МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2011
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2011
Proceedings of the Fortieth Jubilee Spring Conference
of the Union of Bulgarian Mathematicians
Borovetz, April 5–9, 2011

THE NOTION OF MODEL IN SOFTWARE ENGINEERING*

Miloslav Sredkov

The terms *model* and *modelling* are used so intensively in many disciplines that it is hard to narrow down their actual meanings. Even in Software Engineering the understanding of these terms depend very much on the context. We believe that this contributes to the discrepancy of the modelling approaches. In this article we point out some of the problems resulting from this and the importance of having a more proper definition along with set of instruments for it. We look at more general definitions of *model* from outside the bounds of Software Engineering. We go through the modelling of various by-products of software engineering processes. Finally we present our vision about how a common foundation can be used for our benefit.

1. Introduction. The terms *model* and *modelling* play central role in many disciplines of mathematics and social and natural sciences. Significant portion of the scientific effort is being spent in building and revising models as they are "one of the principal instruments of modern science" [1].

In Software Engineering products, processes and resources are being modelled [2], including problem domains, software architecture, software design, user interface, etc. With the advent of approaches like Model Driven Development and Model Driven Architecture [3] the meaning of *model* is being narrowed as more and more people are starting to associate it only with graphical diagrams in a language like UML. This tendency has been present for long enough for some authors to state that the usage of graphical software modelling languages is the traditional approach [4, 5].

In a wider context the term *model* stands for many things: in ordinary language its meanings include type of design, exemplar, and replica; in meta-logic a model of a theory is a set of entities or structure that satisfies the axioms of the theory; in social science the idea is reversed – the formal system is often called model. Attempts to classify types of models exist, but none of them is exhaustive [6]. Models serve many purposes and are of central importance in many scientific contexts [1, 6], but there are still significant lacunae in our understanding of them [1]. The use of inappropriate models or modelling often leads to catastrophic failures [7].

We believe that the lack of common understanding of the nature of modelling has considerable negative impact on the evolution and applications of the Software Engineering for the following reasons:

*This work was partially supported by the Bulgarian National Science Research Fund through contract 02-102/2009.

1. Lots of effort is wasted in understanding models and modelling techniques because of the lack of clear meaning of the accompanying terminology
2. The same also leads to distorted or even incorrect interpretations of them
3. Software Engineering theorist and practitioners are unable to apply research result related to scientific modelling even when it is related to their work
4. The lack of common foundation significantly increases the complexity of the processes and technologies which in turn leads to many negative effects

Other researchers have also recognised the importance of better understanding of the concepts and advocate more extensive teaching of modelling in the educational systems [8, 9].

Reunification of the interdisciplinary fundamentals of the notion of model can provide us with many benefits. These include easier understanding of models and modelling techniques, better synergy between software engineers, and applications of results related to scientific modelling. More importantly, it can show us opportunities to consolidate the wide diversity of modelling techniques and reduce complexity of engineering software.

2. Overview and General Meaning. There have been numerous attempts to provide a definition of what a model is in a way, that it is both wide enough and still useful. Definitions vary from general ones such as "representation of something else in the world" [10] to ones that are meaningful only in very specific contexts such as "a set of constants and functions whose members are the building blocks of a product line." [11]. Some authors emphasise the simplicity of models: "Models represent observed or hypothesized relationships of structure and function in simplified or abstract form." [12]. Rothenberg goes further, stating that "Modeling is a way of dealing with things or situations that are too 'costly' to deal with directly", thus leading to "To model, then, is to represent a particular referent cost-effectively for a particular purpose" [7]. It is unlikely that such definition will be widely accepted, as more often the idea that cost-effectiveness is a property of successful models is adopted while "there are good models and there are bad ones, and many in between" [6].

While there are exact formal definitions of *model* in the mathematical logic, it is often hard to apply them outside the pure mathematical context. We should state that even its meaning in Mathematical Modelling is quite informal: "A mathematical model is a mathematical construction that describes a class of phenomena external to mathematics" [13]. Furthermore, Frigg states that structures can not deal with the fact that models represent things from the world, claiming that pure structuralist view of models is a dead end [14].

Trying to define the terms is in itself an attempt to build a model of the wide domain where models (in some sense) are being used. In this case, as described by De Lange, there is a logical necessity of trade-offs between model attributes: "Any model attribute stands in an increase tradeoff with another iff one is a member of the set of attributes correlated with the intension of the model and the other a member of the set of attributes correlated with the extension of the model." [15]. Therefore, we have to accept that we will be dealing with a definition that is either of little use or too narrow.

