On the evolution of conceptual modeling

(draft version)

Roland Kaschek

School of Engineering and Advanced Technology

Massey University

Wellington, New Zealand

Abstract

Since the 1980s the need increased for overcoming idiosyncrasies of approaches to modeling in the various sub-disciplines of computing. The theoretical model of evolution is used in this paper for analyzing how computing and conceptual modeling have changed. It is concluded that computing has changed into a social phenomenon with a technical core and that therefore relying on (formal) model semantics as the sole tool for the discussion of conceptual modeling is no more adequate. A number of language games of computing is identified and the task set to describe these language games to the extent necessary for deciding whether or not they can serve as the foundation of computing.

Introduction

Conceptual modeling for software development, when considered in general, takes place as an iteration of a sequence of a problem exploration step followed by a solution implementation step, and a solution assessment step. Often many actors, so-called stakeholders are involved and need to have their say in figuring out what software system finally is specified, implemented, purchased, and used. It is thus not likely that a mono causal model will be good at sufficiently well explaining software development.

Evolution is a theoretical model, originating from biology, for the emergence of species that explains the observability of a direction of that emergence towards ever growing complexity of organisms [Gr98]. The important point about evolution theory is that it based on natural selection and mutation in general is considered as being sufficient for a scientific explanation of the natural history of species. Evolution theory thus can explain the emergence of order and complexity of life without making the assumption of a master mind being involved. I use the term evolution theory here metaphorically [RM98] for setting a context for understanding the process of emergence and change of conceptual modeling. The latter is (and has been) changing in consequence of the way independent actors impact each other's life while the considered actors do not necessarily intend to bring about any such changes to conceptual modeling. The focus of this paper thus is not the individual process of creating a conceptual model. Rather, this paper focuses on the changes of conceptualization of conceptual modeling throughout its history.

Conceptual modeling is the activity of creating conceptual models, i.e., models that describe problem structures in a way that still is relatively independent from the technology and strategy used to solve the

Dagstuhl Seminar Proceedings 08181
The Evolution of Conceptual Modeling
http://drops.dagstuhl.de/opus/volltexte/2008/1596

problem. In that sense conceptual modeling has a rather long tradition and is intimately connected to the genesis of mathematics. Models deliver their problem solution aid largely by providing a guiding metaphor [KT07]. For example, the guiding metaphor of mathematical models mainly has been that of equation. Newer such guiding metaphors that are closely related to the genesis and evolution of informatics are algorithm, relation, intelligent system, and dialog. The latter guiding metaphors of model construction have found their utilization in software engineering, data engineering, artificial intelligence, and web engineering.

Brodie, Mylopoulos and Schmidt [B*84] coined the term conceptual modeling in 1984. Back then they were responding to the increasing specialization of those involved in data engineering, artificial intelligence, or programming languages. They aimed at overcoming the unnecessary parts of the emerging divide between these sub-disciplines of computing. Since that time the specialization has increased rather than decreased and hardly anyone oversees the various sub-disciplines of computing to the extent necessary for deciding which of the differences found over these sub-disciplines are necessary and which are accidental. The feeling, however, remains that differences found between the various sub-disciplines of computing are more related to notation and basic vocabulary rather than to the used concepts, the ways to utilize the models, or the ways the models are constructed. Similarly it occurs such that that considered over time in a given sub-discipline of computing the differences that one finds are more related to notation, terminology, and implementation technology rather than the key model ingredients or modeling procedures.

Conceptual models

[A*05, p. 59] defines a “mathematical model of a physical law is a description of that law in the language of mathematics.” Similarly says [BS04, p. 49] that a “… mathematical model of some measurable aspect(s) of the real world is an equation or set of equations …” that “describes” known assertions and can be used to predict new assertions. Also [Kr06] in his definition of model focuses on a model being a substitute or proxy entity for what I call model original, a term I borrow from Stachowiak [73]. By successfully using a model one achieves having a command over the model original that was too difficult or expensive, undesirable, inconvenient, impossible or similar to be achieved without the model. It is common to the above definitions that the concept of model original they employ is some part or aspect of the “real world” and that this real world exists independently from its model. Certainly this reference to a real world not necessarily has to be included in a model concept.

