Requirements Engineering

Lecture 7

Prof. Dr. Alexander Pretschner

Chair of Software & Systems Engineering
TUM Department of Informatics
Technical University of Munich

Orientation

Recap:

- Lecture 6: Goals and Non-functional Requirements
- How can we measure NFRs
- Functional vs. Non-Functional
- Ethics in the RE process

Coming up:

- Functional Requirements
- Domain Models
- Usage Models
- Use Cases
- User Stories
- Function Hierarchies

How can we separate Functional and Non-Functional Requirements?

Example 1 (NFR): “The Scooter app backend must be capable of serving 10,000 clients simultaneously with a response time under 600ms.”

Example 2 (FR): “The user should be able to log into the Scooter app with their personal account.”

Definition: Functional Requirements

Definition Functional Requirements:

External system behavior, characterized by stimulus-response (input-output) pairs. [Davis93]
(Then everything else becomes a non-functional requirement.)

Functional Requirements:

- Often in the form of *"The system should do [xy]".*
- Justified by objectives and collected from various sources (see Lecture 4)
- Specify product features of a system

Critical view

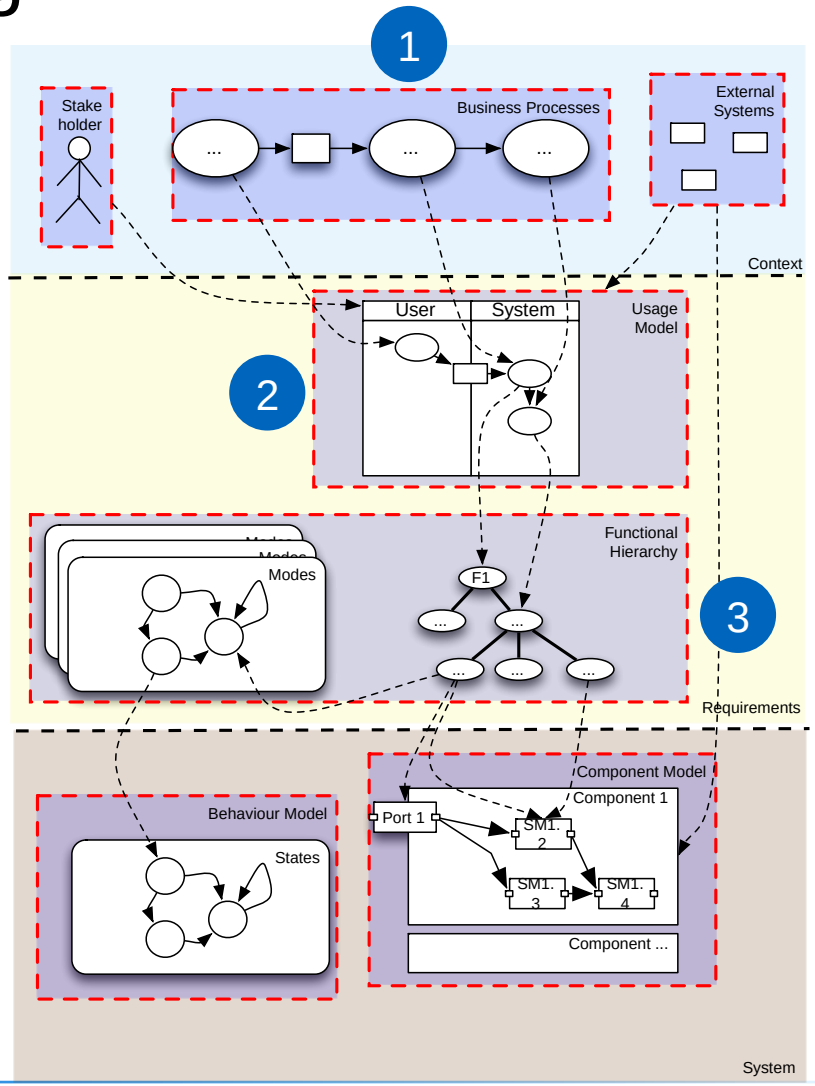
Definition and delimitation between “functional“ and “non-functional” requirements is difficult
→ On which levels of abstraction and in which context can functionality be considered?
→ How can behavior systematically be recorded and described in a structured way?

Levels and Concepts of Behavioral Modeling

1. Analysis of Business Processes
   - Context of the system
   - *Which technical processes should be supported by the system?*

2. Structured specification of Use Cases and Scenarios
   - Behavioral modeling off perspective of the user ("Black Box")
   - *How should the system be used by (external) actors?*

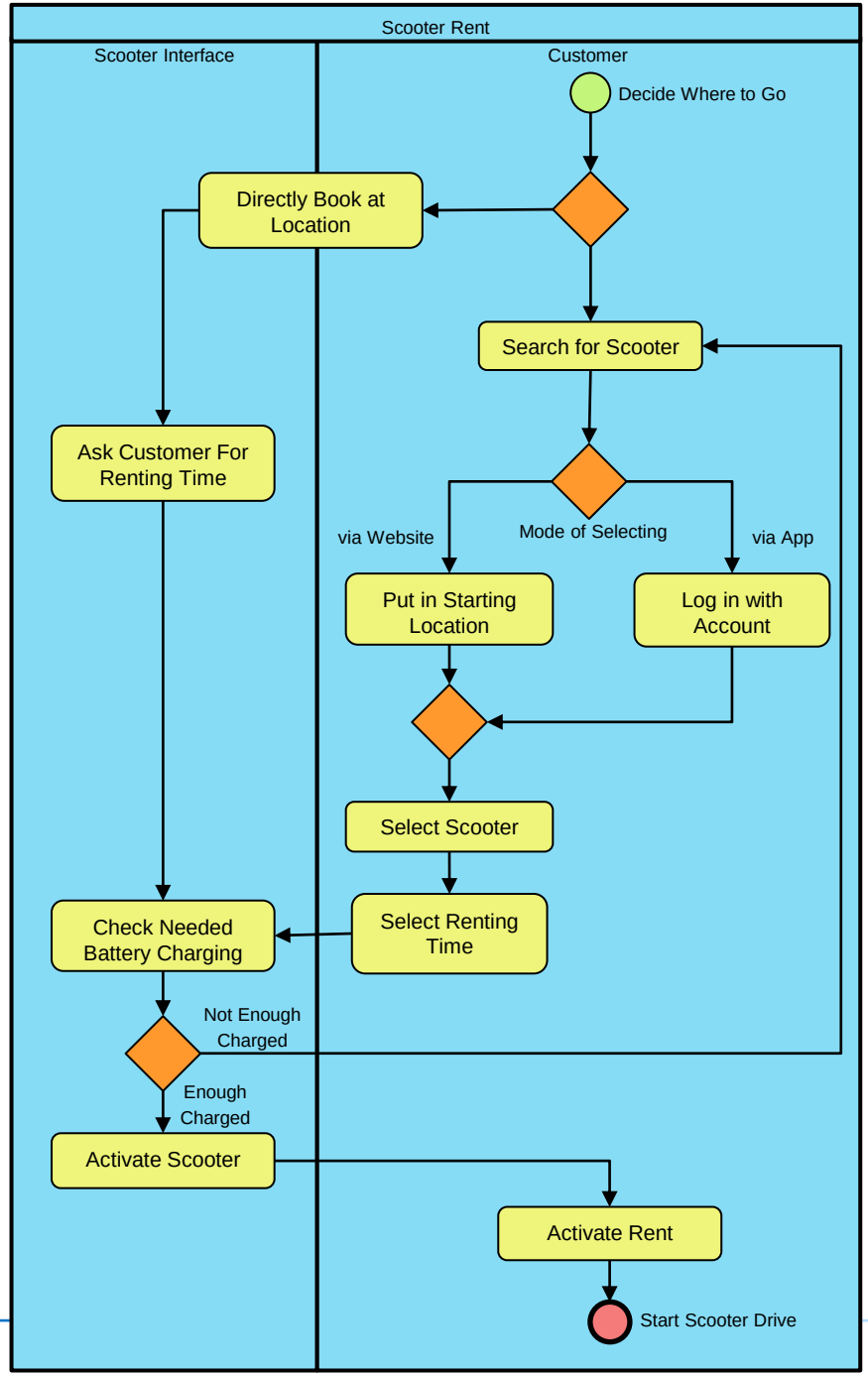
3. Step-by-step refinement to system specification using Function Hierarchies
   - Specification of functions, interfaces, system behavior
   - *Which functions are required?*



Example: Business Process Model

1 Analysis of Business Processes

- Tasks and causal dependencies
- Roles and responsibilities
- Involved business objects



Example Business Process:
Renting a scooter.

