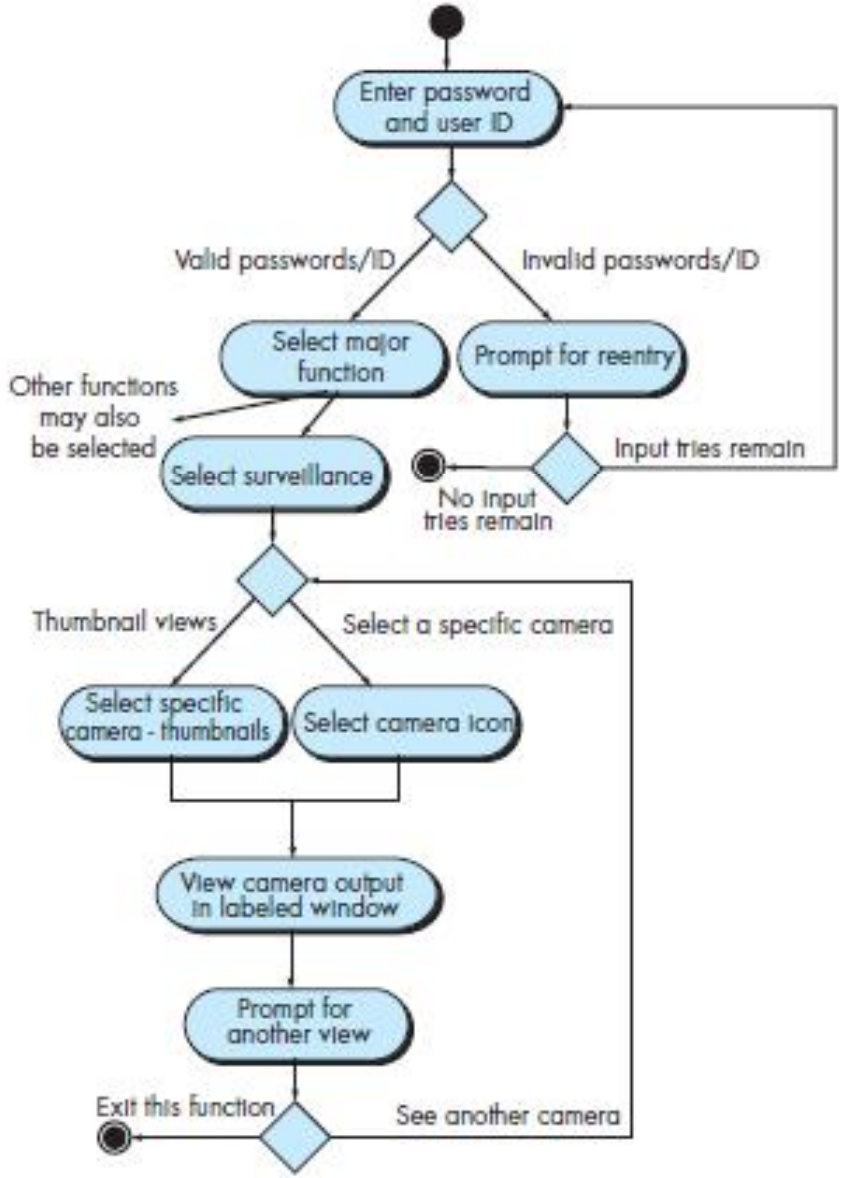
UML Models - Supplement Use Case

- impart information in a clear and concise manner compare to text-based model
  - UML activity diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario
- UML *swimlane diagram*
  - *useful variation of the activity diagram*
  - *allows* you to represent the flow of activities described by the use case
    - at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle

Activity Diagram Notation

- Rounded rectangles to imply a specific system function,
- Arrows to represent flow through the system
- Decision diamonds to depict a branching decision
  - each arrow emanating from the diamond is labeled
- Solid horizontal lines to indicate that parallel activities are occurring

