Implementation of UML diagrams in IS design

Marina Ivašić-Kos, Mile Pavlić, Patrizia Poščić

Faculty of Philosophy, Rijeka

Department of Computer Science

marinai@pefri.hr

mile.pavlic@ris.hr

patrizia@pefri.hr

ISS 2004, Varaždin

1. Introduction

- The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.
- The UML defines numerous diagrams and their meanings, but not software developement procedures, sequence of using diagrams, tasks allocation or similar.
- The UML is a modeling language, not a method or methodology.
- This paper analyses UML v.1.5 diagrams and proposes most appropriate ones for IS modelling, observing use-case driven, iterative and incremental development.
- A proposal for IS development life cycle by using UML, (possible proposal for methodic) will be presented.

2. Views and Diagrams

The UML defines the following diagrams

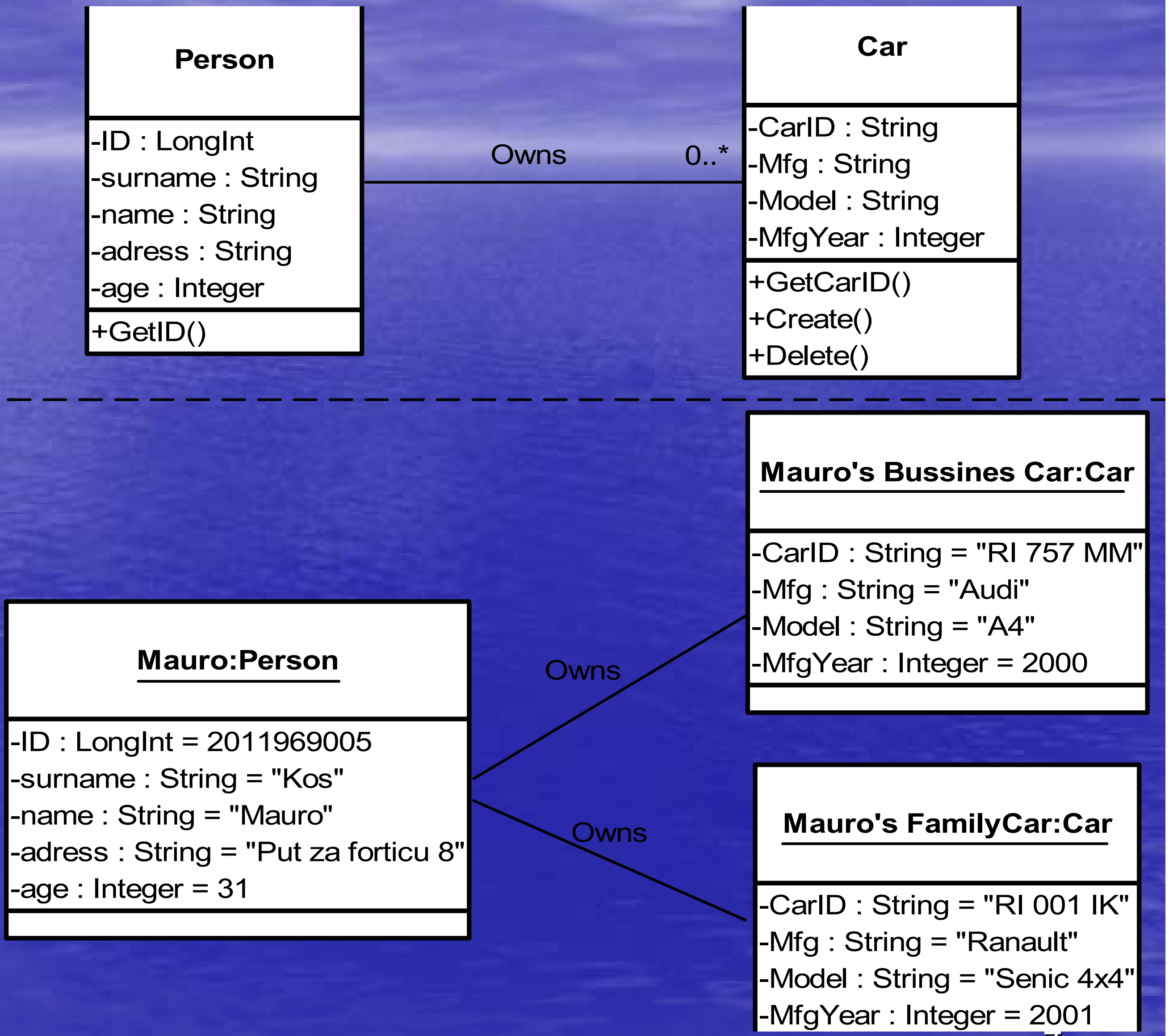
- view of usage........................use case diagram
- statical view ....class, object diagram
- behavioral view ....behavior diagrams:
  - statechart diagram
  - activity diagram
- interactional view ....interaction diagrams:
  - sequence diagram
  - collaboration diagram
- implementational view ....implementation diagrams:
  - component diagram
  - deployment diagram

