UNIT II : Software Analysis

CHAPTER 8 UNDERSTANDING REQUIREMENTS

CHAPTER 9 REQUIREMENTS MODELING: SCENARIO-BASED METHODS

CHAPTER 8 UNDERSTANDING REQUIREMENTS

8.1 Requirements Engineering

8.2 Establishing the Groundwork

- 8.2.1 Identifying Stakeholders
- 8.2.2 Recognizing Multiple Viewpoints
- 8.2.3 Working toward Collaboration
- 8.2.4 Asking the First Questions
- 8.2.5 Non-functional Requirements
- 8.2.6 Traceability

UNIT II : Software Analysis

CHAPTER 8 UNDERSTANDING REQUIREMENTS

CHAPTER 9 REQUIREMENTS MODELING: SCENARIO-BASED METHODS

CHAPTER 8 UNDERSTANDING REQUIREMENTS

8.4 Developing Use Cases

8.5 Building the Analysis Model

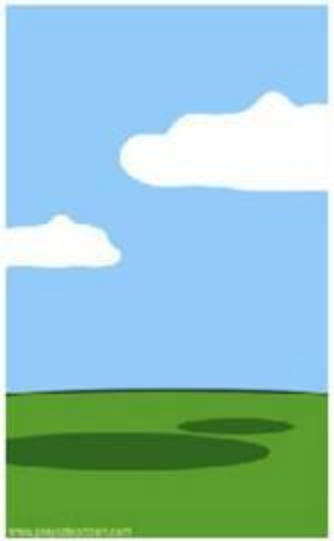
8.5.1 Elements of the Analysis Model

8.6 Negotiating Requirements

8.8 Validating Requirements

What the customer really needed

How the customer explained it



How the project was documented

How the programmer wrote it

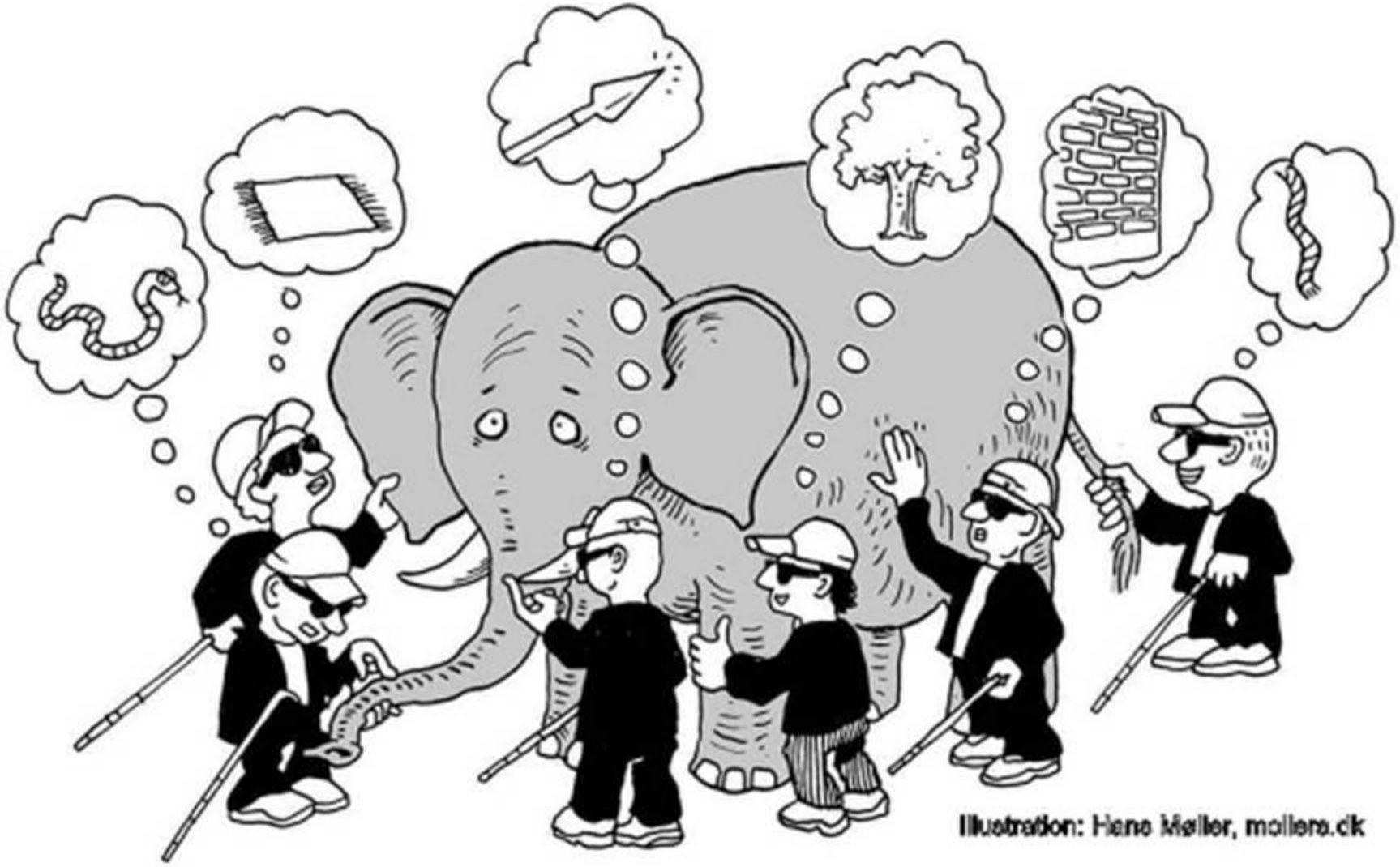
When it was delivered

How the customer was billed

What the digg effect can do to your site

The disaster recover plan

Software Engineering Professionals



8.1 Requirements Engineering

- Many software developers want to **jump right in building** software before they have a clear understanding.
- Improper understanding can lead to a **failed software project.**
- Broad spectrum of tasks and techniques that lead to an understanding of requirements is called ***requirements engineering.***
- Begins with **communication and continues into modeling**.

8.1 Requirements Engineering

Seven tasks

i. **Inception**

- **Business Manager** wants a new system
- **Business need,** new market trend or a service identified
- 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

8.1 Requirements Engineering Seven tasks

ii. **Elicitation**—elicit requirements from all stakeholders: scope, understanding and volatility (change)

- **Share** their requirements honestly: Organized activity
- **Scope:** should be properly defined by developer
- how the **system fits into the needs** of the business and
- how the system is to be used on a **day-today basis.**
- **Poor** understanding by all
- **Change** over time

8.1 Requirements Engineering Seven tasks

iii. **Elaboration**—create an analysis model that identifies **data, function and behavioral requirements. User scenario.**

