Analysis, Design, and Software Architecture (BDSA)

Paolo Tell

Requirements Engineering

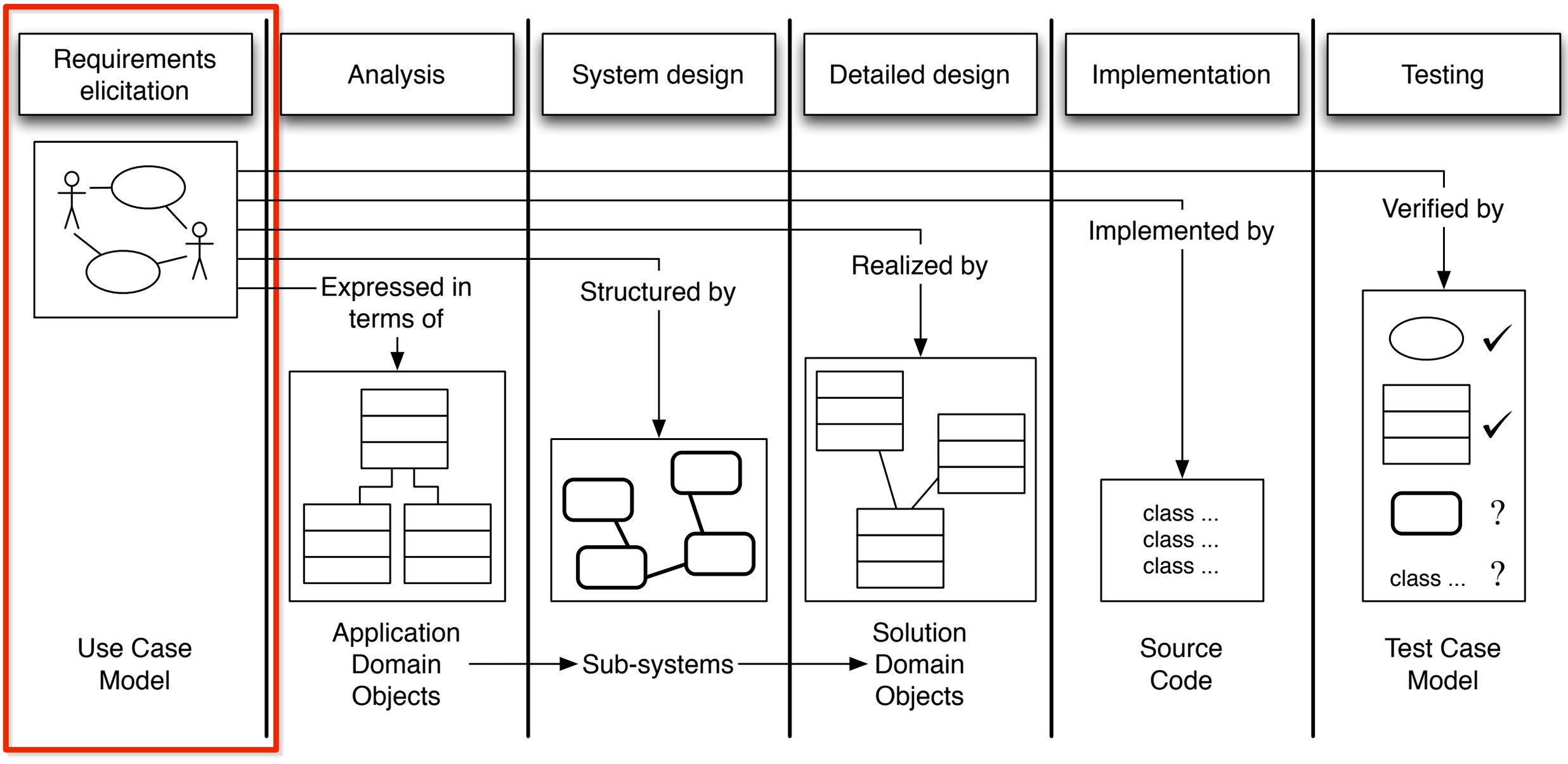
IT University
of Copenhagen

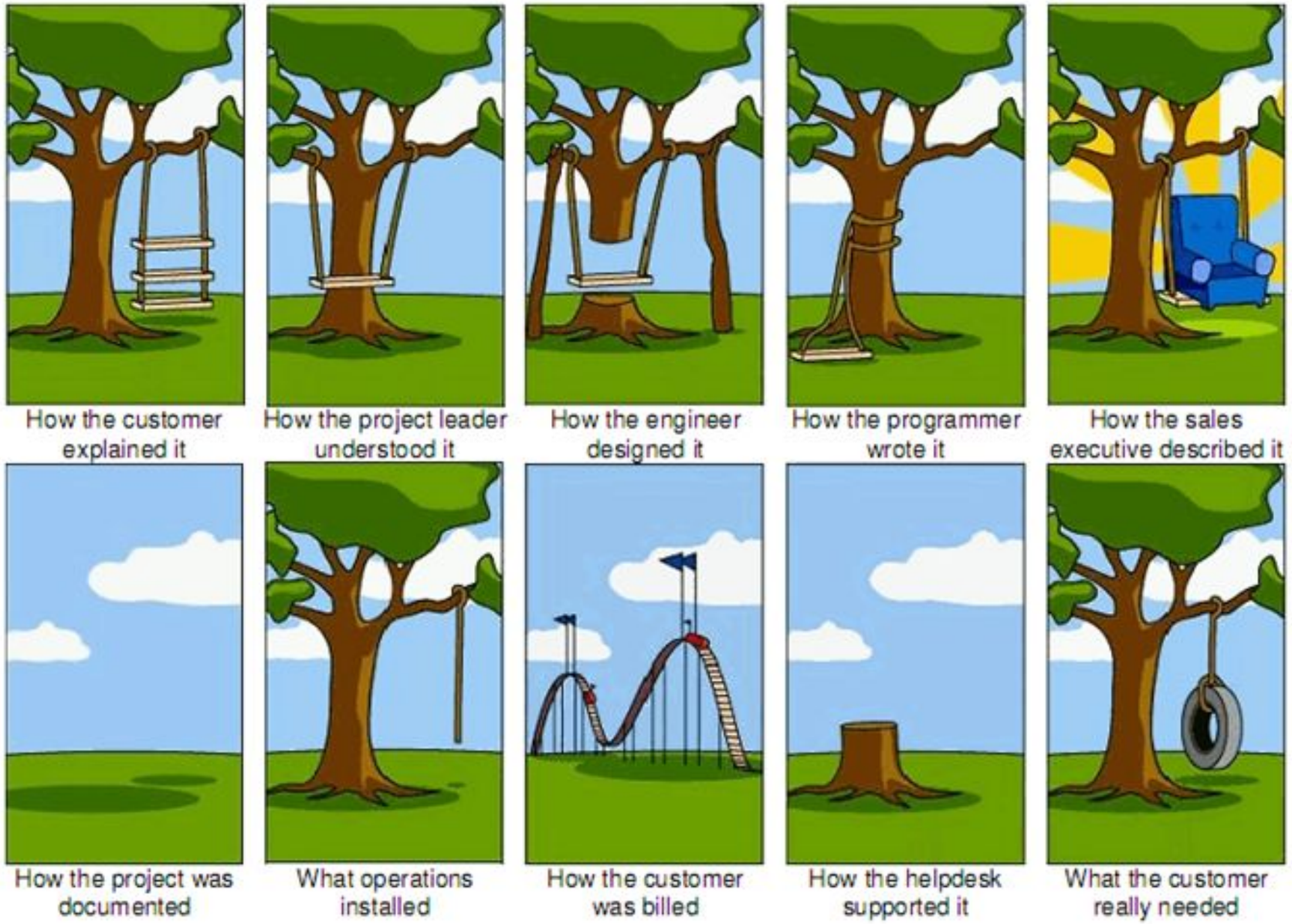
Outline

- Literature
  - [OOSE] ch. 4
  - (Optional) [SE10] ch. 4

- Topics covered:
  - What is requirements engineering?
  - Functional, non-functional, and domain requirements
  - Requirements elicitation and analysis in OO
  - Managing requirements specification
  - The requirements engineering process

Software lifecycle activities





What is requirements engineering?