2.1 Class and Object Diagram

(Static Structure Diagrams)

- static structure of the model
(entities that exist,
 their internal structure,
 and their relationships)

- class diagram shows classes and their relationships, while object diagram shows their specific instances (possible snapshot of a system state)



Person

-ID : LongInt
-surname : String
-name : String
-adress : String
-age : Integer

+GetID()

Car

-CarID : String
-Mfg : String
-Model : String
-MfgYear : Integer

+GetCarID()
+Create()
+Delete()

Mauro:Person

-ID : LongInt = 2011969005
-surname : String = "Kos"
-name : String = "Mauro"
-adress : String = "Put za forticu 8"
-age : Integer = 31

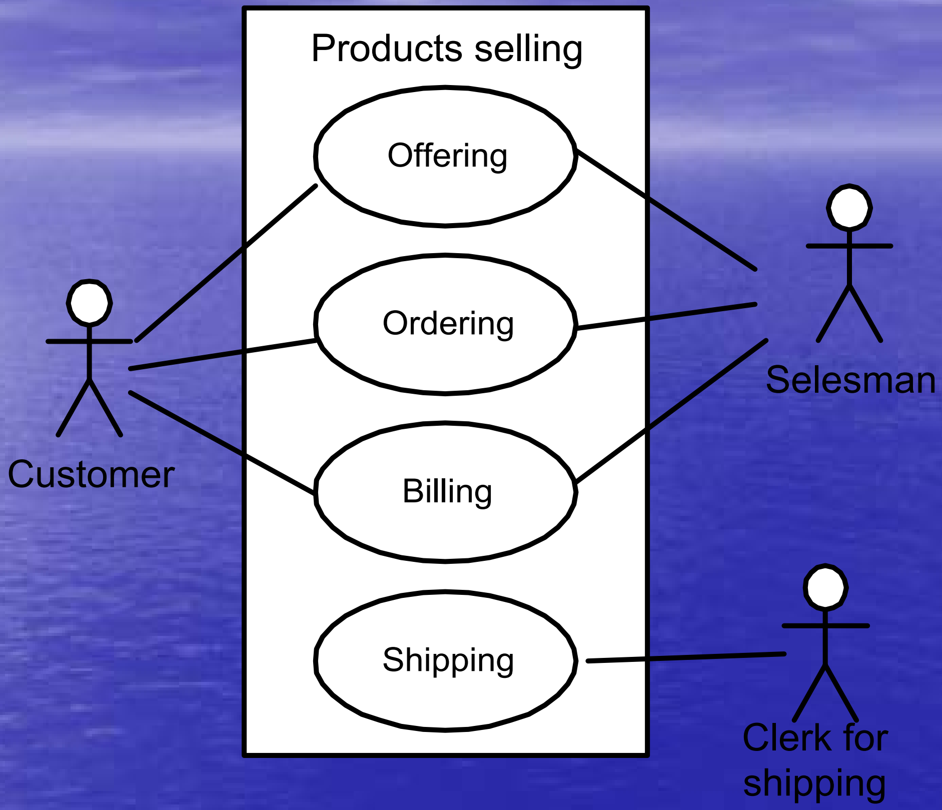
Mauro's Bussines Car:Car

-CarID : String = "RI 757 MM"
-Mfg : String = "Audi"
-Model : String = "A4"
-MfgYear : Integer = 2000