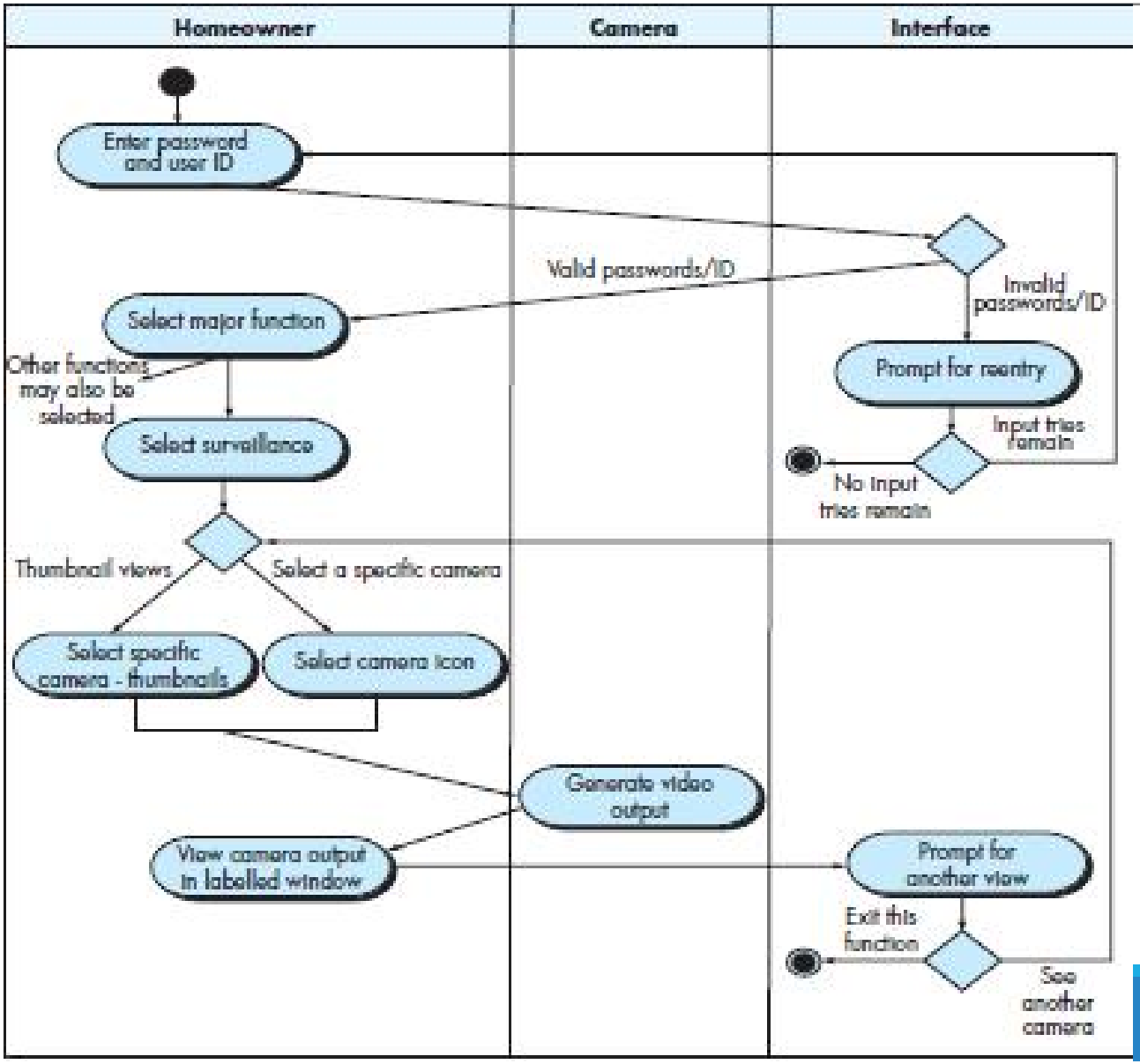
Activity diagram - ACS-DCV use case



Swimlane Diagrams

- Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.
- activity diagram is rearranged so that activities associated with a particular analysis class fall inside the swimlane for that class
- Use cases, along with the activity and swimlane diagrams, are procedurally oriented
  - represent the manner in which various actors invoke specific functions (or other procedural steps) to meet the requirements of the system

Swimlane Diagrams- ACS-DCV use case



- Three analysis classes—**Homeowner, Camera, and Interface**

Class-Based Modeling

- Class-based modeling represents:
  - objects that the system will manipulate
  - operations (also called methods or services) that will be applied to the objects to effect the manipulation
  - relationships (some hierarchical) between the objects
  - collaborations that occur between the classes that are defined.
- The elements of a class-based model include classes and objects, attributes, operations, *Class-responsibility-collaborator (CRC)* models, collaboration diagrams and packages.

