2019 级本科

软件工程

Software Engineering

张昕

zhangxin@cust.edu.cn

计算机科学技术学院软件工程系

Understanding Requirements

Why are requirements important

- Before you begin any technical work, it's a good idea to create a set of requirements for any engineering tasks.
  - These tasks lead to 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.
- Designing and building an elegant computer program that solves the wrong problem serves no one's needs. That's why it's important to understand what the customer wants before you begin to design and build a computer-based system.
  - 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.

Why are requirements important

Work product

- The intent of requirements engineering is to provide all parties with a written understanding of the problem. This can be achieved though a number of work products: usage scenarios, functions and features lists, requirements models, or a specification.

Ensure correctness

- Requirements engineering work products are reviewed with stakeholders to ensure that what you have learned is what they really meant.
- A word of warning: Even after all parties agree, things will change, and they will continue to change throughout the project.

Requirements engineering

- Designing and building computer software is challenging, creative, and just plain fun. In fact, building software is so compelling that many software developers want to jump right in before they have a clear understanding of what is needed.
- The broad spectrum of tasks and techniques that lead to an understanding of requirements is called requirements engineering.
  - 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.
  - It must be adapted to the needs of the process, the project, the product, and the people doing the work.
  - Requirements engineering builds a bridge to design and construction.
- Requirements engineering builds a bridge to design and construction.

Requirements engineering

- Requirements engineering encompasses seven distinct tasks:
  - inception,
  - elicitation,
  - elaboration,
  - negotiation,
  - specification,
  - validation, and
  - management.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Inception
    - How does a software project get started? Is there a single event that becomes the catalyst for a new computer-based system or product, or does the need evolve over time?
      - In some cases, a casual conversation is all that is needed to precipitate a major software engineering effort.
      - Most projects begin when a business need is identifi ed or a potential new market or service is discovered. Stakeholders from the business community (e.g., business managers, marketing people, product managers) defi ne a business case for the idea, try to identify the breadth and depth of the market, do a rough feasibility analysis, and identify a working description of the project's scope.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Dlicitation
    - An important part of elicitation is to establish business goals.
    - Your job is to engage stakeholders and to encourage them to share their goals honestly.
      - Once the goals have been captured, a prioritization mechanism should be established, and a design rationale for a potential architecture (that meets stakeholder goals) can be created.
    - Key problems
      - Problems of scope occur when the boundary of the system is ill-defined or the customers and users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.
      - Problems of understanding are encountered when customers and users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs, omit information that is believed to be "obvious," specify requirements that conflict with the needs of other customers and users, or specify requirements that are ambiguous or untestable.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Dlicitation
    - An important part of elicitation is to establish business goals.
    - Your job is to engage stakeholders and to encourage them to share their goals honestly.
      - Once the goals have been captured, a prioritization mechanism should be established, and a design rationale for a potential architecture (that meets stakeholder goals) can be created.
    - Key problems [cont.]
      - Problems of volatility occur when the requirements change over time. To help overcome these problems, you must approach the requirements-gathering activity in an organized manner.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Elaboration
    - The information obtained from the customer during inception and elicitation is expanded and refi ned during elaboration.
    - This 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 that describe how the end user (and other actors) will interact with the system.
      - Each user scenario is parsed to extract analysis classes—business domain entities that are visible to the end user.
      - The attributes of each analysis class are defined, and the services that are required by each class are identified.
      - The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Negotiation
    - It isn't unusual for customers and users to ask for more than can be achieved, given limited business resources. It's also relatively common for different customers or users to propose conflicting requirements, arguing that their version is "essential for our special needs."
    - Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority.
    - Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Specification
    - In the context of computer-based systems (and software), the term specification means different things to different people.
    - A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.
    - For large systems, a written document, combining natural language descriptions and graphical models may be the best approach. However, usage scenarios may be all that are required for smaller products or systems that reside within well-understood technical environments.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Validation
    - The work products produced as a consequence of requirements engineering are assessed for quality during a validation step.
    - 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.
    - The primary requirements validation mechanism is the technical review.
      - The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking 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), confl icting requirements, or unrealistic (unachievable) requirements.

