SWE209
Object Oriented Analysis and Design

Requirements Elicitation

Note

- This presentation is based on the slides and content of the course main textbook.
- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014
- https://ase.in.tum.de/lehrstuhl_1/component/content/article/43-books/217

Agenda

Big Picture

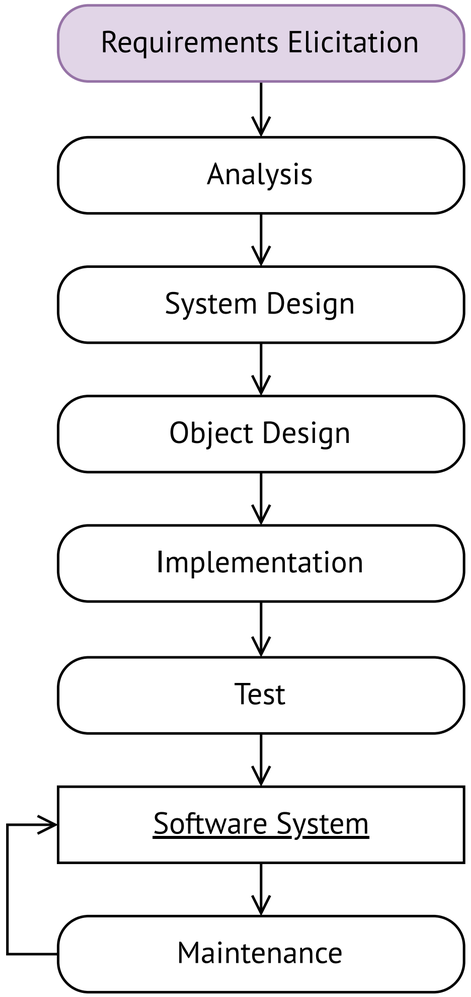
Introduction

Requirements Elicitation Concepts

Requirements Elicitation Activities

Documenting Requirements Elicitation

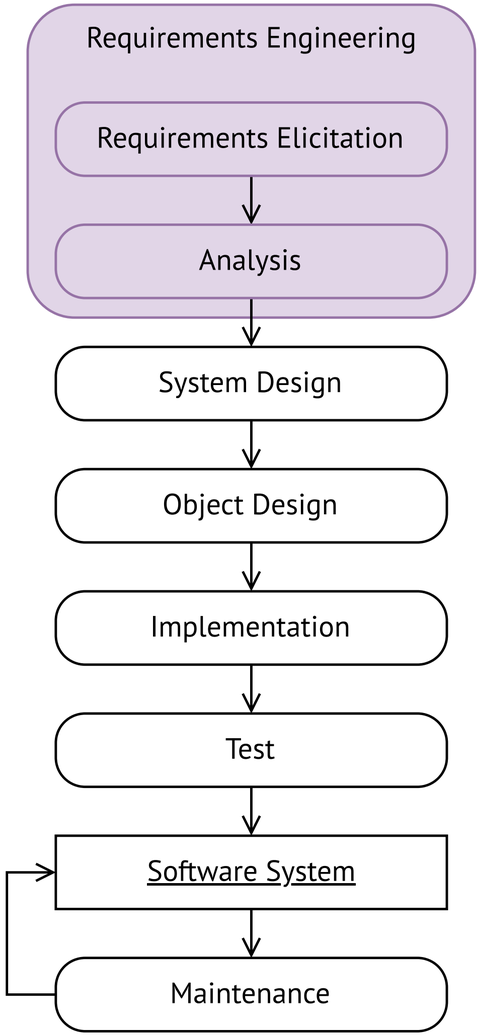
Big Picture


Requirements Elicitation
Analysis
System Design
Object Design
Implementation
Test
Software System
Maintenance

Introduction

- Requirement
  - A feature that the system must have or a constraint that it must satisfy to be accepted by the client.
- Requirements engineering
  - Defining the requirements of the system under construction.
- Requirements engineering → Requirements Elicitation + Analysis
- Requirements elicitation and analysis occur concurrently and iteratively.

Introduction



Introduction

- Requirements elicitation
  - Results in the specification of the system that the client understands.
- Analysis
  - Results in an analysis model that the developers can unambiguously interpret.
- Requirements elicitation is challenging.
  - Requires the collaboration.