What is requirements engineering?

Disclaimer:

"[i]n this chapter, I present a 'traditional' view of requirements rather than requirements in agile processes" [SE10, ch4]

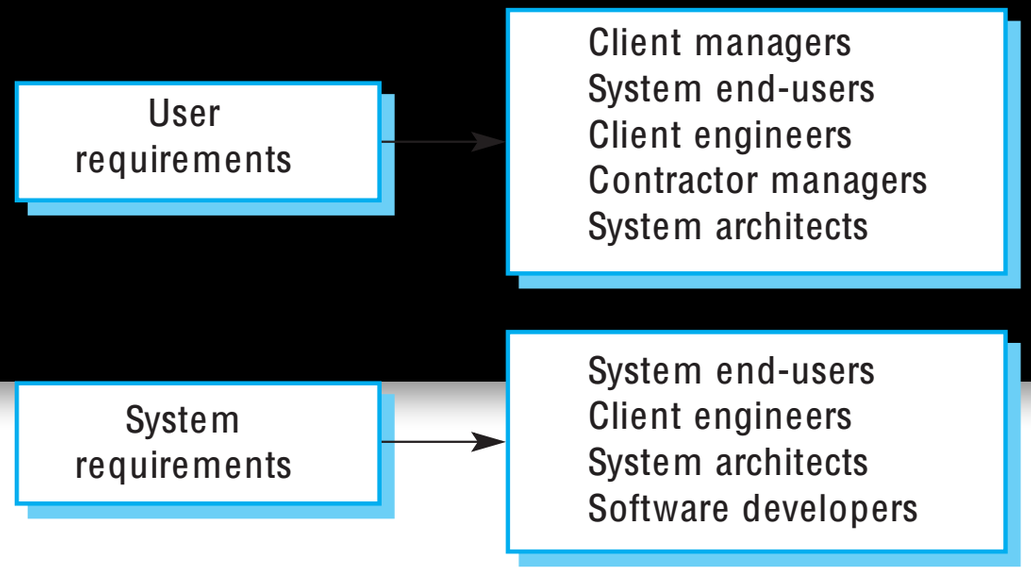
- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

Types of requirements

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

- System requirements
  - A structured document setting out detailed descriptions of the functions of the system, services, and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

Why two types? Different readers!



User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

[1] Sommerville, Ian.
"Software Engineering, 10th edition".
Pearson Education Limited, 2016.

Functional, non-functional, and domain requirements

- Functional requirements
  - Statements of services the system should provide, of how the system should react to particular inputs, and of how the system should behave in particular situations.
  - May state what the system should not do.

- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services. (Remember the different phases in the testing process)

- Domain requirements
  - Constraints on the system from the domain of operation.

Functional, non-functional, and domain requirements

Examples of functional requirements

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘search’ in requirement 1: “*A user shall be able to search the appointments lists for all clinics.*” Do you see any potential source of ambiguity?

Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.

- Consider the term ‘search’ in requirement 1: “*A user shall be able to search the appointments lists for all clinics.*” Do you see any potential source of ambiguity?
  - User intention – search for a patient name across all appointments in all clinics;
  - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic, then search.

Non-functional requirements

- These define system properties and constraints.
  - Examples of properties are: reliability, security, response time, storage requirements, etc.
  - Examples of constraints are: I/O device capability, system representations, programming language, etc.

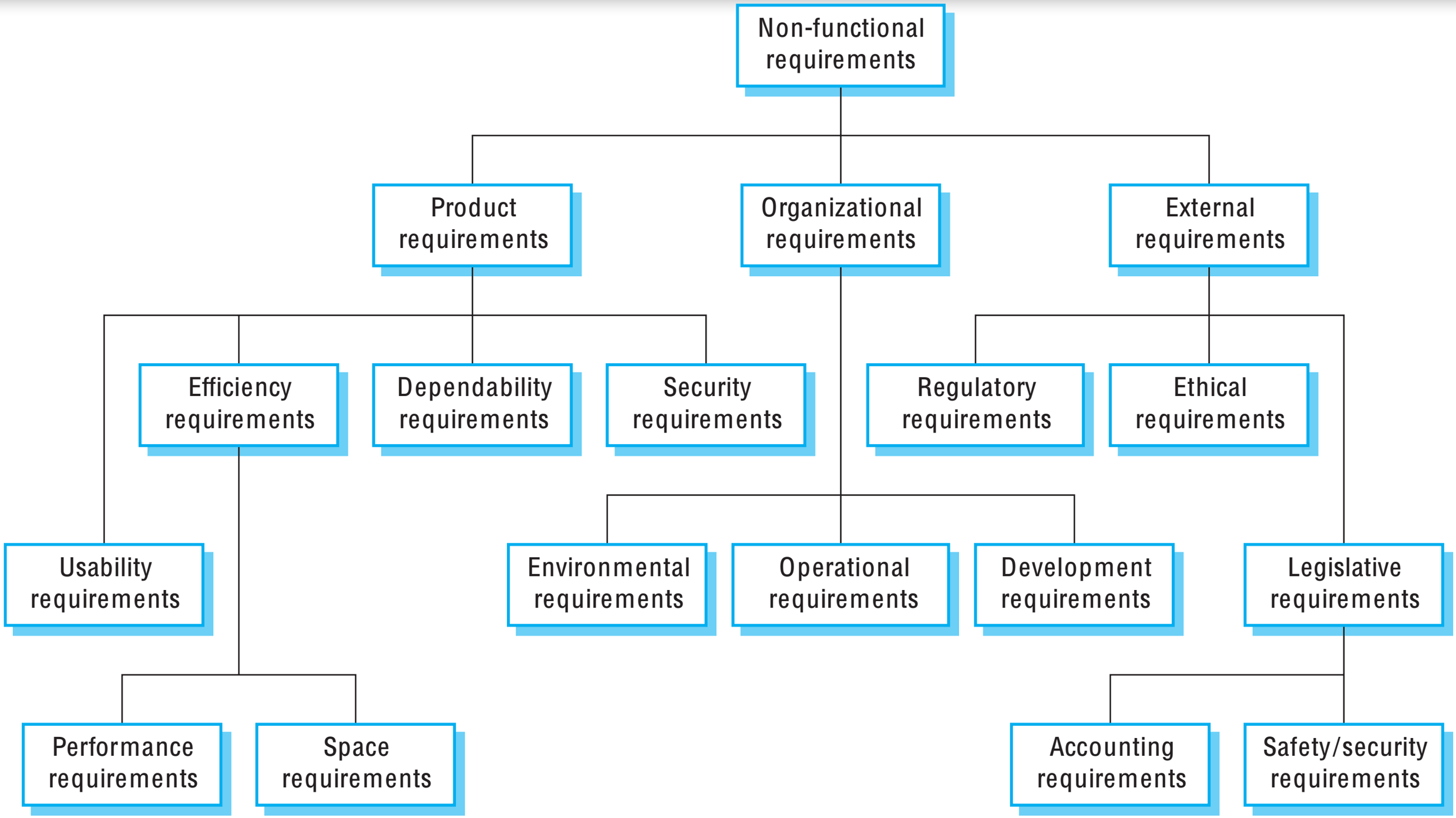
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Non-functional requirements implementation

- Non-functional requirements may affect the overall architecture of a system rather than the individual components. In other words, they may be elevated to become architectural concerns.
  - For example: to ensure that performance requirements are met, you may have to organise the system to minimise communications between components.

- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  - It may also generate requirements that restrict existing requirements.

Types of non-functional requirement



[1] Sommerville, Ian. “Software Engineering, 10th edition”. Pearson Education Limited, 2016.