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Validation [cont.]
    - To illustrate some of the problems that occur during requirements validation, consider two seemingly innocuous requirements:
      - The software should be user friendly.
      - The probability of a successful unauthorized database intrusion should be less than 0.0001.
        - The first requirement is too vague for developers to test or assess. What exactly does "user friendly" mean? To validate it, it must be quantified or qualified in some manner.
        - The second requirement has a quantitative element ("less than 0.0001"), but intrusion testing will be difficult and time consuming. Is this level of security even warranted for the application? Can other complementary requirements associated with security (e.g., password protection, specialized handshaking) replace the quantitative requirement noted?

Requirements engineering

- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
  - Validation [cont.]

Requirements Validation Checklist

① Are requirements stated clearly?
② Can they be misinterpreted?
③ Is the source (e.g., a person, a regulation, a document) of the requirement identified?
④ Has the final statement of the requirement been examined by or against the original source?
⑤ Is the requirement bounded in quantitative terms?
⑥ What other requirements relate to this requirement?
⑦ Are they clearly noted via a cross-reference matrix or other mechanism?
⑧ Does the requirement violate any system domain constraints?
⑨ Is the requirement testable?
⑩ If so, can we specify tests (sometimes called validation criteria) to exercise the requirement?
⑪ Is the requirement traceable to any system model that has been created?
⑫ Is the requirement traceable to overall system/ product objectives?
⑬ Is the specification structured in a way that leads to easy understanding, easy reference, and easy translation into more technical work products?
⑭ Has an index for the specification been created?
⑮ Have requirements associated with performance, behavior, and operational characteristics been clearly stated?
⑯ What requirements appear to be implicit?

Key Issues

Identifying Stakeholders

- Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail.
  - 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?"

A stakeholder is anyone who has a direct interest in or benefits from the system that is to be developed.

Key Issues

Recognizing Multiple Viewpoints

- 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, making the new system easy to sell.
  - Business managers are interested in a feature set that can be built within budget and that will be ready to meet defined market windows.
  - End users may want features that are familiar to them and that are easy to learn and use.
  - Software engineers may be concerned with functions that are invisible to nontechnical stakeholders but that enable an infrastructure that supports more marketable functions and features.
  - Support engineers may focus on the maintainability of the software.
- Each of these constituencies (and others) will contribute information to the requirements engineering process. 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 confl icting requirements) in a way that will allow decision makers to choose an internally consistent set of requirements for the system.
- The goal of effective requirements engineering is to eliminate or at least reduce these problems.

Key Issues

Working toward Collaboration

- Customers (and other stakeholders) should collaborate among themselves (avoiding petty turf battles) and with software engineering practitioners if a successful system is to result.
  - If five stakeholders are involved in a software project, you may have five (or more) different opinions about the proper set of requirements.
- The job of a requirements engineer is to identify areas of commonality (i.e., requirements on which all stakeholders agree) and areas of confl ict or inconsistency (i.e., requirements that are desired by one stakeholder but confl ict with the needs of another stakeholder).
  - In many cases, stakeholders collaborate by providing their view of requirements, but a strong "project champion" (e.g., a business manager or a senior technologist) may make the fi nal decision about which requirements make the cut.

Key Issues

Asking Questions

- First phase: context-free questions that focus on the customer and other stakeholders, the overall project goals and benefits. The questions help to identify all stakeholders who will have interest in the software to be built and to identify the measurable benefit of a successful implementation and possible alternatives to custom software development.
  - 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?
- Second phase: enables 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?

Key Issues

Asking Questions [cont.]

- Final phase: 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?

These questions (and others) will help to "break the ice" and initiate the communication that is essential to successful elicitation.

