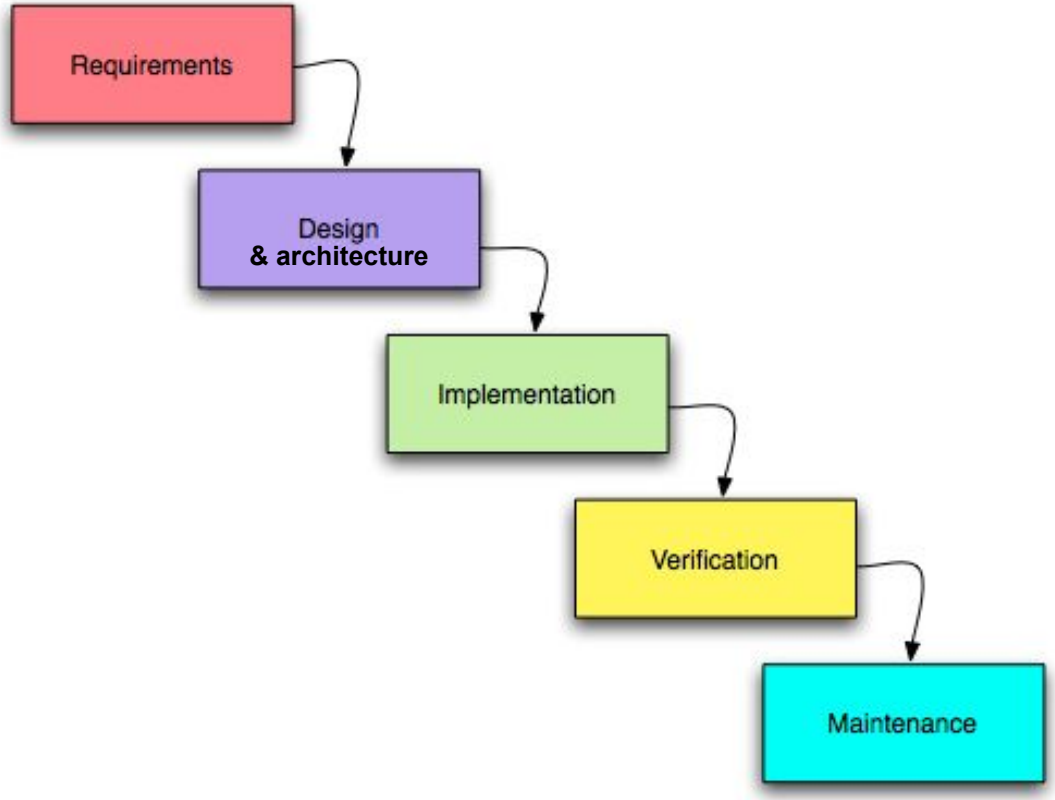
Requirements modeling

Jana Kostičová

SW development process



Importance of pre-construction phases

- Pressman[2001] - normal projects
  - 40% specifications & design
  - 20% building
  - 40 % testing
- Pressman[2001] - large projects
  - 3 % planning
  - 44,5% specifications & design
  - 17,5% building
  - 35 % testing
- Boehm[1987]
  - 60% specifications & design
  - 15% building
  - 25 % testing
- Zelkowitz[1978]
  - 35% specifications & design
  - 20% building
  - 45 % testing
- Brooks[1995]
  - 33% planning (including specifications & design)
  - 17% building
  - 25% component testing
  - 25 % system testing
- Schach[2007]
  - 39% specifications & design
  - 40% building
  - 21 % testing

Example: E-commerce store requirements

Business requirements (Describe high-level objectives of the organization itself)

- Achieve a 20% increase in sales in first year
- Expand customer base by 15% in first year
- ….

Stakeholder requirements (Describe stakeholder/user needs)

- Users want to be able to search the products
- Users want to safely purchase selected products
- Sales dept. wants to be able to use flexible pricing strategies
- ….

Solution (system) requirements (Describe system's functions, services and operational constraints)

- The system shall allow the users to browse the products by category
- The system shall allow the users to search the products by name
- The system shall allow the users to add items to the shopping cart

} **Functional requirements**

- ….
- The system shall adhere to Payment Card Industry Data Security Standard (PCI DSS) compliance for handling and storing payment card information