In order to deal with these obstacles many researchers do not provide a general definition, but instead attempt to classify the kinds of models, and then go into details about each of the specific kinds. Here are several different classifications:

- Black [16] lists *scale*, *analog*, *mathematical*, and *theoretical* models

- Achinstein [17] defines *representational* (physical), *theoretical* (set of assumptions), and *imaginary* (intentionally not true) models
- Harré [18] divides models into:
  - *homeomorphs* – where the model and the prototype are of the same domain, further divided to *scale models* (micro and mega), *teliomophs* (abstracting, divided to reducing and idealising), and *metriomoph* (averaging)
  - *paramorps* – where the domains of the model and the prototype differ
  - *protomophs* (diagrams).
- Frigg [1] distinguishes models of phenomena: *scale*, *idealised*, *analogical*, *phenomological*; *models of data* and *models of theories*. The idealised models are non-exclusively divided into Aristotelian (stripping irrelevant properties) and Galileian (distorting objects towards perfection).
- Rothenberg distinguishes *descriptive models* (providing explanations) or *prescriptive models* (prescribing solutions to problems). [7]

However, none of the classifications is exhaustive [6].

3. Purpose and properties. Apostel defines modelling relationship as a quadruple $R(S, P, M, T)$, where "subject S takes, in view of the purpose P, the entity M as a model for the prototype T". This definition is very broad as it covers most usages of *model*, including those in natural language, in mathematical logic and in social science. However, it does not provide an answer to what is and what isn't a model. He has noted, that "we cannot hope to give one unique structural definition for models" and "if a unification is still to be possible, we should go back to our starting point: the function of models" [19].

This means that in order to dig further into the notion, we should first point out why models are used. The only common claim from the definitions so far, is that models are used for informative purposes. They are instruments for scientific cognition. They allows us to construct, interpret and test theories. They can be used to obtain empirical data, to provide explanations, to make predictions and projections. Modelling of (not yet existing) technologies and structures allows us to design and produce them [6, 7, 1]. As purpose is one of the main variables of each modelling relationship, it is necessary to always pay attention to it. Failure of a model does not necessarily mean that the model is bad, but could mean that it is not appropriate for the specific purpose [6].

Because it is hard to define an exhaustive typology of models, it is often easier to define the important (for the concrete purpose) properties of models instead. Such properties include: form, way to interact with them, accuracy, whether they are discrete or continuous, whether they are inductive or deductive. If we dig into more specific context, additional properties will unfold. E.g. computer simulations of physical phenomena may be characterised by the way they treat time as continuous-time, discrete-event, discrete-time, and hybrid [20]. Two important properties of each model are relevance and usefulness. Each model should be useful, allowing us to obtain information from it and should be relevant to its prototype, representing it close enough for the obtained information to be correct [6].

4. Artifacts and process outcomes. The importance of models in Computer Science and Software engineering is widely recognised, yet there lacks a recognition of all areas where different kinds of models are used. Models are well studied in relevance to Model Driven Development, Object Oriented Modelling, application of formal methods, etc. but still, many of the activities employing modelling are not recognised as such.

In fact, we claim that if we consider the more general notion of *model*, then most of the activities that software engineers perform are actually construction, manipulation or application of models.

In order to identify the entities related to Software Engineering, we will go through the processes of the "ISO/IEC 12207-2008 Systems and software engineering – Software life cycle processes" [21]. The standard provides common framework and terminology for software engineering processes. It contains processes, activities and tasks through the acquisition, supply, development, operation, maintenance and disposal of software products and services. It defines 43 processes and 237 process outcomes. 25 of the processes are "System Life Cycle Processes" divided into 2 "Agreement Processes", 5 "Organizational Project-Enabling Processes", 7 "Project Processes", and 11 "Technical Processes". The other 18 are "Software Specific Processes", which consist of 7 "Software Implementation Processes", 8 "Software Support Processes" and 3 "Software Reuse Processes".

In order to limit the scope to a manageable level, we are going to focus on the "Software Specific Processes" only. The "System Life Cycle Processes" also contains useful artifacts which are being modelled one way or another, but we will omit them since they are not unique to software engineering. In order to identify models, we will look at the outcomes of the selected processes. Process outcome is defined as "observable result of the successful achievement of the process purpose" which is a production of an artifact, a significant change in state or meeting of specified constraint [21]. The "Software Specific Processes" contain 98 outcomes:

- 16 of them describe some activity or action
- 24 describe continuous management or maintenance activity
- The other 58 outcomes consist of production of an artifact or define artifact indirectly.

