Lecture 1

Topic: The importance of Modeling

Purpose of the lecture:

In this lecture students will have to understand the necessity of a modeling technology in social real project work.

Basic terms of the lecture:

Modeling; Architectural models; Mathematical models

Short abstracts:

Now a days modeling technology is developed very fast, and applied in a variety of aspects. What is model? How to model, and the principles of model is the most important concepts of the modeling technology.

Main questions of the lecture:

1) Why we study modeling technology.
2) Describe the main purpose of the modeling technology.
3) What is modeling technology?
4) Describe the principles of the modeling technology.
5) How to use model.

Recommended list of literature sources:

1. Zeigler, B. P., T. G. Kim, and H. Praehofer. (2000). Theory of Modeling and Simulation. New York, NY, Academic Press.
2. Object Management Group, "Unified Modeling Language: Superstructure", Version 2.1.2, November 3, 2007.
3. Dubois, G. (2018) "Modeling and Simulation", Taylor & Francis, CRC Press.
4. Embley, D.W., Thalheim, B. (eds.): Handbook of Conceptual Modeling Theory, Practice, and Research Challenges. Springer, Berlin (2011)

Main contents of the lecture:

If you want to build a dog house, you can pretty much start with a pile of lumber, some nails, and a few basic tools such as a hammer, saw, and tape measure. In a few hours, with little prior planning, you'll likely end up with a dog house that's reasonably functional, and you can probably do it with no one else's help. As long as it's big enough and doesn't leak too much, your dog will be happy. If it doesn't work out, you can always start over, or get a less demanding dog.

If you want to build a house for your family, you can start with a pile of lumber, some nails, and a few basic tools, but it's going to take you a lot longer, and your family will certainly be more demanding than the dog. In this case, unless you've already done it a few dozen times before, you'll be

better served by doing some detailed planning before you pound the first nail or lay the foundation. At the very least, you'll want to make some sketches of how you want the house to look. If you want to build a quality house that meets the needs of your family and of local building codes, you'll need to draw some blueprints as well, so that you can think through the intended use of the rooms and the practical details of lighting, heating, and plumbing. Given these plans, you can start to make reasonable estimates of the amount of time and materials this job will require. Although it is humanly possible to build a house yourself, you'll find it is much more efficient to work with others, possibly subcontracting out many key work products or buying pre-built materials. As long as you stay true to your plans and stay within the limitations of time and money, your family will most likely be satisfied. If it doesn't work out, you can't exactly get a new family, so it is best to set expectations early and manage change carefully.

If you want to build a high-rise office building, it would be infinitely stupid for you to start with a pile of lumber, some nails, and a few basic tools. Because you are probably using other people's money, they will insist upon having input into the size, shape, and style of the building. Often, they will change their minds, even after you've started building. You will want to do extensive planning, because the cost of failure is high. You will be just a part of a much larger group responsible for developing and deploying the building, so the team will need all sorts of blueprints and models to communicate with one another. As long as you get the right people and the right tools and actively manage the process of transforming an architectural concept into reality, you will likely end up with a building that will satisfy its tenants. If you want to keep constructing buildings, then you will want to be certain to balance the desires of your tenants with the realities of building technology, and you will want to treat the rest of your team professionally, never placing them at any risk or driving them so hard that they burn out.

Curiously, a lot of software development organizations start out wanting to build high rises but approach the problem as if they were knocking out a dog house.

Sometimes, you get lucky. If you have the right people at the right moment and if all the planets align properly, then you might, just might, get your team to push out a software product that dazzles its users. Typically, however, you can't get all the right people (the right ones are often already overcommitted), it's never the right moment (yesterday would have been better), and the planets never seem to align (instead, they keep moving out of your control). Given the increasing demand to develop software quickly, development teams often fall back on the only thing they really know how to do wellpound out lines of code. Heroic programming efforts are legend in this industry, and it often seems that working harder is the proper reaction to any crisis in development. However, these are not

necessarily the right lines of code, and some projects are of such a magnitude that even adding more hours to the workday is not enough to get the job done.

