

Software Development Process

A fast overview and a practical example



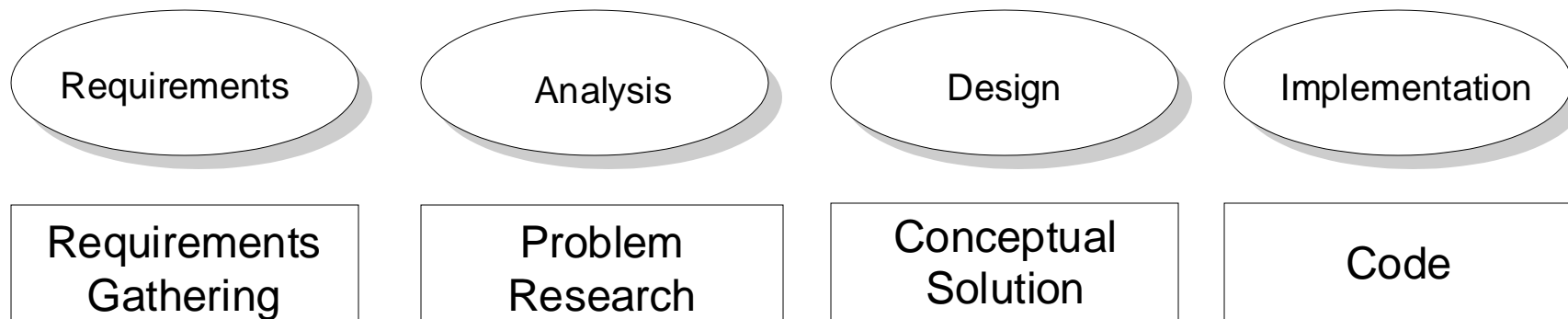
Topics

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

Software Development

SW Engineering 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...

- Documentation and information sharing during the process
- Tests
 - Unitary
 - Integration
 - System (End-to-End)
 - Acceptance
 - ...
- Good practices of requirements, analysis, design, coding, ...
- Human Factor/Skills:
 - Teamwork (soft skill)
 - Communication (soft skill)

Software as a Product

- It is intangible
- It is unusually flexible
- Many software projects are unique
- **But which are the qualities of a good software?**

Qualities of a good software

Metric	Description
Correctness	the degree to which each software adheres to the specified requirements
Portability	the degree of how easy each software can be used in different computer configurations
Reusability	the degree of how easy each software can be reused in the development of other software
Reliability	the frequency and criticality of software failures - a failure is an unacceptable effect or behavior occurring under permitted operating conditions
Maintainability	the degree of how easy changes can be made to the software to meet new requirements and/or correct deficiencies
Efficiency	the degree to which the software fulfills its purpose without wasting resources
...	...

Emphasis on these metrics varies from one project to another

Some problems in SW development

- Misunderstood business
 - Without direct communication, the development team starts guessing what the customer wants/needs
- Business changes
 - New laws, market changes, business priorities
- High defect rates
 - Software is put into production, but the defects are so high that it is not used
- Outdated system
 - Software is deployed to production, but after a while, costs of changes and of repairing defects dictate that the system needs to be replaced

Difficulties and Traps of SW development

Lack of context

Difficulty on identifying the users

Customer/users do not know what they really want/need

Failure on producing/getting value from the SW

Difficulties on prioritizing needs

Difficulties on estimating time and resources

If documents are too long... are not read

If documents are too technical... are not read

Difficulty on developing software incrementally

Feedback cycles are too long

Implementation criterion is “everything is implemented”

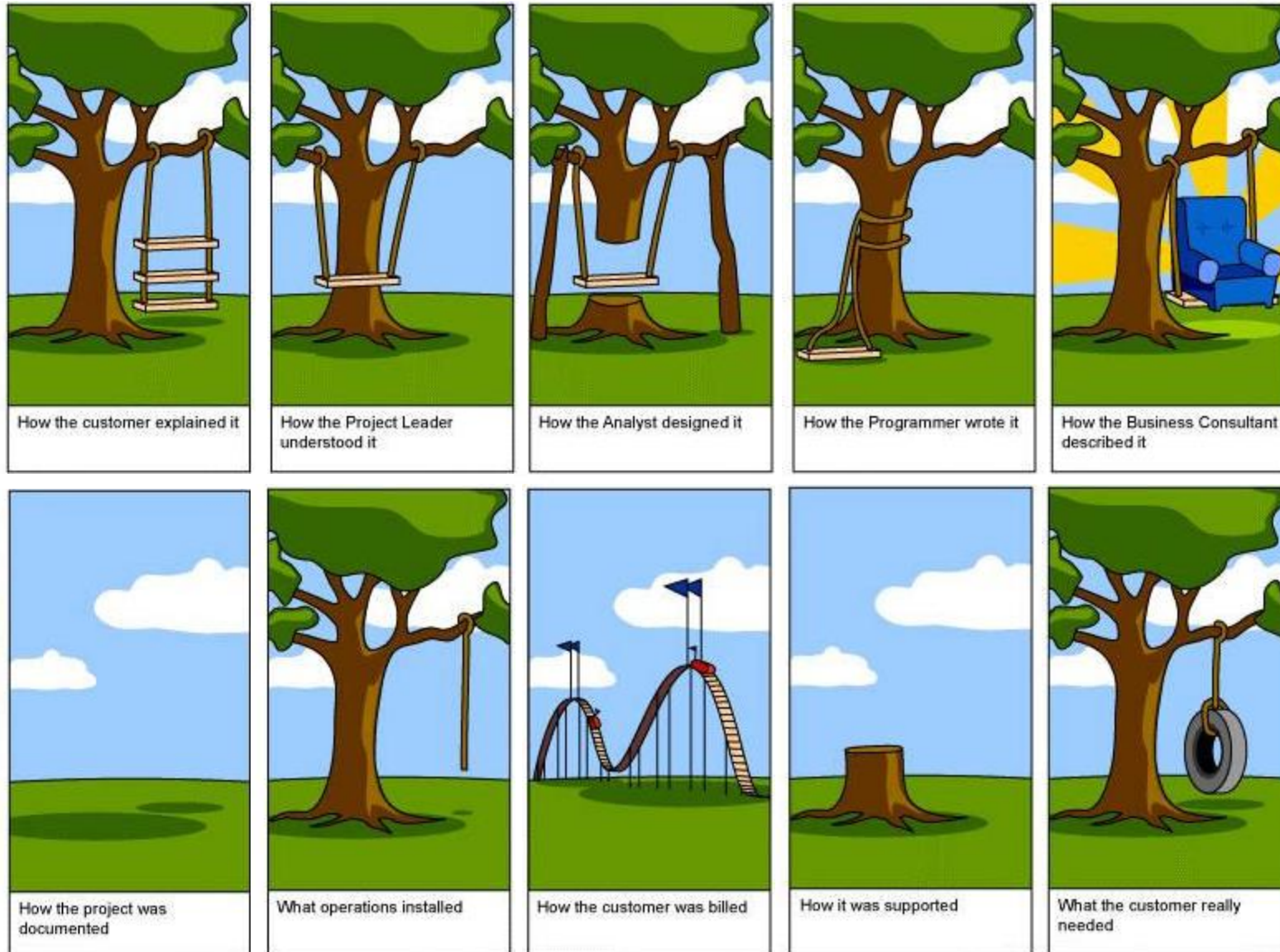
Long time-to-market

Difficulty on doing maintenance

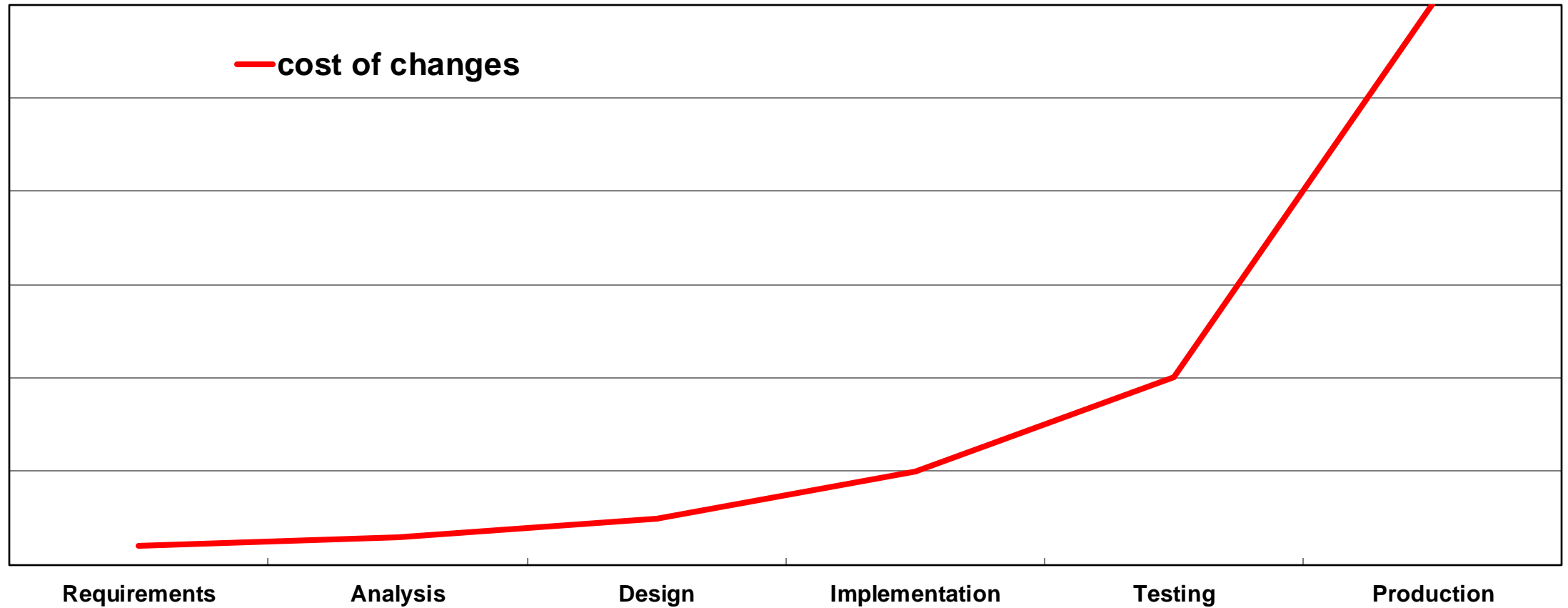
Communication (Failure)