Examples of non-functional requirements

- Product requirement
  - The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

- Organisational requirement
  - Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

- External requirement
  - The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Goals and non-functional requirements

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.

- Goal
  - A general intention of the user such as ease of use.

- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested.

- Goals are helpful to developers as they convey the intentions of the system users.

- Verifiable metrics are useful in quality assurance (QA) and contract management.

Example: a usability requirement

- Goal
  - The system should be easy to use by medical staff and should be organised in such a way that user errors are minimised.

- Testable non-functional requirement
  - Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use.

Metrics for specifying non-functional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

[1] Sommerville, Ian. "Software Engineering, 10th edition". Pearson Education Limited, 2016.

Domain requirements

- Rather than the specific needs of system users, the application domain imposes requirements on the system.
  - For example: a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements may be new functional requirements, constraints on existing requirements, or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Example of domain requirements

- The deceleration of the train shall be computed as:
  - D (train) = D (control) + D (gradient)

- where
  - D (gradient) is 9.81ms2 * compensated gradient/alpha and
  - the values of 9.81ms2 /alpha are known for different types of train.

- This is application domain specific language.

Example of domain requirements

- This is related to an application domain specific language.
- Think about the Danish system:
  - as an unemployed citizen, I can go on vacation from my unemployment. (?????)
- In the case of outsourcing, how can you expect an Indian developer to understand this concept.

Completeness, consistency, clarity, and correctness

- Complete
  - all features of interest are described by requirements.

- Consistent
  - no two requirements of the specification contradicts each other.

- Clear (or unambiguous)
  - a requirement cannot be interpreted in two mutually exclusive ways.

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

Realism, verifiability, and traceability

- Realistic
  - the system can be implemented within given constraints.

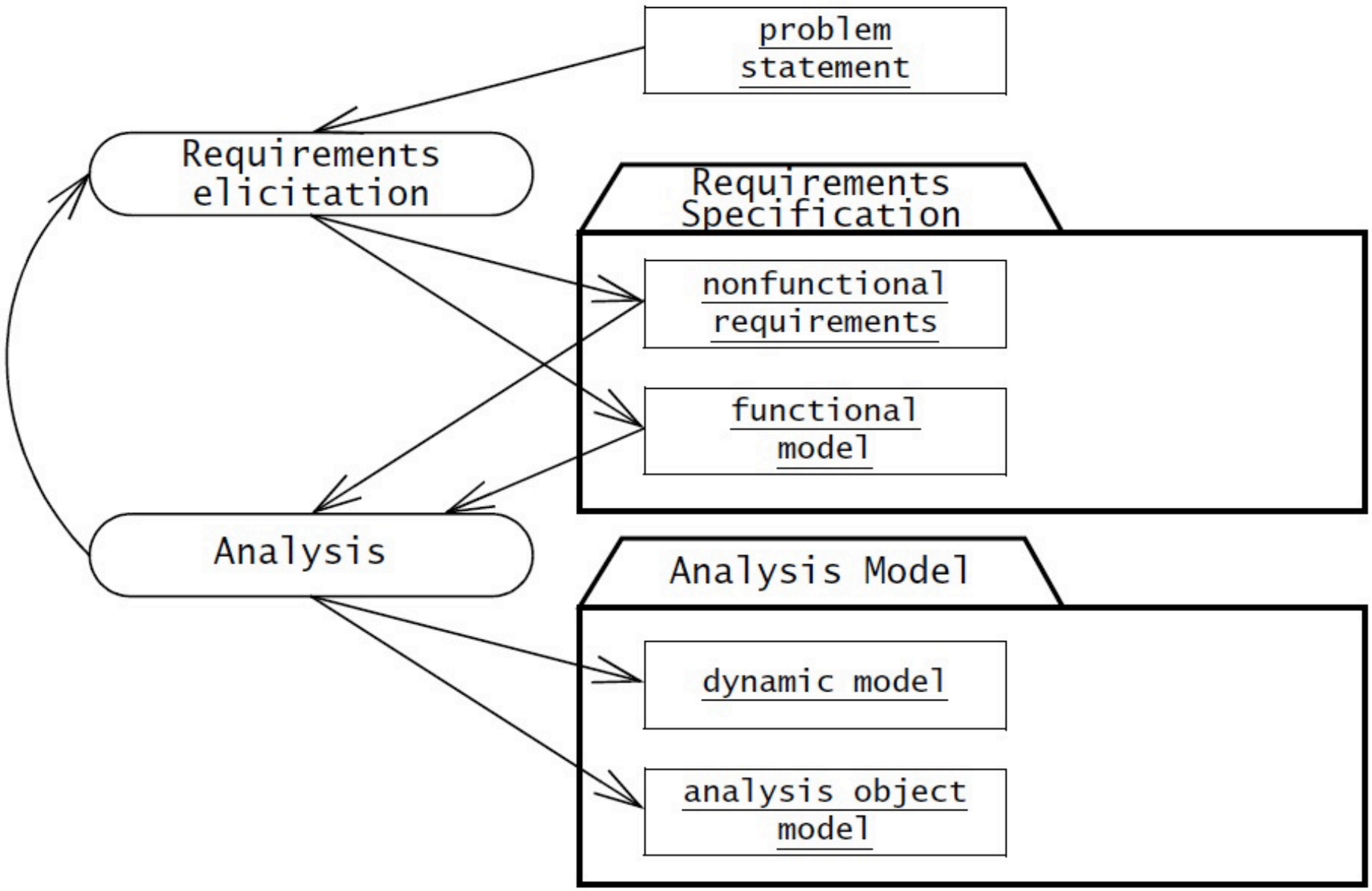
- Verifiable
  - once build, repeated tests can be designed to demonstrate that the system fulfills the requirements specification.

- Traceable
  - each requirement can be traced throughout the development process to its system function, and vice versa.

FURPS+ Model

- **FURPS+** model
- **F**unctional
- **U**sability
- **R**eliability
- **P**erformance
- **S**upportability
- **+**
  - Implementation
  - Interface
  - Operations
  - Packaging
  - Legal
  - ....

- Non-functional requirement are also called:
  - software qualities, quality requirements, architecturally significant requirements, constraints, pseudo requirements.



Requirements elicitation and analysis in OO

Requirement elicitation and analysis

- Two questions need to be answered:
  - How can we identify the purpose of a system?
  - What is inside, what is outside the system?

- These two questions are answered during requirements elicitation and analysis.

- Requirements elicitation
  - Definition of the system in terms understood by the customer and/or user ("Requirements specification").

