Week 3

CS2212
Introduction to Software Engineering

Requirements Engineering

Some images and content from Software Engineering: A Practitioner's Approach 9th Edition By Roger Pressman and Bruce Maxim

Principles that Guide Practice

You often hear people say that software development knowledge has a 3-year half-life: half of what you need to know today will be obsolete within 3 years. In the domain of technology-related knowledge, that's probably about right. But there is another kind of software development knowledge—a kind that I think of as "software engineering principles"—that does not have a three-year half-life. These software engineering principles are likely to serve a professional programmer throughout his or her career.

- Steve McConnell

Principles that Guide Practice

- Chapter 6 contains a large list of such principles.
- We will cover them as they relate to the topic of the week rather than all at once (still suggested you read this chapter however).
- This week we will take a look at the Communications Principles (Section 6.2.1)

Communication Principles

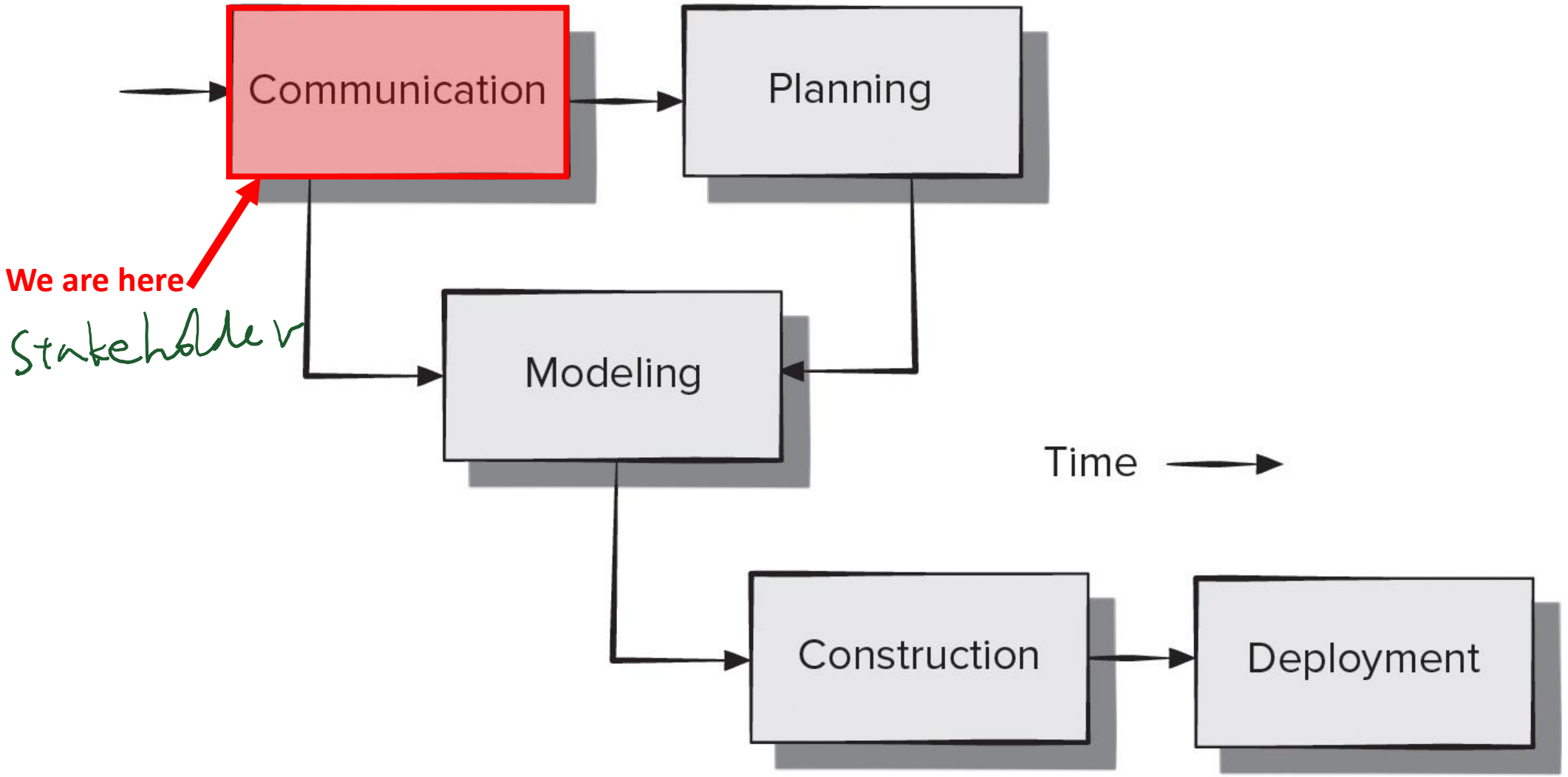


Figure 6.1 from your textbook

Communication Principles

- **Principle #1.** ***Listen.*** Try to focus on the speaker's words, not formulating your response to those words.
- **Principle # 2.** ***Prepare before you communicate.*** Understand a problem before meeting with others. ~~...~~ giving a general idea.
- **Principle # 3.** ***Someone should facilitate the activity.*** Every communication meeting should have a leader to keep the conversation moving in a productive direction.
- **Principle #4.** ***Face-to-face communication is best.*** Visual representations of information can be helpful.
- **Principle # 5.** ***Take notes and document decisions.*** Someone should serve as a "recorder" and write down all important points and decisions.

