The most important ideas and topic of the SE course

As the title of this discipline suggests, SE concerns the concepts, principles, techniques, and technologies we (software developers) may use in order to produce and deliver the required software systems in time, within the established budget, considering the challenges of these activities, especially the requirements changing and at the promised/expected quality. Changings are due to different reasons(rationales): technologies changing, client requirement changing due to an incomplete understanding of the problem and domain changing due to a new and increased level of requirements or services, a.s.o.

The level of quality required for new systems is continuously increasing. By consequence, the complexity of these systems increases in a similar manner. Therefore, we need appropriate approaches, techniques, and tools to manage a permanent increasing complexity. Conceiving, specifying, and using models is the manner proposed to manage software development.

Using models is an ancient technique used from managing the development of new and complex products. However, the abstraction level of software models is higher compared to other kind of models because, in case of software models we usually have many views/models of the same model. We may have a functional view, a dynamic view, an architectural view in case of the analysis model, a system design model, an object design model, user interface design model in case of the design model, a component view, or a deployment view in case of executable view.

The requirements models, the analysis models, the design models, the execution models are different kind of models conceived in different phases of software development, starting with requirements elicitation, and ending with system implementation and testing. Each of the above-mentioned models can be described from different points of view/interest. That is why we prefer to use the term of view, and not the term of model. View is a more detailed term – it describes a model specified from a narrow point of view; by consequence, the number of terms used is diminished, and the views are simpler than a whole model which is described usually by means of different views. So, a view is easier to be understood. Models are meant to support: an easier and quicker understanding of the problem, to support the communication among clients, final users, domain experts and developers, between different categories of developers and between different members of the same team. However, it is important to realize that the objective of models do not stop here. We need to understand that for each development phase, models must

be improved. Passing from one phase to another, models are transformed by adding new components, details and of course by improving the models.

Models are specified by using appropriate languages referred in the literature as modeling languages. In this context, the language we used in the course is the Unified Modeling Language – acronym UML, a general modeling language characterized by a graphical concrete syntax and by a complementary textual language named Object Constraint Language – acronym OCL.

In fact, most of modeling specialists agree that UML is a family of modeling languages, one or more languages for each view. Each diagram is a filtered projection of a view or of a part of a view. Diagrams corresponding to different views describe the same concepts or related concepts from different point of views. The price to pay for increasing the level of abstraction is to check that between the diagrams above mentioned/diagrams associated to different views of the same model, the specifications must be non-contradictory. A remarkably simple example, the method signature in a class diagram, must correspond with the signature of the corresponding message in a sequence diagram or a collaboration diagram. Apart of this non-contradictory situation, all diagrams, all models' specifications must comply with the grammar of the modeling language. In UML, the grammar is also named abstract syntax specified by the metamodel and by attached invariants, named Well Formedness Rules WFRs. Models complying with the above-mentioned rules are compilable models, a similar term with compilable programs.

Model compilability is a mandatory requirement if automatic code generation is among our objectives of using models. This implies a correct understanding of the UML model specification.

The objectives of different kind of models and the relationships between different kind of models depends on the pair of models. The requirements model is meant to support requirement elicitation. The analysis model is meant to support the understanding of the domain specific model. On the other hand, analysis model describes in a more formal manner the requirements model. That is why the analysis model is used in requirements validation. The analysis model describes the problem domain as it is. That is why it is said that we have only one analysis model. Contrary we may have many design models because the design models explain the solution that each designer give to the problem specification. Another feature of the design model is that we have many design models, specified at different abstraction level. At the highest level of abstraction, we have a system design model in which the architecture of the design model is represented. In this model the level of designing the solution is represented by means of relationships between different subsystems. Contrary, the object design model details each subsystems architecture presenting the relationships between classes and interfaces. In the object design model, the usage of design patterns is highlighted. Using design patterns is meant to support a more extensible solution, a more resilient system to possible changes that may appear. The system design is more detailed compared with the analysis model because the full

signature of methods is described. Also, in the design model all the assertions are expected because these supports a complete and unequivocal specification of the system.

It is known that the time frame for the emergence of new software technologies is shorter and shorter. This is another reason for using independent platform models. We are interested in delaying the moment of fixing the technology. This is advantageous because transforming any platform independent model in a platform dependent system may be done in an automated manner (automatic code generation). Once the rules of transformation are successfully tested, the transformation can be applied as many times as needed instantly.

A particularly important aspect related to models concerns the usage of assertions. First, assertions support developers to understand better the problem and the proposed solution. Even in case of simple models, these cannot be specified in a clear and unequivocal manner without using assertions. Moreover, using OCL to specify observers is very profitable.

Finally, testing is a mandatory activity in developing software systems. This is mandatory because we do not have mathematical tools to prove the correctness of a system. To increase our confidence in the system we must conceive tests meant to prove that the system does not work in a correct manner.

The chapters you need to read from the Bruegge's book are:

Chapter 1 - Introduction to Software Engineering
Chapter 2 - Modeling with UML
Chapter 4 - Requirements Elicitation
Chapter 5 – Analysis
Chapter 6 - System Design: Decomposing the System
Chapter 7 - System Design: Addressing Design Goals
Chapter 8 - Object Design: Reusing Pattern Solutions
Chapter 9 - Object Design: Specifying Interfaces
Chapter 10 – Mapping Models to Code
Chapter 11 – Testing

Also, we are advised to use all the slides uploaded on the classroom, the articles/papers, the models. Of course, reading UML and OCL docs is a particularly good activity.