Software Engineering

Chapter 5: Understanding Requirements

A process is a roadmap for building high-quality software.

Contents

- What is Requirements?
- Requirements Engineering (RE) Process
- Steps of RE
- Software Requirements Specification (SRS)

Requirements

- ***Definition***: A **requirement** is a ***condition or capability*** that a software system must have to ***satisfy user needs***, ***solve a problem***, or ***achieve a business objective***.
- In **Software Engineering**, a **requirement** refers to a **statement that defines what a *software system should do*** or ***how it should perform***.
- It represents the ***needs and expectations*** of the ***customer, user, or stakeholder*** that the software must fulfil.
- Suppose you are developing a **university management system**:
  - The system shall allow students to register for courses online.
  - The system shall respond to user actions within 2 seconds.

Why Requirements are **Important**?

- **Foundation** of software development.
- Guide to design, coding, testing, and maintenance.
- Reduce **misunderstandings** between clients and developers.
- Ensure that the final product **meets user needs.**

Without clear requirements, the software may fail to solve the intended problem, leading to wasted time, money, and effort.

Types of Requirements

- Functional Requirements
  - Define **what the system should do** — specific functions or features.
  - The system shall generate student grade reports.
- Non-Functional Requirements
  - Define **how the system should perform** — quality, performance, usability, etc.
  - The system should support 1000 users simultaneously.
- Business Requirements
  - Describe **why** the system is being developed and the **business goals**.
  - To automate student registration and reduce manual work.
- User Requirements
  - Expressed in **natural language** for end-users.
  - A user should be able to change their password
- System Requirements
  - Detailed technical specifications derived from user needs.
  - The system must use a MySQL database.

Requirements Engineering (RE)- Introduction

- **Understanding** the requirements of a problem is the ***most difficult tasks*** software engineer face.
- Even if customer are **explicit in their needs**, it will ***change*** throughout the project.
- Challenges:
  - Collect requirements in disorgamized manner and not verified.
  - Allow change to control us, rather than establishing mechanisms to control change.

It's your worst nightmare. A customer walks into your office, sits down, looks you straight in the eye, and says, "I know you think you understand what I said, but what you don't understand is what I said is not what I meant." - **Ralph Young**

RE - Definition

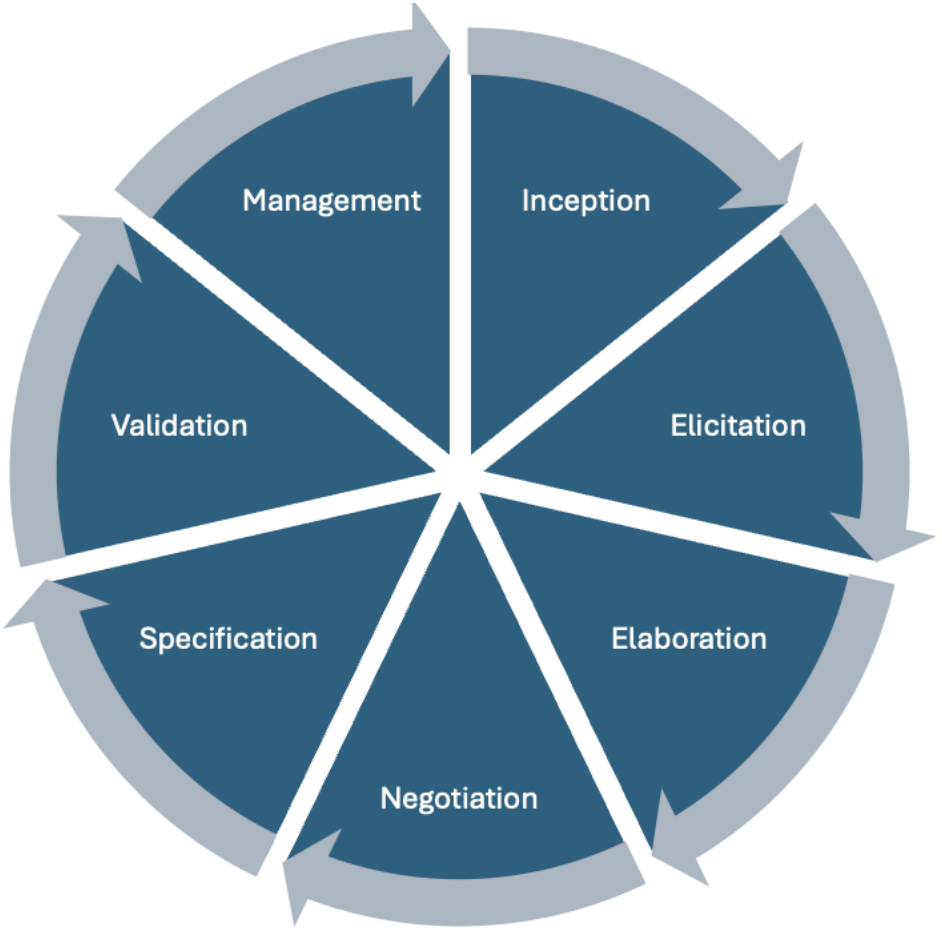
- The ***broad spectrum of tasks and techniques*** that lead to an ***understanding of requirements*** is called requirements engineering.
- In other words, before you start designing or coding, you must clearly understand **what the customer really needs** — not just what they say.
- Software engineers use several **tasks** (like interviews, meetings, analysis, documentation) and **techniques** (like use cases, prototypes, user stories, etc).
- Establishes a **solid base** for design and construction.
- Without it, the resulting software has a high probability of **not meeting customer's needs.**
- A major software engineering action that begins during the **communication activity** and continues into the modeling activity

