Chapter 5

Understanding the requirements

5.1 Requirements engineering

- Requirements engineering is the wide range of activities and methods that result in a comprehension of requirements.
- Requirements engineering is a significant software engineering action that starts during the communication activity and extends into the modelling activity from the standpoint of the software process. It needs to be modified to meet the requirements of the project, the product, the workers, and the procedure.
- Requirements engineering builds a bridge to design and construction.
- Requirements engineering offers an effective framework for comprehending the customer's desires, evaluating necessity, determining feasibility, discussing a viable solution, articulating the solution clearly, confirming the accuracy of the specifications, and overseeing the evolution of requirements into a functional system.

It involves seven specific tasks:

Inception, elicitation, elaboration, negotiation, specification, validation, and management.

1. Inception

- Just chatting informally can unexpectedly lead to a significant software engineering project. However, typically, projects start when a business requirement emerges or a potential new market or service is uncovered.
- Business stakeholders such as business managers, marketing professionals, and product managers craft a business justification for the concept. They aim to gauge the extent and potential of the market, conduct a preliminary feasibility assessment, and outline a functional overview of the project's scope.
- At this stage, it's crucial to establish a foundational understanding of the problem at hand, the individuals seeking a solution, the type of solution desired, and the efficacy of initial communication and collaboration among stakeholders and the software team.

2. Elicitation

Several challenges arise during the process of requirements elicitation.

1. Problems of scope: The system's boundary is unclear, or customers/users provide excessive technical details that might complicate rather than clarify the overall objectives of the system.

2. Problems of understanding: The customers/users are uncertain about their exact needs, lack a thorough understanding of their computing environment's capabilities and constraints, and have incomplete knowledge of the problem domain.

3. Problems of volatility: The requirements change over time.

3. Elaboration

- Elaboration is guided by crafting and enhancing use case scenarios, detailing how the end user (and other actors in the scene) will engage with the system.
- Each user scenario is dissected to extract analysis classes, which are business domain entities observable to the end user. Attributes of each analysis class are specified, and the necessary services for each class are determined. Relationships and cooperation among classes are outlined, and various additional diagrams are generated.

4. Negotiation

- To address the conflicts in requirements gathering, negotiation becomes essential. Customers, users, and other stakeholders are prompted to prioritize requirements and engage in discussions to resolve conflicts.
- Employing an iterative method that focuses on prioritizing requirements, evaluating their costs and risks, and managing internal conflicts, adjustments are made. This may involve eliminating, combining, or modifying requirements to ensure that each party attains some level of satisfaction.

5. Specification

A specification may take various forms, including a written document, a series of graphical representations, and a formal mathematical model, a collection of use case scenarios, a prototype, or a blend of these methods.

INFO

Software Requirements Specification Template

A *software requirements specification* (SRS) is a document that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified.

Karl Wiegers [Wie03] of Process Impact Inc. has developed a worthwhile template (available at **www.processimpact.com/process_assets/srs_template.doc**) that can serve as a guideline for those who must create a complete SRS. A topic outline follows:

Table of Contents
Revision History

1. Introduction
1.1 Purpose
1.2 Document Conventions
1.3 Intended Audience and Reading Suggestions
1.4 Project Scope
1.5 References

2. Overall Description
2.1 Product Perspective
2.2 Product Features
2.3 User Classes and Characteristics
2.4 Operating Environment
2.5 Design and Implementation Constraints
2.6 User Documentation
2.7 Assumptions and Dependencies

3. System Features
3.1 System Feature 1
3.2 System Feature 2 (and so on)

4. External Interface Requirements
4.1 User Interfaces
4.2 Hardware Interfaces
4.3 Software Interfaces
4.4 Communications Interfaces

5. Other Nonfunctional Requirements
5.1 Performance Requirements
5.2 Safety Requirements
5.3 Security Requirements
5.4 Software Quality Attributes