85

Requirements Modeling for Web and Mobile Apps

WebApp and mobile developer

- Agile development and analysis is time consuming
- If WebApp and mobile developer understood the requirements of the problem or product then skip requirements modeling otherwise requirements modeling should be performed
- **How Much Analysis Is Enough?**
  - **The degree to which requirements modeling for** WebApps are emphasized depends on the following factors:
    - Size and complexity of WebApp increment.
    - Number of stakeholders
    - Size of the WebApp team.
    - Degree to which members of the WebApp team have worked together before
    - Degree to which the organization's success is directly dependent on the success of the WebApp.

When Do We Perform Analysis?

- In some Web/Mobile App situations, analysis and design merge. However, an explicit analysis activity occurs when …
  - the Web or Mobile App to be built is large and/or complex
  - the number of stakeholders is large
  - the number of developers is large
  - the development team members have not worked together before
  - the success of the app will have a strong bearing on the success of the business

Requirements Modeling Input

- **An agile version of the generic software process can be** applied when WebApps are engineered.
- The process incorporates a communication activity that identifies stakeholders and user categories, the business context, defined informational and applicative goals, general WebApp requirements, and usage scenarios—information that becomes input to requirements modeling.
- This information is represented in the form of natural language descriptions, rough outlines, sketches, and other informal representations.

Requirements Modeling Output

- Requirements analysis provides a disciplined mechanism for representing and evaluating WebApp content and function.
- While the specific models depend largely upon the nature of the WebApp, there are five main classes of models:
  - **Content** model—identifies the full spectrum of content to be provided by the WebApp. Content includes text, graphics and images, video, and audio data.
  - **Interaction** model—describes the manner in which users interact with the WebApp**.**
  - **Functional** model—defines the operations that will be applied to WebApp content and describes other processing functions that are independent of content but necessary to the end user.
  - **Navigation** model—defines the overall navigation strategy for the WebApp.
  - **Configuration** model—describes the environment and infrastructure in which the WebApp resides.

Content Model – overview

- **Content objects are extracted from use-cases**
  - examine the scenario description for direct and indirect references to content
- Attributes of each content object are identified
- The relationships among content objects and/or the hierarchy of content maintained by a WebApp
  - Relationships—entity-relationship diagram or UML
  - Hierarchy—data tree or UML

Content Model for WebApps

- Contains structural elements that provide an important view of content requirements for a WebApp
- These structural elements encompass content objects and all analysis classes—user-visible entities that are created or manipulated as a user interacts with the WebApp.
- Content can be developed prior to the implementation of the WebApp, while the WebApp is being built, or long after the WebApp is operational.
- It is incorporated via navigational reference into the overall WebApp structure.

Content Model for WebApps

- A content object might be
  - a textual description of a product
  - an article describing a news event
  - an action photograph taken at a sporting event
  - a user's response on a discussion forum
  - an animated representation of a corporate logo
  - a short video of a speech
  - an audio overlay for a collection of presentation slides
- The content objects might be stored as separate files, embedded directly into Web pages, or obtained dynamically from a database - a content object is any item of cohesive information that is to be presented to an end user.
- and indirect references to content. Content objects can be determined directly from use cases by examining the scenario description for direct