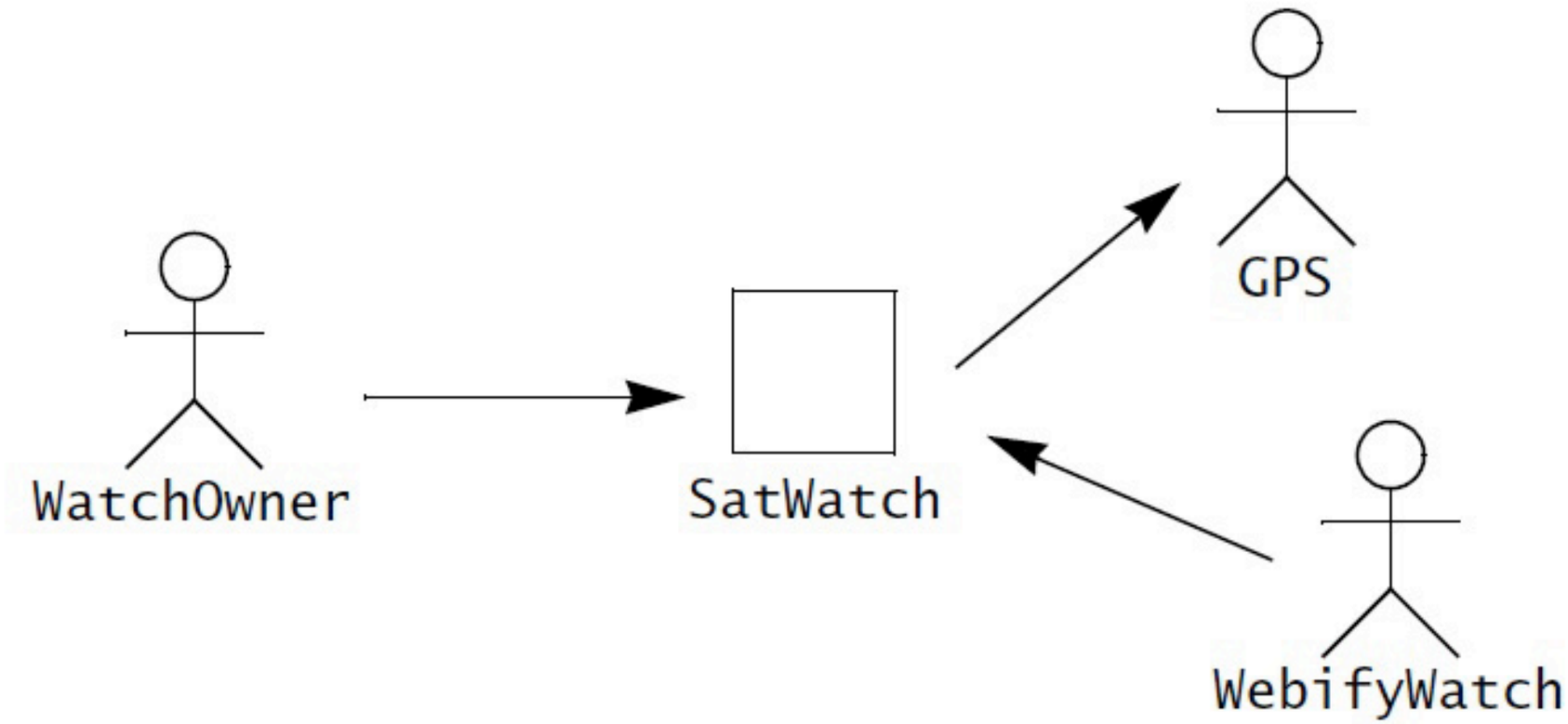
- Analysis
  - Definition of the system in terms understood by the developer (Technical specification, "Analysis model").

- Requirements Process
  - Consists of the activities Requirements Elicitation and Analysis.

Requirements elicitation activities

- Identify Actors
- Identify Scenarios
- Identify Use Cases
- Refine Use Cases
- Identify Relationships between Actors and Use Cases
- Identify Initial Analysis Objects
- Identify Non-functional Requirements

Identify actors



- Questions to identify Actors
  - which users are supported by the system to perform work?
  - which users execute the main functions?
  - which users perform secondary functions (maintenance, operation, etc.)?
  - with what external system (hw/sw) will the system interact?

Identify 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.

Figure 4-6 `warehouseOnFire` scenario for the `ReportEmergency` use case.

Types of scenarios

- As-is scenario
  - Describes a current situation.
  - Commonly used in re-engineering projects. The user describes the system.

- Visionary scenario
  - Describes a future system.
  - Often used in greenfield engineering and interface engineering projects.
  - Visionary scenarios are often not done by the user or developer alone.

- Evaluation scenario
  - Description of a user task against which the system is to be evaluated.

- Training scenario
  - A description of the step by step instructions that guide a novice user through a system.

Heuristics for finding scenarios

- Ask yourself or the client the following questions:

  GPS-based navigation system replacing a paper-based map.

  - What are the primary tasks that the system needs to perform?

    Initialisation, getting the current position, finding north.

  - What data will the actor create, store, change, remove, or add in the system?

    On a map: Pencil the date when a summit was climbed, maybe on the edge of the map a little note, comparison of planned vs. actual route. GPS: Additional data Storage of traveled route, accumulation of total miles driven so far, miles climbed so far.

  - What external changes does the system need to know about?

    Map: Nothing, paper cannot be notified, GPS: Change of location

  - What changes or events will the actor of the system need to be informed about?

    New map has been issued, road repair sites, congestions, hidden radars behind a curve.

- However, don't rely on questions and questionnaires alone

- Insist on task observation if the system already exists (interface engineering or reengineering).
  - ask to speak to the end user, not just to the client.
  - expect resistance and try to overcome it.

Identify 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`.

Content of a use case

- Name of Use Case
- Actors
  - Description of Actors involved in use case
- Entry condition
  - "This use case starts when…"
- Flow of Events
  - Free form, informal natural language
- Exit condition
  - "This use cases terminates when…"
- Exceptions
  - Describe what happens if things go wrong
- Quality Requirements
  - Nonfunctional Requirements, Constraints

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.

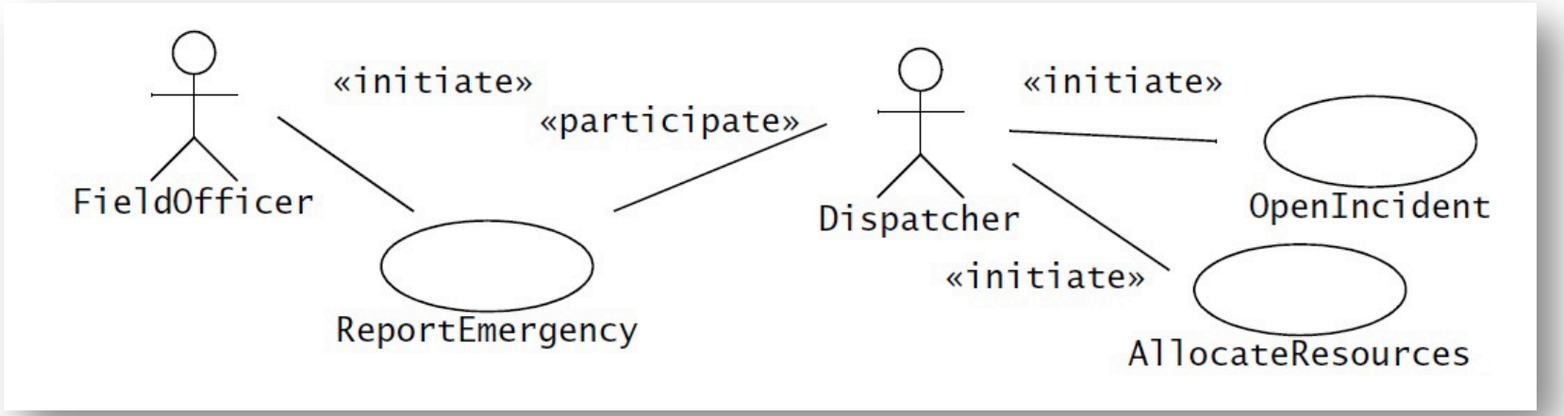
Figure 4-8 Example of use case writing guide.