Introduction

- Requirements elicitation
  - Describe the purpose of the system.
  - Identify a problem area and define a system that addresses the problem.
- Output of requirement elicitation → Requirements specification
  - A contract between the client and the developers.
  - Written in natural language.
  - Represent accurately the external aspects of the system.

Introduction

Scenario-Based Requirements Elicitation

- Scenario-based requirements elicitation
- Sources of information
  - Observation of people (users of the system, clients, related individuals).
  - Interviewing of people. (users of the system, clients, related individuals).
  - Reading documents, manuals, etc. related to the application domain.

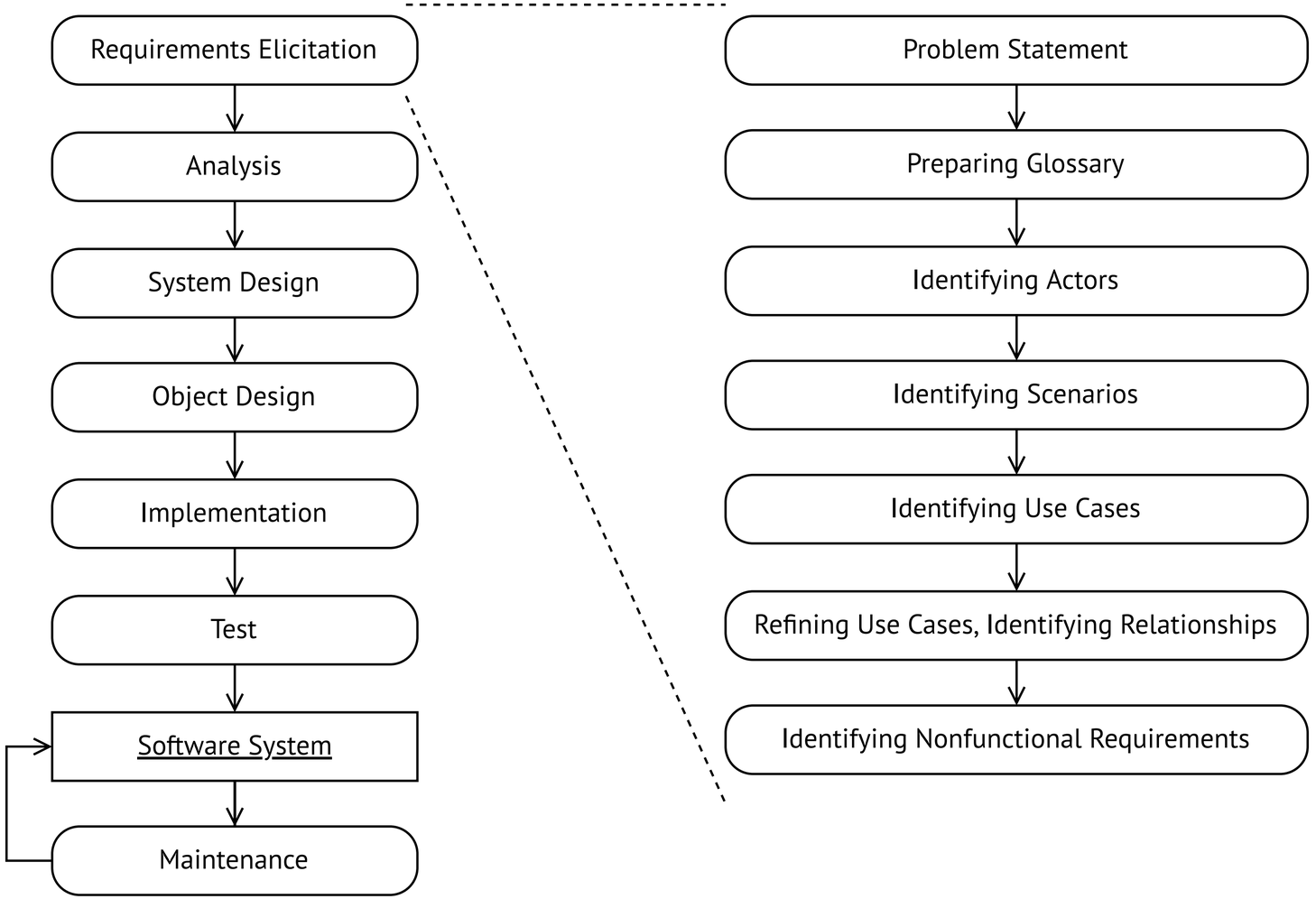
Introduction

Scenario-Based Requirements Elicitation

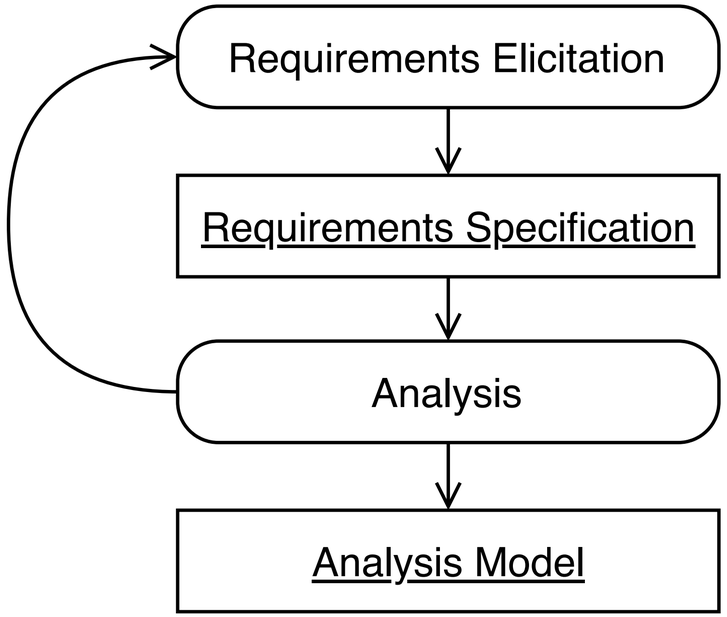
- Represent the user's current work processes as as-is scenarios.
- Develop visionary scenarios describing the functionality to be provided by the future system.
- Validate the system description by reviewing the scenarios and by testing small prototypes.
- Agree on a requirements specification.

Introduction

Requirements Elicitation Activities



Introduction



Introduction

- Requirement elicitation → Focus only on the user's view of the system.
- Example requirements
  - System functionality
  - Interaction between the user and the system
  - Errors that the system can detect and handle
  - Environmental conditions in which the system functions

Introduction

- Example non requirements
  - System structure
  - Implementation technology selected to build the system
  - The system design
  - The development methodology
  - Other aspects not directly visible to the user.

Requirements Elicitation Concepts

Requirements Elicitation Concepts

- Functional Requirements
- Nonfunctional Requirements
- Completeness, Consistency, Clarity, Correctness
- Realism, Verifiability, Traceability
- Greenfield Engineering, Reengineering, Interface Engineering

Requirements Elicitation Concepts

Functional Requirements

- Functional requirements
  - Describe the interactions between the system and its environment.
- Environment
  - The user and any other external system with which the system interacts.

Requirements Elicitation Concepts

Functional Requirements - Sample

`SatWatch` is a wrist watch that displays the time based on its current location. `SatWatch` uses GPS satellites (Global Positioning System) to determine its location and internal data structures to convert this location into a time zone.

