جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila
University of Mohamed Boudiaf - M'sila

1985

Object Oriented Analysis and Design

Chapter 01: Object-Oriented Technology and Software Engineering Methodologies

Prof. Hichem Debbi

hichem.debbi@gmail.com

September 30, 2024

Software Complexity

Brooks suggests, “The complexity of software is an essential property, not an accidental one”.

Why Software Is Inherently Complex

Software Complexity

Brooks suggests, “The complexity of software is an essential property, not an accidental one”.
It derives from four elements :

- Complexity of the problem domain
- Difficulty of managing the development process
- Flexibility possible through software
- Problems of characterizing the behavior of discrete systems

Four Elements of Complexity

1. The Complexity of the Problem Domain

This complexity emerges in the first place from functional requirements in addition to nonfunctional or external requirements.

Four Elements of Complexity

1. The Complexity of the Problem Domain

This complexity emerges in the first place from functional requirements in addition to nonfunctional or external requirements. **Communication Gap:**Basically, external complexity emerges from the fact that: Users generally find it very hard to give precise expression to their needs in a form that developers can understand.

- Users and developers have different perspectives on the nature of the problem and make different assumptions regarding the nature of the solution.
- The problem of gathering requirements and how to interpret them.
- The requirements of a software system often change during its development.

Four Elements of Complexity

1. The Complexity of the Problem Domain

- ***maintenance*:** when we correct errors.
- ***evolution*:** when we respond to changing requirements.
- ***preservation*:** when we continue to use extraordinary means to keep an ancient and decaying piece of software in operation.

Four Elements of Complexity

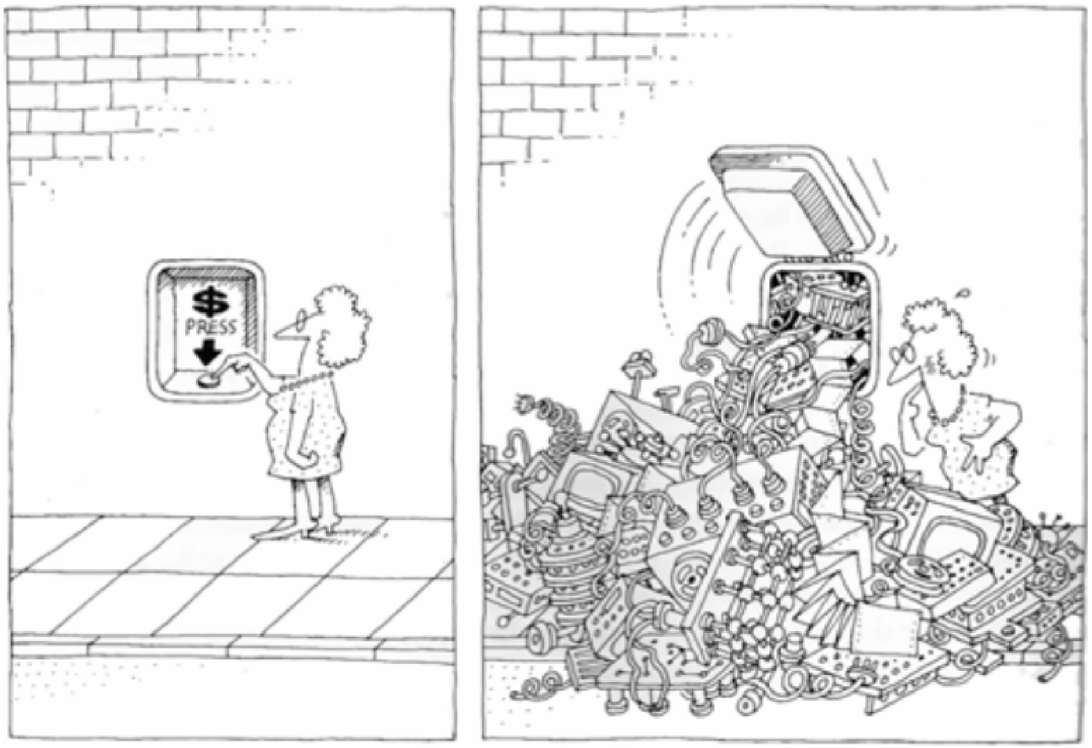


Figure: Illusion of Simplicity

2. The Difficulty of Managing the Development Process

Decomposing the implementation of large software on many work teams of developers is good choice. However, we should guarantee the best communication and collaboration between the developers in the same group, and between the groups.

3. The Flexibility Possible through Software

Software offers the ultimate flexibility, so it is possible for a developer to express almost any kind of abstraction. This flexibility turns out to be a useful property, because it forces the developer to create all the primitive building blocks on which these higher-level abstractions stand.

4. The Problem of Characterizing the Behavior of Discrete Systems

The state of large discrete system is represented by a collection thousands of variables and their current values, and the current address and calling stack of each process within the system . The system is subjected to different events that could lead it from a state to another.
Some events could lead to unexpected fatals. Forgetting to take in account some interactions between events could lead to fatals as well.

Decomposition

Decomposition

Generally, decomposition is a technique that we think intuitively about using it to cope with such a complex task. When designing a complex software system, it is essential to decompose it into smaller and smaller parts, each of which we may then refine independently.

Algorithmic Decomposition

In Algorithmic decomposition, each module in the system denotes a major step in some overall process, and the relationships among various functional elements of the solution must be clarified.

