๒๒๕๐

SOFTWARE ENGINEERING

Week 5

Requirement Engineering

Course ID 06016410,
06016321

Nont Kanungsukkasem, B.Eng., M.Sc., Ph.D.
nont@it.kmitl.ac.th

Requirement Engineering

Quick Look

What is it? Before you begin any technical work, it's a good idea to create a set of requirements for the engineering tasks. By establishing a set of requirements, you'll gain an understanding of what the business impact of the software will be, what the customer wants, and how end users will interact with the software.

Who does it? Software engineers and other project stakeholders (managers, customers, and end users) all participate in requirements engineering.

Why is it important? To understand what the customer wants before you begin to design and build a computer-based system. Building an elegant computer program that solves the wrong problem helps no one.

What are the steps? Requirements engineering begins with inception (a task that defines the scope and nature of the problem to be solved). It moves onward to elicitation (a task that helps stakeholders define what is required), and then elaboration (where basic requirements are refined and modified). As stakeholders define the problem, negotiation occurs (what are the priorities, what is essential, when is it required?). Finally, the problem is specified in some manner and then reviewed or validated to ensure that your understanding of the problem and the stakeholders' understanding of the problem coincide.

What is the work product? Requirements engineering provides all parties with a written understanding of the problem. The work products may include: usage scenarios, function and feature lists, and requirements models.

How do I ensure that I've done it right? Requirements engineering work products are reviewed with stakeholders to ensure that everyone is on the same page. A word of warning: Even after all parties agree, things will change, and they will continue to change throughout the project.

Requirements Engineering

- Requirements engineering is the term for the broad spectrum of tasks and techniques that lead to an understanding of requirements.
- From a software process perspective, requirements engineering is a major software engineering action that begins during the communication activity and continues into the modeling activity.
- Requirements engineering establishes a solid base for design and construction.
- Without it, the resulting software has a high probability of not meeting customer's needs.
- It must be adapted to the needs of the process, the project, the product, and the people doing the work.

Types of Requirements

FURPS+

- Functional requirements (F)
- Non-functional requirements (URPS+)

Functional Requirements

F

- Functional Requirements (aka. behavioral requirement): the activities the system must perform
  - Functionality - What the customer wants! Note that this includes security-related needs.
  - e.g., Feature sets, capabilities, (security)

Non-Functional Requirements **URPS+**

- Non-Functional Requirements: other system characteristics, esp. quality attributes
  - Usability
    - e.g., Human factors, UI format, aesthetics, consistency in the UI, online and context-sensitive help, wizards and agents, user documentation, training materials, and easy to use,
  - Reliability
    - e.g., Availability (failure rate), Failure Extent & time-length (mean time between failure - MTBF, recovery method, error handling), predictability (stability), accuracy,
  - Performance
    - e.g., Response time, throughput, speed, resource consumption, scalability, efficiency, capability

Non-Functional Requirements

- Supportability
  - e.g., ability of technical support to install, configure, and monitor computer products, identify and debug faults, provide maintenance, and restore the product into service.
- +
  - Design constraints, e.g., HW and SW support
  - Implementation, e.g., Development tools and protocols
  - Interface, e.g., Data interchange format
  - Physical, e.g., Size, weight, power consumption
  - portability, compatibility, regulatory, manageability, environmental, data integrity and interoperability

Requirements Engineering Tasks

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

Inception

- Most projects begin with an identified business need or when a potential new market or service is discovered.
- At project inception, you establish
  - a basic understanding of the problem,
  - the people who want a solution, and
  - the nature of the solution that is desired
- Communication between all stakeholders and the software team needs to be established during this task to begin an effective collaboration.

Elicitation

- An important part of elicitation is to understand the business goals
- Goals should be specified precisely and serve as the basis for requirements elaboration, verification and validation, conflict management, negotiation, explanation, and evolution.
- Once the goals are captured, you establish a prioritization mechanism and create a design rationale for a potential architecture
- The intent of elicitation is to transfer ideas from stakeholders to the software team smoothly and without delay.