Content Model for WebApps

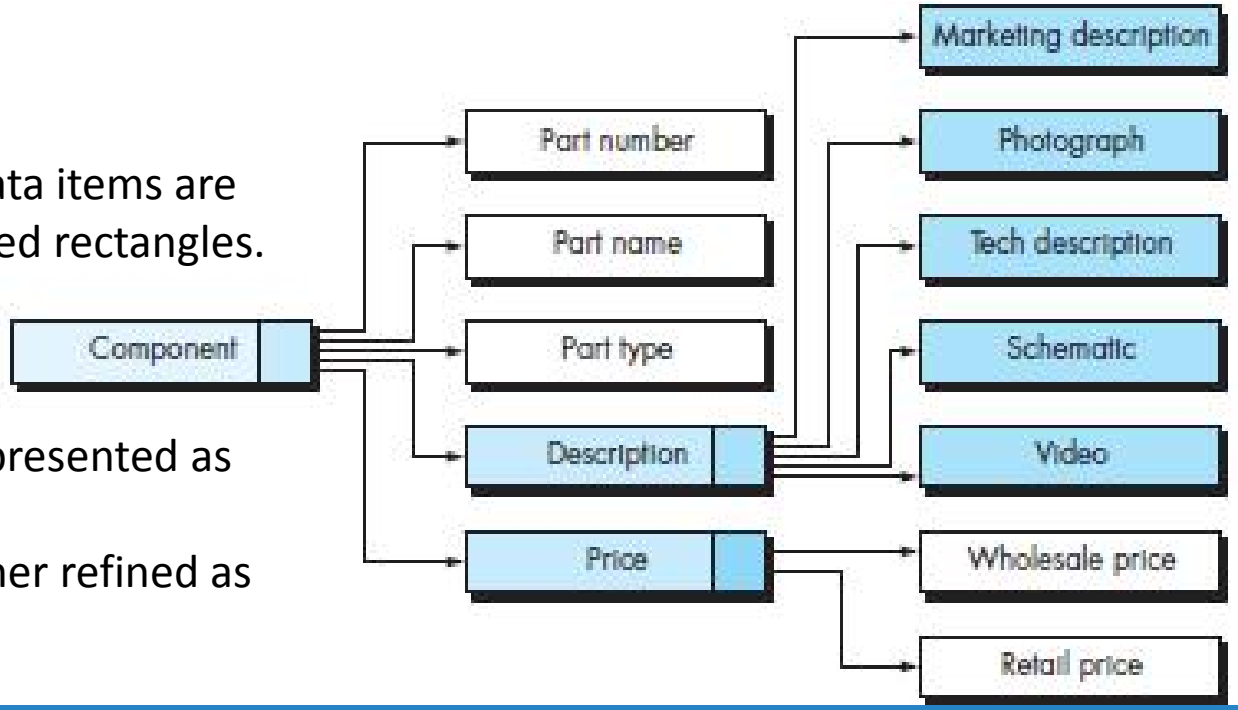
- A data tree can be created for any content that is composed of multiple content objects and data items
- The data tree is developed in an effort
  - to define hierarchical relationships among content objects
  - to provide a means for reviewing content so that omissions and inconsistencies are uncovered before design commences
- The data tree serves as the basis for content design.

Content Model for WebApps

- WebApp that supports *SafeHome is established at* ***www.safehomeassured.com***
- Use case - *Purchasing Select SafeHome Components*
- *Data tree created for a* ***www.safehomeassured*****.com** component

Simple or composite data items are represented as unshaded rectangles.

Content objects are represented as shaded rectangles
Objects would be further refined as the data tree expands



Interaction Model for Web and Mobile Apps

- A“conversation” between an end user and application functionality, content, and behavior can be described using an interaction model that can be composed of one or more of the following elements:

  (1) use cases, (2) sequence diagrams, (3) state diagrams and/or (4) user interface prototypes

- A set of use cases is sufficient to describe the interaction at an analysis level
- when the sequence of interaction is complex and involves multiple analysis classes or many tasks, it is sometimes worthwhile to depict it using a more rigorous diagrammatic form
- Because WebApp construction tools are plentiful, relatively inexpensive, and functionally powerful, it is best to create the interface prototype using such tools.

Functional Model for WebApps

- Many WebApps deliver a broad array of computational and manipulative functions that can be associated directly with content and that are often a major goal of user-WebApp interaction
- Functional requirements must be analyzed, and when necessary, modeled
- The functional model addresses two processing elements of the WebApp, each representing a different level of procedural abstraction:
  (1) user-observable functionality that is delivered by the WebApp to end users
  (2) the operations contained within analysis classes that implement behaviors associated with the class