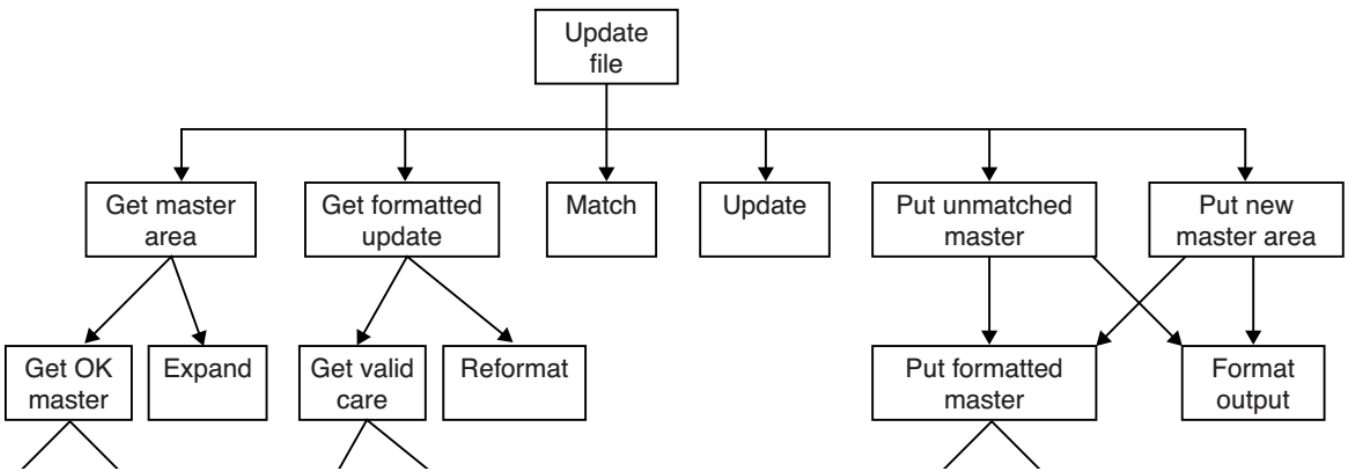


Figure: Algorithmic Decomposition

Object-Oriented Decomposition

The system is decomposed according to the key abstractions in the problem domain. Rather than decomposing the problem into steps such as Get formatted update and Add checksum, we have identified objects such as Master File and Checksum.

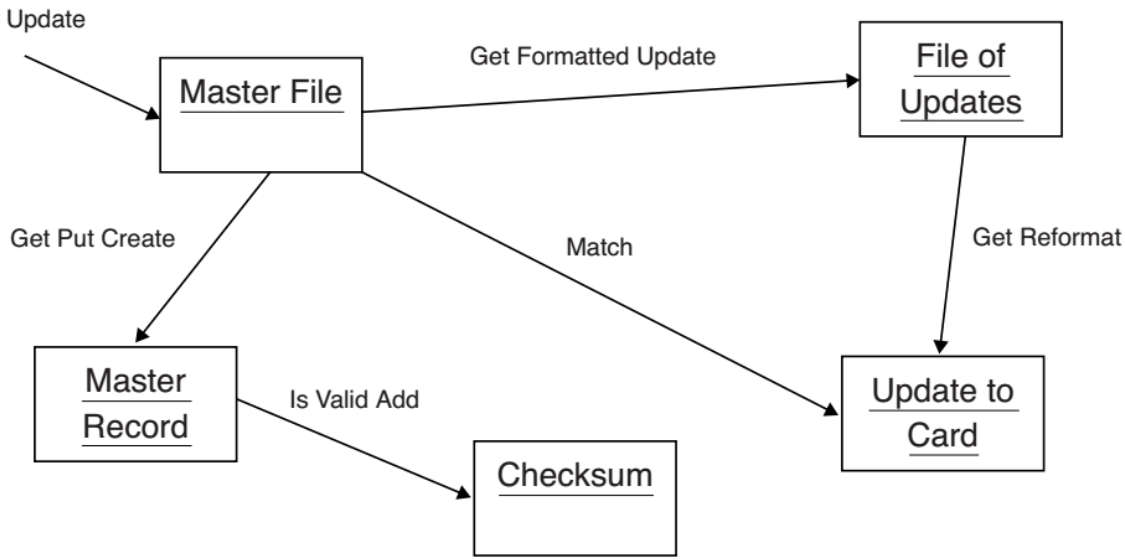


Figure: Object-Oriented Decomposition

Object-Oriented Technology

Object-oriented technology is built on a sound engineering foundation, whose elements we collectively call the object model of development or simply the object model. The object model encompasses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence. By themselves, none of these principles are new. What is important about the object model is that these elements are brought together in a synergistic way.

Object-Oriented Programming

Definition

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

- Object-oriented programming uses objects, not algorithms.
- Each object is an instance of some class.
- Classes may be related to one another via inheritance relationships.

Objects in the Business Word

Figure: Objects in the business word

Classification

Definition

Is about grouping objects that we observe in the domain into classes. The problem with classification is that we can defined such classes from different points of view, each one could have different perspectives. Th goal is to choose the best one, of course withe respect to the problem domain.