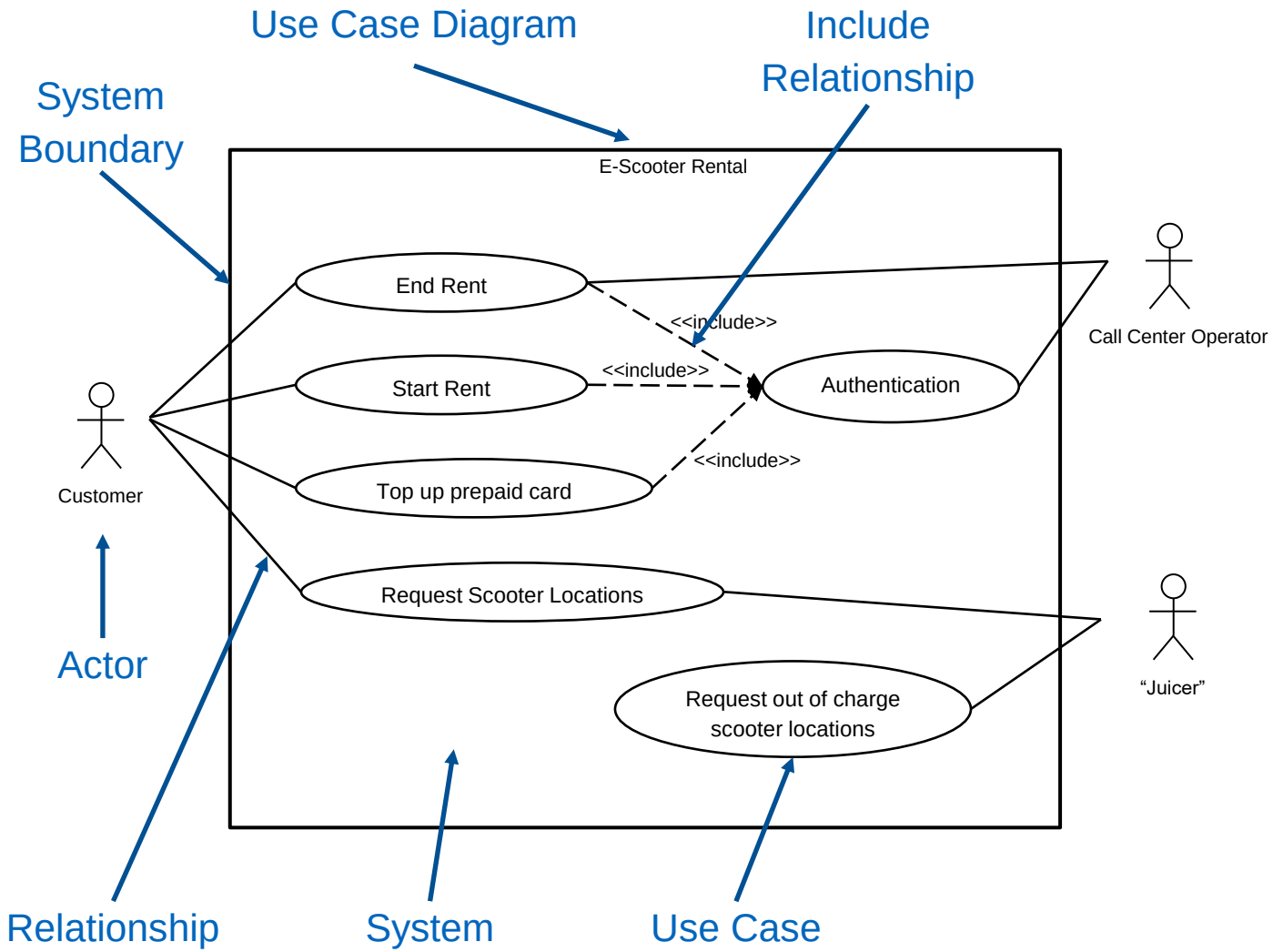
Example: Usage Model

2 Structured specification of Use Cases and Scenarios

- Derivation of tasks to be realized in interaction with the system
- Identification of Use Cases
- Coordination and modeling of interaction scenarios

In Use Case Diagrams Relationships can be:

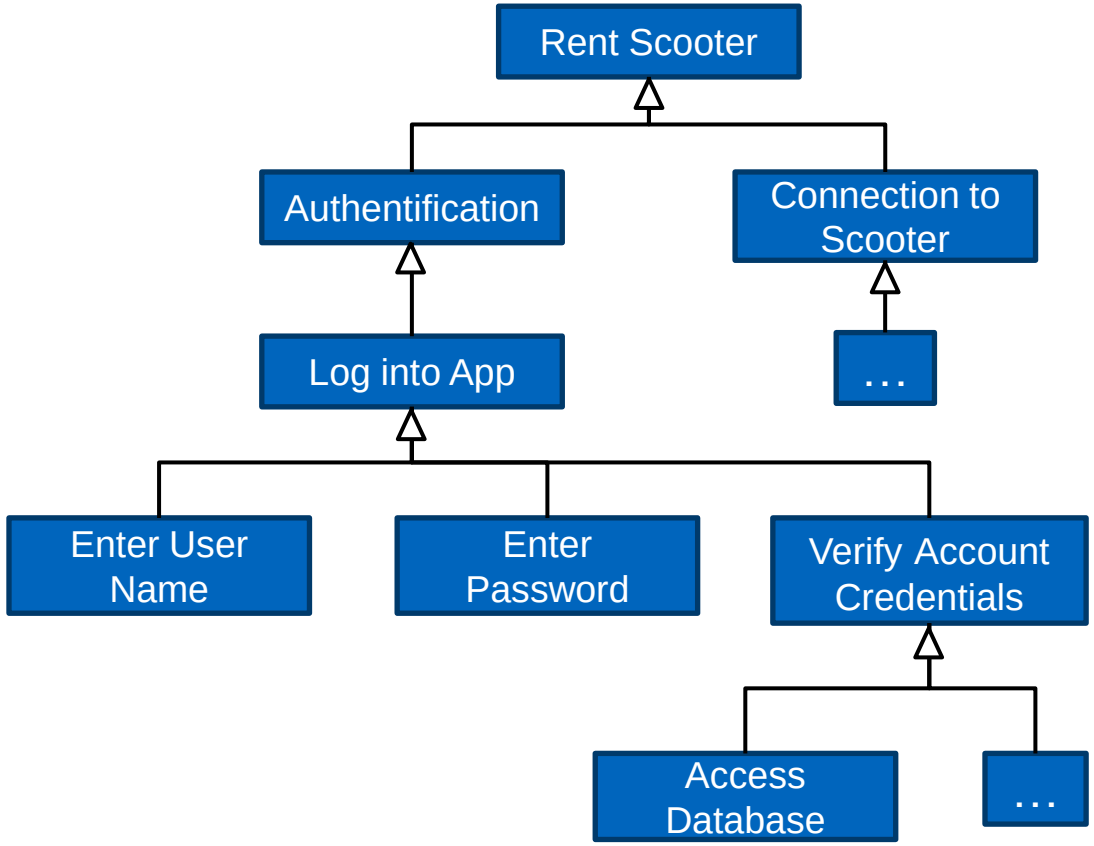
- Generalization
- Include
- Extend
- Extension point



Example: Function Hierarchy

3 Specification using Function Hierarchies

- Identification of system functions based on scenarios
- Refinement of the functions and identifications of interactions
- Specification of the interfaces



What is a Use Case?

Definition Use Case:

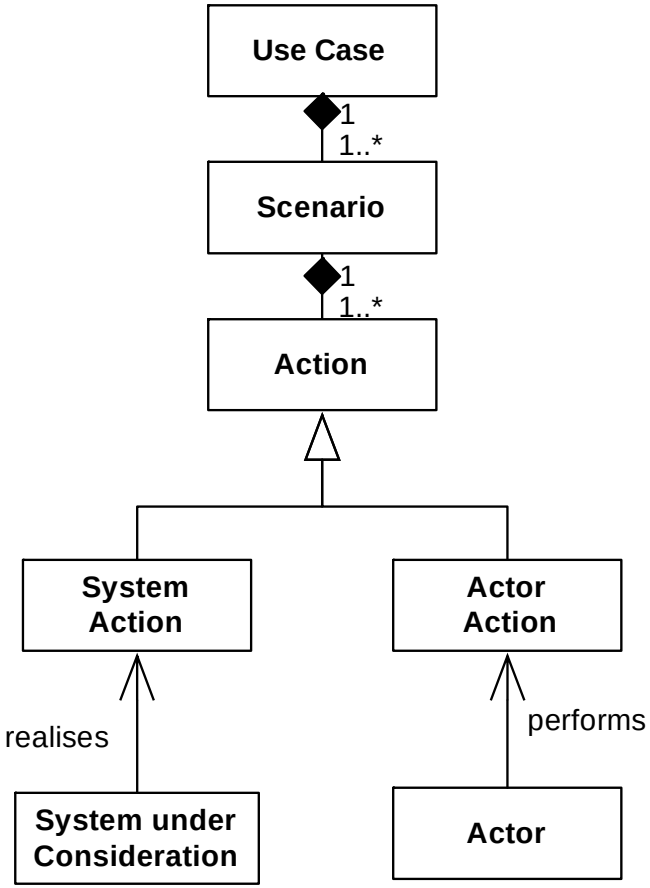
A use case bundles all possible scenarios that can occur when an actor tries to achieve a certain technical goal with the help of the system under consideration.

Notation: structured text, use case diagram (UML), ...

Definition Scenario:

An ordered set of interactions between partners, usually between a system and a set of external actors [Glinz06].

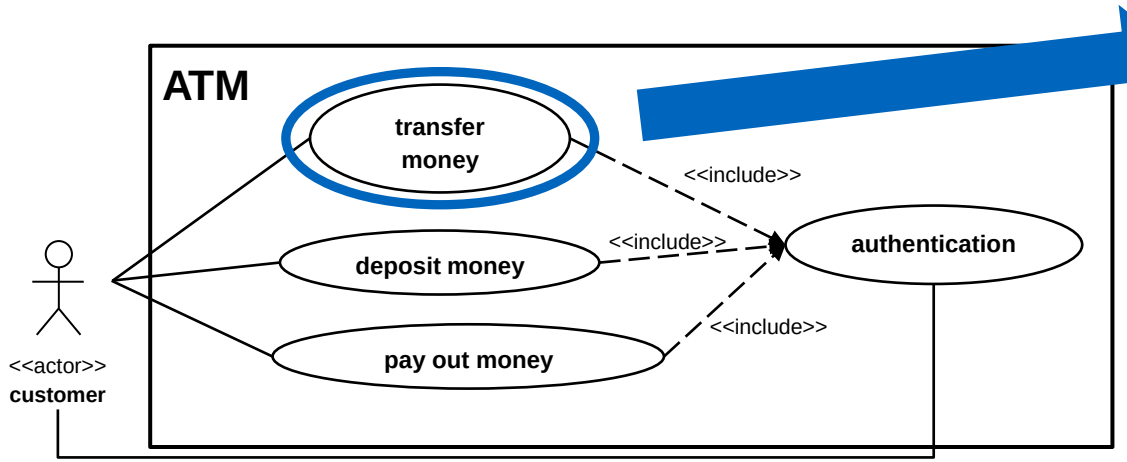
Notation: structured text, sequence/interaction diagrams
(e.g., UML, Message Sequence Charts (MSCs), AutoFocus EETs, ...)



Difference between a Use Case and a Scenario?