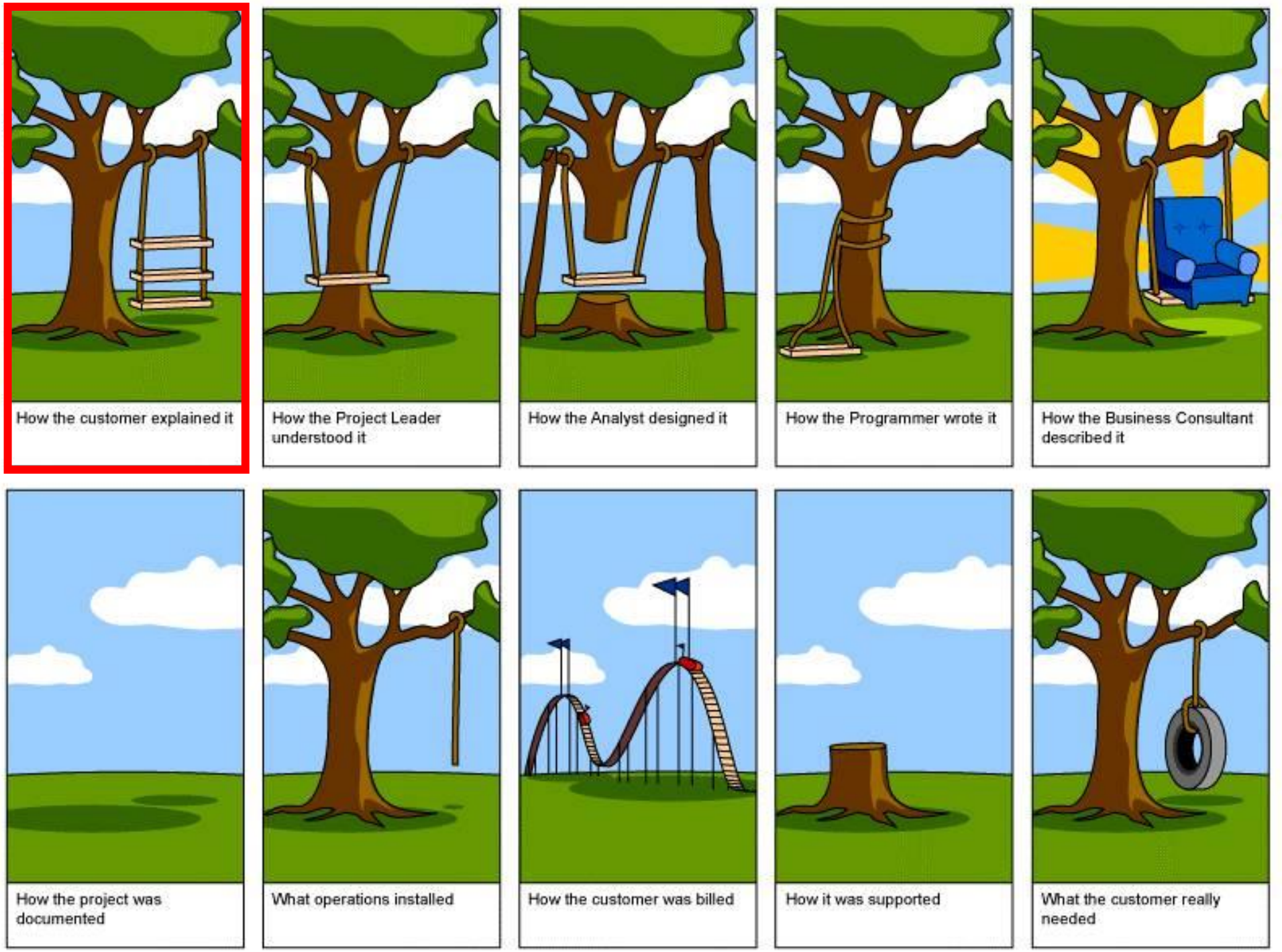
Communication Principles

- **Principle # 6.** ***Strive for collaboration.*** Consensus occurs when collective team knowledge is combined.
- **Principle # 7.** ***Stay focused, modularize your discussion.*** The more people involved in communication the more likely discussion will bounce between topics.
- **Principle # 8.** ***If something is unclear, draw a picture.***
- **Principle # 9.** ***Move On*** (a) Once you agree to something, **move on**; (b) If you can't agree to something, **move on**; (c) If a feature or function is unclear and cannot be clarified at the moment, **move on**.
- **Principle # 10.** ***Negotiation is not a contest or a game.*** It works best when both parties win.

Understanding Requirements

"The hardest single part of building a software system is deciding what to build. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

- Fred Brooks


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

How the customer explained it

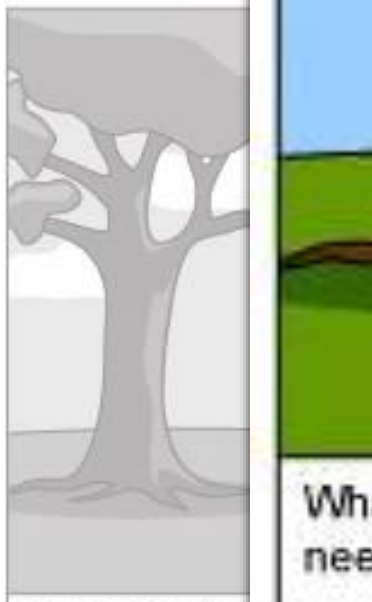
How the Project Lead understood it

What the customer really needed

mmer wrote it

How the Business Consultant described it

How the project was documented



What operations insta

orted

Requirements

- **Requirements** are statements that describe the users' or stakeholders' needs and desires with respect to the software in question.
- **Each requirement:**
  - Is a short and concise piece of information.
  - Says something about the software system.
  - Is agreed upon by stakeholders.
  - Helps solve the customers' problems.

Requirements: Functional vs. Non-Functional

Functional Requirements

- Defines a function of a system or its component.
- May involve calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is supposed to accomplish.
- Generally, expressed in the form ***"system must do <requirement>"***.
- **Example:**
  - The system sends a confirmation email when a new user account is created.
  - The search feature must allow the user to find specific classrooms on a map.

Non-Functional Requirements (NFR)

- A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours.
- Define how a system is supposed to be.
- An overall property of the system as a whole or of a particular aspect and not a specific function.
- Generally, expressed in the form of ***"system shall be <requirement>"***.
- **Example:**
  - The system must be secure.
  - The system should be able to handle 20 million users without performance deterioration.

Functional or Non-Functional?

Try to classify each of the following as a functional or non-functional requirement.

FR

The system sends an approval request after the user enters personal information.

NFR

The website pages should load in 3 seconds with the total number of simultaneous users <5 thousand.

NFR

Emails should be sent with a latency of no greater than 12 hours.

FR

The system must support multiple users and allow them to login to the application via a login screen.

FR

The Sales system should allow users to record customers sales.

NFR

The software should be portable. So moving from one OS to other OS does not create any problem.

FR

The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.

NFR

The system should be secure and not vulnerable to cyber attacks.

FR

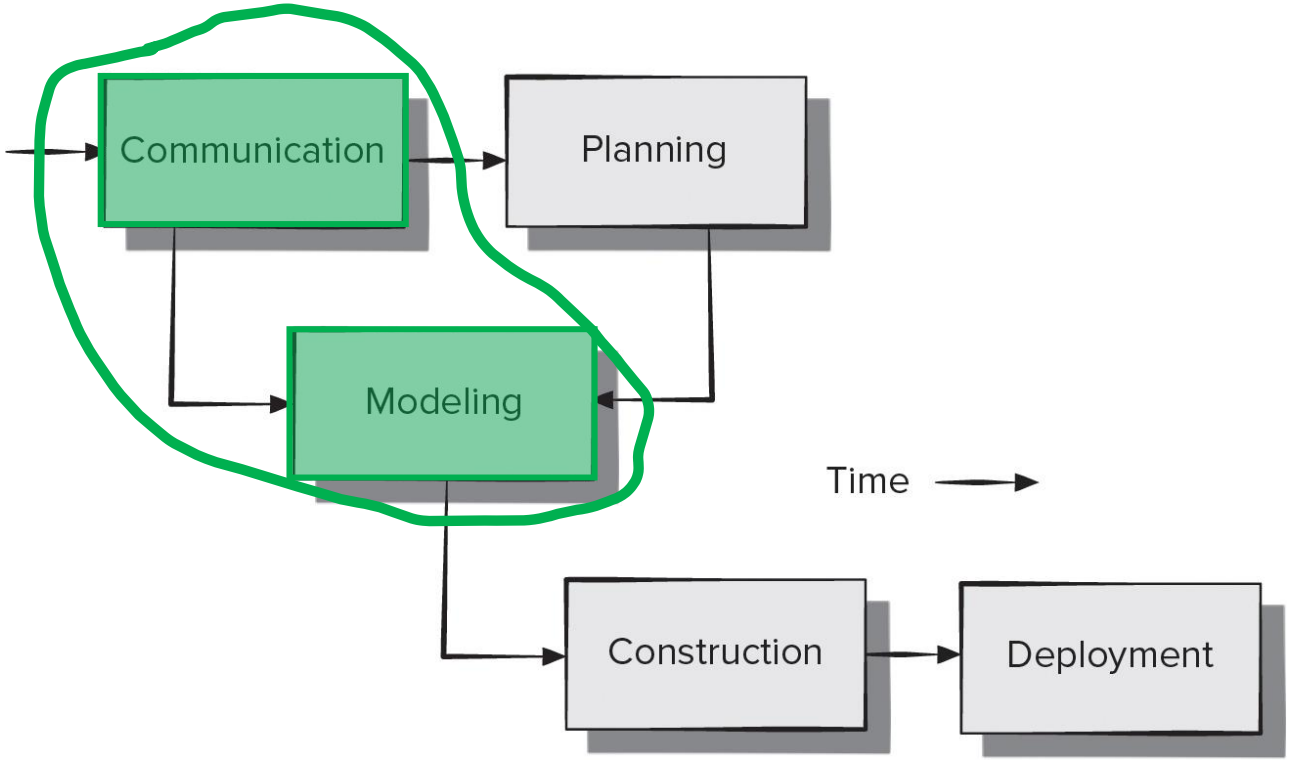
The user interface must be intuitive and easy to use.

FR

Users can create bookmarks of their favorite webpages.

Requirements Engineering

- **Requirements engineering** is the process of developing a complete requirements specification for a project.
- An **action** in the communication **activity** that continues into modeling.



