# Theory and Practice of Software Development
## - Stages in a Debate -

Christiane Floyd

Department of Computer Science
University of Hamburg
Vogt-Kölln-Straße 30, 22527 Hamburg
e-mail: floyd@informatik.uni-hamburg.de

**Abstract.** Starting from the experience gained in organizing TAPSOFT'85, the paper discusses the place of formal methods in software development. It distinguishes two notions of theory: the mathematical science of computation and the treatment of computing as a human activity. An adequate software theory needs to take both theoretical perspectives into account. Therefore, the paper explores the borderline of formalization and human activity in several directions: concerning the role and scope of formalized procedures, the relation between formal models and situated use, the process of learning in software development and the ways computer programs become effective in use. Fundamental assumptions underlying formal methods and their relation to emancipatory approaches such as participatory design are discussed. The paper closes with calling for a dialogical framework for further pursuing these questions.

## 1 Starting a dialogue in 1985

As a co-organizer of the original TAPSOFT conference held at the Technical University of Berlin in March 1985, I have been invited to make a contribution to the tenth anniversary of the ensuing movement. So, let me start by congratulating all those who have participated in the TAPSOFT effort during the past decade, for having achieved a highly successful series of scientific events. And I hope, that it will continue in the future.

Assessing the success of my own involvement in TAPSOFT is a little more difficult. When Hartmut Ehrig originally approached me about co-organizing a conference on formal methods, he wanted me to bring in the practice perspective - the "P". Though we probably did not have the same notion of "P" in mind, I agreed and suggested to discuss the *relevance of formal methods* in software development. And he, in turn, agreed to that. Thus, we created a field of tension between two scientific communities, one focussing on formal approaches and their foundations, and the other reflecting the role of formal approaches in practice, and we set the stage for a controversial discussion. The resulting conference organization is explained in detail in the contribution by Ehrig and Mahr in this volume [EhM 95]. It comprised three parts: two colloquia and an advanced seminar.

However, the discussion we planned for did not actually come about. In the context of the individual colloquia, the members of each community discussed largely among themselves. In the plenary sessions, there were impressive talks from both sides, but no dialogue. My role as a hostess kept me from bringing in a strong position myself. In retrospect I believe, I may have been the only one to consider this dialogue an essential element of TAPSOFT. I also arranged an evening discussion on responsibility. This, I believe, was an important event for quite a few people who were able to voice and exchange their human concerns. The discussion there was very lively, but it took place outside the scientific program.

As soon as the conference organization moved from Berlin to Pisa, the outlook has changed. Since then, I have come to think of the TAPSOFT conferences simply as of a platform for promoting formal methods.

Now I am happy to see that Peter Mosses wishes to take up the discussion again. When inviting me, he encouraged me to say something about my work on software development and reality construction. He also expressed his hope that I might promote an exchange between the formal methods group and the system development group at the University of Aarhus, both very prominent in the international discussion in their respective fields of interest. Therefore, I will focus here on the limits and borderlines of formalization. I will not concentrate on the technical issues in programming, but on the place of formalization in design.

Looking back at TAPSOFT'85 once more, I would like to bring out two reasons for the lack of actual discussion. On one hand, it is a question of personal motivation and of an appropriate *style of interaction*. TAPSOFT was a very friendly gathering, but the conference took place at a time, when controversies on formalization (both oral and written) tended to be put forth in a hostile and arrogant manner on both sides. This experience may have made the participants unwilling to speak up and carry on an open dispute in a large group. If we want dialogue, we need to cultivate an attitude of mutual respect.

Even more important, however, in my opinion, was the absence of a *common level of reference* that would allow the two parties to engage in a fruitful exchange. I remember taking part in a panel discussion where I suggested that we needed a theory of the human processes involved in software development. This was met with strong rebuke by another panel member from the formalist community - he took it entirely for granted that "theory" must refer to mathematical theory and thus misunderstood completely what I tried to say.

Between 1985 and 1995, some basic elements of the theory I was looking for have been formulated. Therefore, I have now taken up the challenge to make a dialogical contribution to TAPSOFT'95.

In doing so, it seems natural to start from the position I presented shortly after TAPSOFT'85 at the conference on *Development and Use of Computer-based Systems and Tools - in the Context of Democratization of Work* held at the University of Aarhus in August 1985, leading to the publication of [BEK 87]. This conference was also a highly successful scientific event and thus there is another ten years' anniversary I wish to commemorate here.