Classification

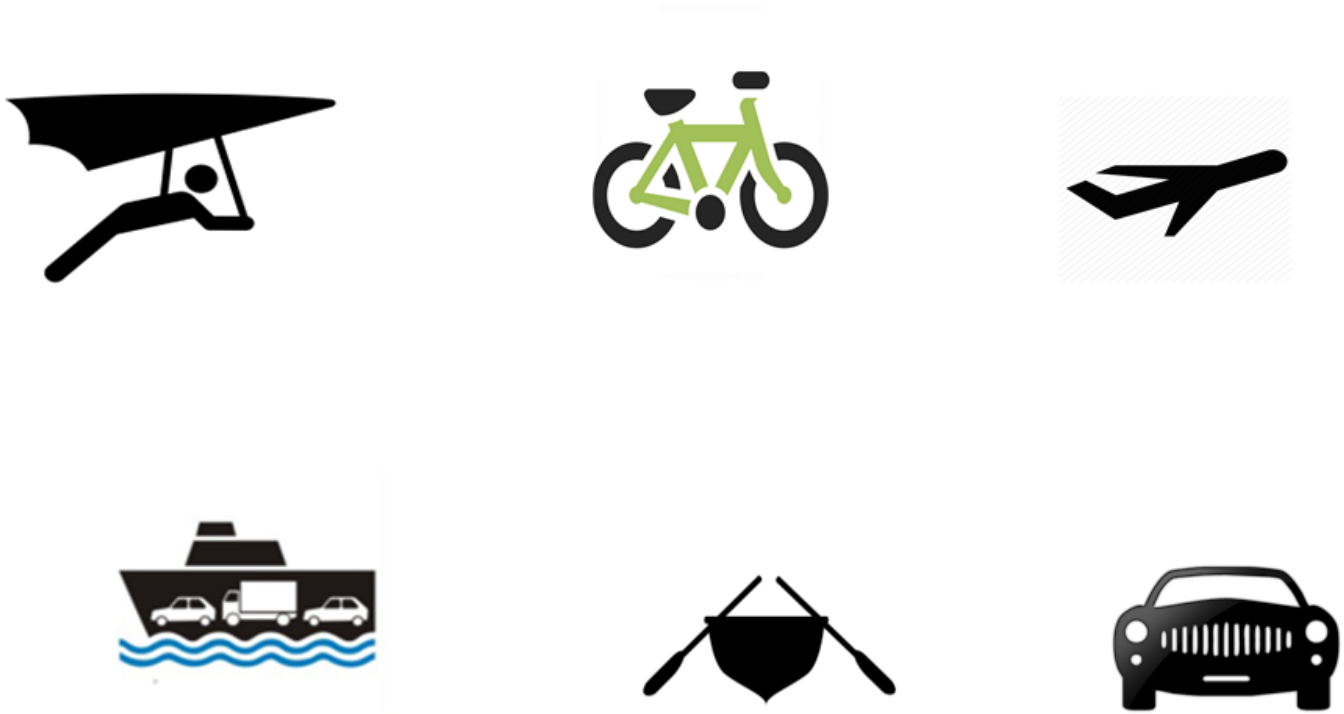


Figure: Vehicles

Classification

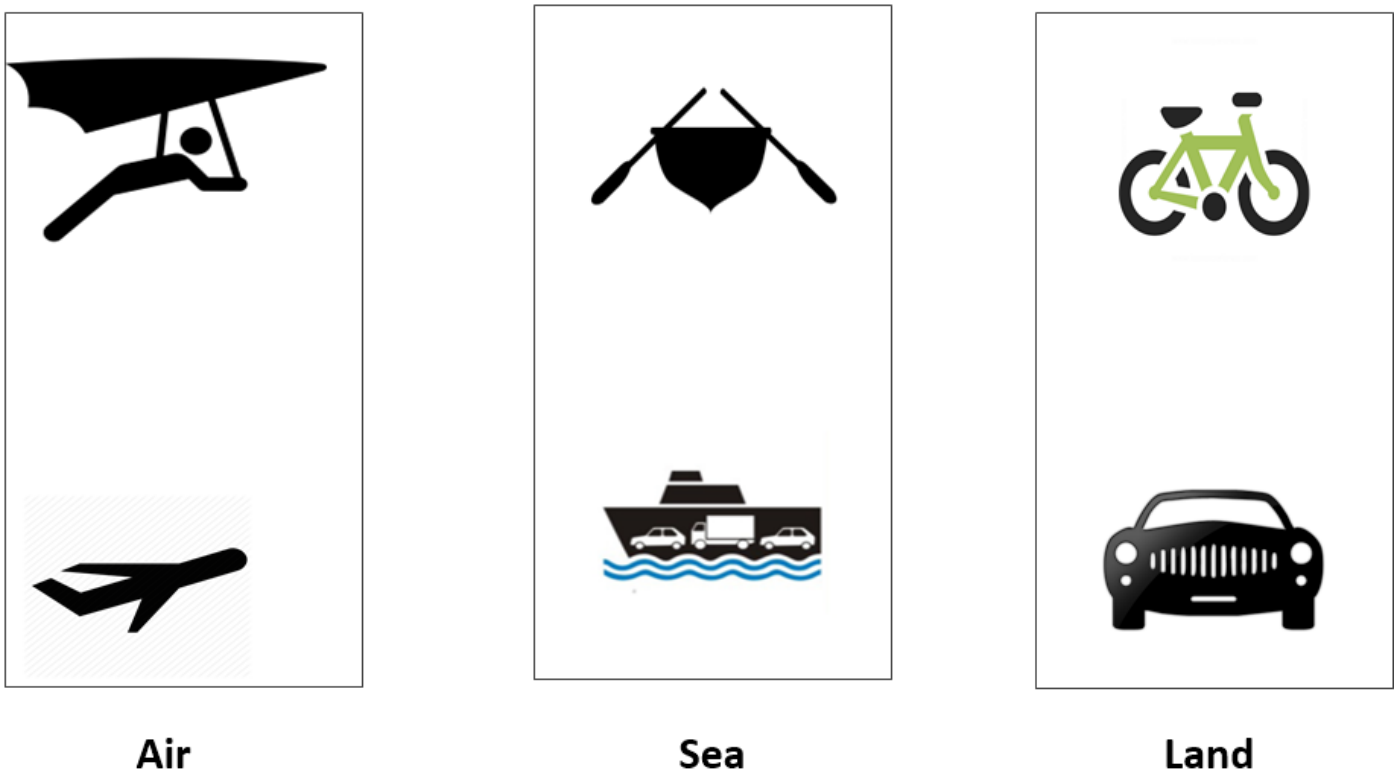
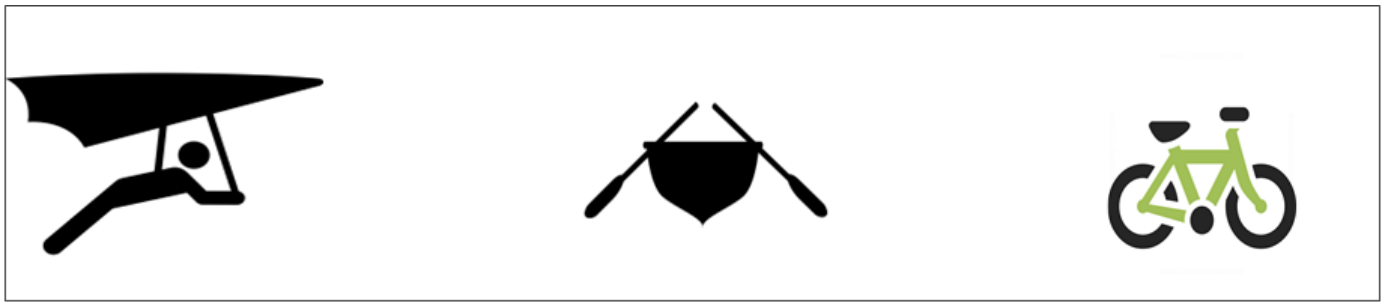


Figure: Vehicles

Classification



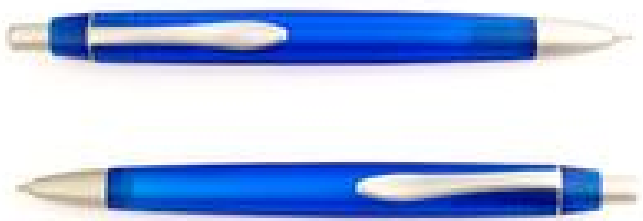
Unpowerd

Powerd

Figure: Vehicles

Objects, Identical or Equal ?

If we have two blue pens that share exactly the same characteristics such as : blue ink, half full, same manufacturer, same model,... Although the pens have the same attributes and could do the same thing, we say that the pens are **equal** but not **identical**.