- The information obtained from the customer during inception and **elicitation is expanded and refined during** elaboration.
- Focuses on developing
  - Refined requirements model: **identifies various aspects of software function, behavior, and information.**
  - **Creation and refinement of user scenarios :** end user will interact with the system.
  - Each user scenario is **parsed to extract analysis classes—business** domain entities that are visible to the end user. The attributes of each analysis class are defined, and the services that are required by each class are identified. The **relationships and collaboration between classes** are identified, and a variety of supplementary diagrams are produced.

iv. Negotiation

- Customers **ask for more than** can be achieved, given limited business resources.
- Conflicting requirements, arguing that their version is essential for our special needs.
- Agree on a deliverable system that is **realistic for developers** and customers.
- **Iterative approach** assesses their cost and risk, and addresses internal conflicts, requirements are **eliminated, combined, and/or modified** so that each party achieves some measure of satisfaction.

8.1 Requirements Engineering Seven tasks

v. **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
- Standard Template

- Written document, combining **natural language descriptions** and graphical models.
- However, **usage scenarios** may be all that are required for smaller products or systems that reside within **well-understood technical environments.**

8.1 Requirements Engineering Seven tasks

vi. **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.
- work products conform to the standards established for the process, the project, and the product.

➢ **Technical review method, team includes all stake holders**

8.1 Requirements Engineering Seven tasks

vi. Validation—example

- **Consider 2 seemingly innocuous requirements:**
  1. The software should be user friendly.
  2. The probability of a successful unauthorized database intrusion should be less than 0.0001.

➢ First: Too vague for developers to test or assess. What exactly does **"user friendly" mean**? To validate it, it must be quantified or qualified in some manner.