The Aarhus 85 conference was like a scientific watershed in my life. On one hand, it gave me the opportunity to present a theoretical basis for my own work in software engineering. The positive reaction to this greatly encouraged me to delve into studying the epistemological foundations of software development. This became my predominant area of research for several years. On the other hand, it was one of the first events to make explicit that there was an emerging scientific community dealing with human and social issues in computing. Thus, the role of this conference in promoting encounters and forming alliances was similar to that of TAPSOFT, only for a different group.

At this conference, I also became acquainted with the work on *model monopoly* and *dialogue* by Stein Bråten [Brå 73], which has been very helpful to me ever since. In the context of the present paper, I would like to apply his ideas to the interaction between two theoretical paradigms dealing with software development. In 1985, the *mathematical science of computation* was already well under way, while only fragmentary attempts for understanding *computing as a human activity* had been published (cf. section 2).

Thus, the formalist community held what Bråten calls a model monopoly on software theory. If, in a situation of model monopoly, two partners engage in a dialogue, this cannot be symmetric, since one actor is *model-strong*, while the other is *model-weak*. The model-weak actor has to try to express his or her concerns in the terms proposed by the model-strong actor. This can only lead to further strengthening the position of the model-strong actor. This is well in keeping with my impressions from TAPSOFT and other attempts at discussion at that time. The formalist community wanted to change practice by introducing formal approaches. They did not see the need to understand practice in its own right. On the other side, the practice-oriented community seemed pre-occupied with fighting off formal approaches as though they were a threat. But they were not in a position to offer an alternative.

What is needed to resolve the model monopoly is to formulate an alternative position and allow the two to interact. The aim of the present paper is to sketch such a platform for dialogue about theory and practice of software development, as seen from my side.

Another prerequisite for dialogue is to clarify our mutual positions. I would like to illustrate this by reference to the specific debate about formal methods. The success story reported by Ehrig and Mahr in [EhM 95] is impressive. But it smoothes out the original controversy.

At the time of TAPSOFT, the idea put forth by the formalist community was that a complete and strictly formal (for example, algebraic) specification should be prepared beforehand as a basis for implementation. This was considered necessary, in keeping with the idea of providing formal semantics for the program.

I suppose that from the formalist point of view the main point of interest here is the use of formal concepts in dealing with a practical problem. But from the human activity point of view, a formalized procedure is implied, prescribing at what time and for what purposes these concepts are supposed to be worked with in software development projects. When and how this can or must be done, makes all the difference. In my argumentation, I never opposed formal concepts as such, but I spent a lot of energy objecting to what I still consider senseless manners of proceeding in using them.

Instead, I argued for partial formalization, for the flexible, semi-formal use of formal concepts [Flo 85]. Reading [EhM 95], it is not entirely clear to me any more, what the proponents now mean by using formal methods in software development. In particular, the success reported from the practical project relies on semi-formal use of formal approaches - exactly along the lines that I have advocated and practiced in 1985. The paper shows that many of the original claims associated with formal methods could not be fulfilled. Thus, the success reported rests on *restating more realistic claims* with respect to formal methods and on subsuming large parts of basic software engineering and programming languages research into the TAPSOFT movement.

So, what are we talking about? What was really the part contributed by strictly formal methods to the success reported? And why are the objections raised by critics like myself not quoted as having been confirmed by practical experience?

Once an alternative model is available, we have the conditions for a symmetric dialogue - at least in principle. In practice, conducting such a dialogue depends on the willingness of the participants. On creating and maintaining environments and on allocating resources for allowing it to take place. I believe that this task is ahead of the computer science community as a whole now and in the near future.

## 2 Two notions of theory about computing

In my paper *Outline of a Paradigm Change in Software Engineering* in [Flo 87], I contrast two perspectives on software development:

- The *product-oriented perspective* regards software as a product standing on its own, consisting of a set of programs and related defining texts.

- The *process-oriented perspective*, on the other hand, views software in connection with human learning, work and communication, taking place in an evolving world with changing needs.

The notion of "product" is used in a very general sense here. It refers to any tangible result of human activity, while the notion of "process" denotes the unfolding activity in the situation. Bringing out these complementary perspectives is helpful for understanding many spheres of human activity. For most contexts of interest in software development, the relevant products are *formal artifacts,* such as programs or defining documents. Processes are carried out by individuals working on their own or, more often, by groups of developers and users and other people responsible for embedding these artifacts in *human context.*

The two perspectives can be expressed concretely for concepts such as programs, quality and method. In any software development project both the product- and the process-oriented perspectives need to be adopted and reconciled. I criticize the established software engineering tradition for focussing almost exclusively on the product-oriented perspective and declaring a concern with the process-oriented perspective to be outside the realm of computer science. I suggest that a shift of emphasis is needed, giving greater priority to the process-oriented perspective.