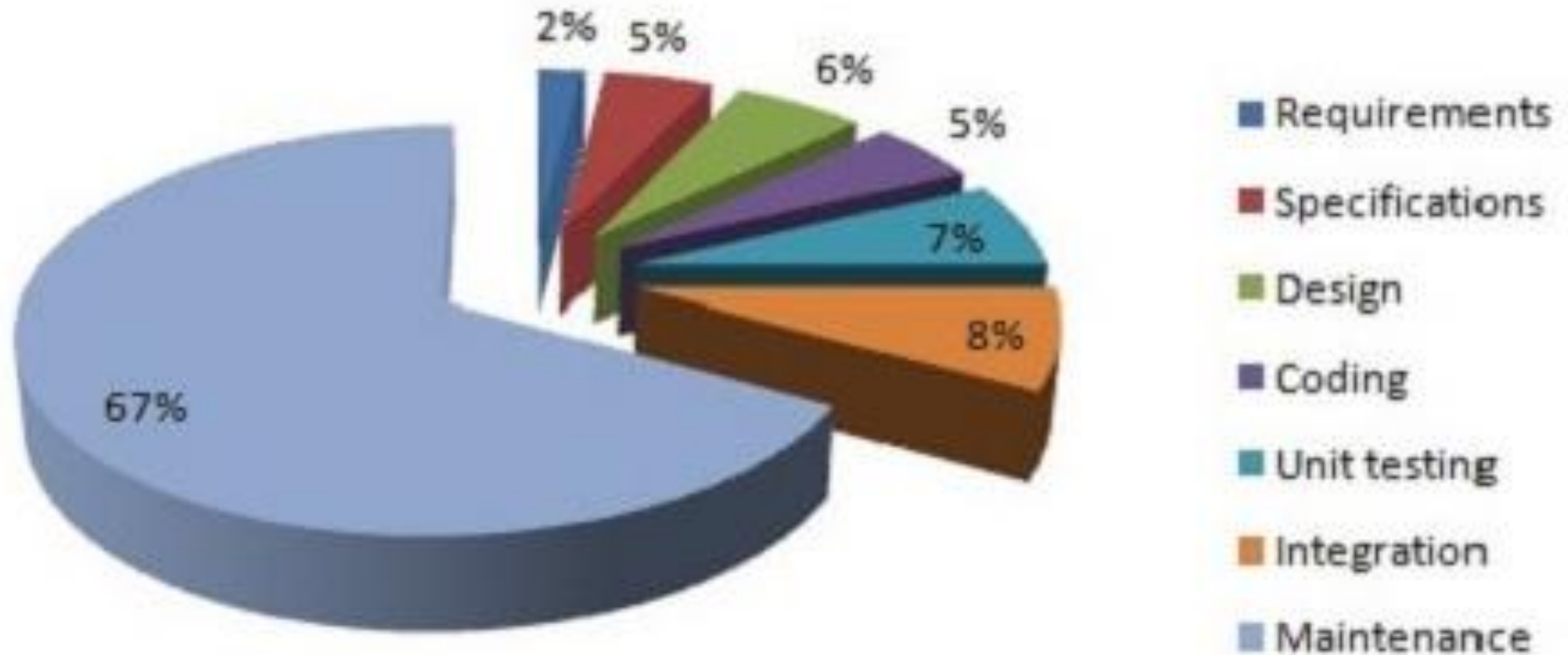
Metaphor / Perspectives



Costs of changing Software



Costs Per activity



Source : Digital Aggregates

...Considering these issues and...

- Software systems are increasingly important in our daily lives
- Software development failures can cost much more than the cost of the software and the time spent
- Effective software engineering is therefore fundamental and urgent
 - Introducing the notion of process / method / model
 - To impose a disciplined and well-defined process on SW development
 - More effective and efficient

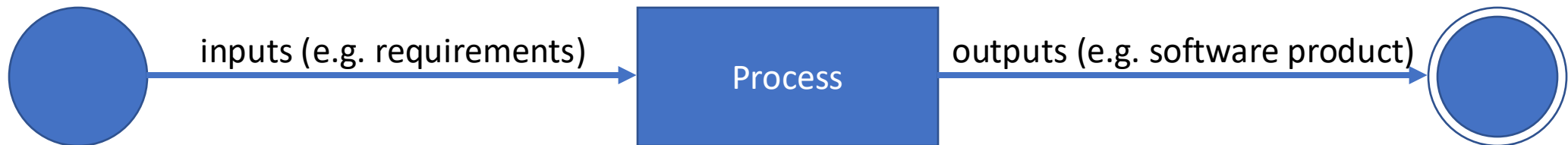
Software Development Process

Why do we need SDP for SW development?

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

What is a process?

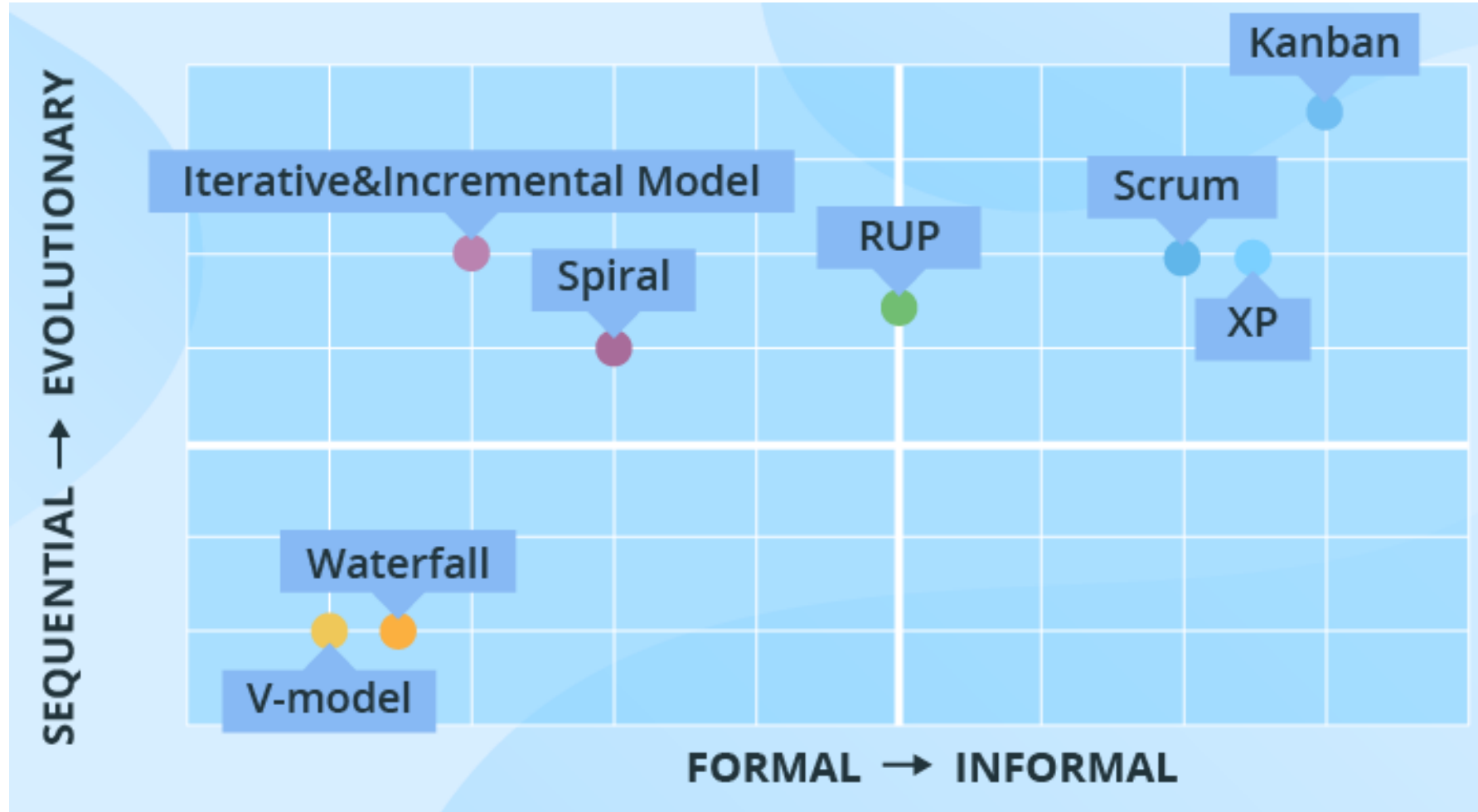
- A specification defining **who** does **what**, **when** and **how** in order to meet a given goal
- It comprehends a **set of interrelated activities** that transform a set of inputs in a set of outputs



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 SW development processes



Core Software Engineering Activities (1/6)

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

Core Software Engineering Activities (2/6)

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

+ Human Factors
(teamwork and communication)

Core Software Engineering Activities (3/6)

- **Requirements gathering**

- ~~Feasibility study~~

- **Analysis**

- **Design**

- **Implementation**

- ~~Testing~~

- ~~Deployment~~

- ~~Configuration management~~

- ~~Operations and Maintenance~~

- ~~Project management~~



+ Human Factors
(teamwork and communication)

Core Software Engineering Activities (4/6)

- **Requirements gathering**

- Aims at identifying and eliciting the requirements of the software product to be developed with the stakeholders

- **Feasibility study**

- Aims at assessing if the intended software project is feasible in terms such as technologically, financially and practically

- **Analysis**

- Aims at getting a deeper understanding of the scope of the project, related domain business and development constraints

- **Design**

- Aims at outcoming a conceptual solution that enables to fulfill the software requirements, comprising artifacts from a coarser granularity (i.e., architectural level) to a finer granularity

Core Software Engineering Activities (5/6)

- **Implementation**

- Aims at the programming/coding of the software program in a suitable language and in accordance with the design

- **Testing**

- Aims at the verification and validation of the developed software by specifying and conducting different kinds of tests

- **Deployment**

- Aims at installing (and integrating) the software in a computational environment (e.g. personal computer) in order to be used/operated by its users

Core Software Engineering Activities (6/6)

- Configuration management
 - Aims at managing, organizing and controlling changes in items
 - e.g. requirements, government policy and rules, code, project schedule/resources) that may affect the final product
- Operations and Maintenance
 - Aims at ensuring and keeping the software operating properly throughout time by, for instance, carrying out users training, fixing bugs/errors, updating the software to fulfill environmental and/or technological changes
- Project management
 - Aims at planning, scheduling, doing resource allocation, checking the execution progress of the software

Human Factors (Soft Skills)

- Teamwork
 - Software development involves and requires an effective cooperation and communication between several and distinct participants
 - e.g. customer, stakeholders, users, development team, project manager
- Communication
 - Essential and transversal to all activities aiming that the different participants communicate with each other by using the proper means
 - to promote the correct / intended interpretation of what is being communicated
 - e.g. natural/informal, semi-formal and formal languages

The Process to Adopt

1. Requirements

- Identify what the stakeholders want
- Clarify and record restrictions and requirements

2. Analysis

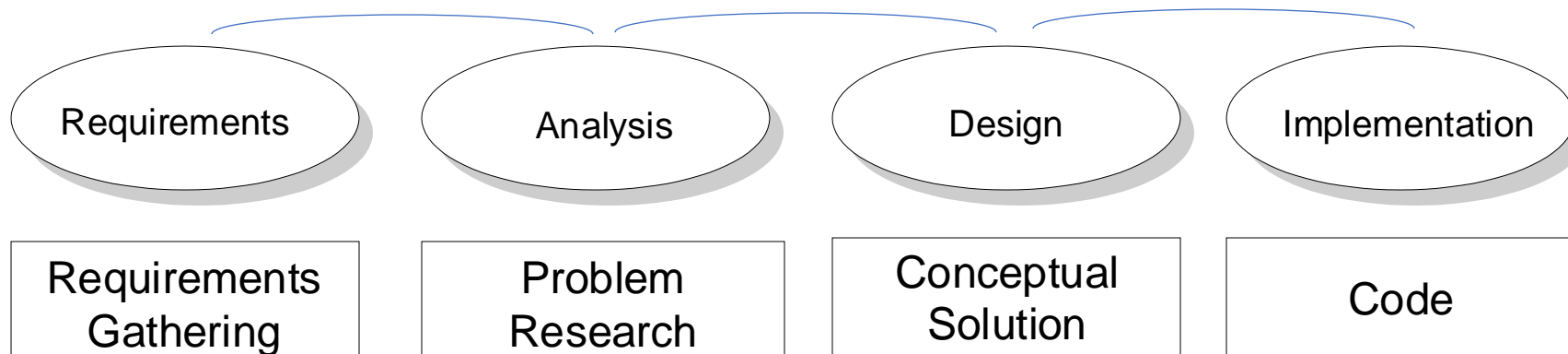
- Do the right thing
- Problem and requirements research
- Exploring requirements details (e.g. OO analysis)