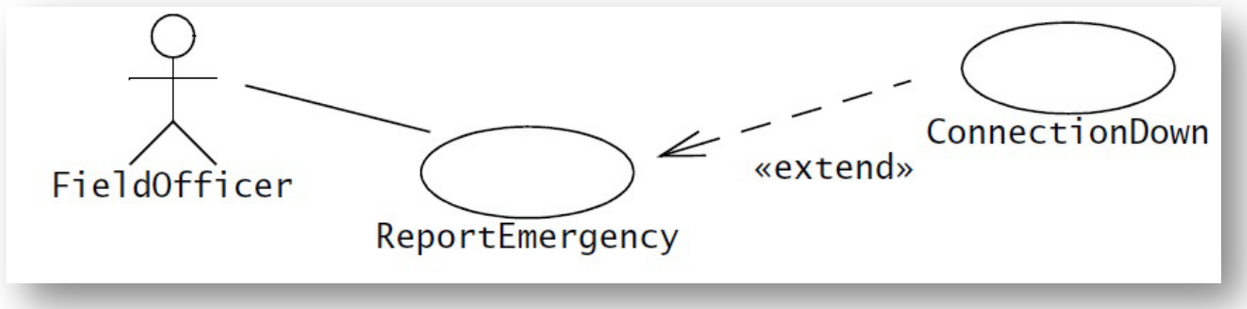
Refine use cases

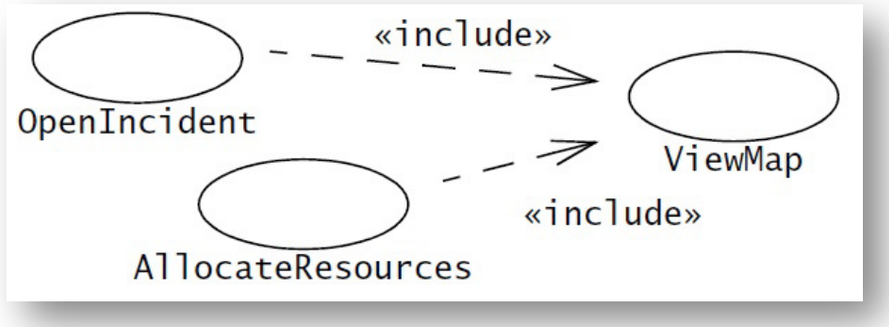
- Continuous refining, rewriting, and extending (and deleting!).
- Seek to
  - validate (w. clients/users/customers)
  - remove unambiguity
  - provide sufficient details
  - make them complete
  - consistent
  - precise (delete stuff!)

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`.
Entry condition	• ...

Identify relationships between actors and use cases



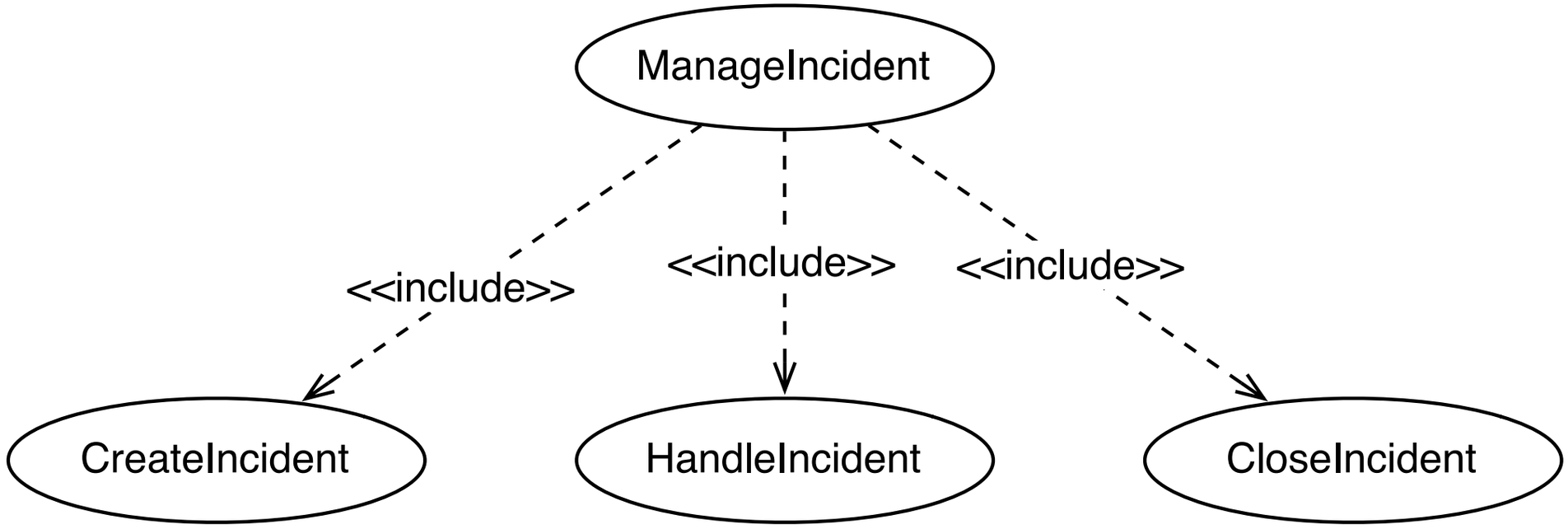




Use Case Associations

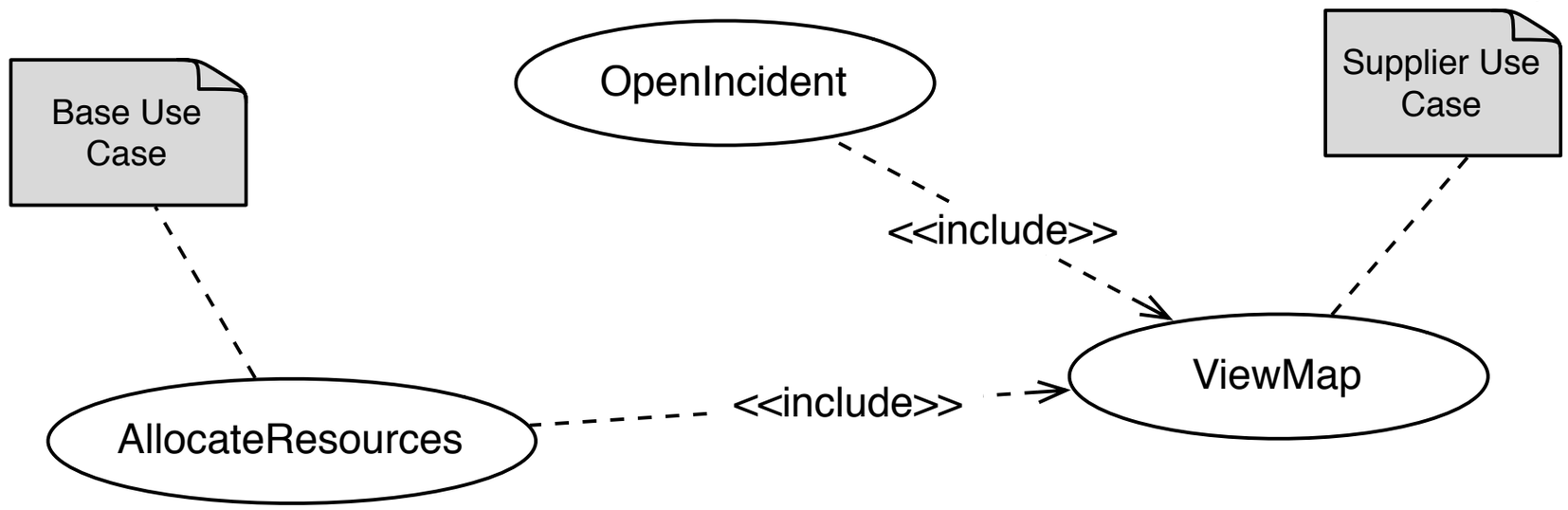
- Dependencies between use cases are represented with use case associations
- Associations are used to reduce complexity
  - Decompose a long use case into shorter ones
  - Separate alternate flows of events
  - Refine abstract use cases
- Types of use case associations
  - Includes
  - Extends
  - Generalisation