Key Issues

Nonfunctional Requirements

- A nonfunctional requirement (NFR) can be described as a quality attribute, a performance attribute, a security attribute, or a general constraint on a system.
- There is a lopsided emphasis on functionality of the software, yet the software may not be useful or usable without the necessary non-functional characteristics.
- Nonfunctional requirements are often listed separately in a software requirements specification.
- Two-phase approach of identifying non-functional requirements
  - First phase
    - A set of software engineering guidelines is established for the system to be built. These include guidelines for best practice, but also address architectural style and the use of design patterns.
    - A list of NFRs (e.g., requirements that address usability, testability, security or maintainability) is then developed. 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 , confl icting , or independent.

Key Issues

Nonfunctional Requirements

- A nonfunctional requirement (NFR) can be described as a quality attribute, a performance attribute, a security attribute, or a general constraint on a system.
- There is a lopsided emphasis on functionality of the software, yet the software may not be useful or usable without the necessary non-functional characteristics.
- Nonfunctional requirements are often listed separately in a software requirements specification.
- Two-phase approach of identifying non-functional requirements [cont.]
  - Second phase
    - 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.

Key Issues

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.
  - Rows of the traceability matrix are labeled using requirement names and columns can be labeled with the name of a software engineering work product (e.g., a design element or a test case).
  - A matrix cell is marked to indicate the presence of a link between the two.
  - The traceability matrices can support a variety of engineering development activities. They can provide continuity for developers as a project moves from one project phase to another, regardless of the process model being used.
  - Traceability matrices often can be used to ensure the engineering work products have taken all requirements into account.
- As the number of requirements and the number of work products grows, it becomes increasingly difficult to keep the traceability matrix up to date.
  - Nonetheless, it is important to create some means for tracking the impact and evolution of the product requirements

Eliciting Requirements

- Requirements elicitation (also called requirements gathering) combines elements of problem solving, elaboration, negotiation, and specification.
- 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 fl ow 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 goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements.

Eliciting Requirements

- Many stakeholder concerns (e.g., accuracy, data accessibility, security) are the basis for nonfunctional system requirements. As stakeholders enunciate these concerns, software engineers must consider them within the context of the system to be built. Among the questions that must be answered are as follows:
  - 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?

The answers to these and other questions will evolve over time.

Eliciting Requirements

Usage Scenarios

- As requirements are gathered, an overall vision of system functions and features begin to materialize.
  - However, it is diffi cult 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 that provide a description of how the system will be used.

Eliciting Requirements

Elicitation Work Products

- The work products produced as a consequence of requirements elicitation will vary depending on the size of the system or product to be built.
- For most systems, the work products include:

  (1) a statement of need and feasibility,

  (2) a bounded statement of scope for the system or product,

  (3) a list of customers, users, and other stakeholders who participated in requirements elicitation,

  (4) a description of the system's technical environment,

  (5) a list of requirements (preferably organized by function) and the domain constraints that applies to each,

  (6) a set of usage scenarios that provide insight into the use of the system or product under different operating conditions, and

  (7) any prototypes developed to better define requirements.

Each of these work products is reviewed by all people who have participated in requirements elicitation.

Eliciting Requirements

Agile Requirements Elicitation

- Within the context of an agile process, requirements are elicited by asking all stakeholders to create user stories .
- Each user story describes a simple system requirement written from the user's perspective.
  - User stories can be written on small note cards, making it easy for developers to select and manage a subset of requirements to implement for the next product increment.
  - Proponents claim that using note cards written in the user's own language allows developers to shift their focus to communication with stakeholders on the selected requirements rather than their own agenda.
- Although the agile approach to requirements elicitation is attractive for many software teams, critics argue that a consideration of overall business goals and nonfunctional requirements is often lacking.
  - In some cases, rework is required to accommodate performance and security issues.
  - In addition, user stories may not provide a sufficient basis for system evolution over time

Eliciting Requirements

Service-Oriented Methods

- Service-oriented development views a system as an aggregation of services.
  - A service can be "as simple as providing a single function, for example, a request/response-based mechanism that provides a series of random numbers, or can be an aggregation of complex elements, such as the Web service API".
- Requirements elicitation in service-oriented development focuses on the defi nition of services to be rendered by an application.
  - As a metaphor, consider the service provided when you visit a fine hotel.
  - A doorperson greets guests. A valet parks their cars.
  - The desk clerk checks the guests in.
  - A bellhop manages the bags. The concierge assists guest with local arrangements.
  - Each contact or touchpoint between a guest and a hotel employee is designed to enhance the hotel visit and represents a service offered.