3. Design

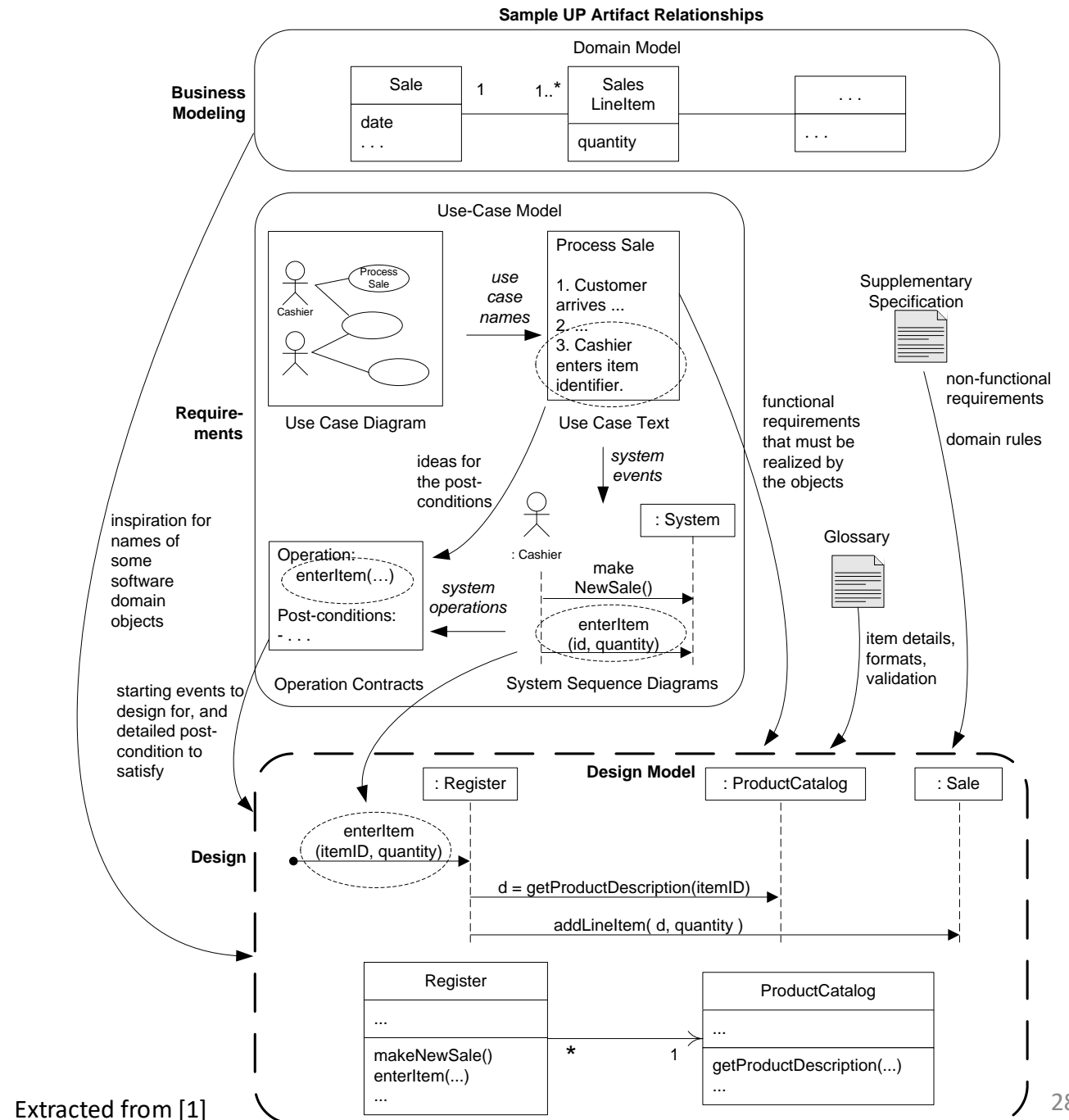
- Do the thing right
- Conceptual solution that aims to fulfill the requirements
- Lead to implementation, but it is not implementation

4. Implementation

- Build the thing



Artifact overview



SDP throughout an example

Platform for Outsourcing Tasks (adapted from another project)

Project – Platform for Outsourcing Tasks

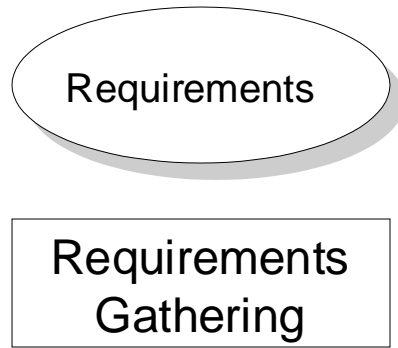
- Excerpts from the Project Specification Document

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

This product is expected to be accessed by several users with different roles, such as:

- Organization Employee: someone acting on behalf of a particular organization; employees are responsible for specifying and cataloging tasks for later publication by the organization.
- Freelancers: people who propose to carry out tasks published by 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

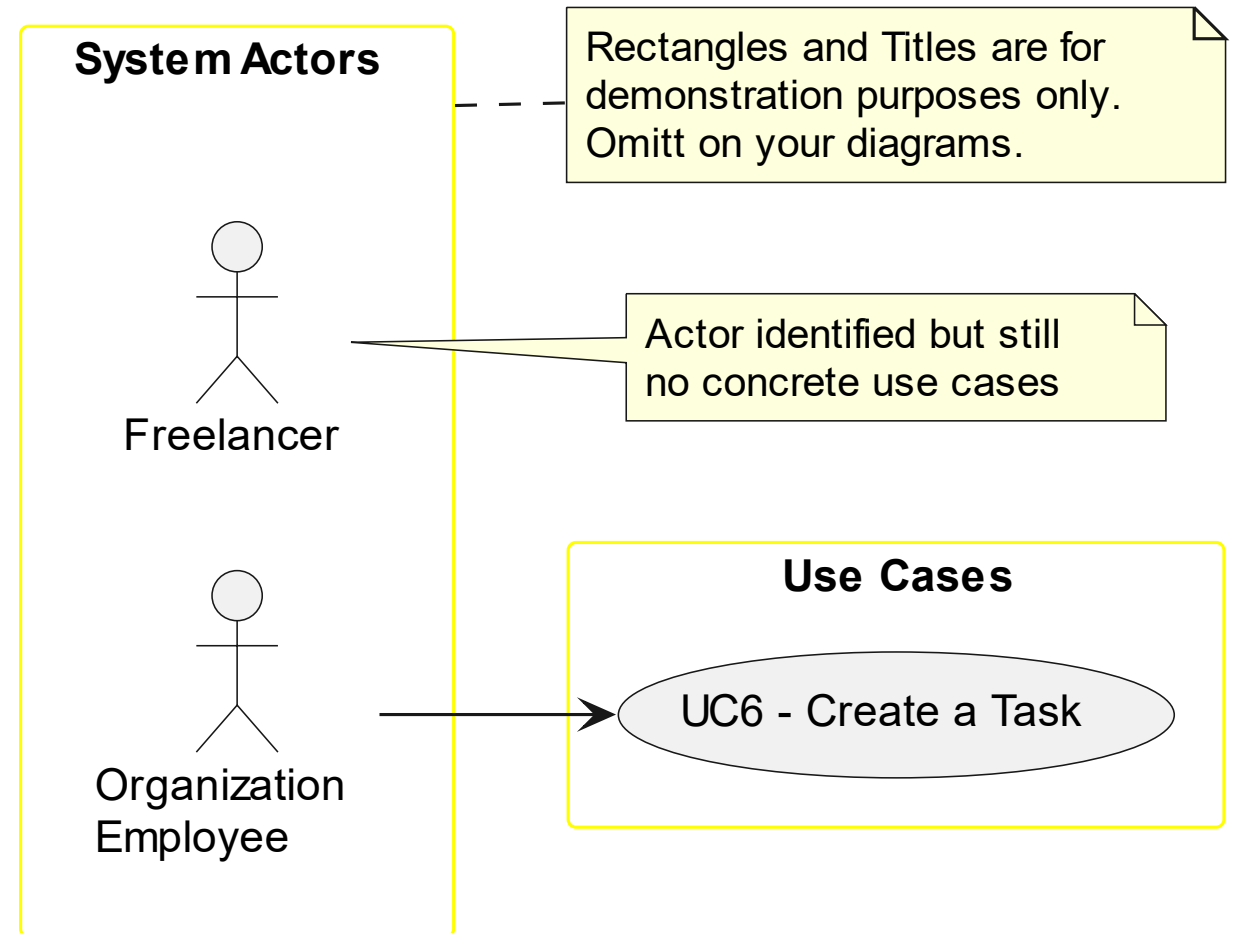
Done by:

- Document analysis
- Task observation
- Interviewing users and stakeholders
- Questionnaires
- Brainstorming

Use Case Diagram (UCD)

*Platform for
Outsourcing Tasks*

- Identifying System Actors
 - Freelancer
 - Organization Employee
- Use Cases
 - Elementary business processes
 - Performed in the system by actors



Capturing Requirements (1/3)

US006 - 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 a 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 POT (virtual currency internal to the platform).

Capturing Requirements (2/3)

1.3. Acceptance Criteria

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

1.4. Found out Dependencies

- There is a dependency on "US003 - Create a task category" as there must be at least one task category 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:
 - a task category

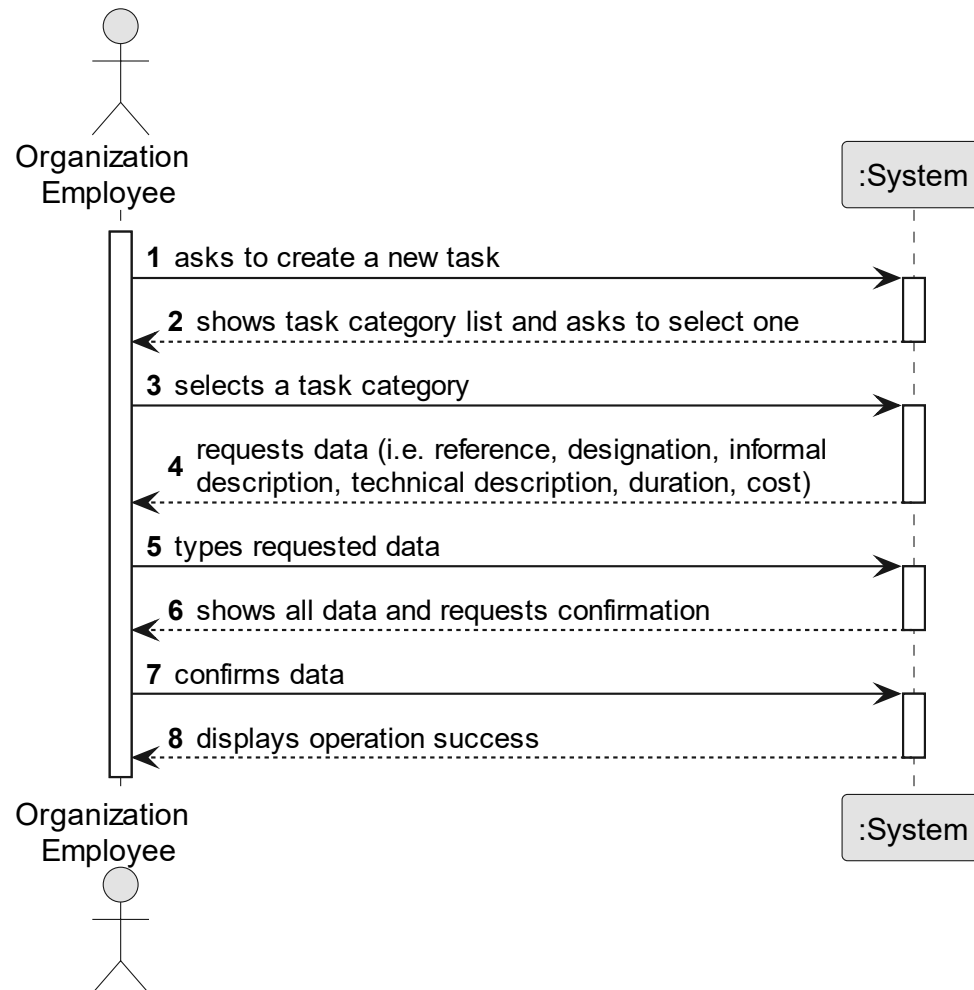
Output Data:

