

# Software Development Process

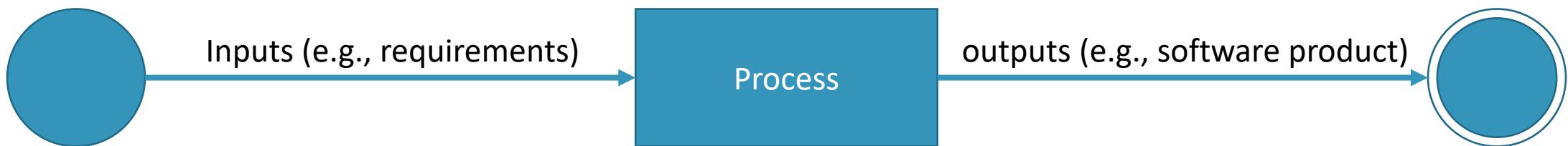
A fast overview and a practical example

# Topics

- Software Development Process (SDP)
  - Why is needed?
  - Popular SDP
  - Core activities
  - Artifacts overview
- SDP throughout an example
  - From requirements to code
  - Introducing artifacts and UML
- Promoted working method

# What is a process?

- A specification defining **who** does **what**, **when** and **how** in order to meet a given goal
- It comprehends a **set of interrelated activities** that transform a set of inputs in a set of outputs



# Why do we need SDP for SW development?

- SW development easily becomes a chaotic activity characterized by “coding and composing”
- SW is written without an underlying plan, becoming full of short-term decisions
- This (might) work while SW is small and has little complexity
  - As SW size and complexity grows, it becomes increasingly difficult to add new features
  - Bugs become increasingly relevant and more difficult to solve

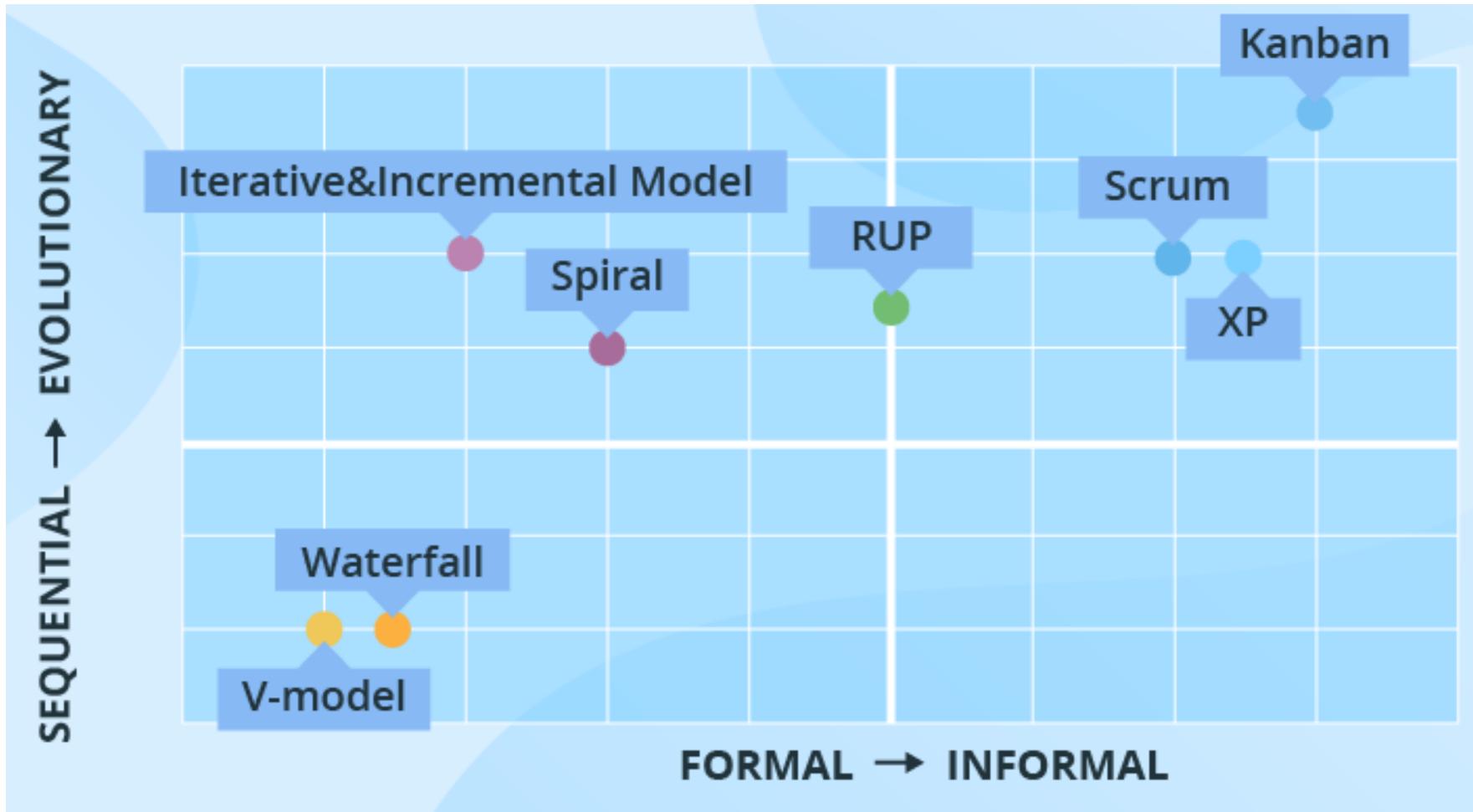
# Some problems in SW development

- Misunderstood business (rules)
- Business changes
- High defect rates
- Outdated system
- Deviations in SW delivery scheduling

# Software Development Process (SDP)

- Coherent sequence of practices systematizing the development or evolution of SW systems
- Set of activities, restrictions and resources that produce a desired result
  - Each activity must have a clear input and a clear output criterion
- Structures the activities by establishing an order or sequence
  - Set of principles or criteria that explain the objectives of each activity
  - Decision points in Planning
  - Synchronization points for the collaborative work of team

# Some popular SDP



Extracted from [2]

# Core software engineering activities

- Requirements gathering
- Feasibility study
- Analysis
- Design
- Implementation
- Testing
- Deployment
- Configuration management
- Project management
- Operations and Maintenance

# Core software engineering activities studied in ESOFT

- Requirements gathering
- Feasibility study
- Analysis
- Design
- Implementation
- Testing
- Deployment
- Configuration management
- Project management
- Operations and Maintenance

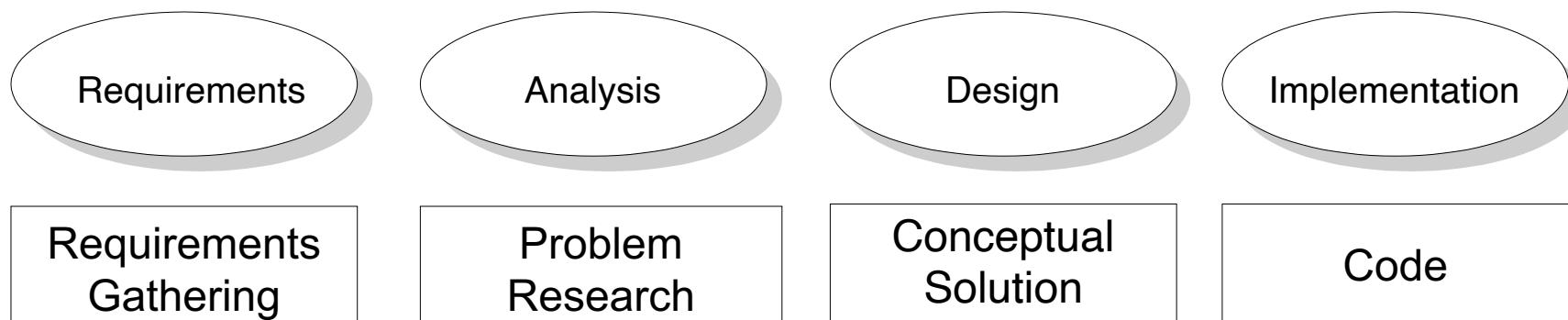
+ Human Factors  
(teamwork and communication)

# ESOFT: Goals

- Initial and core knowledge, capabilities and competencies of Software Engineering (SE), especially in the scope of information systems, about:
- Iterative and Incremental (I&I) Software Development Process (SDP) regarding
  - Requirements engineering
  - Analysis based on Object Oriented Paradigm (OOP)
  - Design based on Object Oriented Paradigm (OOP)
  - Implementation (prototype) of the resulting design

# SW development is more than programming

- Knowing a programming language is necessary but not enough to create (good) software
- Between a good idea and good software there is much more than programming, namely