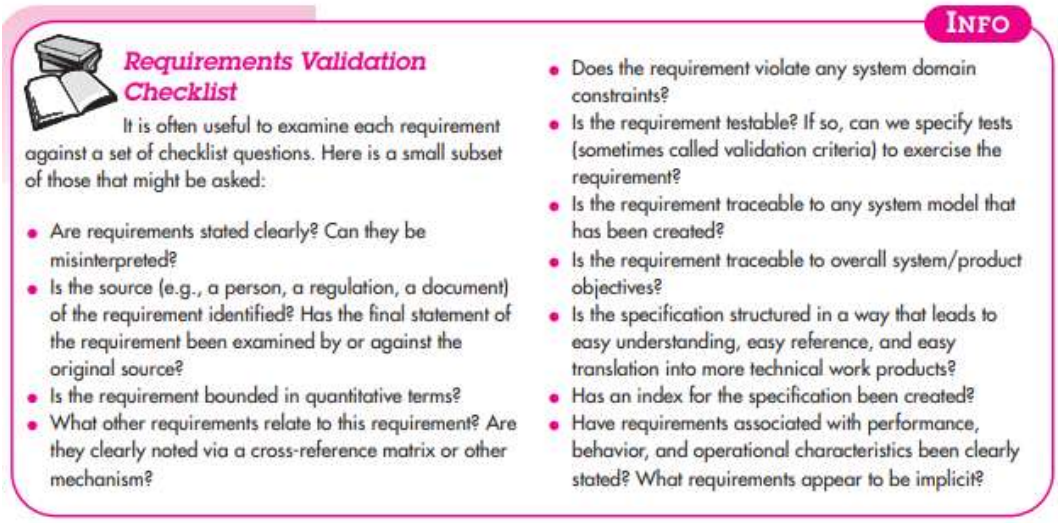
6. Other Requirements

Appendix A: Glossary
Appendix B: Analysis Models
Appendix C: Issues List

A detailed description of each SRS topic can be obtained by downloading the SRS template at the URL noted earlier in this sidebar.

6. Validation

The main method for validating requirements is through technical review. This review involves a team comprising software engineers, customers, users, and other stakeholders who scrutinize the specification for errors in content or interpretation, areas necessitating clarification, omissions, inconsistencies (especially prevalent in large-scale products or systems), conflicting requirements, or requirements that are impractical or unattainable.



INFO

Requirements Validation Checklist

It is often useful to examine each requirement against a set of checklist questions. Here is a small subset of those that might be asked:

- Are requirements stated clearly? Can they be misinterpreted?
- Is the source (e.g., a person, a regulation, a document) of the requirement identified? Has the final statement of the requirement been examined by or against the original source?
- Is the requirement bounded in quantitative terms?
- What other requirements relate to this requirement? Are they clearly noted via a cross-reference matrix or other mechanism?
- Does the requirement violate any system domain constraints?
- Is the requirement testable? If so, can we specify tests (sometimes called validation criteria) to exercise the requirement?
- Is the requirement traceable to any system model that has been created?
- Is the requirement traceable to overall system/product objectives?
- Is the specification structured in a way that leads to easy understanding, easy reference, and easy translation into more technical work products?
- Has an index for the specification been created?
- Have requirements associated with performance, behavior, and operational characteristics been clearly stated? What requirements appear to be implicit?

7. Requirements management

Requirements for computer-based systems evolve over time, and the need for modifications in requirements persists throughout the system's lifecycle. Requirements management encompasses a range of tasks aimed at enabling the project team to identify, control and requirements and any alterations to them as the project progresses.

5.2 Establishing the groundwork

Steps required to establish the groundwork for the understanding of software requirements

5.2.1 Identifying Stakeholders

A stakeholder is

- Anyone who has a direct interest in or benefits from the system that is to be developed.
- Anyone who benefits in a direct or indirect way from the system which is being developed.
- Identified stakeholders are: Business operations managers, product managers, marketing people, internal and external customers, end users, consultants, product engineers, software engineers, and support and maintenance engineers.

5.2.2 Recognizing Multiple Viewpoints

- The requirements of the system will be examined from various perspectives.
- For instance, the marketing team focuses on functions and features that will attract the potential market, making the new system easy to sell.
- Business managers are interested in a feature set that can be developed within budget and ready to meet defined market windows.
- End users prefer features that are familiar and easy to learn and use.
- Software engineers are concerned with functions that may be invisible to non-technical stakeholders but that provide an infrastructure supporting more marketable functions and features.
- Support engineers prioritize the maintainability of the software.
- All stakeholder information, including any inconsistent and conflicting requirements are organised in a manner that enables decision-makers to select a set of internally consistent requirements for the system.