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

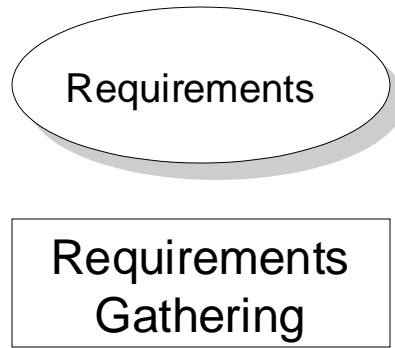
Capturing Requirements (3/3)

Platform for
Outsourcing Tasks

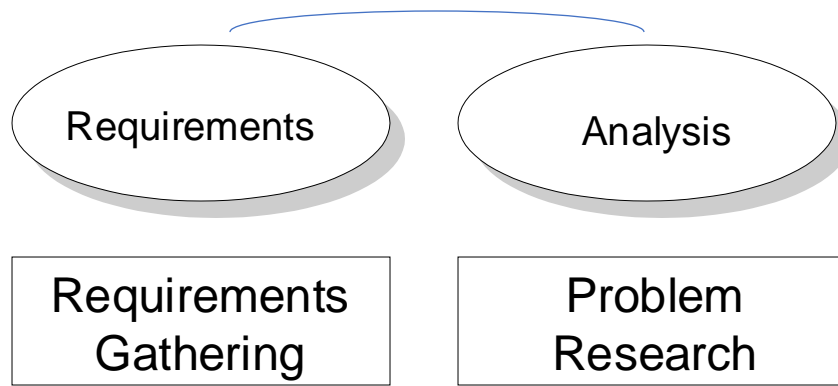
1.6. System Sequence Diagram (SSD)



From requirements to code (2/4)



From requirements to code (2/4)

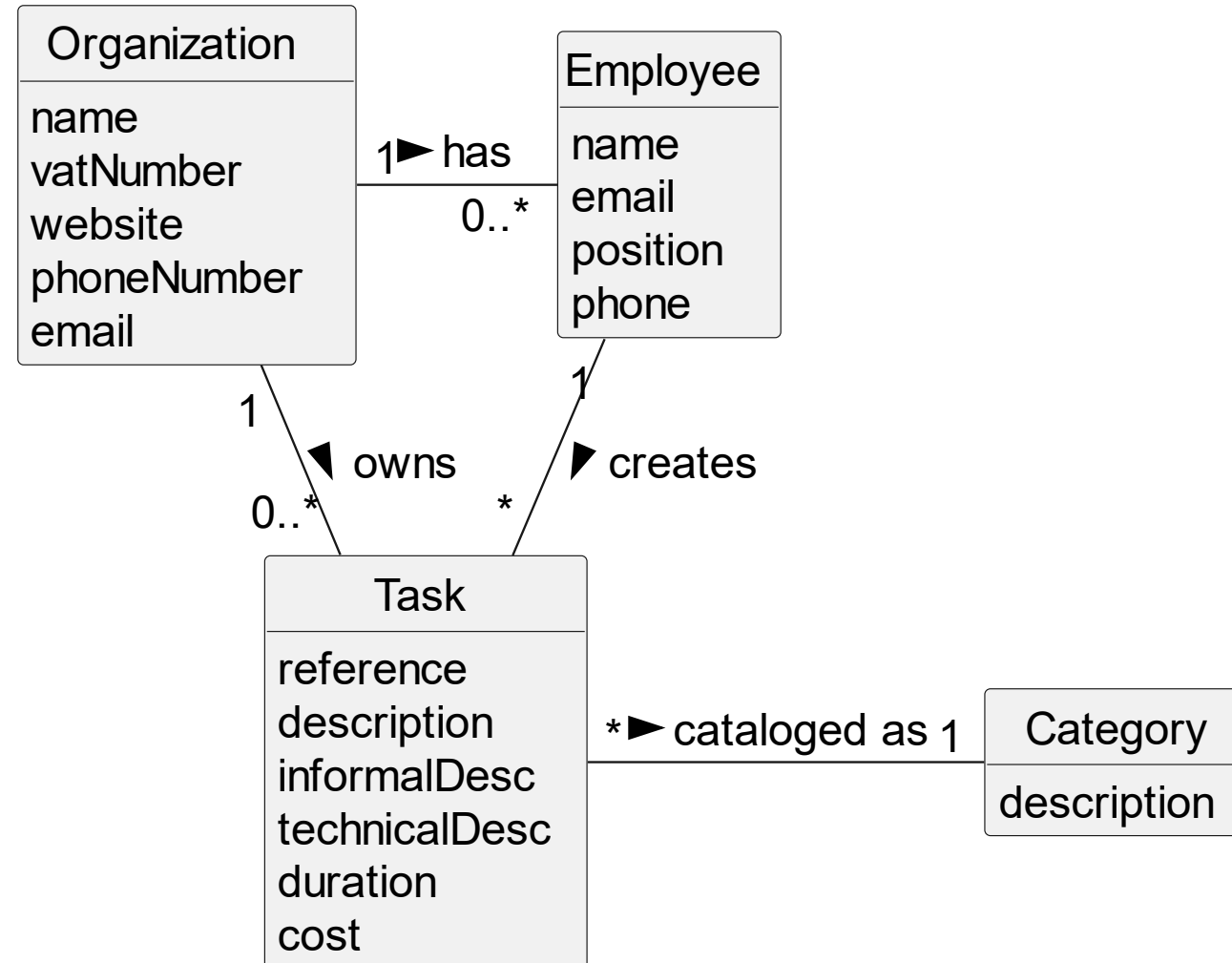


Analysis

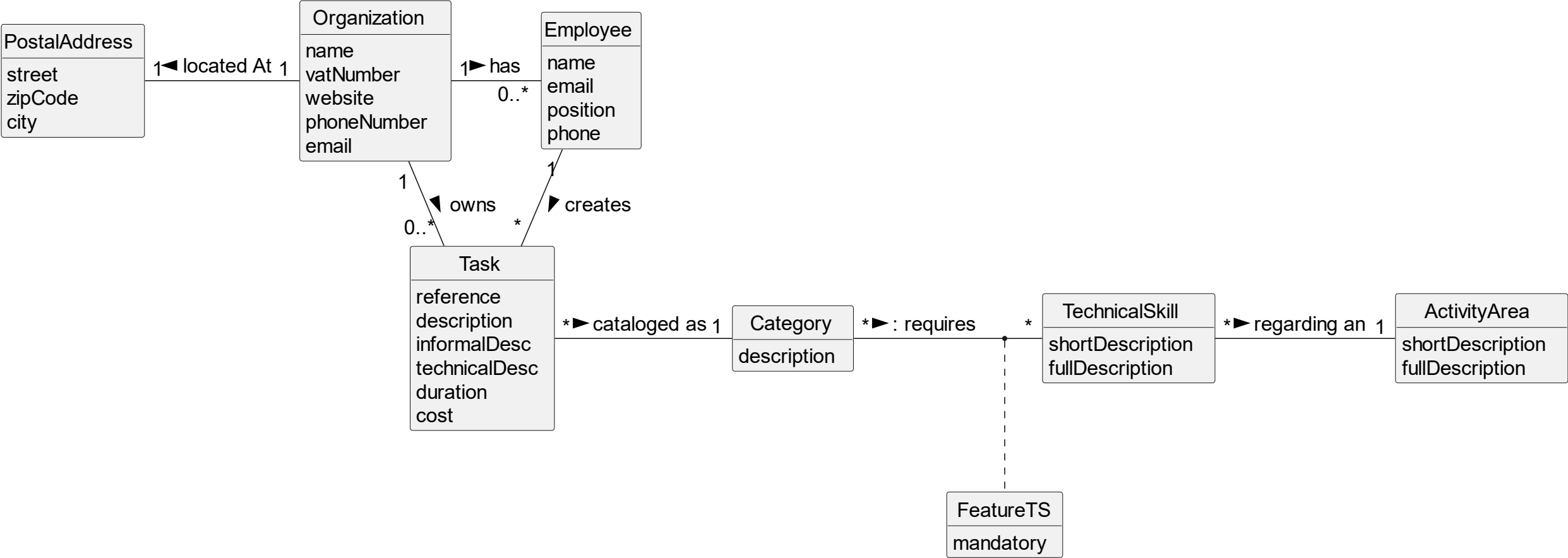
- It implies:
 - Acquiring domain/business knowledge
 - Understanding the domain/business
 - Reasoning about domain/business
- Analysis oriented by:
 - Objects → OO Analysis → Domain (Object) Model
 - Classifying domain elements/concepts
 - Finding relationships between domain concepts
 - Processes or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

Relevant Domain Model Excerpt for US 006

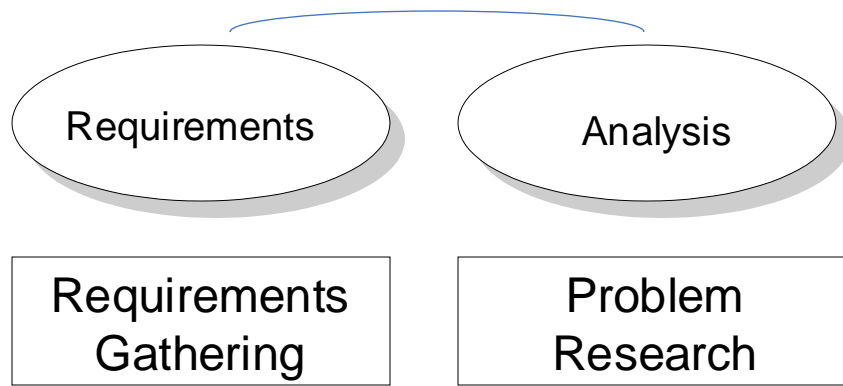
Platform for
Outsourcing Tasks



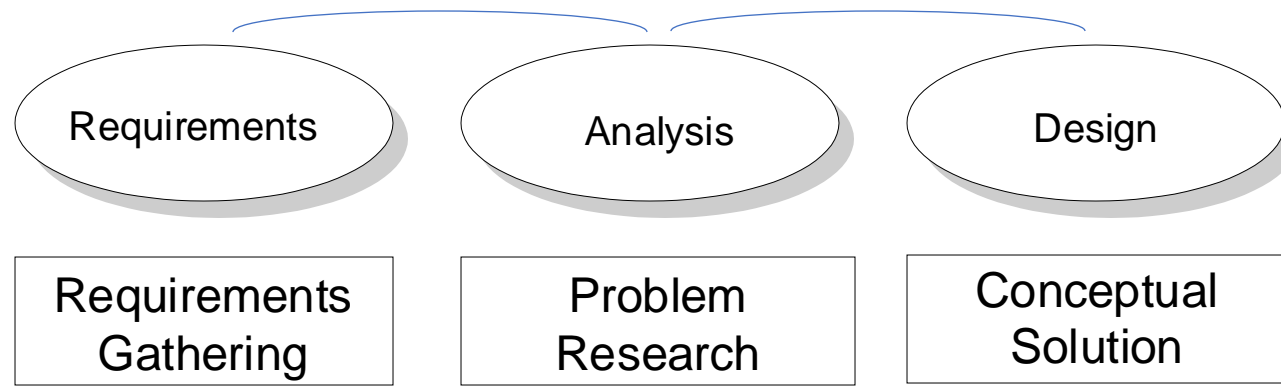
Domain Model (for the Project)