Taking these two perspectives seriously, we are also lead to reexamine the very idea of "theory and practice of software development". It is no longer adequate to distinguish "formal theory" from "informal practice" to which this theory is applied, we are now faced with two different notions of theory and their respective relation to practice.

In what follows, I shall address three levels:

- software development as a (collection of) human process(es) giving rise to formal artifacts and their embedding in human contexts,
- the theoretical frameworks needed for dealing with these aspects, and their potential interleavement,
- the nature of the discipline concerned with software development in computer science, a question which belongs to the philosophy of science.

In investigating the relation of theory and practice, I will draw extensively on a very interesting book [CFR 93] on program verification, whose aim is to make a contribution to the philosophy of computer science. Colburn, one of the editors, shows in his introduction [CFR 93, p.7], how for several sciences the philosophy of that science centers around a basic reductionist question. For example in biology, the controversy between vitalism and mechanism deals with the question, whether or not life can be explained in terms of inorganic processes. Colburn suggests: *Can Computer Science be reduced to a branch of mathematics?* as the basic reductionist question for our discipline. The book contains an excellent selection of articles from the proponents of the mathematical paradigm on one hand, and from some of the most prominent critics on the other. I have the pleasure that [Flo 87] is also included.

The mathematical is traced back to John McCarthy. In his seminal paper *Towards a mathematical science of computation*, [McC 63], he asks "What are the entities with which computer science deals?" and gives the answer "These are problems, procedures, data spaces, programs representing procedures in particular programming languages, and computers". While some of these entities are expressed in terms that nowadays are no longer in use, the formal research programme initiated by McCarthy, and continued by Floyd, Hoare, Dijkstra and so many other great proponents stayed on the course that was laid out here.

It is interesting to look at the collection of entities (latin for "beings") a little closer. What kind of an entity is a problem? What is its ontological status? Does it have a way of existing on its own? Whose problem is it and who decides on the options for solutions? Surely, the problems considered here must have been posed by someone and passed on in the community. Similarly for procedures: can we consider a procedure as an entity without referring to the class of human activities it is meant to standardize? On the other hand, what kind of entity is a computer here? Can we treat it as a formal automaton with no concern for its technical counterpart? It is clear that the mathematical science proposed by McCarthy should treat these entities as formal objects with no reference to their human and technical context. And it does. But all of these entities, even the elusive problems, are *artifacts*, resulting from human activity and used within human activity.

So, the mathematical paradigm stands for the product-oriented theory perspective about software. In principle, no way of organizing human activity is associated with the mathematical paradigm. But in practice, many authors rely on recommending predefined processes such as top-down development. Thus product-formalization and process-formalization tend to be combined. I shall denote this combination as formal methods approach for the time being, but eventually argue for a separation of these two aspects.

In contrast, I would like to denote the process-oriented theory perspective by *computing: a human activity*, the title chosen by Peter Naur for his recent book [Nau 92], which contains a collection of his articles written in the course of the past three decades. It starts with "The Place of Programming in a World of Problems, Tools and People" [Nau 92, pp. 1-8], published in 1965. Here, we find some of McCarthy's entities again - but what a shift of emphasis. It is not a science about formal objects any more, but about the activity of programming considered in its context, and about problems seen in connection with people on one hand and with tools on the other. Naur applied this view both to the use of computers in human activity and to the activity of software development itself. Thus, amongst other issues, he is concerned with the place of formalization in human insight. He is also the most-featured author in Colburn et al., since he has published articles both with and as a critic of the mathematical paradigm.

The two notions of theory suggested here differ in many ways, and yet they are closely related:

- They are inherently complementary, one cannot be treated without implying the basis for the other. This is exemplified in the very titles. When arguing for a mathematical science, McCarthy needs to refer to the human activity "computation", and conversely, when denoting the human activity of interest, Naur calls on the nucleus of mathematical science, "computing".

- Their historical development is quite different. McCarthy's article started a research programme, the mathematical paradigm was taken up and pursued hotly by many researchers from the 1960s onwards and came to full bloom in the seventies and eighties. The resulting theory has a strong coherence, enabling the community to take a common scientific stand. By contrast, the theory about computing as a human activity was slow in starting. For a long time, it manifested itself in critical positions formulated by individuals in different contexts, and only started gaining momentum in the mid-eighties. There is no central theme here, but a network of individual and related views.

- The relation of these notions of theory to the practice of computing is different. While the formal theory of computing deals with an ideal practice, the human activity approach rests on actual practical experience in developing and using formal artifacts in human contexts. Thus, the nature of one is prescriptive and that of the other reflective.