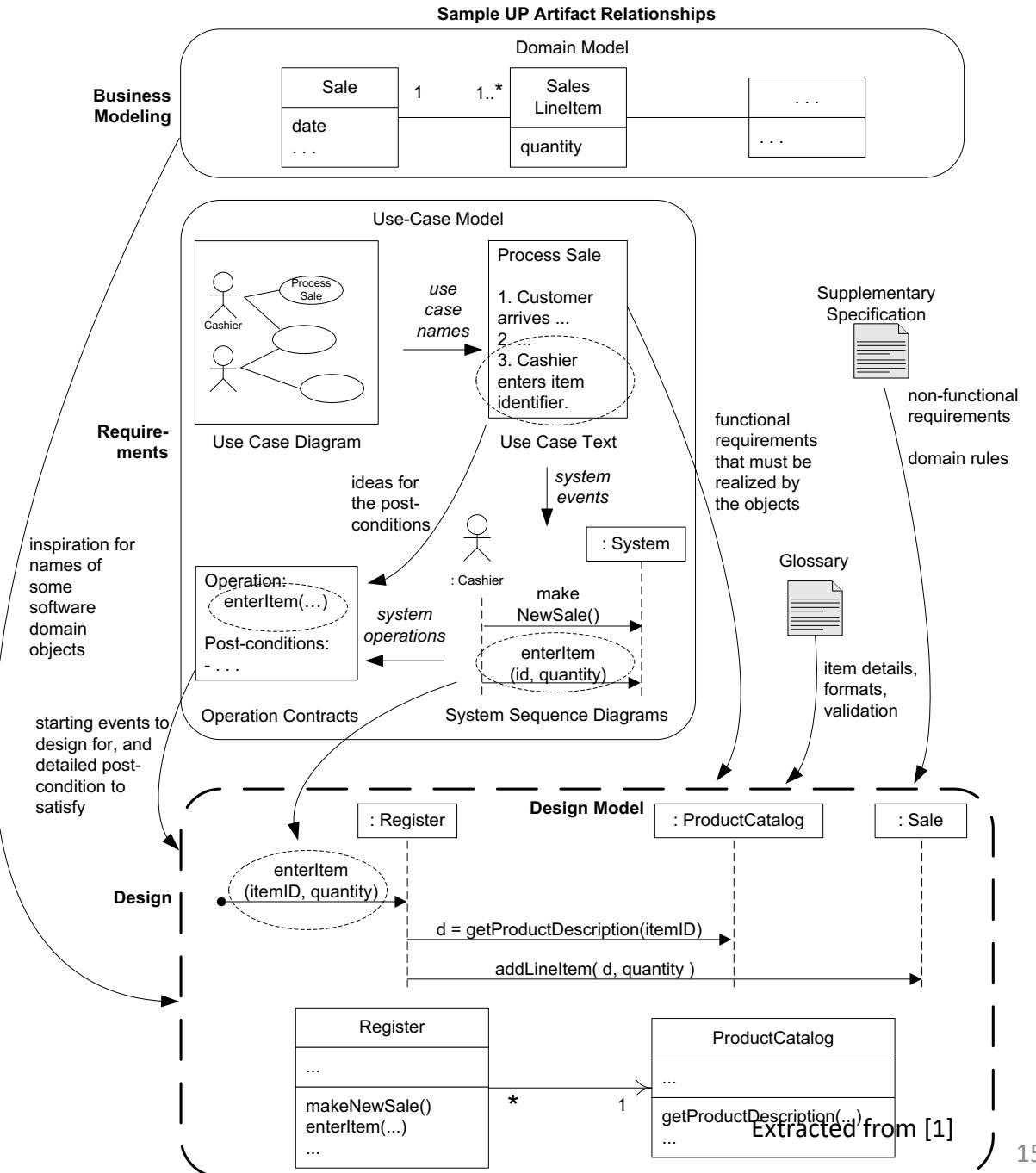
# But also...

- SW development process
- Documentation and information sharing during the process (e.g., UML)
- Tests
  - Unitary
  - Functional
  - Integration
  - Acceptance
- Good practices of requirements, analysis, design, coding, ...
- Human Factor/Skills:
  - Teamwork (soft skill)
  - Communication (soft skill)

# Requirements vs. Analysis vs. Design

- Requirements
  - Identify what the stakeholders want
  - Clarify and record restrictions and requirements
- Analysis
  - Do the right thing
  - Problem and requirements research
  - Exploring requirements details (e.g., OO analysis)
- Design
  - Do the thing right
  - Conceptual solution that aims to fulfill the requirements
  - Lead to implementation, but it is not implementation
    - e.g., architectural design, OO use-case design, database design
- Implementation
  - Build the thing

# Artifact overview



# SDP throughout an Example

(extracted/adapted from a previous project)

# Project – Platform for Outsourcing Tasks

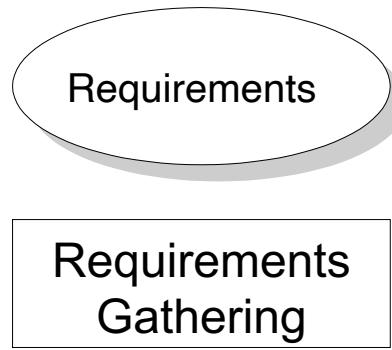
- Excerpts from the project's specification document:

A startup needs to develop a software product that, on the one hand, allows any interested organization to be registered on it, in order to be able to publish tasks and manage the process of awarding those tasks to freelancers; and on the other hand, allows freelancers to easily access these tasks and be able to apply for them.

It is expected that the product will be accessed by several users with different roles, such as:

- Organization Employee: someone acting on behalf of a particular organization. Employees have the responsibility to specify and catalog tasks it for later publication by the organization;
- Freelancers: people who propose to perform the tasks published by the organizations.

# From requirements to code (1/4)

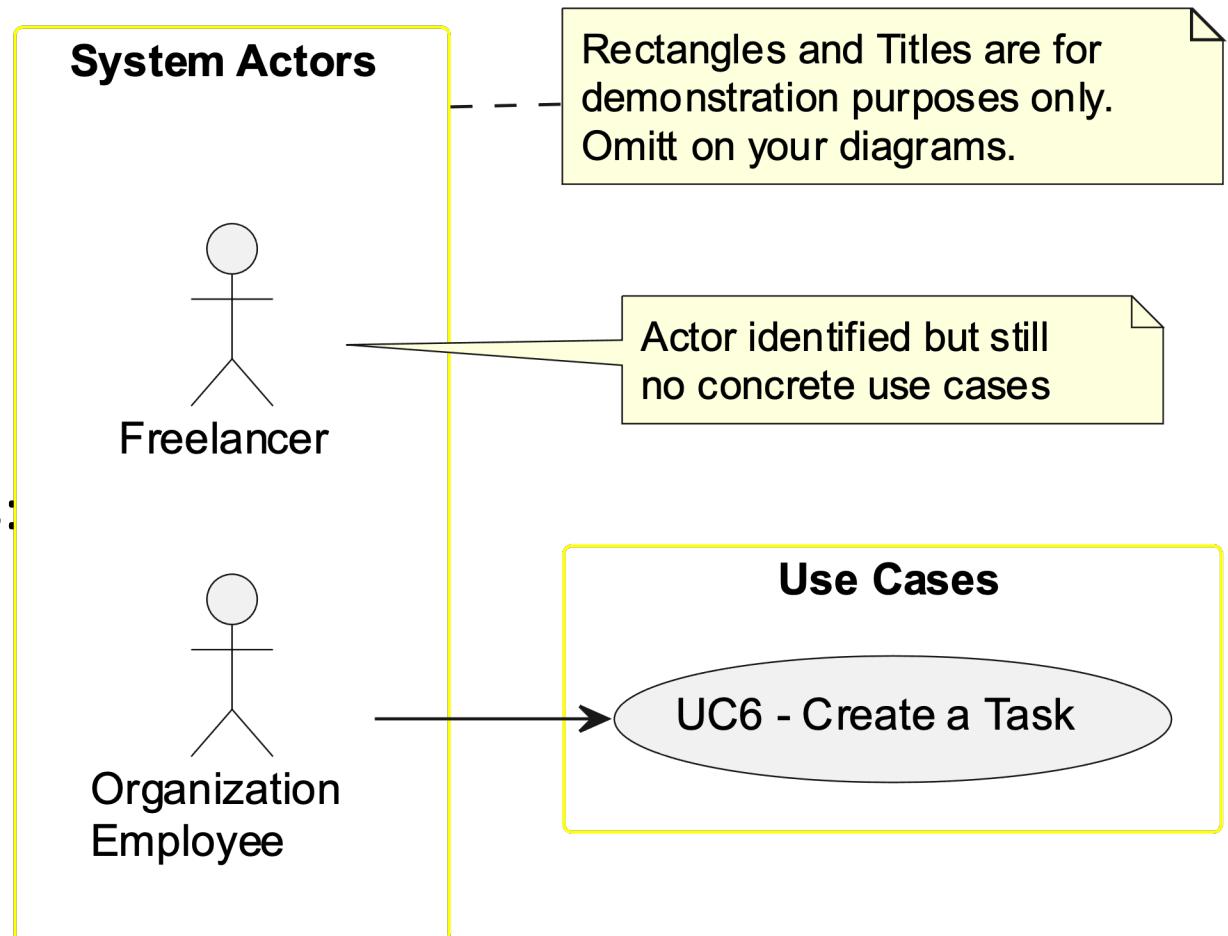


# Requirements

- Identify what stakeholders want or need
    - Which are the stakeholders and their interests?
    - What is the system purpose?
    - Which are the system boundaries?
    - Which are the system actors?
  - Record restrictions and requirements
    - (In)Formal documents/artifacts
    - Forming a common understanding
- 
- Done by:
- Documents' analysis
  - Interviewing users and stakeholders
  - Task observation
  - Workshops
  - Questionnaires

# Use Case Diagram (UCD)

- Identifying System Actors
  - Freelancer
  - Organization Employee
- Use Cases
  - Elementary business processes
  - Performed in the system by actors:
    - “Organization Employee” “creates a task”



# Capturing Requirements

## **US 006 - To create a Task**

### **1. Requirements Engineering**

---

#### **1.1. User Story Description**

As an organization employee, I want to create a new task in order to be further published.

#### **1.2. Customer Specifications and Clarifications**

##### **From the specifications document:**

Each task is characterized by having a unique reference per organization, a designation, an informal and a technical description, an estimated duration and cost as well as the its classifying task category.

As long as it is not published, access to the task is exclusive to the employees of the respective organization.

##### **From the client clarifications:**

**Question:** Which is the unit of measurement used to estimate duration?

**Answer:** Duration is estimated in days.

---

**Question:** Monetary data is expressed in any particular currency?

**Answer:** Monetary data (e.g. estimated cost of a task) is indicated in POTs (virtual currency internal to the platform).

# Capturing Requirements (cont.)

## 1.3. Acceptance Criteria

- **AC1:** All required fields must be filled in.
- **AC2:** Task reference must have at least 5 alphanumeric chars.
- **AC3:** When creating a task with an already existing reference, the system must reject such operation and the user must have the chance to modify the typed reference.

## 1.4. Found out Dependencies

- There is a dependency to "US003 Create a task category" since at least a task category must exist to classify the task being created.