➢ Second: **Has quantitative element ("less than 0.0001"),** but intrusion testing will be difficult and time consuming. Is this level of security even warranted for the application? Can other complementary requirements associated with security (e.g., password protection, specialized handshaking) replace the quantitative requirement noted?

8.1 Requirements Engineering Seven tasks

vii. **Requirements management:** requirement change management

- Requirements for **computer-based systems change**, and the desire to change requirements persists throughout the life of the system.
- It is a set of activities that help the project **team identify, control, and track requirements and changes** to requirements at any time as the project proceeds.
- Many of these activities are identical to the **software configuration management (SCM)** techniques.

FINACLE ISLAMIC BANKING SOLUTION TO POWER UAE'S EMIRATES ISLAMIC BANK

- Emirates Islamic Bank (EI), a leading Sharia-compliant banking institution in the Middle East, **selected the Infosys (NYSE: INFY) Finacle Islamic Banking solution** to **power its operations and to support its vision of making Dubai the center of global Islamic Banking 2015.**
- **Developed C/C++, Java; released in 1999.**
- Leverage a world-class platform for serving customers with **greater efficiency, manage its operations more effectively, and meet the required reporting and compliance obligations.**

FINACLE ISLAMIC BANKING SOLUTION TO POWER UAE'S EMIRATES ISLAMIC BANK (CONT'D)

Highlights:

- Cost-effective, customer-centric and Sharia-compliant banking platform with proven scalability and a **lean and open architecture** for easy implementation. **Lean Thinking** (Best and advanced process)

- Wide range of products and services that will be compliant with the stringent **Sharia business rules and accounting rules.**

- Based on a modern **Service-Oriented Architecture (SOA) platform with Straight Through Processing (STP) capabilities.** SOA across platform compatibility, STP automated online payment

- Ensure **transparency of accounting and profit-sharing, along with rigorous operational risk control.**

8.2 Establishing the Groundwork

- **Steps required** to establish the groundwork for an understanding of software requirements—to get the project started in a way that will keep it moving forward toward a successful solution

- **A) Identify stakeholders (benefit directly or indirectly)**
  - **List grows: Business operations managers, product managers, marketing people, customers, end users, consultants, product engineers, support engineer, software engineer.**
  - "who else do you think I should talk to?" is asked every stakeholder.

8.2 Establishing the Groundwork

- **B) Recognize multiple points of view** and category these in-consistent and conflicting requirements.

  - **Business managers:** feature set that can be built within budget and that will be ready to meet defined market windows.
  - **End users:** features that are familiar to them and that are easy to learn and use.
  - **Software engineers:** functions that are invisible to nontechnical stakeholders but that enable an infrastructure that supports more marketable functions and features.
  - **Support engineers:** maintainability of the software.

8.2 Establishing the Groundwork

- **C) Work toward collaboration** within stakeholder and with SE Practitioners. **Identify areas of commonality and conflicts**. (priority points 10 for requirements: voting system)

  - Stakeholders collaborate by providing their view of requirements.

8.2 Establishing the Groundwork

- **D) First set of questions to "break the ice" and identify who are the stakeholders. Context free 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?

- **Next set of questions to gain a better understanding of problem and allow customers to voice their perceptions**
  - How would you characterize "good" output that would be generated by a successful solution?
  - What problems will this solution address?
  - Can you show or describe me the business environment in which solution will be used?
  - Will special performance issues or constraints affect the way the solution is approached?

8.2 Establishing the Groundwork

(D) Set of Questions (Cont'd)

- **Final set of questions focuses on the effectiveness of the communication activity itself. Meta questions**
  - 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?

8.2 Establishing the Groundwork

- **E) 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:

- **First:** to create a matrix using each NFR as a column heading and the system SE guidelines (best practice) a row labels: sandbox, coding styles, framework, language etc. **Decide guidelines** are conflicting, independent, complimentary etc.

- **Second:** Team **prioritizes each nonfunctional** requirement by creating a homogeneous set of nonfunctional requirements using a set of decision rules **that establish which guidelines to implement and which to reject.**

8.2 Establishing the Groundwork

- **F) Traceabilty–** *Traceability is a software engineering term that refers to* ***documented links between*** software engineering work products (e.g., requirements and test cases).

- A *traceability matrix allows a* ***requirements engineer to represent the relationship*** **between requirements** and other software engineering work products.