The information stored in `SatWatch` and its accuracy measuring time is such that the watch owner never needs to reset the time. `SatWatch` adjusts the time and date displayed as the watch owner crosses time zones and political boundaries. For this reason, `SatWatch` has no buttons or controls available to the user.

`SatWatch` determines its location using GPS satellites and, as such, suffers from the same limitations as all other GPS devices (e.g., inability to determine location at certain times of the day in mountainous regions). During blackout periods, `SatWatch` assumes that it does not cross a time zone or a political boundary. `SatWatch` corrects its time zone as soon as a blackout period ends.

`SatWatch` has a two-line display showing, on the top line, the time (hour, minute, second, time zone) and on the bottom line, the date (day, date, month, year). The display technology used is such that the watch owner can see the time and date even under poor light conditions.

When political boundaries change, the watch owner may upgrade the software of the watch using the WebifyWatch device (provided with the watch) and a personal computer connected to the Internet.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Concepts

Nonfunctional Requirements

- Nonfunctional requirements
  - Describe aspects of the system that are not directly related to the functional behavior of the system.
  - Include a broad variety of requirements.
  - Two categories:
    - Quality Requirements
    - Constraints (Pseudo Requirements)

Requirements Elicitation Concepts

Nonfunctional Requirements

- Quality Requirements
  - Usability
  - Dependability
    - Reliability, robustness, safety.
  - Performance
    - Response time, throughput, availability.
  - Maintainability
  - Portability

Requirements Elicitation Concepts

Nonfunctional Requirements

- Constraints (Pseudo Requirements)
  - Implementation requirements
  - Interface requirements
  - Operations requirements
  - Packaging requirements
  - Legal requirements

Requirements Elicitation Concepts

Completeness, Consistency, Clarity, Correctness

- Requirements are continuously validated with the client and the user.
- Requirement validation → Checking that the specification is;
  - Complete
  - Consistent
  - Clear (Unambiguous)
  - Correct
- These are also the properties of requirements specification.