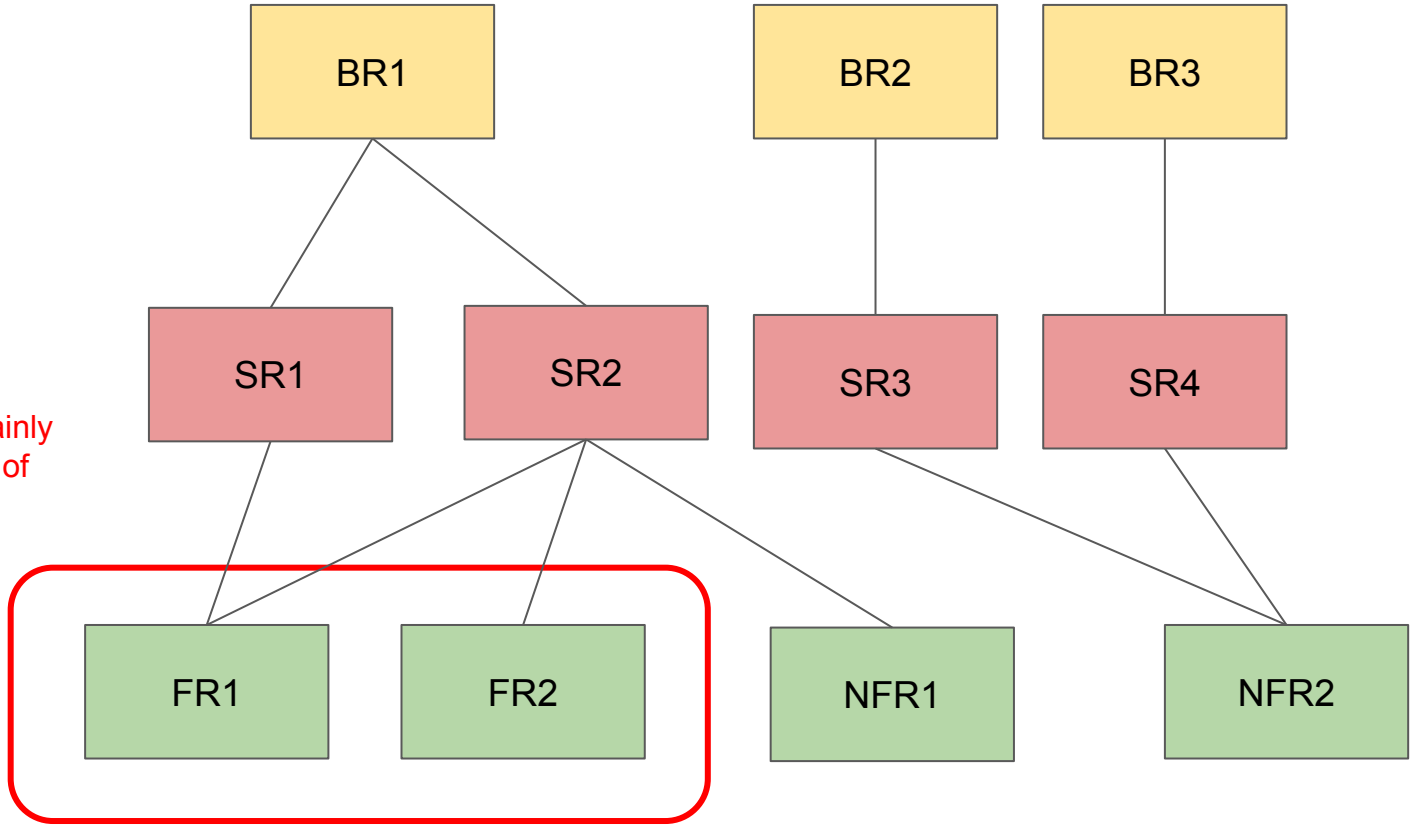
} **Nonfunctional requirements**

- ….

Stakeholder: an individual, group or organization who may affect or be affected by the result of the project

(BABOK® Guide requirements classification schema)

Requirements traceability



We will focus mainly on the modeling of functional requirements

Functional requirements

- Describes required functionality
- Input for next phases of SW development process (together with nonfunctional requirements)
- Typical form
  - Natural language text (free format, typically structured)
  - UML Use Case model
    - Use Case diagrams
    - Use Case and Actor descriptions
  - Dynamic UML diagrams - activity diagrams, state diagrams, ….
  - User stories and supporting documents
- Functional requirements are often supported by GUI models
  - Partially describes also non-functional requirements

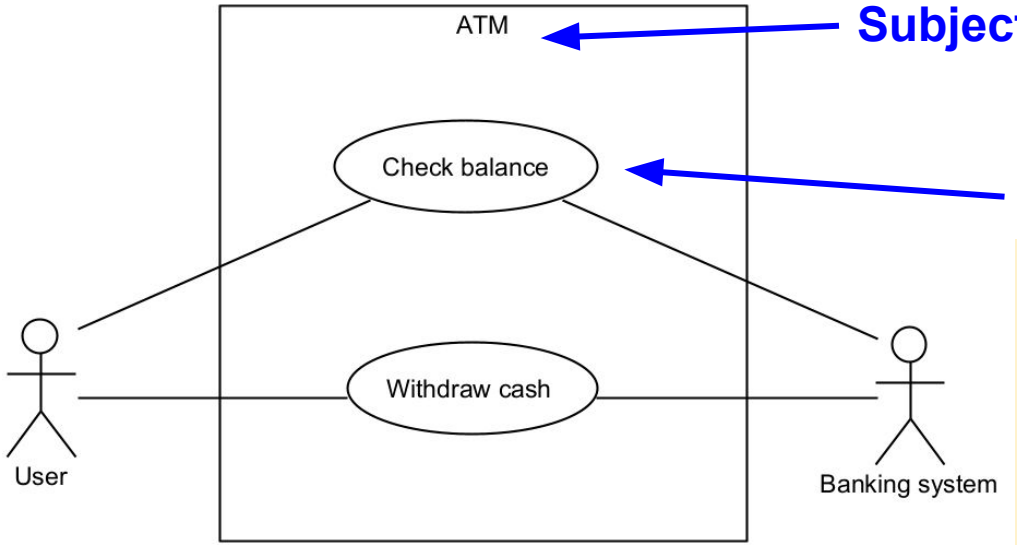
A common problem: the stakeholders do not know UML … ☹

In GUI-centric applications it might be more convenient to use wireframes/mockups and their description instead of diagrams

Use Case Diagram

= behavioral UML diagram

- High-level view of the system: actors, use cases, subjects and their relationships
- Typically easy-to-understand for stakeholders



Subject

= a system under consideration to which the Use Case applies

Actor

= an external entity that interacts with a system (e.g., human users, external hardware, or other systems)

Use Case

= a set of behaviors performed by the system, which yields an observable result that is of value for actors or other stakeholders of the system