Requirements Engineering

This action entails the following tasks:

- Inception
- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Requirements management

Requirements Engineering

This action entails the following tasks:

- **Inception**
- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Requirements management

Inception

- A basic understanding of the problem
- The people who want a solution
- The nature of the solution that is desired
- Important to establish effective customer and developer communication.

Requirements Engineering

This action entails the following tasks:

- Inception
- **Elicitation**
- Elaboration
- Negotiation
- Specification
- Validation
- Requirements management

Elicitation

- Elicit requirements and business goals form from all stakeholders.
- Goal is a long-term aim that a system must achieve.
- Goals deal with both functional and non-functional requirements.
- Establish prioritization mechanism.

Requirements Engineering

This action entails the following tasks:

- Inception
- Elicitation
- **Elaboration**
- Negotiation
- Specification
- Validation
- Requirements management

Elaboration

- Focuses on developing a refined requirements model that identifies aspects of software function, behavior, and information.
- Creation of user scenarios (user stories, use cases, etc.).
- Scenarios parsed to extract analysis classes and relationships between them.

Requirements Engineering

This action entails the following tasks:

- Inception
- Elicitation
- Elaboration
- **Negotiation**
- Specification
- Validation
- Requirements management

Negotiation

- Agree on the scope of a deliverable system that is realistic for developers and customers.
- Customers and stakeholders may ask for more than can be achieved or propose conflicting requirements.
- Conflicts need to be reconciled via negotiation.

Requirements Engineering

This action entails the following tasks:

- Inception
- Elicitation
- Elaboration
- Negotiation
- **Specification**
- Validation
- Requirements management

Specification

Can be any or all of the following:

- A written document (e.g., a Software Requirements Specification or SRS).
- A set of graphical models (e.g., use case diagrams).
- A formal mathematical model.
- A collection of user scenarios (e.g., use cases or user stories).
- A prototype.

Requirements Engineering

This action entails the following tasks:

- Inception
- Elicitation
- Elaboration
- Negotiation
- Specification
- **Validation** ~~is~~ revising, examining.
- Requirements management

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.

Requirements Engineering

This action entails the following tasks:

- Inception
- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- **Requirements management**

Requirements Management

Set of traceability activities to help the project team identify, control, and track requirements and their changes to requirements as the project proceeds.

Establishing the Groundwork

Identifying Stakeholders

- Stakeholders are *"anyone who benefits in a direct or indirect way from the system which is being developed".*
- The goal is to create a list of people who will contribute input as requirements are elicited.
- Don't be afraid to ask *"Who else do you think I should talk to?".*
- Stakeholder list grows as you are told who else to talk to.

Establishing the Groundwork

Recognize Multiple Points of View

- Every stakeholder will have their own perspective and point of view.
- While this might lead to inconsistent or conflicting requirements, it is critical to embrace and take advantage of the diversity of opinions here.

Work Toward Collaboration

- Identify areas of commonality and areas of conflict or inconsistency, and bring people together to express their views on requirements to guide final decisions.

Example Scenario

- You are undergoing requirements gathering for a web based application (e.g. a new social media website).
- The marketing department stakeholder representative wants large banner ads on the site. This is important for revenue generation and attracting potential advertisers.
- The user experience (UX) team is concerned the large banner ads will negatively impact the user experience and hurt their intuitive user interface design.
- **How could this conflict be resolved or successfully negotiated?**

Example Scenario

Some conflict resolution technique:

- **Agreement:** stakeholders work together to negotiate a solution to the conflict.
- **Compromise:** use alternative parts of various solutions to try and come up with a solution that could be a compromise for all stakeholders.
- **Voting:** ask all stakeholders involved with the requirements and/or the conflict itself to vote on a set of alternative options.
- **Overruling:** more senior stakeholder's requirements or proposed solution is the one that will be taken forward as the resolution.

Establishing the Groundwork

Asking the First Questions

- Who is behind the request for this work?
- Who will use the solution?
- What will be the economic benefit of a successful solution?
- Is there another source for the solution that you need?

These questions help identify the stakeholders, the benefits of a successful implementation, and possible alternatives to custom development.

Establishing the Groundwork

The Next Questions

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

These questions enable you to get a better understanding of the problem and allows the stakeholder to voice his or her perceptions about a solution.

Establishing the Groundwork

Finally ask:

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

These "meta questions" focus on the effectiveness of the communication activity itself and can be quite useful.

Requirements Gathering

- Combines elements of problem solving, elaboration, negotiation, and specification.
- Stakeholders work together with development team to:
  - Identify the problem.
  - Proposes elements of the solution.
  - Negotiate different approaches.
  - Specify a preliminary set of requirements.

Collaborative Requirements Gathering

A collaborative approach to requirements gathering applies some variation of the following guidelines:

- **Meetings** (real or virtual) are conducted and attended by both software engineers and stakeholders.
- **Rules** for preparation and participation are established.
- **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"*** (worksheets, flip charts, wall stickers or virtual forum) is used.

Collaborative Requirements Gathering

The goal here is:

- To identify the problem
- Propose elements of the solution
- Negotiate different approaches, and
- Specify a preliminary set of solution requirements

As requirements are gathered, an overall vision of system functions and features begins to materialize

Collaborative Requirements Gathering

A variety of work products can be created by requirements elicitation:

- 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 the process
- 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

communication with stakeholder

not necessary.

User Scenarios: User Stories

Title:	Priority:	Estimate:
User story **As a** [*type of user],* **I want to** [*perform some task]* **so that I can** [*achieve some goal].*		
Acceptance Criteria **Given that** [*some context],* **when** [*some action is carried out]* **then** [*a set of observable outcomes should occur].*		

Write a User Story

- Let's try writing a user story (you can do this individually or in a group).
- **Write a user story that describes the "favourites" (aka bookmarks) feature available on most Web browsers.**
- Use the template to the right.

Title:	Priority:	Estimate:

User story

As a [*type of user*],

I want to [*perform some task*] print page.

so that I can [*achieve some goal*]. Save the content on the webpage.

Acceptance Criteria

Given that [*some context*], user ....

when [*some action is carried out*] user press print

then [*a set of observable outcomes should occur*].

the content of the ~~website~~ webpage is saved as a pdf file.

Epics

- **Epics** are large bodies of work that can be broken down into a number of smaller tasks (i.e. **user stories**).
- **Epics** can be seen as a collection of related and interdependent **user stories**.
- The completion of these related **user stories** would lead to the completion of the **epic**.
- In an **agile process** such as scrum, **user stories** are something the team can commit to finish within a **single sprint**. While **epics** take longer to complete (e.g. **multiple sprints**).

User Scenarios: Use Cases

- **Use cases** are a collection of **user scenarios** that describe the thread of usage of a system.
- **Use cases** make it easier to understand **how the functions and features of a system will be used** by different classes of end users.
- This, in turn, facilitates more technical software engineering activities.
- **Use cases** can take many forms, including **narrative text**, an **outline of tasks or interactions**, a **template-based description**, or a **diagrammatic representation**, etc.

User Scenarios: Use Cases

external to the system.

- Each scenario is described from the point of view of an ***"actor"***—a person or device that interacts with the software in some way.
- It is important to note that an **actor** and an **end user** are **not necessarily the same thing**.
- A typical **user** may play a number of different roles when using a system, whereas an **actor** represents a class of entities that play **just one role** in the context of the **use case**.
- Understanding **actors** is a critical part of understanding requirements

Use Cases vs. User Stories

Use Cases

- Short or lengthy descriptions
- User flow or interaction
- In-depth guidance
- Detailed to write

User Stories

- Short descriptions
- Requirement's who and why
- General guidance
- No technical detail

Use Cases

initiate use case

involve the use case

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 external environment changes?
- What information does the **actor** desire from the system?
- Does the **actor** wish to be informed about unexpected changes?