Requirements Elicitation Concepts

Completeness, Consistency, Clarity, Correctness

- Complete
  - All features of interest are described by requirements.
- Consistent
  - No two requirements of the specification contradict each other.
- Clear (Unambiguous)
  - A requirement cannot be interpreted in two mutually exclusive ways.

Requirements Elicitation Concepts

Completeness, Consistency, Clarity, Correctness

- Correct
  - Requirements specification represents accurately the system that the client needs and that the developers intend to build.

Requirements Elicitation Concepts

Completeness, Consistency, Clarity, Correctness

Complete—All features of interest are described by requirements.

Example of incompleteness: The `SatWatch` specification does not specify the boundary behavior when the user is standing within GPS accuracy limitations of a state's boundary.

Solution: Add a functional requirement stating that the time depicted by `SatWatch` should not change more often than once very 5 minutes.

Consistent—No two requirements of the specification contradict each other.

Example of inconsistency: A watch that does not contain any software faults need not provide an upgrade mechanism for downloading new versions of the software.

Solution: Revise one of the conflicting requirements from the model (e.g., rephrase the requirement about the watch not containing any faults, as it is not verifiable anyway).

Unambiguous—A requirement cannot be interpreted in two mutually exclusive ways.

Example of ambiguity: The `SatWatch` specification refers to time zones and political boundaries. Does the `SatWatch` deal with daylight saving time or not?

Solution: Clarify the ambiguous concept to select one of the mutually exclusive phenomena (e.g., add a requirement that `SatWatch` should deal with daylight saving time).

Correct—The requirements describe the features of the system and environment of interest to the client and the developer, but do not describe other unintended features.

Example of fault: There are more than 24 time zones. Several countries and territories (e.g, India) are half an hour ahead of a neighboring time zone.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Concepts

Realism, Verifiability, Traceability

- Three more properties of requirements specification.
- Realistic
  - The system can be implemented within constraints.
- Verifiable
  - Repeatable tests to demonstrate that the system fulfills the requirements specification.
- Traceable
  - Each requirement can be traced.

Requirements Elicitation Concepts

Greenfield Engineering, Reengineering, Interface Engineering

- Requirements elicitation activities can be classified into three categories, depending on the source of the requirements.
  - Greenfield engineering, reengineering, and interface engineering
- Greenfield engineering
  - Development starts from scratch.
  - Is triggered by a user need or the creation of a new market.
  - The requirements are extracted from the users and the client.

Requirements Elicitation Concepts

Greenfield Engineering, Reengineering, Interface Engineering

- Reengineering
  - Redesign and reimplementation of an existing system.
  - Triggered by technology enablers or by business processes.
  - The requirements of the new system are extracted from an existing system.