If you really want to build the software equivalent of a house or a high rise, the problem is more than just a matter of writing lots of softwarein fact, the trick is in creating the right software and in figuring out how to write less software. This makes quality software development an issue of architecture and process and tools. Even so, many projects start out looking like dog houses but grow to the magnitude of a high rise simply because they are a victim of their own success. There comes a time when, if there was no consideration given to architecture, process, or tools, the dog house, now grown into a high rise, collapses of its own weight. The collapse of a dog house may annoy your dog; the failure of a high rise will materially affect its tenants.

Unsuccessful software projects fail in their own unique ways, but all successful projects are alike in many ways. There are many elements that contribute to a successful software organization; one common thread is the use of modeling.

Modeling is a proven and well-accepted engineering technique. We build architectural models of houses and high rises to help their users visualize the final product. We may even build mathematical models to analyze the effects of winds or earthquakes on our buildings.

Modeling is not just a part of the building industry. It would be inconceivable to deploy a new aircraft or an automobile without first building models from computer models to physical wind tunnel models to full-scale prototypes. New electrical devices, from microprocessors to telephone switching systems, require some degree of modeling in order to better understand the system and to communicate those ideas to others. In the motion picture industry, storyboarding, which is a form of modeling, is central to any production. In the fields of sociology, economics, and business management, we build models so that we can validate our theories or try out new ones with minimal risk and cost.

What, then, is a model? Simply put,

A model is a simplification of reality.

A model provides the blueprints of a system. Models may encompass detailed plans, as well as more general plans that give a 30,000-foot view of the system under consideration. A good model includes those elements that have broad effect and omits those minor elements that are not relevant to the given level of abstraction. Every system may be described from different aspects using different models, and each model is therefore a semantically closed abstraction of the system. A model may be

structural, emphasizing the organization of the system, or it may be behavioral, emphasizing the dynamics of the system.

Why do we model? There is one fundamental reason.

We build models so that we can better understand the system we are developing.

Through modeling, we achieve four aims.

1. Models help us to visualize a system as it is or as we want it to be.
2. Models permit us to specify the structure or behavior of a system.
3. Models give us a template that guides us in constructing a system.
4. Models document the decisions we have made.

Modeling is not just for big systems. Even the software equivalent of a dog house can benefit from some modeling. However, it's definitely true that the larger and more complex the system, the more important modeling becomes, for one very simple reason:

We build models of complex systems because we cannot comprehend such a system in its entirety.

There are limits to the human ability to understand complexity. Through modeling, we narrow the problem we are studying by focusing on only one aspect at a time. This is essentially the approach of "divide-and-conquer" that Edsger Dijkstra spoke of years ago: Attack a hard problem by dividing it into a series of smaller problems that you can solve. Furthermore, through modeling, we amplify the human intellect. A model properly chosen can enable the modeler to work at higher levels of abstraction.

Saying that one ought to model does not necessarily make it so. In fact, a number of studies suggest that most software organizations do little if any formal modeling. Plot the use of modeling against the complexity of a project and you'll find that the simpler the project, the less likely it is that formal modeling will be used.

The operative word here is "formal." In reality, in even the simplest project, developers do some amount of modeling, albeit very informally. A developer might sketch out an idea on a blackboard or a scrap of paper to visualize a part of a system, or the team might use CRC cards to work through a scenario or the design of a mechanism. There's nothing wrong with any of these models. If it works, by all means use it. However, these informal models are often *ad hoc* and do not provide a common language that can easily be shared with others. Just as there exists a common language of blueprints for the construction industry, a common language for electrical engineering, and a common language

for mathematical modeling, so too can a development organization benefit by using a common language for software modeling.

Every project can benefit from some modeling. Even in the realm of disposable software, where it's sometimes more effective to throw away inadequate software because of the productivity offered by visual programming languages, modeling can help the development team better visualize the plan of their system and allow them to develop more rapidly by helping them build the right thing. The more complex your project, the more likely it is that you will fail or that you will build the wrong thing if you do no modeling at all. All interesting and useful systems have a natural tendency to become more complex over time. So, although you might think you don't need to model today, as your system evolves you will regret that decision, after it is too late.