Actors & Use cases

Actors are always placed outside the system

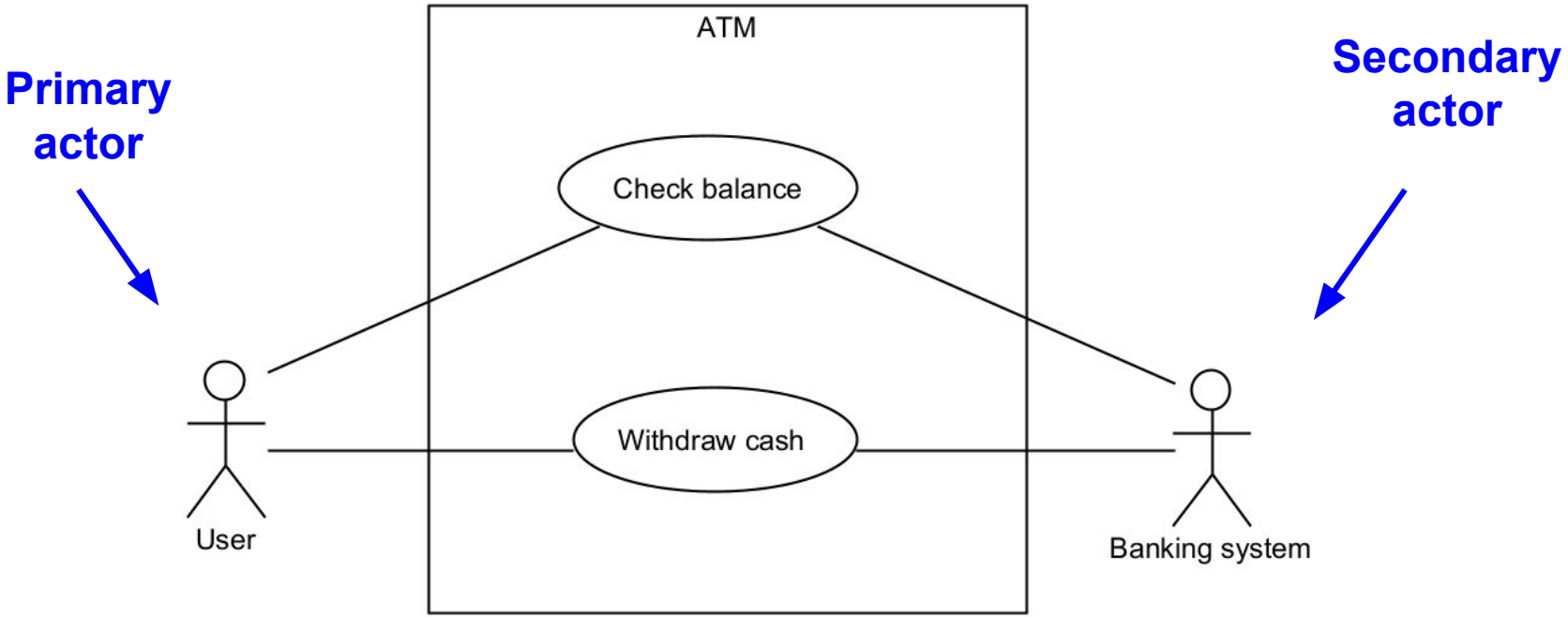
Actor represents a role, it means

- A single physical instance may play the role of several different Actors
- A given Actor may be played by multiple different instances

Actors are further defined as follows *

- **Primary actor:** has a defined user goal that the system aims to fulfill. This actor initiates the corresponding use case.
- **Secondary actor:** provides a supporting role to complete the use case.

** Cockburn[2000]*


Primary actor
Secondary actor
ATM
Check balance
Withdraw cash
User
Banking system