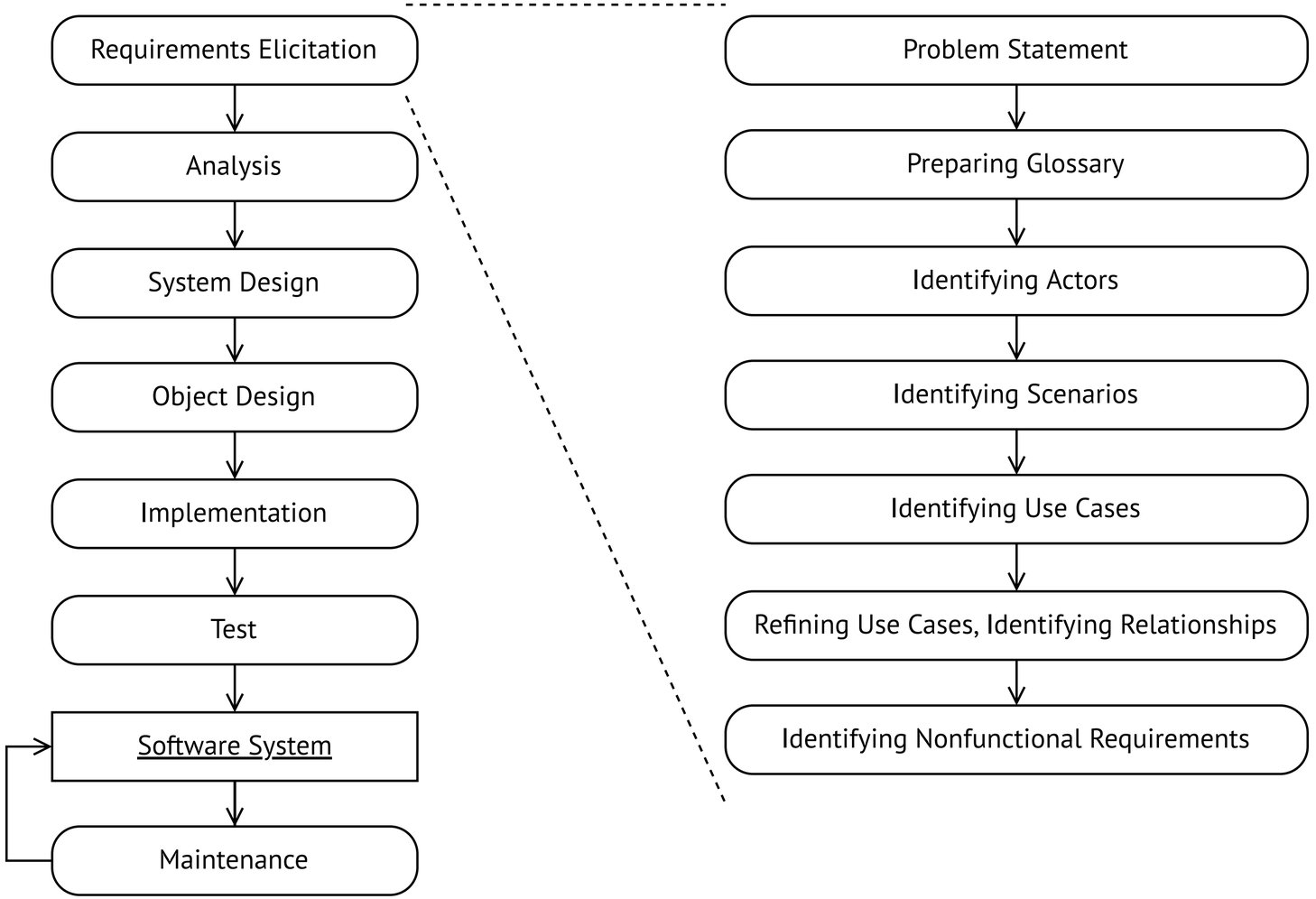
Requirements Elicitation Concepts

Greenfield Engineering, Reengineering, Interface Engineering

- Interface engineering
  - The redesign of the user interface of an existing system.
  - The legacy system is left untouched except for its interface.
  - A type of reengineering project

Requirements Elicitation Activities

Requirements Elicitation Activities



Requirements Elicitation Activities

Problem Statement

- After an initial meeting with the client, the problem statement is written.
- Problem statement is composed of;
  - Problem
  - Objectives
  - Functional Requirements
  - Nonfunctional Requirements
  - Target Environment

Requirements Elicitation Activities

Preparing Glossary

- Obstacles → Differing terminology
- Needed → A clear unique terminology
- Preparing Glossary
  - Identify the participating objects.
  - Name them, and describe them.
- The glossary should use application domain terms.
  - Names and descriptions

Requirements Elicitation Activities

Preparing Glossary

- Examples of terms that need to be clarified:
- Recurring nouns in the use cases (e.g., Incident)
- Real-world entities that the system must track (e.g., FieldOfficer, Resource)
- Real-world processes that the system must track (e.g., EmergencyOperationsPlan)
- Use cases (e.g., ReportEmergency)
- Data sources or sinks (e.g., Printer)
- Artifacts with which the user interacts (e.g., Station)

Requirements Elicitation Activities

Preparing Glossary

- Glossary is part of the requirements specification.
- Starts during the first meeting with the client.
- Updated throughout the development process.
- Benefits:
  - Help distinguishing parts, objects of the system
  - Eliminates repetition in models.
  - Eliminates ambiguity

