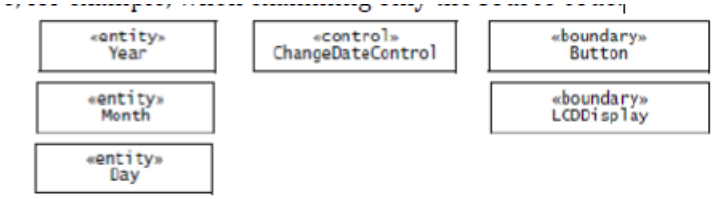
II. Please explain what do you mean by: entity, boundary and control objects. In which model these objects are used? How are these represented in UML?

The analysis object model consists of **entity, boundary**, and **control** objects. **Entity objects** represent the persistent information tracked by the system. **Boundary objects** represent the interactions between the actors and the system. **Control objects** are in charge of realizing use cases. Modeling the system with entity, boundary, and control objects provides developers with simple heuristics to distinguish different, but related concepts.

To distinguish between different types of objects, UML provides the stereotype mechanism to enable the developer to attach such meta-information to modeling elements. For example, in figure above, we attach the «control» stereotype to the ChangeDateControl object. In addition to stereotypes, we may also use naming conventions for clarity and recommend distinguishing the three different types of objects on a syntactical basis: control objects may have the suffix Control appended to their name; boundary objects may be named to clearly denote an interface feature (e.g., by including the suffix Form, Button, Display, or Boundary); entity objects usually do not have any suffix appended to their name. Another benefit of this naming convention is that the type of the class is represented even when the UML stereotype is not available, for example, when examining only the source code.



III. What is the purpose on modeling in Software Engineering?

To enable a better management of the complexity by supporting:

- an easier and better understanding of the problem,
- an easier communication among developers and clients, a clearer communication inside developers,
- reusing at a higher abstraction level because modeling languages are independent from different technologies,
- simulation of different parts of the system in order to validate some ideas and solutions, a.s.o.

IV. Please explain what do you mean by "Design by Contract" (shortly DBC)? Which are the assertions used in DBC? What are the advantages of DBC vs. Defensive Programming?. 1pt

As the name suggests, **DBC** is **a design methodology** considering that relationships between two software components are **very similar to relationships governing the world of business**, where the relationships are clearly stated in **contracts**. In each contract, both the rights and duties of the two parties: client and server are mentioned. In the software domain, client constraints (duties) are named **preconditions** and server constraints **postconditions**. DBC offers many advantages compared with defensive programming:

- the constraints are not restricted to the parameter values,
- all constraints are public, being located outside of the method body. So, the methods' body are smaller and by consequence easier to understand.

- pre & postconditions support the understanding of types because these are located in the interface
- pre & postconditions support testing.

V. From the point of view of granularity level, how many kinds of design to you know? Please describe shortly each of them.

There are two types of design:

- **the system design**, concerning the design of the system architecture in large. The components of system design are: subsystems, partitions, layers. At this level there are some architectural styles: multi layer, client-server, repository, pipe&filter, MVC a.s.o
- **the object design**, concerning the design of the subsystem components inside the subsystem. At this levels we discuss about design patterns.

IV. Using your own words, please explain all the information included in an UML model which can be used to generate the code correspondig to the model, in an automated manner. Which information is in your oppinion not yet used in the code generated and why? Which are in your opinion the drawbacks of automated code generation? Why?

Almost all the tools supporting automatic code generation of UML models uses the architectural information which specifies all the components of the architectural view (classes, interfaces and packages including their structure and behavioral elements - opertions), and different kinds of relationships among these components. Inheritance and associations among classes, implementation between classes and interfaces, import of different components of a package into other packages (specified in UML by dependency relationships). There are different levels of detail regarding these implementations. The complete management of all kinds of associations is the most importat. Also, there are differences among the implementation of explicit constructors. However, we may conclude that all the architectural information could be used in direct engineering. The usage of the information provided by the behavioral view is done only partial. There are some kind of diagrams like STDs and ADs(Activity Diagrams) whose information is apropriate to be used in direct engineering. The Sequence Diagrams or Collaboration/Communication Diagrams do not offer the whole information needed generating in an automated manner all the code corresponding to the described behavior. Only method composition and the sequence of sending messages is specified in diagrams, the rest of the code must be implemented manually.

However, the code corresponding to observers and constraints can be completely generated in an automated manner, as happened in OCLE for example.

To be really uselful, the automatic code generation needs to be preceded by checking UML model compilability and behavior checking by means of simulations of apropriate model instantiations.

II. What do we mean by requirement elicitations? In this context, please shortly describe requirement elicitations activities. Please mention the information used in the template used to describe use cases. Name the UML view specified by means of use cases.x

Requirements elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem. Such a definition is called a **requirements specification** and serves as a contract between the client and the developers.