- The foundations of these theories are different: one rests on formal logic and mathematics, while the other rests on approaches in the humanities for understanding human learning and communication, individual and cooperative work.

The gist of my contribution is that an adequate treatment of software development must comprise both theoretical perspectives. Thus, I take a stand on Colburn's reductionist question. In what follows, I shall not argue for intermingling these perspectives into one common super-theory, but for *creating a dialogical framework*, taking both perspectives into account. Such a framework has to allow for the formal properties of software on one hand and for the human context of their development and use on the other. I will refer to such a framework as *software theory*. This conversational use of the term theory is in keeping with the ideas proposed in Germany under the name "Theorie der Informatik" [CNPRSSS 92].

## 3 Basic questions for a software theory

In this and the following sections, I shall take you on something like a guided tour through the world of human activities associated with computing. We shall concern ourselves with the *stuff that software is made of,* in particular with the borderlines between the formal and the informal, the static and the dynamic, the controlled and the unfolding aspects of it.

In order to examine these borderlines, we need to address questions such as: How does software emerge? How is it embedded in human contexts? What can and should be formalized? How do we decide what to model? How do we relate the model to reality? How are operational models effective? How do we judge software quality? How should humans interact with computers? Let me start by pointing out that when thinking in this way about software we always imply assumptions on the nature of human thinking, on the being of formal objects and on desirable ways of handling human affairs. In doing so, whether or not we admit it, we draw on epistemology, ontology and ethics. Moreover, we also refer to ourselves and to our professional role. To the claims we make about computing in society. To the way we deal with power, hierarchy and control.

Some of the basic works addressing these questions from various angles have appeared since the mid-eighties.

Many of these contributions come from Scandinavian countries, including Naur's work which has already been mentioned. On the basis of a study of Scandinavian approaches [FMRSW 89] I have come to the conclusion that the focus of computer science in the Scandinavian countries has been different from the beginning. It is not primarily concerned with technical innovation and formal artifacts as such, but with their explicit orientation to human use. It is also characterized by cooperation between researchers working in different fields.

This is evidenced by Simula, for example, a language that had a profound influence on programming methodology, but at the same time addressed itself towards a particular class of applications. Simula was developed by Ole Johan Dahl and Kristen Nygaard, a member of the formalist community and an application-oriented scientist. As we know, Simula created the technical basis for object-orientation and for user-oriented modelling. Also, the connection between computing and work design was already studied in the early 1960s in Norway - much earlier than anywhere else.

The ideas and the technology developed by scientists in the generation of Naur, Dahl and Nygaard had a profound influence on the younger scientists in Scandinavia, as they were passed on through education. Most Scandinavians I know refer to their programming practice with Simula sooner or later, when they try to explain the way they think. The borderline between what is in and outside of computer science becomes quite different when taking this approach.

Kristen Nygaard was also the pioneer in studying the social nature of software development, pointing out *perspectivity* as fundamental in what we consider relevant for modelling and programming [Nyg 86]. And he is responsible for founding a whole school of "systems development" in Oslo and here in Aarhus which has made many important contributions to studying the human activity of computing. For example, Susanne Bødker gave an excellent theoretical treatment of human computer interface [Bød 87]. Pelle Ehn came up with basic guidelines for designing computer based artifacts for supporting human work [Ehn 88]. Bo Dahlbom and Lars Mathiassen treated computers in context in philosophical terms [DM 93]. And so on.

By now, these approaches have acquired a great significance in the international discussion. "A Scandinavian Challenge" was the subtitle chosen by [BEK 87], the book resulting from the Aarhus conference in 1985. This challenge has been taken up by many researchers elsewhere, since meaningful ways of applying information technology are now a dominant concern. For the rest of us working in this field, it is sometimes a bit strange to be considered part of the "Scandinavian school".

In the United States, the discussion centered along the philosophical assumptions embodied in computer science. While the early classic Dreyfus mainly focussed on artificial intelligence, Winograd and Flores set the stage with [WF 86], their well-known book on philosophical foundations for design. From an anthropological point of view, Lucy Suchman showed in [Suc 87] how to think about software use in terms of situated action.

My own project was to study the epistemological foundations of software development. In cooperation with a number of colleagues in Germany, I arranged a small interdisciplinary workshop held in September 1988 under the name of *Software Development and Reality Construction.* This suggestive phrase was used to indicate a range of philosophical questions pertaining to the practice of software development and use. The participants were scientists from several fields in computer science and from other disciplines, such as linguistics, philosophy, psychology and sociology. The resulting book [FZBKS 92] discusses the nature of computer science as a scientific discipline and the theoretical foundations and systemic practice required for human-oriented system design. It comprises a series of contributions by different authors, arranged so as to form a coherent whole.