Object representation

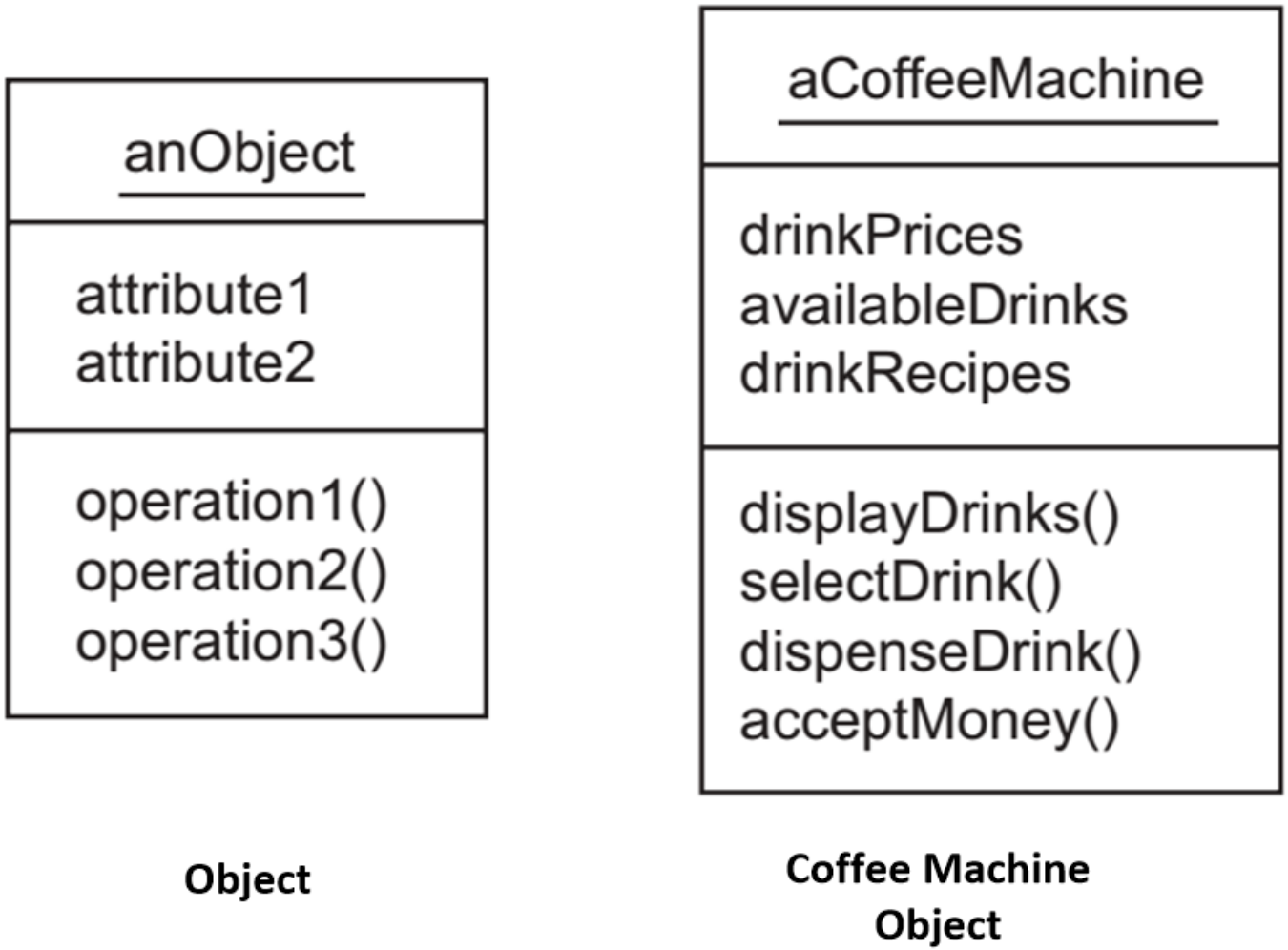


Figure: Vehicles

Object-Oriented Design

Definition

Object-oriented design is a method of design that consists of the process of object oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design.

Difference between OOP and OOD

The emphasis in programming methods is primarily on the proper and effective use of particular language mechanisms. By contrast, design methods emphasize the proper and effective structuring of a complex system

Object-Oriented Analysis

Definition

Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.

The relationship between OOP, OOD, and OOA

The products of object-oriented analysis serve as the models from which we may start an object-oriented design; the products of object-oriented design can then be used as blueprints for completely implementing a system using object-oriented programming methods.

Method vs Methodology

Method vs Methodology

Method

A method is a disciplined procedure for generating a set of models that describe various aspects of a software system under development, using some well-defined notation.

Method vs Methodology

Method

A method is a disciplined procedure for generating a set of models that describe various aspects of a software system under development, using some well-defined notation.

Methodology

A methodology is a collection of methods applied across the software development lifecycle and unified by process, practices, and some general, philosophical approach.

Software Development Methodology

Software development methodology or software development process is a repeatable process that we can follow from the earliest stages of software development through to the maintenance of an installed system. A methodology should specify what we're expected to produce as we follow the process. A methodology will also include advice or techniques for resource management, planning, scheduling and other management tasks.

Requirements

Requirements capture is about discovering what we're going to achieve with our new piece of software and has two aspects.

Requirements

Requirements capture is about discovering what we're going to achieve with our new piece of software and has two aspects.

- **Business modeling** involves understanding the context in which our software will operate – if we don't understand the context, we have little chance of producing something to enhance that context.
- **System requirements modeling** means deciding what capabilities the new software will have and writing down those capabilities. We need to be clear about what our software will do and what it won't do.

Analysis

Analysis means understanding what we're dealing with. Before we can design a solution, we need to be clear about the relevant entities, their properties and their inter-relationships. We also need to be able to verify our understanding.

Design

In the design phase, we work out how to solve the problem. In other words, we make decisions, based on experience, estimation and intuition, about what software we will write and how we will deploy it.

Design

In the design phase, we work out how to solve the problem. In other words, we make decisions, based on experience, estimation and intuition, about what software we will write and how we will deploy it.

- **System design** Breaks the system down into logical subsystems (processes) and physical subsystems (computers and networks), decides how machines will communicate, chooses the right technologies for the job.
- **subsystem design** we decide how to cut each logical subsystem into effective, efficient and feasible code.

Specification

Specification usually is not seen as a standalone phase, but rather as a description of the system's design resulted from analysis and design phase through transforming business and system requirements. We could refer to specifications as "contracts". Specifications could be formal or informal.