With respect to defining what a model is one frequently finds that view of models as some kind of artifact that describes something (see, e.g. [Ac65, St73, Ba03]). This view is compatible with the attitude towards modeling as for example customary in the UML community that considers models as words or sentences encoded in a modeling language, i.e., UML. That view is of highest importance, as implicit in a description is one who describes and makes choices about what to describe, how to do so and what for. Models are thus made for a purpose, have an area of application, a particular way of suitable usage, and an intended user. I explore this idea in more detail below by analyzing Stachowiak’s general model theory.

Herbert Stachowiak has provided a huge body of work regarding models. His general model theory (GMT) in particular is documented in [St73, St83, St92]. The GMT has been reused lately by Petkoff, Ludewig, Hesse, and Broy & Rumpe in [Pe98, Lu02, He06, BR07]. See also [KT07] for further references regarding conceptual modeling. Stachowiak has characterized models in two ways: an ontological one (in which he says what a model is) and a pragmatic one (in which he says in how and what for conceptual models are used). His pragmatic characterization was received better by his audience, as listed above, than his ontological characterization. I restrict myself in this paper to his pragmatic characterization. Aspects of ontological characterization of model were, however, discussed in [KT07]. Stachowiak states that models have the **mapping property** (have an original), **truncation property** (the model lacks some of the predicates accredited to the original), and **pragmatic property** (the model use is only justified for particular model users, tools of investigation, period of time, and similar). It is constitutive for GMT that between model M and original O is defined what I call an **icomorphism** ι, i.e., a pair (γ , τ) of partial mappings $\gamma : O \rightarrow M$, $\tau : M \rightarrow O$ that I call **grounding** and **transfer** respectively. Stachowiak considers models and original as semiotic entities that actually are sets of predicates.

Stachowiak considers abstraction to play an important role in modeling. The abstraction relationship between model and original in Stachowiak's theory is specified solely by not including into the model some of the predicates accredited to the original. The grounding thus, where it is defined, is injective. The grounding γ only enables interpreting the chosen original's predicates in terms of the model. The transfer's task is interpreting the original's predicates in terms of the model. That in fact two such mappings are needed follows from a further important property of models that I call **extension property** i.e. that to models usually are accredited properties that are not accredited to the model originals. I thus summarize the GMT in stating that modeling is abstraction plus sense making. Abstraction is achieved by choosing appropriate original's predicates. Sense making is achieved by interpreting via a grounding the chosen original's predicates in terms of the model.

Kreyszig in [Kr06, p. 2, 6] identifies modeling as a three step procedure. First, obtain a model; second, analyze the model; and third, transfer any findings from the model to its original. The mappings grounding and transfer as introduced above just permit making these steps in an organized fashion.

The early historic development of mathematics is essentially a consequence of modeling for practical problem solving. Struik writes for example [St72, p. 34]: "The oriental mathematics emerged as a practical science for doing calendar calculations, managing harvest, organization of public buildings, and for simplifying the collection of taxes." For this paper I assume conceptual modeling starts out as early mathematics and unfolds with mathematics over millennia until in recent history computers were invented and revolutionized conceptual modeling. Certainly conceptual modeling will have additional sources that are ignored here such as philosophy, theology, and law.

Focusing on mathematics drivers of conceptual modeling can easily be identified as well-funded demand for problem solution, needs to organize teaching mathematics sensibly, and the need to find a sound base for mathematical research. It appears, however, that despite all efforts for organizing teaching and research appropriately mathematical modeling in the pre-computer era was more like an art rather than a controlled process.