RE: Bridge to Design & Construction

- What is the starting point?
  - At the feet of the project stakeholders (Managers, Customers, & users)
    - Business need is defined, user scenarios are described, functions and features are delineated, project constraints are identified
  - With a broader system definition
- Allows you to examine the context of the software word to be performed:
  - The **specific needs** that design and construction must address.
  - The **priorities** that guide the order in which word is to be completed.
  - The **information, functions, and behaviours** that will have a profound impact on the resultant design.

RE : 7 Tasks

- Inception
- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Management



RE : 7 Tasks

- **Inception (Ice Breaking Phase)**:
  - *Ask a set of questions:*
    - How does a software project get started?
    - Is there a single event that becomes the catalyst for a new computer-based system or product?
    - Does the need evolve over time?

    → ***No defitnitive answers for those questions***
  - In general, most projects begin when a *business need is identified* or a potential new market or service is discovered.
  - *Establish the following:*
    - 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

RE : 7 Tasks

- **Elicitation** – Elicit requirements from all stakeholders
  - It certainly seems simple enough by doing
    - Ask the customer, the users, and others
    - What the objectives for the system or product are?
    - what is to be accomplished, how the system or product fits into the needs of the business?
  - But it isn’t simple. It is very hard!
  - To establish business goals
    - Engage stakeholders and encourage them to share their goals honestly.
    - after recording goal, a prioritization mechanism should be established, and a design a potential architecture.

RE : 7 Tasks

- **Elicitation** – Three problems
  - *Problems of Scope*
    - Occur when the boundary of the system is ***ill-defined*** or the customers and users specify ***unnecessary technical detail*** that may confuse.
  - *Problems of Understanding*
    - customers and users are ***not completely sure*** of what is needed.
    - ***don't*** have a full ***understanding*** of the ***problem domain.***
    - specify requirements that ***conflict with the needs*** of other customers and users.
  - *Problems of Volatility*
    - Occur when the requirements change over time

To overcome these problems, the requirements-gathering activity should be done in an organized manner.

RE : 7 Tasks

- **Elaboration**
  - The ***information*** obtained from the customer during inception and elicitation is ***expanded and refined.***
  - Focuses on developing a ***refined requirements model*** that identifies various aspects of software ***function, behavior, and information.***
  - It is driven by the creation and refinement of ***user scenarios*** that describe how the end user will interact with the system.