Requirements Elicitation Activities

Preparing Glossary - Sample Glossary

Dispatcher	Police officer who manages Incidents. A Dispatcher opens, documents, and closes incidents in response to EmergencyReports and other communication with FieldOfficers. Dispatchers are identified by badge numbers.
EmergencyReport	Initial report about an Incident from a FieldOfficer to a Dispatcher. An EmergencyReport usually triggers the creation of an Incident by the Dispatcher. An EmergencyReport is composed of an emergency level, a type (fire, road accident, other), a location, and a description.
FieldOfficer	Police or fire officer on duty. A FieldOfficer can be allocated to at most one Incident at a time. FieldOfficers are identified by badge numbers.
Incident	Situation requiring attention from a FieldOfficer. An Incident may be reported in the system by a FieldOfficer or anybody else external to the system. An Incident is composed of a description, a response, a status (open, closed, documented), a location, and a number of FieldOfficers.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Identifying Actors

- Questions to identify actors:
  - Which user groups are supported by the system to perform their work?
  - Which user groups execute the system's main functions?
  - Which user groups perform secondary functions, such as maintenance and administration?
  - With what external hardware or software system will the system interact?

Requirements Elicitation Activities

Identifying Actors

- Actors → System boundary
- Sometimes it is hard to distinguish actors from objects.
- A database subsystem can at times be an actor, while in other cases it can be part of the system.
- Actors are outside of the system boundary; they are external.
- Subsystems and objects are inside the system boundary; they are internal.

Requirements Elicitation Activities

Identifying Scenarios

- Questions to identify scenarios:
  - What are the tasks that the actor wants the system to perform?
  - What information does the actor access? Who creates that data? Can it be modified or removed? By whom?
  - Which external changes does the actor need to inform the system about? How often? When?
  - Which events does the system need to inform the actor about? With what latency?

Requirements Elicitation Activities

Identifying Scenarios

Scenario name	`warehouseOnFire`
Participating actor instances	`bob, alice:FieldOfficer` `john:Dispatcher`
Flow of events	1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the "Report Emergency" function from her `FRIEND` laptop. 2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene, given that the area appears to be relatively busy. She confirms her input and waits for an acknowledgment. 3. John, the `Dispatcher`, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the `Incident` site and sends their estimated arrival time (ETA) to Alice. 4. Alice receives the acknowledgment and the ETA.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Identifying Scenarios

- Common types of scenarios:
  - As-is scenarios → Current situation
  - Visionary scenarios → Future system
  - Evaluation scenarios → User tasks against which the system is to be evaluated
  - Training scenarios → Tutorials used for introducing new users to the system

Requirements Elicitation Activities

Identifying Use Cases

- Generalize scenarios and identify high-level use cases.
  - Name use cases
  - Attach use cases to the initiating actors
  - Provide high-level descriptions of each use case

Requirements Elicitation Activities

Identifying Use Cases

Use case name	`ReportEmergency`
Participating actors	Initiated by `FieldOfficer` Communicates with `Dispatcher`
Flow of events	1. The `FieldOfficer` activates the "Report Emergency" function of her terminal. 2. `FRIEND` responds by presenting a form to the `FieldOfficer`. 3. The `FieldOfficer` completes the form by selecting the emergency level, type, location, and brief description of the situation. The `FieldOfficer` also describes possible responses to the emergency situation. Once the form is completed, the `FieldOfficer` submits the form. 4. `FRIEND` receives the form and notifies the `Dispatcher`. 5. The `Dispatcher` reviews the submitted information and creates an `Incident` in the database by invoking the `OpenIncident` use case. The `Dispatcher` selects a response and acknowledges the report. 6. `FRIEND` displays the acknowledgment and the selected response to the `FieldOfficer`.
Entry condition	• The `FieldOfficer` is logged into `FRIEND`.
Exit conditions	• The `FieldOfficer` has received an acknowledgment and the selected response from the `Dispatcher`, OR • The `FieldOfficer` has received an explanation indicating why the transaction could not be processed.
Quality requirements	• The `FieldOfficer`'s report is acknowledged within 30 seconds. • The selected response arrives no later than 30 seconds after it is sent by the `Dispatcher`.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Identifying Use Cases

- Describing a use case entails specifying four fields:
  - Entry conditions, exit conditions, flow of events, quality requirements
- Describe the **entry** and **exit conditions** of a use case:
  - Understand the conditions under which a use case is invoked
  - Understand the impact of the use case

Requirements Elicitation Activities

Identifying Use Cases

- Describe the **flow of events** of a use case:
  - Interaction between actors and system
  - Which actions are accomplished by the actor?
  - Which actions are accomplished by the system?
- Describe the **quality requirements** associated with a use case:
  - Elicit nonfunctional requirements in the context of a specific functionality.

