CSE4006 Software Engineering

05. Requirement Engineering

Scott Uk-Jin Lee

Department of Computer Science and Engineering
Hanyang University ERICA Campus

1st Semester 2015

Requirement Engineering

- The hardest single part of building a software system is deciding what to build
- Requirement engineering provides mechanisms to:
  - understand what customer really wants
  - analyse requirements
  - evaluate implementation possibility
  - negotiate a reasonable solution
  - specify solution concisely without any ambiguity

Requirement Engineering

- **Inception**
  - ask a set of questions that establish:
    - basic understanding of the problem (what)
    - the people who want a solution (who)
    - the nature of the solution that is desired
    - the effectiveness of **preliminary communication** and collaboration between stakeholders and software engineers
- **Elicitation**
  - elicit requirements from all stakeholders
- **Elaboration**
  - create an **analysis model** that identifies **data**, **function** and **behavioral** requirements
- **Negotiation**
  - agree on a deliverable system that is realistic for software engineers and stakeholders

Requirement Engineering

- **Specification**
  - can be any one (or more) of the following:
    - a written document
    - a set of models
    - a collection of user scenarios (use-cases)
    - a prototype
- **Validation**
  - are view mechanism that looks for
    - errors in content or interpretation
    - areas where clarification may be required (ambiguity)
    - missing information (incomplete requirement)
    - inconsistencies - a major problem when large products or systems are engineered
    - unrealistic (unachievable) requirements
- **Requirements Management**

Inception

- Identify stakeholders
  - “Who else do you think I should talk to?”
- Recognize multiple points of view
  - different stakeholders → various point of view and opinions
- Work toward collaboration
- The first questions (context-free)
  - 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?

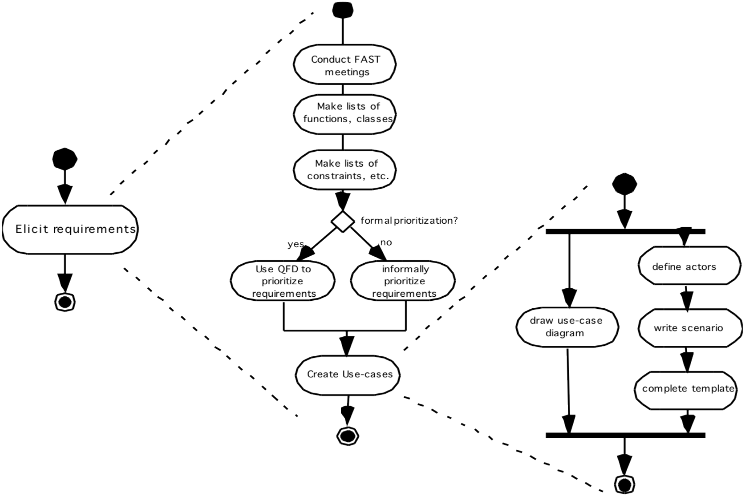
Eliciting Requirements

- Collaborative requirements gathering
  - meetings are conducted and attended by both software engineers and customers
  - an agenda is suggested
    - distributed in advance with product request
  - a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
  - a "definition mechanism"
    - work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum

Eliciting Requirements

- Goal
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches
  - specify a preliminary set of solution requirements
- Requirement gathering
  1. collaborative elicitation
  2. individual list of attendees ⇒ combined lists
  3. combined lists are shortend, lengthened, or reworded to suit development system ⇒ consensus lists

Eliciting Requirements

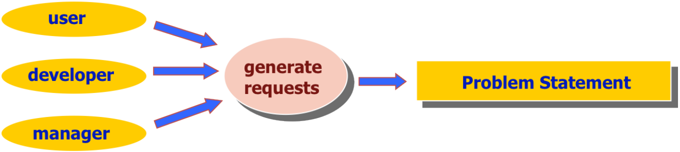


Quality Function Deployment (QFD)

- QFD is a technique to convert customers' requirements into the technical requirements for software
- Requirements of customers
  - Normal Requirement
    - graphical displays, specific system functions, defined levels of performance
  - Expected Requirement
    - ease of human/machine interaction, overall operational correctness, reliability, ease of SW installation
  - Exciting Requirement
    - provides various formats in word processing software

Quality Function Deployment (QFD)

- **Function deployment** determines 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 during each of the three deployments
  - Value should be one that are perceived by the customer



Elicitation Work Products

- Work products = the result of 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 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

lab(se);

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"
  - actor: a person or device that interacts with the software in some way
  - actor indicates the role