The 58 artifacts consist of the following:

- (A): 17 significant artifacts, such as requirements, changes, architectural design, documentation, etc.
- (B): 12 strategies for implementation, integration, validation, etc.
- (C): 7 sets of criteria or standards, such as criteria for acceptance, standards for documentation, etc.
- (D): 7 results of technical or managerial decisions which impact the scope or form of other artifacts, such as what documentation to be produced, how domain models will be represented, etc.
- (E): 9 result summaries or statuses of other activities, such as results from testing, status of problems, etc.
- (F): 6 lists of problems, defects or other issues that need to be tracked until they are closed.

There is also one special item that may or may not be considered artifact – *software configuration*. It may be important to define the exact meaning of this term, but this problem is beyond the scope of this paper.

5. Models and the software engineer. The significant artifacts (A) are: software item, requirements, impact of the requirements, prioritisation of the requirements, changes, architectural design, external and internal interfaces of software items, detailed design, external interface of software units, software units, documentation, domain architecture, domain model, domain assets, reuse opportunities, reuse capability, and reuse

potentials. When a relatively large software system is being engineered, most of these artifacts are being used, that is, they are represented in some form, constructed, manipulated and processed. In fact, a typical non-trivial software system contains different kinds of software units (written in different languages, or with different role), which may need to be designed, represented, and documented in different ways. Different kinds of requirements, changes, etc. may also be required.

The artifacts from (B), (C), (D), (E) and (F) in simple projects may be represented in text documents, spreadsheets, wiki pages or other human readable form. In more complex projects however, strategies (B) are often created and manipulated using specific project management software, result summaries (E) need to be created or processed automatically, and lists of problems (F) are tracked using a dedicated bug (or ticket) tracking systems. Organising these artifacts and picking the right form to represent them is vital for software projects and can be really challenging.

Although few of the representations of these artifacts, such as diagrams of object oriented design, are explicitly called *models*, in the more general sense of the term most of them are such. This means, that a software engineering team may have work with hundreds of different kinds of models even in a single software project. In fact, the cases stated so far are just a fraction of the models that a team may have to deal with. Software engineering models, problem domain models, computational models, models for concurrency, etc. are of great importance for the software industry.

Even though some of the advancing technologies aim to simplify the life of the software engineer by unifying representations, thus reducing the number of modelling techniques, they can not compensate for the growing complexity and diversity of software projects. Now, in addition to the many programming languages and tools, a software engineer may have to work with concepts like *modelling language*, *meta-model*, *execution model*, *concurrency model*, *domain specific modelling* etc.

It is also important to state that this phenomenon is not new. Various objects have been modelled since the dawn of software engineering. The modelling approaches have undergone many changes (e.g. a program code description is more often modelled as a set of in-line comments rather than as separate documentation on paper), but they still have not reached a stable state. Because a suitable definition for *model* have not appeared naturally for so many years, we may conclude that it is unlikely that it will emerge without a significant amount of research towards it.

6. Towards a common foundation. If a software engineering team have to deal with hundreds of models, then the lack of clear comprehension of what a model is should be worrying. A common notion that will simplify understanding and instruments based on it applicable to a wide range of employed models need to be developed. Successful creation of such *modelling framework* may have a very positive effect on software engineering.

Model Driven Architecture and Model Driven Development try to address this issue. In the recent years related tools and techniques show tremendous progress. While these approaches are valuable steps into the right direction, they are not general enough to address all the issues of modelling. In fact, they are dealing with a very narrow meaning of *model*, thus both not approaching the whole problem and further increasing the discrepancy.

It is not certain whether a complete solution to the problem, e.g. such *modelling*

framework, exist at all. Our intuition shows, that potential solutions should be of pragmatic nature. They should be focused on the specific capabilities and limitations of the skills of the software engineers and the imperfections of the real software projects. Even if we are not able to provide a complete solution, an attempt to do so may reveal other ways to handle the diversity in Software Engineering.

7. Conclusion and future work. In this paper we presented that the lack of clear understanding of the notion of *model* in respect to Software Engineering is a significant problem that should be dealt with. More work is needed to pin-point the concrete problems and provide hints for solutions but before this constructing an actual modelling framework may not be a feasible goal. There are many directions from which the problem can be approached: philosophical, mathematical, logical, practical etc. Probably work should be done in many of them.