- A use case is a set of scenarios
- A scenario is an ordered set (sequence) of interactions (functional requirements)
- Use cases can be described with the Use Case Template of Cockburn
- The interactions in a scenario can be unfolded into smaller and smaller interactions (e.g., “buy a coke” and “put coin into machine”)
- Some scenarios end with success, some end with failure

Use Cases and Scenarios (1/2)



1

USE CASE #	<the name is the goal as a short active verb phrase>	
Context of Use	<a longer statement of the context of use if needed>	
Scope	<what system is being considered black box under design>	
Level	<one of summary, primary task, subfunction>	
Primary Actor	<a role name for the primary actor, or a description>	
Stakeholder and Interests	Stakeholder	Interest
	<stakeholder name>	<put here the interest of the stakeholder>
	<stakeholder name>	<put here the interest of the stakeholder>
Preconditions	<what we expect is already the state of the world>	
Minimal Guarantees	<the interests as protected on any exit>	
Success Guarantees	<the interests as satisfied on a successful ending>	
Trigger	<the action upon the system that starts the use case>	
Description	Step	Action
	1	<put here the steps of the scenario from trigger to goal delivery and any cleanup after>
	2	<...>
	3	
Extensions	Step	Branching Action
	1a	<condition causing branching> : <action or name of sub use case>
Technology and Data Variations		
	1	<list of variations>

2

A **Use Case** is a state- and property-oriented structuring of the system behavior which

1. has purpose and context (business processes).
2. collects several scenarios.

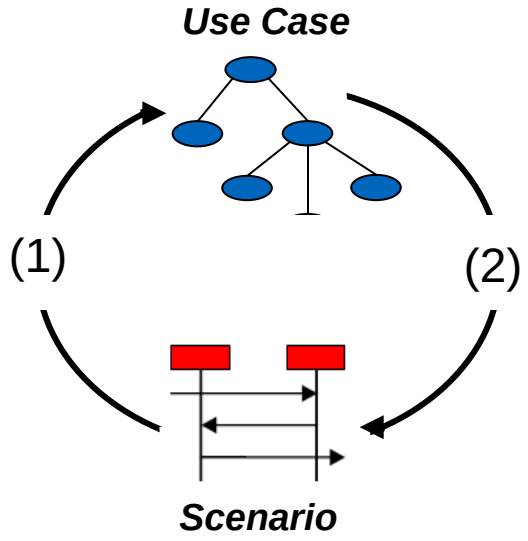
Use Cases and Scenarios (2/2)

Use Cases summarize a number of scenarios for a targeted system usage.

- Use Case:

  Task, objective, performance for the "users", task structuring, causal dependencies (also pre- and post-conditions)
- Scenario:

  Sequence of events (work steps, events, interactions)



Iterative development
(see Refinement and abstraction of goals - VL 5)

(1) Combine scenarios into tasks and structure

(2) Collect task-related scenarios, "play through" and analyze

Use Cases vs. Scenarios vs. Functional Requirements

Use Case:

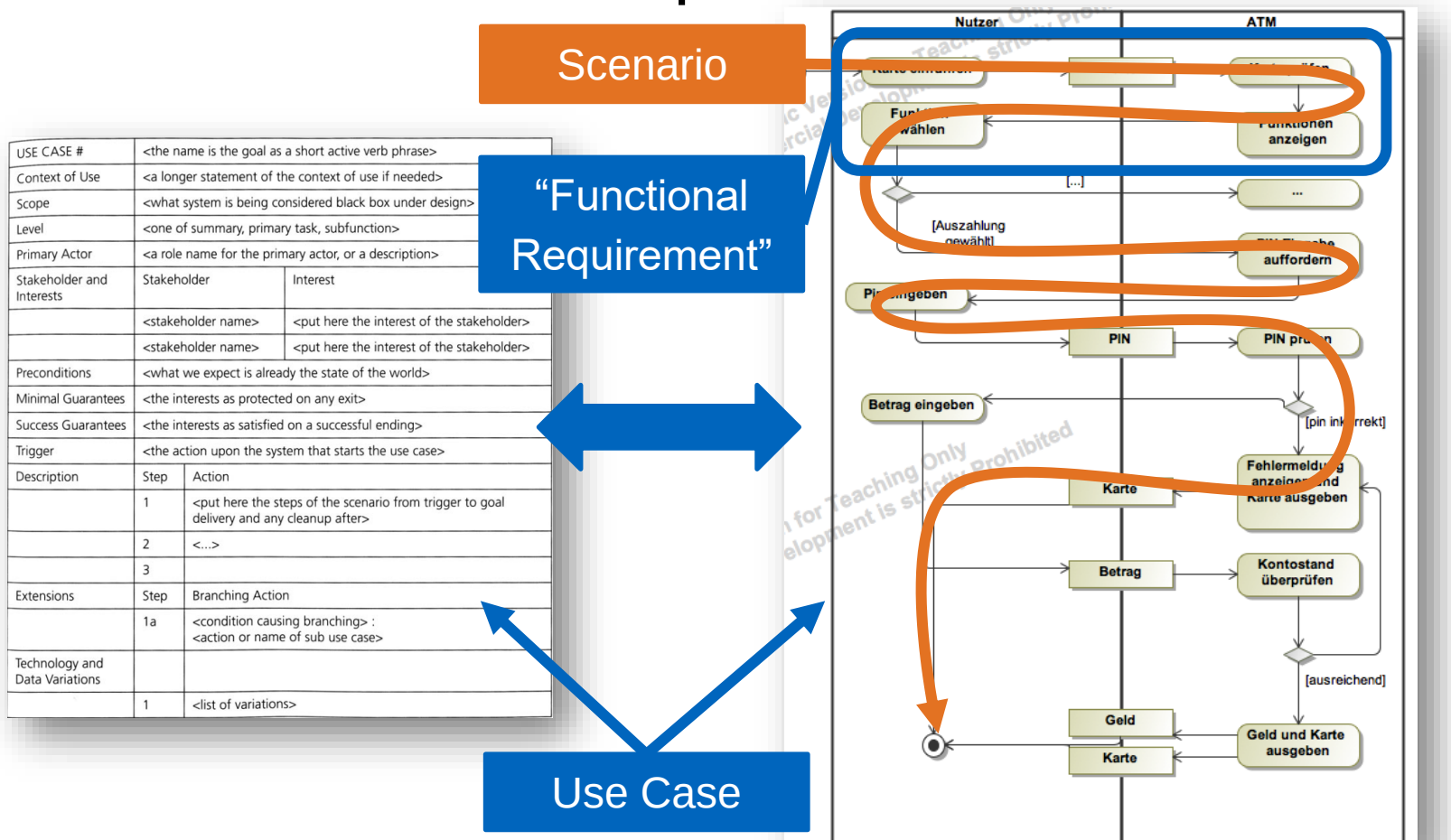
Set of all scenarios

Scenario:

A concrete interaction process

Functional Requirement:

A (system)-response to a stimulus (take with a grain of salt!)



USE CASE #	<the name is the goal as a short active verb phrase>	
Context of Use	<a longer statement of the context of use if needed>	
Scope	<what system is being considered black box under design>	
Level	<one of summary, primary task, subfunction>	
Primary Actor	<a role name for the primary actor, or a description>	
Stakeholder and Interests	Stakeholder	Interest
	<stakeholder name>	<put here the interest of the stakeholder>
	<stakeholder name>	<put here the interest of the stakeholder>
Preconditions	<what we expect is already the state of the world>	
Minimal Guarantees	<the interests as protected on any exit>	
Success Guarantees	<the interests as satisfied on a successful ending>	
Trigger	<the action upon the system that starts the use case>	
Description	Step	Action
	1	<put here the steps of the scenario from trigger to goal delivery and any cleanup after>
	2	<...>
	3	
Extensions	Step	Branching Action
	1a	<condition causing branching> : <action or name of sub use case>
Technology and Data Variations		
	1	<list of variations>

Specification of Use Cases

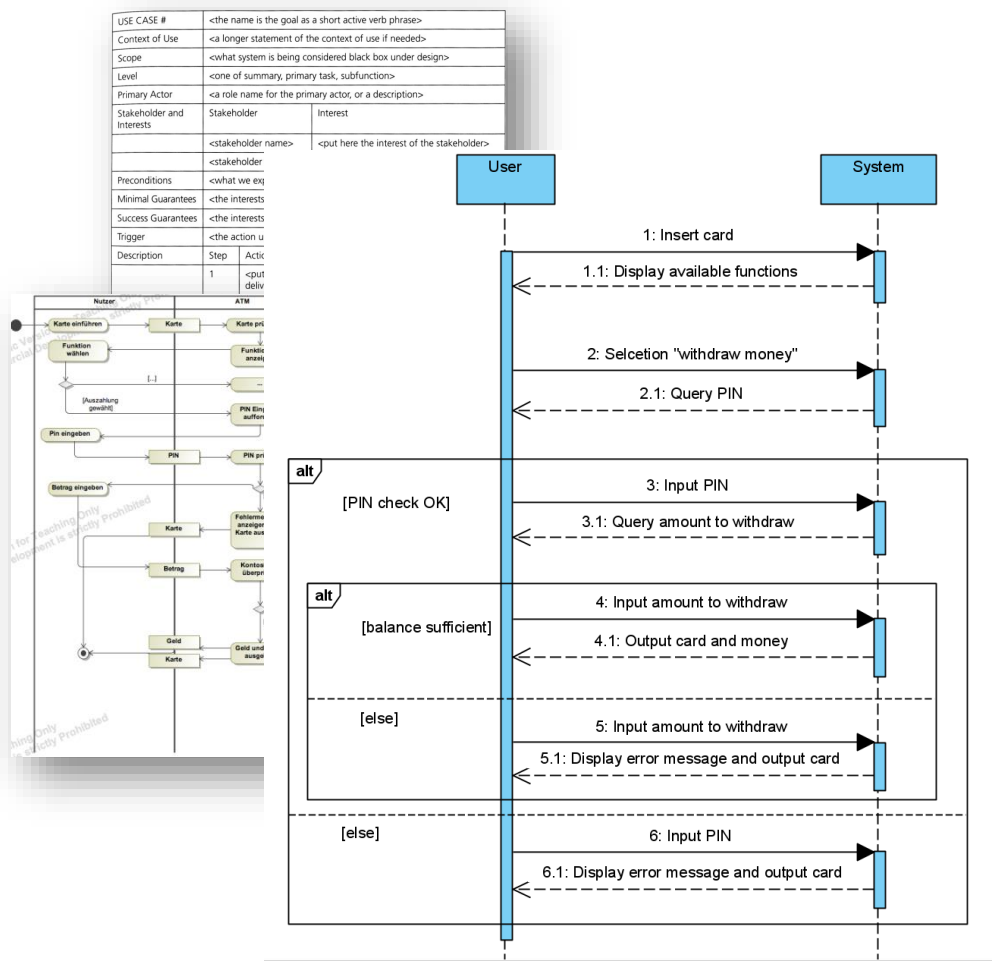
Modeling of Use Cases

- Using structured text
- By using models (Activity Diagrams, Message Sequence Charts)

Core Components

Context of use (remember design thinking: "user journey" extends this idea)

- Primary actors and their goals
- Precondition: Assumptions about system environment and the initial situation
- Postcondition: Utilization goal and target situation
- Scenarios: Work steps and required Interaction between actors (users, neighboring systems) and system
  → Control Flow
  → Scenarios (instances per use case)



Use Case Template of Cockburn

A scenario is a sequence of goal-achieving actions

The main success scenario is the most common scenario:
- Is a simple/typical story without too much complexity
- Cockburn: Description without Extension or Sub-Variations

The use case collects all the scenarios which are related to the goal of the primary actor:
- The "set of possible sequences" is called a "use case“
- The "sequence of (concrete) interactions" is called a "scenario" in the use cases
- A scenario consists of steps, each step is an action.