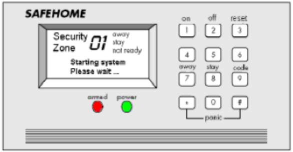
    e.g., a machine operator = 4 actors (programmer, tester, monitor, trouble shooter)
  - each actor has one or more goals when using system

lab(se);

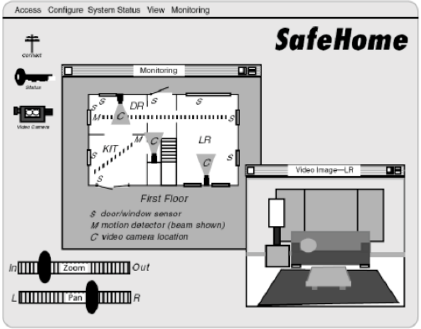
Use-Cases

- 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?

Example: SafeHome Project

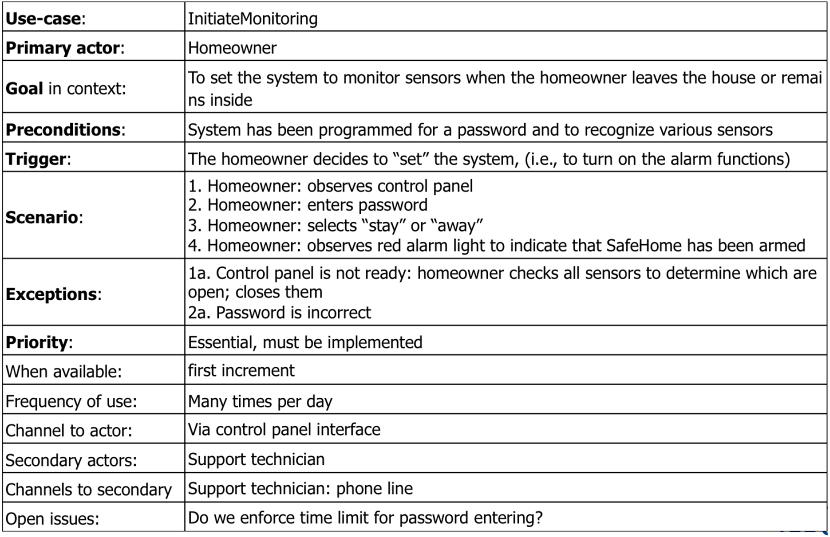


SafeHome control panel



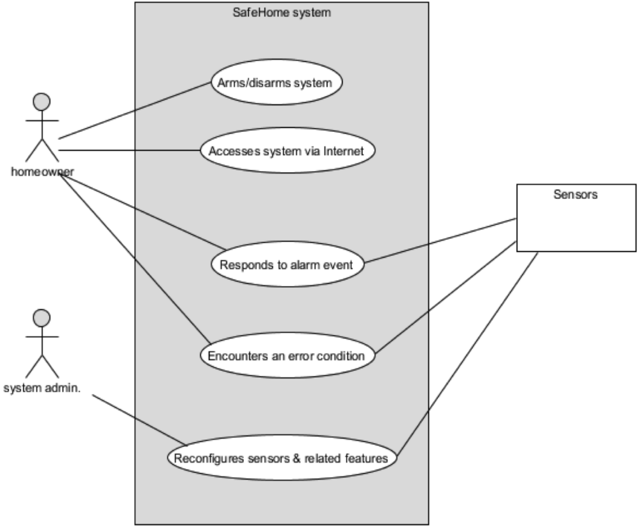
Preliminary screen layout for video monitoring

Example: SafeHome Project



Use-case:	InitiateMonitoring
Primary actor:	Homeowner
Goal in context:	To set the system to monitor sensors when the homeowner leaves the house or remai ns 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 red alarm light to indicate that SafeHome has been armed
Exceptions:	1a. Control panel is not ready: homeowner checks all sensors to determine which are open; closes them 2a. Password is incorrect
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
Channels to secondary	Support technician: phone line
Open issues:	Do we enforce time limit for password entering?