Obvious dimensions of conceptual modeling are (1) the **model subject**, i.e., that which is modeled including its meta-modeling; (2) the **modeling procedure**, i.e., proceeding, role and life cycle models; (3) basic **conceptualizations of modeling** such as a art, engineering activity, business or otherwise; and (4) the **model handling technology**, such as manual, manually controlled, program controlled, computer algebra systems, and so forth. Obviously these dimension can serve as a conceptual framework for observing change in conceptual modeling.

The evolution of conceptual modeling

Functionally conceptual models are utterances regarding the model original and a number of different ways of the model to be related to its original has been identified [KT07]. Conceptual modeling in principle is affected from the technologization of the word that was diagnosed by Ong [On96]. In fact, at least since Kempelen's Chess Turk and later Babbage there have been substantial initiatives towards automation of human cognitive functions. From a historical point of view the most obvious change that happened to conceptual modeling is indeed its technologization. The invention of mechanical calculators and program controlled machines such as looms (for more detail see, e.g. Ifrah [If00]) had profound impact on how and what for conceptual models were used. Model handling nowadays is largely program controlled and model creation no longer is essentially a one-person-job. Like using conceptual models it has become a social activity.

The languages used for defining and handling models have continued to become ever more high-level. These languages are used to hide the technological base of model handling from the individuals involved in this. A stage has been reached in which a design notation such as the UML can be considered as a programming language. For that to be possible substantial investments into hardware, software, and education of staff were required. Conceptual modeling has become a business, i.e., the business of software development. The resources spent for it must pay off. That drives fundamental changes. The speed with which models can be developed and made operational has become an issue. Digitalization of media via computers not only helps new areas to emerge in which model subjects are found, such as computer games, health informatics, or e-government. The need to amortize the investments in technology and staff also pushes for new applications. Modeling procedures for mainstream problem classes in software development are well-documented.

Conceptual modeling nowadays mainly takes place during software development. Software development and with it conceptual modeling has turned into a social activity. At least six different actor roles can be identified: business (or application) expert, requirements engineer, developer, tester, maintainer, and project manager. It continues to be important that computers can be used for computing values of functions for given arguments. However, the way computers are conceptualized is changing. The fact that computers can be considered as meta-media becomes more and more important. A convergence is ongoing of media that once were clearly distinguished such as film, music, book, and game. One of the early triggers and indicators of that process of convergence is Bush [Bu45]. Note that for Bush the goal of that process was to aid human problem solvers. Turing has addressed the relationship between human thought and program controlled symbol manipulation [Tu50].

The traditional theoretical model of computation is the Turing table. That role, however, is challenged by Turing table's incapability to function as theoretical model for intentionally non-terminating procedures, dialog, parallel computation, and the evaluation of material predicates. A revised theoretical model of computation is thus required. It is the view of this paper that any such model needs to take into consideration that computation has turned into a social phenomenon.

One such way of recognizing this social nature of computation is to acknowledge the existence of the roles in software development as indicated above and have integrated theoretical models for each of these roles. Based on recognizing these roles six different language games relevant for software development can be identified easily: First, a cooperative language game in which application experts use software (and its documentation) for solving (business) problems. In this game the model semantics is constituted by business best practices and organization conventions. The models mainly serve for strategy building. Second, there is a refutative language game in which models serve as specification for an implementation that is sought for. Here model semantics in fact is formal. These two games are intermediated by a third and a fourth game, i.e., the coordination and inquiry game, respectively. In the coordination game project leaders make sure that appropriate specifications are worked out in a sensible amount of time and that these then are implemented appropriately. The fourth language game exactly is about obtaining the appropriate specifications. Not only is the model semantics different in these games while the model as a token, a structured pattern, or text is or maybe the same. Also the rules of the game are different. In the cooperative game first priority is given to getting a given job done with the available resources. In the refutative game, however, first priority is given to getting a correct implementation of a given specification. At an even higher level of intermediation the cooperative and the refutative language game are connected to each other by a testing game and a maintenance game.