## 1.5 Input and Output Data

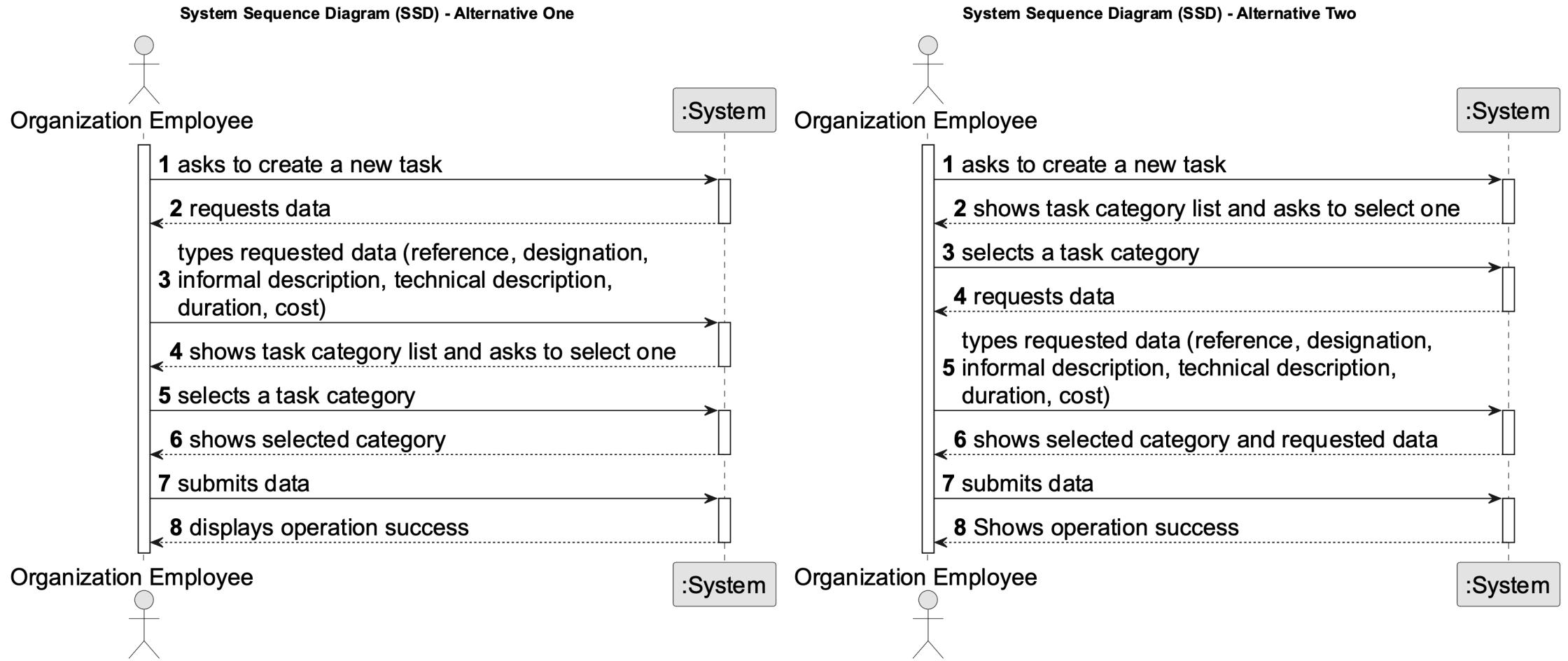
### Input Data:

- Typed data:
  - a reference,
  - a designation,
  - an informal description
  - a technical description
  - an estimated duration
  - an estimated cost
- Selected data:
  - Classifying task category

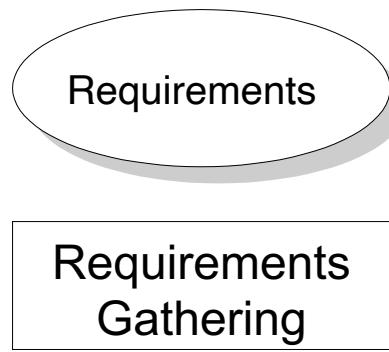
### Output Data:

- List of existing task categories
- (In)Success of the operation

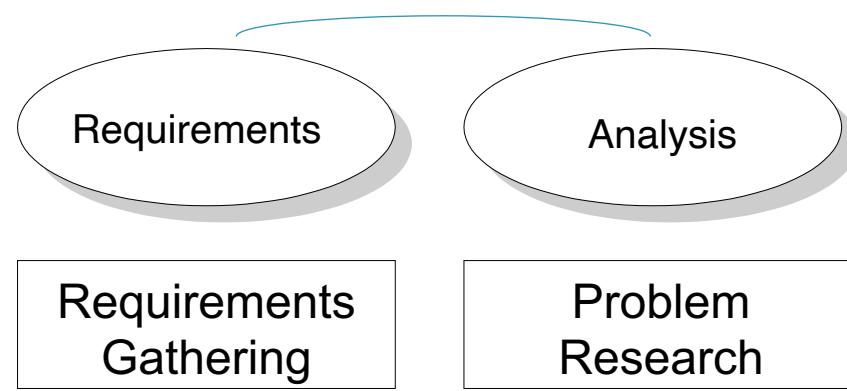
# Capturing Requirements (cont.)



# From requirements to code (2/4)



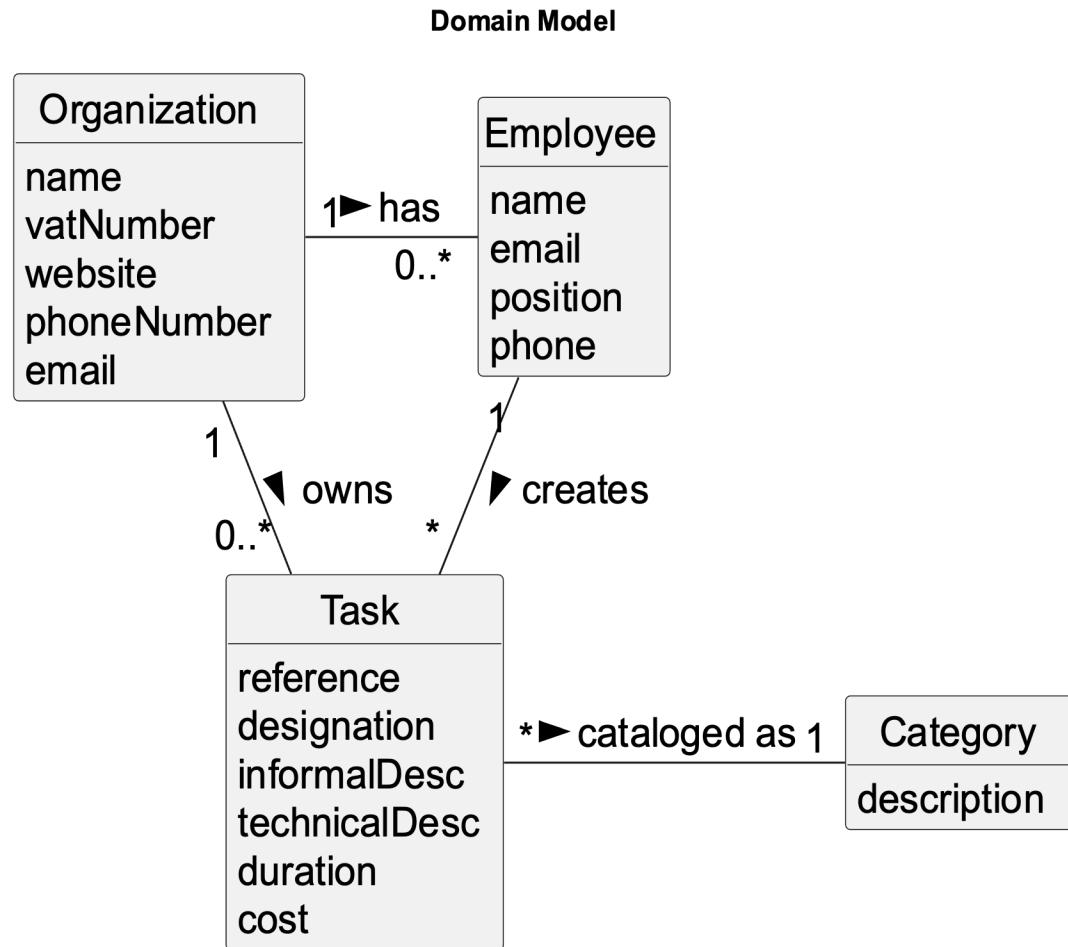
# From requirements to code (2/4)