Use-Case Diagram


SafeHome system
Arms/disarms system
Accesses system via Internet
homeowner
Sensors
Responds to alarm event
Encounters an error condition
system admin.
Reconfigures sensors & related features

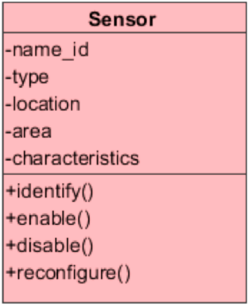
lab(se);

Building Analysis Model

- Elements of the analysis model
  - Scenario-based elements
    - Functional: processing narratives for software functions
    - Use-case: descriptions of the interaction between an "actor" and the system
  - Class-based elements
    - Implied by scenarios (scenarios implies a set of "objects")
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

Class Diagram

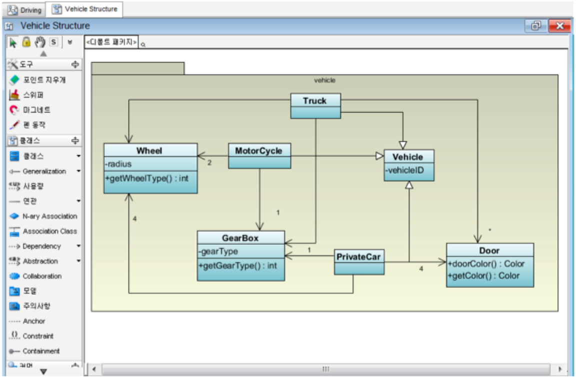
- from the SafeHome system



Sensor

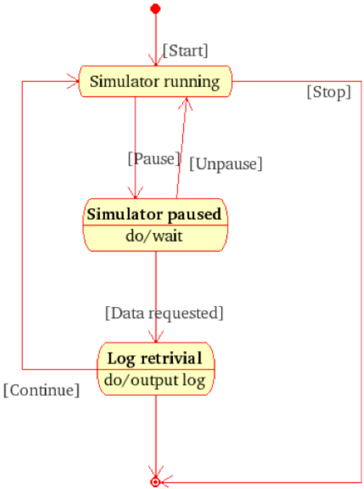
-name_id
-type
-location
-area
-characteristics

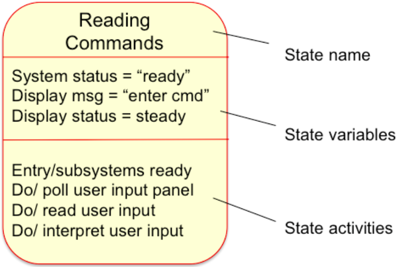
+identify()
+enable()
+disable()
+reconfigure()



Visual Paradigm screenshot

State Diagram





Example UML state diagram

Negotiating Requirements

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders “win conditions”
  - Win conditions are not always obvious
- Negotiate 비현실적인 제약(e.g. 말도 안되게 짧은 기간, 적은 비용)
  - Work toward a set of requirements that lead to “win-win”

Validating Requirements

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?

lab(se);

Validating Requirements

- Do any requirements conflict with other requirements?
- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
- Has requirement patterns been used to simplify the requirement model?
  - Has every pattern been validated?
  - Is every pattern consistent with the requirements of customer?

Specification Guidelines

- use a layered format that provides increasing detail as the “layers” deepen
- use consistent graphical notation and apply textual terms consistently (stay away from aliases)
- be sure to define all acronyms
- be sure to include a table of contents
  - ideally, include an index and/or a glossary
- write in a simple and unambiguous style
  - see “editing suggestions”
- always put yourself in the reader’s position
  - Would I be able to understand this if I wasn’t intimately familiar with the system?

Editing Suggestions

- Be on the lookout for persuasive connectors, ask why?
  - keys: certainly, therefore, clearly, obviously, it follows that ...
- Watch out for vague terms
  - keys: some, sometimes, often, usually,ordinarily, most, mostly
- When lists are given, but not completed, be sure all items are understood
  - keys: etc., and so forth, and so on, such as
- Be sure stated ranges don't contain unstated assumptions
  - e.g., Valid codes range from 10 to 100. Integer? Real? Hex?
- Beware of vague verbs such as handled, rejected, processed
- Beware "passive voice" statements
  - e.g., The parameters are initialized. By what?
- Beware "dangling" pronouns
  - e.g., The I/O module communicated with the data validation module and its control flag is set. Whose control flag?

Editing Suggestions

- When a term is explicitly defined in one place, try substituting the definition for other occurrences of the term
- When a structure is described in words, draw a picture
- When a structure is described with a picture, try to redraw the picture to emphasize different elements of the structure
- When symbolic equations are used, try expressing their meaning in words
- When a calculation is specified, work at least two examples
- Look for statements that imply certainty, then ask for proof
  - keys; always, every, all, none, never
- Search behind certainty statements & be sure restrictions or limitations are realistic