Software models in practice in student projects

Educators Symposium @ Models 2015 - Panel Summary

Moderator: Mira Balaban

Panelists: Dan Chiorean, Betty Cheng, Peter Clarke, Gunter Mussbacher, Tim Lethbridge

Participants: Omar Badreddin, Geri Georg, Grischa Liebel, Dave Stikkolorum, Emilio Pelozo, Arnon Sturm

1 Introduction – Panel Theme

While teaching modeling at Ben-Gurion University of the Negev over the years, we aim at achieving various goals:

- We teach modeling, intending that students will be the "messengers" of Model Driven Engineering (MDE) in the software world.
- We hope that students that were exposed to models will start using them when designing and developing their projects, and later on, will integrate and propagate models in industry projects.
- If we believe that MDE should prevail in software construction, we must guarantee that students start and continue using models during a project life cycle.

However, it seems that achieving these goals is challenging. To better understand that phenomena, the panel we had at the Modeling Educator Symposium was asked to address the following: To what extent do students that were taught a detailed modeling course continue using models in their studies and in particular in their capstone projects? The rationale was that if our own students do not use models, how can we expect the vision/message of MDE to spread around?

In order to address the issues mentioned above the panelists were asked to refer to the following aspects:

- Framework of **teaching** modeling
  - How do you teach modeling?
  - What difficulties do you encounter?
- Framework of **using** models
  - How are models used?
  - To what extent?
- **Perception** of models
  - How do students perceive modeling?
  - Their usefulness?
  - Their role – essential or a burden?
- **'Repair'**: How to increase the usage of models in software development?

- **Modeling vs. programming**: Differences and similarities in nature, perception, teaching, usage.

2 Opinions

Mira Balaban

The curriculum of the Software Engineering (SE) program at Ben-Gurion University, consists of the following: A modeling course (4th semester); an SE fundamentals course (5th semester) in which the students use models for communication purposes; an SE-project workshop (6th semester) in which the students perform a one-semester group project in a guided manner, and an SE graduation project, in which the students perform a year-long group project.

Although students receive extensive modeling education, we have some frustrating observations: Most students are not happy with the modeling aspect, modeling is mostly enforced, students do not recognize the important role of models in project design, and sometimes, models are even automatically created from the software (which bypasses the abstraction value of modeling).

Dan Chiorean

At Babes-Bolyai University modeling is taught and used in three courses: Software Engineering, DSLs, and CASE Tools. Using models in all life cycle development phases is a common feature of all of these three courses. The projects are realized by individuals or by teams. Insights from these courses consist of the following:

- From OO Languages/Technology courses students have the preconception that (1) models are just for "drawing"; and (2) modeling is time consuming and is a burden.
- From students projects – agile methodologies are appropriate – however, agile methodologies are not focused on modeling at all levels and in producing code by M2C transformation.
- The immaturity and in many cases the complexity of the provided tools are major drawbacks, as well.

One way of addressing the problems is to start with a kind of inverted curriculum, so first teach modeling by using a convincible examples and appropriate tools. For these examples it is important to highlight the advantages obtained by using the MDE paradigm against classical paradigms that do not support modeling.

Peter J. Clarke

At Florida International University, modeling education is being integrated in three undergraduate courses: software engineering, software testing, as well as the undergraduate capstone projects, and a graduate course in advanced software engineering and

software design. In software engineering the focus is on software development process, whereas in software design the focus is on meta-models for DSML, generating visual model editors using EMF/GMF. Various tools for this are also being used.

Some of the insights we have are the following:

- From a pedagogical point of view the approach has been very useful; from the student's point of view modeling is a burden.
- Students want to start writing code as soon as possible, based on their experience in previous courses.
- Students create models, then implement the application ignoring the models.
- A threat is that industry partners are pushing for specific tools and agile development e.g., Scrum.
- A greater threat is that some faculty buying into the push for agile, or are anti-modeling (Kingdom view).

To overcome the issues above we might want to:

- Keep on using models.
- Introduce modeling in early stages of the curriculum (e.g. introducing flowcharts in CS1).
- Deal with more complex software that will require good models for purposes of abstraction, validation, etc.
- Try to incorporate executable models in the curriculum.

Tim Lethbridge

At the University of Ottawa, we currently teach modeling using Umple as well as live examples on the board. Before Umple, students would tend to understand syntax, but misunderstand the semantics and pragmatics. The current educational approach allows students to sketch, then design, then generate systems.

Following our experience, students perceive models as useful as code, because we treat them as just abstract code (with diagrams as a view), as well as, being critical for success. To further increase the use of models, modeling and programming should be interchangeable; tools need to facilitate editing, validation, as well as generation of executable artifacts. Also, there is a need to incrementally adjust to how people work.

Gunter Mussbacher

The current software engineering program offered by the Electrical and Computer Engineering Department of McGill University consists of the following courses related to modeling: a model-based programming course (introduces key modeling concepts for structural and behavioral models and explains the semantics of models with the help of code generation; students are asked to build a system with code generated from structural and behavioral models), an introduction to software engineering course (focuses on software engineering fundamentals and includes a course project to be implemented on three platforms while reusing as much code as possible with the help of modeling),

a requirements engineering course (where models are used to communicate and clarify requirements), a software validation course (where models are used to drive testing efforts), and a two-term design project course in the final year (where students are implementing a system of their choice, hopefully with the help of models).

Some observations:

- Students do understand the merits of modeling, model-based analysis, and code generation. I have yet to meet a student who given a theoretical choice between writing code from scratch and generating code will choose the former.
- On the other hand, I have also met many students who skip over the "modeling part" and jump right into the implementation. The reason for this is most of the time related to tooling. A tool typically gets one chance. If a student perceives a modeling tool as getting in the student's way or not allowing the student to do what the student wants to do, the student will fall back on coding.
- The early model-based courses are recent additions to the software engineering program. Hence, it is not known yet whether the early focus on modeling has an impact on how often models are used for the final design project course.

Betty Cheng

At Michigan State University, when we teach advanced software engineering courses, we are trying to abstract the students from the code level. Having the students dealing with requirements rather than implementation allows them to focus on modeling, as eliciting the requirements requires capabilities other than programming.

In order to adhere to the given task of requirement specification, students must not move directly into coding, they need to do some abstraction through modeling.

Feedbacks from program alumni indicate that they appreciate modeling as they reach professional maturity.

3 Summary

The discussion revealed three approaches in modeling education:

1. Teaching modeling using an integrated environment that smoothly combines code and models, which clarifies the role of models in software construction.
2. Teaching modeling in a carefully designed process of requirement analysis and problem solving.
3. Teaching modeling using explicit formulation of models – syntax plus semantics, followed by teaching of their usage and design process.

Summarizing the participants' views over the panel topic, it seems that there is an agreement on the fact that students do not generally grasp the value of model abstraction. Rather, they tend to perceive models as a burden with a limited benefit, thus not using them as would have been expected. Nevertheless, most participants think that the appreciation for models grows when students graduate and experience software development in real industrial applications. The panelists also suggested to conduct a survey that checks these points among graduates of the different approaches.