Implementation

Writing pieces of code that work together to form subsystems, which in turn collaborate to form the whole system. By delivering a good design of the system, programmers are free to decide which programming techniques they use, in condition that they respect the specifications resulted from the previous phases.

Testing

When our software is complete, it must be tested against the system requirements to see if it fits the original goals. It is a good idea for programmers to perform small tests as they go along, to improve the quality of the code that they deliver. Generally speaking, however, major tests should not be designed, implemented or carried out by the developers who wrote the software.

Deployment

In the deployment phase, we're concerned with getting the hardware and software to the end users, along with manuals and training materials. This may be a complex process, involving a gradual, planned transition from the old way of working to the new.

Maintenance

As software developers, we're normally interested in maintenance because of the ***faults (bugs)*** that are found in our software. We must find the faults and remove them as quickly as possible, rolling out fixed versions of the software to keep the end users happy. As well as faults, our users may discover ***deficiencies*** (things that the system should do but doesn't) and ***extra requirements*** (things that would improve the system). From the business point of view, we would hope to fix and improve our software over time to maintain competitive advantage

Waterfall Methodology

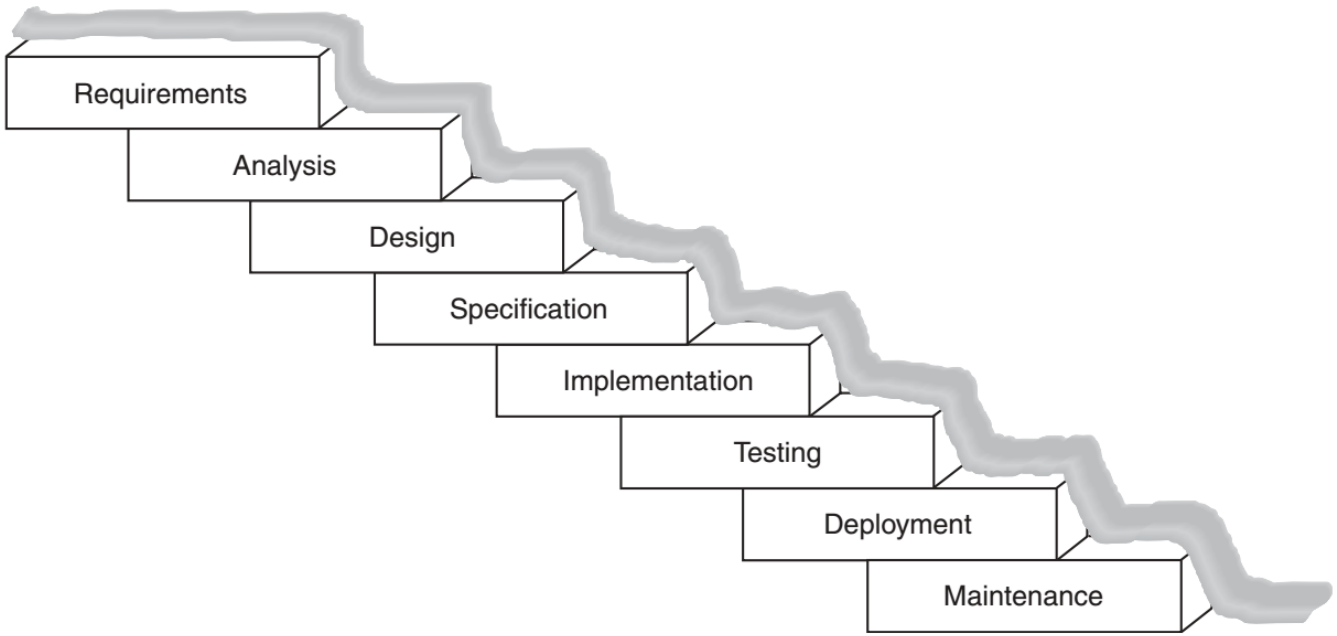


Figure: Waterfall Methodology

Spiral Methodology

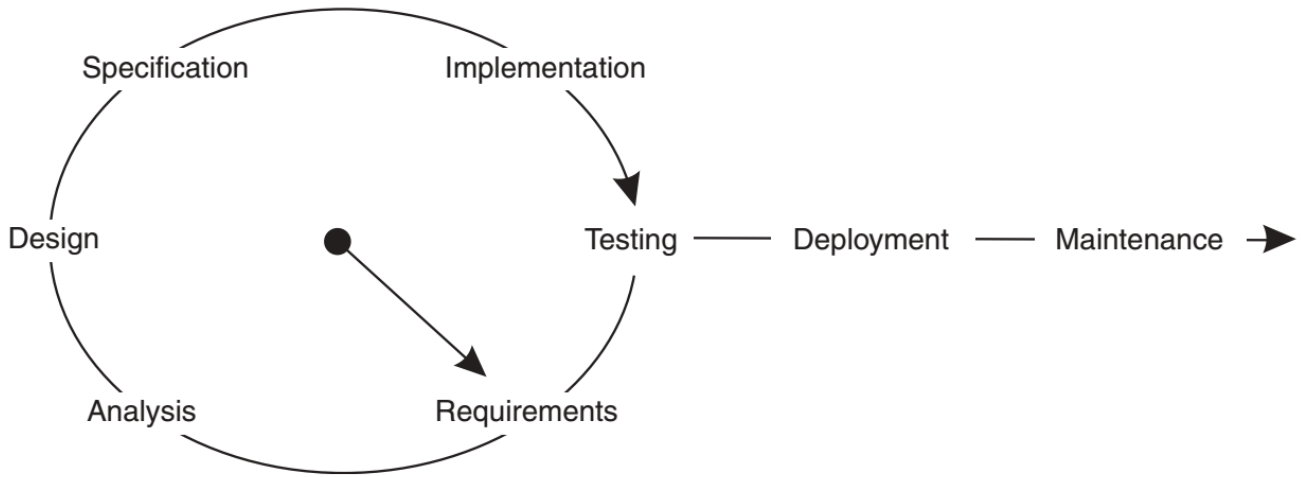
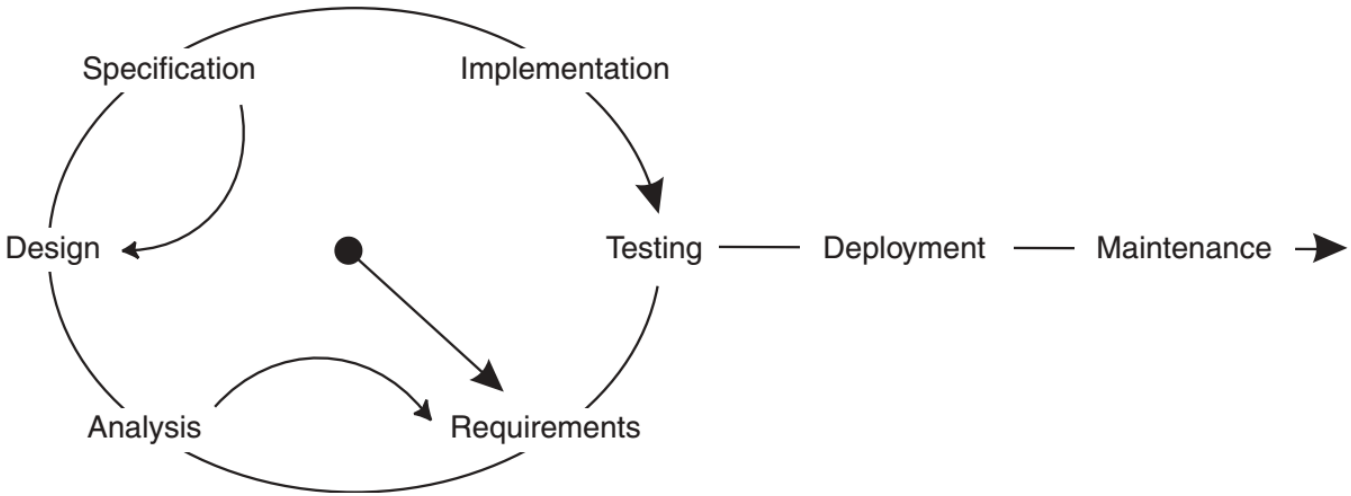