From requirements to code (3/4)



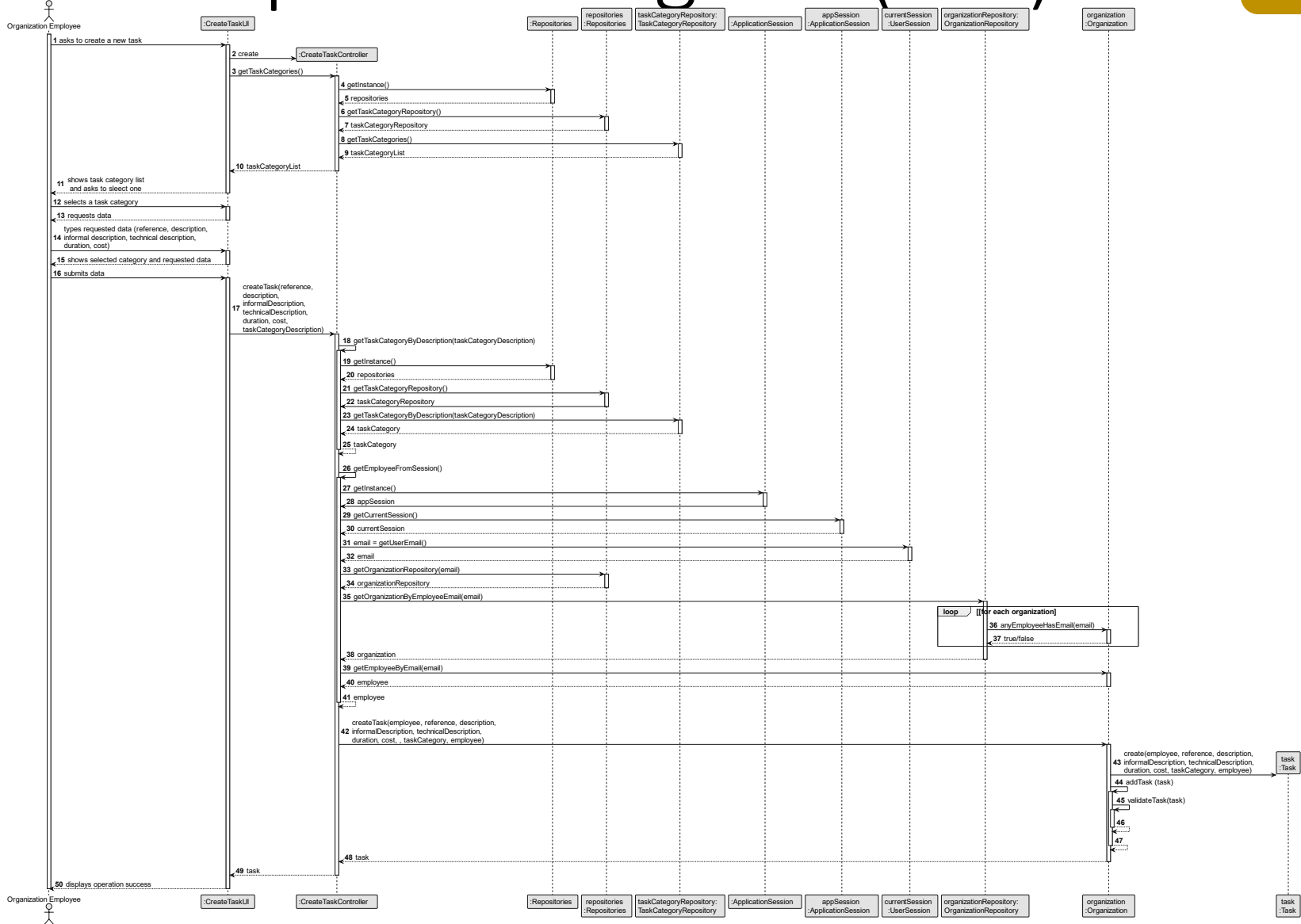
From requirements to code (3/4)



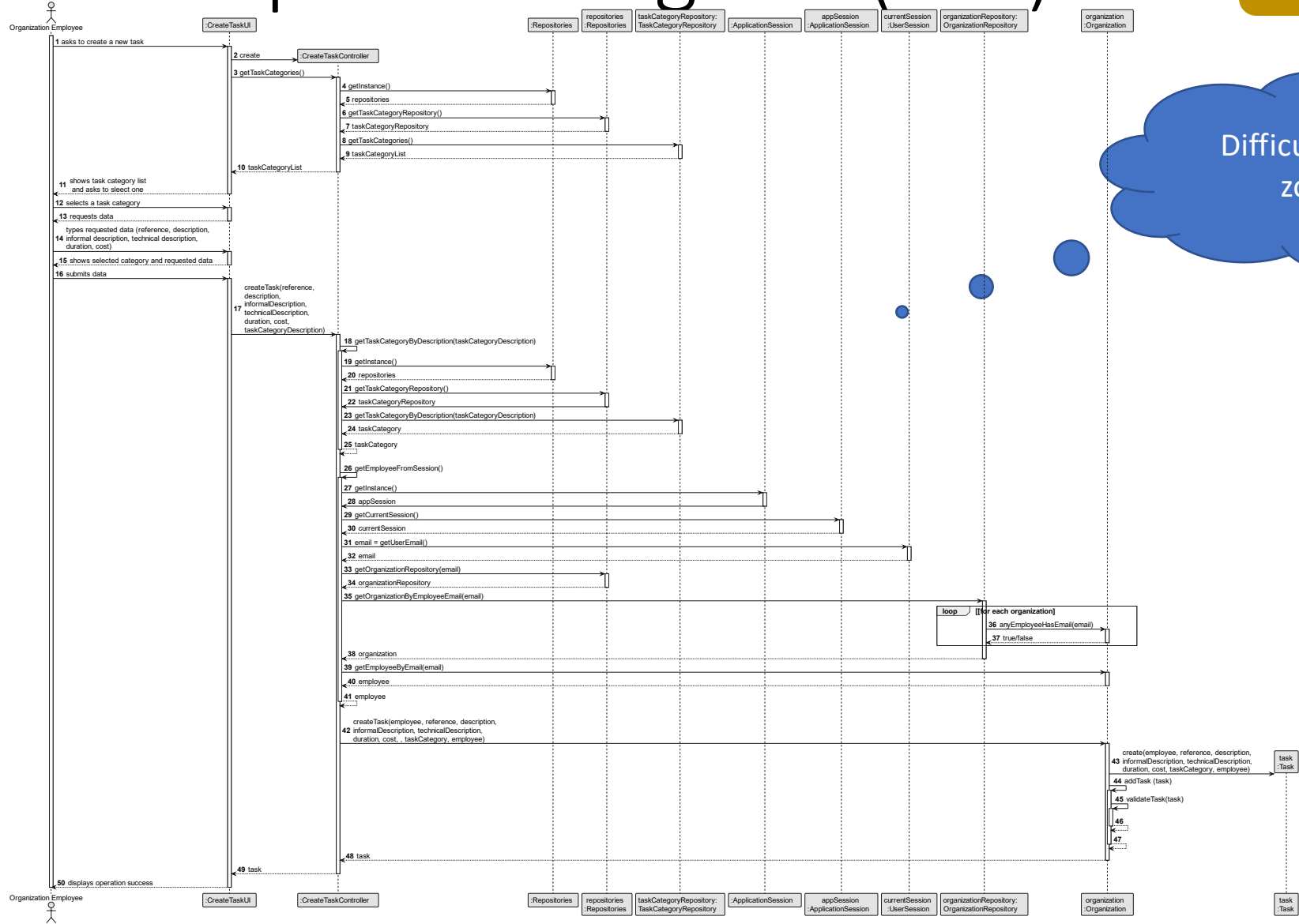
Design – User Story Realization

- 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 designed together (i.e. at the same time)

US 006 – Sequence Diagram (Full)