Eliciting Requirements

Service-Oriented Methods [cont.]

- Techniques for eliciting requirements must also acquire information about the brand and the stakeholders' perceptions of it.
  - In addition to studying how the brand is used by customers, analysts need strategies to discover and document requirements about the desired qualities of new user experiences.
  - User stories are helpful in this regard.
- The requirements for touchpoints should be characterized in a manner that indicates achievement of the overall service requirements. This suggests that each requirement should be traceable to a specific service.

Developing Use Cases

- A use case captures a contract . . . [that] describes the system's behavior under various conditions as the system responds to a request from one of its stakeholders . . .
  - A use case tells a stylized story about how an end user (playing one of a number of possible roles) interacts with the system under a specific set of circumstances.
    - The story may be narrative text, 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.

Developing Use Cases

Procedure of writing a use case

- The first step in writing a use case is to define the set of "actors" that will be involved in the story.
  - Actors are the different people (or devices) that use the system or product within the context of the function and behavior that is to be described.
  - Actors represent the roles that people (or devices) play as the system operates.
  - Defined somewhat more formally, an actor is anything that communicates with the system or product and that is external to the system itself.
  - Every actor has one or more goals when using the system.
- 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 external entities (often, but not always, people) that play just one role in the context of the use case.

Developing Use Cases

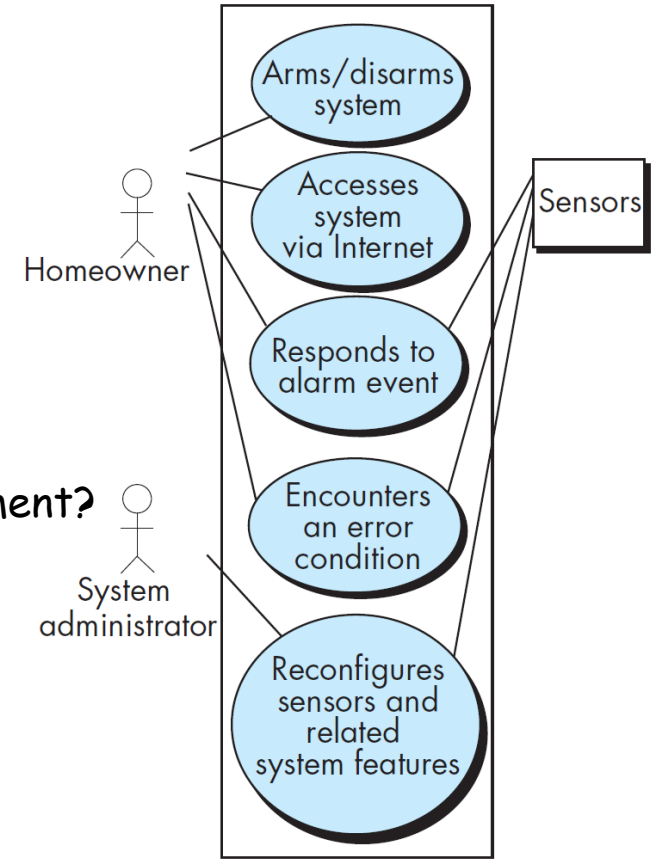
Procedure of writing a use case

- Because requirements elicitation is an evolutionary activity, not all actors are identified during the first iteration. It is possible to identify primary actors during the first iteration and secondary actors as more is learned about the system.
  - 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

Procedure of writing a use case

- Once actors have been identified, use cases can be developed. Jacobson suggests a number of questions that should be answered by a use case:.
  - 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?