Figure: Spiral Methodology

Iterative Methodology

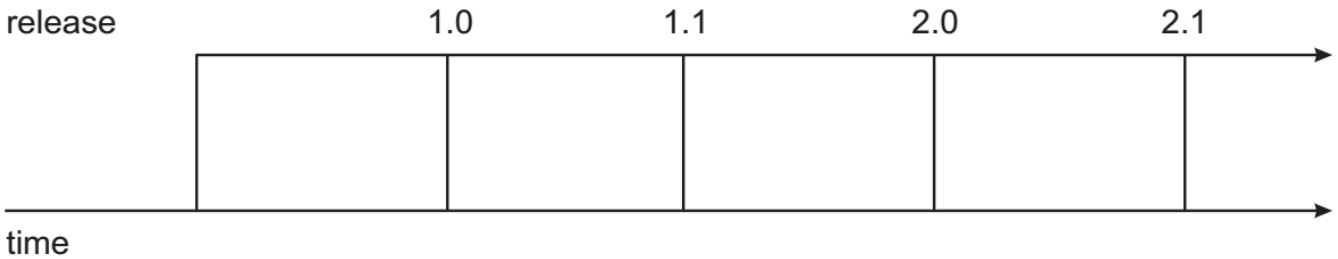


Iterative Methodology

Characteristics

- In contrast to Spiral, it allows us to iterate over the phases, moving backwards and forwards, or round and round.
- Classical phases remind us what we should be doing at each stage
- The artifacts (diagrams, descriptions, code, etc.) that we produce are gradually improved as we move towards deployment

Incremental Methodology

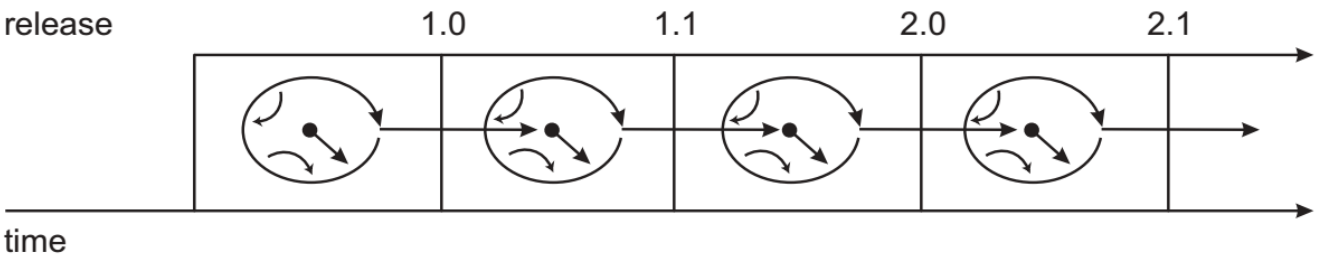


Incremental Methodology

Characteristics

- The delivery of software is incremental in practice despite our prior need for that.
- Making versions of our software is in the heart of incremental methodology

Combining spiral, iterative and incremental development



Macro Process and Micro Process

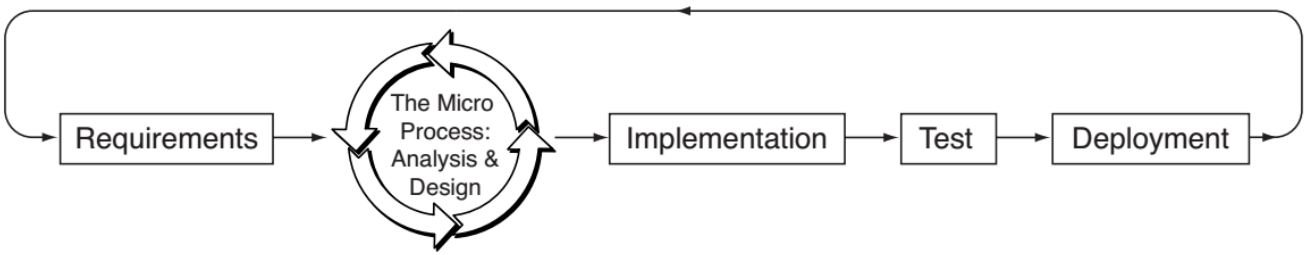


Figure: The Micro Process within the Macro Process

Micro Process

- The micro process takes the requirements provided by the macro process and produces design specifications that are implemented, tested, and deployed in the macro process.

Micro Process

- The micro process takes the requirements provided by the macro process and produces design specifications that are implemented, tested, and deployed in the macro process.
- Analysis takes the system requirements and produces an initial solution, and design takes the results of analysis and produces a specification that can be efficiently implemented.