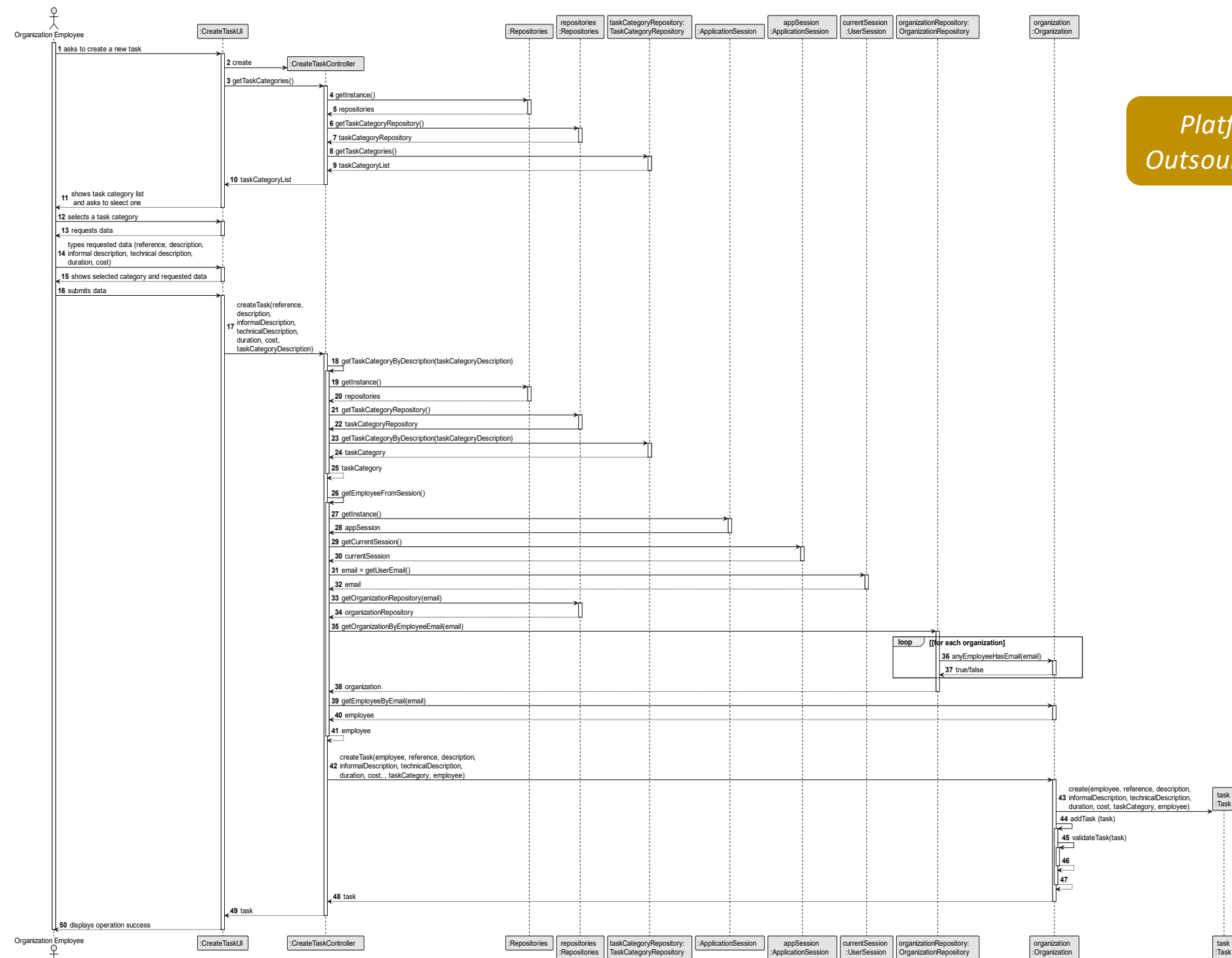


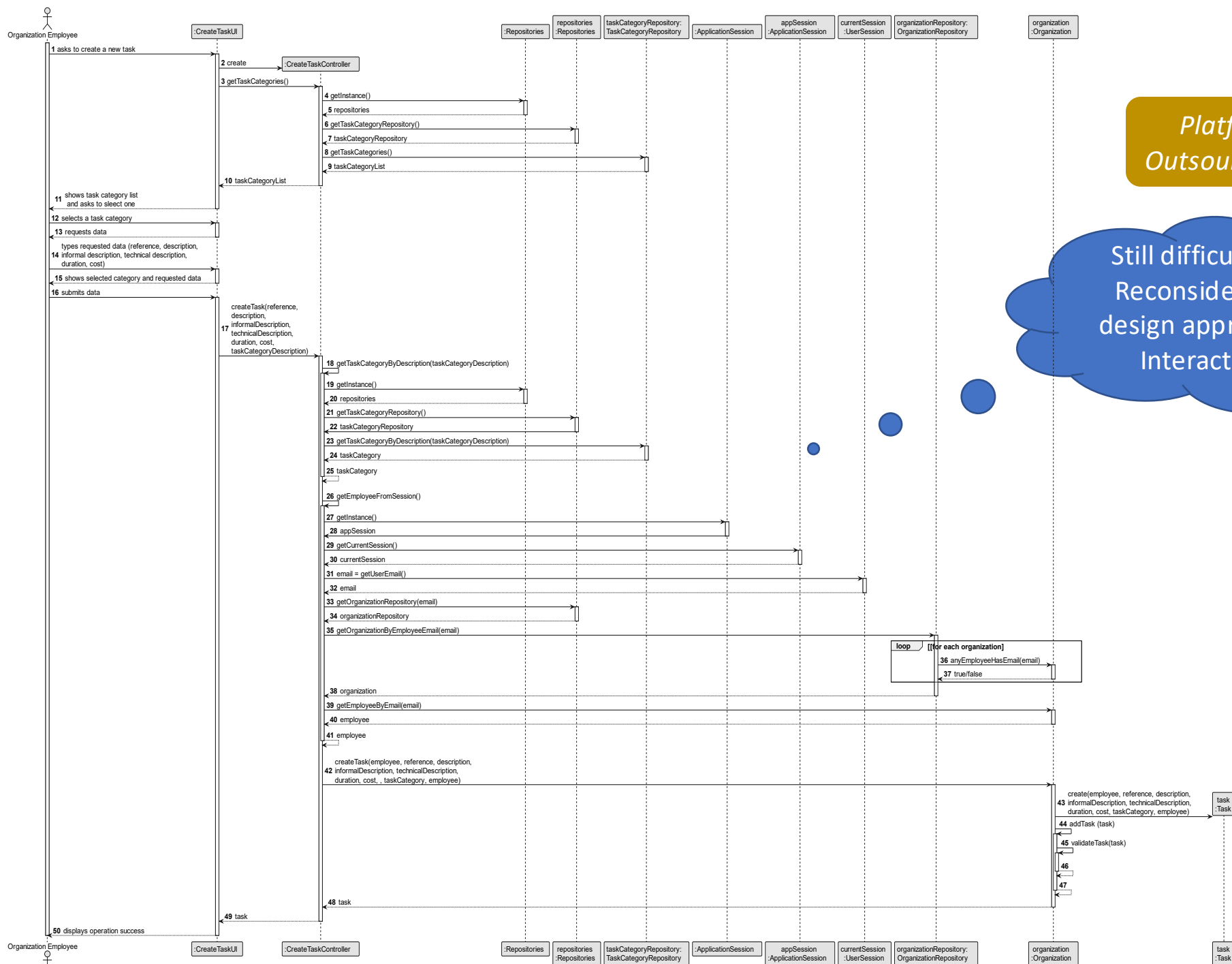
US 006 – Sequence Diagram (Full)