<<include>> Functional decomposition



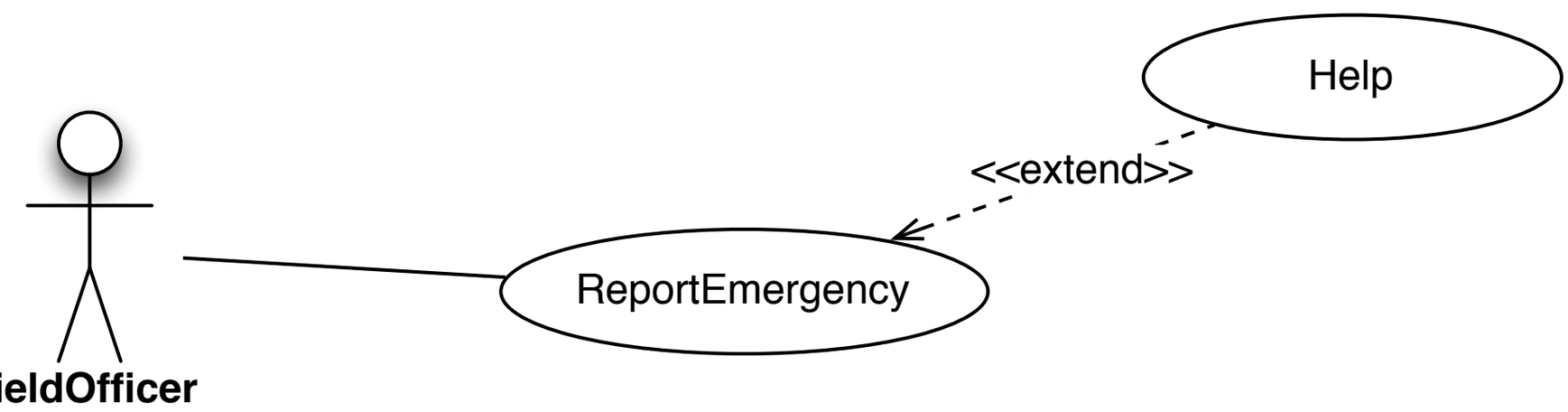
- Problem: a function in the original problem statement is too complex.
- Solution: describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into shorter use cases.

<<include>> Reuse of existing functionality



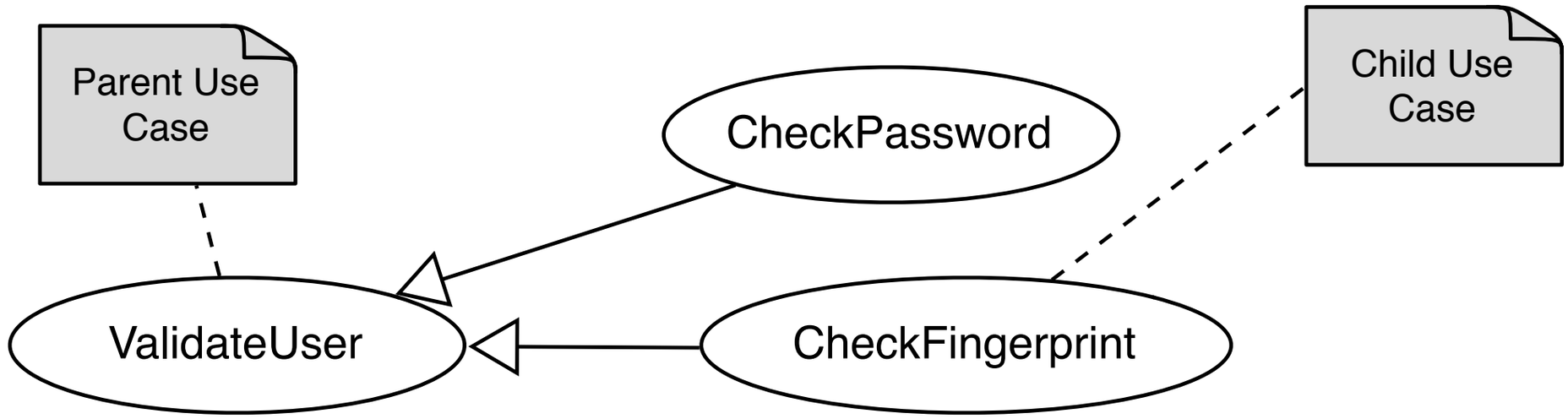
- Problem: There are overlaps among use cases. How can we reuse flows of events instead of duplicating them?
- Solution: The includes association from use case A to use case B indicates that an instance of use case A performs all the behavior described in use case B ("A delegates to B")
- Example: Use case "ViewMap" describes behavior that can be used by use case "OpenIncident" ("ViewMap" is factored out)

<<extend>> Association for use cases



- Problem: the functionality in the original problem statement needs to be extended.
- Solution: an extend association from use case A to use case B.
- Example: "ReportEmergency" is complete by itself, but can be extended by use case "Help" for a scenario in which the user requires help.

Generalisation in use cases



- Problem: we want to factor out common (but not identical) behaviour.
- Solution: the child use cases inherit the behaviour and meaning of the parent use case and add or override some behaviour.
- Example: "ValidateUser" is responsible for verifying the identity of the user. The customer might require two realisations: "CheckPassword" and "CheckFingerprint".

Identify initial analysis objects

- Terms
- Nouns in the use cases
- Real-world entities
- Real-world processes
- Use cases
- Data sources / sinks
- Artifacts for interaction

- NB – always use application domain terms (not systems terms)

Example

Table 4-2 Participating objects for the `ReportEmergency` use case.

`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`.

Identifying non-functional requirements

Table 4-3 Example questions for eliciting 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*	• What are the foreseen extensions to the system? • Who maintains the system?

Content of a use case

- Name of Use Case
- Actors
  - Description of Actors involved in use case
- Entry condition
  - "This use case starts when…"
- Flow of Events
  - Free form, informal natural language
- Exit condition
  - "This use cases terminates when…"
- Exceptions
  - Describe what happens if things go wrong
- Quality Requirements
  - Nonfunctional Requirements, Constraints

Identify 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`.

Managing requirements specification

Managing requirements specification

- Negotiating specifications & requirements with clients
  - iterative process
  - involves technical staff working with customers
  - may involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.

- Maintaining traceability
  - trace/follow the development of a requirements
  - manage relationship between requirement and the system implementation (very, very difficult!)
  - cross-reference between requirements, models, code, tests, etc.

- Documenting requirements
  - requirements analysis document (RAD)
  - forms the contract w. client (very, very important!)
  - needs to be subject to rigorous configuration management (baseline, ...)