Mauro's FamilyCar:Car

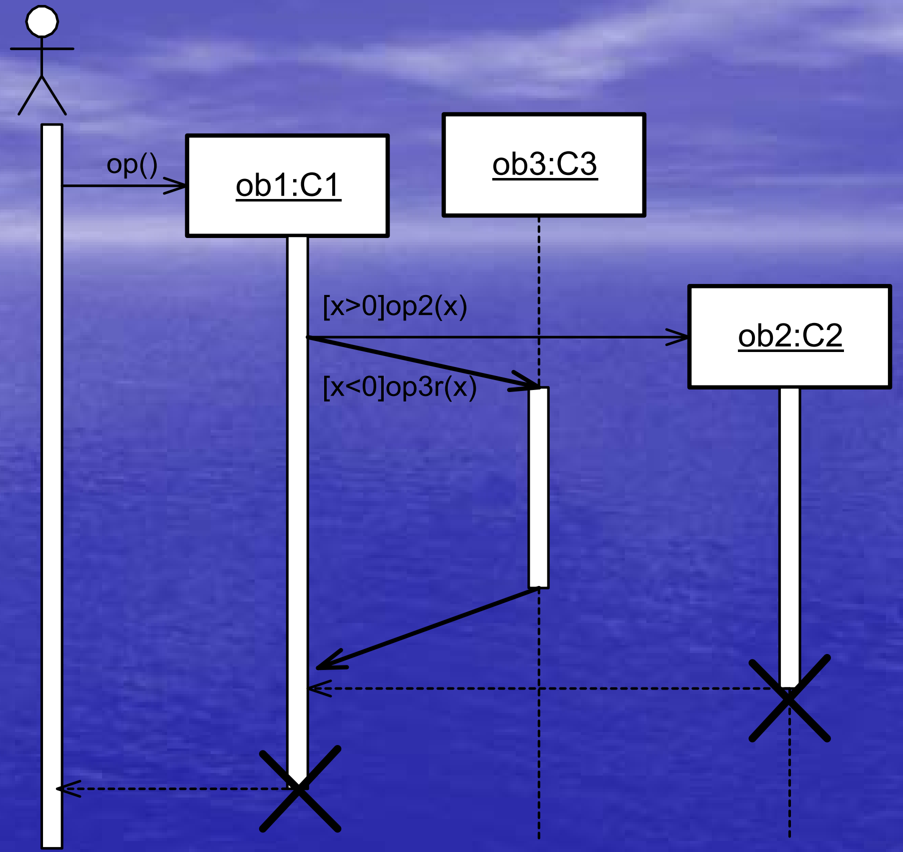
-CarID : String = "RI 001 IK"
-Mfg : String = "Ranault"
-Model : String = "Senic 4x4"
-MfgYear : Integer = 2001