Micro Process

- The micro process takes the requirements provided by the macro process and produces design specifications that are implemented, tested, and deployed in the macro process.
- Analysis takes the system requirements and produces an initial solution, and design takes the results of analysis and produces a specification that can be efficiently implemented.
- The analysis is considered complete when it accurately represents the system requirements, is consistent, and can serve as a good basis for design

Micro Process

- The micro process takes the requirements provided by the macro process and produces design specifications that are implemented, tested, and deployed in the macro process.
- Analysis takes the system requirements and produces an initial solution, and design takes the results of analysis and produces a specification that can be efficiently implemented.
- The analysis is considered complete when it accurately represents the system requirements, is consistent, and can serve as a good basis for design
- The design is considered complete when it is detailed enough to be implemented and tested

Micro Process

- Analysis aims to analyze the system's behavior through studying the problem domain

Micro Process

- Analysis aims to analyze the system's behavior through studying the problem domain
- Careful analysis leads to a robust and understandable design

Micro Process

- Analysis aims to analyze the system's behavior through studying the problem domain
- Careful analysis leads to a robust and understandable design
- it is neither possible nor desirable to carry out a complete analysis before allowing design to commence

Micro Process

- Analysis aims to analyze the system's behavior through studying the problem domain
- Careful analysis leads to a robust and understandable design
- it is neither possible nor desirable to carry out a complete analysis before allowing design to commence
- Analysis and design are performed at multiple levels of abstraction throughout the development lifecycle.

Micro Process - Activities

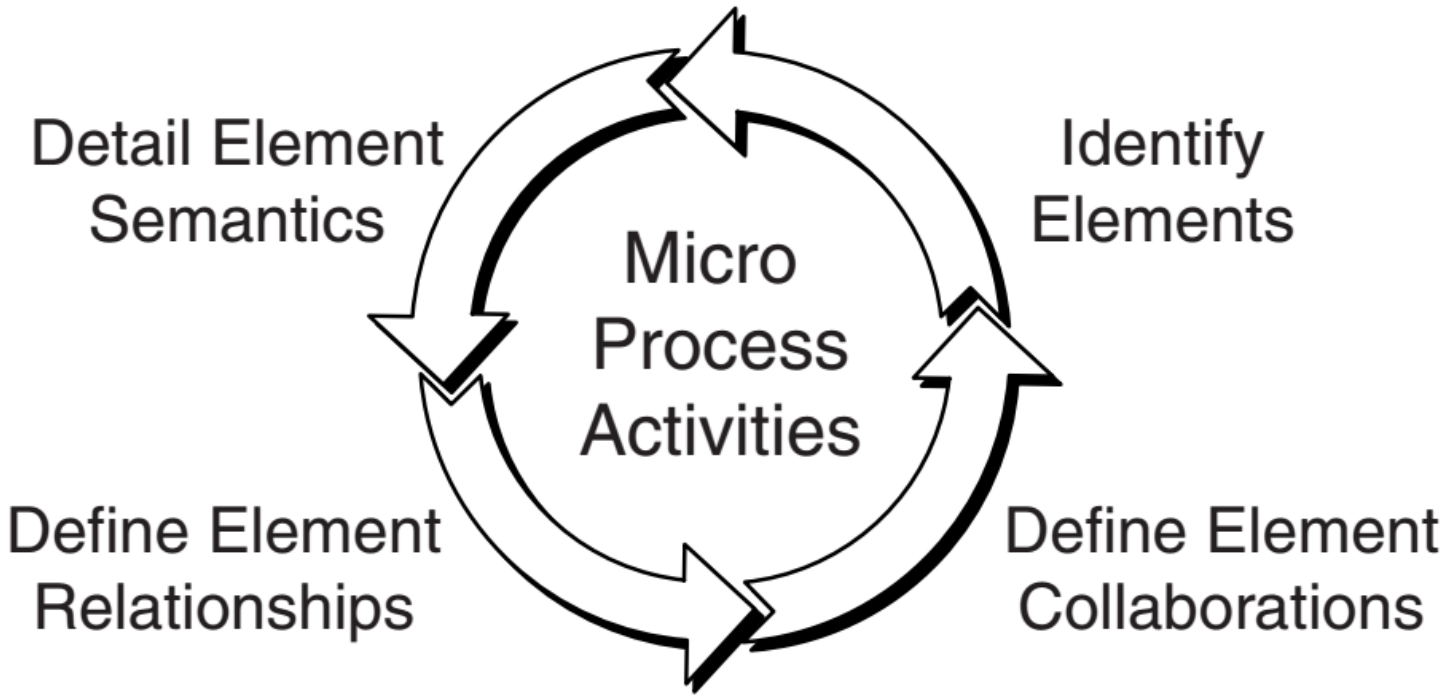


Figure: The Micro Process Activities

Micro Process - Activities

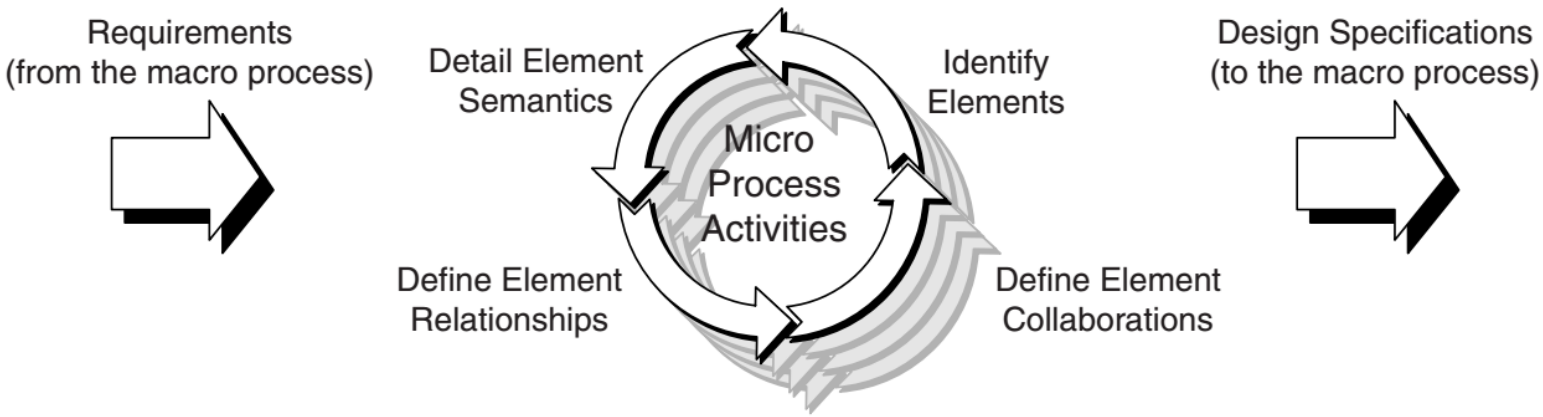


Figure: The Micro Process Activities

Micro Process - Activities

- **Identify the elements :** Discover the elements to work with
- **Define the collaborations :** Describe how the identified elements collaborate to provide the system's behavioral requirements
- **Define the relationships :** Define the relationships between the elements that support the element collaborations
- **Define the semantics :** Establish the behavior and attributes of the identified elements

1. Identify the elements - Description

- Identify the key elements that will be used to describe the solution at a particular level of abstraction.