- Rows of the **traceability matrix are labeled using requirement names and columns can be labeled with the name of a software engineering work product** (e.g., a design element or a test case).

8.2 Establishing the Groundwork

- **F)Traceabilty– (Cont'd)**
- Traceability matrices can support
  - variety of engineering development activities.
  - provide continuity for developers as a project moves from one project phase to another, regardless of the process model being used.
  - ensure the engineering work products have taken all requirements into account.

- As the number of requirements and the number of work products grows, it becomes **increasingly difficult to keep the traceability** matrix up to date.
- Nonetheless, it is **important to create some means for tracking** the impact and evolution of the product requirements

8.4 Developing Use Cases

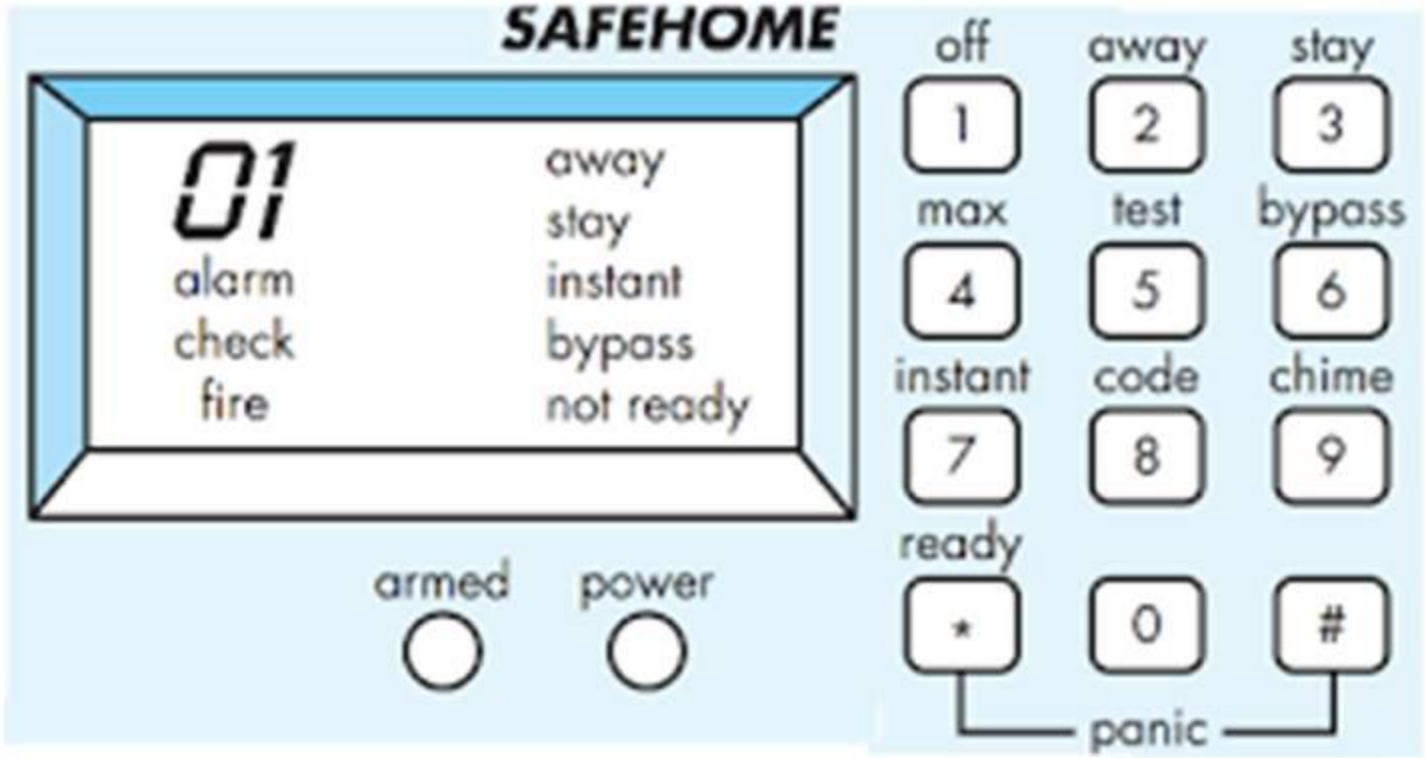
- Use case **tells a stylized story** about how an end user interacts.
- Story can be a **narrative text, an outline of tasks or interactions, a template-based description, or a diagrammatic representation.**
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s) (to support the primary actors)?
  - 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?