5.2.3 Working toward Collaboration

- The role of a requirements engineer is to pinpoint areas of commonality (i.e., requirements that all stakeholders agree on) and areas of conflict or inconsistency (i.e., requirements desired by one stakeholder that conflict with the needs of another).
- Collaboration doesn't always mean that requirements are decided by committee. Often, stakeholders contribute their perspectives on requirements, but a decisive "project champion" (such as a business manager or senior technologist) ultimately determines which requirements are accepted.

INFO

Using "Priority Points"

One way of resolving conflicting requirements and at the same time better understanding the relative importance of all requirements is to use a "voting" scheme based on *priority points*. All stakeholders are provided with some number of priority points that can be "spent" on any number of requirements. A list of requirements is presented, and each stakeholder indicates the relative importance of each (from his or her viewpoint) by spending one or more priority points on it. Points spent cannot be reused. Once a stakeholder's priority points are exhausted, no further action on requirements can be taken by that person. Overall points spent on each requirement by all stakeholders provide an indication of the overall importance of each requirement.

5.2.4 Asking the First Questions

The first set of context-free questions focuses on the customer and other stakeholders. For example:

- 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?

The next set of questions enables you to gain a better understanding of the problem and allows 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?

The final set of questions focuses on the effectiveness of the communication

- 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?

5.3 Eliciting Requirements

Requirements elicitation (or requirements gathering) involves aspects of problem-solving, elaboration, negotiation, and specification. To promote a collaborative, team-oriented approach, stakeholders work together to identify the problem, propose solution elements, negotiate various approaches, and specify an initial set of solution requirements.

5.3.1 Collaborative Requirements Gathering

The objective is to recognize the problem, suggest components of the solution, discuss various strategies, and outline an initial set of solution requirements, all within an environment that supports achieving the objective.

Basic guidelines

- Meetings are conducted and attended by both software engineers and other 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" (can be a customer, a developer, or an outsider) controls the meeting.
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

The objective is to recognize the problem, suggest components of the solution, discuss various strategies, and outline an initial set of solution requirements, all within an environment that supports achieving the objective.

During inception, the developer and customers write "product request". A meeting location, time, and date are determined; a facilitator is appointed; and participants from the software team and other stakeholder groups are invited to join. The product request is shared with all attendees prior to the meeting.

A narrative about the home security function that is to be part of 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. It can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

Prior to the meeting, each participant is asked to review the product request and create several lists: one of objects within the environment surrounding the system, another of objects the system will produce, and a third of objects the system will use to carry out its functions. Additionally, participants should

compile a list of services (processes or functions) that interact with or manipulate these objects. Finally, they need to develop lists of constraints (such as cost, size, and business rules) and performance criteria (such as speed and accuracy).

The goal is to create an agreed-upon list of objects, services, constraints, and performance criteria for the system that will be developed.

Each mini-specification is an elaboration of an object or service. For example, the mini-spec for the SafeHome object Control Panel might be:

> The control panel is a wall-mounted unit that is approximately 9 × 5 inches in size. The control panel has wireless connectivity to sensors and a PC. User interaction occurs through a keypad containing 12 keys. A 3 × 3 inch LCD color display provides user feedback. Software provides interactive prompts, echo, and similar functions.

The mini-specs are shared with all stakeholders for discussion, where additions, deletions, and further details are made. This process may reveal new objects, services, constraints, or performance requirements that will be added to the initial lists.

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.)

5.3.2 Quality Function Deployment

QFD defines requirements in a way that maximizes customer satisfaction.

To achieve this, QFD focuses on understanding what is valuable to the customer and integrating these values throughout the engineering process. QFD identifies three types of requirements.

1. Normal requirements

The objectives and goals outlined for a product or system during meetings with the customer. If these requirements are met, the customer is satisfied. Examples of normal requirements include graphical displays, specific system functions, and defined levels of performance.