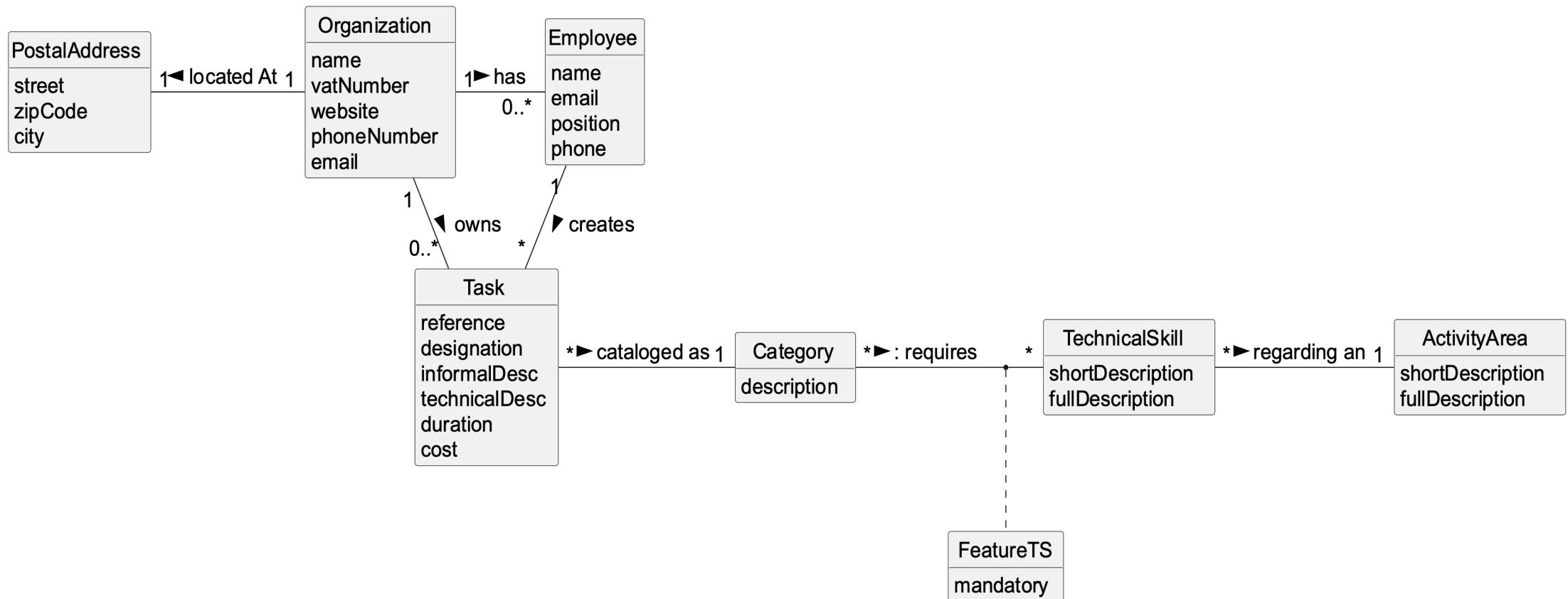
# Analysis

- It implies:
  - Acquiring domain/business knowledge
  - Understanding the domain/business
  - Reasoning about domain/business
- Analysis oriented by:
  - Objects → OO Analysis → Domain (Object) Model
    - Classifying domain elements/concepts
    - Finding relationships between domain concepts
  - Processes or Activities

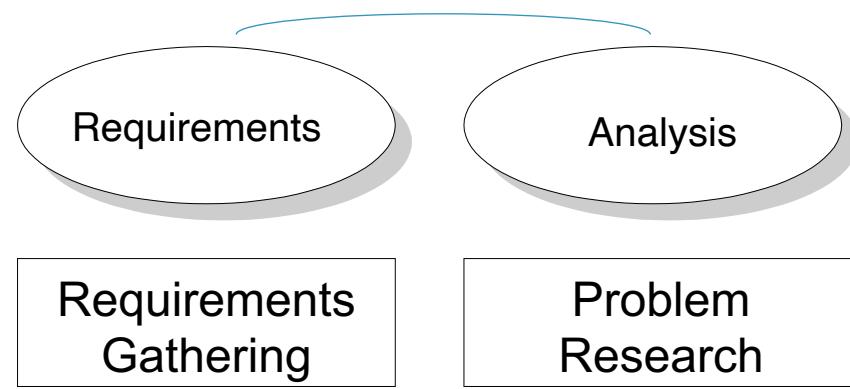
# Domain Model Excerpt Relevant for US 006



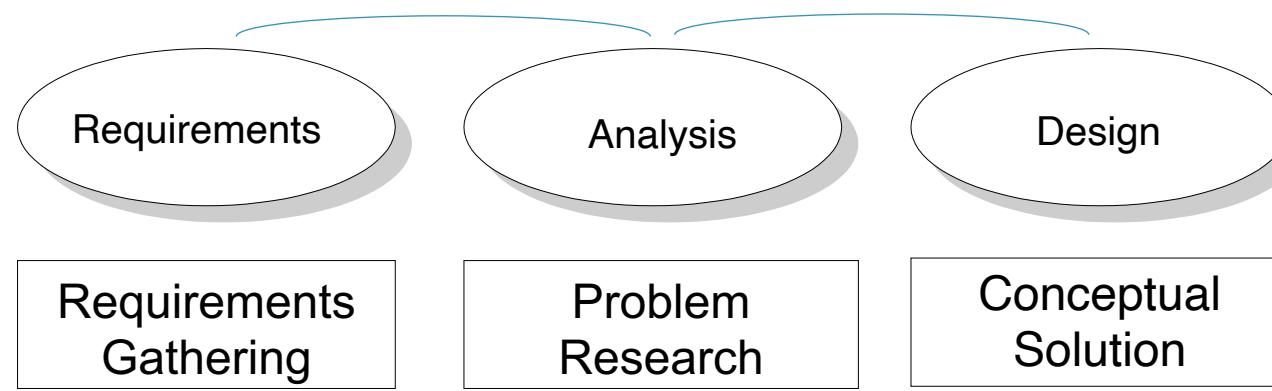
# Domain Model (for the Project)