Relationships

Actor - Actor

- Generalization

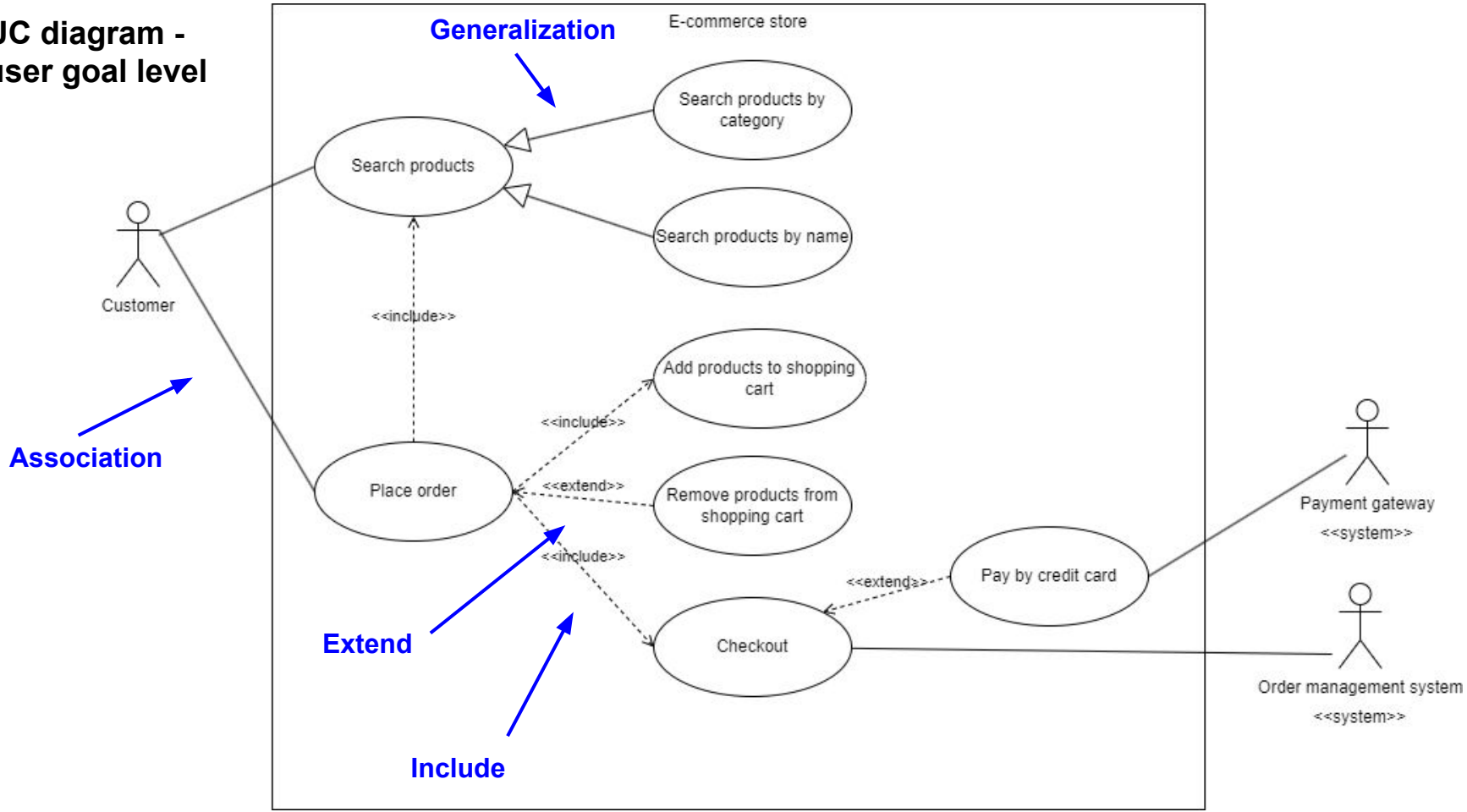
Actor - Use Case

- Association
  - Represents interaction, associated Use case typically capture a user goal

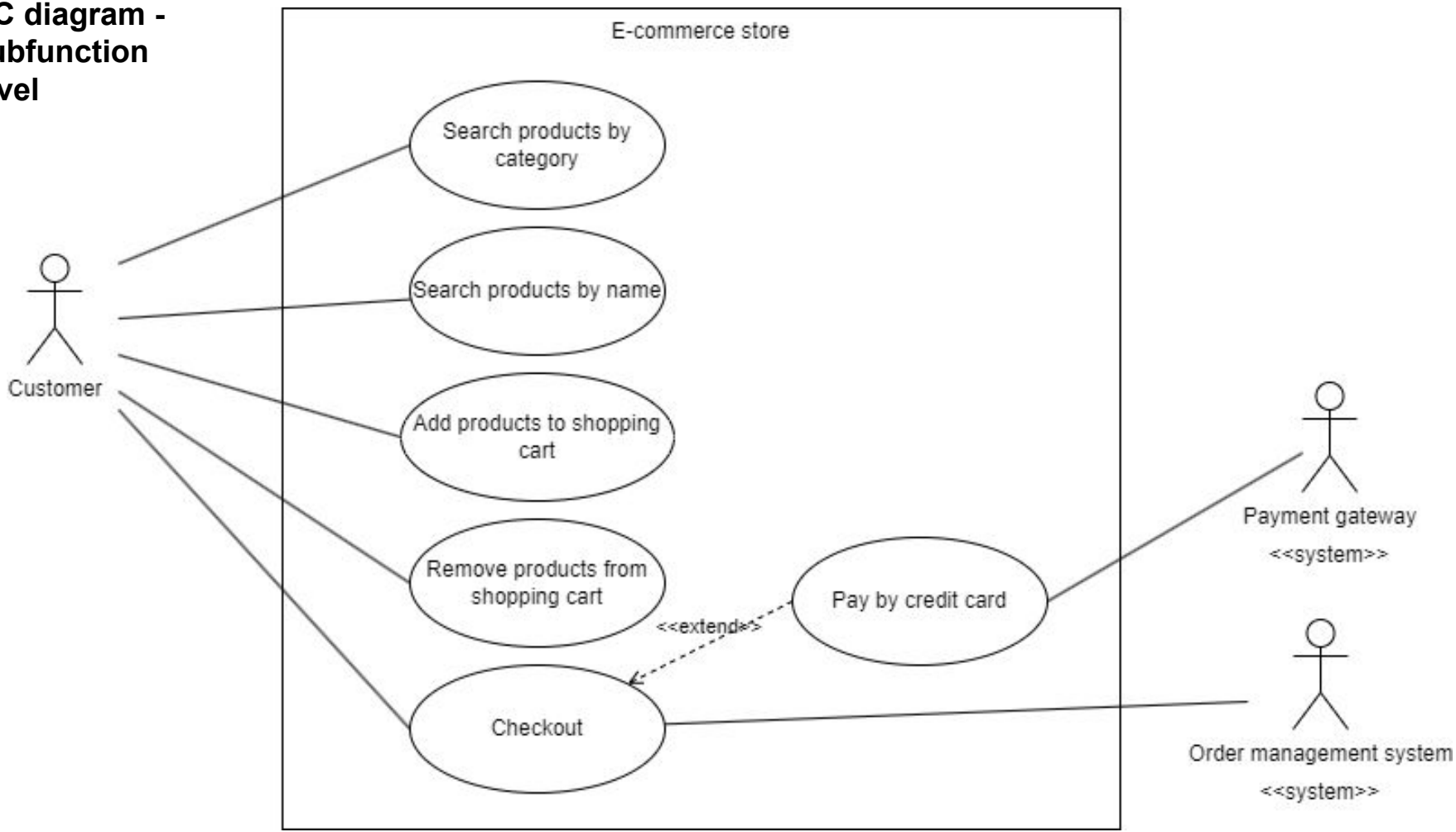
Use Case - Use Case

- Generalization
- “Include” dependency
  - An including use case always contains the behavior defined in another, included (base), use case. Included use case can be seen as subroutine.
- “Extend” dependency
  - The behavior defined in the extending use case can be inserted into the behavior defined in the extended use case

UC diagram - user goal level



UC diagram - subfunction level



Best practices

- Actor name
  - Nouns, unified and understandable from business point of view
  - Example: Customer, Supplier, Printer
- Use case name
  - Should begin with the imperative form of the verb
  - Examples: **Open** order, **Print** order
- Common domain glossary across the model
  - Log in to **the system**, Log out from **the application** -> WRONG
  - Log in to **the system**, Log out from **the system**
- Take into account time sequence of Use Cases
  - The first UC on the top, another below it, etc.
  - It i not always possible to find such time sequence
- Primary vs secondary actors
  - Primary actors are placed on left-hand side, secondary actors on right-hand side of the corresponding use case
- Avoid GUI and implementation details
- Minimize usage of “extend” dependency and generalization between UCs
  - Semantic can be too complex and confusing
- Keep the diagram consistent within the chosen level (user goal level, subfunction level, …)

This should be followed throughout the specification

Use case description

- UML use case diagram only provides a "big picture" of the interaction between actors and the system
- More details are typically provided by

  1. Text descriptions: Goal, Preconditions, Postconditions, Algorithm, …
  2. Behavioral UML diagrams - activity diagrams, state diagrams, …

- A detailed description of all use cases can lead to extensive documentation that is difficult to maintain. It can be reduced as follows:
  - Provide detailed description only for critical or complex use cases
  - Provide short description for simpler use cases, focusing on key aspects

Note: The concept of “use case” was introduced already in 1987 by Ivar Jacobson while UML (including Use Case modeling) was standardized in 1997.