REFERENCES

[1] R. Frigg, S. Hartmann. Models in science. Stanford Encyclopedia of Philosophy, 2006.

[2] N. Maneva, Kr. Kraichev, Kr. Manev. Mathematical Models-Based Software Modernization. *Mathematica Balkanica*, 2010.

[3] A. G. Kleppe, J. Warmer, W. Bast. MDA explained: the model driven architecture: practice and promise. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[4] D. Brestovansky. Exploring Textual Modeling using the Umple Language. PhD thesis, Citeseer, 2008.

[5] T. Kühne. What is a Model. Language Engineering for Model-Driven Software Development, 4101:2006–04, 2005.

[6] U. Mäki. Models: Philosophical aspects. In Neil J. Smelser and Paul B. Baltes, editors, International Encyclopedia of the Social & Behavioral Sciences. Pergamon, Oxford, 2001, 9931–9937.

[7] J. Rothenberg and United States. Defense Advanced Research Projects Agency. The nature of modeling. Citeseer, 1989.

[8] M. Thomas. Modellbildung im Schulfach Informatik. Modellbildung, Computer und Mathematikunterricht. Hildesheim: Verlag Franzbecker, 2000, 39–46.

[9] M. Thomas, A.U.S.D.E.R.B. DEUTSCHLAND, P.D.R.A. SCHWILL, and P.D.R.S. FRIEDRICH. Informatische Modellbildung. Modellieren von Modellen als ein zentrales Element der Informatik für den allgemeinbildenden Schulunterricht. Potsdam, Diss, 2002.

[10] K. Dowding. There must be end to confusion: policy networks, intellectual fatigue, and the need for political science methods courses in British universities. *Political Studies*, **49** (2001), No 1, 89–105.

[11] D. Batory, J. N. Sarvela, A. Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 2004, 355–371.

[12] B. Winterhalder. Models. In Darwin and Archaeology: A Handbook of Key Concepts, chapter 12. Bergin & Garvey Westport, 2002, 201–224.

[13] B. Shternberg, M. Yerushalmy. Models of functions and models of situations: on the design of modeling-based learning environments. (Eds R. Lesh, H. M. Doerr) Beyond constructivism: models and modeling perspectives on mathematics problem solving, learning and technology, Lawrence Erlbaum Associates, 2003, 479–498.

[14] R. Frigg. Models and representation: Why structures are not enough. Measurement in Physics and Economics Discussion Paper Series, 2002.
[15] R. De Langhe. A general argument for tradeoffs in model building.
[16] M. Black. Models and metaphors: Studies in language and philosophy. Cornell University Press Ithaca, NY, 1962.
[17] P. Achinstein. Concepts of science. Johns Hopkins Press Baltimore, MD, 1968.
[18] R. Harré. The principles of scientific thinking. Macmillan, 1970.
[19] L. Apostel. Towards the formal study of models in the non-formal sciences. *Synthese*, **12** (1960), No 2, 125–161.
[20] D. A. Van Beek, J. E. Rooda. Languages and applications in hybrid modelling and simulation: Positioning of Chi. *Control Engineering Practice*, **8** (2000), No 1, 81.
[21] ISO/IEC/IEEE. 12207-2008 Systems and Software. *Engineering – Software Life Cycle Processes*, 2008.

Faculty of Mathematics and Informatics
Sofia University
5, J. Bourchier Blvd
1164 Sofia, Bulgaria
e-mail: miloslav@gmail.com

ПОНЯТИЕТО МОДЕЛ В СОФТУЕРНИТЕ ТЕХНОЛОГИИ

Милослав А. Средков

Понятията модел и моделиране се използват толкова интензивно в много дисциплини, че е трудно да им се придаде конкретно значение. Дори в Софтуерните технологии, разбирането на тези понятия силно зависи от контекста. Ние смятаме, че това допринася за неконсистентността между подходите за моделиране там. В тази статия посочваме някои от произтичащите проблеми, както и важността да има подходяща дефиниция, съчетана с подходящи инструменти. Преглеждаме по-общите дефиниции на модел отвъд границите на Софтуерните технологии. Минаваме през моделирането на различни вторични продукти на процесите от Софтуерните технологии. Накрая представяме нашата визия относно използването на такава обща основа за наша полза.