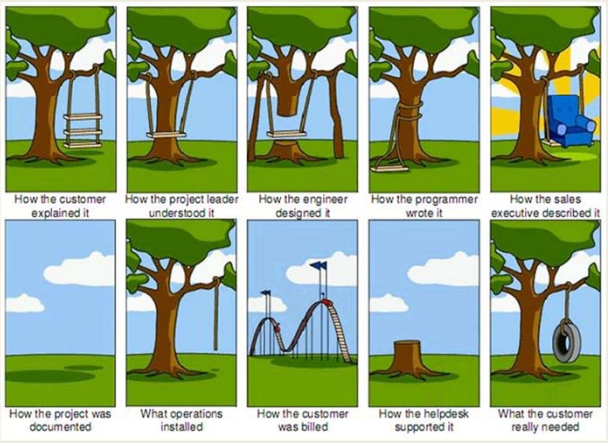
Requirements Engineering



What is requirement engineering (RE)

- A broad spectrum of tasks and techniques that lead to an understanding of requirements.
- Builds a bridge to design and construction of software.
- Provides the appropriate mechanism for
  - Understanding what the customer wants,
  - analyzing need,
  - assessing feasibility,
  - negotiating a reasonable solution
  - specifying the solution unambiguously
  - validating the specification
  - Managing the requirements as they are transformed into an operational system

Requirements Engineering-I

- Inception – ask a set of questions that establish …
  - basic understanding of the problem
  - the people who want a solution
  - the nature of the solution that is desired, and
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
- Elicitation – elicit requirements from all stakeholders
  - Stakeholders including the people who may directly or indirectly benefit from a software product: business operation manager, marketing people, customers, etc.
- Elaboration – create an analysis model that identifies data, function and behavioral requirements
- Negotiation – agree on a deliverable system that is realistic for developers and customers

RE encompasses seven distinct tasks: inception, elicitation, elaboration, negotiation, specification, validation, and management.

Requirements Engineering-II

- 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
- 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 management
  - Establish traceability between requirements and software artifacts as the development proceeds.

Inception

- Identify stakeholders
  - "who else do you think I should talk to?"
- Recognize multiple points of view
- Work toward collaboration
- The first questions
  - Who is behind the request for this work?
  - Who will use the solution?
    - A person using a product may not be the person requesting it.
    - E.g., Marketing people request it for a scope of customers.
  - What will be the economic benefit of a successful solution
    - Commercial product has to have a business purpose.
  - Is there another source for the solution that you need?

a stakeholder as "anyone who benefits in a direct or indirect way from the system which is being developed

Because many different stakeholders exist, the requirements of the system will be

explored from many different points of view. For example, the marketing group is interested

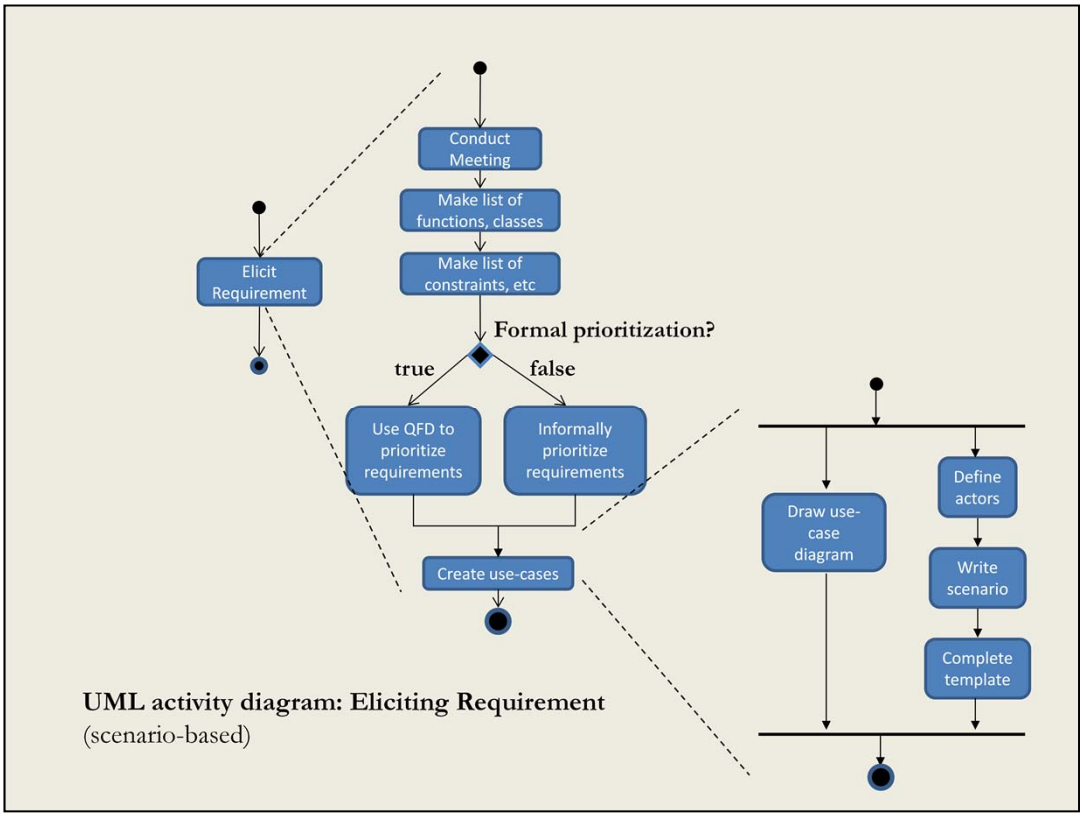
in functions and features that will excite the potential market, Business managers are interested in a feature set that can