RE : 7 Tasks

- **Negotiation** - agree on a deliverable system that is realistic for developers and customers.
- ***Problems***:
  - Customers and users can ask for ***more than can be achieved***, given limited business resources.
  - Different customers or users can propose ***conflicting requirements***, arguing that their version is “essential for our special needs”
- ***Negotiations***
  - Mitigate these conflicts
  - Customers, users, and other stakeholders are asked to ***rank requirements*** and then discuss conflicts in priority
  - Prioritizes requirements, accesses their cost and risk, and addresses internal conflicts → *Towards an iterative approach*

RE : 7 Tasks

- **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
  - Any combination of these
- Follow standard template for specification (Suggested by researchers)
- The formality and format of a specification varies with the size and the complexity of the software to be built:
  - **For large systems**, a written document, combining natural language description and graphical models may be the best approach
  - **For small systems**, usage scenarios may be all that required

RE : 7 Tasks

- **Validation** – a review mechanism that looks for
  - Erro in content or interpretation
  - Areas where clarification may be required
  - Missing information
  - Inconsistencies (a major problem when large producats and systems are engineered)
  - Conflicting or unrealistic requirements
  - Perform technical Review by review team (Software Engineers, customers, users and other stakeholders)

RE : 7 Tasks

- **Validation** – Requirement Representation problem
- Example:
  - R1: The software should be user friendly.
  - R2: The probability of a successful unauthorized database intrusion should be less than 0.0001

  - **Validation required:**
  - R1: Too vague. To validate it, it must be quantified or qualified in some manner.
  - R2: An 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 replace the quantitative requirement noted?

RE : 7 Tasks

- **Requirements management** – Manage the change of requirements
  - Requirements change over time thouout the life of a software or systems
  - RM 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.
  - Activities: Identification, Traceability, Prioritization, Change Management, Version Control, verification and validation

Inception: Establishing the Ground Work

- What are the steps to get the project started in a way that will keep it moving forward toward a successful solution?

- ***Step 1: Identifying Stakeholders***
- ***Step 2: Recognizing Multiple viewpoints***
- ***Step 3: Working toward collaboration***
- ***Step 4: Asking the first questions***

Step 1: Identifying Stakeholders

- **Stakeholder**: Anyone who benefits in a ***direct or indirect*** way from the system which is being developed.
  - *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 getting the successful system, or is open to different risk for the failure.

Step 2: Recognizing Multiple viewpoints

- The requirements of the system will be explored from many different point of view:
  - ***Marketing group***: functions and features that will excite the potential market.
  - ***Business managers***: a feature set that can be built within budget and that will meet defined market windows.
  - ***End users***: Features that are familiar to them and 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.

Step 3: Working toward collaboration

- Collaboration requires to avoid inconsistency and conflict.
- Discussion among stakeholders to mitigate conflict.
- In many cases, stakeholders collaborate by providing their view of requirements, but a strong "***project champion***" (***a business manager or a senior technologist***) may make the ***final decision*** about which requirements ***make the cut.***

Step 4: Asking the first questions

- ***Context free questions***: identify stakeholders & Measureable benefit:
  - 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?
- ***Problem specific questions***: Understanding the problem
  - 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?
- **Meta Questions: Validate the questions and asnwers**

Step 4: Asking the first questions

- ***Meta Questions***: Validate the questions and asnwers
  - 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?

→ *Focuses on the effectiveness of the communication activity itself*

Elicitation: Eliciting the requirements

- Requirements gathering
- Combines elements of problem solving, elaboration, negotiation, and specification.
- The goal is:
  - To identify the problem
  - Propose elements of the solution
  - Negotiate different approaches and
  - Specify a prelimiary set of solution requirements

Process of eliciation

- ***Step 1: Collaborative requirements Gathering***: follow basic ***guidelines*** and a ***facilitator*** control meetings/communcaition.
- ***Step 2: Quality function deployment (QFD)***: Qulity management technique.
- ***Step 3: Usage scenarios:*** Understand users interactions through communication.
- ***Step 4: Elicitaion work products:***

Eliciation: **QFD**

- QFD is a ***quality management technique*** that translates the needs of the customer into ***technical requirements*** for software.
- Focuses on customer satisfaction
- QFD identifies three types of requirements
  - Normal Requirements
  - Expected requirements
  - Exciting requirements (Wow factor)

Elicitation **work products**

- Work products include:
  - 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.

Developing Use Cases

- A use case is a description of how a user (called an *actor*) ***interacts with a system*** to achieve a specific goal.
- "*What will the user do with the system, and how will the system respond?*"
- Use cases help to:
  - ***Understand user needs*** — by focusing on what users want to accomplish.
  - ***Bridge communication*** between clients and developers (simple, scenario-based language).
  - ***Define system boundaries*** — what is inside and outside the system.
  - ***Form the basis*** for design, development, and testing later.