Principles of Modeling

The use of modeling has a rich history in all the engineering disciplines. That experience suggests four basic principles of modeling. First,

The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.

In other words, choose your models well. The right models will brilliantly illuminate the most wicked development problems, offering insight that you simply could not gain otherwise; the wrong models will mislead you, causing you to focus on irrelevant issues.

Setting aside software for a moment, suppose you are trying to tackle a problem in quantum physics. Certain problems, such as the interaction of photons in space-time, are full of wonderfully hairy mathematics. Choose a different model and suddenly this inherent complexity becomes doable, if not exactly easy. In this field, this is precisely the value of Feynmann diagrams, which provide a graphical rendering of a very complex problem. Similarly, in a totally different domain, suppose you are constructing a new building and you are concerned about how it might behave in high winds. If you build a physical model and then subject it to wind tunnel tests, you might learn some interesting things, although materials in the small don't flex exactly as they do in the large. Hence, if you build a mathematical model and then subject it to simulations, you will learn some different things, and you will also probably be able to play with more new scenarios than if you were using a physical model. By rigorously and continuously testing your models, you'll end up with a far higher level of confidence that the system you have modeled will behave as you expect it to in the real world.

In software, the models you choose can greatly affect your world view. If you build a system through the eyes of a database developer, you will likely focus on entity-relationship models that push behavior into triggers and stored procedures. If you build a system through the eyes of a structured analyst, you will likely end up with models that are algorithmic-centric, with data flowing from process to process. If you build a system through the eyes of an object-oriented developer, you'll end up with a system whose architecture is centered around a sea of classes and the patterns of interaction that direct how those classes work together. Executable models can greatly help testing. Any of these approaches might be right for a given application and development culture, although experience suggests that the object-oriented view is superior in crafting resilient architectures, even for systems that might have a large database or computational element. That fact notwithstanding, the point is that each world view leads to a different kind of system, with different costs and benefits.

Second,

Every model may be expressed at different levels of precision.

If you are building a high rise, sometimes you need a 30,000-foot viewfor instance, to help your investors visualize its look and feel. Other times, you need to get down to the level of the studsfor instance, when there's a tricky pipe run or an unusual structural element.

The same is true with software models. Sometimes a quick and simple executable model of the user interface is exactly what you need; at other times you have to get down and dirty with the bits, such as when you are specifying cross-system interfaces or wrestling with networking bottlenecks. In any case, the best kinds of models are those that let you choose your degree of detail, depending on who is doing the viewing and why they need to view it. An analyst or an end user will want to focus on issues of what; a developer will want to focus on issues of how. Both of these stakeholders will want to visualize a system at different levels of detail at different times.

Third,

The best models are connected to reality.

A physical model of a building that doesn't respond in the same way as do real materials has only limited value; a mathematical model of an aircraft that assumes only ideal conditions and perfect manufacturing can mask some potentially fatal characteristics of the real aircraft. It's best to have models that have a clear connection to reality, and where that connection is weak, to know exactly how those models are divorced from the real world. All models simplify reality; the trick is to be sure that your simplifications don't mask any important details.

In software, the Achilles heel of structured analysis techniques is the fact that there is a basic disconnect between its analysis model and the system's design model. Failing to bridge this chasm causes the system as conceived and the system as built to diverge over time. In object-oriented systems, it is possible to connect all the nearly independent views of a system into one semantic whole.

Fourth,

No single model or view is sufficient. Every nontrivial system is best approached through a small set of nearly independent models with multiple viewpoints.

If you are constructing a building, there is no single set of blueprints that reveal all its details. At the very least, you'll need floor plans, elevations, electrical plans, heating plans, and plumbing plans. And within any kind of model, you need multiple views to capture the breadth of the system, such as blueprints of different floors.

The operative phrase here is "nearly independent." In this context, it means having models that can be built and studied separately but that are still interrelated. As in the case of a building, you can study electrical plans in isolation, but you can also see how they map to the floor plan and perhaps even their interaction with the routing of pipes in the plumbing plan.