Use-Cases for SafeHome

- **Four actors are defined:** homeowner (a user), setup manager (likely the same person as homeowner but playing a different role), sensors (device attached to the system), and the monitoring and response subsystem (the central station that monitors the home security function).

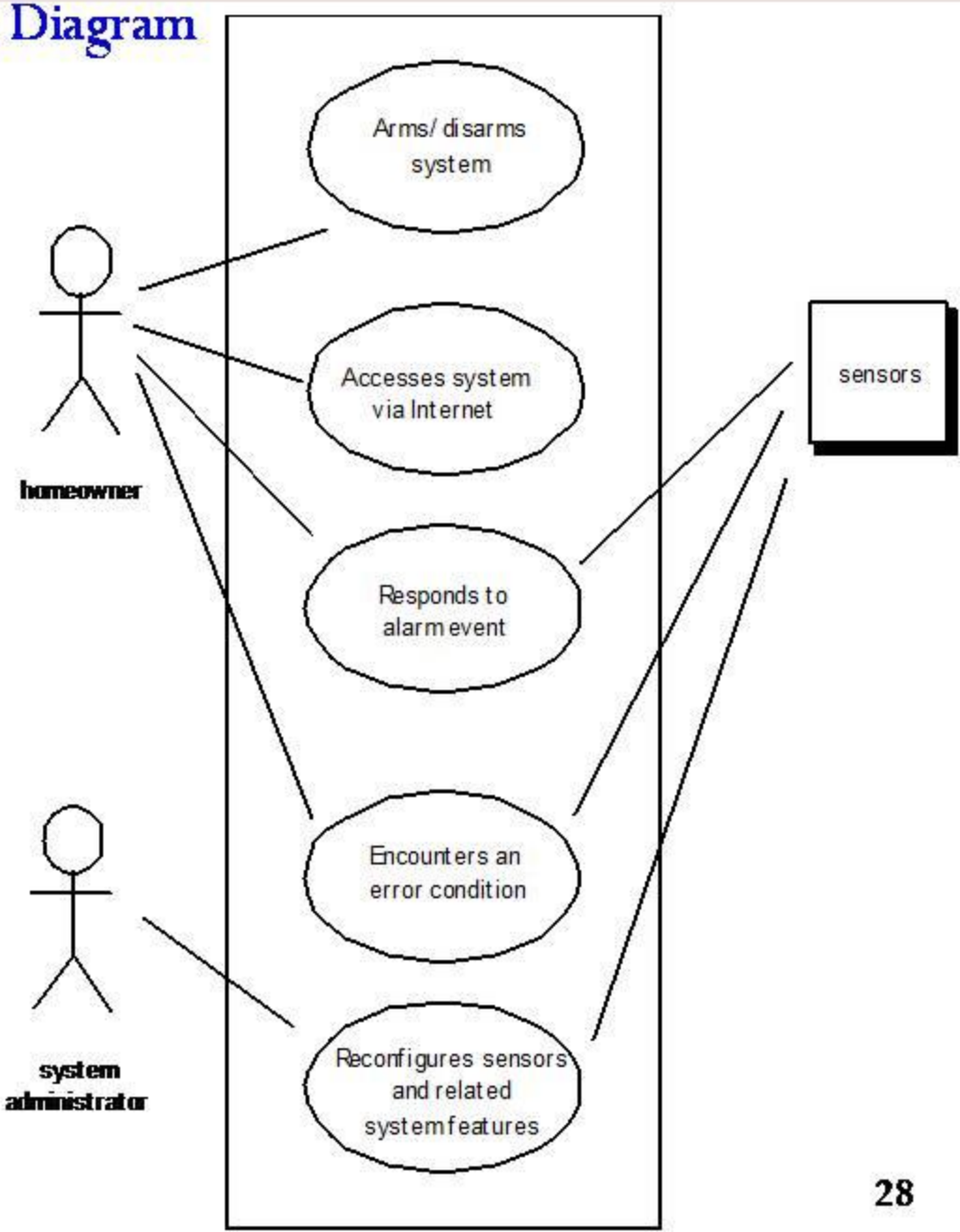
- **Consider homeowner actor (brief story)**
  - Enters a password to allow all other interactions
  - Inquires about the status of a security zone
  - Inquires about the status of a sensor
  - Press the panic button in an emergency
  - Activates/deactivate the security system