2.2 Use Case Diagram



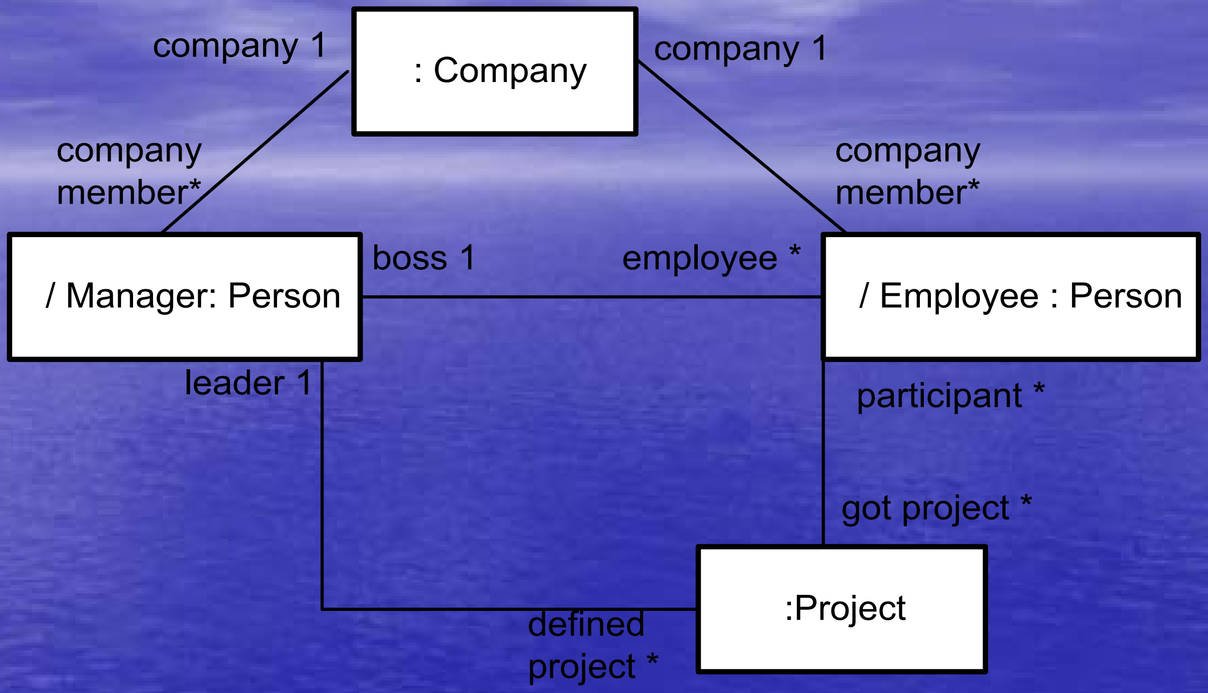
- defines requirements and describes functionality of a system
- explains how actor can use the system
- described from the actor's point of view (system behavior experienced by user)

2.3 Sequence Diagram (Interaction Diagram)



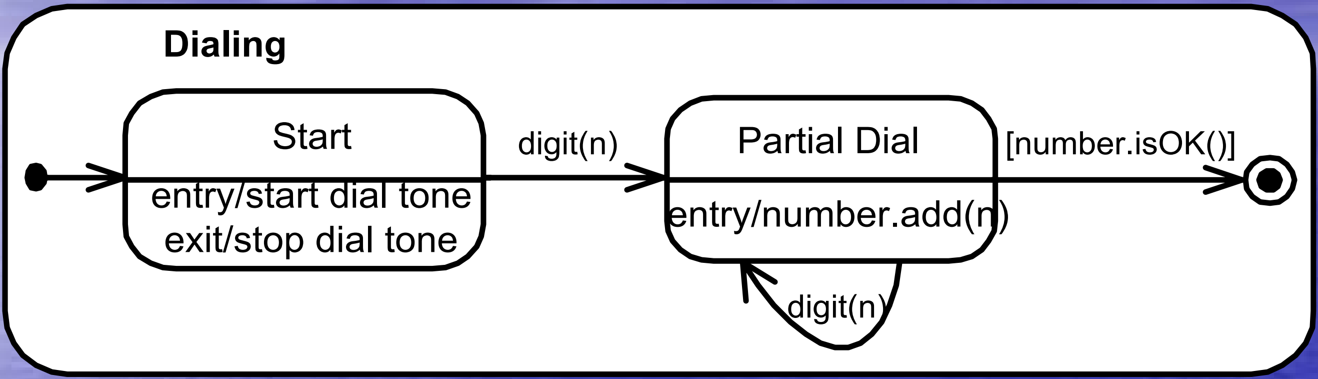
- shows explicit sequential exchange of message among objects through a flow or function (vertikal time lines)
- better for real-time specifications of communication among objects and for complex scenarios
- exists in: - general form (describing all possible sequences)
  - form of instance (sequence according to the general form)

2.3 Collaboration Diagram
(Interaction Diagram)



- presents interactions among participants playing different roles and their relationships that are meaningful for a given set of goals
- better for understanding the effect at given instance or for procedure design
- two levels of abstraction:
  - specialized level (describing classifier or association role, message)
  - instance level (describing object, common relationships, stimuli)