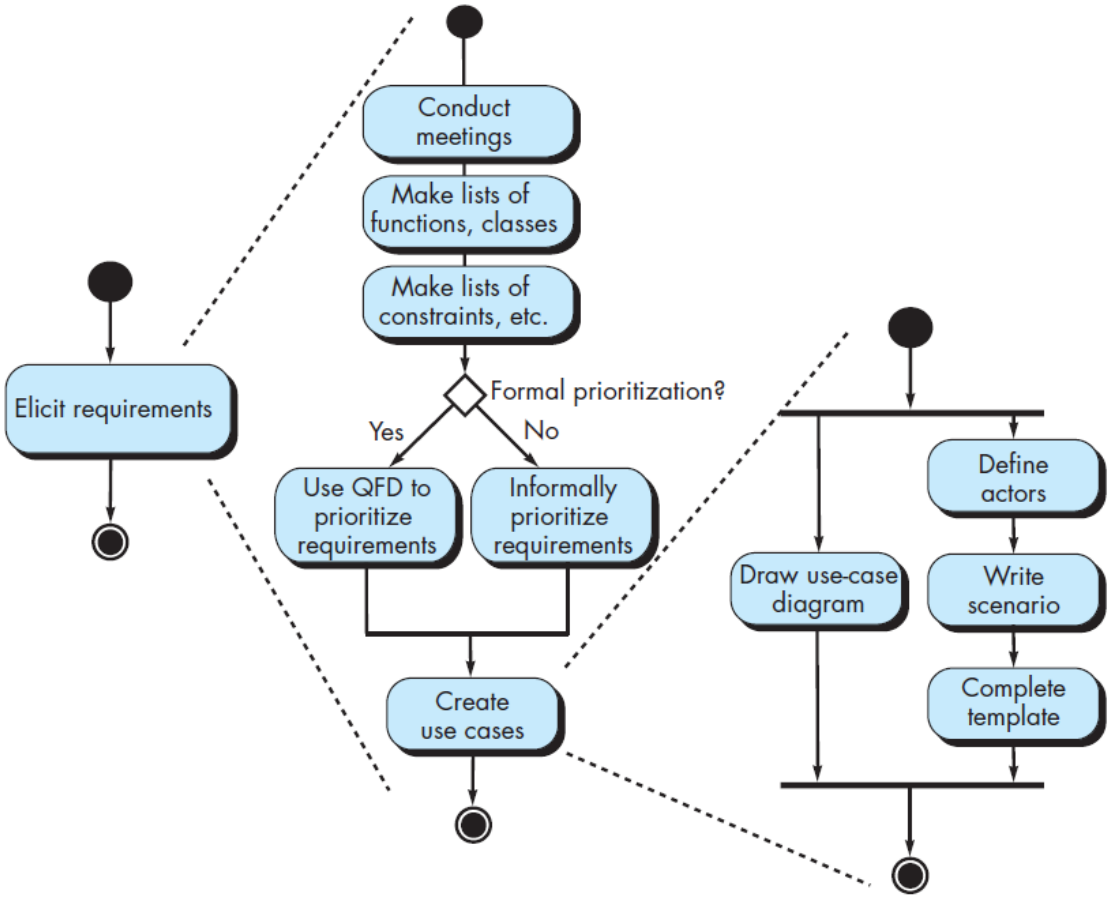
Building The Analysis Model

- The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system.
- As the analysis model evolves, certain elements will become relatively stable, providing a solid foundation for the design tasks that follow.

Building The Analysis Model

Elements of the Analysis Model

- There are many different ways to look at the requirements for a computer-based system.