It seems such that one can classify model construction and use roughly into the following phases:

1. Until 1819: individual manual model construction, individual manual model use.
2. 1820 – 1950: individual and manual model construction, collective mechanical manually controlled model use.
3. 1951 – 1968: individual program controlled model construction & closed group program controlled model use.
4. 1968 - 1990: closed group program controlled model construction and use.
5. Since 1980: interactive computing.[1]
6. Since 1990: open group program controlled model construction and used.

According to [If00, p. 127] the first automatic calculator that found large scale commercial use was the Arithometer of Charles-Xavier Thomas of Colmar. It was invented in 1820. Note that the prehistory of this device is quite long and, again according to Ifrah (p. 121), includes Wilhelm Schickard's Calculating Clock that was invented in 1623. Interestingly it seems to be such that Schickard, an astronomer by profession, did in no way think about using his device for performing calculation that were relevant to him professionally. Ifrah reports about a letter of Schickard to Kepler in which the Calculating Clock is

[1] I abuse here the begin of the object oriented age as indicated by Wegner and Goldin in [WG99].

discussed with a clear focus on how amazing it was to automate calculation rather than on how to use professionally the device that automated calculations. While remarkable progress was made in technologizing calculation it lasted until 1910 that Jay Randolph Monroe (see Ifrah, p. 140) got completely rid of manual interventions during multiplication and division. I date here the invention of program controlled model construction somewhat arbitrarily with 1951 the year of publication of Betty Holberton's "Sort-Merge Generator" an early form of compiler; see [MH81, p. 9]. By that time, as Murray Hopper reports, computers widely were seen as calculation devices and text processing was understood to be something completely different from calculation. I mark the end of the era of individual model construction by 1968, the year of the Garmisch Software Engineering conference. Winograd reports in [Wi …, p. …] that by ???? the dominating activity of computer scientists was text processing rather than computation. For more respective information see the preface of [Ka06]. Given the back then non-existing use of computers for processing pictures, music, and film it appears as a quite safe assumption that nowadays the subject of computing more adequately can be described as media processing rather than computation in the traditional sense.

This phase model highlights two dominant aspects of change affecting conceptual modeling, i.e. (1) its technologization, towards the capability of creating and using conceptual models under program control; and (2) its socialization towards establishing open communities as creators and users of models. These two concerns are linked together by the underlying process of industrialization of conceptual modeling.

Conceptual models as social constructs

An early source pointing out the social determination of knowledge and science is Ludwik Fleck [Fl35]. Using the example of syphilis Fleck shows how scientific ideas and facts depend on the community that uses them and how they are changed in the course of scientific progress. Further respective papers of Fleck are collected in [Fl83]. Presupposing that dependency of knowledge and fact on the respective community shows that computing suffers from a specific problem. That emerging discipline, which at its begin was determined by mathematicians, physicians, and engineers continued to attract and involve folks with different mindset and education. It is well-known that nowadays many involved in computing do not have any kind of appropriate formal education. I think that condition is not likely to change in the near future. The most obvious indicator for that change of folks involved in computing is the so-called software crisis. Guys with a mindset of management and control noticed that the level of control could not be achieved that appeared to them as necessary. An attempt was then made to make software development follow more strictly more detailed software process models. Also the size of projects was reduced and projects were sourced out for increasing the achievable level of control. Finally, with the invention of so-called agile techniques at least for technically less challenging projects the importance of software process models was reduced again.

Two fundamental and non-trivial observations that result from the social nature of models are that (1) experience is required for choosing the model most appropriately in a given situation; and (2) that models not necessarily have to be correct. In [KT07] it has been shown in detail that these two observations originate from the philosophy of science debate of model use in the natural sciences. It is

obvious that these observations apply to computing, as (1) respective experience is needed for knowing that a formal model of, say, a database application often will not be of much help in solving any of the problems for solution of which a company might have implemented that application in the first place; and (2) the cost for removing a given error from the software as being in use might be prohibitive and so workarounds might be used for the time being. Such errors would only be documented rather than removed.