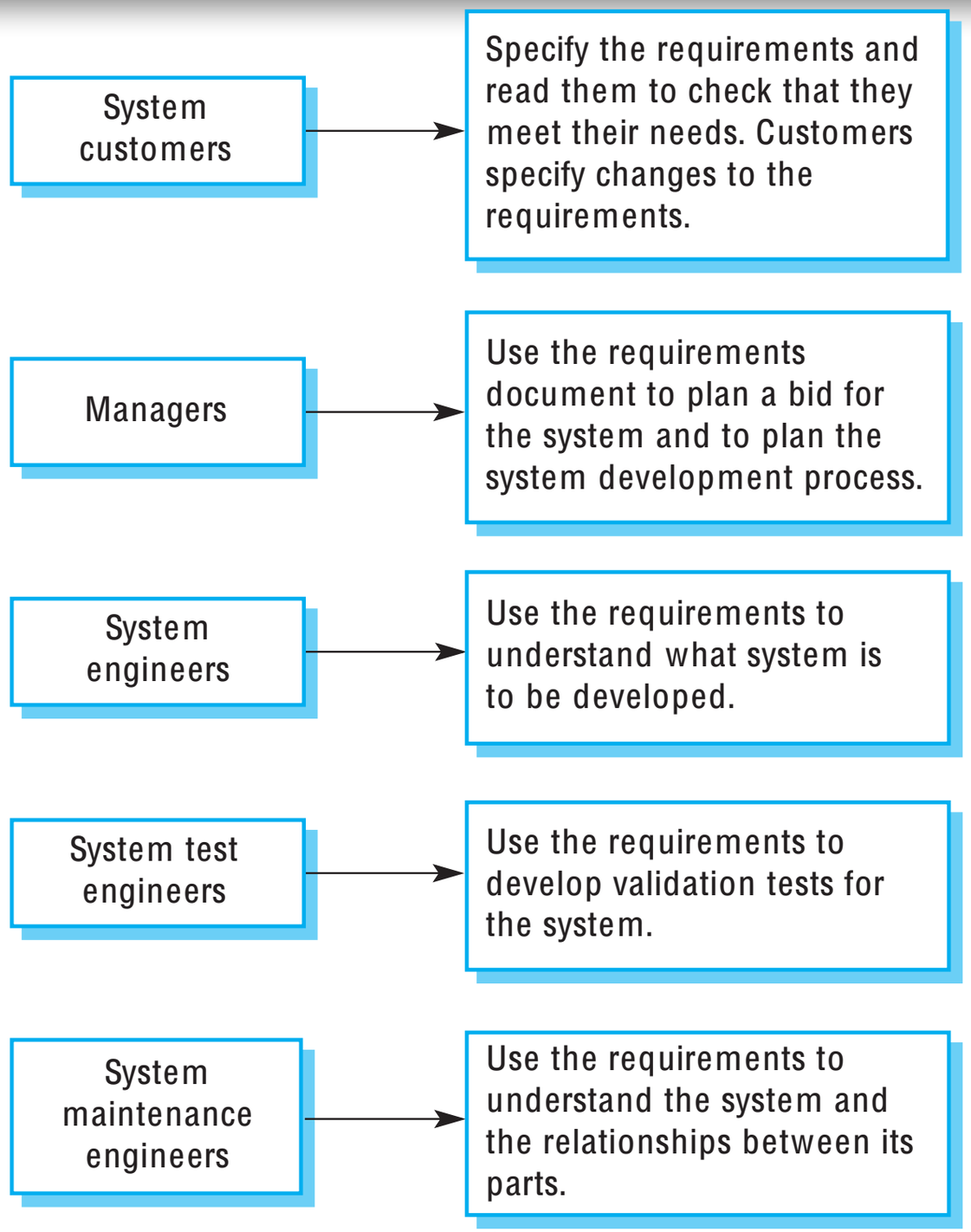
The software requirements document

- The software requirements document is the official statement of what is required of the system developers.

- Should include both a definition of user requirements and a specification of the system requirements.

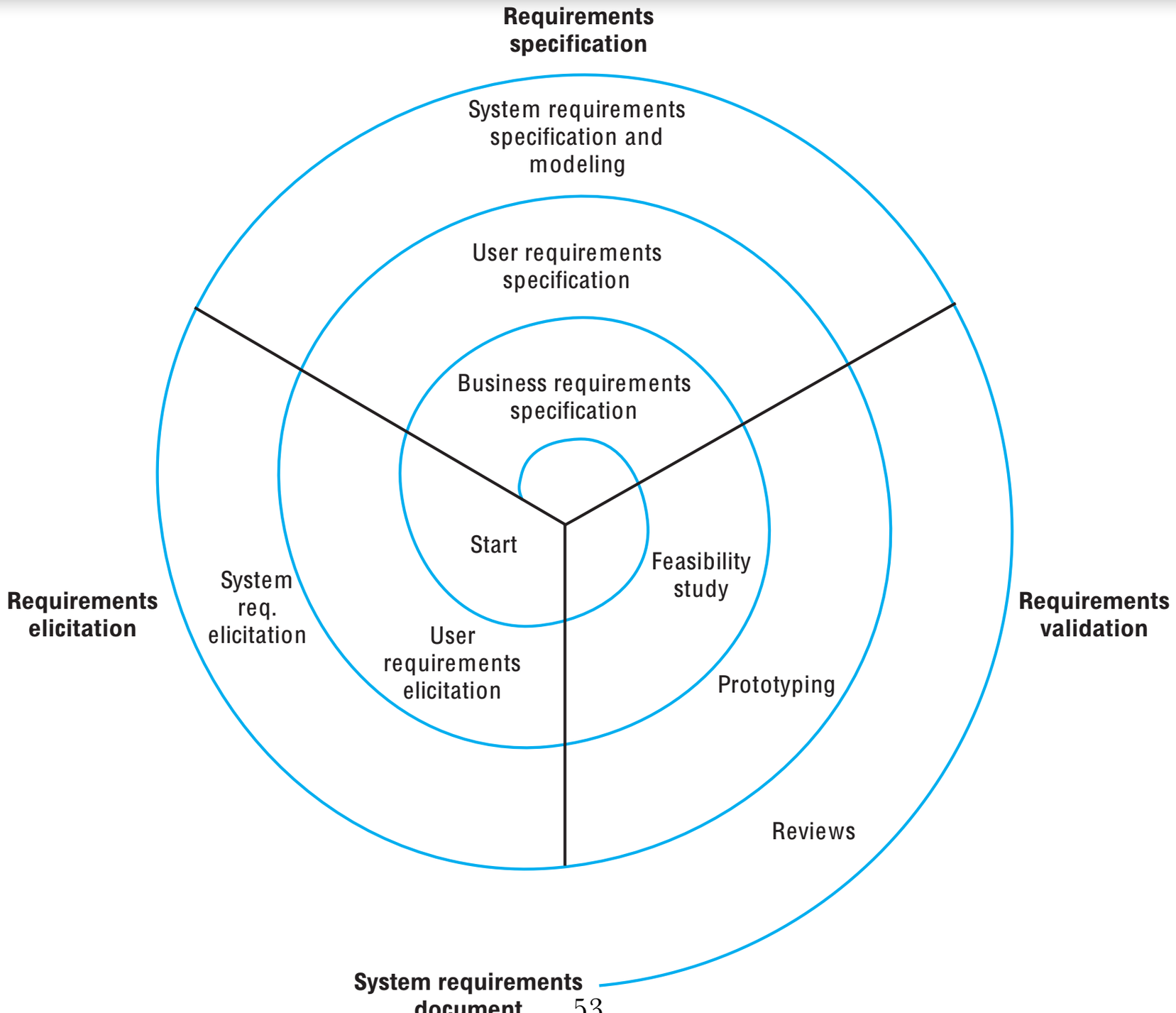
- It is <u>not</u> a design document.
  - As far as possible, it should set off WHAT the system should do rather than HOW it should do it.