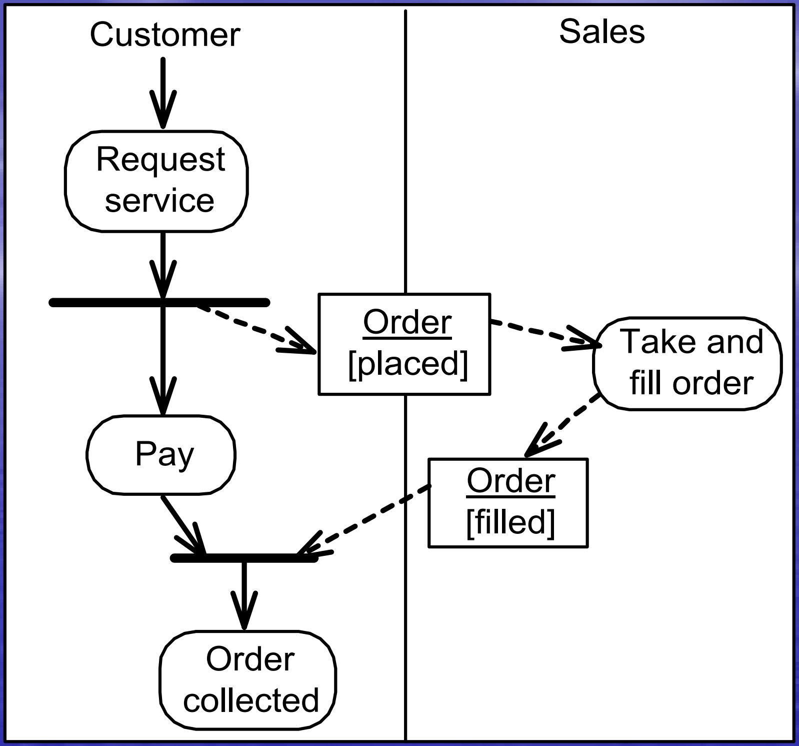
2.4 Statechart Diagram
(Behavior Diagram - State Machine)



- shows behavior of model elements capable of dynamic behavior
- describes possible sequences of states and actions through which element can proceed during its lifetime while reacting to received event instances (signals, operation invocations)

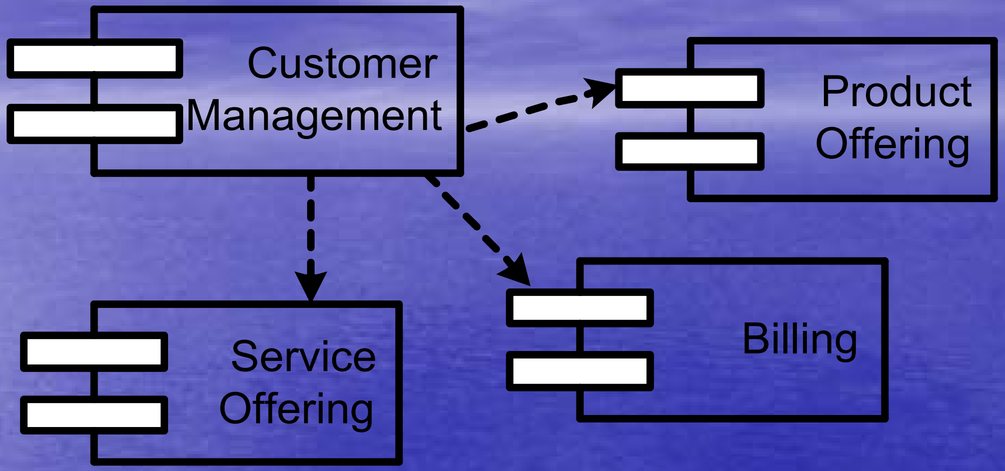
The statechart diagram is quite opposite to class description because it presents all possible states that a class object can assume and events it causes as state changes.