Requirements Elicitation Activities

Identifying Use Cases

Simple Use Case Writing Guide

- Use cases should be named with verb phrases. The name of the use case should indicate what the user is trying to accomplish (e.g., `ReportEmergency`, `OpenIncident`).
- Actors should be named with noun phrases (e.g., `FieldOfficer`, `Dispatcher`, `Victim`).
- The boundary of the system should be clear. Steps accomplished by the actor and steps accomplished by the system should be distinguished (e.g., in Figure 4-7, system actions are indented to the right).
- Use case steps in the flow of events should be phrased in the active voice. This makes it explicit who accomplished the step.
- The causal relationship between successive steps should be clear.
- A use case should describe a complete user transaction (e.g., the `ReportEmergency` use case describes all the steps between initiating the emergency reporting and receiving an acknowledgment).
- Exceptions should be described separately.
- A use case should not describe the user interface of the system. This takes away the focus from the actual steps accomplished by the user and is better addressed with visual mock-ups (e.g., the `ReportEmergency` only refers to the "Report Emergency" function, not the menu, the button, nor the actual command that corresponds to this function).
- A use case should not exceed two or three pages in length. Otherwise, use include and extend relationships to decompose it in smaller use cases, as explained in Section 4.4.5.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Identifying Use Cases - Example of a Poor Use Case

Figure 4-8 Example of use case writing guide.

Use case name	`Accident`	*Bad name: What is the user trying to accomplish?*
Initiating actor	Initiated by `FieldOfficer`	
Flow of events	1. The `FieldOfficer` reports the accident.	
	2. An ambulance is dispatched.	*Causality: Which action caused the `FieldOfficer` to receive an acknowledgment?* *Passive voice: Who dispatches the ambulance?*
	3. The `Dispatcher` is notified when the ambulance arrives on site.	*Incomplete transaction: What does the `FieldOfficer` do after the ambulance is dispatched?*

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Refining Use Cases and Identifying Relationships

- Focus completeness and correctness.
- Identify functionality not covered by scenarios
  - Document it by refining use cases or writing new ones.
- Describe seldom occurring cases and exception handling as seen by the actors.
- More details about the features
- More details about the constraints

Requirements Elicitation Activities

Refining Use Cases and Identifying Relationships

Use case name	`ReportEmergency`
Participating actors	Initiated by `FieldOfficer` Communicates with `Dispatcher`
Flow of events	1. The `FieldOfficer` activates the "Report Emergency" function of her terminal. 2. `FRIEND` responds by presenting a form to the officer. *The form includes an emergency type menu (general emergency, fire, transportation) and location, incident description, resource request, and hazardous material fields.* 3. The `FieldOfficer` completes the form by *specifying minimally the emergency type and description fields*. The `FieldOfficer` may also describe possible responses to the emergency situation *and request specific resources*. Once the form is completed, the `FieldOfficer` submits the form. 4. `FRIEND` receives the form and notifies the `Dispatcher` *by a pop-up dialog*. 5. The `Dispatcher` reviews the submitted information and creates an `Incident` in the database by invoking the `OpenIncident` use case. *All the information contained in the `FieldOfficer's` form is automatically included in the `Incident`. The `Dispatcher` selects a response by allocating resources to the `Incident` (with the `AllocateResources` use case) and acknowledges the emergency report by sending a short message to the `FieldOfficer`.* 6. `FRIEND` displays the acknowledgment and the selected response to the `FieldOfficer`.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Refining Use Cases and Identifying Relationships

- During this activity:
  - The elements that are manipulated by the system are detailed.
  - The low-level sequence of interactions between the actor and the system are specified.
  - Access rights are specified.
  - Missing exceptions are identified and their handling specified.
  - Common functionality among use cases are factored out.

Requirements Elicitation Activities

Refining Use Cases and Identifying Relationships

- What we do during the identification of relationships?
- Relationships: communication, extend and include relationships.
- What we do in the context of **communication relationships**?
  - Identify communication relationships that are not yet identified.
  - Refine the identified communication relationships.
  - Specify which actor can invoke a specific use case.
  - Specify which actors communicate with a specific use case.
  - Result → Specification of access control for the system at a coarse level.