# From requirements to code (3/4)



# From requirements to code (3/4)



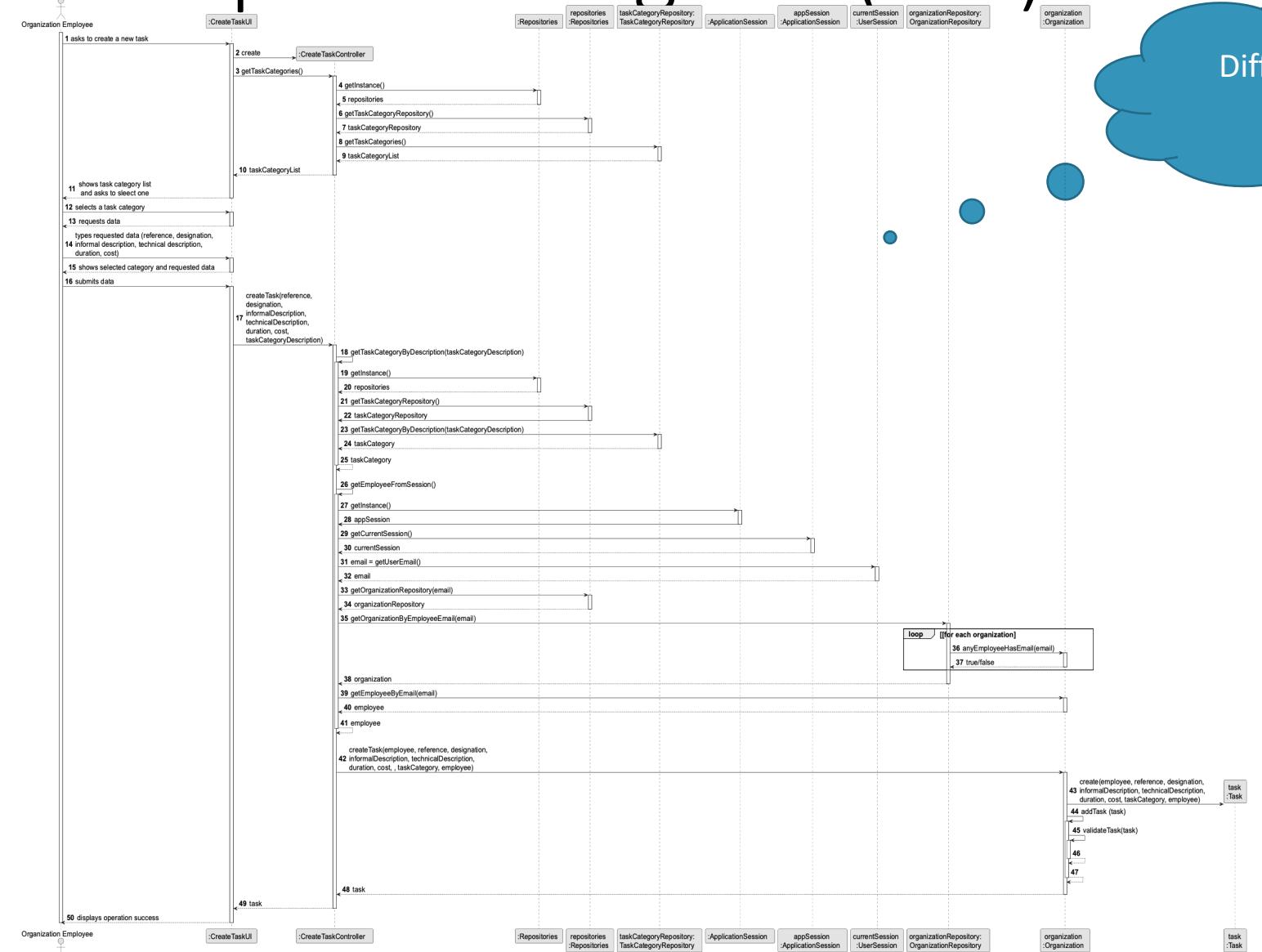
# Design – User Story Realization

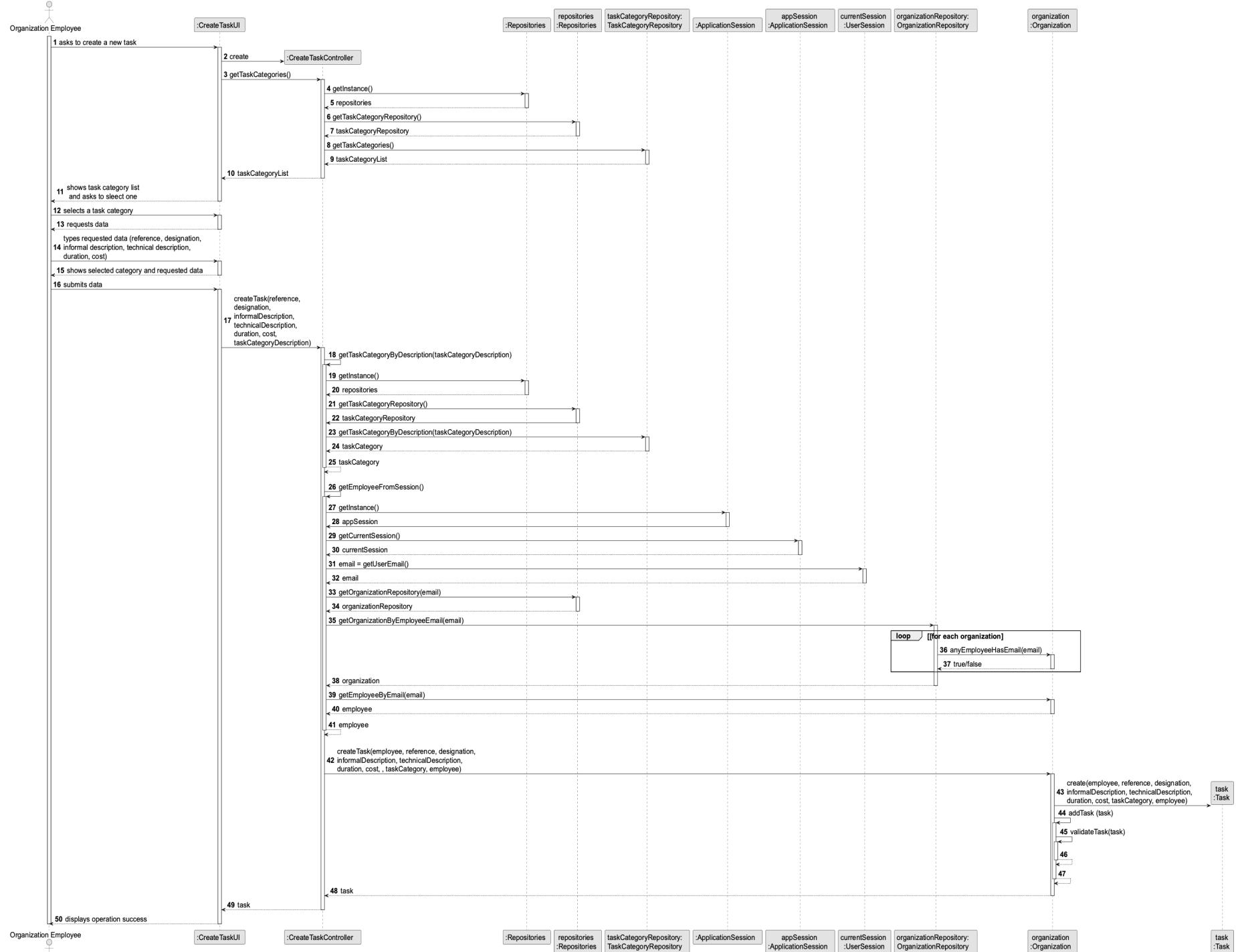
- Which pieces constitutes the software being developed?
  - Classes, their attributes and methods
  - But which classes?
- Rationale to determine which classes
  - Oriented by responsibilities assignment
  - Promoting domain concepts to software classes
  - Fabrication of pure software classes
- Design Model
  - Sequence diagram (SD) – Dynamic View
  - Class diagram (CD) – Static View
  - SD and CD are developed together (at the same time)

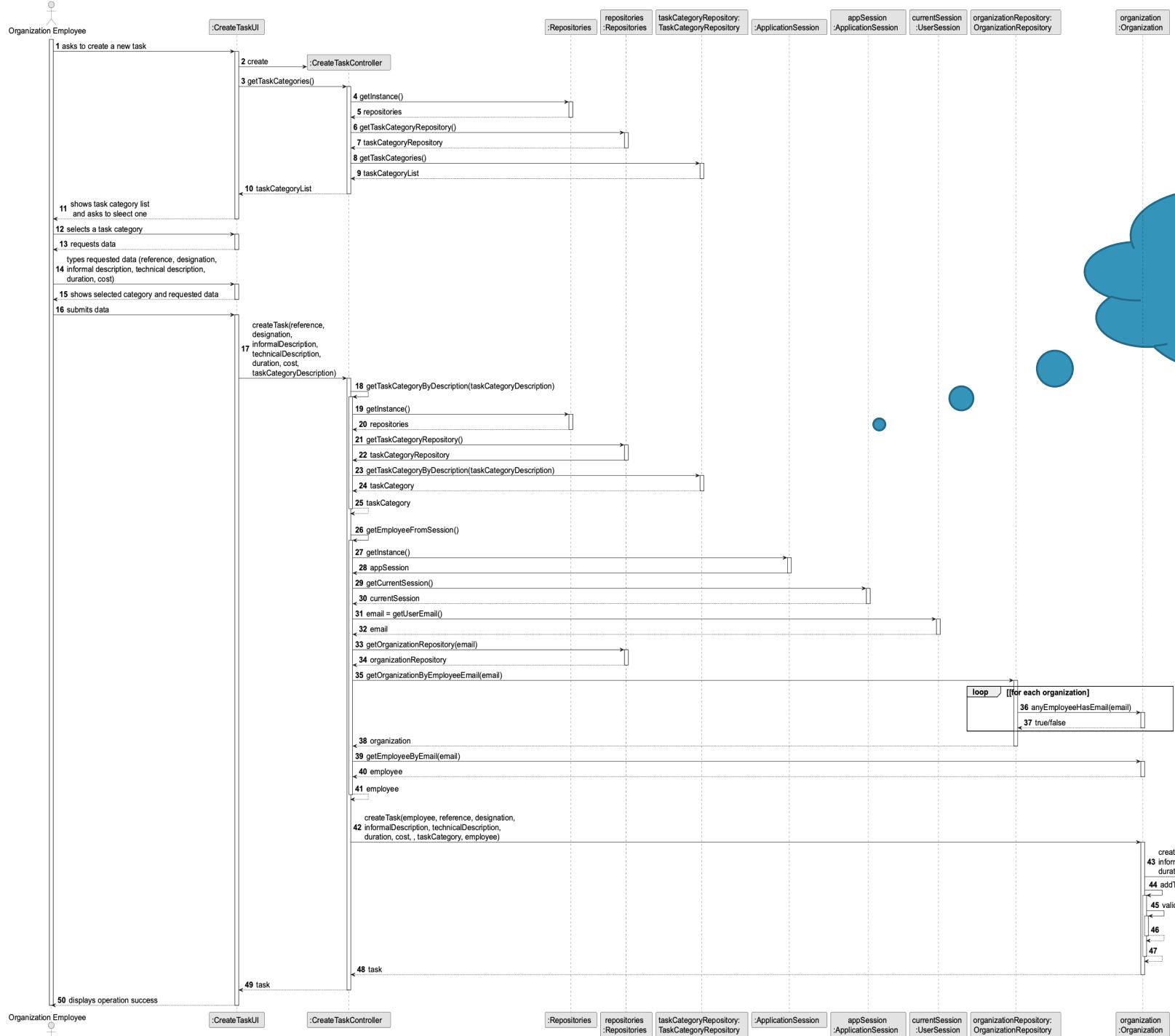
# US 006 – Sequence Diagram (Full)