2.5 Activity Diagram
(Behavior Diagram)



- a special case of the statechart diagram (all states are actions/ activities and transitions are triggered by completion of actions/activities in the original states)
- used to describe activities performed within an operation (where most of the events represent the completion of internally-generated actions)
- presents specifications of messages being sent or received within activities and objects representing either action input or output

2.6 Component Diagram
(Implementation Diagram)



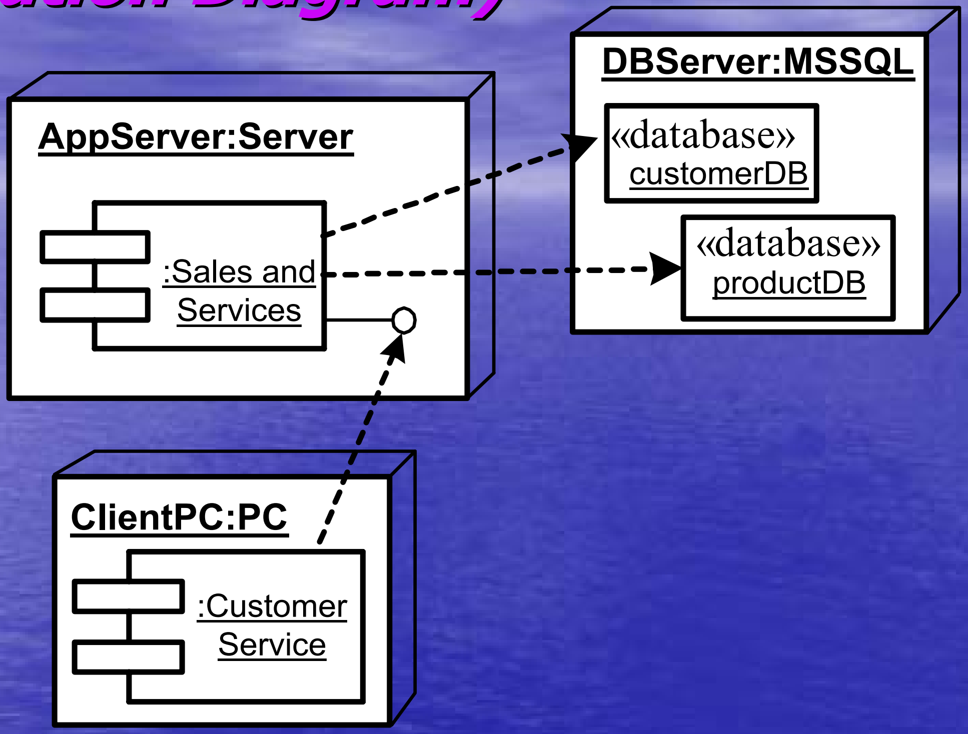
- graph of components (software components including source code, binary code and exe files) connected by dependency relationships

- only in a type form (for instances deployement diagram is used)

- shows physical structure of code in terms of code components (some exist in compilation time, other in link time, run time or in several times)

- may present interface and calling dependencies among components (for classifier that have operations and interface)

2.7 Deployment Diagram (Implementation Diagram)



- graph of nodes (physical object that represents a processing resource having a memory and processing capability; devices, human resources mechanical processing) connected by communication associations

- shows the configuration of process element in real time, program components representing software code units in real time and object that reside on them

3. Using Diagrams in IS Design

- D. Rosenberg: «Modeling with UML is similar to sitting down to an huge plate of food, the thought that you can't possibly eat everything on the plate just kills your appetite before you get started.