Difficult to read, zoom it

Platform for Outsourcing Tasks

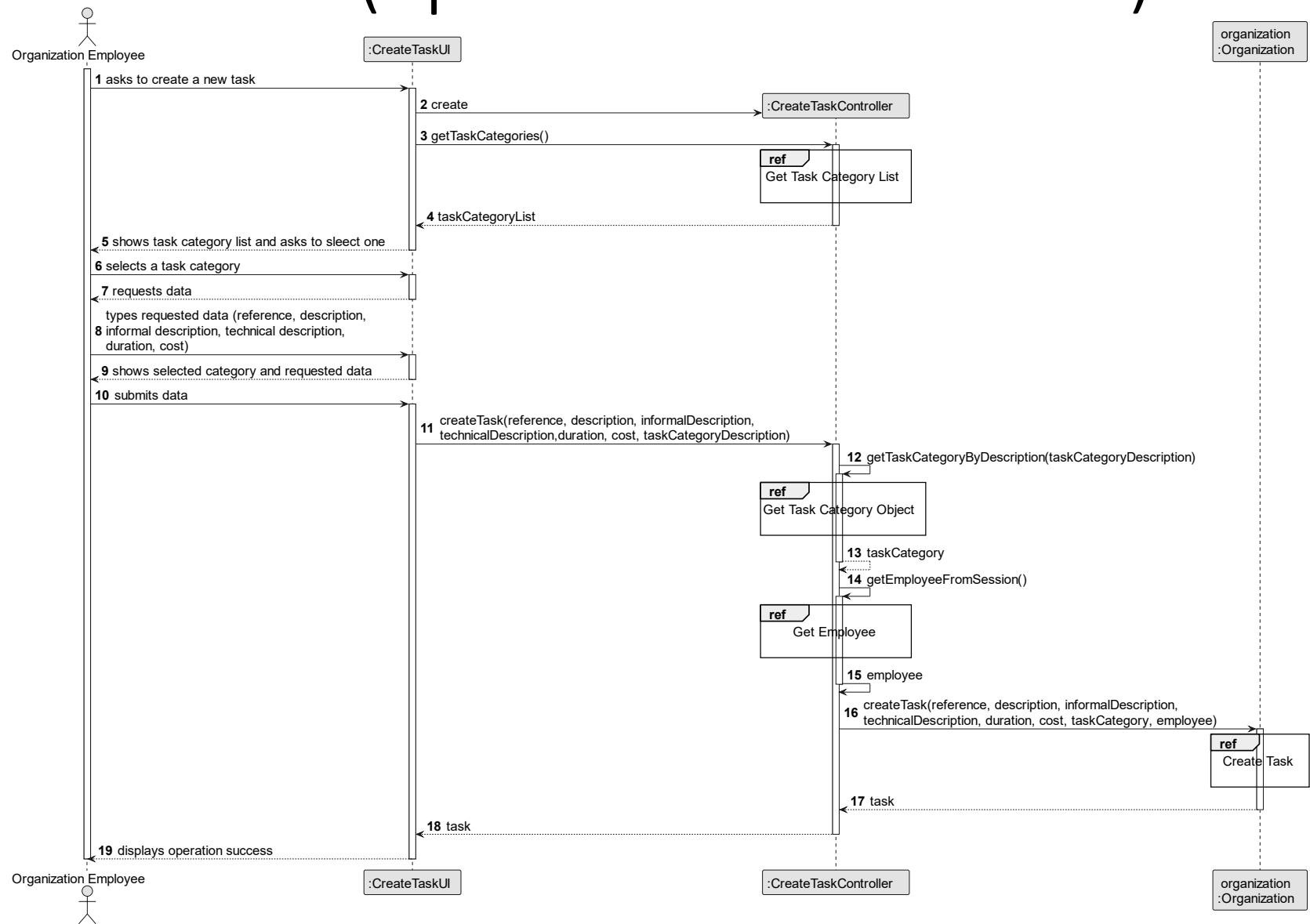


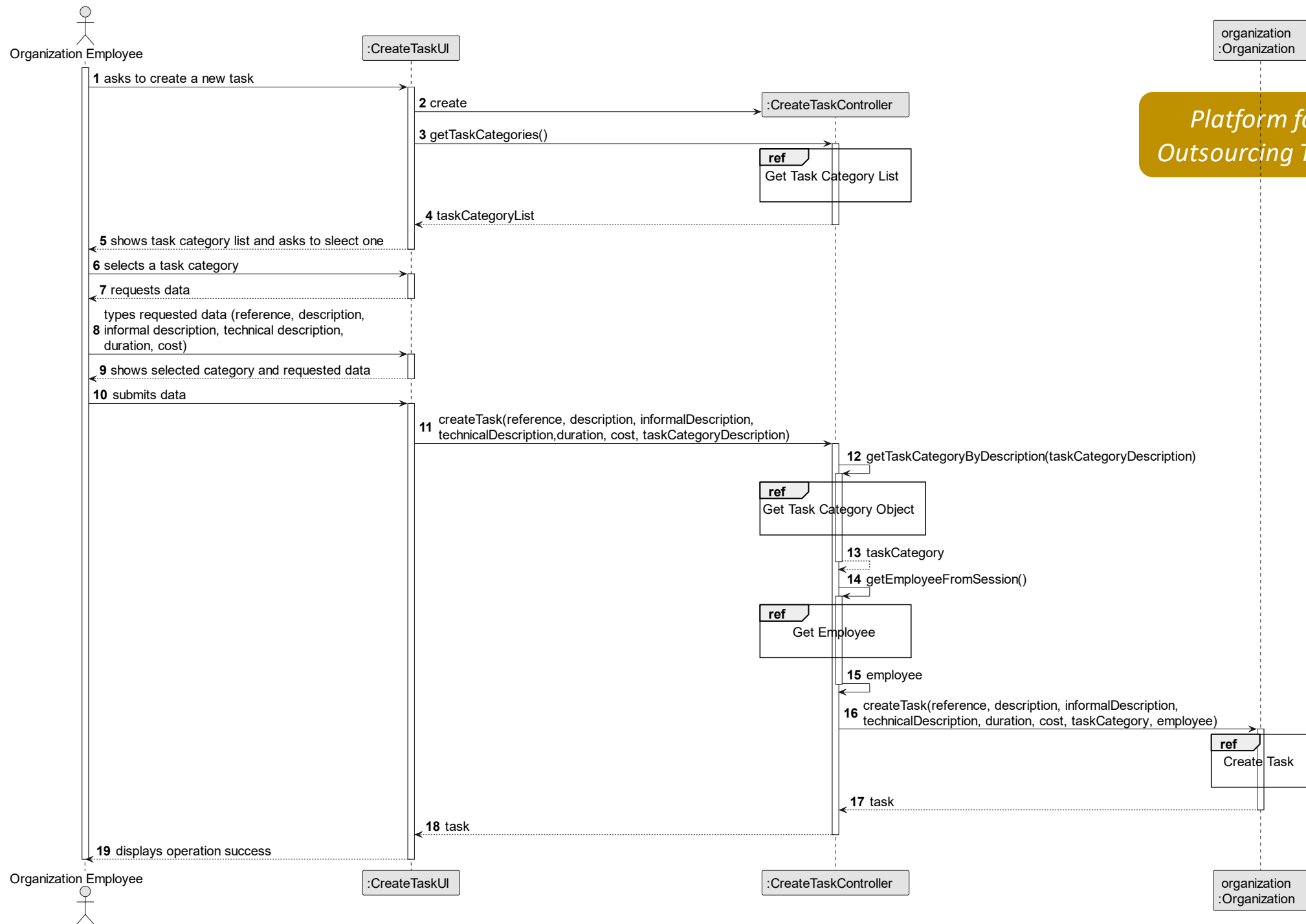


Platform for Outsourcing Tasks

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

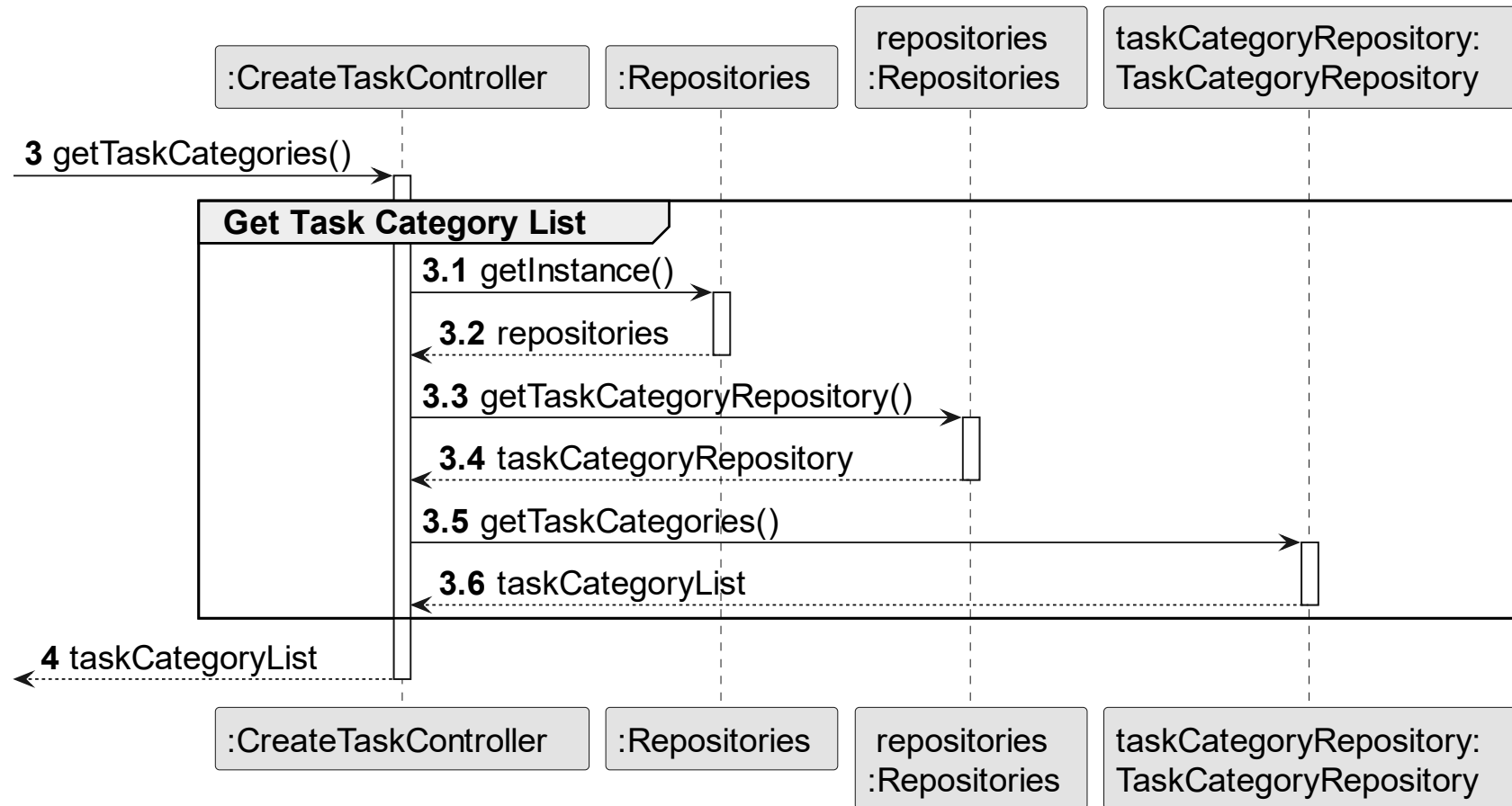
US 006 – SD (Split Interaction Use)





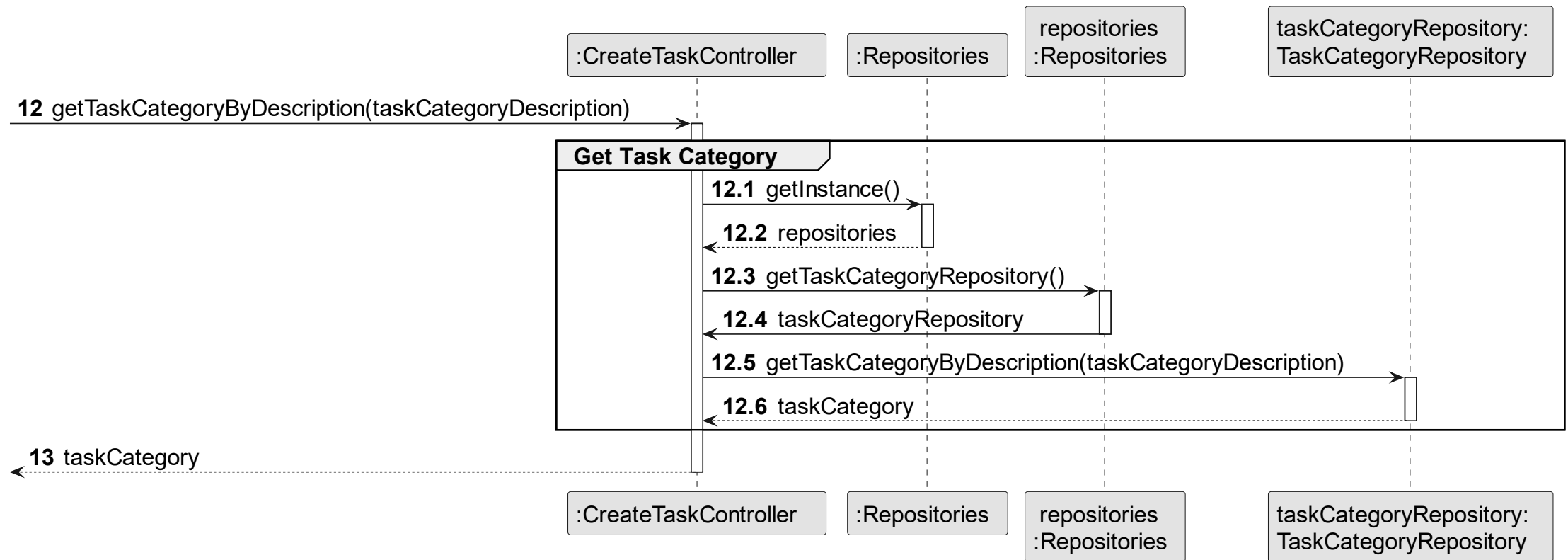
US 006 – SD: ref: Get Task Category List