→ A use case is the collection of possible scenarios

USE CASE 5	Buy Goods	
Goal in Context	**Buyer issues request directly to our company, expects goods shipped and to be billed.**	
Scope & Level	**Company, Summary**	
Preconditions	**We know Buyer, their address, etc.**	
Success End Condition	**Buyer has goods, we have money for the goods.**	
Failed End Condition	**We have not sent the goods; Buyer has not spent the money.**	
Primary, Secondary Actors	**Buyer, any agent (or computer) acting for the customer. Credit card company, bank, shipping service**	
Trigger	**purchase request comes in.**	
DESCRIPTION	**Step**	**Action**
	1	Buyer calls in with a purchase request
	2	Company captures buyer's name, address, requested goods, etc.
	3	Company gives buyer information on goods, prices, delivery dates, etc.
	4	Buyer signs for order.
EXTENSIONS	**Step**	**Branching Action**
	3a	Company is out of one of the ordered items: 3a1. Renegotiate order
	4a	Buyer pays directly with credit card: 4a1. Take payment by credit card (use case 44)
SUB-VARIATIONS	**Step**	**Branching Action**
	1	Buyer may use phone in, fax in, use web order form, electronic interchange

Discussion: Scenarios and Use Cases

Write the main success scenario for one E-Scooter use case.

Write a scenario for one of the E-Scooter use cases.

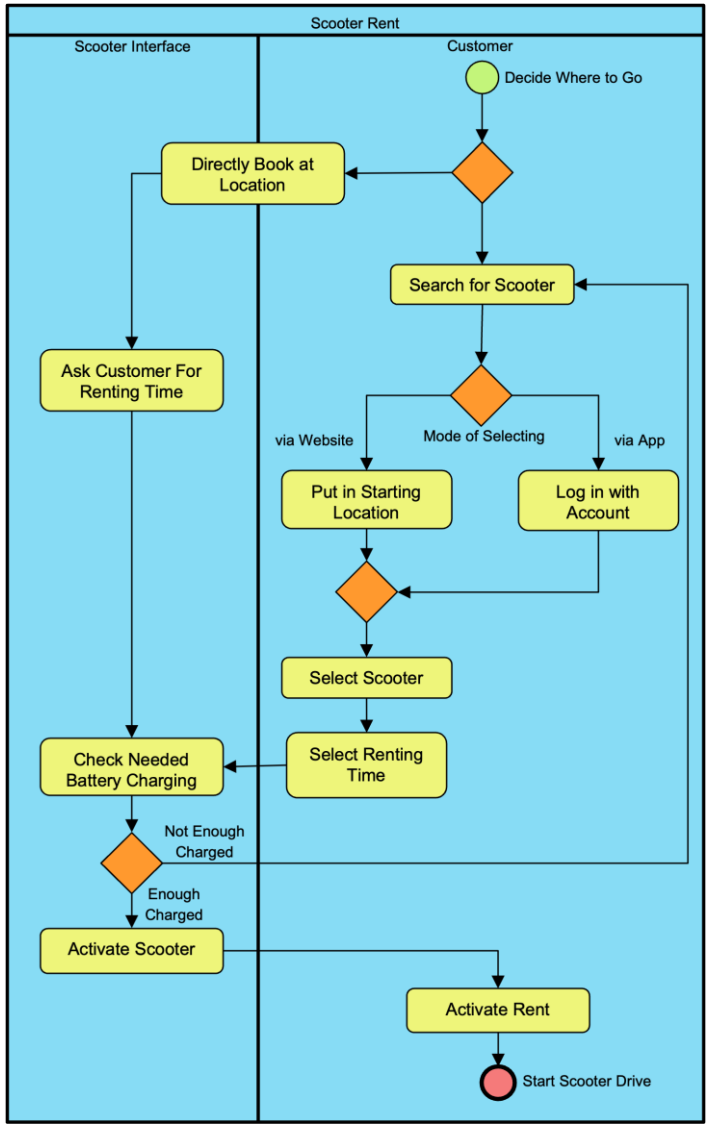
What is a Business Process?

Definition Business Process:

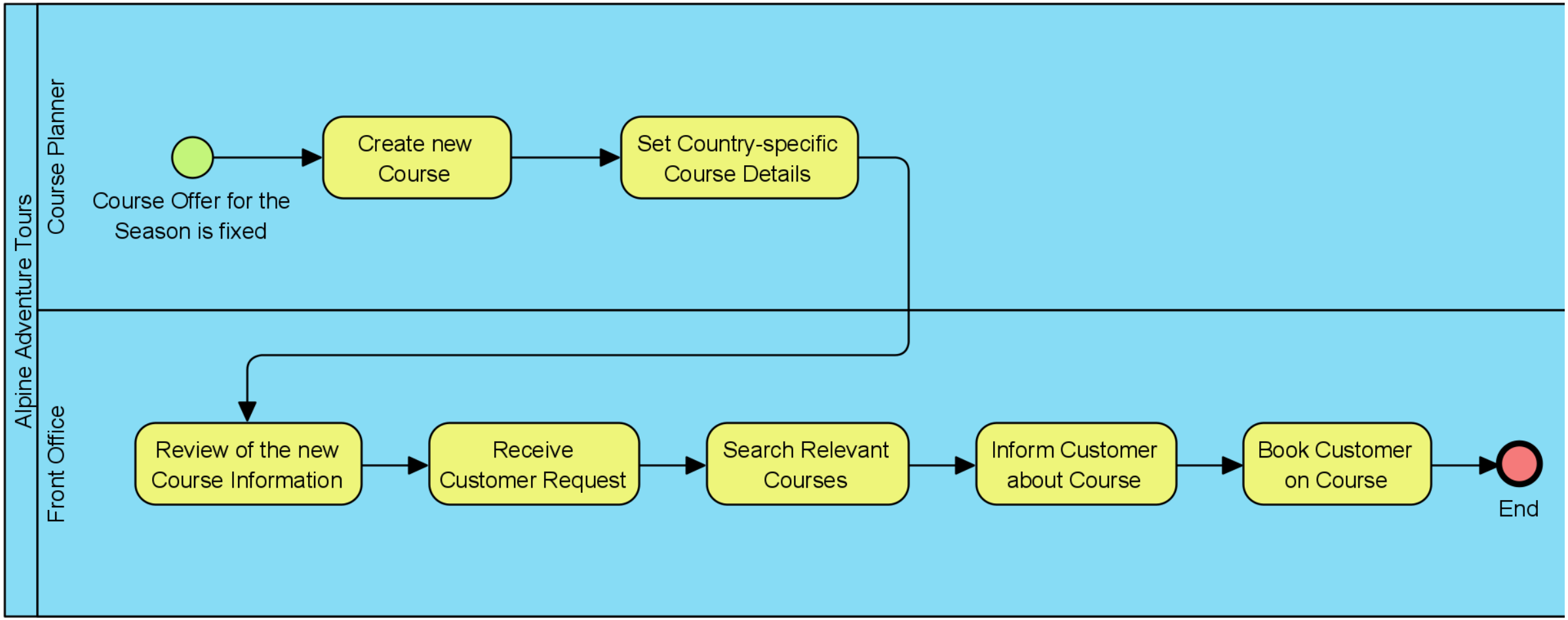
A business process is a sequence of work steps for the fulfillment of a goal.