8.4 Developing Use Cases



8.4 Developing Use Cases

Use-Cases for SafeHome: Use-Case Diagram



Use-Cases for SafeHome

Use case (presenting a high-level story) for system activation follows:

- 1. The homeowner observes the *SafeHome* control panel to determine if the system is ready for input. If the system is not ready, a not ready message is displayed, and the homeowner must physically close windows or doors so that not ready message disappears. (not ready means sensor is open when door window is open)

- 2. The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password. If it is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further actions.

Use-Cases for SafeHome

Use case (presenting a high-level story) for system activation follows:

- 3. The homeowner selects and keys in stay or away to activate the system. Stay activates only perimeter sensors (inside motion detecting sensor are deactivated). Away activates all sensors.

- 4. When activation occurs, a read alarm light can be observed by the homeowner.

Use-Cases for SafeHome

- **Uses cases are further elaborated to provide more details.**

- **Precondition:** system has been programmed for a password and to recognize various sensors

- **Trigger:** the homeowner decided to set the system to turn on the alarm function.

- **Scenario:** 1. homeowner observes control panel, 2. enter password, 3. select stay or away, 4. observes red alarm light to indicate that the house has been armed.

Use-Cases for SafeHome

- **Uses cases are further elaborated to provide more details.**
- **Exceptions:** 1. control panel is not ready (check door open) 2. password is incorrect 3. correct password is not recognized.
- **Priority:** Essential, must be implemented.
- **When available:** first increment.

- **Frequency of use:** Many times per day.
- **Channel to actor:** via control panel interface.
- **Secondary actor:** support technician, sensors.
- **Channels to secondary actors:** phone line for technician, hardwired and radio frequency interface for sensors.

Use-Cases for SafeHome

- **Uses cases are further elaborated to provide more details.**

- **Open issue:**
- 1. should there be a way to activate the system without the use of a password?
- 2. should the control panel display additional messages?

- 3. How much time does the homeowner has to enter the password from the time the first key is pressed?
- 4. Is there a way to deactivate t0 system before it actually activates?

8.5 Building The Requirement Models

- As requirements evolve, certain elements of model will become stable, providing a solid foundation for the design.

- However, other elements of the model may be more volatile, indicating that stakeholders do not yet fully understand requirements.

- **Analysis model and the methods intention**:
  - description of the required informational,
  - functional, and
  - behavioral domains

- Different modes of representation: an approach that has a higher probability of uncovering omissions, inconsistencies, and ambiguity.

8.5 Building The Requirement Models

8.5.1 Elements of the Analysis Model

- Elements of the analysis model that are common to all models.

A. Scenario-based elements

- Functional—processing narratives for software functions
- Use-case—descriptions of the interaction between an "actor" and the system

B. Class-based elements

- Each usage scenario implies a set of objects (actors) that are categorized into classes. State diagram

8.5 Building The Requirement Models

8.5.1 Elements of the Analysis Model

- Elements of the analysis model that are common to all models.

C. Behavioral elements

- State diagram is one method by depicting its states and the events that cause the system to change state.

