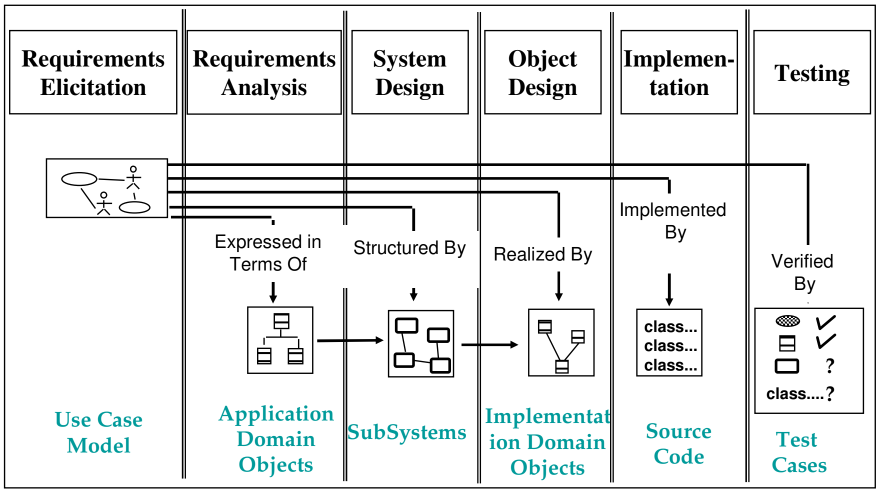
Requirements Engineering

Chapter 4

Overview

- Requirements Engineering
  - The first step in finding a solution for a data processing problem
  - Mutual understanding of the problem between developers and clients
  - Identify and document user requirements concerning function, performance, reliability, etc.
- Requirement:
  - a condition or capability needed by a user to solve a problem or achieve an objective (IEEE90a)
- Output:
  - requirements specification document
  - Contract for the customer
  - Starting point for design
- Iterative and cooperative process - analyzing, documenting, testing understanding of problem

Software Lifecycle Activities



Requirements Engineering

- Requirements Engineering
  - Elicitation
    - Results in the specification of the system that the client can understand (the “Problem Description”)
  - Analysis
    - An analysis model that developers can unambiguously interpret (the “Problem Specification”)
  - Elicitation is more challenging because you have to collaborate with people, usually with a different background
    - Focus of the textbook and this class: Scenarios and Use Cases can help bridge the gap
    - Start with a description of the functionality (Use case model). Then proceed by finding objects (object model).

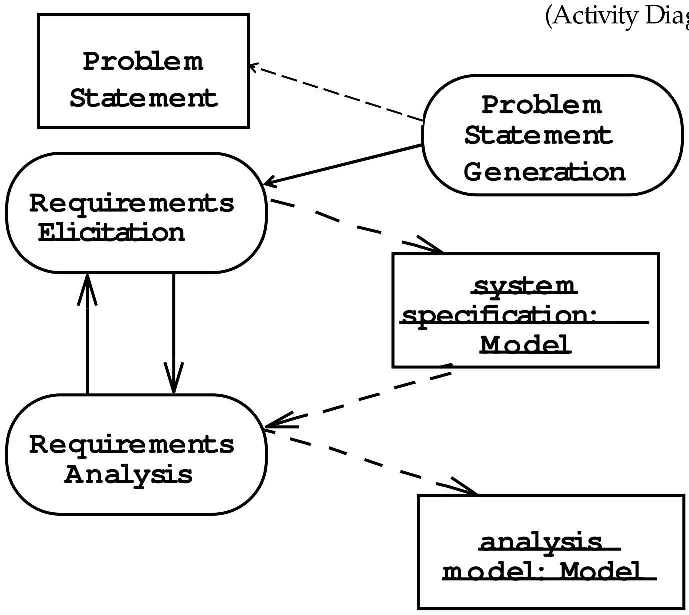
Defining the System Boundary is Often Difficult

What do you see here?

Crucial is the definition of the system boundary: What is inside, what is outside the system?

Products of Requirements Process

(Activity Diagram)



Requirements Elicitation Concepts

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

Functional Requirements

- Describe the interactions between the system and its environment independent of its implementation
  - Environment includes the user and any other external system with which the system interacts
  - System services expected by end users

Functional requirements do not focus on any implementation details!