“Fully dressed” description

= detailed description, it typically contains the following elements:

1. Attributes
   - Name: use case name
   - Goal: a longer statement of the goal
   - Scope: subsystem, application, ...
   - Pre-conditions: state before use case execution
   - Post-conditions: state after use case execution
   - Trigger: action upon which is use case started
   - Primary actor
   - Secondary actors
2. Algorithm (flow of events)
   - Primary scenario, extensions and variations
3. Additional information
   - E.g., priority, related non-functional requirements, …

The concept of fully dressed vs casual use case descriptions comes from Cockburn[2000]

Use case name: Checkout
Goal: To order products in the shopping cart
Preconditions: Non-empty shopping cart
Postconditions: Successfully placed order for the products in the shopping cart or information about error in order processing

Trigger: Customer chooses an option to proceed to checkout.
Primary actor: Customer
Secondary actor: Order management system

GUI independent phrases

1. Customer chooses an option to proceed to checkout.
2. System displays the content of the shopping cart
3. Customer confirms that he/she wants to proceed to checkout.
4. Customer enters phone no. and e-mail.
5. System validates entered data.
6. If data is not valid, systems informs about detected errors and UC returns back to step 4.
7. Customer chooses one of the following delivery methods - delivery service or personal pickup.
8. If the chosen delivery method is “Delivery service”
   a. User enters the delivery address
   b. System validates entered data.
   c. If data is not valid, system informs about detected errors and UC returns back to step 8a.
9. Customer chooses one of the following payment methods - credit card or bank transfer.
10. If the chosen payment method is “Bank transfer”, system displays payment instructions.
11. If the chosen payment method is “Credit card”
    a. System calls **UC “Pay by credit card”.**
    b. If the payment by credit card failed, UC returns to step 9.
    c. System informs about successful payment.
12. Order management system registers the order.
13. If registering the order fails
    (steps 13a-13b happen in any order)
    a. System informs the customer about error in processing the order
    b. System sends notification to technical support
    c. Use Case terminates (failure)
14. System performs the following steps in parallel
    (steps 14a-14b happen in any order)
    a. System confirms that the order has been placed successfully
    b. System sends confirmation e-mail to the customer.
15. Use case terminates (success)

UC is still simplified, for example it is not mentioned what “display shopping cart” means, free delivery is assumed, …

“Casual” description

= short description, the structure is not predefined

UC Checkout

Customer initiates checkout process. System displayed content of the shopping cart and customer confirms it. Customer

- Enters phone no. and e/mail
- Chooses delivery method (delivery service or personal pickup). In case of delivery service, he/she enters also delivery address.
- Chooses payment method (bank transfer, credit card). In case of credit card, UC Pay by credit card is called.

Order management system registers the order.

- In case of failure, system informs about error in processing order and if the order has been already paid, it ensures the customer that amount paid will be refunded to given credit card.
- In case of success, system informs that the order has been placed successfully. It provides payment instruction if the bank transfer was chosen. It informs that the order has been paid if the credit card was chosen and the payment was successful.

Activity diagram

= behavioral UML diagram

- A visual means of describing algorithms / workflows

- **Action:** a specific unit of work or operation
- **Partition (swimlane):** a logical grouping of activities or actions
- **Special nodes**
  - **Initial node:** the starting point of the activity diagram
  - **Final node:** the ending point of the activity or process
- **Conditionals**
  - **Decision:** a point where a decision is made, the control flow can follow different paths based on conditions
  - **Merge:** a point where different control flows converge back into a single flow
- **Concurrency**
  - **Fork:** a point in the process where the control flow splits into multiple concurrent control flows
  - **Join:** a point where multiple concurrent control flows converge back into a single control flow
- **Objects and object flows**
- **Parameters**

Log in, log out, register

- Are these use cases?

  Use case = “a set of behaviors performed by the system, which yields an **observable result** that is of **value** for actors or other stakeholders of the system”