Concepts (simplified)

- Function/Aim (Purpose/Intention - Why?)
- Process steps (tasks/activities) and causal dependencies/sequence (interaction sequence - what? when? how?)
- Business objects (with what?)
- Distribution of tasks to involved roles (by whom?)
- organizational units, system users
- Systems, processes, HW/SW components of the system environment



Example of a Business Process according to BPMN

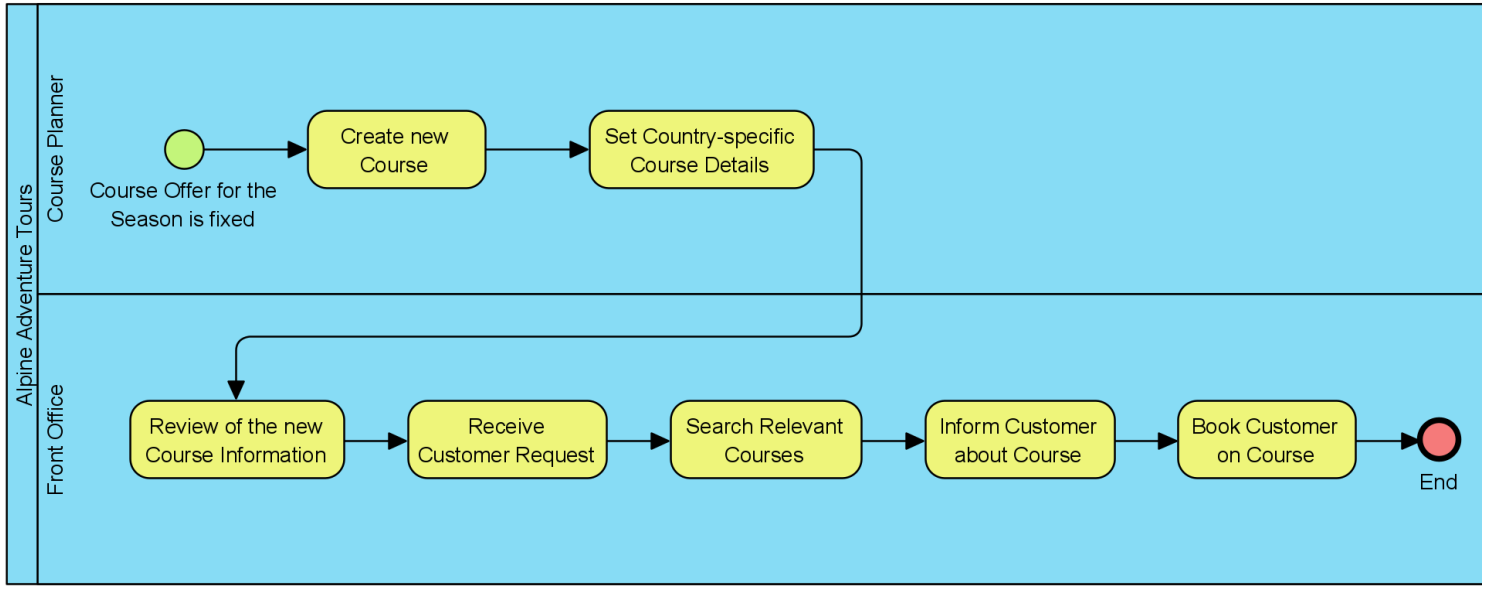


From Business Process Instances to Business Processes

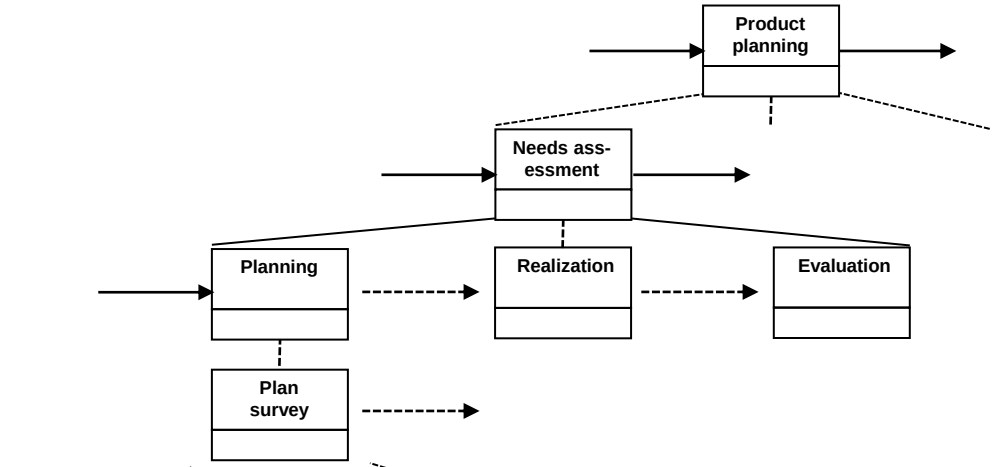
- A business process is a set of business process instances
- Describing a set of process instances is often difficult

→Therefore: Description of selected, representative instances, e.g.:

- *Create new Course*
- *Review Information*
- *Book Customer*



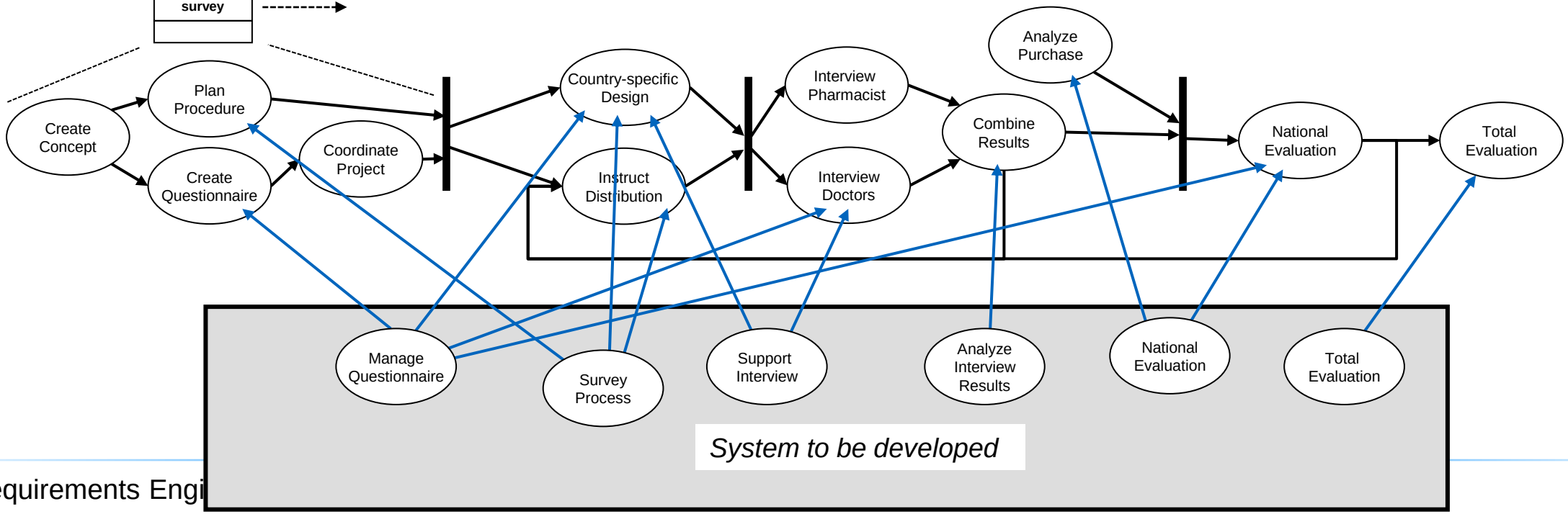
From Business Processes to Functions: Basic Idea



- Which actor (or role) leads which steps?
- Which steps are supported by the system?

→ Identification of functions that must be available in a system.

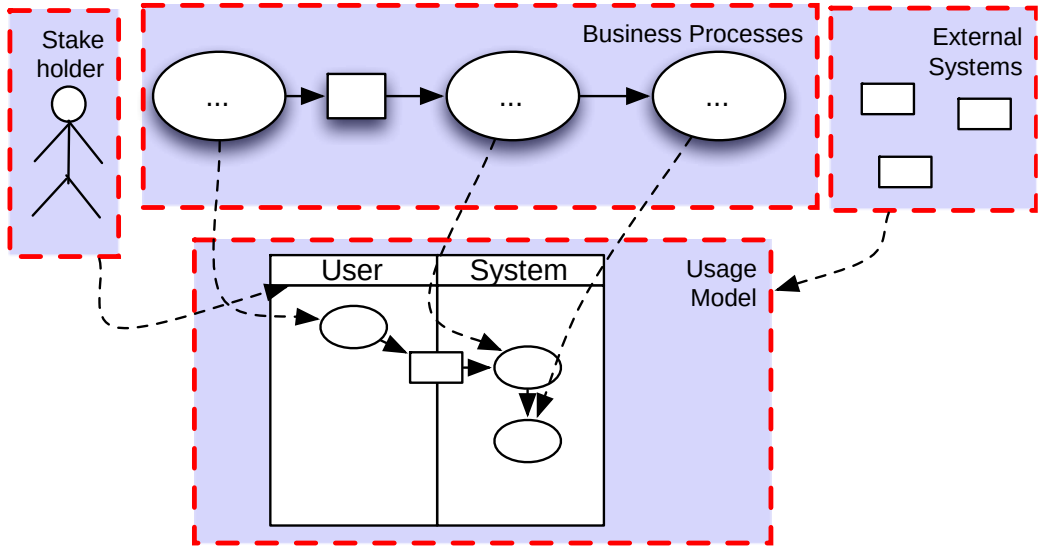
→ The way in which this use takes place can be recorded in use cases (see following slide).



Use Cases: From Business Processes to Functions

Capture and Model Business Processes

1. Identification of tasks and steps (building structures)
   → Surveys
2. Specification of partial steps, causal order and actors
   → Surveys, observations, analysis of target models
3. Identification of business objects



Analyze Processes and Identify Functions

1. Identification of use case candidates
   → tasks be performed by the system, which by external systems, which remain manual?
2. Definition of which sub-steps are to be performed by the user and which sub-steps are to be realized by the system
3. Derivation of interaction scenarios
   → Going through different scenarios, often interactively with modification of the processes based on new findings → Basis for function hierarchies