Example Use Cases: SafeHome

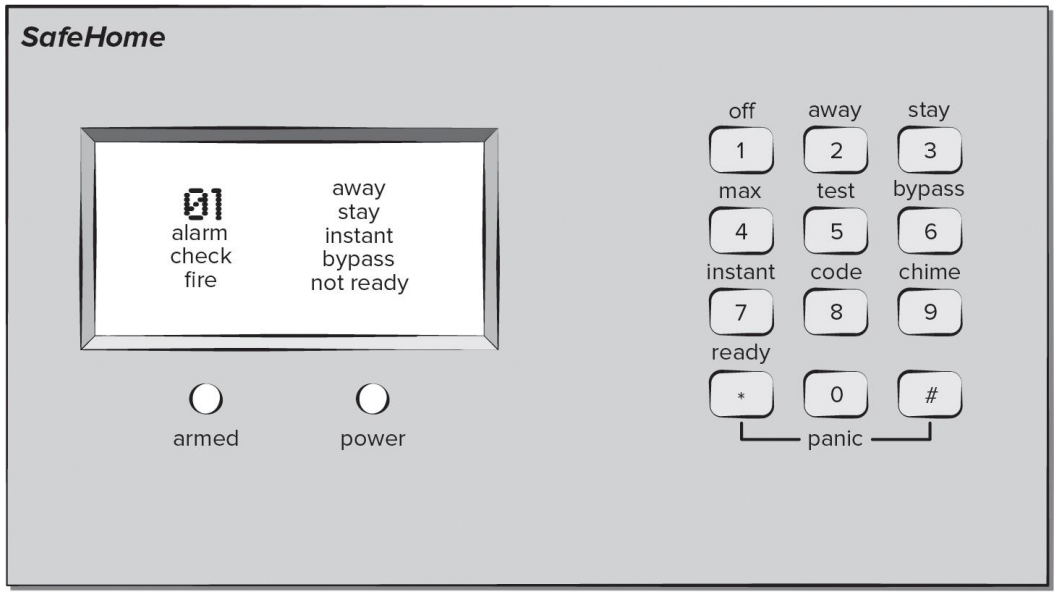
Scenario

Security system where homeowner uses control panel, tablet or cell phone to control and monitor their home security system.

The homeowner: ← **The homeowner is an actor.**

1. Arms/disarms the system with a pin.
2. Access the system remotely via the internet.
3. Responds to alarm events.
4. May encounter and deal with error conditions.

Control Panel



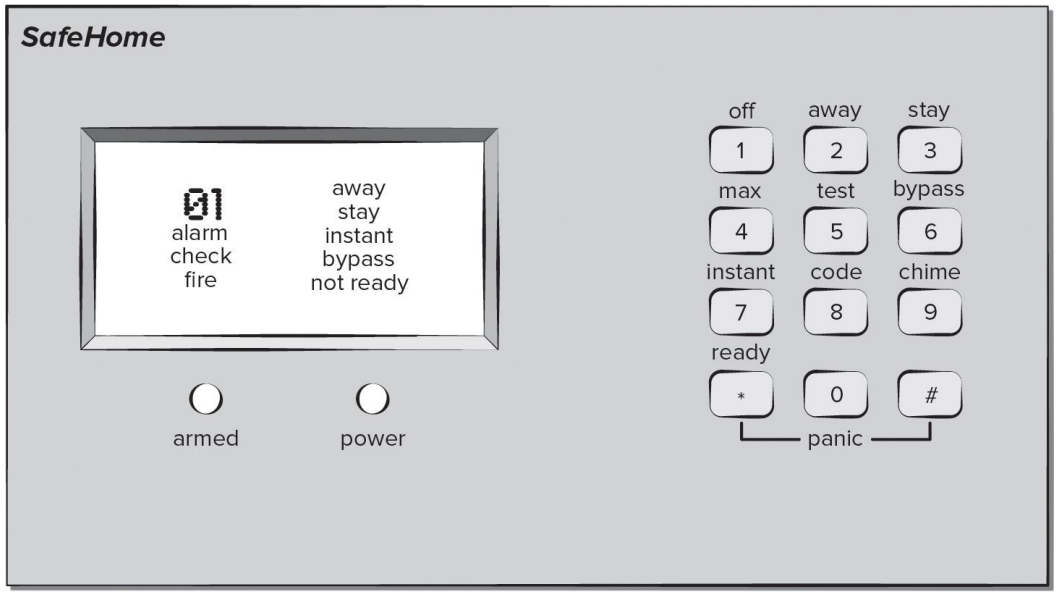
Each of these is it's own Use Case.

Example Use Cases: SafeHome

Scenario

Security system where homeowner uses control panel, tablet or cell phone to control and monitor their home security system.

Control Panel



The system administrator:

1. Reconfigures sensors and related functions.

2. May also be a homeowner and can do all things a homeowner can.

Example Use Cases: SafeHome

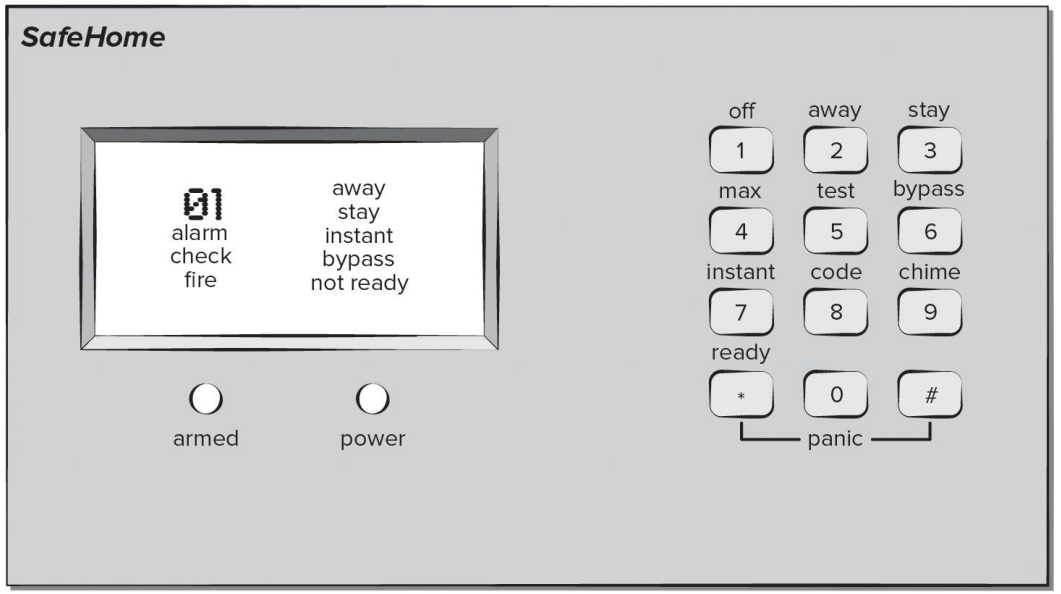
Scenario

Security system where homeowner uses control panel, tablet or cell phone to control and monitor their home security system.

Sensors:

1. Respond to alarm events.
2. May encounter and deal with error conditions.
3. Are configured and reconfigured by a system administrator.

Control Panel



Example Use Case: SafeHome

Specifically we will be looking at the situation where the homeowner uses the control panel to activate the system which follows these steps:

1. Homeowner observes the control panel to determine if the system is ready for input. If it is not ready, a message is displayed on the LCD (could be due to a door/window being open).

2. Homeowner uses the keypad to key in a four-digit password. Password is compared to one stored in the system. If incorrect, panel will beep and reset for additional input. If correct, panel awaits further action.

3. Homeowner keys in “stay” or “away” to activate the system. “Stay” activates only perimeter sensors and “away” activates all sensors.