Throughout the evolution of conceptual modeling folks of new kind came into computing. The usage of conceptual models has to be considered from a number of respective perspectives. Several of these were not considered as relevant in a time when mathematical models were created and used by the same person, when that model was essentially handled manually by the model creator, and when the model was not expected to be used interactively[2]. For example traditionally the single point of concern was getting right the so-called functional requirements of an application. Certainly it was and remains important that using a given software system the required functions can be computed. However, other qualities of a software system such as learnability, maintainability, memorizability, portability, reusability and others (see e.g. [G*04]) became important. It turns out that what I call (conceptual) usage model has become important. That model gives an account of how the software system should be used. While it has profound impact on the success with which a software system is going to be used it is relatively independent from the functions provided. Furthermore any formal semantics of that kind of model is likely to be not of much help in using the software system since (in many cases) its intended users are neither willing nor in a position to understand that semantics. It, by the way, turns out that the conceptual usage model not only is important for “high tech”. Rather it plays an important role in everyday things [No02], see also [No04].

Language games

If computing and, along with it, conceptual modeling has turned into a social phenomenon or activity and if formal semantics cannot be used as the sole foundation of conceptual modeling what then could possibly be used instead? Above I have already stated that conceptual models functionally are utterances by means of which one refers to one of the model’s originals. For finding a sensible theoretical foundation of conceptual modeling one therefore might want to look at explanations of how language and its use work. One of the approaches that might work well together with Stachowiak’s GMT is Wittgenstein’s language games [Wi53]. Wittgenstein points out that in learning natural language training is involved and that the purpose of that training is the acquisition of the vocabulary. Wittgenstein furthermore points out that this training is different from explaining. He uses the term language game for types of activity sequences the purpose of which is vocabulary acquisition. He uses the term language game also in a more general sense as the unit of language use that is interwoven into activity.

Wittgenstein has explicitly mentioned a number of language games. Among these are: describe or create an item based on a given description, invent a story, make a conjecture, state and check a

[2] The latter is, however, not entirely accurate since it is believed widely that some interactive models such as the Antikythera mechanism in astronomy were already used more than 2000 years ago [Ch06, F*06].

hypothesis. In computing we have identified six basic roles of agents: requirements engineer, business (or application) expert, (software) developer, tester, maintainer, and project manager. It appears as clear that what they do can be conceptualized as a language game. The consequence of this is obvious as well as lucky. One can have a formal semantics governing parts of the language game of developers while other concerns govern the other language games. That not necessarily excludes mathematical or logical tools from being used in these games. The description of these language games at the required level of detail is, however, beyond the scope of this paper.

Conceptual modeling completeness

In this paper I assume that the evolution of conceptual modeling continues to change it such that it becomes an ever more social concept and that for major parts of it very high level programming languages are available that resemble modeling and design languages such as the UML. Then computer applications happen to be media and implementing these thus must address the main characteristics of media. Since media are entities that intermediate actors each medium is characterized by a number of actor interfaces and the properties of the actor interaction channel provided. An actor may be considered as a stimulus-state-response relation. That actor's state can be used for specifying an actor's intentionality. The system concept thus still seems to be a conceptual framework expressive enough for the conceptual modeling of the future. An entity qualifies as a system (for more detail se e.g. [AN90]) if one reasonably can refer to it in terms of the following concepts:

- *Quality*; i.e., a particular way of responding to stimuli. Different responses may be generated to repeated stimuli.
- *Intentionality*; i.e. a theory conceptualizing the artifact's quality.
- *Consistency*; i.e., any structure of operational artifact parts that by exchanging and processing a so-called flow implement the artifact.
- *Constraint*; i.e. any set of conditions on the artifact's quality.