D. Flow-oriented element

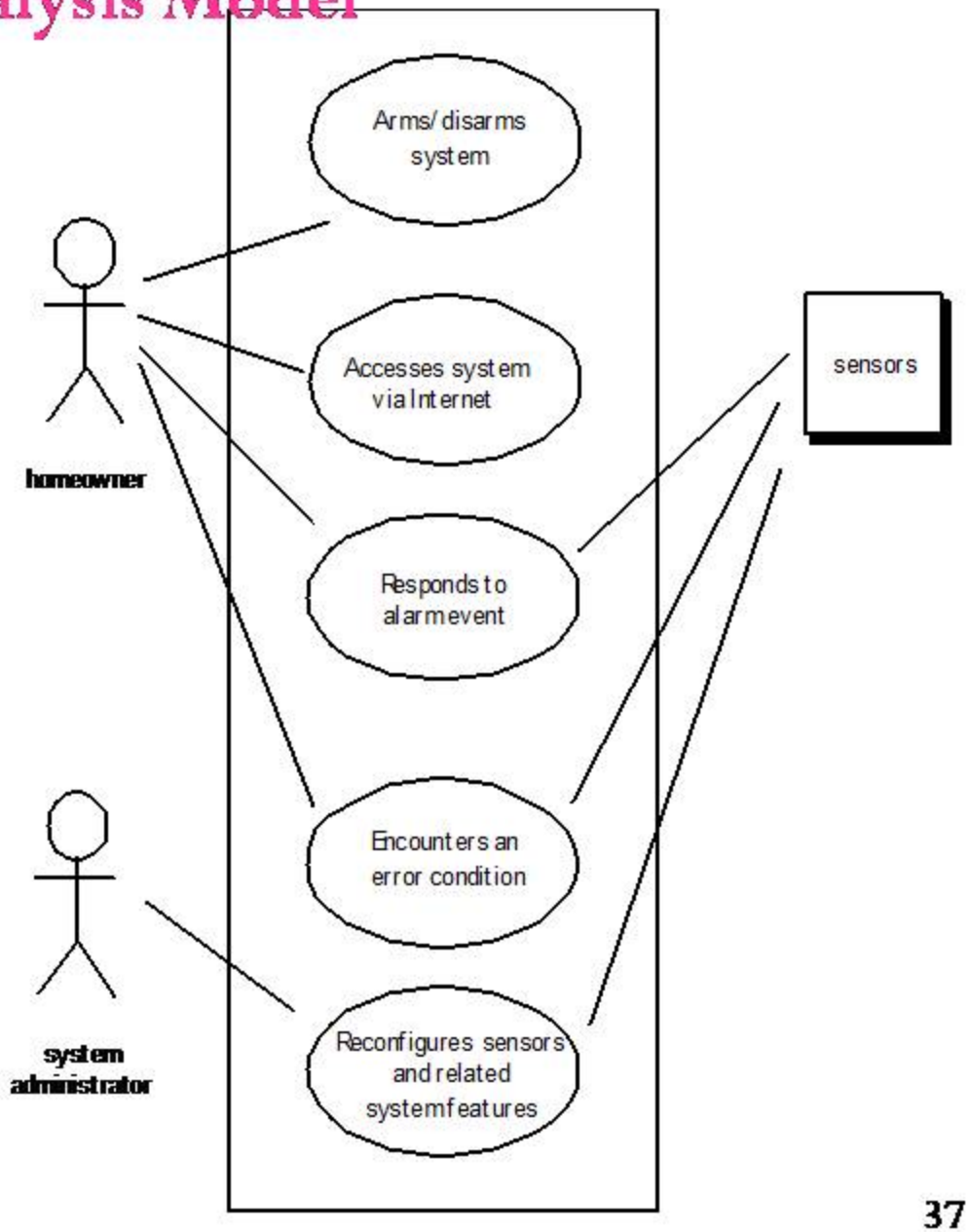
- Data flow diagram, input, transform it and produce output.