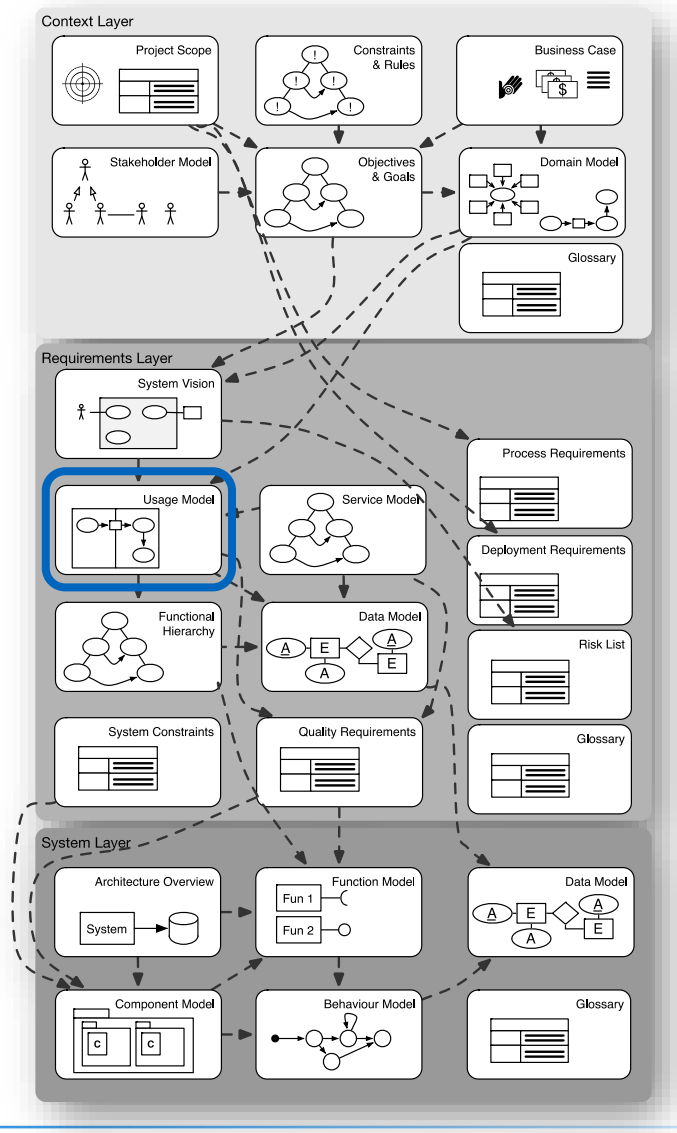
Meaning of the Usage Model

Meaning of the Usage Model

- Application domain analysis: processes between external actors (users, neighboring systems) and the system in context; business processes to be supported
- Essential basis for voting on
- External system behavior (especially unclear/"critical" system behavior) scenarios and their treatment)
  - Identification of functions

Use of the Usage Model

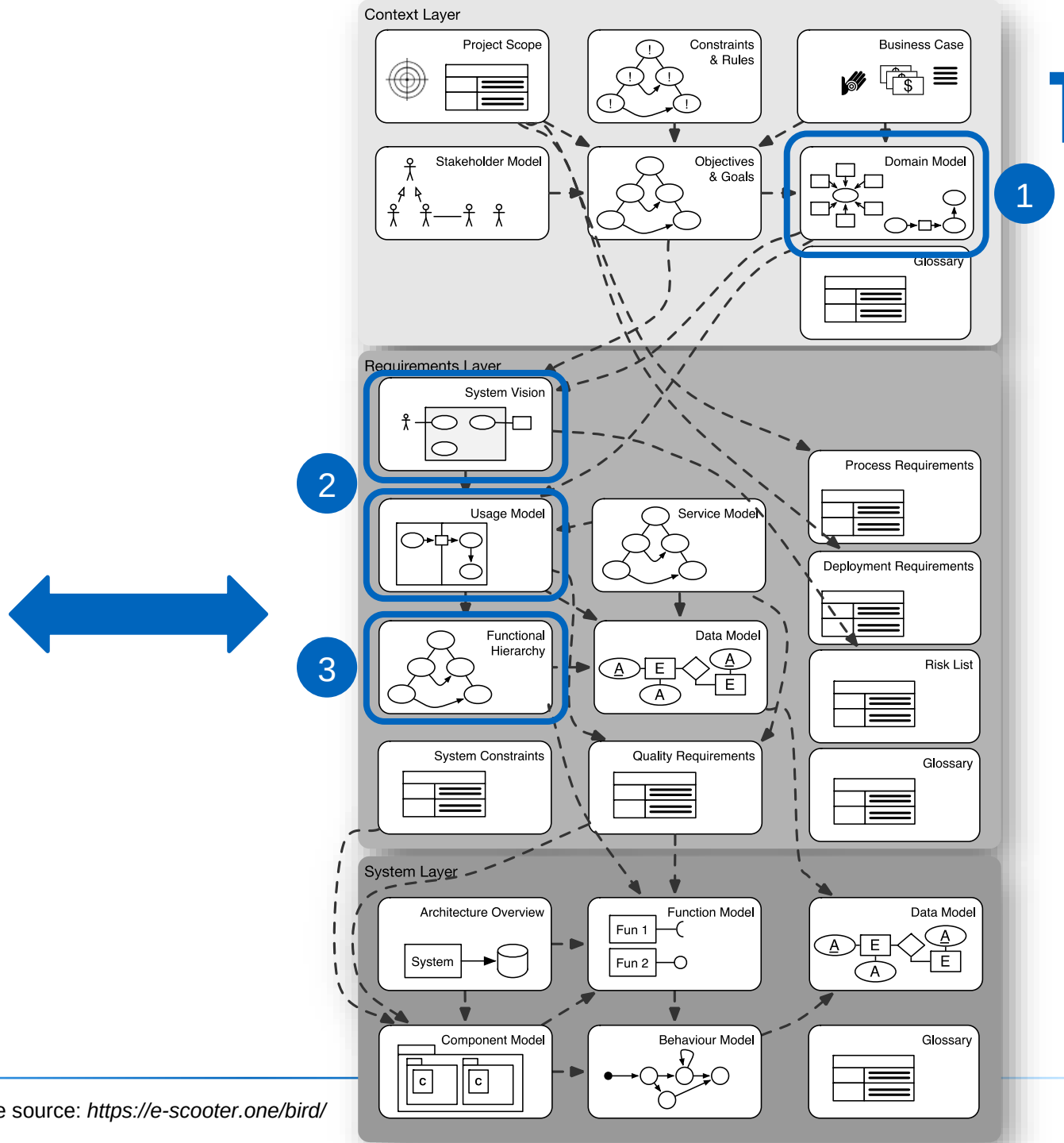
- Basis of communication: "playing through" contextual system utilization → Review and consolidation of processes
- Basis for:
  - Planning and definition of systems (rule scenarios, exception scenarios)
  - Design (architecture, communication protocols, interfaces, ...)
  - Specification of test cases (acceptance tests), detailed effort estimates, ...
  - Formation of function hierarchies as transition to the design



Example

+

Payment of the scooter ride



 Picture source: *https://e-scooter.one/bird/*

What is a Function Hierarchy?

Definition Function:

A (usage) function is an excerpt from the behavior of the system, which can be perceived at the system interface and serves a certain purpose.

Related terms: usage function, service, feature, user-visible function

Definition Function Hierarchy:

Decomposition/structuring of functions and sub-functions as well as (intentional/unintentional) dependencies (feature interaction) to analyze their required behavior definition according to different principles, such as

- Tasks - subtasks
- Causal/temporal order
- Communication Relations
- Distribution to system components

Importance of Function Hierarchies

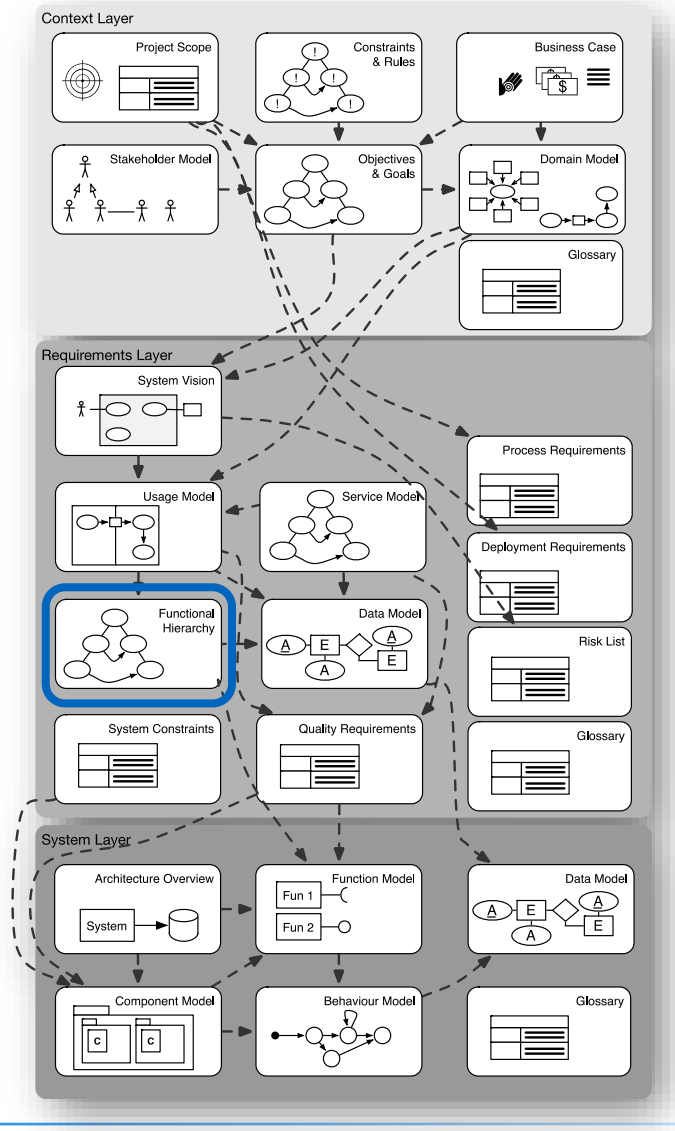
Importance of Function Hierarchies

- Overview of functions for analyzing system behavior
- Functions are
  1. identified on the basis of use cases,
  2. analyzed/tuned by means of Use Cases and
  3. structured in hierarchies

→ Essential basis for design of the (internal) system behavior

Use of the Function Hierarchies

- Creation of syntactic interfaces: Specification of the amount of inputs and outputs at system interfaces, which serve the realization of the function
- Specification of the behavior (models)
- Definition of a logical component architecture



Discussion: Function Hierarchy

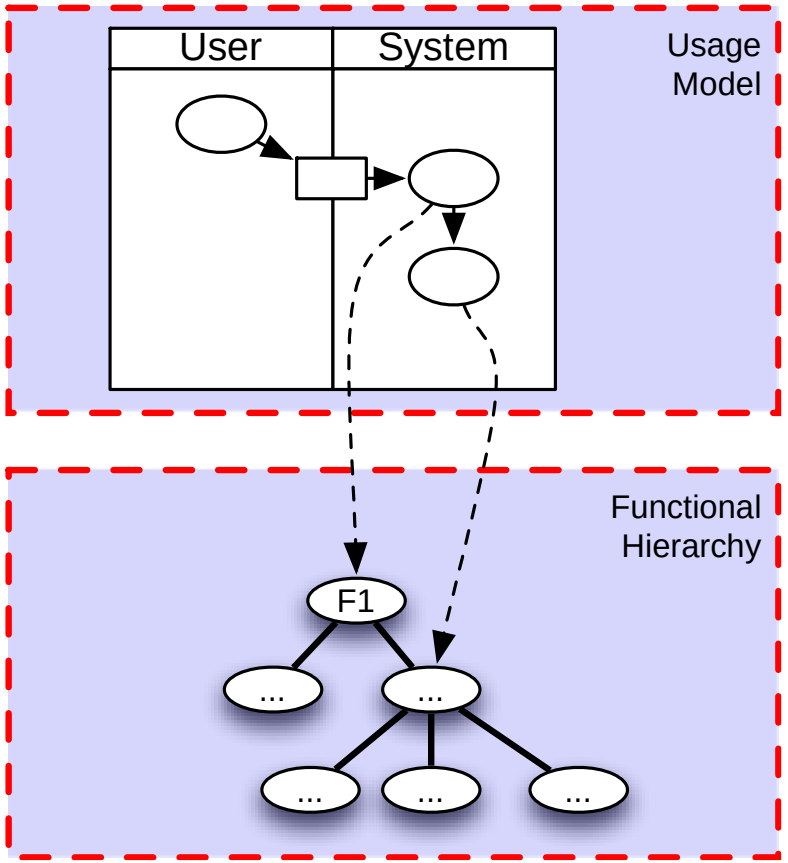
How could one decompose the Function „The user pays for the ride“?