2. Expected requirements

These requirements are inherent to the product or system and may be so basic that the customer does not explicitly mention them. Their absence will lead to significant dissatisfaction. Examples of expected requirements are: ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation.

3. Exciting requirements

These features exceed the customer's expectations and are highly satisfying when included. For example, software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multi-touch screen, visual voice mail) that delight every user of the product.

QFD gathers requirements through customer interviews and observations, surveys, and analysis of historical data (such as problem reports). This information is compiled into a ***customer voice table***, which is reviewed with the customer and other stakeholders. Various diagrams, matrices, and evaluation methods are then employed to identify expected requirements and try to uncover exciting requirements.

5.3.3 Usage Scenarios

As requirements are collected, a comprehensive vision of the system's functions and features starts to take shape. However, it is challenging to advance to more technical software engineering tasks without understanding how these functions and features will be utilized by various types of end users.

To accomplish this, developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed. The scenarios, often called use cases, provide a description of how the system will be used.

5.3.4 Elicitation Work Products

The work products generated from requirements elicitation will differ based on the system's or product's size. For most systems, these work products typically 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.

5.4 Developing use cases

Use cases are defined from an actor's point of view. An actor is a role that people (users) or devices play as they interact with the software.

- The initial step in crafting a use case involves identifying the group of "actors" participating in the scenario.
- Actors encompass the diverse individuals or devices utilizing the system or product within the context of the function and behavior being depicted.
- They embody the roles individuals (or devices) assume during the system's operation. More precisely, an actor is any entity external to the system that interacts with it. Each actor possesses one or more objectives while engaging with the system.
- An Actor represents a class of external entities that play just one role in the context of the use case
- After careful review of requirements, the software for the control computer requires four different modes (roles) for interaction: programming mode, test mode, monitoring mode, and troubleshooting mode. Therefore, four actors can be defined: programmer, tester, monitor, and troubleshooter.

- Since requirements elicitation is an iterative process, not all actors are necessarily identified in the initial iteration. Primary actors can be identified in the first iteration, while secondary actors may become apparent as more is learned about the system.
- Primary actors are those directly involved in achieving the required system function and deriving its intended benefits, interacting frequently with the software.
- Secondary actors, on the other hand, support the system, enabling primary actors to carry out their tasks.
- Once actors have been identified, use cases can be developed.

Number of questions that should be answered by a use case:

1. Who is the primary actor, the secondary actor(s)?
2. What are the actor's goals?
3. What preconditions should exist before the story begins?
4. What main tasks or functions are performed by the actor?
5. What exceptions might be considered as the story is described?
6. What variations in the actor's interaction are possible?
7. What system information will the actor acquire, produce, or change?
8. Will the actor have to inform the system about changes in the external environment?
9. What information does the actor desire from the system?
10. Does the actor wish to be informed about unexpected changes?

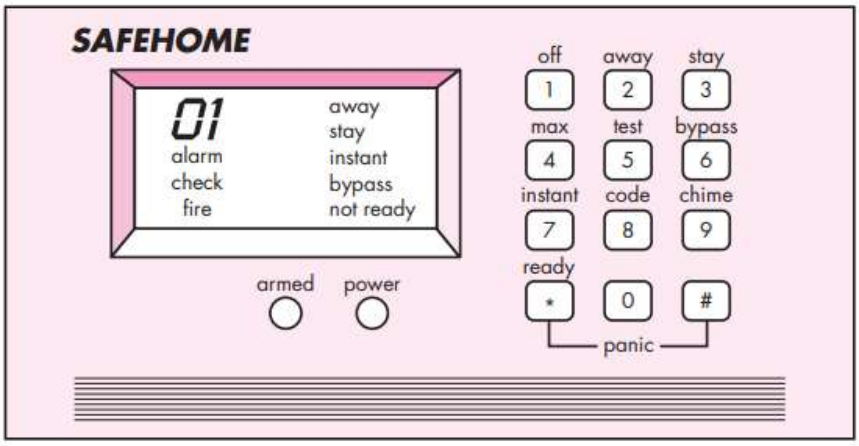
Referring back to the fundamental **SafeHome** requirements, we establish four actors: **the homeowner** (a user), **the setup manager** (potentially the same individual as the homeowner but fulfilling a distinct role), **sensors** (devices integrated into the system), and **the monitoring and response subsystem** (the central station overseeing the SafeHome home security operations). For this illustration, we focus solely on the **homeowner actor**. The **homeowner actor** engages with the home security function through various means, utilizing either the alarm control panel or a PC:

- Enters a password to allow all other interactions.
- Inquires about the status of a security zone.
- Inquires about the status of a sensor.
- Presses the panic button in an emergency.
- Activates/deactivates the security system.