Elaboration

- The elaboration task focuses on developing a refined requirements model that identifies various aspects of software function, behavior, and information
- Elaboration is driven by the creation and refinement of user scenarios obtained during elicitation.
- These scenarios describe how the end users (and other actors) will interact with the system.
- The key is to describe the problem in a way that establishes a firm base for design and then move on.
- Do not obsess over unnecessary details

Negotiation

- Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority.
- There should be no winner and no loser in an effective negotiation. Both sides win, because a “deal” that both can live with is solidified
- You should use an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts

Specification

- A specification can be one (or a combination) of the following:
  - a written document
  - a set of graphical models
  - a formal mathematical model
  - a collection of usage scenarios
  - a prototype

https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc

Validation

- Requirements validation examines the specification to ensure
  - that all software requirements have been stated unambiguously;
  - that inconsistencies, omissions, and errors have been detected and corrected; and
  - that the work products conform to the standards established for the process, the project, and the product.
- Formal technical review mechanism 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.

Validation: example

- The software should be user friendly.
- The probability of a successful unauthorized database intrusion should be less than 0.0001.

Requirements Management

- Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system.
- Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.
- Many of these activities are identical to the software configuration management (SCM) techniques

1 Establishing the Groundwork

1) Identifying Stakeholders

- A stakeholder is anyone who benefits in a direct or indirect way from the system which is being developed.
- At inception, you should create a list of people who will contribute input as requirements are elicited
- The initial list will grow as stakeholders are contacted because every stakeholder will be asked:
  - "Whom else do you think I should talk to?"

2) Recognizing Multiple Viewpoints

- Because many different stakeholders exist, the requirements of the system will be explored from many different points of view.
- As information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.
- You should categorize all stakeholder information (including inconsistent and conflicting requirements) in a way that will allow decision makers to choose an internally consistent set of requirements for the system.

3) Working Toward Collaboration

- The job of a requirements engineer is to identify:
  - areas of commonality (i.e., requirements on which all stakeholders agree)
  - areas of conflict or inconsistency (i.e., requirements that are desired by one stakeholder but conflict with the needs of another stakeholder)

4) Asking the First Questions

- Questions asked at the inception of the project should be “context free.”
- The first set of context-free questions focuses on the customer and other stakeholders and the overall project goals and benefits. 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?

4) Asking the First Questions

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

4) Asking the First Questions

- The final set of questions focuses on the effectiveness of the communication activity itself.
  - 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?
- The Q&A session should be used for the first encounter only and then replaced by a requirements elicitation format that combines elements of problem solving, negotiation, and specification.

5) Nonfunctional Requirements

- Develop a list of NFRs
  - A simple table lists NFRs as column labels and software engineering guidelines as row labels.
  - A relationship matrix compares each guideline to all others, helping the team to assess whether each pair of guidelines is complementary, overlapping, conflicting, or independent.
- The team prioritizes each nonfunctional requirement by creating a homogeneous set of nonfunctional requirements using a set of decision rules that establish which guidelines to implement and which to reject.

6) Traceability

- Traceability is a software engineering term that refers to documented links between software engineering work products (e.g., requirements and test cases).
- A traceability matrix allows a requirements engineer to represent the relationship between requirements and other software engineering work products.

Requirement Traceability Matrix

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run

Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

Sheet1 Sheet2 Sheet3

2 Requirements Gathering

Collaborative Requirements Gathering

- Meetings (either real or virtual) 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 worksheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

Goal

- identify the problem
- propose elements of the solution
- negotiate different approaches
- specify a preliminary set of solution requirements.

Product Request

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. We might also consider using voice control of the system using some technology like Alexa.

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, can be programmed by the homeowner, and will automatically contact a monitoring agency and the owner's cell phone when a situation is detected.

Collaborative Requirements Gathering

- At this stage, critique and debate are strictly prohibited.
- Avoid the impulse to shoot down a customer's idea as "too costly" or "impractical."
- The idea here is to negotiate a list that is acceptable to all.
  - To do this, you must keep an open mind.
- The objective is to develop a consensus list of objects, services, constraints, and performance for the system to be built
- During all discussions, the team may raise an issue that cannot be resolved during the meeting

Basic Concerns

- Can we build the system?
- Will this development process allow us to beat our competitors to market?
- Do adequate resources exist to build and maintain the proposed system?
- Will the system performance meet the needs of our customers?