Requirements Elicitation Activities

Refining Use Cases and Identifying Relationships

- What we do in the context of **extend relationships**?
  - Separate exceptional and optional flows of events.
  - Result → Shorter base use cases
  - Distinguishing common cases from exceptional cases.

Requirements Elicitation Activities

Refining Use Cases and Identifying Relationships

- What we do in the context of **include relationships**?
  - Factoring out shared behavior from use cases.
  - Result → Shorter descriptions and fewer redundancies.
  - Rule of thumb → Behavior should only be factored out into a separate use case if it is shared across two or more use cases.

Requirements Elicitation Activities

Refining Use Cases and Identifying Relationships

Heuristics for developing scenarios and use cases

- Use scenarios to communicate with users and to validate functionality.
- First, refine a single scenario to understand the user's assumptions about the system. The user may be familiar with similar systems, in which case, adopting specific user interface conventions would make the system more usable.
- Next, define many not-very-detailed scenarios to define the scope of the system. Validate with the user.
- Use mock-ups as visual support only; user interface design should occur as a separate task after the functionality is sufficiently stable.
- Present the user with multiple and very different alternatives (as opposed to extracting a single alternative from the user). Evaluating different alternatives broadens the user's horizon. Generating different alternatives forces developers to "think outside the box."
- Detail a broad vertical slice when the scope of the system and the user preferences are well understood. Validate with the user.

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Identifying Nonfunctional Requirements

- Nonfunctional requirements can impact the work of the user in unexpected ways.
- Accurately elicit all the essential nonfunctional requirements
- Collaboration of client and developer is important.

Requirements Elicitation Activities

Identifying Nonfunctional Requirements

Category	Example questions
Usability	• What is the level of expertise of the user? • What user interface standards are familiar to the user? • What documentation should be provided to the user?
Reliability *(including robustness, safety, and security)*	• How reliable, available, and robust should the system be? • Is restarting the system acceptable in the event of a failure? • How much data can the system loose? • How should the system handle exceptions? • Are there safety requirements of the system? • Are there security requirements of the system?
Performance	• How responsive should the system be? • Are any user tasks time critical? • How many concurrent users should it support? • How large is a typical data store for comparable systems? • What is the worse latency that is acceptable to users?
Supportability *(including maintainability and portability)*	• What are the foreseen extensions to the system? • Who maintains the system? • Are there plans to port the system to different software or hardware environments?

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Requirements Elicitation Activities

Identifying Nonfunctional Requirements

Implementation	• Are there constraints on the hardware platform? • Are constraints imposed by the maintenance team? • Are constraints imposed by the testing team?
Interface	• Should the system interact with any existing systems? • How are data exported/imported into the system? • What standards in use by the client should be supported by the system?
Operation	• Who manages the running system?
Packaging	• Who installs the system? • How many installations are foreseen? • Are there time constraints on the installation?
Legal	• How should the system be licensed? • Are any liability issues associated with system failures? • Are any royalties or licensing fees incurred by using specific algorithms or components?

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Documenting Requirements Elicitation

Documenting Requirements Elicitation

RAD

- The result of the requirements elicitation → Requirements Specification
- Requirements Specification → Requirement Analysis Document
- RAD → Requirements Analysis Document
- RAD = Requirement Specification + Analysis Model
- RAD = Requirement Elicitation + Analysis

Documenting Requirements Elicitation

RAD

- Audience for the RAD
  - Clients
  - Users
  - Project managers
  - System analysts → Developers who participate in the requirements
  - System designers → Developers who participate in the system design

Documenting Requirements Elicitation

RAD

1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Objectives
2. Current system
3. Proposed system
   3.1 Overview
   3.2 Functional requirements
   3.3 Nonfunctional requirements
   3.4 System models
      3.4.1 Scenarios
      3.4.2 Use case model
      3.4.3 Object model
      3.4.4 Dynamic model
      3.4.5 User interface
4. Glossary

Documenting Requirements Elicitation

RAD

- RAD is written after the use case model is stable.
- RAD can be updated throughout the development process.
- RAD is the baseline and put under configuration management.
- The revision history section of the RAD is important.

References

- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.
- Object Management Group, OMG Unified Modeling Language Superstructure, Version 2.2., http://www.omg.org/2009.

Thank you.