Platform for
Outsourcing Tasks

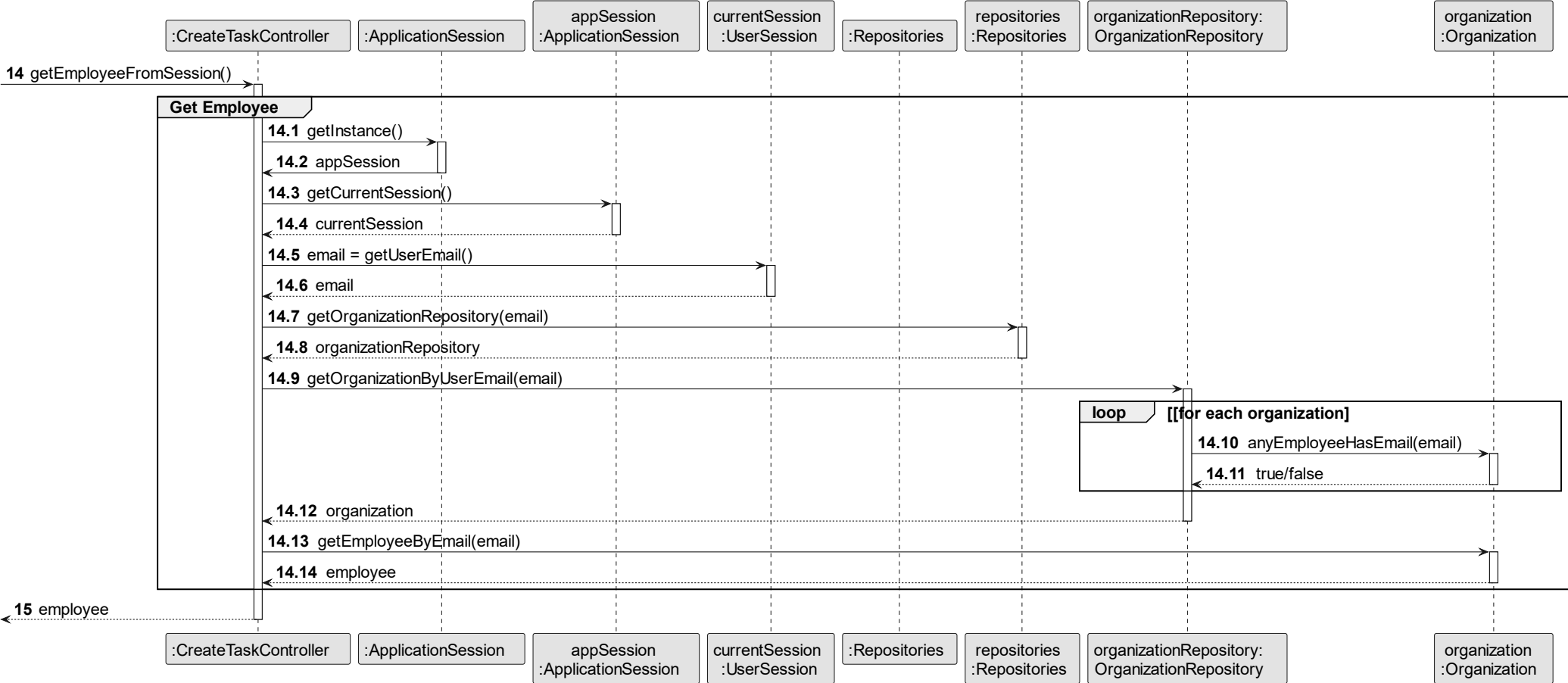


US 006 – SD: ref: Get Task Category

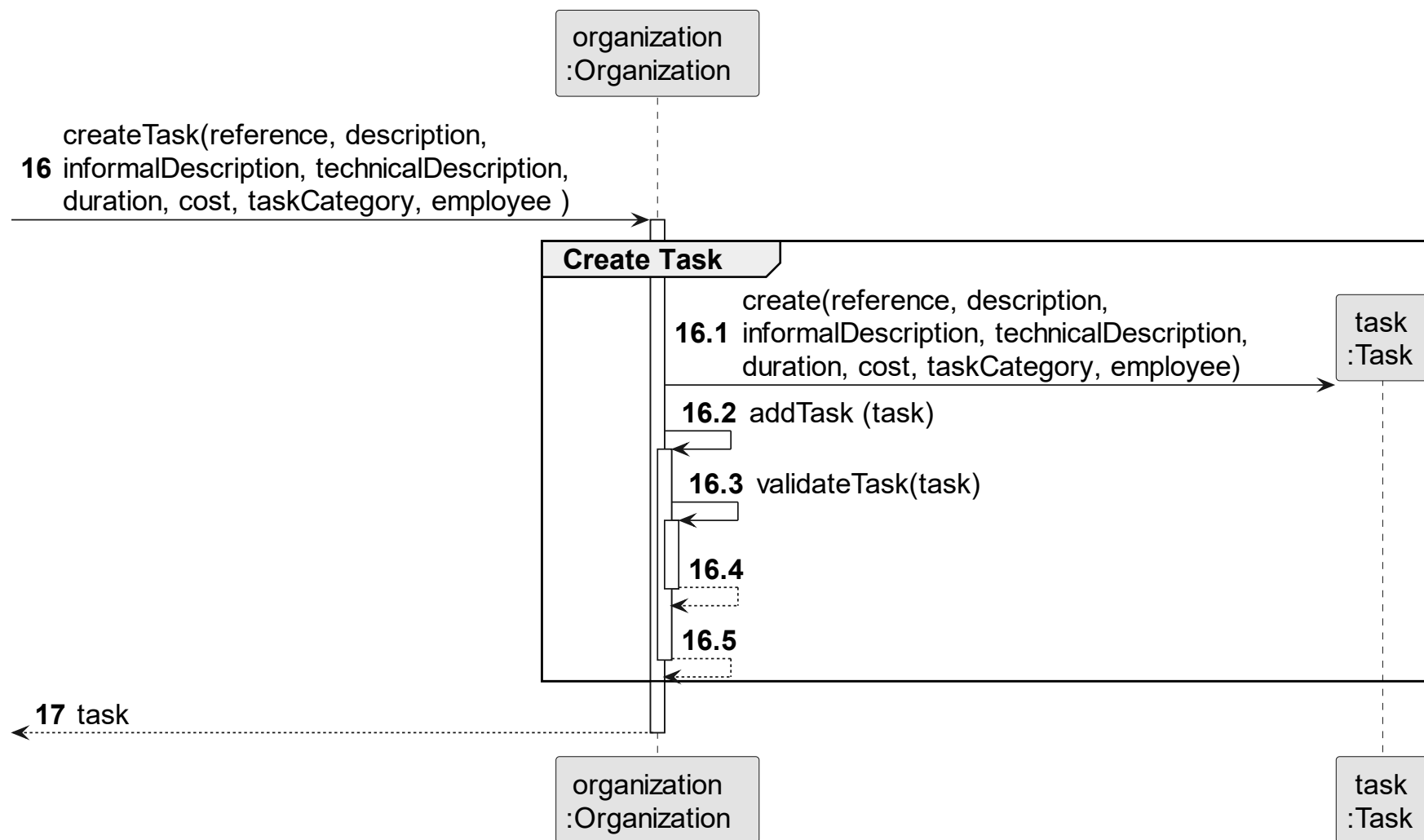
Platform for
Outsourcing Tasks



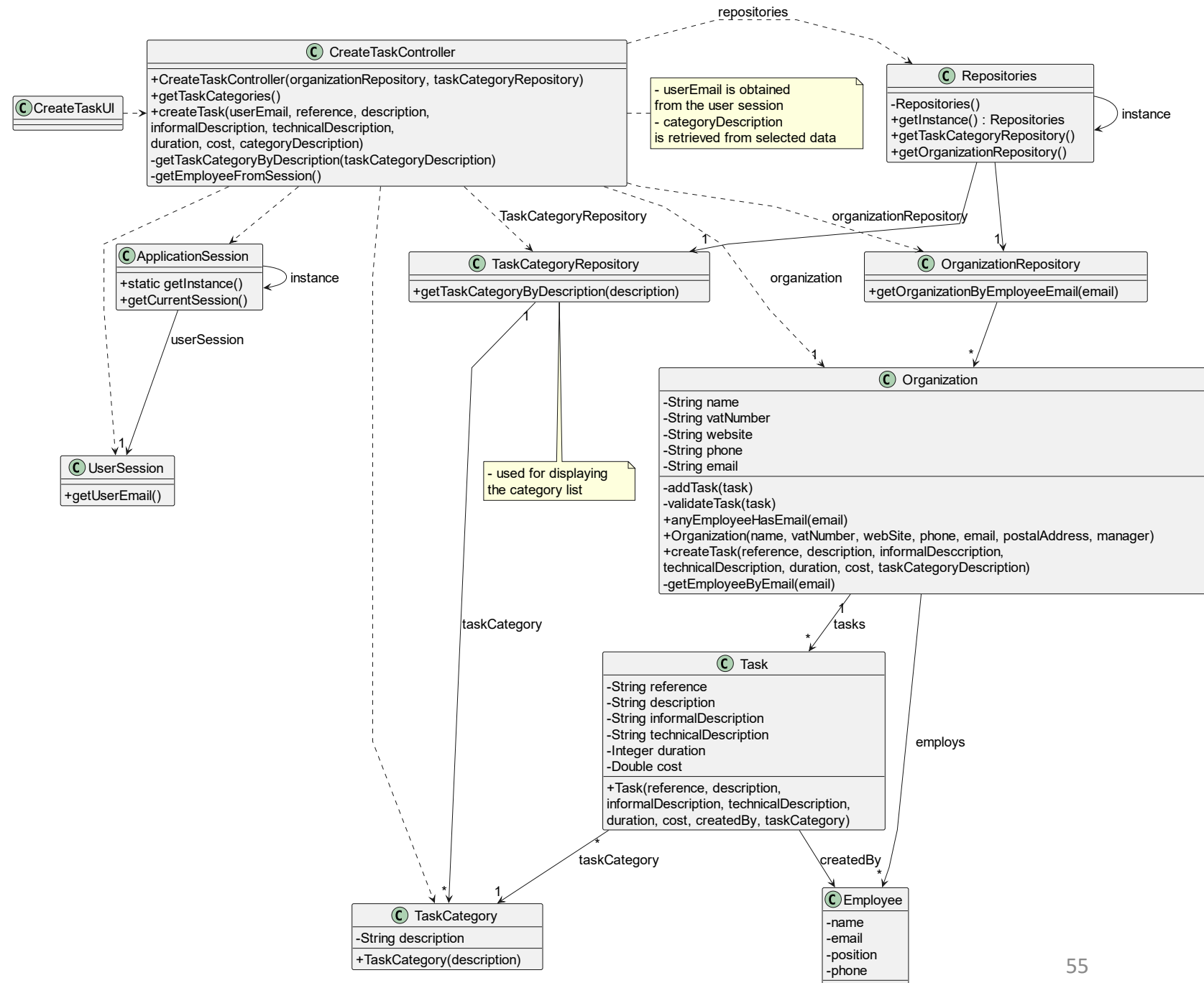
US 006 – SD: ref: Get Employee



US 006 – SD: ref: Create Task

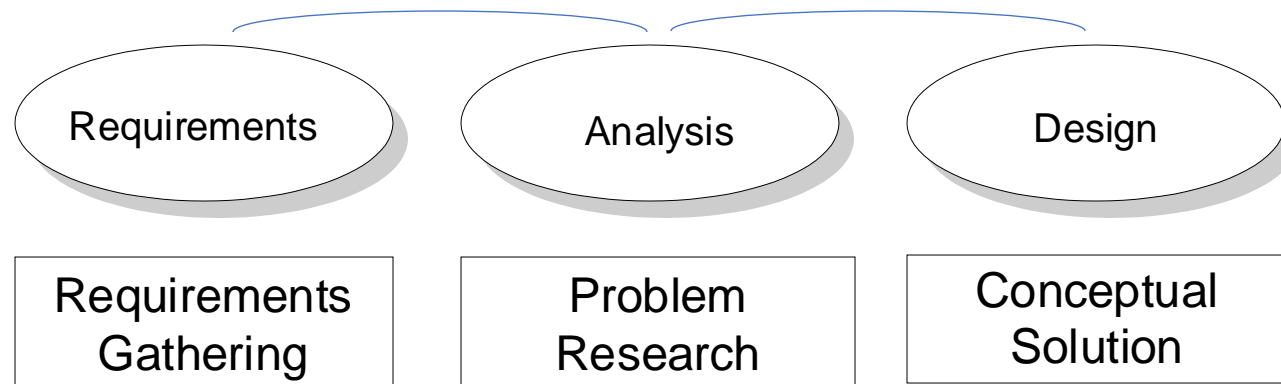


US 006 – Class Diagram

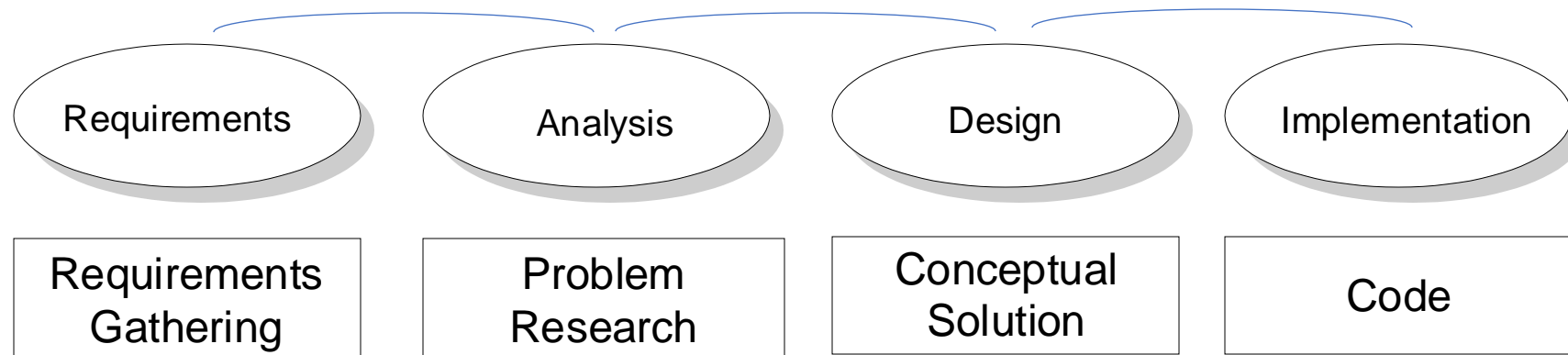


Platform for
Outsourcing Tasks

From requirements to code (4/4)



From requirements to code (4/4)



Coding (excerpt from CreateTaskController)

*Platform for
Outsourcing Tasks*

```
public Task createTask(String reference, String description, String informalDescription,  
                      String technicalDescription, Integer duration, Double cost,  
                      String taskCategoryDescription) {  
  
    TaskCategory taskCategory = getTaskCategoryByDescription(taskCategoryDescription);  
  
    Employee employee = getEmployeeFromSession();  
    Organization organization = getOrganizationRepository().getOrganizationByEmployee(employee);  
  
    if (organization != null) {  
        Task newTask = organization.createTask(reference, description, informalDescription, technicalDescription,  
                                              duration, cost, taskCategory, employee);  
        return newTask;  
    }  
  
    return null;  
}
```


Coding (excerpt from Organization Class)

Platform for
Outsourcing Tasks

```
public 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.  
    Task task = new Task(reference, description, informalDescription, technicalDescription, duration, cost,  
                           taskCategory, employee);  
  
    if (addTask(task)) {  
        return task;  
    }  
  
    return null;  
}
```

Summary

Promoted Working Method (1/2)



Some mentioned **artifacts** have not been introduced yet!

- 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 engineering practices
- Promotes best engineering practices too
- Clear, simple and easy to adopt

References & Bibliography

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