Example Functional Requirements for SatWatch

- SatWatch is a wristwatch that displays the time based on its current location using GPS satellites to determine its location and convert this to a time zone.
- SatWatch adjusts the time and date as the watch owner crosses time zones and political boundaries. For this reason, SatWatch has no buttons or controls available to the user.
- During blackout periods, SatWatch assumes that it does not cross a time zone or a political boundary.
- 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, month, year).
- When political boundaries change, the watch owner may upgrade the software using the WebifyWatch device and a personal computer attached to the Internet.

Nonfunctional Requirements

- User visible aspects of the system not directly related to functional behavior; includes quality and constraints.
  - Usability
  - Reliability
  - Performance
  - Supportability
  - Implementation requirements
  - Interface requirements
  - Legal requirements
- Examples
  - As the SatWatch has no buttons, no software faults requiring the resetting of the watch should occur
  - SatWatch should measure time within 1/100th of a second over 5 years
  - SatWatch must be written using Java to comply with company policy
  - SatWatch complies with the software and physical interface defined by the WebifyWatch API

The Four C's

- Requirements are continuously validated with the client and user. This involves checking that the specs are:
  - Complete
    - All possible scenarios described, including exceptions
    - Ex: Specs do not specify boundary behavior within GPS accuracy limits
  - Consistency
    - Requirements do not contradict itself
    - Ex: WebifyWatch API requires user input, but there is a requirement of no buttons
  - Clarity
    - No ambiguity
    - Ex: Not clear if the watch deals with daylight savings time
    - Ex: Time expressed in local time or GMT?
  - Correctness
    - Correctly describes the features and environment
    - Ex: Specs indicate handling 24 time zones, but there are really > 24

Realism, Verifiability, Traceability

- Requirements specs must be
  - Realistic
    - Ex: "Product shall be error free"
  - Verifiable
    - Ex: "System should be user friendly"
    - Ex: "Response time should usually be less than 2 seconds"
    - Better ways to express requirements to be verifiable?
  - Traceable
    - Each requirement can be traced throughout the software development to its corresponding system functions, and each system function back to its set of requirements
  - Ranked
    - Rank requirements for importance or stability
      - Usually "must have", "desirable", "optional" sufficient
      - Stability could reflect likelihood of expected changes
      - Gives developers the direction to focus attention

Categories of Elicitation

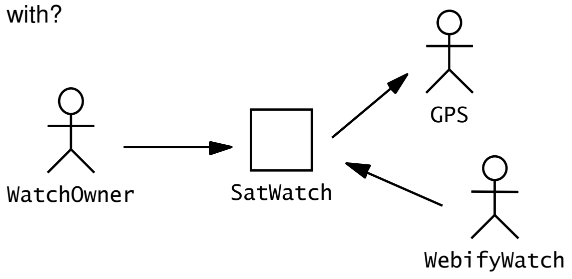
- Greenfield Engineering
  - Development from scratch, no prior system exists
- Reeingineering
  - Redesign and reimplementation of an existing system
- Interface Engineering
  - Redesign of the interface of an existing system
  - Legacy system left untouched except for its interface

Requirements Elicitation Activities

- Identifying actors
- Identifying scenarios
- Identifying use cases
- Refining use cases
- Identifying relationships among use cases
- Identifying non-functional requirements

Identifying Actors

- Actors represent external entities that interact with the system
  - May be human or an external system
  - E.g. for SatWatch, may be the human user, GPS satellites, WebifyWatch device, etc.
  - Questions to identify actors:
    - Which user groups are supported directly?
    - Which user groups perform secondary functions, e.g. maintenance?
    - What external hardware or software system will the system interact with?



Scenarios

- "A narrative description of what people do and experience as they try to make use of computer systems and applications" [M. Carrol, Scenario-based Design, Wiley, 1995]

- A concrete, focused, informal description of a single feature of the system used by specific actor(s).
  - Do not attempt to describe all possible situations or descriptions of decisions

- Scenarios can have many different uses during the software lifecycle
  - ***Requirements Elicitation***: As-is scenario, visionary scenario
  - ***Client Acceptance Test:*** Evaluation scenario
  - ***System Deployment:*** Training scenario.

Types of Scenarios

- As-is scenario:
  - Used in describing a current situation. Usually used in re-engineering projects. The user describes the system.
- Visionary scenario:
  - Used to describe a future system. Usually used in greenfield engineering and reengineering projects.
  - Can often not be done by the user or developer alone
- Evaluation scenario:
  - User tasks against which the system is to be evaluated.
- Training scenario:
  - Step by step instructions that guide a novice user through a system

How do we find scenarios?