Requirements elicitation includes the following activities:

- **Identifying actors.** During this activity, developers identify the different types of users the future system will support.
- **Identifying scenarios.** During this activity, developers observe users and develop a set of detailed scenarios for typical functionality provided by the future system. Scenarios are concrete examples of the future system in use. Developers use these scenarios to communicate with the user and deepen their understanding of the application domain.
- **Identifying use cases.** Once developers and users agree on a set of scenarios, developers derive from the scenarios a set of use cases that completely represent the future system. Whereas scenarios are concrete examples illustrating a single case, use cases are abstractions describing all possible cases. When describing use cases, developers determine the scope of the system.
- **Refining use cases.** During this activity, developers ensure that the requirements specification is complete by detailing each use case and describing the behavior of the system in the presence of errors and exceptional conditions.
- **Identifying relationships among use cases.** During this activity, developers identify dependencies among use cases. They also consolidate the use case model by factoring out common functionality. Between use cases we may have: include relationships, extend relationships and generalization/specialization relationships
- **Identifying nonfunctional requirements.** During this activity, developers, users, and clients agree on aspects that are visible to the user, but not directly related to functionality. These include constraints on the performance of the system, its documentation, the resources it consumes, its security, and its quality.

A use case specifies all possible scenarios for a given piece of functionality. A use case is initiated by an actor. After its initiation, a use case may interact with other actors, as well. A use case represents a complete flow of events through the system in the sense that it describes a series of related interactions that result from its initiation.

The template used to describe use cases contains:

- Use case name
- Participating actors
- Flow of events
    - Normal (Main) flow of events
    - Exceptional flow of events (described by means of extend relationships and by mentioning extension points)
- Entry conditions
- Exit conditions
- Quality requirements

The view described by means of use cases if the functional view.

III. Please mention the techniques that can be used in order to deal with faults. 1 pt

The techniques used to deal with faults depends on the moment of system development and system usage we are. So:

- **Before the system release**, the techniques used for fault avoidance are:
  - The use of a methodology to reduce complexity,
  - The configuration management to prevent inconsistencies,
  - Doing verifications to prevent algorithmic faults,
  - Use Reviews
- **While system (a part of the system or the whole system) is running**, techniques supporting **Fault detection:**
  - **Testing:** Activities to provoke failures in a planned way,
  - **Debugging:** Finding and removing the cause (Faults) of an observed failure,
  - **Monitoring:** Delivering information about state => Used during debugging
- **Recover from failure after release**, techniques supporting **Fault tolerance:**
  - Exception handling,
  - Modular redundancy.

III. Please explain the main rationales of using diagrams in modeling. What's the main advantage of using diagrams to specify models?

When using graphical modeling language, diagrams are used first of all, to specify/construct models. Diagrams are well suitted to suport the communication between different category of developers: analysts, designers, programmers, testers, between different categories involved in software projects: clients, end users, domain experts, developers a.s.o. This is because because an image value more that 1000 words. Moreover, compared with textual specifications, diagrams support a better, easier and finner filtration of information.

IV. Which is the generic name of modeling languages using diagrams? But of modeling languages that can't use diagrams? What's the name of the syntax describing the categories of languages previously mentioned? But of the syntax describing the grammar of the modeling language? 1.5pt

Taking into account nature of the formalism used to specify models, modeling languages are included in one of the two categories/families of modeling languages are known as graphical modeling languages or textual modeling languages. In fact, most of graphical modeling languages also use a textual formalism. In case of the UML, the textual formalism which complements the graphical specification is OCL. So, graphical modeling languages try to combine the advantages of both categories of languages. The concrete syntax can be graphical or textual and describes the category of modeling languages. However the first category is named graphical modeling language due to the fact that mainly, the language is graphical and the textual part only complements the graphical languages. The grammar of modeling languages is specified by the abstract syntax, described by means of the metamodel and of constraints usually referred as Well Formedness Rules(WFRs).

V. Please explain the manner in which using modeling supports the increase of abstraction in specifying problems and solution to different problems. Why increasing the abstraction in specification is the most important vehicle of software development?

Abstraction mean the use of only the most important/representative information when describe a problem or a solution to a problem. Apart from programming languages, modeling languages split the above mentioned description in many parts, each concerning a separate view:structural/architectural, behavioral, deployment, implementation, testing. In this manner, the abstraction is increased by reducing the number of concepts. The price to pay is the need to check that between different views of the same concept or of related concepts, there are no contradictions. The capacity of people in managing/operating simultaneously with different concepts is restricted at 7+/- 2 conceps. That's why reducing the number of concepts is crucial when working with complex problems.

3. Using your own words, please describe shortly the Software Engineering concept.

A set of principles, methods, methodologies, techniques and tools supporting the production of quality software:

- within a given budget,
- before a given deadline,
- while changes occurs.

4. What it is and what is not software testing? Please explain which is the main mistake of managers when assign team members to software testing activities.

Testing support discovering as much as possible bugs in the system in order to fix them. Testing is not to prove that a software has not bugs and runs only as previewed, simply because this is not possible. "Testing can only show the presence of bugs, not their absence" (Dijkstra).

Testing is often viewed as a job that can be done by beginners. Managers would assign the new members to the testing team, because the experienced people detested testing or are needed for the more important jobs of analysis and design. Unfortunately, such an attitude leads to many problems. To test a system effectively, a tester must have a detailed understanding of the whole system, ranging from the requirements to system design decisions and implementation issues. A tester must also be knowledgeable of testing techniques and apply these techniques effectively and efficiently to meet time, budget, and quality constraints.