be built within budget. End users may want features that are familiar to them and that are easy to learn and use.

customers (and other stakeholders) must collaborate among themselves (avoiding petty turf battles) and with software engineering practitioners

Eliciting Requirements

- Meetings are conducted and attended by both software engineers and customers
- Rules for preparation and participation are established
- An agenda is suggested
- 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 goal is
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches, and
  - specify a preliminary set of solution requirements



UML activity diagram: Eliciting Requirement
(scenario-based)

Quality Function Deployment (QFD)

- Function deployment determines the "value" (as perceived by the customer) of each function required of the system
- Information deployment identifies data objects and events
- Task deployment examines the behavior of the system
- Value analysis determines the relative priority of requirements

Quality function deployment (QFD) is a quality management technique that translates

the needs of the customer into technical requirements for software. QFD "concentrates

on maximizing customer satisfaction from the software engineering process"

Quality Function Deployment (QFD)

Identifies three types of requirements

- Normal requirements: The objectives and goals that are stated for a product or system during meetings with the customer.
- Expected requirements: These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction.
- Exciting requirements: These features go beyond the customer's expectations and prove to be very satisfying when present.

Elicitation Work Products

- 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.

Use-Cases

- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "actor"—a person or device that interacts with the software in some way
  - Actors v.s. users:
    - An actor is a role and a user may have multiple roles.
    - An actor can be an object providing inputs. E.g., sensors

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

to materialize. However, it is difficult to move into more technical software engineering

activities until you understand how these functions and features will be

used by different classes 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* [Jac92], provide a description of how

the system will be used.

Use-Cases

- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

UML Characteristics

- The de facto standard software modeling language in practice.
- Object-orientation
- Visualization, expressive
- Independent on processes and methodologies
- Easy to understand and use concepts, notations and structures

UML Constructs

- Basic Building Blocks
  - Things:
    - Structural
      - Class, interface, collaboration, use case, active class, component, node
    - Behavioral
      - Interaction, state machine
    - Grouping
      - Package
    - Annotational
      - Note

UML Constructs

- Basic Building Blocks (Cont.)
  - Relationships:
    - Dependency
    - Association
    - Generalization
    - Realization
  - Diagrams
- Rules
- Common mechanisms

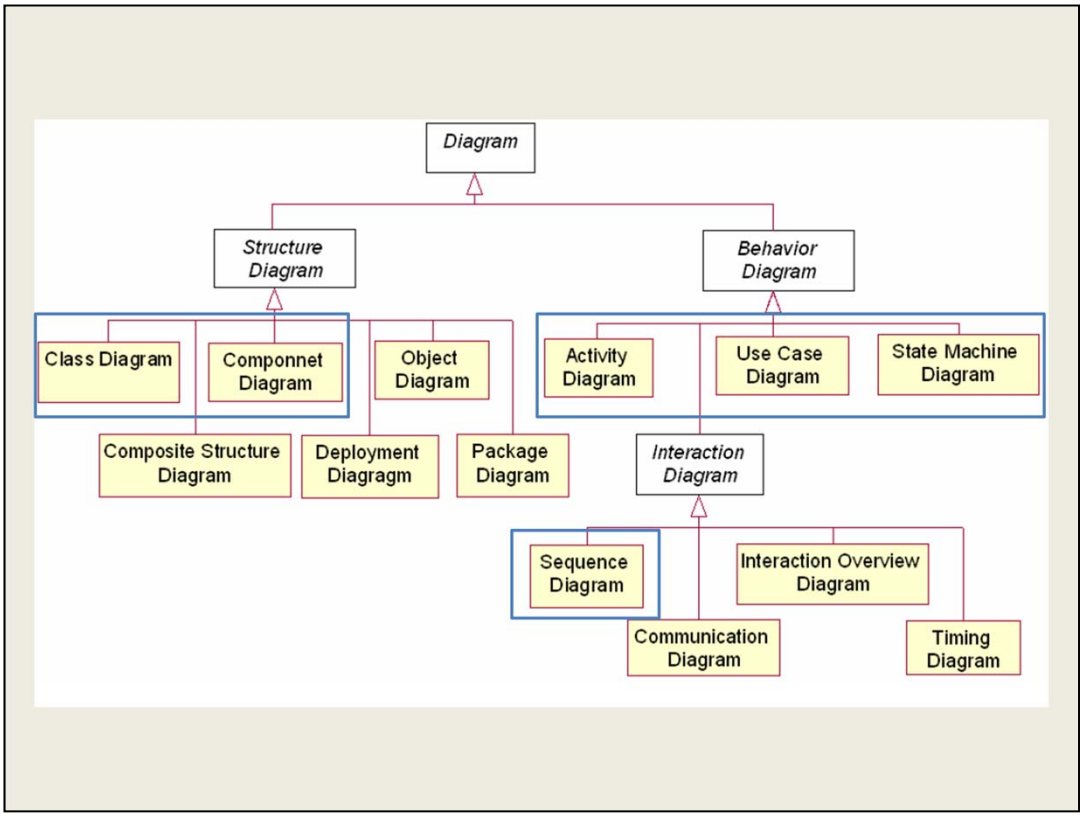
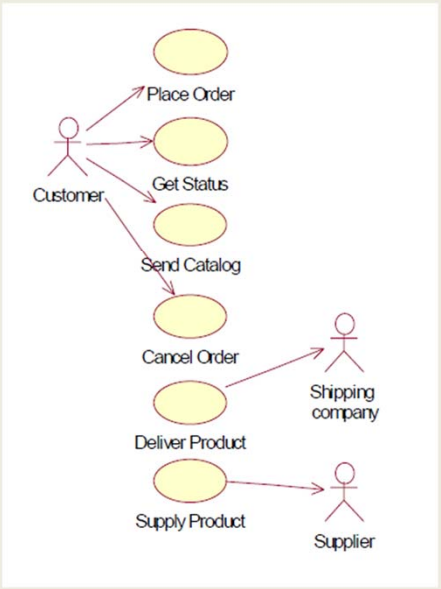