4. When activated a red alarm light can be observed on the panel.

The Actor Template

The Actor Template

Actor:	*The actor's name.*
Description:	*Brief description of the actor and its role in the system. This description should be no more than a small paragraph and should give the reader an understanding of the role of the actor in the organization.*
Aliases:	*Any other names by which this actor may be known. A simple list is sufficient. Say 'None' if there are none.*
Inherits:	*The ancestors for the actor. Some actors may be specialized types of other actors. A simple list of the names of the ancestors will suffice. Say 'None' if there are none.*
Actor Type:	*Whether the actor is a person or an external system, as well as other forms of typing (e.g. are they an API, hardware device, etc.).*
Active/Passive:	*If this actor is active or passive. They are the ones who initiate a use case. In most cases active actors both send and receive input/output from the system. Passive actors are part of a use case but do not initiate it and are not the main focuses of the case (e.g. they may only receive output or only give input after the case is triggered).*

Example Use Case: SafeHome

Identify The Actors

Actor:	**Homeowner**
Description:	The homeowner interacts with the home security system using the control panel, tablet, or cell phone. They are the end user of the system and control the arming/disarming of the system.
Aliases:	User, End User
Inherits:	None
Actor Type:	Person, End User
Active/Passive:	Active

Example Use Case: SafeHome

Identify The Actors

Actor:	**System Administrator**
Description:	System administrator with the ability to reconfigure sensors, reset passwords, and modify security system features.
Aliases:	Admin, Administrator
Inherits:	**Homeowner**
Actor Type:	Person
Active/Passive:	Active

Example Use Case: SafeHome

Identify The Actors

Actor:	**Sensor**
Description:	The various security sensors attached to the system, including motion detectors, cameras, door/window sensors, etc.
Aliases:	None
Inherits:	None
Actor Type:	External device
Active/Passive:	Passive

Would be active, if it initiates a use case (e.g. sets off an alarm).

The Use Case Template

Use Case:	*A name given to the use case.*
Primary Actor:	*The main actor in the use case; the use case is from their perspective.*
Secondary Actor:	*Other actors involved in the use case.*
Goal in Context:	*The overall scope of the use case, provides a brief description of the use case and its purpose.*
Preconditions:	*What is known to be true before the use case is initiated.*
Trigger:	*What event or condition gets the use case started or invoked.*
Scenario:	*A numbered series of steps that capture the narrative of the use case, outlining what the actors do in this use case and what happens as a result.*
Alternatives:	*If alternative behaviour is possible at any of the steps outlined in the Scenario, it should be described here in a similar fashion. Say 'None' if there is no such alternative behaviour.*
Exceptions:	*Identify potential issues or situations that may arise from the various steps of this use case.*
Priority:	*One of: Highest, High, Medium, Low, Lowest. This is the priority your team is assigning to this use case (how important it is to implement, how essential it is to the function and goals of the application, etc.).*
….Extra Fields…	*Additional headings and information may be provided as you deem necessary.*

Example Use Case: SafeHome

The Use Case Template

Use Case:	
Primary Actor:	
Secondary Actor:	
Goal in Context:	
Preconditions:	
Trigger:	
Scenario:	
Alternatives:	

⋮

Example Use Case: SafeHome

The Use Case Template

⋮

Exceptions:	
Priority:	
….Extra Fields…	

Example Use Case: SafeHome

The Use Case Template

Exceptions:	1. Control panel is **not ready:** Homeowner checks all sensors to determine which are open and then closes them. 2. Password is incorrect (control panel beeps once): Homeowner reenters correct password. 3. Password not recognized and user locked out: 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:	Highest, must be implemented.
When Available:	First increment.
Frequency of Use:	Many times per day.
Channel to actor:	Via control panel interface.
Channels to secondary actors:	**Sensors:** hardwired and radio frequency interfaces.

Extra fields and additional information

Example Use Case: SafeHome

The Use Case Template

⋮

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?

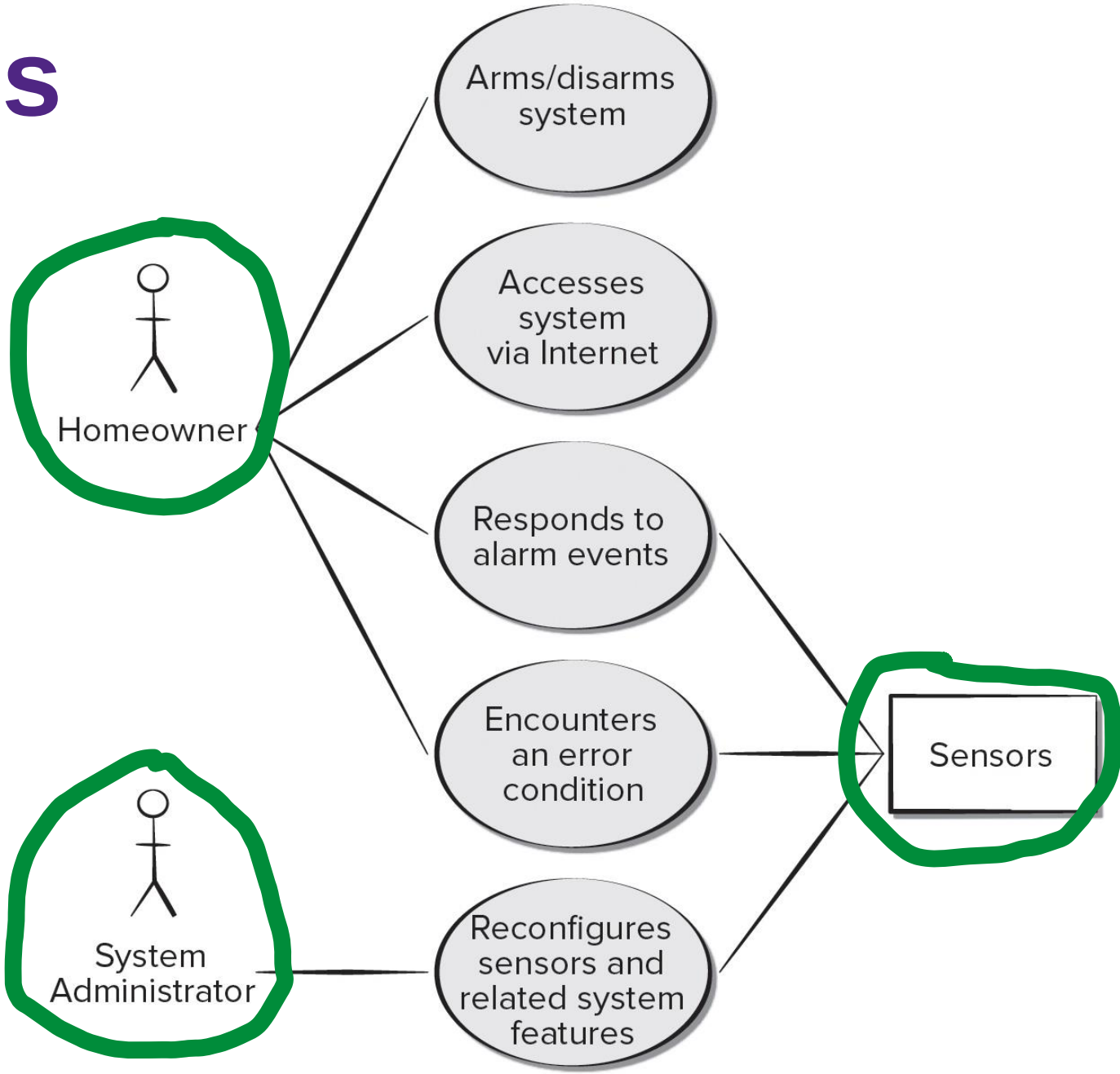
UML Use Case Diagrams

- **Use case diagrams** are a visual way of capturing **use cases**.
- Typically, a **use case diagram** is an overview of **all use cases** (at least from a key actor's perspective) and **how they are related**.
- Act as a graphical table of contents for the **use case** set.
- Visually depict the relationships between **use cases** and **actors**.
  - Which actors carry out which use cases.
  - Which use cases include other use cases.
- As such, **use case diagrams** provide a **big picture of the functionality** of the software system in question.