1. Identify the elements - Description

- Identify the key elements that will be used to describe the solution at a particular level of abstraction.
- Identifying elements at one level of abstraction involves evolving the previous level, which results in new and different elements.

1. Identify the elements - Description

- Identify the key elements that will be used to describe the solution at a particular level of abstraction.
- Identifying elements at one level of abstraction involves evolving the previous level, which results in new and different elements.
- It should focus just on identifying the elements and describing them at a high level (a brief description)

1. Identify the elements - Steps

- During analysis, designers work in conjunction with domain experts to identify the elements

1. Identify the elements - Steps

- During analysis, designers work in conjunction with domain experts to identify the elements
- During design, some elements identified during analysis may turn out to be actual classes, and others may turn out to be simply attributes

1. Identify the elements - Steps

- During analysis, designers work in conjunction with domain experts to identify the elements
- During design, some elements identified during analysis may turn out to be actual classes, and others may turn out to be simply attributes
- It is preferable to investigate similar systems at similar levels of abstraction
- Design patterns should be used to refine analysis elements into design elements.

1. Identify the elements - Products

A reasonably stable analysis/design model is a good indicator about this activity

2. Defining Element Collaborations - Description

- The purpose of the second micro process activity, defining element collaborations, is to describe how the identified elements work together to provide the system's behavioral requirements.

2. Defining Element Collaborations - Description

- The purpose of the second micro process activity, defining element collaborations, is to describe how the identified elements work together to provide the system's behavioral requirements.
- UML interaction diagrams (specifically, sequence and communication diagrams) are very effective for representing these collaborations

2. Defining Element Collaborations - Steps

- **1. Analyze the behavior :** allocating responsibilities to the elements that participate in providing the behavior (i.e., the elements identified during the previous micro process step)
- **2. Collect patterns :** among the realizations, and express these patterns in terms of more abstract, generalized realizations

2. Defining Element Collaborations - Products

The primary products of this micro process activity are the realizations that indicate how the identified elements collaborate to perform the behavioral requirements that are in scope.

2. Defining Element Collaborations - Products

The primary products of this micro process activity are the realizations that indicate how the identified elements collaborate to perform the behavioral requirements that are in scope.
As an indicator of success :

2. Defining Element Collaborations - Products

The primary products of this micro process activity are the realizations that indicate how the identified elements collaborate to perform the behavioral requirements that are in scope.
As an indicator of success :

- A consistent set of elements and responsibilities that provide the required functional behavior of the system.
- A separation of responsibilities between those elements should be clarified.
- There should not be two elements very similar responsibilities

3. Defining Element Relationships - Description

- The purpose of the third micro process activity is to define the relationships between the elements that support the element collaborations defined in the previous micro process activity.

3. Defining Element Relationships - Description

- The purpose of the third micro process activity is to define the relationships between the elements that support the element collaborations defined in the previous micro process activity.
- Identifying the relationships is done to solidify the boundaries of each element and to clearly represent which elements collaborate with each other.

3. Defining Element Relationships - Description

- The purpose of the third micro process activity is to define the relationships between the elements that support the element collaborations defined in the previous micro process activity.
- Identifying the relationships is done to solidify the boundaries of each element and to clearly represent which elements collaborate with each other.
- For the elements whose states drive how they collaborate with other elements, we may use UML state machine diagrams that capture the key state changes

3. Defining Element Relationships - Steps

- **1. Identification of associations :** the initial identification of a semantic connection between elements

3. Defining Element Relationships - Steps

- **1. Identification of associations :** the initial identification of a semantic connection between elements
- **2. Refinement of associations :** Into more semantically rich relationships (e.g., aggregations, dependencies, and so on)

3. Defining Element Relationships - Products

As an indicator of success for this activity, we have specified by the end the relationships among the elements at a specific level of abstraction regarding cohesion, coupling, and completeness.

4. Detailing Element Semantics - Description

- The purpose of this activity is to clarify the behavior and attributes of each identified element at a specific level of abstraction, to define semantics that are consistent across all scenarios the element participates in.

4. Detailing Element Semantics - Description

- The purpose of this activity is to clarify the behavior and attributes of each identified element at a specific level of abstraction, to define semantics that are consistent across all scenarios the element participates in.
- In this activity, the element semantics are refined at the current level of abstraction in sufficient enough detail to enable the identification of elements at the next level of abstraction.

4. Detailing Element Semantics - Description

- The purpose of this activity is to clarify the behavior and attributes of each identified element at a specific level of abstraction, to define semantics that are consistent across all scenarios the element participates in.
- In this activity, the element semantics are refined at the current level of abstraction in sufficient enough detail to enable the identification of elements at the next level of abstraction.
- While the first three activities focus on an outside view of the elements and how they collaborate, this activity focuses on each of the elements individually.

4. Detailing Element Semantics - Steps

Detailing an element's semantics involves the selection of the structures and algorithms that describe the structure and the behavior of the element.

4. Detailing Element Semantics - Products

The primary measure of goodness for this activity is simplicity. Elements having complex semantics, could mean that they are poorly represented.

Summery about Analysis and Design

- Successful projects apply a well-managed iterative and incremental development lifecycle.
- The choice of lifecycle style (e.g., waterfall, iterative, ..) affects the macro process, and the choice of analysis and design techniques (e.g., structured, object-oriented, ..) affects the micro process
- Despite the macro process chosen, each of the micro techniques can be applied equally
- The purpose of the micro process is to take the requirements provided by the macro process and produce analysis and design specifications that are fed back into the macro process

References

1. Grady Booch et al, *Object-oriented Analysis and Design with Applications, 3rd edition* , Addison-Wesley Professional, 2007
2. Mike O'Docherty, *Object-Oriented Analysis and Design Understanding System Development with UML 2.0* , WILEY, 2005