Log in, log out, register

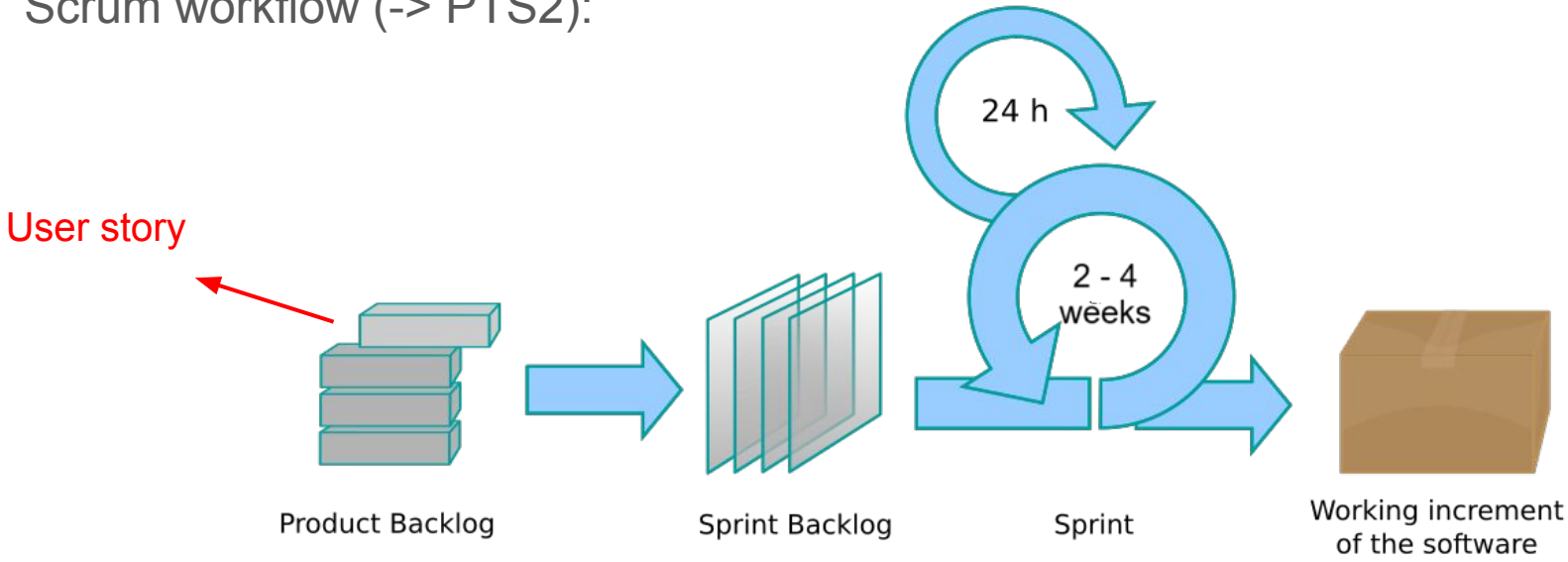
- “Technical” functions implementing non-functional requirements
  - Example NFR: “Data are accessible by authorized parties only”
- These are not use cases in the strictest sense
- No consensus on how to model them

Example solutions

- Login / logout / register are not modeled as use cases
  - UC diagram: Constraint “User is logged in”
  - UC descriptions: Precondition “User is logged in”
- Login / logout / register are modeled as use cases
  - UC diagram: Separate use cases + constraint “User is logged in” / “User is registered”
  - UC descriptions: Precondition “User is logged in” / “User is registered”
  - Actor “User” may need to be differentiated into “Unregistered user” / “Registered user”
  - Could make sense at more technical level

Agile analysis - SCRUM

Scrum workflow (-> PTS2):



Source: Wikimedia

User stories

= an informal, general explanation of a software feature written from the perspective of the end user or customer

- Used especially in agile methodologies (e.g., Scrum)
- Common template

```
As a <role> I want to <capability>, so that <I receive benefit>
```

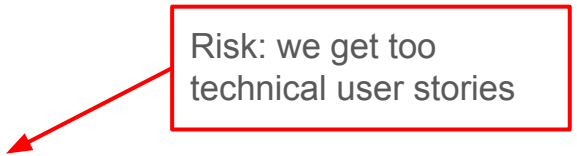
(“So that” part is optional)

- User stories are placeholders for further discussion, they can be split / refined to more detailed specification if needed
  - Sometimes they are even not considered to be true requirements
  - The granularity of user stories changes over time, the final user stories (those to be implemented) represent small piece of work that can be implemented within a short iteration
- Acceptance criteria
  - Conditions that the given feature must fulfill in order to be accepted by stakeholders
  - Provides more details about User story

User story evolution

Scrum assumes the existence of a product backlog that is continuously filled with raw user stories, but does not prescribe how those stories are generated.

1. **Raw user stories**
   - Initial, unrefined stories
   - They capture the high-level goal but lack detailed acceptance criteria, context, or specific technical details
2. **Refined user stories**
   - Raw stories are typically refined during Story refinement / backlog grooming meetings - continuous process (partially during Sprint planning)
   - Raw user stories are
     - Further specified, priotitized
     - Broken down into smaller ones if needed
     - Supplemented by acceptance criteria
   - Refined stories are ready for implementation within a single sprint
3. **Even more refined user stories**
   - During sprint
   - A refined user story is further specified to facilitate design / implementation / testing



As a customer, **I want to** search for products by a product name **so that** I can easily find and select the products I want to buy.

Acceptance criteria:

- Given a customer's search criteria (partial product name), the system displays a list of products that match the specified search.
- If no products match the search criteria, the system shows a message informing the customer that no products were found.
- Each product in the list displays the following mandatory information:
  - Product name, Product code, Price per item, Stock quantity
- Each product in the list displays the following optional information (if available):
  - Product photo, Product description
- For each product with a stock quantity greater than 0, the customer can input a quantity (less than or equal to the available stock) and add the selected quantity to the shopping cart.

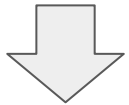
\+ Other specification, e.g. wireframes / mockups, algorithm (text or activity diagram), …

Acceptance criteria

- Free text
- Or template: `Given … when … then …`

Origin: Behavior-driven development [D.North]

Given a customer's search criteria (partial product name), the system displays a list of products that match the specified search.



Given: The customer is on the product catalog page and enters a non-empty product name (partial or complete) as a search criterion.

When: The customer initiates the search for products.

Then: The system displays a list of products that match the specified search criterion

Preconditions

Trigger

Expected behavior

User stories vs use cases - standalone

Standalone User story vs standalone Use case (in UC diagram)

User story (US)		**Use case (UC)**
Role	~	Primary actor
Capability	~	Use case name
Benefit	~	UC description / “Rationale”

At requirements level, UCs primary actors typically overlap significantly with USs roles.

USs acceptance criteria vs UCs postconditions

- **UCs postconditions**
  = the state of the system after the Use Case has been completed
- **USs acceptance criteria**
  = the conditions to be met so that the implementation of US is accepted by the stakeholders
  The may refer to the postconditions but also intermediate conditions / behaviors of the system.

Example - password reset

Use Case: "Reset password"

Postconditions (success scenario):

- The user's password is updated.
- The password reset link becomes invalid after it is used.
- A success message is displayed to the user confirming the password reset.

User Story: "As a user, I want to reset my password so that I can access my account if I forget my password."