Developing use cases

- User cases are defined from an ***actor's point of view***.
- **Actors:**
  - An actor is a role that ***people or devices*** play as they interact with the software.
  - An actor is anything that communicates with the system or product and that is ***external*** to the system itself
  - Every actor has ***one or more goals*** when using the system
  - An actor and an end user are ***not*** necessarily the ***same thing***
- **Types of Actors:**
  - ***Primary Actor*****:** Interact to achieve required system function and derive the intended benefit from the system.
  - ***Secondary Actor***: Support the system so that primary actors can do their work.

Developing use cases

- Set of questions for for a use case:
  - 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 **exceptions** 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 case Template [Cockburn]

- Use case
- Primary actor
- Goal in context
- Preconditions
- Trigger
- Scenario
- Exceptions
- Priority
- When available
- Frequency of use
- Channel to actor
- Secondary Actors
- Channels to secondary actors
- Open Issues

Example: SafeHome – basic use case

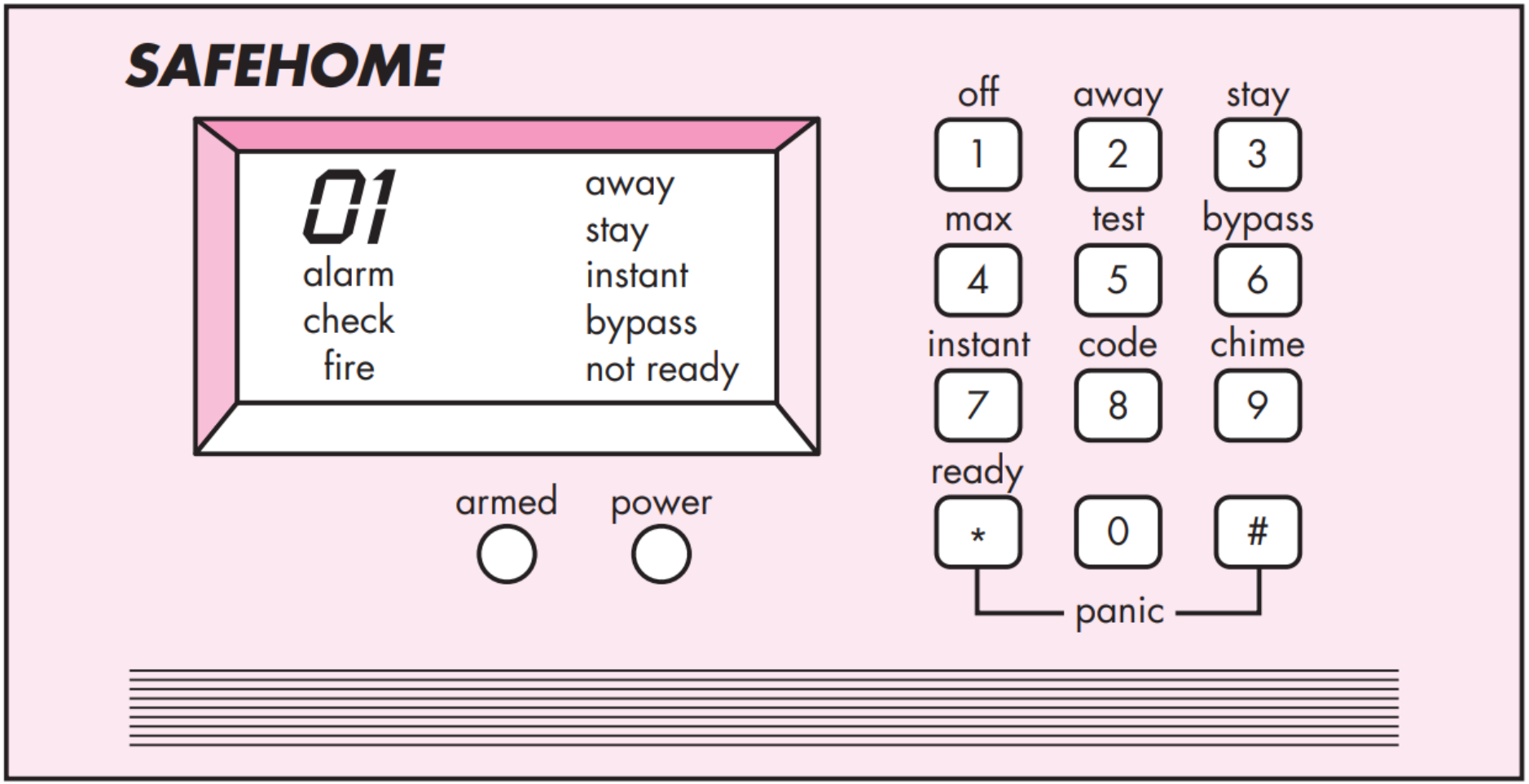
1. The homeowner observes the SafeHome control panel to determine if the system is ***ready for input***. If the system is not ready, a not ready ***message is displayed on the LCD display***, and the homeowner must physically close windows or doors so that the not ready message disappears. [A not ready 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 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 control panel