Usage Scenarios

- The scenarios, often called use cases, provide a description of how the system will be used.

Elicitation Work Products

- For large systems, the work products may 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.

3 Developing Use Cases

Use Case

- A use case tells a stylized story about how an end user (playing one of several possible roles) interacts with the system under a specific set of circumstances.
- The story may be narrative text (a user story), an outline of tasks or interactions, a template-based description, or a diagrammatic representation.
- Regardless of its form, a use case depicts the software or system from the end user's point of view.

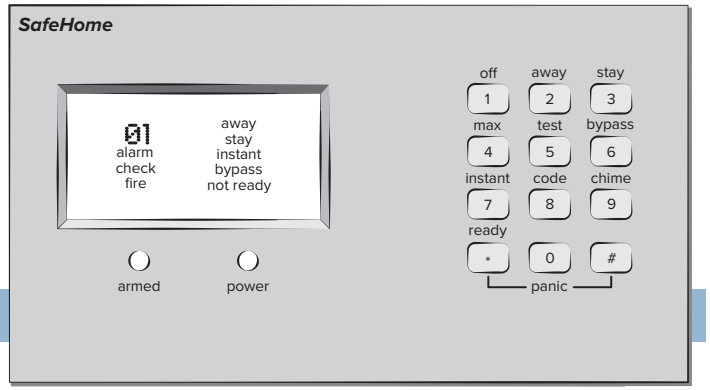
Actors

- The first step in writing a use case is to define the set of "actors"
- An actor is anything that communicates with the system or product.
- An actor and an end user are not necessarily the same thing.
- A typical user may play several different roles when using a system, whereas an actor represents a class of external entities (often, but not always, people) that play just one role in the context of the use case.
- Primary actors interact to achieve required system function and derive the intended benefit from the system. They work directly and frequently with the software.
- Secondary actors support the system so that primary actors can do their work.

Developing Use Cases

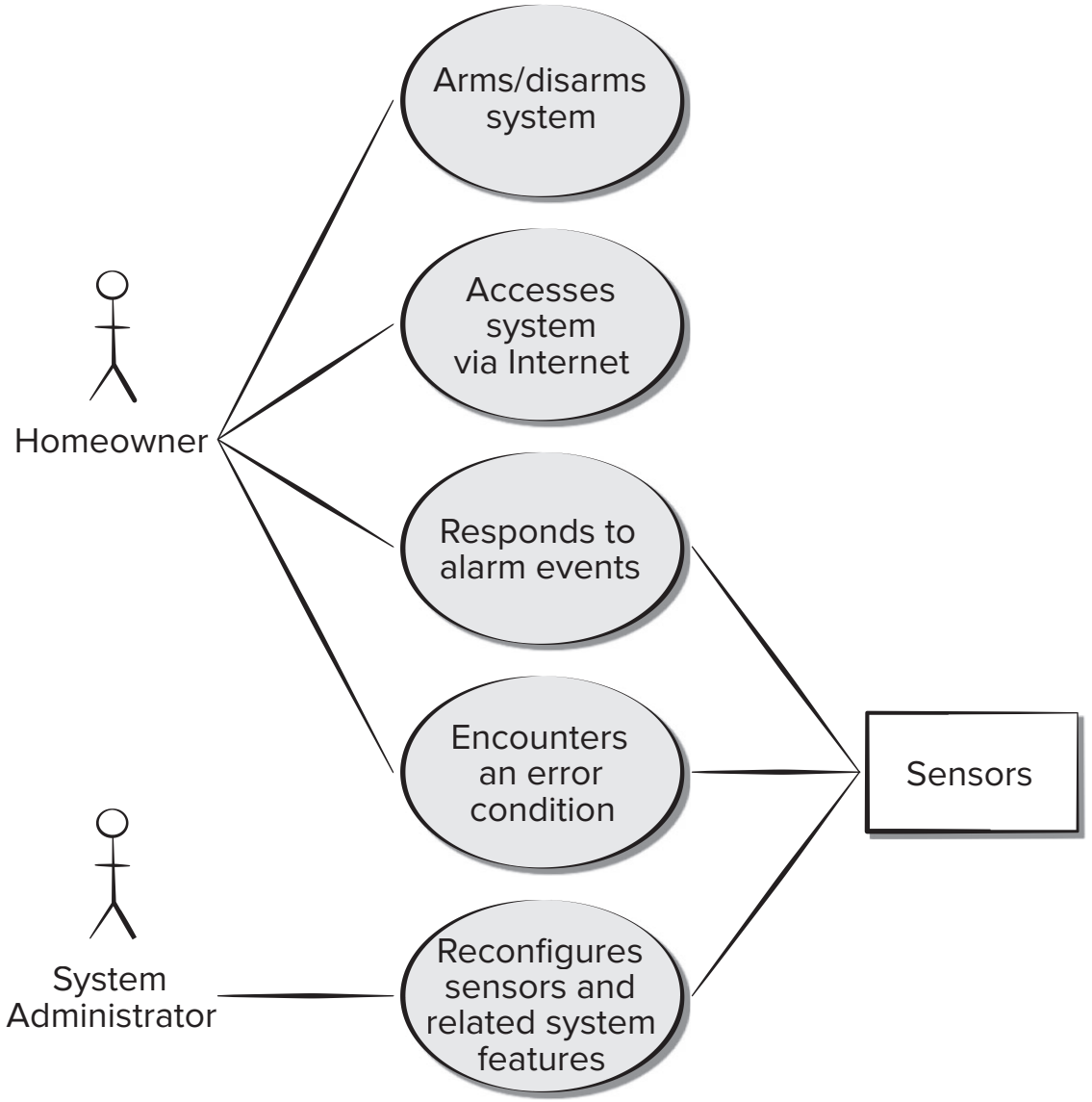
- Once actors have been identified, use cases can be developed by answering these questions:
  - Who is the primary actor, the secondary actor(s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What exceptions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