8.5 Building The Requirement Models

8.5.1 Elements of the Analysis Model

A. Scenario-based elements

Use-Case Diagram



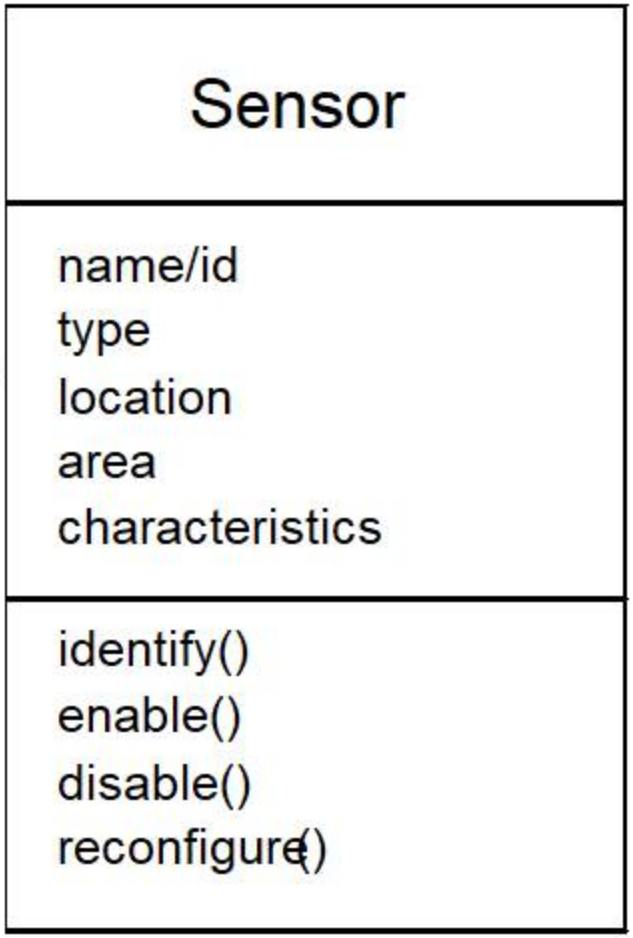
8.5 Building The Requirement Models

8.5.1 Elements of the Analysis Model

B. Class-based elements

Class Diagram

From the *SafeHome* system ...

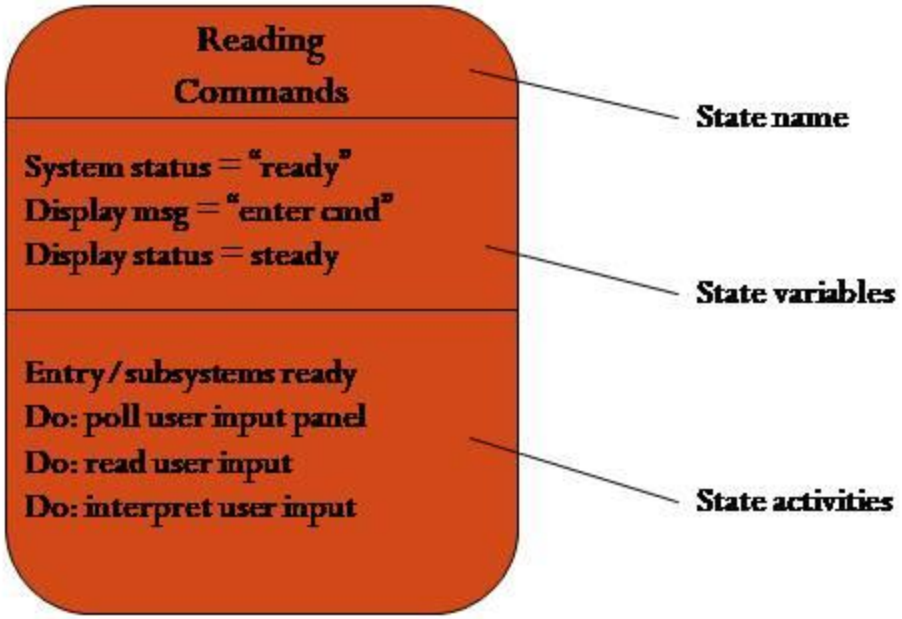


8.5 Building The Requirement Models

8.5.1 Elements of the Analysis Model

C. Behavioral elements

UML State Diagram



Software embedded within control panel that is responsible for reading user input.

8.6 Negotiating Requirements

- Even after inception, elicitation and elaborations, customer requirements are **still not sufficient** in details to proceed to next activity.

- Stakeholders need to balance **functionality, performance and other product or system characteristics against cost and time-to-market.**

- Thus, **negotiation is to develop a project plan** that meets stakeholder needs while at the same time reflecting the real-world **constraints such as time, people and budget.**

8.6 Negotiating Requirements

➢ 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" for all concerned including the software team.

8.6 Negotiating Requirements

- Fricker et al. replaced traditional handoff of requirements specifications to software teams with a bidirectional communication process called ***handshaking.***
- *Software team proposes* **solutions to requirements, describes their impact, and communicates their intentions** to customer representatives (CRs).
- CRs **review the proposed solutions, focusing on missing features and seeking clarification of novel requirements.**
- Handshaking allows detailed requirements to be delegated to software teams.
- Handshaking tends to improve identification, analysis, and selection of variants and promotes win-win negotiation.

8.8 Validating Requirements

- 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?

8.8 Validating Requirements

- 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?