Functional Model for WebApps

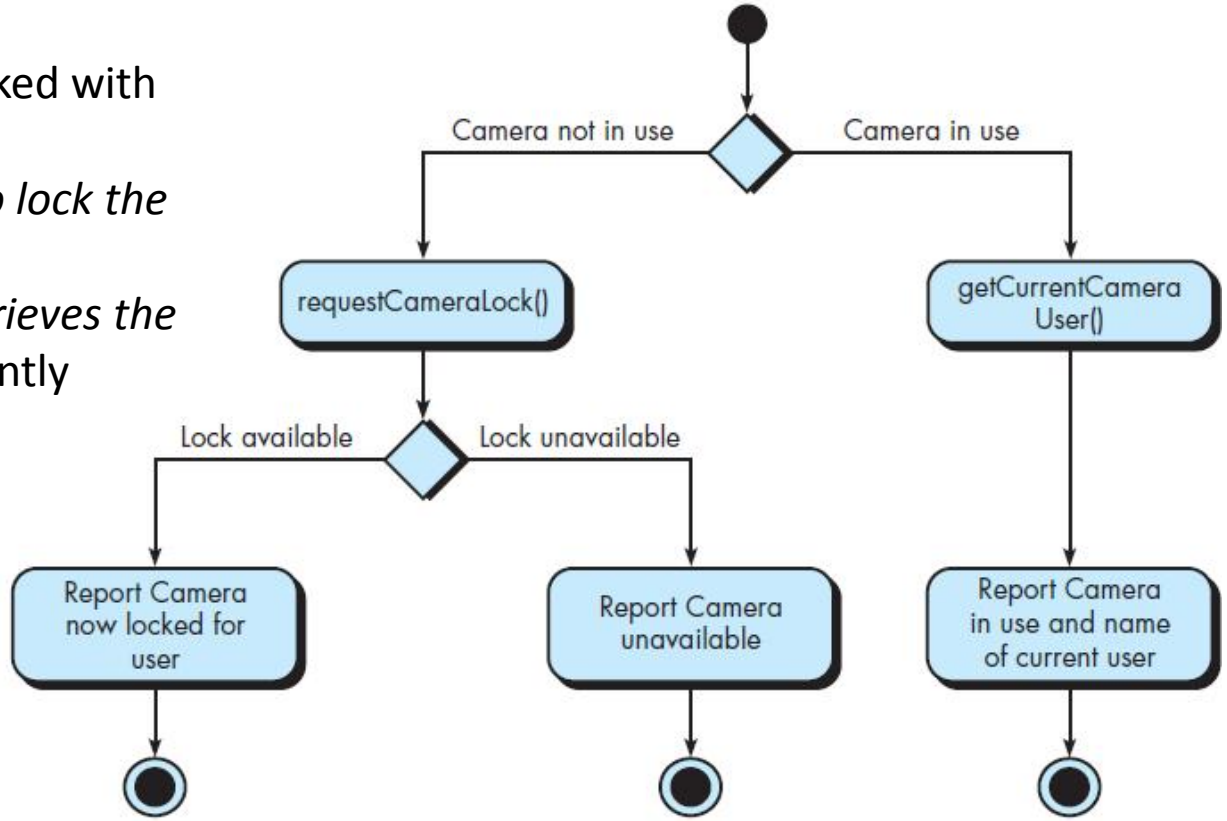
- User-observable functionality encompasses any processing functions that are initiated directly by the user
  - may actually be implemented using operations within analysis classes
  - but from the point of view of the end user, the function is the visible outcome
- At a lower level of procedural abstraction, the requirements model describes the processing to be performed by analysis class operations
- These operations manipulate class attributes and are involved as classes collaborate with one another to accomplish some required behavior.
- Regardless of the level of procedural abstraction, the UML activity diagram can be used to represent processing details

Functional Model for WebApps

- use case *Control cameras* for www.safehomeassured.com
  - activity diagram for the *takeControlOfCamera() operation* that is part of the Camera analysis class used within the *Control cameras*

two additional operations are invoked with the procedural flow:

- *requestCameraLock() - tries to lock the camera for this* user,
- *getCurrentCameraUser() - retrieves the name of the user who is* currently controlling the camera



Configuration Model - overview

- Server-side
  - Server hardware and operating system environment must be specified
  - Interoperability considerations on the server-side must be considered
  - Appropriate interfaces, communication protocols and related collaborative information must be specified
- Client-side
  - Browser configuration issues must be identified
  - Testing requirements should be defined