# US 006 – Sequence Diagram (Full)

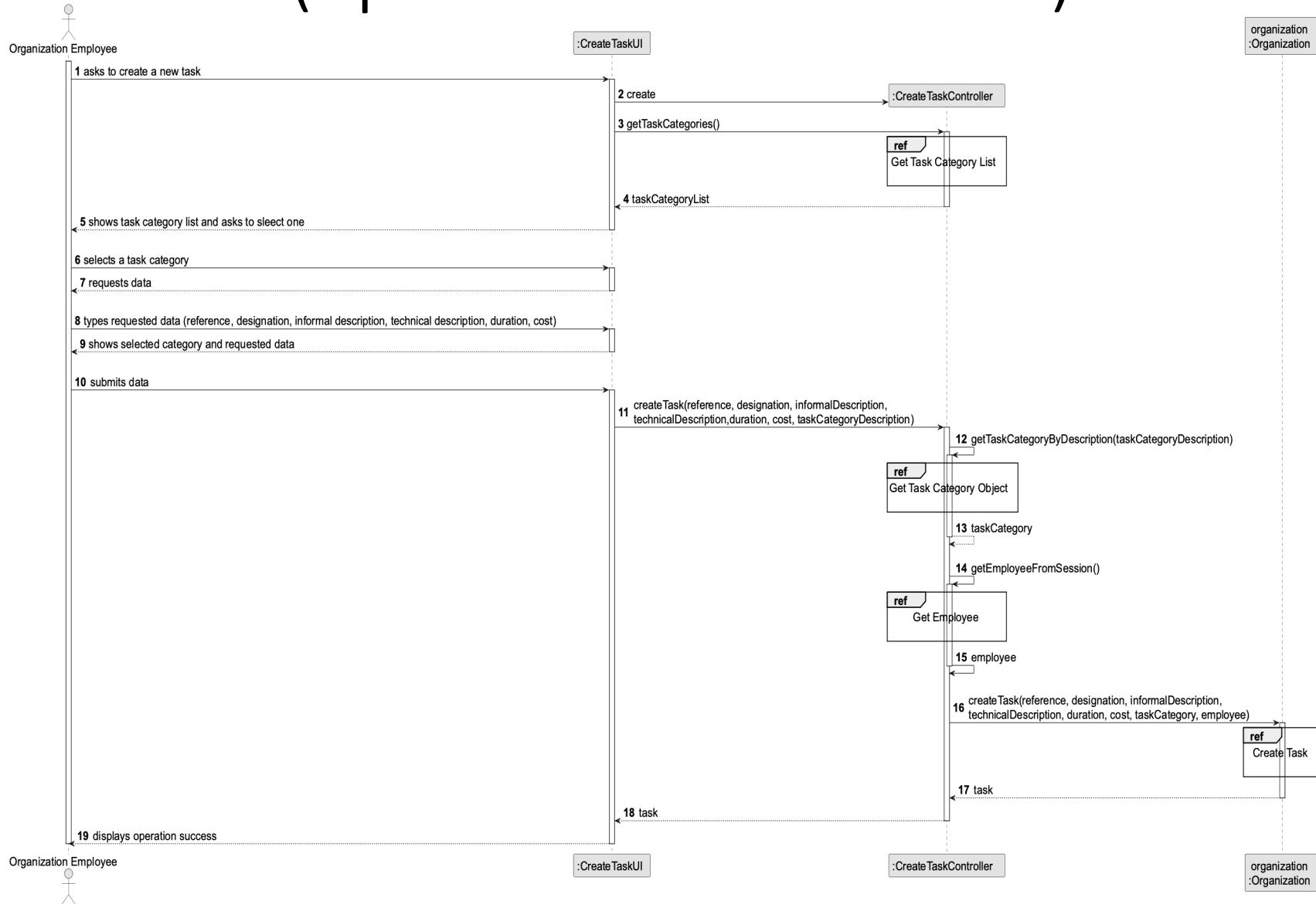


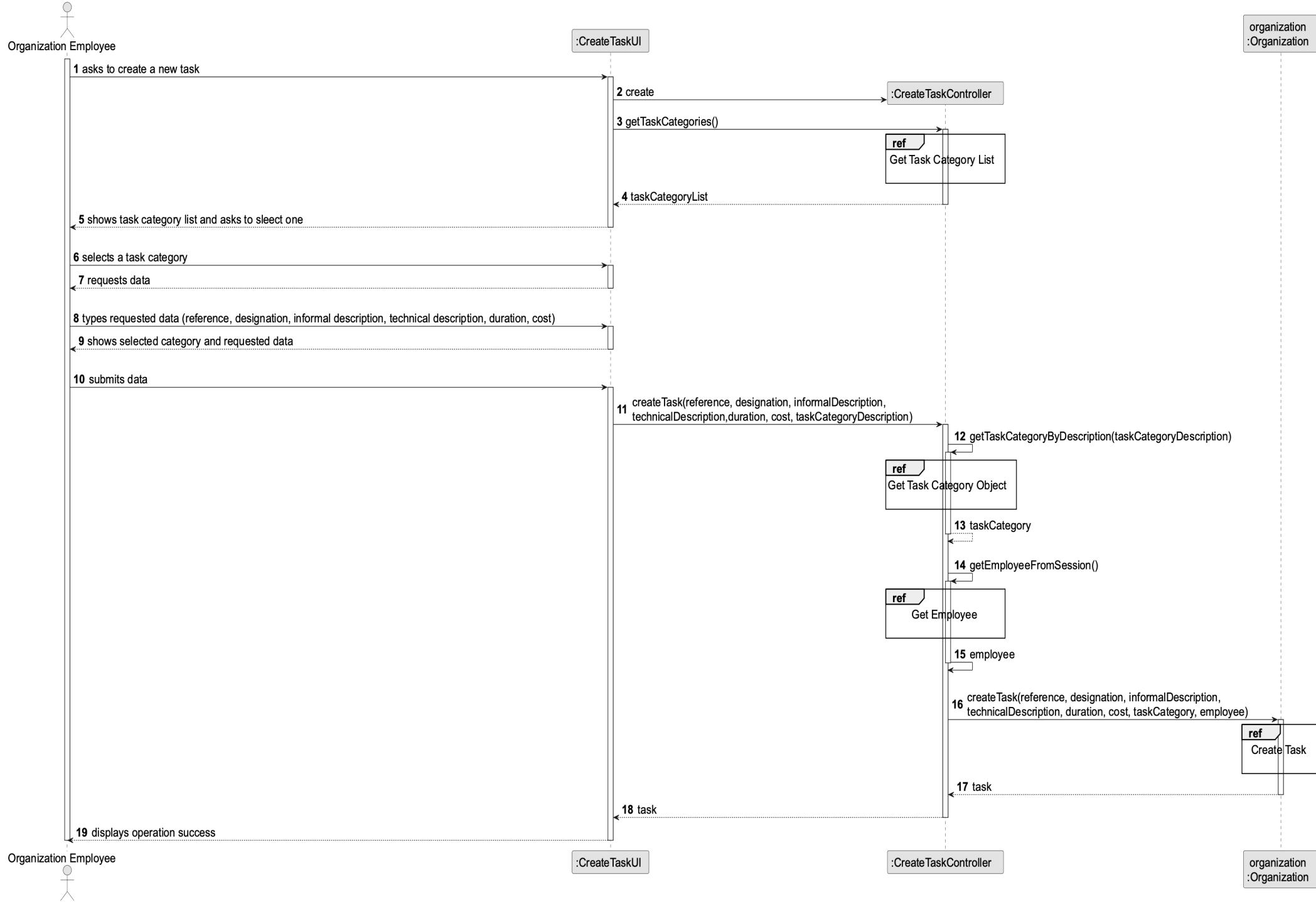




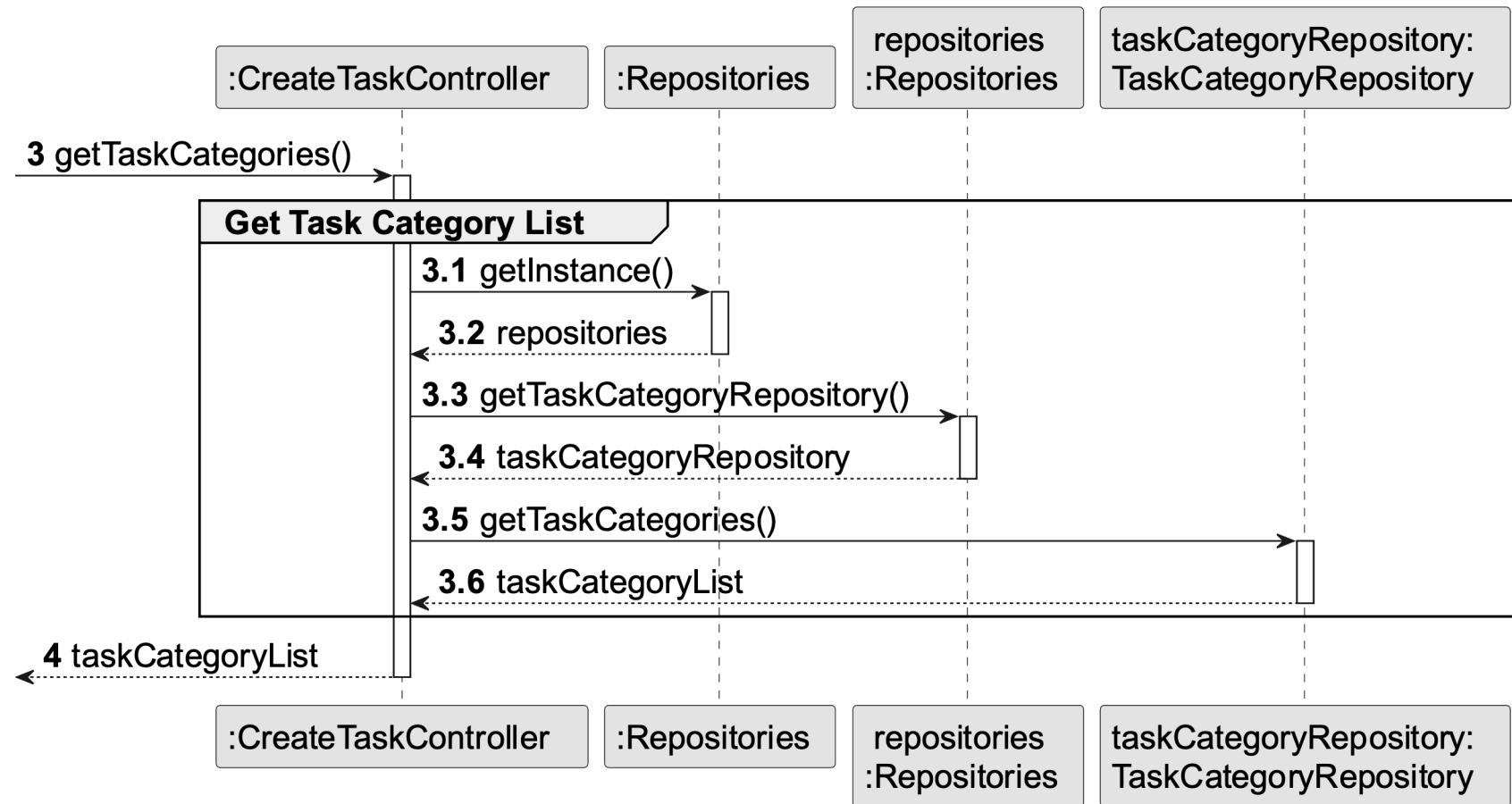
Still difficult to read,  
Reconsider a better  
design approach using  
Interaction Use