System stimuli often are called inputs and responses to them are called outputs. The operational artifact parts are usually called component. An interface of a system S is a component of S that is capable of directly exchanging a flow with a system S' not belonging to S. In [AN90] the term subsystem is used for distinguishing those system components that are considered as a system from those that are not considered that way.

Turing machines originally were a model for conceptualizing the quality of stateless systems. Relational; machines (see for example [Va98]) and persistent Turing machines (see for example [Go00, G*01]) are models for conceptualizing the quality of state bearing systems. Wegner [We97] has claimed the expressiveness of interactive computing be higher than the one of algorithms. As is customary he has associated the latter with Turing machines. Looking at Turing machines as predicate evaluators reveals that there is in fact something peculiar about them. While it is fine that they always and under all circumstances evaluate the predicate "1 > 2" to "false" there are other predicates that should be evaluated differently. Consider for example "I am taller than Jose". It depends on who the evaluator is and when they actually do the evaluation whether or not that predicate will be evaluated it to "true". Of course I have presupposed here that it is known who "Jose" is. The brief discussion of Fleck's work

above shows that one cannot help making such assumptions, as it rests on the decision of a community which phenomena are taken into account and how they are conceptualized. Computing not only has turned into a social phenomenon. It, moreover, depends on the world that it acknowledges, i.e. the computation device interacts with. In fact Goldin and Keil say [GK01, p. 809] "[w]hat is feasible in one environment is not so in others …" when they mention that an intrusion of the environment into computation has happened.

Following the general idea of [Or97] I call a language formal if the outcome of evaluating of each of its utterances is invariant against the actual evaluator, the evaluation conditions, and the utterance's subject area. I call then further a language material if evaluating its utterances may depend on the actual evaluator, the evaluation conditions, or the utterance's subject area. What happens then by means of the feedback incorporated into Turing machines by considering persistent Turing machines is that a material language becomes the subject of machine processing. There are at least two ways of looking at modern computing if one focuses on expressiveness. The first one is the language used and the second the mode of interaction with computational devices. While I have here focused on the language [We97, Go00, GK01, G*01] focus on the mode of interaction. In [Go00] it was shown that sequences of interaction with non-state bearing devices cannot exhaust the interaction sequences with state-bearing devices. That result was to be expected, as it was, in a piece of art, already anticipated, for example, in the 1993 film "Groundhog Day" directed by Harold Ramis. In that film the main character Phil (embodied by Bill Murray) comes into a situation in which he is stuck at Groundhog Day (i.e. February 2nd in Punxsatawney) and everyone but himself keeps no memory of that day, which happens to happen all over again. The film shows then how Phil makes use of that condition in educating himself in various forms of art, expertise, and knowledge which in the end makes him capable of winning the love of Rita (embodied by Andie McDowell). In this setting it is obvious that an agent (Phil) equipped with memory can take advantage of others (such as Rita) without memory of that day because Phil, presupposed to have all the time he wants, can check out any finite interaction sequence with any character he wants, learn about them, and use that knowledge for pursuing his goals. Note the refreshing discussion of relationships between literature and advanced technology simulating human cognitive functions in [Stro].

Certainly, presupposing more conventional views of the world, in which one can have knowledge of the world only by inquiring it, there is a connection between evaluating material phrases (such as material predicates, collections, and functions) and interaction. However, interaction of computational devices not necessarily implies use of such phrases since the interacting devices could be restricted to using formal language. It seems, for example, that Turing machines with oracle could be seen that way. I consider thus the language aspect as being more fundamental than the interaction aspect. I therefore consider the evolution of computing as taking a pragmatic turn, as what distinguishes material languages from formal ones is deixis (i.e. using indexing terms or phrases the resolution of which is actually determined by the conversation in which it is used), which in linguistics is discussed under language pragmatics. Additionally to material predicates also material collections (i.e. classes) have been added to the computing toolbox. For example the set "my_houses.set" that I define right now and that contains as element exactly the houses I own now unfortunately is and stays empty for all time

since sets (of the Zermelo-Fraenkl set theory) are immutable. However, the class “my_houses.cls” that I define right now and that likewise contains all the houses I own now has the chance of getting more substantial in the (unlikely) event that I mature sufficiently. Anyway, that class not necessarily stays empty for all time. Obviously I could consider that class as a sequence of sets and could consider material predicates as sequences of formal predicates. Considering associations between classes that way suggests that functions between sets cannot exhaust material functional associations between classes.