- Don’t expect the client to be verbal if the system does not exist (greenfield engineering)
- Don’t wait for information even if the system exists
- Engage in a dialectic approach (evolutionary, incremental engineering)
  - You help the client to formulate the requirements
  - The client helps you to understand the requirements
  - The requirements evolve while the scenarios are being developed

Heuristics for finding Scenarios

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don’t rely on *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 software contractor
  - Expect resistance and try to overcome it

Example: FRIEND Accident Management System

- What needs to be done to report a “Cat in a Tree” incident?
- What do you need to do if a person reports “Warehouse on Fire?”
- Who is involved in reporting an incident?
- What does the system do, if no police cars are available? If the police car has an accident on the way to the “cat in a tree” incident?
- What do you need to do if the “Cat in the Tree” turns into a “Grandma has fallen from the Ladder”?
- Can the system cope with a simultaneous incident report “Warehouse on Fire?”

Scenario Example: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. She confirms her input and waits for an acknowledgment.
- 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.
- Alice received the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

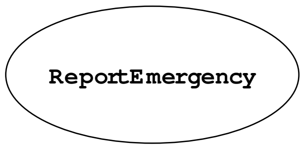
- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.
- Participating actors
  - Bob, Alice and John

Next goal, after the scenarios are formulated:

- Identify Use Cases
  - A scenario is an instance of a use case; i.e. a use case specifies all possible scenarios for a given piece of functionality
- Find all the use cases in the scenario that specifies all possible instances of how to report a fire
  - Example: "Report Emergency " in the first paragraph of the scenario is a candidate for a use case
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the Entry Condition
  - Describe the Flow of Events
  - Describe the Exit Condition
  - Describe Exceptions
  - Describe Special Requirements (Constraints, Nonfunctional Requirements

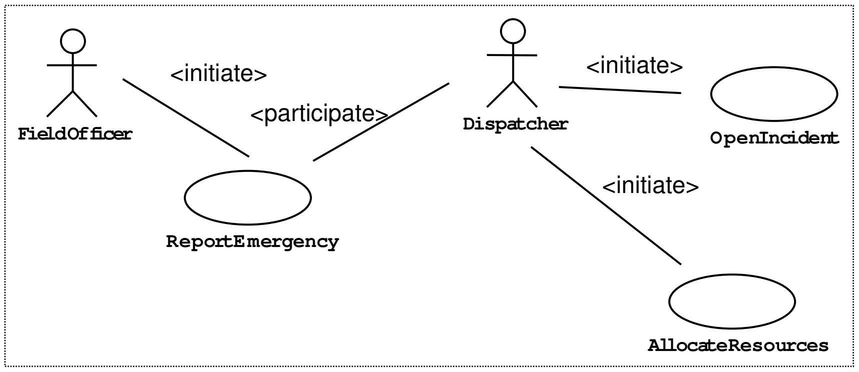
Use Cases

- A use case is a flow of events in the system, including interaction with actors
- It is initiated by an actor
- Each use case has a name
- Each use case has a termination condition
- Graphical Notation: An oval with the name of the use case



Use Case Model: **The set of all use cases specifying the complete functionality of the system**

Example: Use Case Model for Incident Management



Heuristics: How do I find use cases?

- Select a narrow vertical slice of the system (i.e. one scenario)
  - Discuss it in detail with the user to understand the user's preferred style of interaction
- Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - Discuss the scope with the user
- Use illustrative prototypes (mock-ups) as visual support
- Find out what the user does
  - Task observation (Good)
  - Questionnaires (Bad)

Use Case Example: ReportEmergency

- Use case name: ReportEmergency
- Participating Actors:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
- Exceptions:
  - The FieldOfficer is notified immediately if the connection between her terminal and the central is lost.
  - The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the central is lost.
- Flow of Events: **on next slide.**
- Special 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.

Use Case Example: ReportEmergency Flow of Events

- The **FieldOfficer** activates the “Report Emergency” function of her terminal. FRIEND responds by presenting a form to the officer.

- The FieldOfficer fills 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, at which point, the **Dispatcher** is notified.

- 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 emergency report.

- The FieldOfficer receives the acknowledgment and the selected response.