# US 006 – SD (Split Interaction Use)

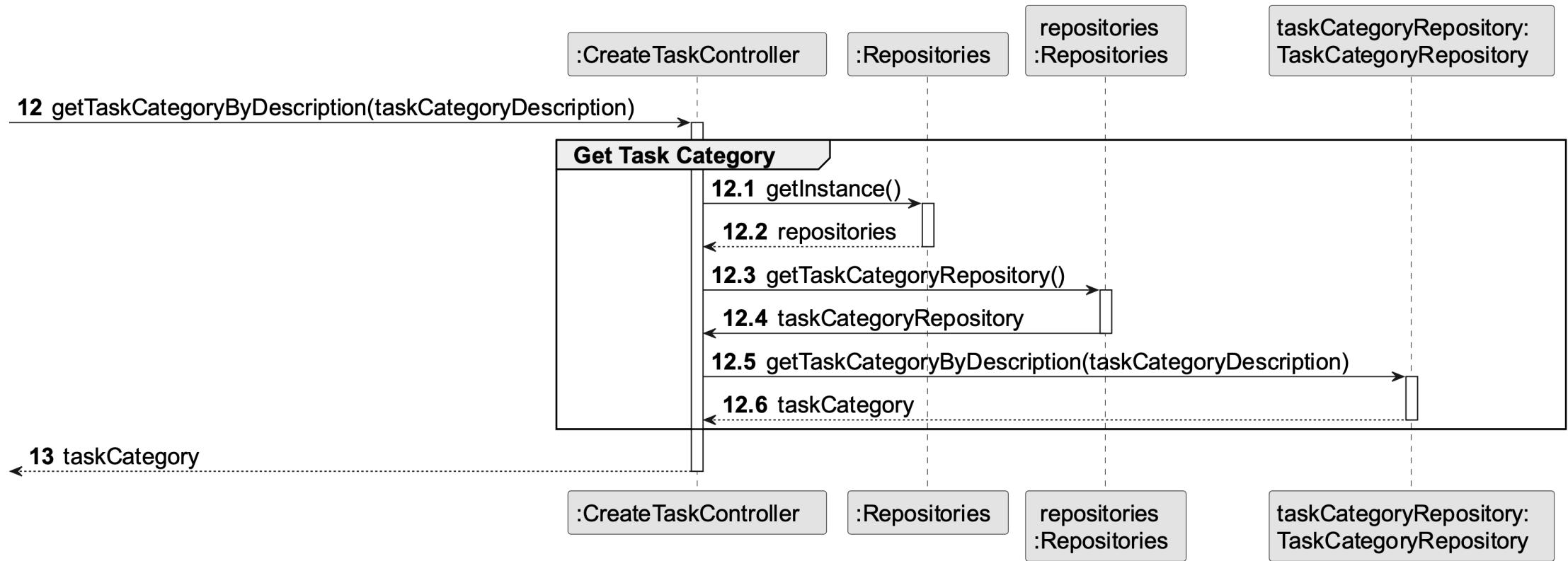




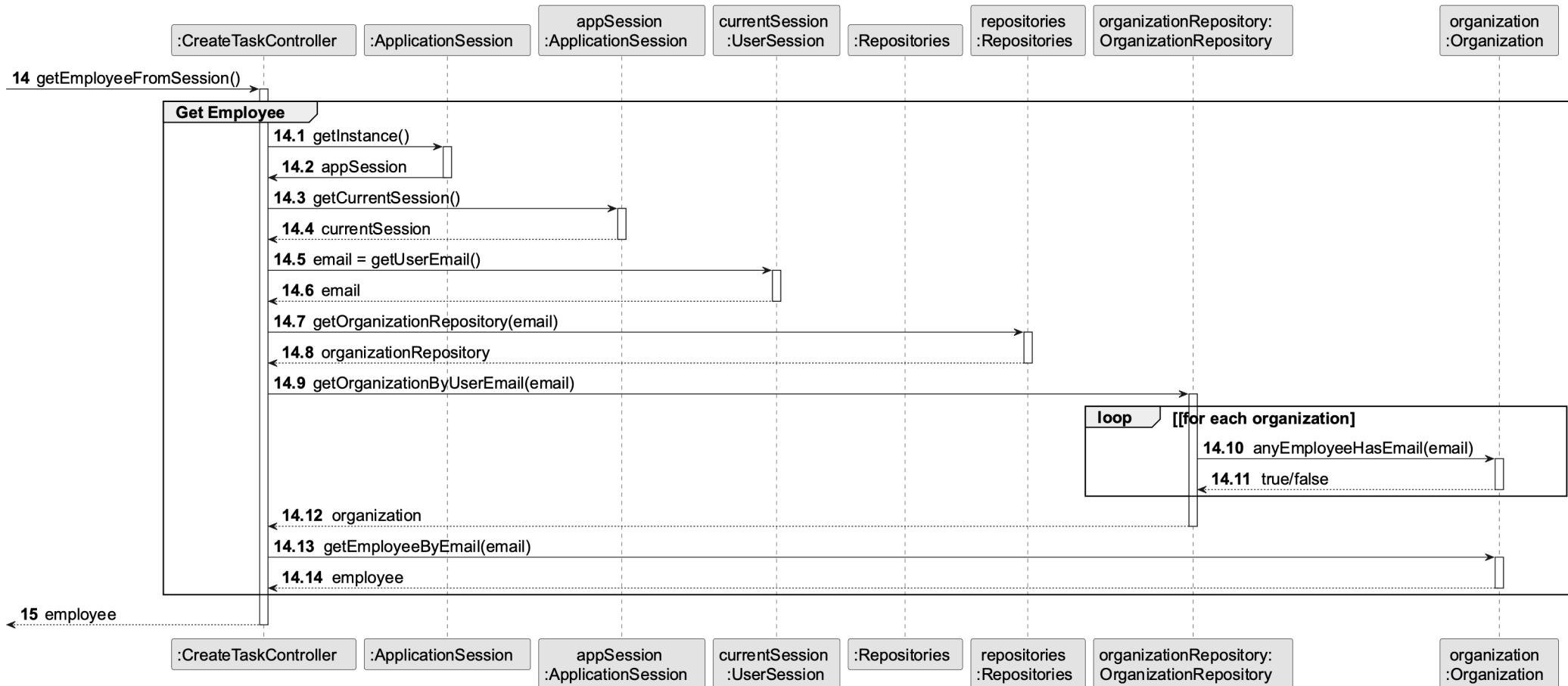
# US 006 – SD: ref: Get Task Category List