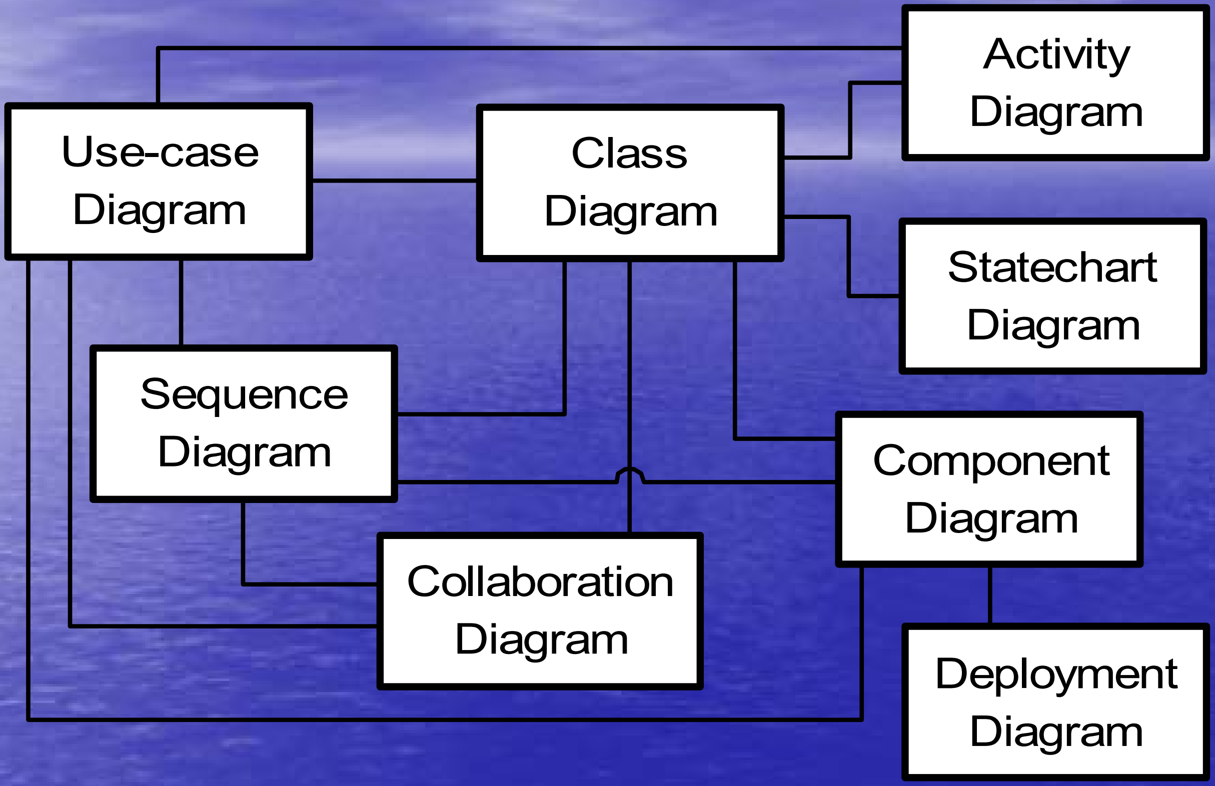
  A similar phenomenon can occur with UML modeling. The thought of having to produce a complete set of sequence, collaboration, state, deployment, use case, and class diagrams that comprehensively covers each and every use case of the system with a fully detailed dynamic model can intimidate a team right out of object-oriented analysis and design. «

- Not all diagrams have to be used in solving a particular problem (only those that facilitate communication, contibute to a better problem understanding and simplify development)

- To build a model of an IS, it is necessary to identify an appropriate set of objects and assign them adequate system behavior (the most appropriate are **use-case**, **class** and **interaction** diagrams)

- Other diagrams may be used as well (**component** diagrams for distributed applications, **object** diagrams for complex class diagrams)

3.1 UML diagrams from an iterative point of view



- The UML authors recommend use case-driven, iterative, and incremental approach

- Diagrams are interconnected due to the iterative nature of OO modeling and incremental approach

- The **use case** diagram defines requirements and context of the system at the highest level of abstraction (system functionality)