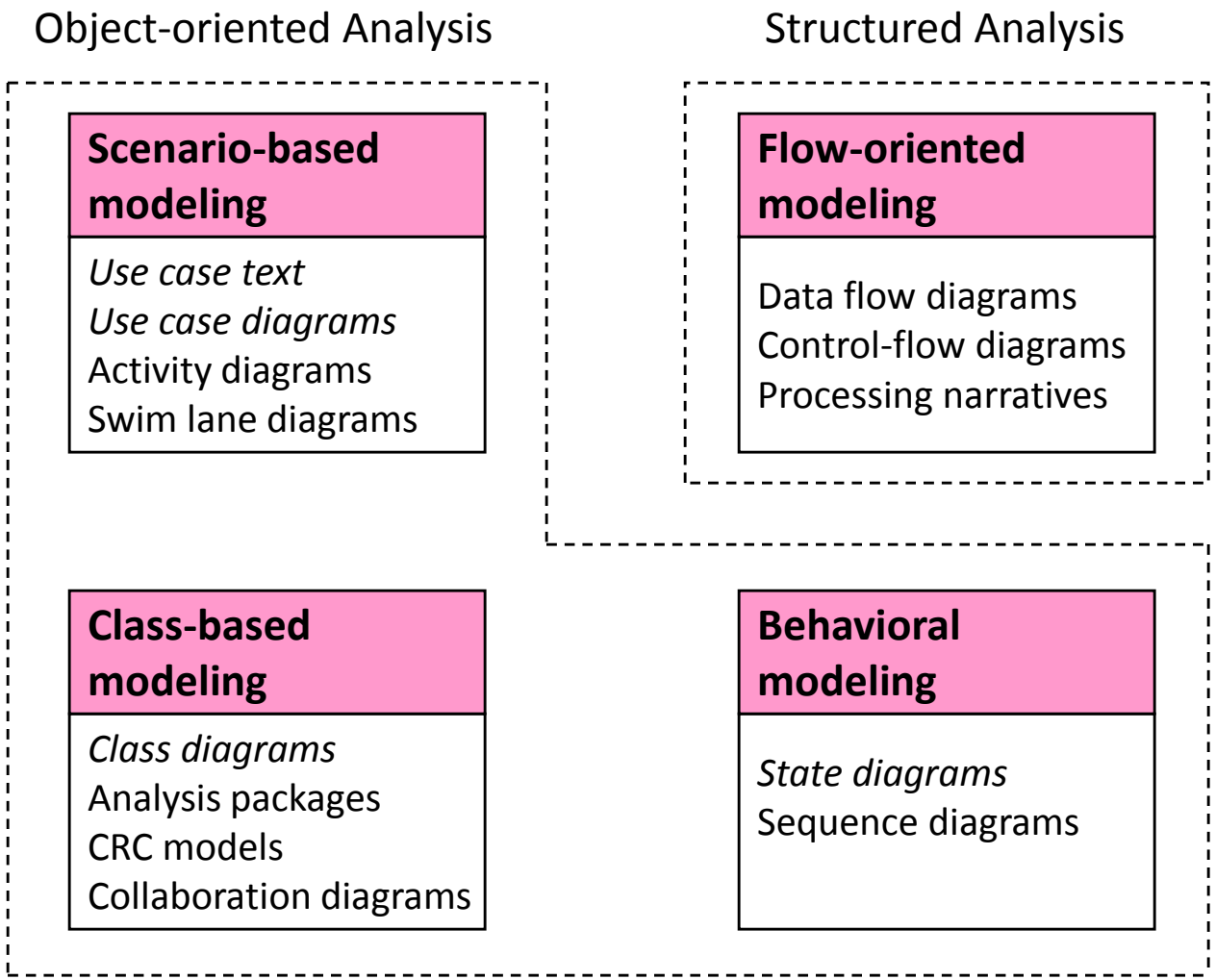
Configuration Models for WebApps

- In some cases, the configuration model is nothing more than a list of server-side and client-side attributes
- However, for more complex WebApps, a variety of configuration complexities (e.g., distributing load among multiple servers, caching architectures, remote databases, multiple servers serving various objects on the same Web page) may have an impact on analysis and design
- The UML deployment diagram can be used in situations in which complex configuration architectures must be considered
- Specify interoperability with existing product databases and monitoring applications

Navigation Modeling

- It considers how each user category will navigate from one WebApp element to another
- Questions asked and answered for overall navigation requirements analysis :
  - Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority for presentation?
  - Should certain elements be emphasized to force users to navigate in their direction?
  - How should navigation errors be handled?
  - Should navigation to related groups of elements be given priority over navigation to a specific element?
  - Should navigation be accomplished via links, via search-based access, or by some other means?
  - Should certain elements be presented to users based on the context of previous navigation actions?
  - Should a navigation log be maintained for users?
  - Should a full navigation map or menu be available at every point in a user's interaction?
  - Should navigation design be driven by the most commonly expected user behaviors or by the perceived importance of the defined WebApp elements?
  - Can a user "store" his previous navigation through the WebApp to expedite future usage?
  - For which user category should optimal navigation be designed?
  - How should links external to the WebApp be handled? Overlaying the existing browser window? As a new browser window? As a separate frame?

Summary: Elements of the Analysis Model



103