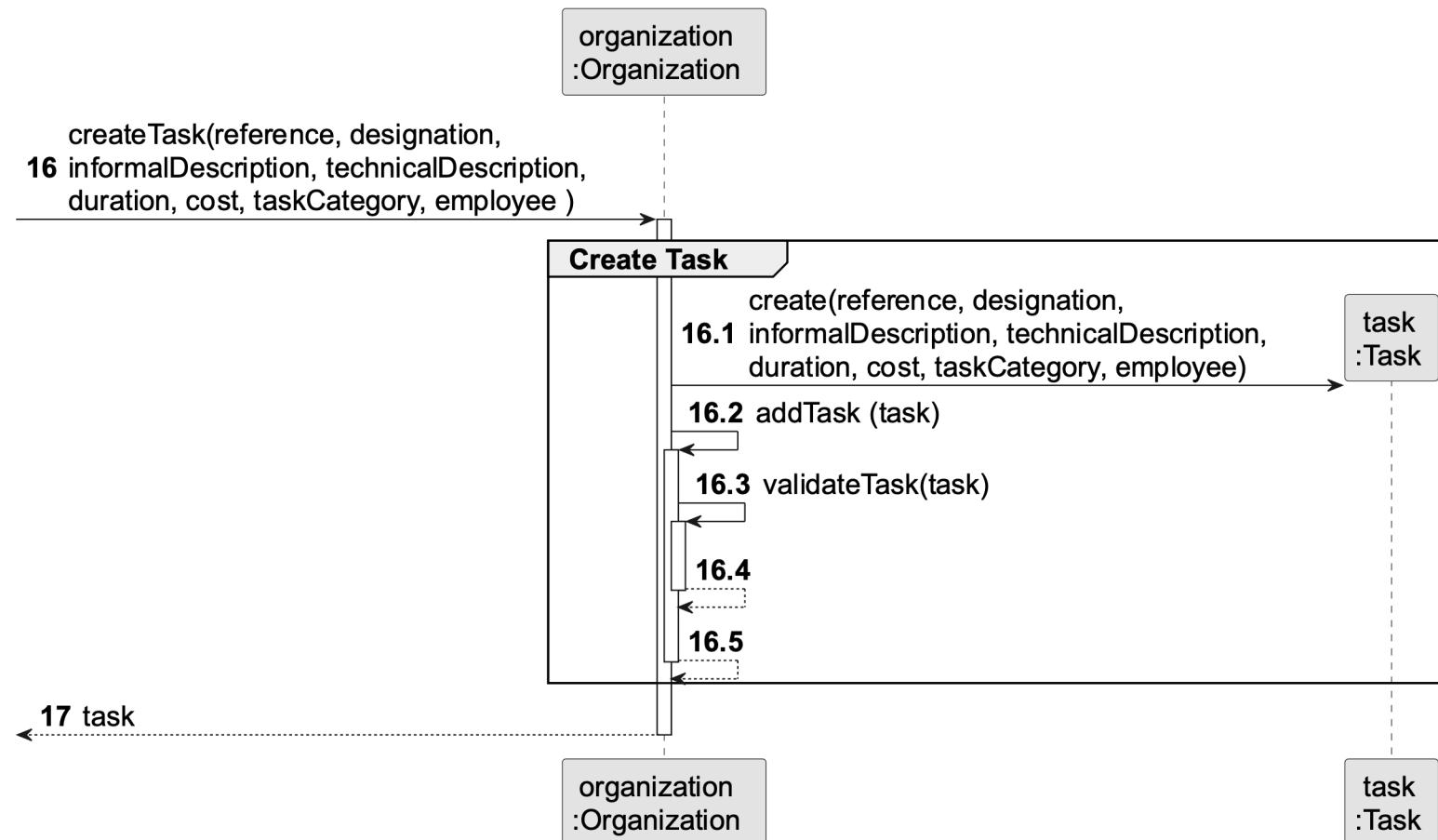
# US 006 – SD: ref: Get Task Category



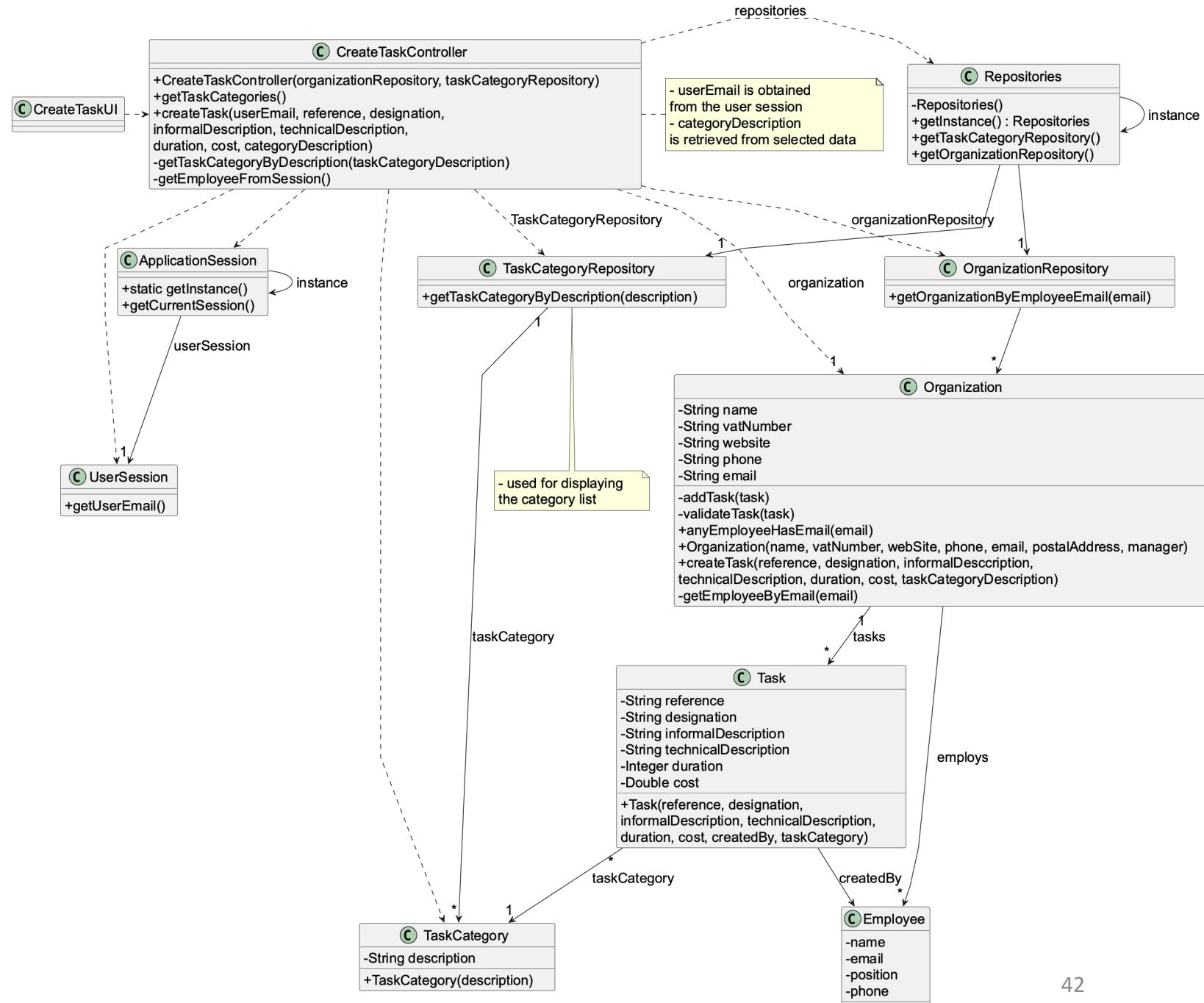
# US 006 – SD: ref: Get Employee



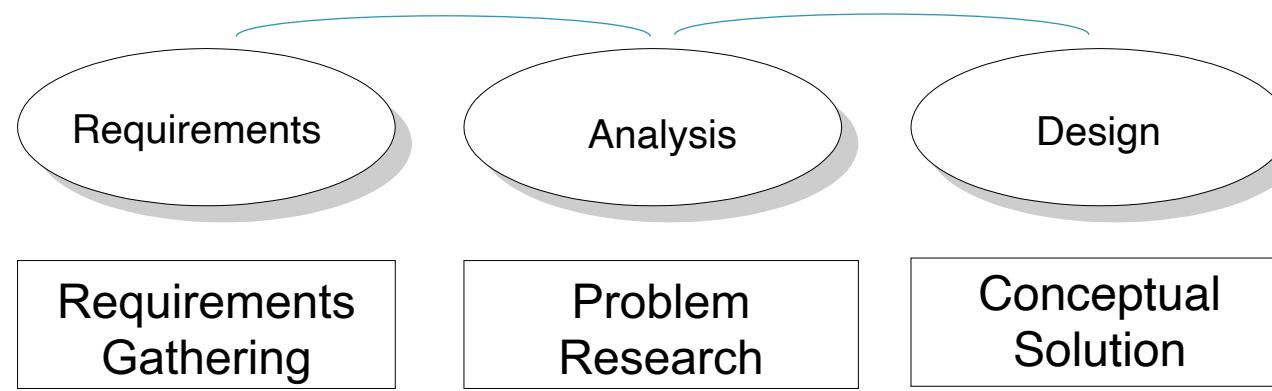
# US 006 – SD: ref: Create Task



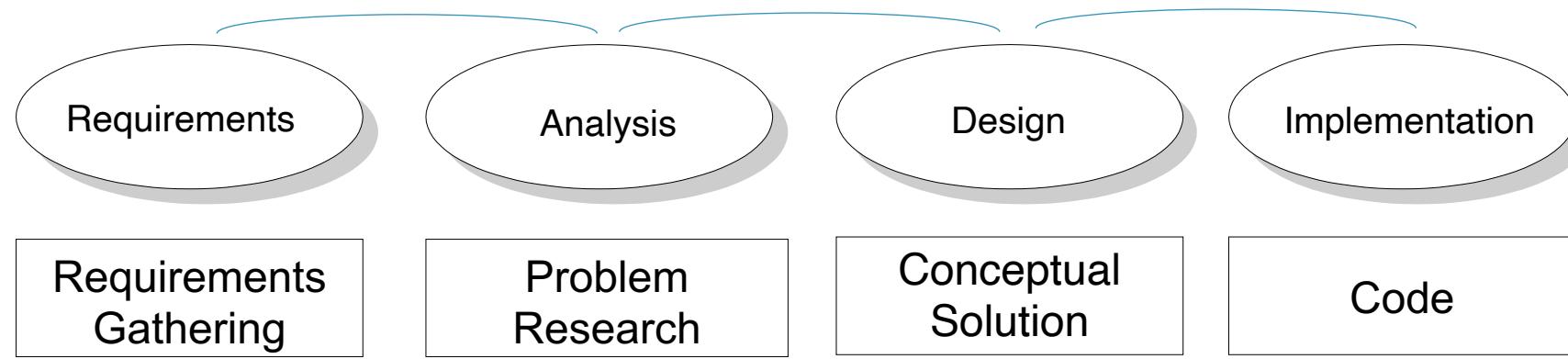
# US 006 – Class Diagram



# From requirements to code (4/4)



# From requirements to code (4/4)



# Coding (excerpt from CreateTaskController)

```
public Optional<Task> createTask(String reference, String description, String informalDescription,
                                 String technicalDescription, Integer duration, Double cost,
                                 String taskCategoryDescription) {

    TaskCategory taskCategory = getTaskCategoryByDescription(taskCategoryDescription);

    Employee employee = getEmployeeFromSession();
    Optional<Organization> organization = getOrganizationRepository().getOrganizationByEmployee(employee);

    Optional<Task> newTask = Optional.empty();

    if (organization.isPresent()) {
        newTask = organization.get()
            .createTask(reference, description, informalDescription, technicalDescription, duration, cost,
                       taskCategory, employee);
    }

    return newTask;
}
```

# Coding (excerpt from Organization Class)

```
public Optional<Task> createTask(String reference, String description, String informalDescription,
                                 String technicalDescription, Integer duration, Double cost,
                                 TaskCategory taskCategory, Employee employee) {

    // When a Task is added, it should fail if the Task already exists in the list of Tasks.
    // In order to not return null if the operation fails, we use the Optional class.
    Optional<Task> optionalValue = Optional.empty();

    Task task = new Task(reference, description, informalDescription, technicalDescription, duration, cost,
                         taskCategory, employee);

    if (addTask(task)) {
        optionalValue = Optional.of(task);
    }
    return optionalValue;
}
```

# Testing: Ensure is it not possible to create a Task Class with a null Reference

```
@Test
void ensureTaskReferencesNotNull() {
    //Arrange
    Employee employee = new Employee("john.doe@this.company.com");
    TaskCategory taskCategory = new TaskCategory("Task Category Description");

    //Act and Assert
    assertThrows(IllegalArgumentException.class,
        () -> new Task(null, "description", "informal description", "technical description", 1, 1d,
            taskCategory, employee));
}
```

# Testing: Ensure that it is not possible to add two Tasks with the same reference

```
@Test
void ensureAddingDuplicateTaskFails() {
    //Arrange
    Organization organization = new Organization("123456789");
    Employee employee = new Employee("john.doe@this.company.com");
    TaskCategory taskCategory = new TaskCategory("Task Category Description");
    //Add the first task
    Optional<Task> originalTask =
        organization.createTask("Task Description", "Task Category Description", "informal description",
            "technical description", 1, 1d, taskCategory, employee);

    //Act
    Optional<Task> duplicateTask =
        organization.createTask("Task Description", "Task Category Description", "informal description",
            "technical description", 1, 1d, taskCategory, employee);

    //Assert
    assertTrue(duplicateTask.isEmpty());
}
```

# Summary

# Promoted Working Method (1/2)



Some mentioned  
artifacts were not  
yet introduced!

- Sequence of Engineering Activities
  - Requirements
    - Use Cases / User Stories / Acceptance Criteria / FURPS+ / Other Texts
  - Analysis
    - Inputs&outputs / Domain Concepts / Domain Model
  - Design
    - Method signatures / Classes / Components / Modularization
  - Testing
    - Specify a set of tests covering all/most common and uncommon scenarios
  - Implementation
    - Code the design methods and classes
- Repeat the above sequence as needed for each use case

# Promoted Working Method (2/2)

- Activities
  - Each one has a well-defined information/artifacts as input
  - Each one has a well-defined artifacts as output
  - Outputs of one activity are used as inputs on other activities
- Supported by best enginnering practices
- Promotes best enginnering practices too
- Clear, simple and easy to adopt

# Bibliography

- [1] Larman, Craig; Applying UML and Patterns; Prentice Hall (3rd ed.); ISBN 978-0131489066
- [2] <https://www.scnsoft.com/blog/software-development-models>