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Componnet Diagram
Object Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Composite Structure Diagram
Deployment Diagragm
Package Diagram
Interaction Diagram
Sequence Diagram
Interaction Overview Diagram
Communication Diagram
Timing Diagram

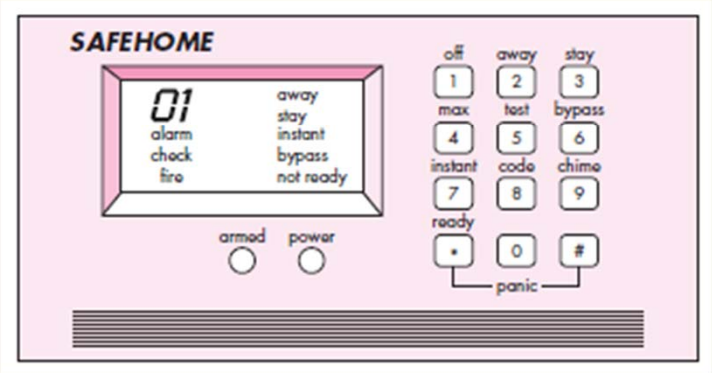
Use Case Diagram

- Created to visualize the interaction of your system with the outside world.



Example: SafeHome

- Control panel / User interface



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.

Example: SafeHome

Basic use cases

- 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.

Example: SafeHome

Template

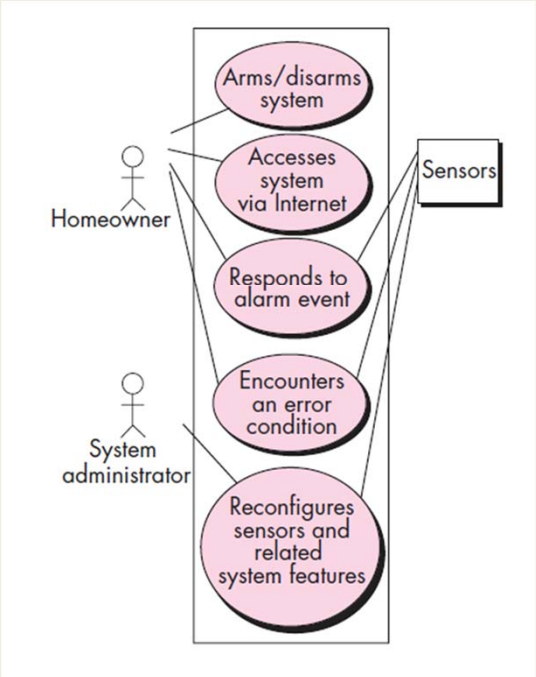
- Use case name
- Primary actor
- Goal in context
- Preconditions
- Trigger
- Scenario
- Exceptions
- Priority
- When available
- Frequency of use
- Channel to actor
- Secondary actors
- Channels to secondary actors
- Open issues

Example: SafeHome

Example use case

- **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

Use-Case Diagram



Summary

- RE: concept
- RE: seven tasks
- Tasks
  - Inception
  - Elicitation
    - Use case development

About feedback collection;

Homeworks: emphasizing on policy (no excuse; notification in advance if having difficulties, otherwise zero point, see syllabus)

Homework 2 release, check Blackboard/Piazza