The same is true of object-oriented software systems. To understand the architecture of such a system, you need several complementary and interlocking views: a use case view (exposing the requirements of the system), a design view (capturing the vocabulary of the problem space and the solution space), an interaction view (showing the interactions among the parts of the system and between the system and the environment), an implementation view (addressing the physical realization of the system), and a deployment view (focusing on system engineering issues). Each of these views may have structural, as well as behavioral, aspects. Together, these views represent the blueprints of software.

Depending on the nature of the system, some views may be more important than others. For example, in data-intensive systems, views addressing static design will dominate. In GUI-intensive systems, static and dynamic use case views are quite important. In hard real time systems, dynamic process views tend to be more important. Finally, in distributed systems, such as one finds in Web-intensive applications, implementation and deployment models are the most important.

Why are Models Useful?

Why are models important and useful for students?

Modeling Methodology for Physics Teachers (more info) (1998) offers one of the most compelling reasons to use models in an introductory geoscience classroom. "Scientific practice involves the construction, validation and application of scientific models, so science instruction should be designed to engage students in making and using models."

In addition:

- Models provide an environment for interactive student engagement. Evidence from science education research shows that significant learning gains are achieved when students participate in interactive engagement activities. Thus, it is important that the learning environment/activity created around a model provide an interactive engagement experience.
- Working with models can enhance systems thinking abilities.
- Models and model development are useful for helping students learn quantitative skills such as graphing, graphical analysis, and visualization; statistics; computational skills, mathematics, ...
- Many models allow one to perform sensitivity studies to assess how changes in key system variables alter the system's dynamic behavior. Such sensitivity studies can help one identify leverage points of a system to either help one affect a desire change with a minimum effort or to help estimate the risks or benefits associated with proposed or accidental changes in a system.
- Earth System Models such as those at Earth-System Models of Intermediate Complexity (more info) allow us to perform experiments related to the Earth System without altering and potentially harming the actual Earth. Many experiments, like understanding the future effects of atmospheric carbon dioxide increase, are taking place in the actual Earth System today but the results of these will not be know for 50 to 100 years. An Earth System model can run several such simulations using different assumptions in a matter of hours to days. The same is true for most models.
- The knowledge gained while using models and the understanding of model development and implementation are transferable to other disciplines related to the Earth system.

How to Use Models

In thinking about how to incorporate modeling activities into introductory geoscience courses, there are two important classes of considerations: **technical** and **pedagogical.**

Technical Considerations include:

- Acquiring the models or ideas in a useable form.
- Identification and use of the proper equipment for physical demonstration models.
- In the case of mathematical models, computers simulations of analogous systems, visualization models, or statistical models one must learn how to operate and manipulate the modeling environment or software.

These technical considerations are clarified under our discussions related to each specific type of model.

Pedagogical Considerations

There are several things to keep in mind when using or creating modeling activities for instruction.

- Keep the activity as interactive as possible. When you find that you're spending a majority of your time lecturing to the students about what to do or how things work, try to think of ways you can get them working through ideas in groups, lab, interactive lectures, etc.
- Including students in the development process and/or providing opportunities for them to experiment with the model or modify it can increase students' understanding of the model and its relationship to the physical world.
- Creating opportunities for students to analyze and comment on the models behavior increases their understanding of the relationships between different inputs and rates.
- Creating opportunities for students to validate the model, i.e. compare model predictions to observations, increases their understanding of its limits.
- Stress that models are not reality and that a model's purpose is to help bridge the gap between observations and the real world. An important reason to use a model is that you can perform experiments with models without harming the system of interest.
- Make sure that students think about the underlying assumptions of a model and the domain of applicability. Try to ask questions that can help check their understanding. For example, simple exponential growth assumes that the percent growth rate remains fixed and in real world systems it only applies for so long before the system becomes overstressed. Having students identify underlying assumptions of a model and their domain of applicability can help them gain an appreciation of what a model can and cannot do.
- Models can be used to introduce specific content. A model can introduce students to important terms as well as provide an environment to explore relevant processes.
- Models can be used to explore "What-if" scenarios. "What if Atmospheric CO_2 doubles?" is a common example for a climate model.
- Models can be used explore the sensitivity of a system to variations in its different components.