Use Case Diagrams

Actors

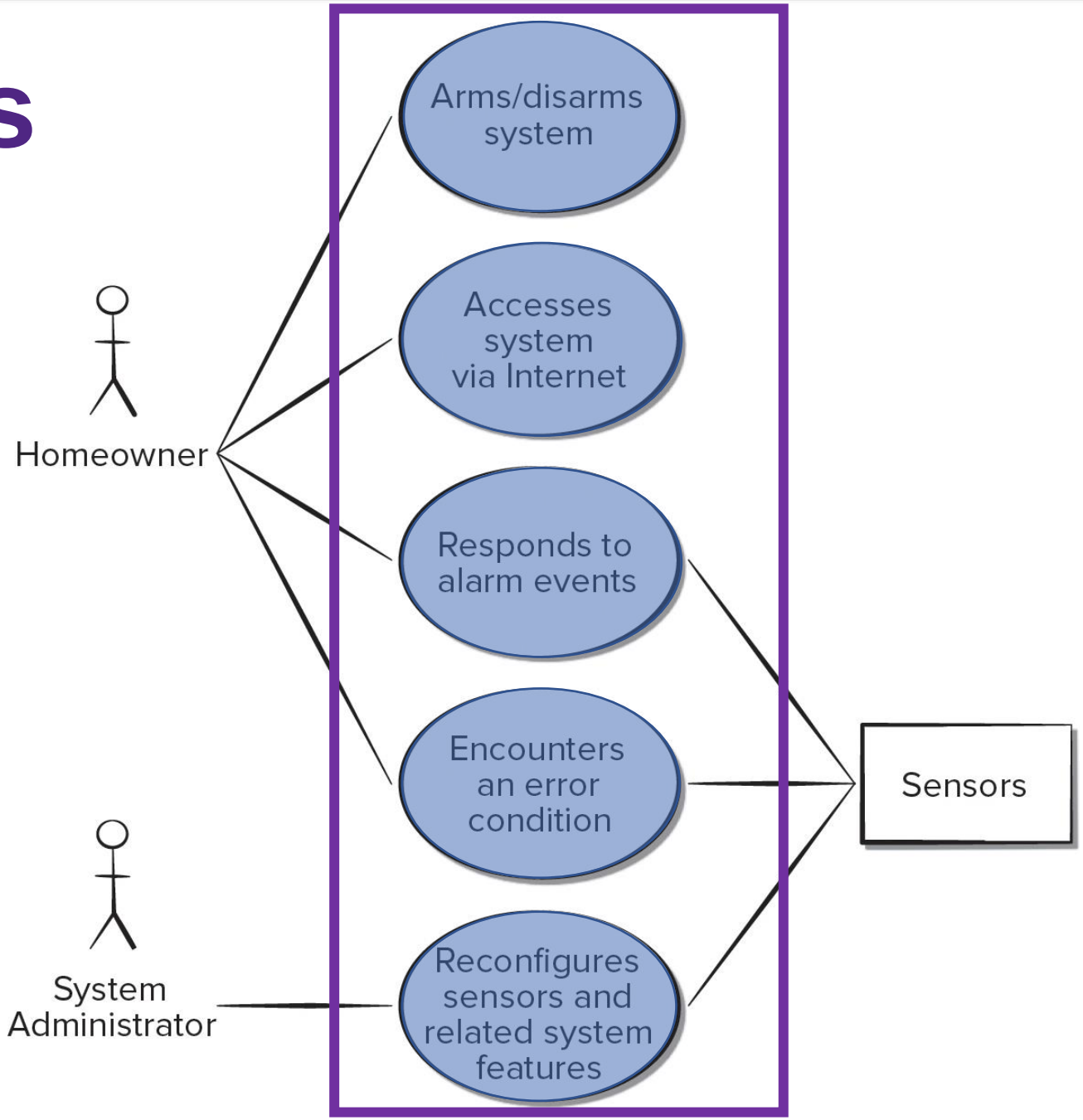
- Stick figure in a use case diagram represents an actor that is associated with one category/role of user or other element that interacts with the system.
- Complex systems may have many actors.
- In some cases, non-person actors are represented by different symbols (e.g., sensor is shown in a box).



Use Case Diagrams

Use Cases

- Use cases are displayed as ovals.
- The actors are connected by lines to the use cases that they carry out.
- Use cases are placed in a rectangle, but the actors are not; the rectangle represents the **boundaries of the system**.