Users of a requirements document



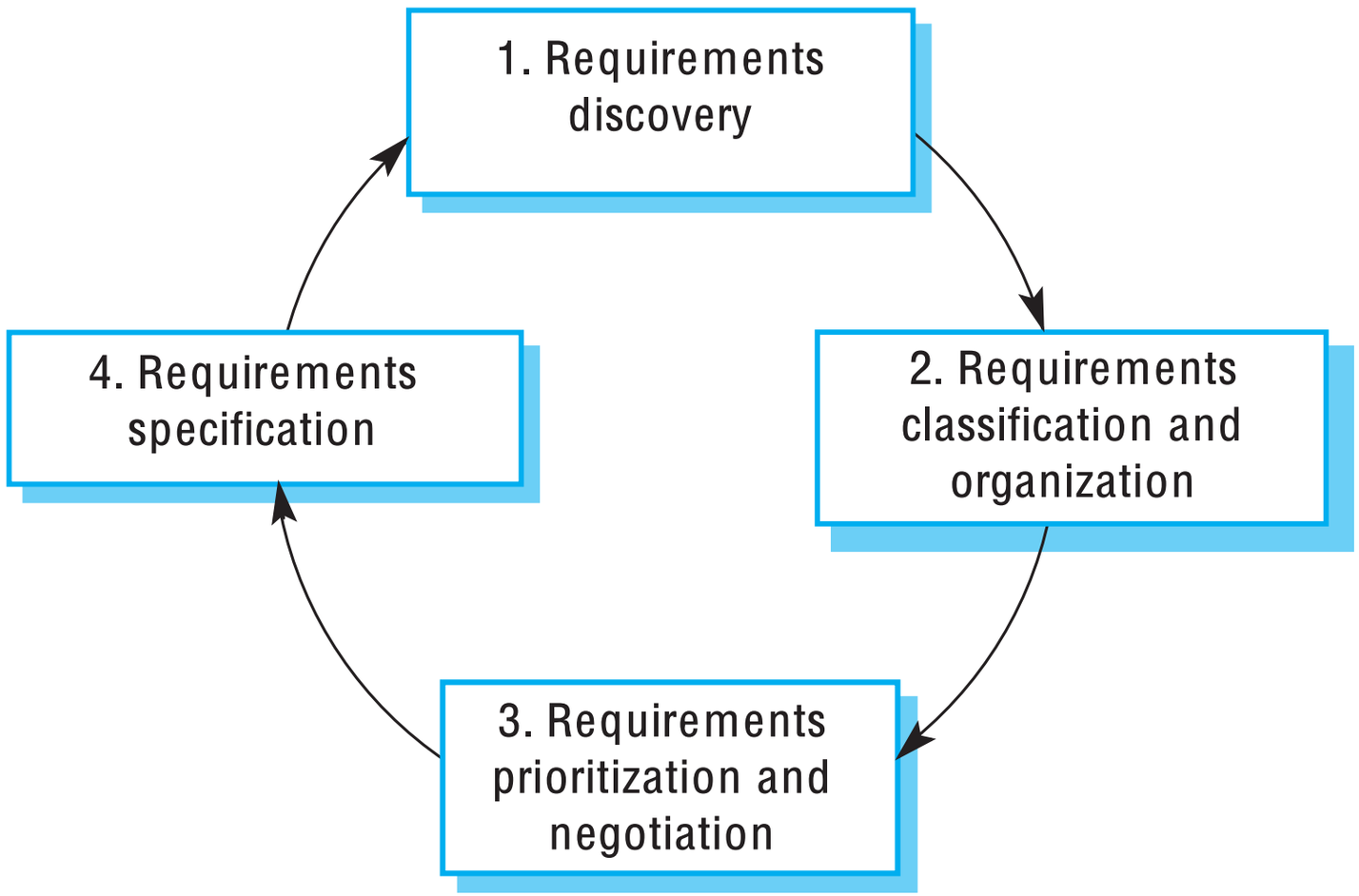
[1] Sommerville, Ian. “Software Engineering, 10th edition”. Pearson Education Limited, 2016.

A spiral view on requirements engineering



[1] Sommerville, Ian. “Software Engineering, 10th edition”. Pearson Education Limited, 2016.

The requirements elicitation and analysis process

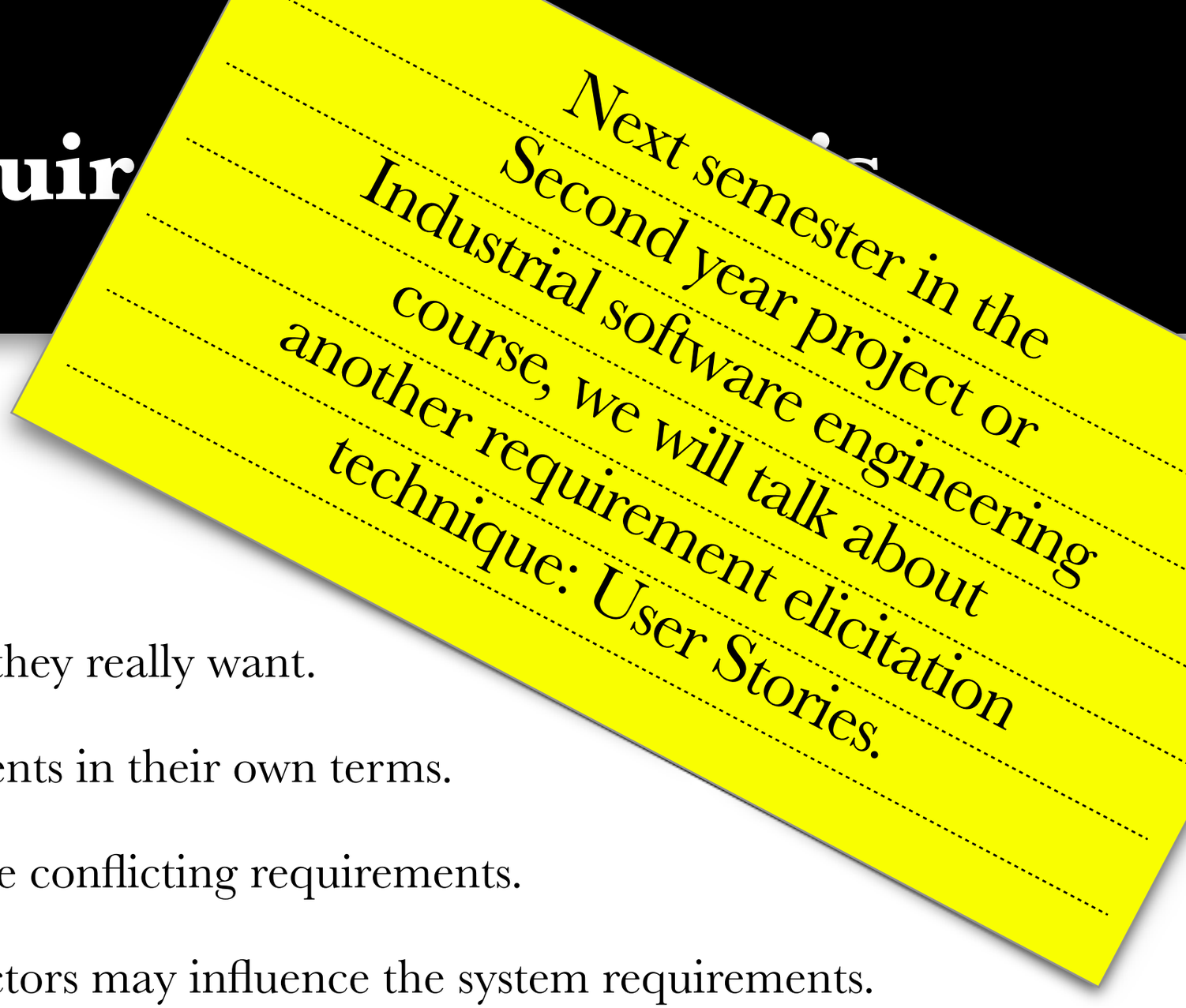


[1] Sommerville, Ian. “Software Engineering, 10th edition”. Pearson Education Limited, 2016.

Problems of requirements analysis

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

Problems of requir



- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 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—navigational paths and screen mock-ups
4. Glossary

Figure 4-16 Outline of the Requirements Analysis Document (RAD). Sections in *italics* are completed during analysis (see next chapter).

Summing up

Key Points I

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
  - Functional requirement: statements of the services that the system must provide
  - Non-functional requirements: constraints on the system being developed and the development process being used; often relate to the emergent properties of the system and therefore apply to the system as a whole.
  - Domain requirements: requirements on the system imposed by the application domain.
  - Requirements must be complete, consistent, clear, and correct as well as realistic, verifiable, and traceable

- The requirements engineering process
  - an iterative process including requirements elicitation, specification and validation.

- Requirements elicitation and analysis is an
  - iterative process that can be represented as a spiral of activities that includes: requirements discovery, requirements classification and organisation, requirements negotiation and requirements documentation.

- Requirements elicitation and analysis in OO
  - Different steps involved: identify Actors, Scenarios, and Use Cases; Relationships between Actors and Use Cases; Initial Analysis Objects; and Non-functional Requirements.

- The software requirements analysis document (RAD) is an agreed statement of the system requirements that should be organised so that both system customers and software developers can use it. Often used as the contract for the project.

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 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—navigational paths and screen mock-ups
4. Glossary

Figure 4-16 Outline of the Requirements Analysis Document (RAD). Sections in *italics* are completed during analysis (see next chapter).

This Lecture

- Literature
  - [OOSE] ch. 4
  - (Optional) [SE10] ch. 4

- Topics covered:
  - What is requirements engineering?
  - Functional, non-functional, and domain requirements
  - Requirements elicitation and analysis in OO
  - Managing requirements specification
  - The requirements engineering process