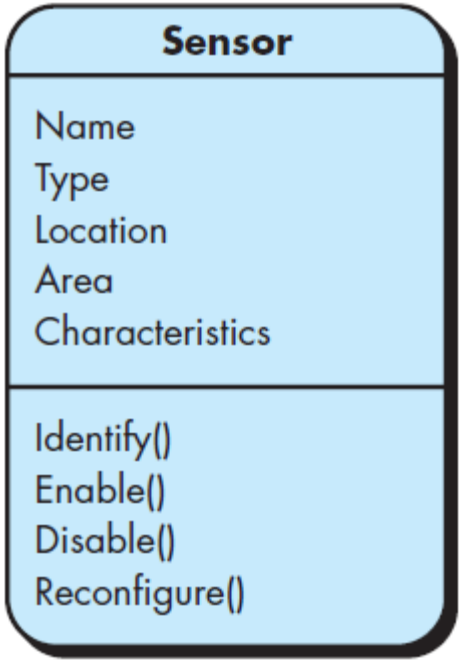


- Scenario-based elements
  - The system is described from the user's point of view using a scenario-based approach.
  - For example, basic use cases and their corresponding use case diagrams evolve into more elaborate template-based use cases.
- Scenario-based elements of the requirements model are often the first part of the model that is developed. As such, they serve as input for the creation of other modeling elements.

Building The Analysis Model

Elements of the Analysis Model

- There are many different ways to look at the requirements for a computer-based system.

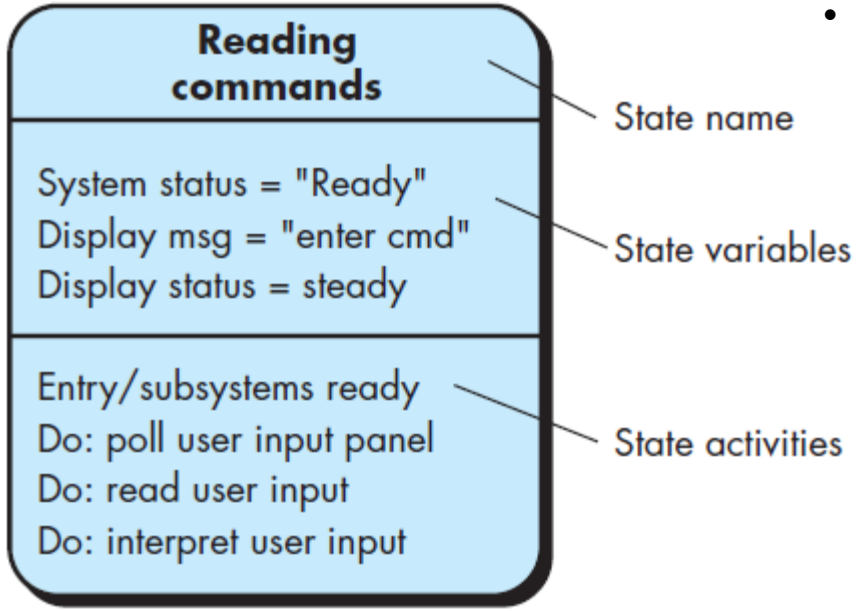


- Class-based elements
  - Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system. These objects are categorized into classes—a collection of things that have similar attributes and common behaviors.
  - Note that the diagram lists the attributes of sensors (e.g., name, type) and the operations (e.g., identify, enable) that can be applied to modify these attributes. In addition to class diagrams, other analysis modeling elements depict the manner in which classes collaborate with one another and the relationships and interactions between classes.

Building The Analysis Model

Elements of the Analysis Model

- There are many different ways to look at the requirements for a computer-based system.



- Behavioral elements
  - The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied. Therefore, the requirements model must provide modeling elements that depict behavior.
  - The state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state.
    - A state is any observable mode of behavior. In addition, the state diagram indicates what actions (e.g., process activation) are taken as a consequence of a particular event.

Validating Requirements

- The requirements represented by the model are prioritized by stakeholders and grouped within requirements packages that will be implemented as software increments.
- 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 specifi ed 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 confl ict with other requirements?

Validating Requirements

- The requirements represented by the model are prioritized by stakeholders and grouped within requirements packages that will be implemented as software increments.
- A review of the requirements model addresses the following questions: [cont.]
  - 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 refl ect 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?


CHANGCHUN UNIVERSITY OF SCIENCE AND TECHNOLOGY
1958