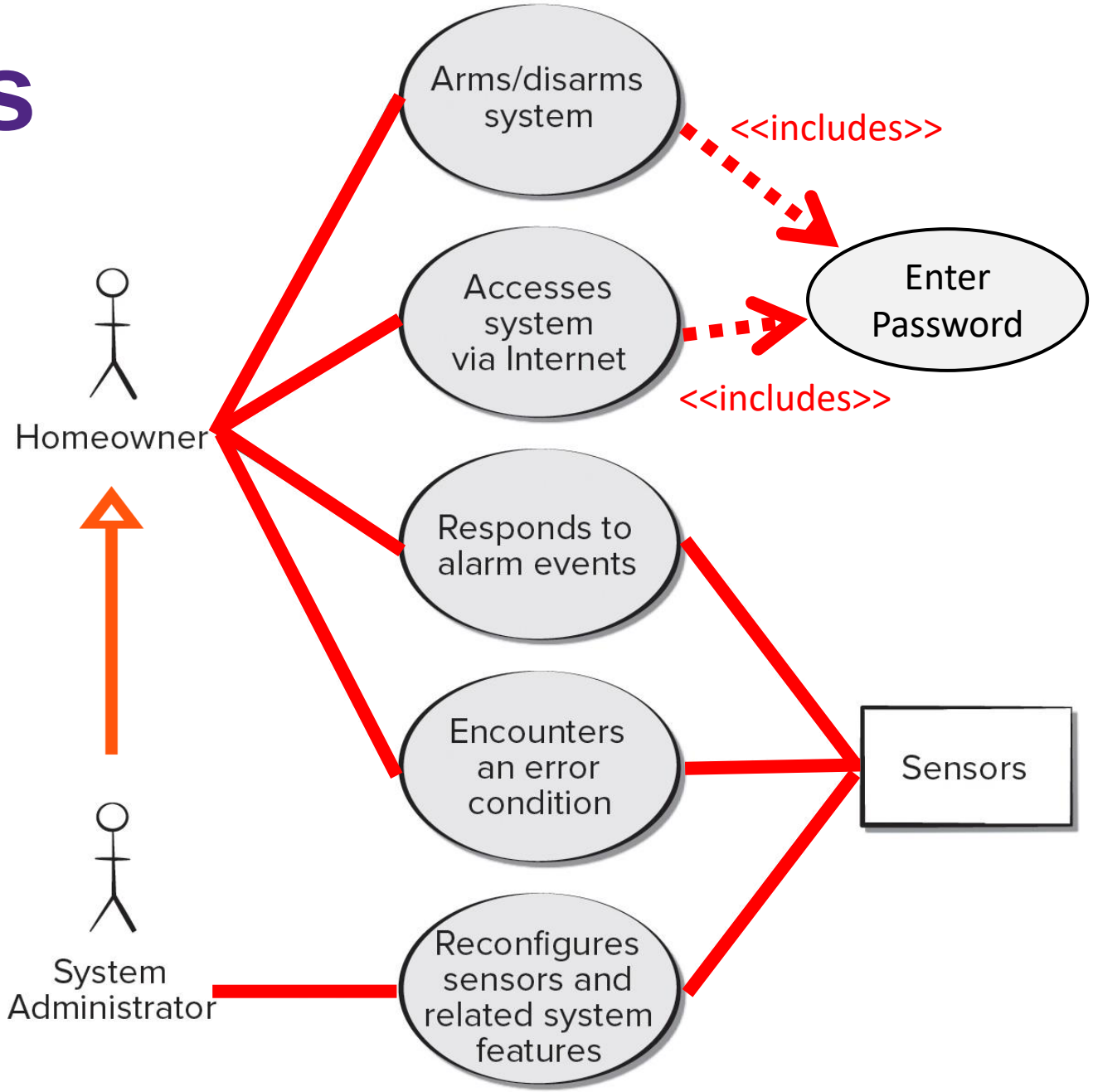
Use Case Diagrams

Relationships

Association

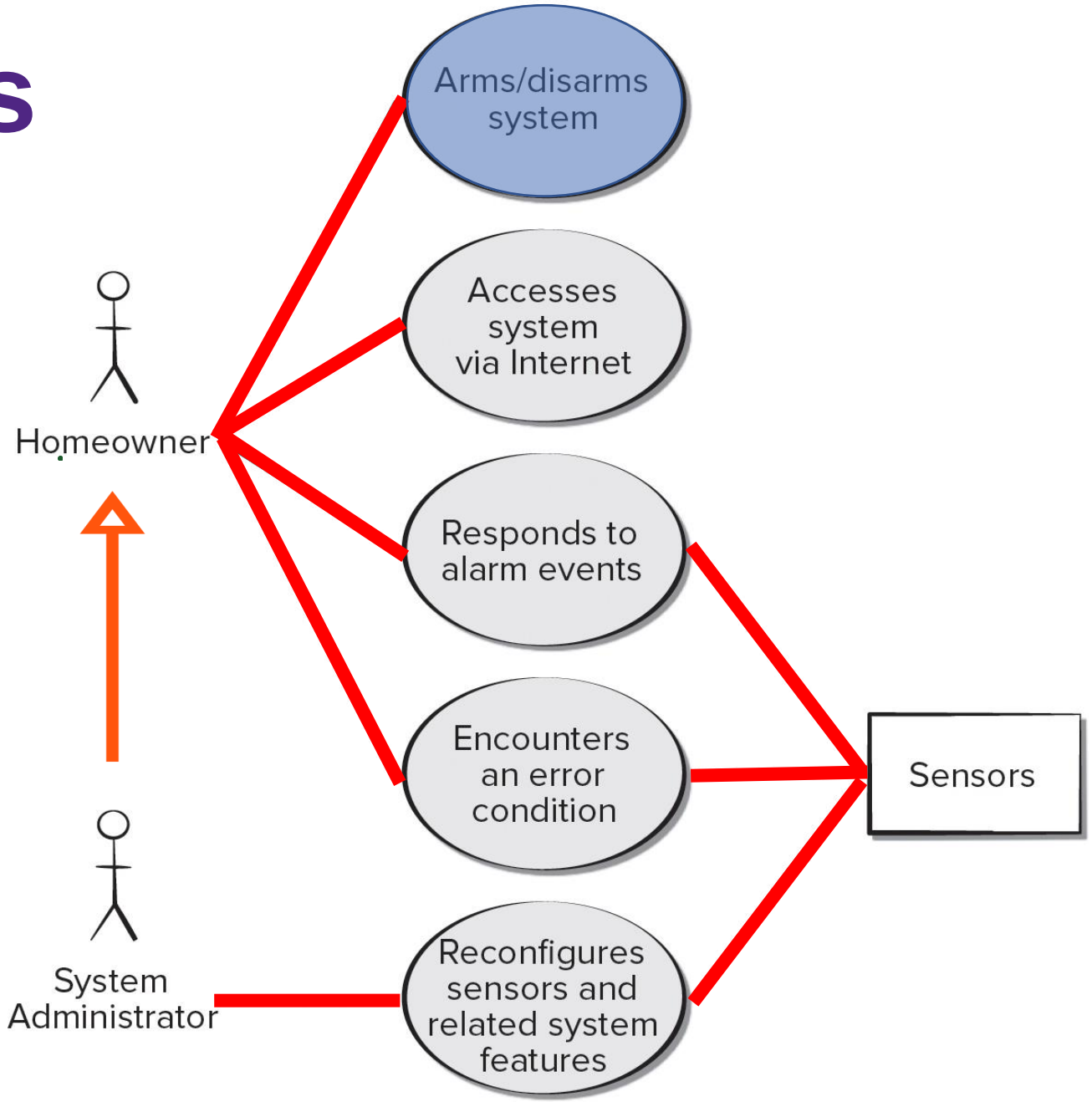
Dependency *(for includes/extends)*

Generalizes



Use Case Diagrams

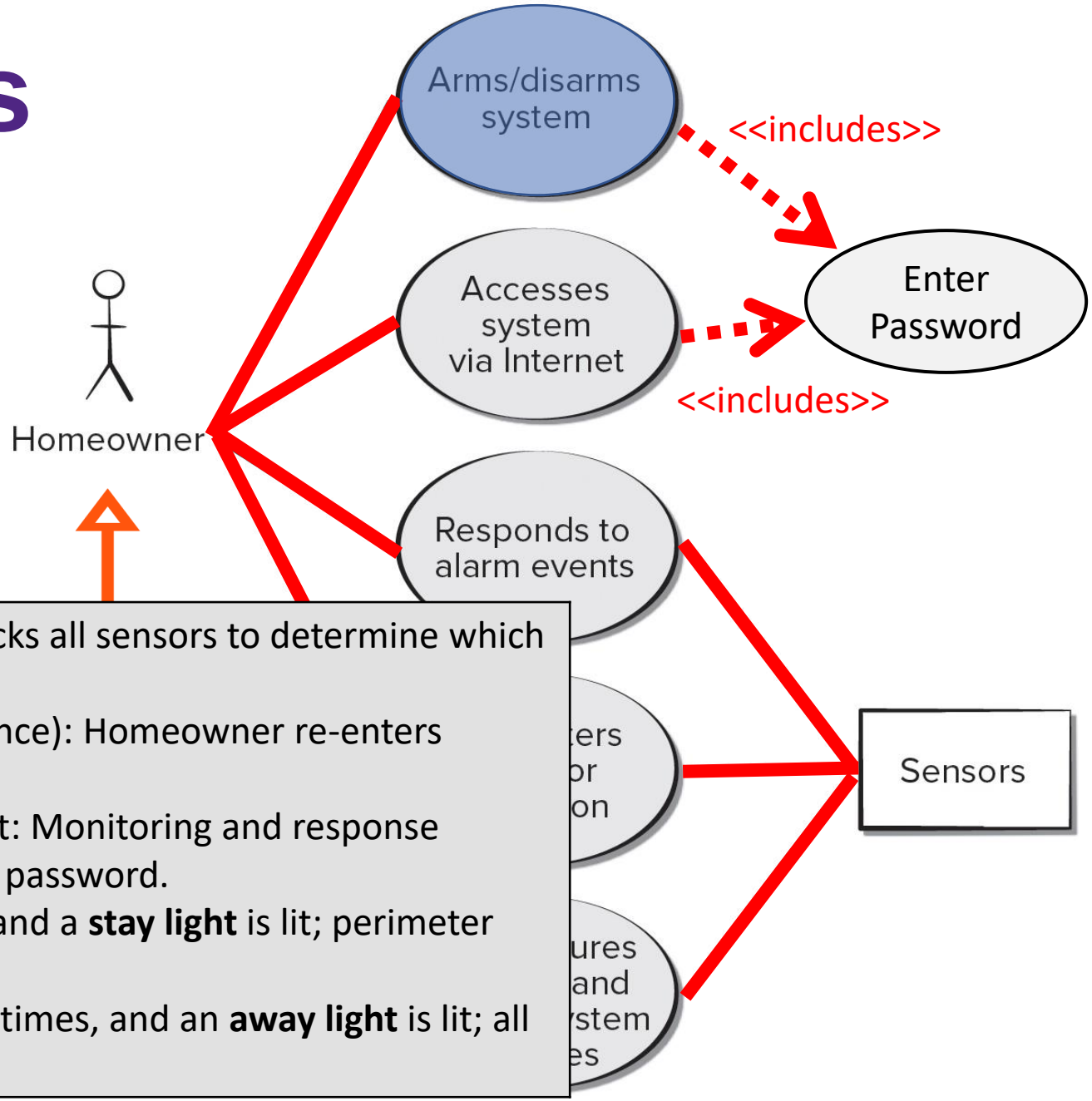
Scenario:	1. Homeowner observes control panel. 2. Homeowner enters password. 3. Homeowner selects “stay” or “away”. 4. Homemowner observes red alarm light to indicate that SafeHome has been armed.