Considering the situation in which the homeowner uses the control panel, the basic use case for system activation follows:

1. The homeowner observes the *SafeHome* control panel (Figure 5.1) 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.]

FIGURE 5.1
SafeHome control panel



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 5.1) 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.

Use cases are often written informally. However, use the template shown here to ensure that you've addressed all key issues.

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.

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

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?

SAFEHOME

Developing a High-Level Use-Case Diagram

The scene: A meeting room, continuing the 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 spent a fair amount of time talking about *SafeHome* home security functionality. During the break I sketched a use case diagram to summarize the important scenarios that are part of this function. Take a look.

(All attendees look at Figure 5.2.)

Jamie: I'm just beginning to learn UML notation.[14] So the home security function is represented by the big box with the ovals inside it? And the ovals represent use cases that we've written in text?

Facilitator: Yep. And the stick figures represent actors—the people or things that interact with the system as described by the use case . . . oh, I use the labeled square to represent an actor that's not a person . . . in this case, sensors.

Doug: Is that legal in UML?

Facilitator: Legality isn't the issue. The point is to communicate information. I view the use of a humanlike stick figure for representing a device to be misleading. So I've adapted things a bit. I don't think it creates a problem.

Vinod: Okay, so we have use-case narratives for each of the ovals. Do we need to develop the more detailed template-based narratives I've read about?

Facilitator: Probably, but that can wait until we've considered other *SafeHome* functions.

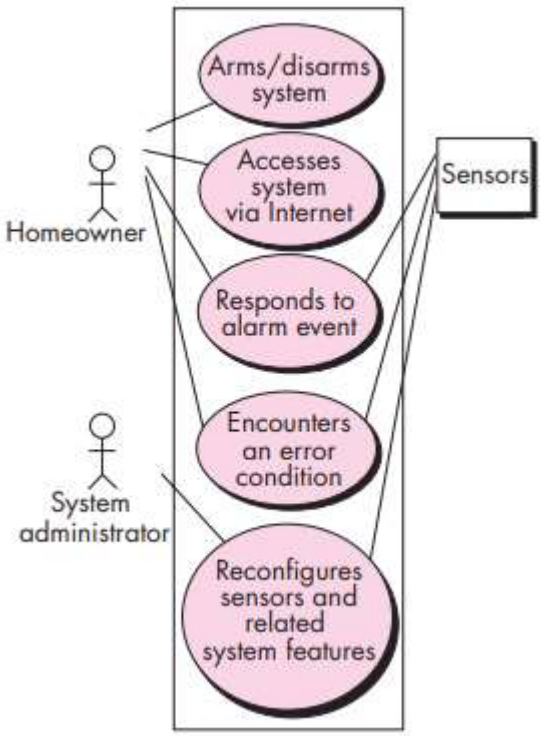
Marketing person: Wait, I've been looking at this diagram and all of a sudden I realize we missed something.

Facilitator: Oh really. Tell me what we've missed.

(The meeting continues.)

FIGURE 5.2

UML use case diagram for *SafeHome* home security function



SOFTWARE TOOLS

Use-Case Development

Objective: Assist in the development of use cases by providing automated templates and mechanisms for assessing clarity and consistency.

Mechanics: Tool mechanics vary. In general, use-case tools provide fill-in-the-blank templates for creating effective use cases. Most use-case functionality is embedded into a set of broader requirements engineering functions.

Representative Tools:[15]
The vast majority of UML-based analysis modeling tools provide both text and graphical support for use-case development and modeling.
Objects by Design
(**www.objectsbydesign.com/tools/umltools_byCompany.html**) provides comprehensive links to tools of this type.