Another Use Case Example: Allocate a Resource

- Actors:
  - ***Field Supervisor:*** This is the official at the emergency site....
  - ***Resource Allocator:*** The Resource Allocator is responsible for the commitment and decommitment of the Resources managed by the FRIEND system. ...
  - ***Dispatcher:*** A Dispatcher enters, updates, and removes Emergency Incidents, Actions, and Requests in the system. The Dispatcher also closes Emergency Incidents.
  - ***Field Officer:*** Reports accidents from the Field

Another Use Case Example: Allocate a Resource

- *Use case name:* AllocateResources
- *Participating Actors:*
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
  - Resource Allocator
  - Field Supervisor
- *Entry Condition*
  - The Resource Allocator has selected an available resource.
  - The resource is currently not allocated
- *Flow of Events*
  - The Resource Allocator selects an Emergency Incident.
  - The Resource is committed to the Emergency Incident.
- *Exit Condition*
  - The use case terminates when the resource is committed.
  - The selected Resource is now unavailable to any other Emergency Incidents or Resource Requests.
- *Special Requirements*
  - The Field Supervisor is responsible for managing the Resources

Order of steps when formulating use cases

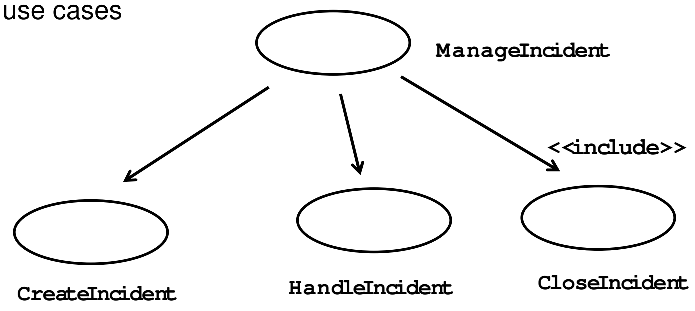
- First step: name the use case
  - Use case name: ReportEmergency
- Second step: Find the actors
  - Generalize the concrete names ("Bob") to participating actors ("Field officer")
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)
- Third step: Then concentrate on the flow of events
  - Use informal natural language

Use Case Associations

- A use case model consists of use cases and use case associations
  - A use case association is a relationship between use cases
- Important types of use case associations: Include, Extends, Generalization
- Include
  - A use case uses another use case ("functional decomposition")
- Extends
  - A use case extends another use case, just like in OOP
- Generalization
  - An abstract use case has different specializations

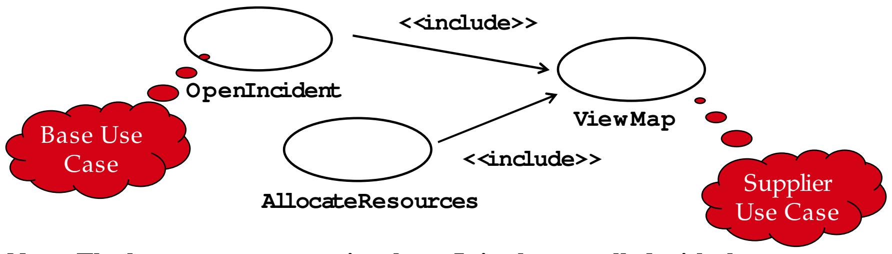
<<Include>>: Functional Decomposition

- Problem:
  - A function in the original problem statement is too complex to be solvable immediately
- Solution:
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases



<<Include>>: Reuse of Existing Functionality

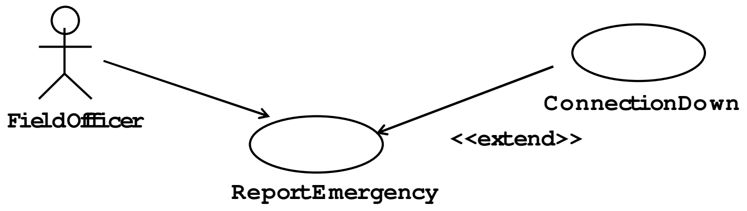
- Problem:
  - There are already existing functions. How can we *reuse* them?
- Solution:
  - The *include association* from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B ("A delegates to B")
- Example:
  - The use case "ViewMap" describes behavior that can be used by the use case "OpenIncident" ("ViewMap" is factored out)



Note: The base case cannot exist alone. It is always called with the supplier use case

<Extend>> Association for Use Cases

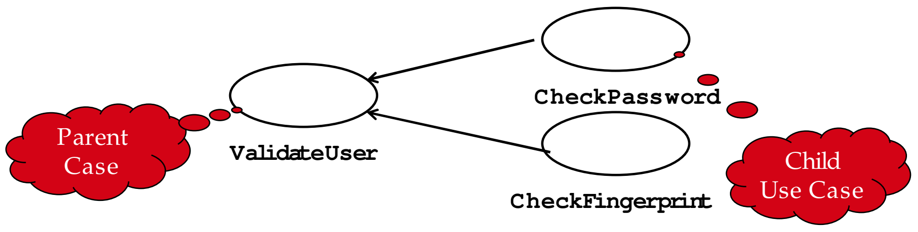
- Problem:
  - The functionality in the original problem statement needs to be extended.
- Solution:
  - An *extend association* from a use case A to a use case B indicates that use case B is an extension of use case A.
- Example:
  - The use case “ReportEmergency” is complete by itself , but can be extended by the use case “ConnectionDown” for a specific scenario in which the user requires special help



Note: The base use case can be executed without the use case extension in extend associations.