References

[A*05] Howard Anton, Irl Bivens, Stephen Davis. Calculus. John Wiley & Sons, Inc. 8th ed. 2005.

[Bu45] Vannevar Bush. As we may think. Atlantic Monthly 19, 11 (1945), pp. 101 – 108. The paper was reprinted in Noah Wardrip-Fruin, Nick Montfort. The new media reader. The MIT Press, 2003. pp. 37 – 47.

[BR07] Manfred Broy, Bernhard Rumpe. Modular hierarchic modeling as base of software and systems development (In German). Informatik Spektrum, vol. 30, no. 1. 2007. pp. 3 – 18.

[BS04] Roger Browne, Subhas Mukhopadhyay. Mathematics for engineers and technologists. Pearson Education New Zealand. 2nd ed. 2004.

[B*84] Michael Brodie, John Mylopoulos, and Joachim Schmidt. On conceptual modeling: perspectives from artificial intelligence, databases, and programming languages. Springer Verlag: New York, NY et al. 1984.

[Ch06] François Charette. High tech from ancient Greek. Nature, vol. 444 (2006), pp. 551 – 552.

[Fl35] Ludwik Fleck. Entstehung und Entwicklung einer wissenschaftlichen Tatsache. Benno Schwabe & Co. Basel, 1935. The book was re-published as: T. Trenn, R. Merton (eds.) The genesis and development of scientific fact. Chicago, 1979. See also: Suhrkamp Taschenbuch Wissenschaft 312. Suhrkamp Taschenbuch Verlag. Frankfurt, 1980.

[Fl83] Ludwik Fleck. Erfahrung und Tatsache. Suhrkamp Taschenbuch Wissenschaft 404. Suhrkamp Taschenbuch Verlag. Frankfurt, 1983.

[F*06] T. Freeth et al. Decoding the ancient Greek astronomical calculator known as the Antikythera Mechanism. Nature, vol. 444 (2006), pp. 587 – 591.

[Go00] Dina Goldin. Persistent Turing machines as a model of interactive computing. Proceeding FoIKS 2000. Springer Verlag.

[Gr98] Richard Gregory. The Oxford companion to the mind. Oxford University Press. Oxford, New York, 1998. Paperback edition of the 1987 edition.

[GK01] Dina Goldin, David Keil. Interaction, evolution, and intelligence. 0-7803-6657-3/01/$10 © 2001 IEEE.

[G*04] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. Software qualities and principles. Ch. 101 of [Tu04].

[G*04a] Dina Goldin, Scott Smolka, Paul Attie, Elaine Sonderegger. Turing machines, transition systems, and interaction. Information and computation, vol. 194, 2004. pp. 101 – 128.

[He06] Wolfgang Hesse. Models – Janus faces of software development – or: with Janus from the A- to the S class (In German). Heinrich Mayr, Ruth Breu (eds.). *Proceedings of Modellierung 2006, 22 – 24 March, Innsbruck*. GI Lecture Notes in Informatics, P82. Bonn. 2006. pp. 99 – 113.

[If00] George Ifrah. The universal history of numbers: the computer and the information revolution. The Harvill Press: London, UK. 2000.

[Ka06] Roland Kaschek (ed.). Intelligent assistant systems.

[Kr06] Erwin Kreyszig. Advanced engineering mathematics. John Wiley & Sons, Inc. 9th ed. 2006.