Acceptance criteria (success scenario):

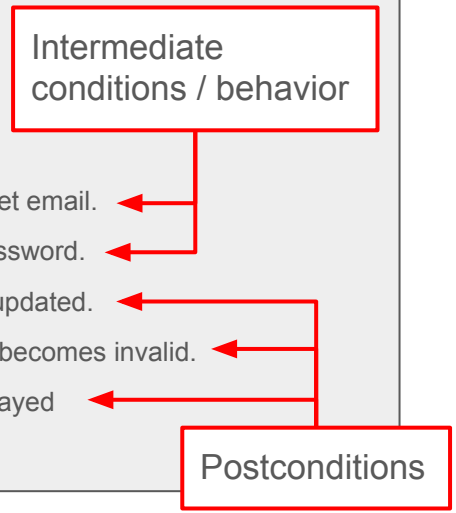
Given the user is registered, When they choose option to reset password Then they should receive a password reset email.

Given the user receives the reset email, When they click on the reset link Then they should be able to enter new password.

Given the user enters new and valid password, When they confirm the new password Then the user's password is updated.

Given the user enters new and valid password, When they confirm the new password Then the password reset link becomes invalid.

Given the user enters new and valid password, When they confirm new password Then a success message is displayed to the user confirming the password reset.



As a customer, **I want to** search products **so that** I can easily find and select the products I want to buy.

~ UC Search products

As a customer, **I want to** add products to shopping cart **so that** I can save the products before checkout.

~ UC Add products to shopping cart

As a customer, **I want to** remove products from shopping cart **so that** that I can update my selection before checkout.

~ UC Remove products from shopping cart

As a customer **I want to** checkout **so that** I can complete my purchase.

~ UC Checkout

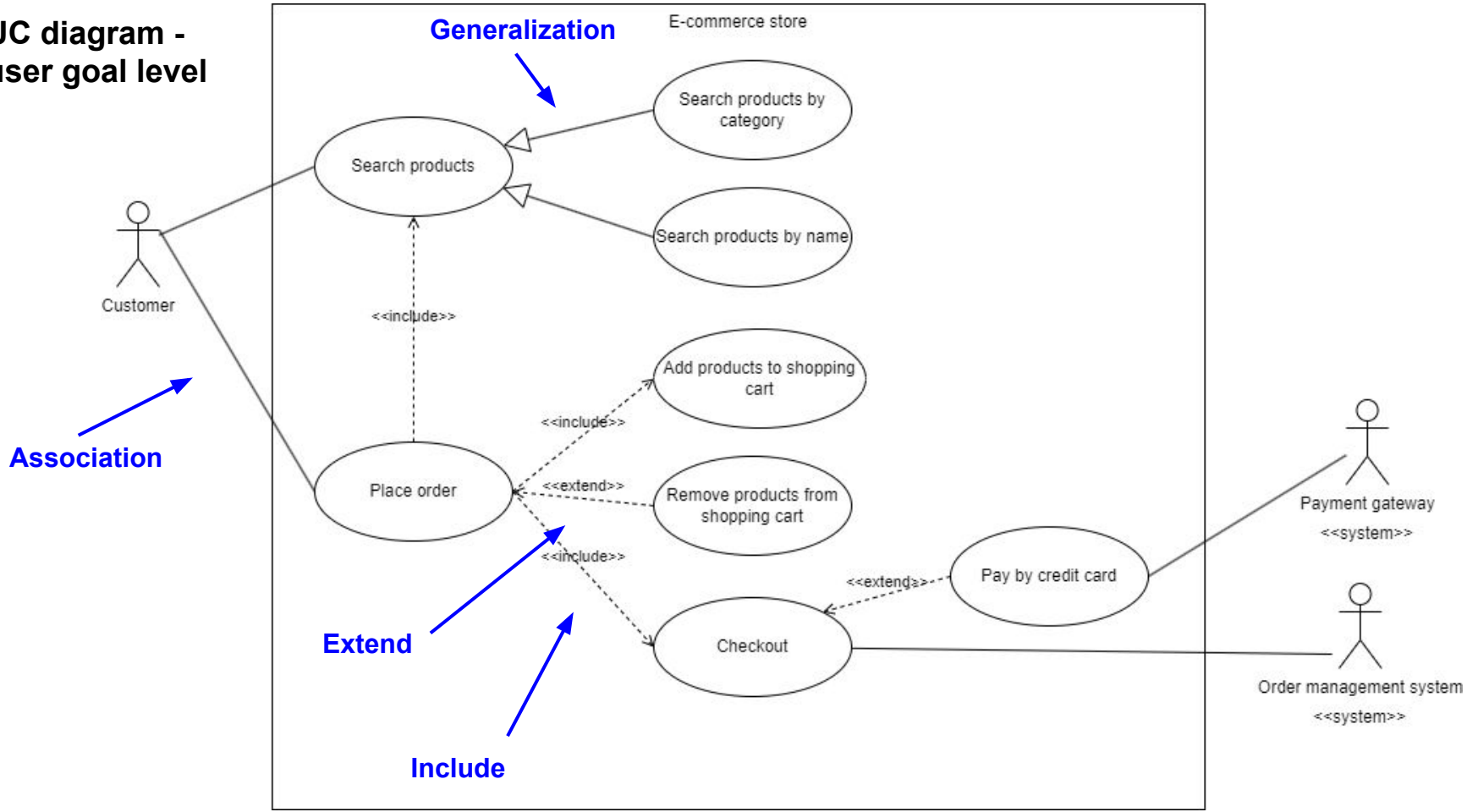
As a customer **I want to** pay by credit card **so that** I can finalize my order.