Generalization association in use cases

- Problem:
  - You have common behavior among use cases and want to factor this out.
- Solution:
  - The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.
- Example:
  - Consider the use case “ValidateUser”, responsible for verifying the identity of the user. The customer might require two realizations: “CheckPassword” and “CheckFingerprint”



How to Specify a Use Case (Summary)

- 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
- Special Requirements
  - Nonfunctional Requirements, Constraints

Identifying Nonfunctional Requirements

- Usability
  - What is the level of expertise of the user?
  - What user interface standards are familiar?
  - What documentation should be provided?
- Reliability
  - How reliable, available, and robust should the system be?
  - Are there safety requirements?
  - Are there security requirements?
  - How should the system handle exceptions?
- Performance
  - How responsive?
  - Are any user tasks time critical?
  - Concurrent users?
  - How large is a typical data store for comparable systems?
  - What is the worst latency possible?
- Supportability
  - What are foreseen extensions?
  - Who maintains the system?
  - Future plans for ports?

Identifying Nonfunctional Requirements

- Implementation
  - Are there constraints on the hardware?
  - Are there constraints imposed by maintenance or testing?
- Interface
  - Should the system interact with other systems?
  - How will data be imported/exported?
  - What standards are in place?
- Packaging
  - Who installs the system?
  - Time constraints on install?
  - Other software dependencies?
- Legal
  - Licensing?
  - Liability issues with system failures?
  - Royalties or licensing fees?

Other Elicitation Techniques

- Aside from scenario-based elicitation:
- Asking
  - Interviews, Brainstorming, questionnaires, group discussions, focus groups
    - Outspoken users may bias the outcome
    - Delphi technique: written information exchanged iteratively until consensus reached
- Task Analysis
  - Identify and analyze tasks
  - Form into a hierarchy of subtasks carried out by people working in the domain
  - Stop at the point when users “refuse” to decompose tasks further

Other Elicitation Techniques

- Ethnography
  - Study the people (users) in their natural settings
  - Analyst becomes an apprentice
  - Anthropologist in the jungle
- Form analysis
  - Analyze any forms being used to gain information about the problem
  - Forms provide info about domain data objects, properties, and interrelations
- Natural language descriptions
  - Operating instructions in writing
  - Good for background but natural languages notorious for leading to ambiguities
  - Often natural language descriptions are not kept up to date

Other Elicitation Techniques

- Derivation from an existing system
  - Use current system to formulate requirements of the new system
  - Use analogous system from another organization to derive the requirements (domain analysis)
  - Useful for identifying reusable components, concepts and structures
- Prototyping
  - Prototype constructed to elicit requirements as discussed earlier

Requirements Specification

- Requirements Specification
  - End product of the requirements engineering phase
  - Document specifies the system requirements
    - Informal to precise, mathematical representation
    - Serves as a mechanism to communicate with users
      - May even take on different forms for different audiences
    - Starting point for design phase

IEEE Standard 830

- IEEE Standard 830 gives a template for structuring requirements
- Textbook uses a variant of IEEE 830 focused around scenarios and use cases
- See webpage for details and a similar template
  - http://wwwbruegge.in.tum.de/OOSE/RequirementsAnalysisDocumentTemplate
- Should add ranking of requirements

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, acronyms, abbreviations
   - 1.4 References
   - 1.5 Overview
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***
     - 3.5.5 User interface – navigational paths and screen mock-ups

Discussed in chapter 5

Summary of Lessons

- Requirements elicitation involves constant switching between perspectives
  - High-level vs. detailed
  - Client vs. Developer
  - Activity vs. Entity
- Requirements elicitation requires a substantial involvement from the client
- Scenarios:
  - Great way to establish communication with client
  - Different types of scenarios: As-Is, visionary, evaluation and training
  - Use cases: Abstraction of scenarios
- Developers should not assume that they know