From Scenarios to Functions: Basic Idea

1. Identify system functions based on the use cases / scenarios
   (context of use with assumptions, preliminary and post conditions, triggers, ...)
2. Refine and structure system functions using a hierarchy
3. Identify dependencies between the functions
4. Define external interfaces
   (Dialogs, Data, ...)

Function Hierarchies allow smooth transition into the design

- Definition of a logical component architecture
- Specification of **interface** behavior by means of automata
- Iterative data modelling (continuous completion)



Discussion: Interfaces and Contracts

How to define an Interface?

What is the difference between an Interface and a Contract?

What is a User Story?

Definition User Story:

A user story describes functionality that will be valuable to either a user or purchaser of a system or software. User stories are composed of three aspects: [Cohn04]

- A written description of the story used for planning as a reminder
- Conversations about the story that serve to flesh out the details of the story
- Tests that convey and document details and that can be used to determine when a story is complete

More details about User stories:

- Represent customer requirement rather than document them
- Traditionally, stories are written on cards with test cases on the back side of the card
- Details can be expressed as additional stories

Writing good User Stories

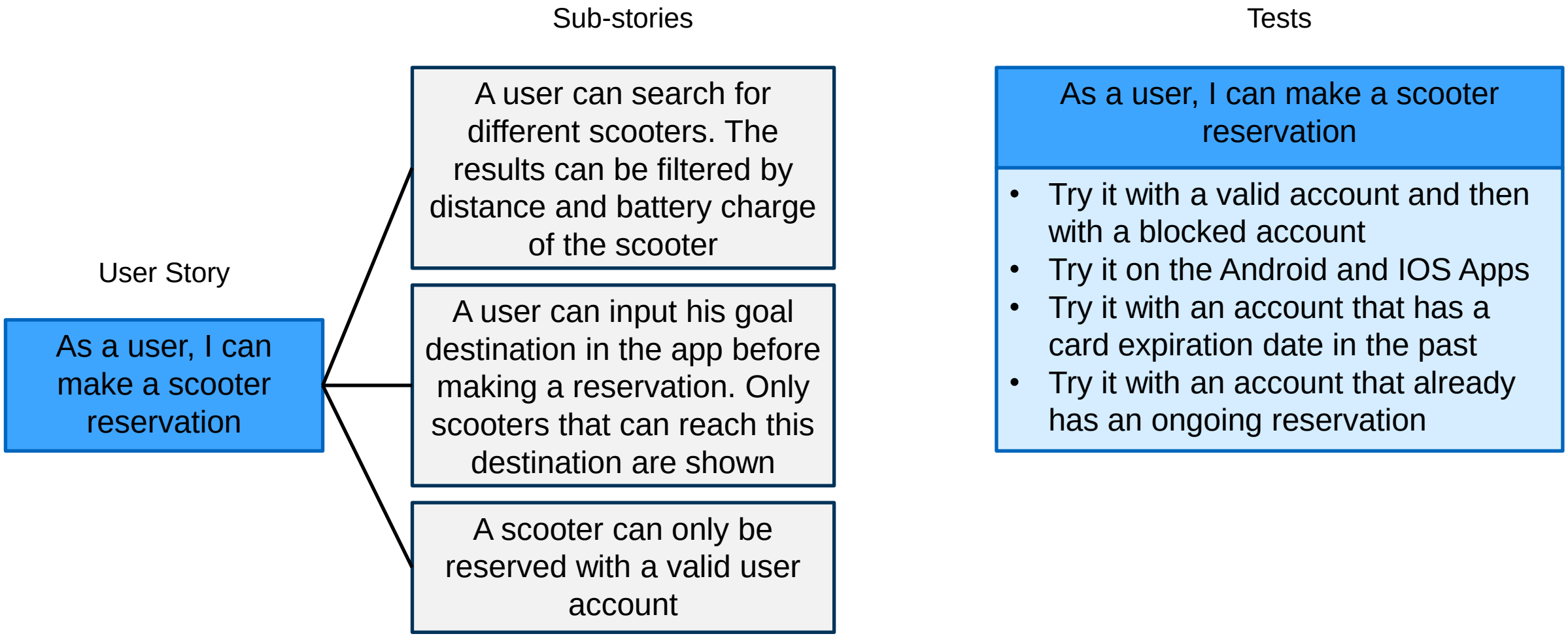
A good User Story is:

- **Independent:** No dependencies between stories as they lead to planning problems
- **Negotiable:** User Stories are no requirements, but description of functionality to be negotiated
- **Valuable:** User Stories should be valuable to users or customers
- **Estimatable:** Developers should be able to estimate the size of a story
- **Small:** Long stories are difficult to understand and should be split up in a series of smaller stories
- **Testable:** Stories must be written so as to be testable. Passing all tests proves deployment

Comparison between User Stories and Use Cases (flaky!):

- User Stories: Written in users' everyday language, small scale and easy to use information; more geared towards needs; "atomic"; clear role; "invitation to discuss"
- Use Cases: Written in users' business language, organize requirements, list of steps; more geared towards requirements; multiple scenarios; "can be used as basis in a contract"

Examples of User Stories



Discussion: User Story

How could a possible user story in our rent-a-scooter example look like“?

What are possible test cases for the story?

General Problems with Behavioral Modeling

Behavioral modeling does not start with the design!

→ Use of modeling techniques allow seamless transition to the design

→ Danger of solution orientation (drifting into the draft)

→ Multiple facets/languages need to be synchronized

The tendency is therefore: Make sure that

- There is focus on the analysis of the intended usage behavior from the overall system view (black box)
- Use cases and scenarios are used to understand and analyze the context and intention of the users
- Function hierarchies can be used,
  - to systematically implement a change of perspective from usage scenarios to user-visible functions
  - to systematize an (iterative) transition into the design

System Models

Definition System Model:

A system model is a meta model which defines an abstract syntax. This abstract syntax is usually specified in a modeling tool.

A system model describes certain model views of a system, which determine the structure or the behavior of this system and their relations among each other.

Definition Views:

Views show a system from a certain perspective (viewpoint). They offer suitable modeling concepts for these perspectives and thus enable a precise observation and analysis of these perspectives. A view is an instance of a viewpoint.

System Models determine what requirements are talking about:

- What is a system? What are the essential characteristics, views and structures?
- Which components and structures does a system have?

Difference between System Models and Models of a System

System Model (“meta model"):
Which constructs are necessary to describe models. Conceptual Model using different views on a System.

→ System is modeled in two consistent views e.g., class diagram and sequence diagram

≠

Model of a System (“artifact model"):
Ideally built on system models like content models. Structuring of Models in structure models.

→ Artefact model like e.g., BPMN, State Chart Diagram, Use Cases

Examples for System Models

Wide range of system models and modeling languages available and used in practice:

- Software: Programming models like UML (UML is a set of coupled and consistent meta models)
- Embedded: Control engineering like MATLAB-Simulink models

Example System Model:

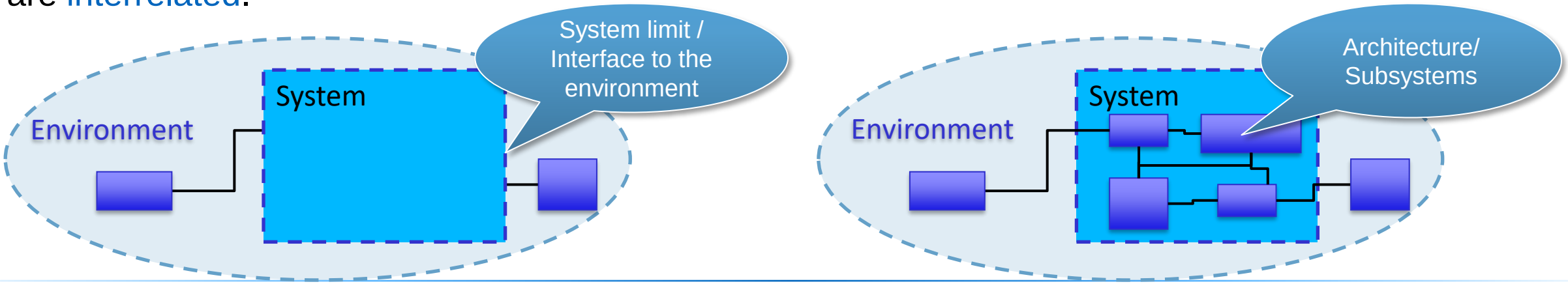
- There are two view types defined in a meta model like a sequence diagram and class diagram
- Each actor in the sequence diagram represents an instance of a class in a class diagram
- If a name of an actor (which is present as a class in the class diagram) is missing in the sequence diagram, then inconsistency is present in the model
- Syntactic inconsistency needs to be avoided with system models.
  (What distinguishes syntax from semantics?)

What is a System?