[KT07] Roland Kaschek, Bernhard Thalheim. Towards a theory of conceptual modeling. Manuscript, Palmerston North, New Zealand. 2007.

[Lu02] Jochen Ludewig. Models in software engineering – an introduction and critique (In German). In Martin Glinz, Günther Müller-Luschnat (eds.). *Proceedings of Modellierung 2002*, 25 – 27 March, Tutzing. GI Lecture Notes in Informatics, P-12. Bonn. 2002. pp. 7 – 22.

[MH81] Grace Murray Hopper. Keynote address to the 1978 ACM SIGPLAN 1978 conference on the history of programming languages. In Richard Wexelblat (ed.). History of programming languages. Academic Press. New York et al., 1981. pp. 7 – 20.

[No02] Donald Norman. The design of everyday things. Basic Books, 2002. 2nd edition of Donald Norman. The psychology of everyday things. Basic Books, 1988.

[No04] Donald Norman. Emotional design: why we love (or hate) everyday things. Basic Books, 2004.

[On96] Walter Ong. Orality and literacy: the technologizing of the word. Routledge. London and New York, 1996. 7th reprint of the 1982 edition.

[Or97] Erich Ortner. Methodenneutraler Fachentwurf. B. G. Teubner Verlagsgesellschaft, Leipzig, 1997.

[Pe98] Boris Petkoff. Knowledge management: from computer centered to user oriented communication technology (In German). Addison-Wesley: Bonn, Germany et al. 1998.

[RM98] Arnim Regenbogen, Uwe Meyer. Wörterbuch der philosophischen Begriffe. Felix Meiner Verlag. Hamburg, 1998.

[St72] Dirk Struik. Abriss der Geschichte der Mathematik. VEB Deutscher Verlag der Wissenschaften. Berlin. 1972.

[St73] Herbert Stachowiak. Allgemeine Modelltheorie. Vienna, New York. Springer Verlag, 1973.

[St83] Herbert Stachowiak. Erkenntnisstufen zum Neopragmatismus und zur Modelltheorie. In Herbert Stachowiak (ed.) Modelle – Konstruktionen der Wirklichkeit. Wilhelm Fink Verlag. München, 1983, pp. 87 - 146.

[St92] Herbert Stachowiak. Modell. In Helmut Seiffert, Gerard Radnitzky (eds.) Handlexikon zur Wissenschaftstheorie. Deutscher Taschenbuch Verlag. München, 1992. Reprint of the 1989 edition by Ehrenwirth.

[Stro] Ernst Strouhal. Eine Flexible Geschichte. Kempelens Türke: eine Schach-Metaphern-Maschine aus dem Spätbarock. Accessed on 2nd April 2008 from http://www.karlonline.org/402_5.htm.

[Tu50] Alan Turing. Computing machinery and intelligence. Mind 59, 236 (1950), pp. 433 – 460. The paper was reprinted in Noah Wardrip-Fruin, Nick Montfort. The new media reader. The MIT Press, 2003. pp. 50 – 64.

[Tu04] Allen Tucker (ed.). Computer science handbook. Chapman & Hall/CRC: Boca Raton, Fl. 2nd ed. 2004.

[Va98] Moshe Vardi. Computational model theory: an overview. Journal of the IGPL, vol. 6, no. 4, 1998, pp. 601 – 623.

[Wi53] Ludwig Wittgenstein. Philosophische Untersuchungen. In Ludwig Wittgenstein Werkausgabe Band 1. Suhrkamp Verlag. Frankfurt, 1999, pp. 231 – 485.

[We97] Peter Wegner. Why interaction is more powerful than algorithms. Communications of the ACM, vol. 40, no. 5, 1997, pp. 80 – 91.

[Wi…] Terry Winograd. …

[WG99] Peter Wegner, Dina Goldin. Interaction as a framework for modeling. In Peter Chen et al. (eds.). Conceptual modeling: current issues and future dimensions. Springer LNCS 1565, 1999.