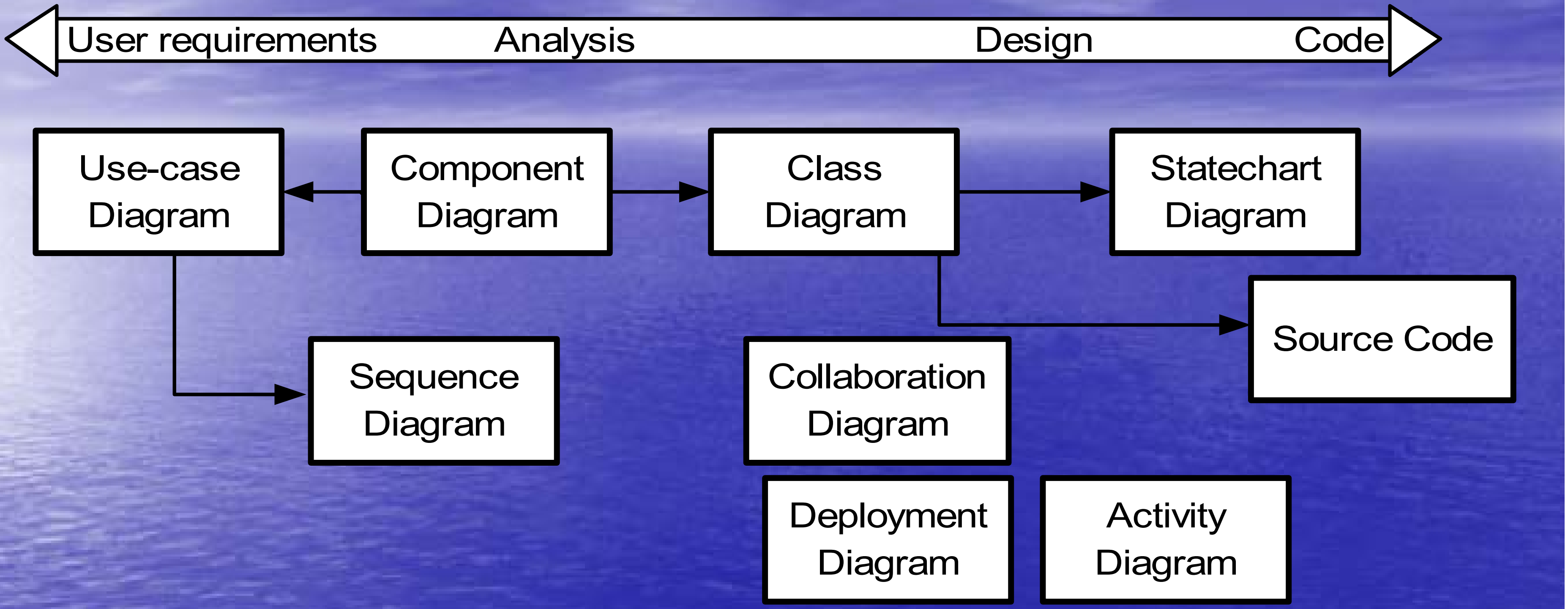
- Lower levels of abstraction can be shown on **interaction** diagrams that present the behavior distribution among classes (Sequence diagram shows logic of a particular use case, while collaboration diagram presents logics consisting of several use cases with asynchronous messages exchanged among objects)

- While drawing sequence diagrams, operations and new objects are identified, thus while drawing up the dynamic model, the system static structure is built

- The static structure of the system is presented on **class** diagram (At analyses, only objects and operations necessary for understanding the system can be defined, while at design should be defined all of them in details)

- The completeness and integrity of the system static structure is provided through the use case and sequence diagrams

3.2 UML diagrams from a serial point of view



- In reviewing IS development phases and diagrams predominantly used at particular phase, a serial usage of diagrams can be considered.

- Although the iterative approach is promoted, the fact remains that it is first necessary to draw up a strategic study, to identify and analyze requirements and then design software and produce code

Way of using UML diagrams according to the IS development life cycle phases (proposal for methodic)

Phase	Diagram/Method
Strategic study	Use-case diagram Other strategic planning methods
Defining requirements	Use-case diagram Textual description
Analyses	Use-case diagram – detailed operations Sequence diagram Class diagram Components diagram
Design	Class diagram; Object diagram Use-case diagram – detailed operations Sequence diagram; Collaboration diagram Components diagram; Deployment diagram
Prototyping and softver production	State diagram Activity diagram Implementation using CASE tools (Rose, ...) and programming in OOPL (Java, C++, ...)
Implementation and application	Entering database initial states; Users training; Testing; Writing help; Optimization; Final testing – delivery; Application of software product

4. Conclusion

- A complex system is best approached through a small set of nearly independent views of a model.

- UML is rather complex, so it is necessary to define sets of diagrams to be used in the context of particular problem domain. Strict set boundaries should not be defined because, if necessary, other diagrams and techniques, defined either within or outside the UML, may be used.

- All UML diagrams may not necessarily be implemented in development of the IS. Implementation primarily depends on complexity and size of a system, applied CASE tool, programing language, OO database and other conditions.

- Use case, sequence and class diagrams are most appropriated for IS modeling. This set may be expanded with object, components and collaboration diagrams.

- According to the IS development life cycle phases, a way of using UML diagrams is proposed (possible proposal for methodic). Further research would define the sequence of diagram usage more precisely.