~ UC Pay by credit card

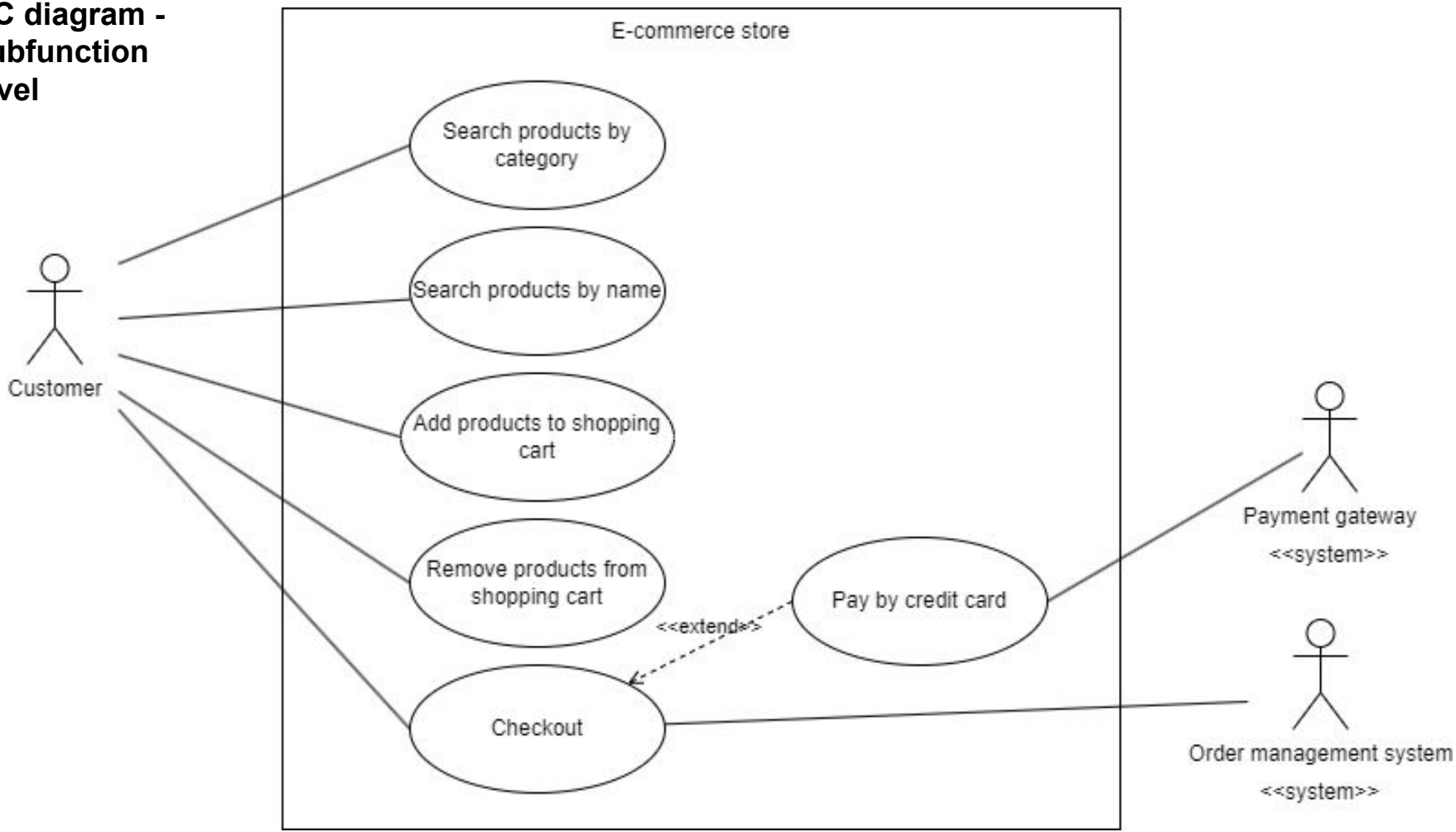
User stories vs use cases - mutual relationships

- User stories
  - None or weak mutual relationships
    - Focus on agile development
  - Higher level stories -> lower level stories
    - Raw story -> Refined story - no clear process how to handle original raw stories after refinement, context might be difficult to access or lost
  -
- Use cases
  - Several types of mutual relationships (include, extend, generalization)
    - Focus on waterfall / long iteration development
  - Higher level use cases -> lower level use cases
    - Multiple granularity levels in a specification
    - Context is available

UC diagram - user goal level



UC diagram - subfunction level



User stories vs. use cases

User cases

(+) complex functionalities with many internal dependencies can be captured by single UC or a group of related UCs (typically at user goal level), then the context is not lost

(+) named user goals provide a summary of project's scope

(-) a single UC may represent a large chunk of work and thus be difficult to develop within a short iteration

UCs fully-dressed descriptions:

(-) a lot of writing for analysts, a lot of reading for stakeholders

(-) difficult to maintain

User stories (refined)

(+) short, easy to read, and understandable

(+) represent small increments of functionality that can be developed in short iterations

(+) refined just-in-time, avoiding too-early specificity

(+) easy to maintain

(-) they do not capture the larger context, thus they are not sufficient to perform a more complex analysis / design

(-) they do not provide summary of project scope

References & further reading

- A. Cockburn. Writing Effective Use Cases, Addison–Wesley, 2000
- I. Jacobson. Object Oriented Software Engineering: A Use Case Driven Approach, 1992
- D. Leffingwell. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, 2010
- R. Červenka, UML Use Cases, Activities
- IIBA. Guide to Business Analysis Body of Knowledge (Babok Guide) v3, 2015
- OMG. OMG Unified Modeling Language. Version 2.5.1, December 2017
- K. Fakhroutdinov. The Unified Modeling Language, 2018
- D. North: Introducing BDD, 2006
- Ken Schwaber and Jeff Sutherland: The 2020 Scrum GuideTM, 2020
- Scrum.org

References - importance of pre-construction phases

- R. S. Pressman. Software Engineering: A Practitioner's Approach, 2001
- B.W. Boehm, Industrial Software Metrics Top 10 List, 1987
- M.V. Zelkowitz. Perspectives on software engineering, 1978
- F. Brooks Jr.. Mythical Man-Month, The: Essays on Software Engineering, Anniversary Edition, 1995
- S.R. Schach, Object-Oriented and Classical Software Engineering, 2007.