Example: basic use case



- 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 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: use case diagram



Example: detailed descriptions of 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, that is, to turn on the alarm functions.

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

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:

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: 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.

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?

4 Building the Analysis Model

Building the Analysis Model

- The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system.
- The analysis model is a snapshot of requirements at any given time. You should expect it to change.

Elements of the Analysis Model

- Scenario-Based Elements
  - Use case
- Class-Based Elements.
  - Class diagram
  - CRC modeling
- Behavioral Elements.
  - Identifying events with the use case
  - State diagram
  - Activity diagram
- Functional Elements
  - Procedural view
  - Sequence diagram

Analysis Patterns

- Analysis patterns suggest solutions (e.g., a class, a function, a behavior) within the application domain that can be reused when modeling many applications.
- Analysis patterns are integrated into the analysis model by reference to the pattern name.
- They are also stored in a repository so that requirements engineers can use search facilities to find and reuse them.

5 Negotiating Requirements

Negotiating Requirements

- The intent of these negotiations is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team.
- 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
  - Work toward a set of requirements that lead to "win-win"

Win-Win

- Win-Win
  - stakeholders win by getting the system or product that satisfies most their needs, and
  - you (as a member of the software team) win by working to realistic and achievable budgets and deadlines
- Handshaking
  - The software team proposes solutions to requirements, describes their impact, and communicates their intentions to customer representatives.
  - The customer representatives review the proposed solutions, focusing on missing features and seeking clarification of novel requirements.
  - Requirements are determined to be good enough if the customers accept the proposed solution.

6 Requirements Monitoring

Requirements Monitoring

- Requirements monitoring can be extremely useful when incremental development is used.
- It encompasses five tasks:
  1. distributed debugging uncovers errors and determines their cause
  2. run-time verification determines whether software matches its specification
  3. run-time validation assesses whether the evolving software meets user goals
  4. business activity monitoring evaluates whether a system satisfies business goals
  5. evolution and co-design provides information to stakeholders as the system evolves.

7 Validating Requirements

Validating Requirements

- A review of the requirements model addresses the following questions:
  - Is each requirement consistent with the overall objectives for the system or 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?
  - 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?
  - Have requirements patterns been used to simplify the requirements model? Have all patterns been properly validated? Are all patterns consistent with customer requirements?