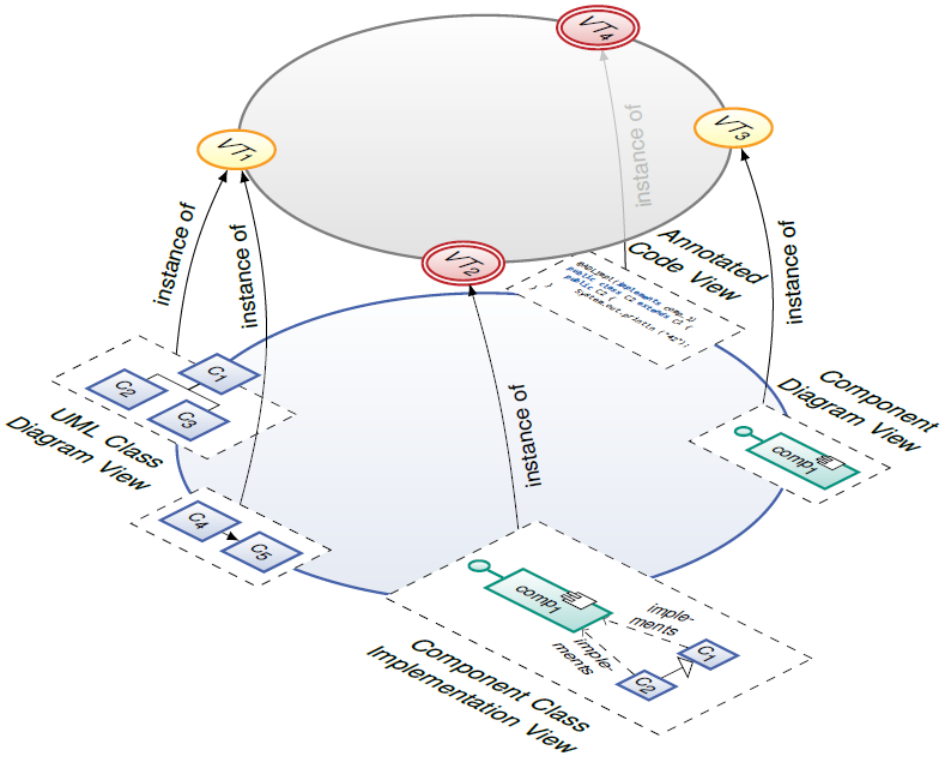
Definition System:

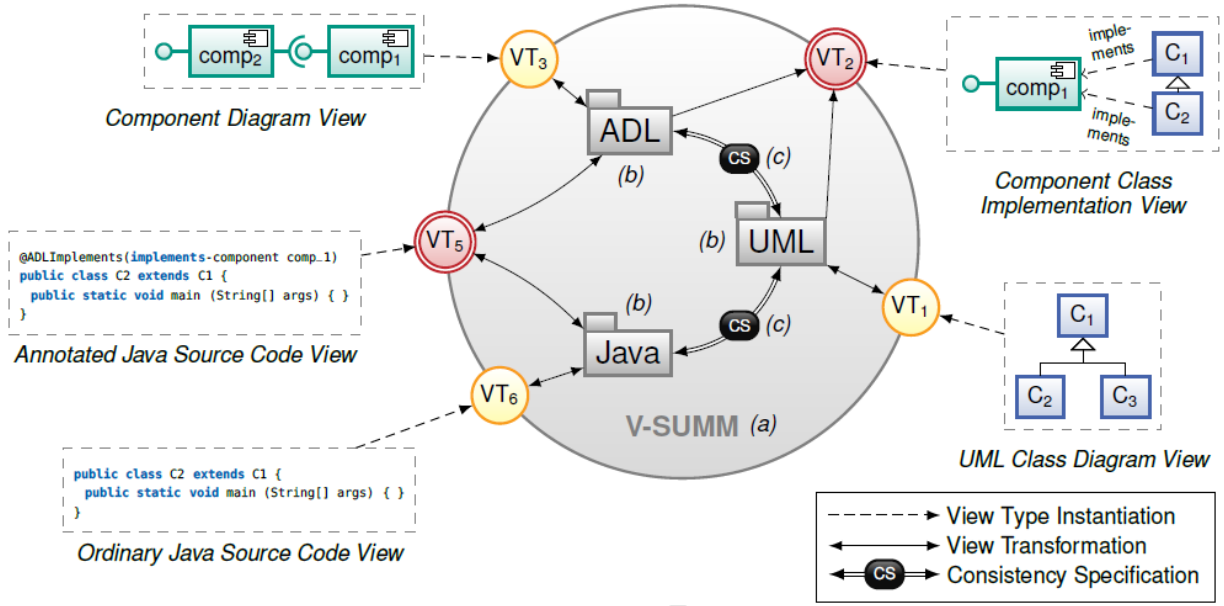
A system is delimited from its environment (operational context) by defining its system boundary.
A system has an interface that defines (1) which forms of interaction between a system and its environment are possible (static/syntactic interface) and (2) which behavior the system shows from the context view (interface behavior, dynamic interface, interaction view).

A system has an architecture. The architecture consists of elements (components, subsystems) that are interrelated.



Projective vs. Synthetic





Projective vs. Synthetic

Projective (“god model"):
Holds all models for all views. God models gets projected into requested views.

→ Consistency is given “for free”

VS.

Synthetic (“coupling of models"):
Different views or view types implemented with specific tools are pairwise coupled.

→ worst case quadratic couplings

Is the effort (cost, maintenance, collaboration) justifiable for the benefits (higher quality) that those models promise?
→ Probably not for software
→ But probably for hardware

Critical Aspects when using Models

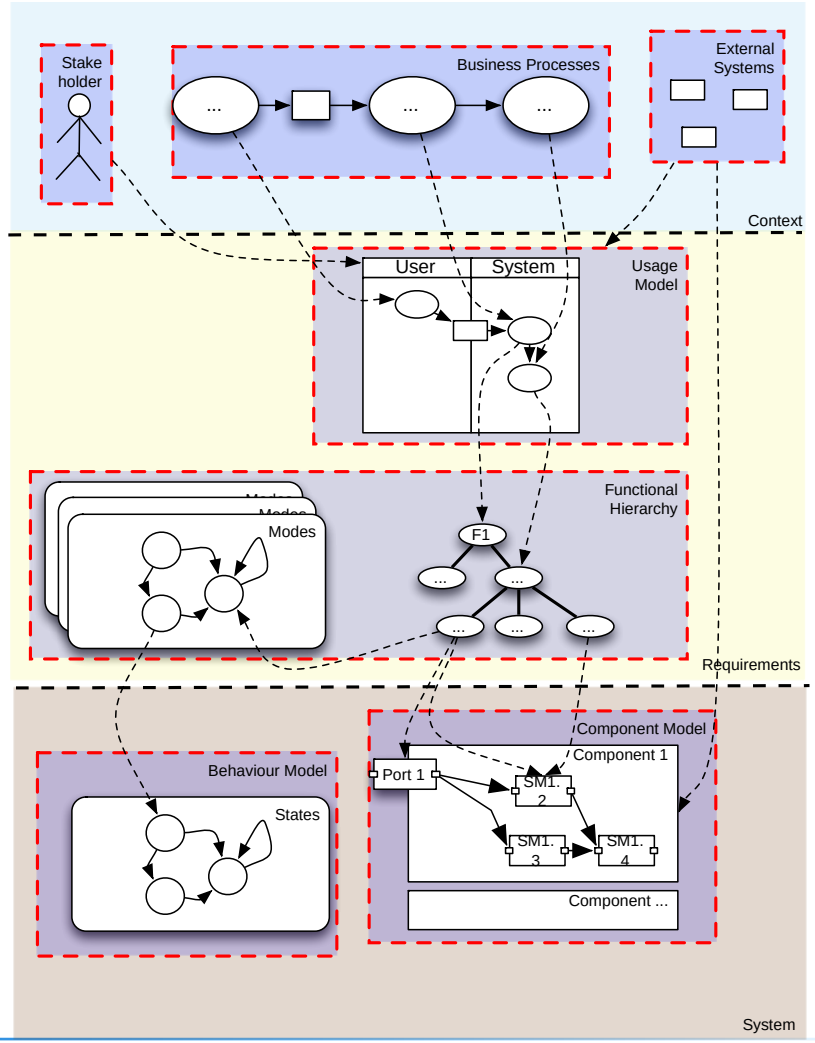
- Effort/Benefit when using models (e.g., detailed scenarios for "trivial" issues; synchronization!)
- Complexity of models and choice of notation
- Analysis and documentation of scenarios is crucial:

→ Scenarios form a communication bridge between
  – Stakeholders (for analysis/coordination)
  – Architects (draft)
  – Testers (specification of test cases)

→ **Trade-Off:** The more precise/formal the modeling, the more targeted questions can be asked and be handled by the stakeholders **- but the more you define!**
  → explicit feedback to stakeholders, their goals and requirements
  → Exposure of need for clarification, conflicts
  → Vote, negotiate, prioritize and decide

Summary

- **Structured view on functionality** has vital importance for RE
- **Functionality** must usually be seen in an **operational context**
- Functions can be described on **different levels** and in **different forms**
  - Business process models
  - Use Cases and Scenarios
  - Function hierarchies
  - Multiple models need to be integrated



Outline and Outlook

Terms and Definitions
Context-specific nature of SE and RE
Quality models for requirements
Engineering Models
Stakeholder and Requirements Elicitation
Goals and Goal-oriented RE
Non-functional Requirements
Functional Requirements

Formalization

Agile Processes
Requirements Management and Quality Assurance
Trends in Research

Literature

- Davis, Alan M. Software Requirements: Objects, Functions, and States. Prentice-Hall, Inc., 1993.
- Glinz, Martin. "Requirements Engineering I." *Nicht funktionale Anforderungen. Universität Zürich, Institut für Informatik, Zürich* (2006). https://files.ifi.uzh.ch/rerg/arvo/ftp/re_I/Kapitel_08_Szen.pdf
- Alistair, Cockburn. “Writing Effective Use Cases” Humans and Technology in preparation for Addison-Wesley Longman, 2000. https://www.infor.uva.es/~mlaguna/is1/materiales/BookDraft1.pdf
- Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.