Detailed Description of Use cases

- **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, i.e., to turn on the alarm functions.
- **Scenario**:
  1. Homeowner: observes control panel
  2. Homeowner: enters password
  3. Homeowner: selects “stay” or “away”
  4. Homeowner: observes read alarm light to indicate that SafeHome has been armed

Detailed Description of Use cases (cont..)

- **Exceptions**:

  1. Control panel is not ready: homeowner checks all sensors to determine which are open; closes them.

  2. Password is incorrect (control panel beeps once): homeowner reenters correct password.

  3. Password not recognized: monitoring and response subsystem must be contacted to reprogram password.

  4. Stay is selected: control panel beeps twice and a stay light is lit; perimeter sensors are activated.

  5. Away is selected: control panel beeps three times and an away light is lit; all sensors are activated

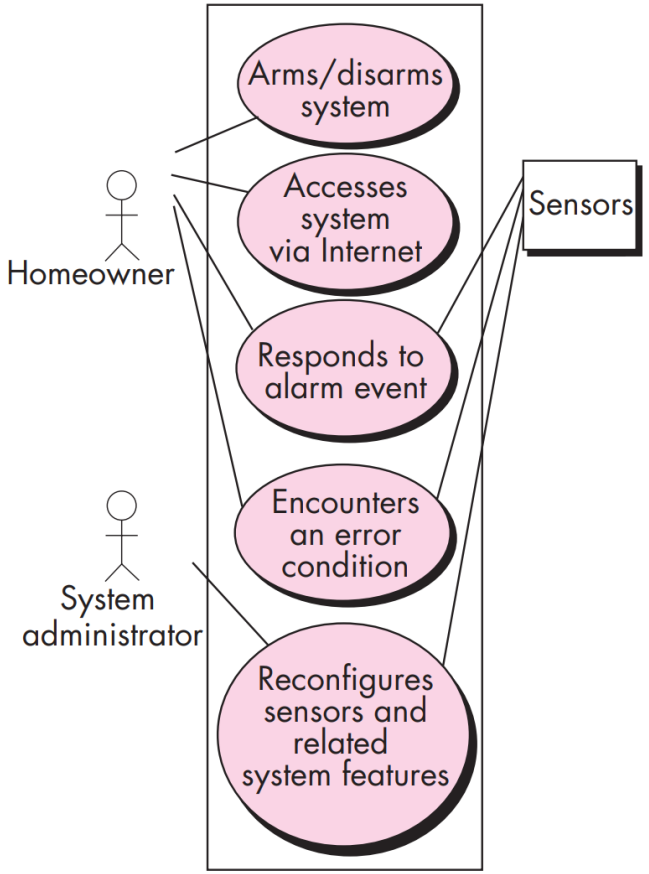
- **Priority**: Essential, must be implemented
- **When available**: First increment

Detailed Description of Use cases (cont..)

- **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

UML Use case diagram for Safehome Security function



Software Requirements Specification (SRS)

- A Software Requirements Specification (**SRS**) is a **formal document** that describes in detail:
  - What the software system should do,
  - How it should perform, and
  - All the constraints under which it must operate.
- It serves as a **contract** between the client and the development team, ensuring that everyone has a common understanding of the system requirements.
- In other words, *SRS is a detailed **blueprint** of the software before development starts.*

Lab Assignment (10%)

- SRS on University Management System.

References

- Chpater 5: Understanding the Requirements (Pressman - 7 edition)

Thank you