Use Case Diagrams

Scenario:	1. User logins in (see **Enter Password**) 2. Homeowner selects “stay” or “away”. 3. Homemowner observes red alarm light to indicate that SafeHome has been armed.

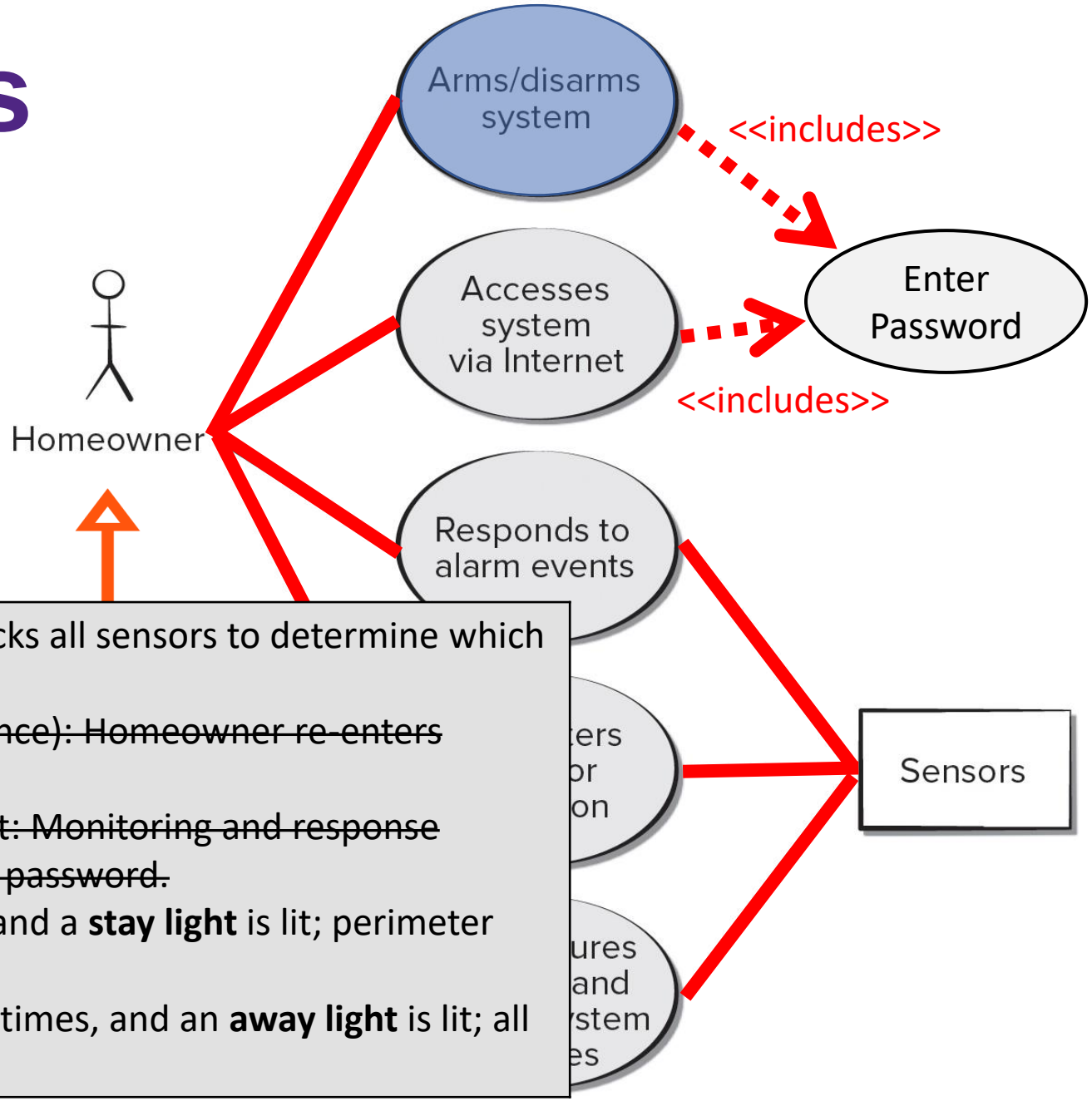
Exceptions:	1. Control panel is **not ready:** Homeowner checks all sensors to determine which are open and then closes them. 2. Password is incorrect (control panel beeps once): Homeowner re-enters correct password. 3. Password not recognized and user locked out: Monitoring and response subsystem must be contacted to re-program 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.



Use Case Diagrams

Scenario:	1. User logins in (see **Enter Password**) 2. Homeowner selects “stay” or “away”. 3. Homemowner observes red 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 and then closes them. ~~2. Password is incorrect (control panel beeps once): Homeowner re-enters correct password.~~ ~~3. Password not recognized and user locked out: Monitoring and response subsystem must be contacted to re-program 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.



Final Thoughts on Use Case Diagrams

- Use case diagrams are helpful in ensuring that you have covered all of the functionality of the system as you get to see the system as a whole.
- Note that **none of the details of use cases are included in the diagrams**, however, and such details need to be stored separately.
- These details are still important to the software development process, and are often considered to be more important than the overall use case diagram.
- Important not to over complicate a use case diagram. <<*includes*>> should also be added only if they make the diagram simpler, not more complex.

Use Case Diagrams Activity

Create an Use Case Diagram for this scenario.

Fill in any extra details needed.

- Create a Use Case Diagram for the CS1 ASK Tool.
- This is the tool we use in class to ask/answer questions live in-class, enter groupwork codes, enter participation tickets, and view our participation stats.
- Your instructor uses this tool to generate groupwork and participation keys, view participation statistics for a given student, view class participation statistics as a whole, and view incoming questions/answers during class.
- Assume that the CS1 ASK Tool communicates with an external system (OWL) when validating codes and viewing participation statistics.
- Brainstorm what actors and use cases are involved in this scenario and then create a Use Case Diagram.
- You do not have to detail or explain the Use Cases, just name them.

Negotiating Requirements

- Ideally, **inception**, **elicitation**, and **elaboration** tasks determine requirements cleanly and in sufficient detail to move forward.
- This rarely happens in reality, and some form of **negotiation** is necessary.
- In **negotiation**, stakeholders are asked to balance functionality, performance, and other product or system characteristics against cost and time-to-market needs.
- The intent of **negotiation** is to develop a project plan that meets stakeholder needs while at the same time reflecting the real world constraints on the project.

Negotiating Requirements

- **Negotiation** typically involves:
  - **Identification of key stakeholders** who will be involved in the **negotiation**.
  - Determination of each of the **stakeholder's "win conditions"**, which are not always obvious.
  - **Negotiation of the stakeholders' various win conditions** to reconcile them to a set of "win-win" conditions for all concerned.
- Successful completion of these steps achieves a **win-win result that gets the majority of stakeholder needs met**, while doing so in a way that is **achievable by the team** considering time and budget constraints.

Validating Requirements

- As each element of the requirements model is created, it is examined for **inconsistency**, **omissions**, and **ambiguity**.
- A review of the requirements model addresses the following questions:
  - 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?

Validating Requirements

- **More questions:**
  - Is each requirement **bounded and unambiguous**?
  - Does each requirement have **attribution**? That is, is a source (generally, a specific individual) noted for each requirement?
  - Do any requirements **conflict** with other requirements?
  - Is each requirement **achievable** in the technical environment that will house the system or product?
  - Is each requirement **testable**, once implemented?

Validating Requirements

- **Even More questions:**
  - 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?

Requirements Monitoring

- Monitoring is especially needed in incremental development to ensure that requirements continue to be met over time.
  - **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