As I pointed out in [Flo 92], there is a *Leitmotiv* underlying this whole line of research: How do we understand people in relation to computers? This comes up in different ways: Are human beings in their cognitive faculties similar to computers? Can computers, in principle, be likened to human beings? How should computers be allowed to interfere with human affairs?

Computer science has inherited fundamental views equating humans with computers. Even in formulating his basic theoretical concepts, Turing explained the functioning of his universal machine by likening it to a man proceeding according to rules. The overall

social context of the emerging computer science was taylorism as a way of rationalizing and optimizing human work by rules, so that groups performing work would behave like machines. A very interesting analysis of the interplay of the theoretical basis of computer science and the socio-cultural developments in the 1920s and 1930s is given in [Hei 93].

While equating humans and computers is explicit in artificial intelligence, it is also implied by many approaches in software engineering and human-computer interface, whenever we rely solely on rule-based human thinking and predictable functional behaviour. There is a direct connection between these questions and formalization, since what can be formalized can also be mechanized. Formalizing and automating human activity is inherent in computing.

To characterize the specific way in which formalization occurs in computing, I shall use the term *operational form*. By this, I mean an abstract characterization of the informational basis and the functional content of defined human activity. Note that this terminology avoids reducing human activity to operational form. It merely aims at characterizing that aspect of human activity that is amenable to being formalized and automated. The practice of computing, in particular software development, always involves bringing out and automating operational form. This presumes routine in human activity and a modelling process in which this routine is explicitly defined. It rests on characterizing the objects we work with by informational attributes and our dealing with them as manipulation of these attributes.

A program executed on a computer can then be seen as *auto-operational form*. What is of basic interest to the theory I have in mind, is how operational form, and in particular, auto-operational form, relates to situated human activity.

My aim in the following sections is to sketch different orientations for research in a dialogical framework drawing on both the formalist and the human activity oriented theoretical perspectives. I will support my arguments by some of the positions taken by different authors mentioned in this section. Rather than discussing their individual contributions one by one, I would like to group them according to the issues involved.

## 4 The role and scope of formalized procedures

Computing invariably has to do with identifying and standardizing discrete operational steps and grouping them into complex formalized procedures, which give rise to algorithmic structures to be executed on a computer.

Thus, the basic way of thinking in our discipline involves several instances:

- an instance to *set up rules*: a human, who makes or identifies rules (for example the software developer complying with the wishes of the customer, the method developer prescribing how to proceed in software development and so on),
- an instance to *impose rules*, often a non-human agent (the computer controlling work processes or technical systems), and
- an instance to *carry out rules*, either human or technical.

It is difficult to write about these instances without equating people and computers, since we have to choose between "which" and "who". This seemingly linguistic triviality reflects the fact that the mutual "rights" of these instances depend entirely on our views of their interaction and thus on decisions taken in systems design. There is, however, little concern in computer science with how to embed formalized procedures in human activity.

Imposing automated formalized procedures leads to basic questions, since the computer brings about a new quality. While in general, the rules imposed on human activity have to be sufficiently clear and unambiguous for humans to follow, here they must be formulated in machine-interpretable terms. While humans interpret rules as they apply to the needs of the specific situation, programs always operate according to their predefined model. And while humans tend to associate rules with exceptions, computers do not.

As was first pointed out by Lehman in [Leh 80], automating a sphere of human activity is not a problem that can be specified in formal terms, thus the need to distinguish between "specification-based" S-programs and "embedded" E-programs. One important borderline to explore in software theory is the scope and the limits of process formalization and to develop criteria for design choices here. This is connected with the identification and the connection between the individual operation steps to be automated, with the possibility for humans to interfere and with the organization of computer-supported work. We find these questions arising both in the process of software development, and in concepts for software application.

Thus the instance setting up the rules must have a mental picture of the class of possible situations and a sufficiently rich understanding to allow for any potentially relevant activity at any time. There are two basic options for embedding formalized procedures in the richness of human situations. One is to rely on a formal model of the class of use situations, and to automate a set of rules for how to proceed according to this model. The other is to leave the use context open and to offer a repertoire of resources to use in self-organizing work. In this way, I shall differentiate between "centralized control" and "situated coordination" of cooperative work.

The choice between these options arises at many levels in computing. For example: how do we view the embedding of computers in work processes? The design of human-computer interfaces? The automation of the software development process with CASE tools? Or the application of formal methods in software development?

Because of their basic importance for computers in connection with work design, many authors have addressed these questions. For example, I have implicitly referred to Lucy Suchman's discussion in [Suc 87] just now: rather than as rule systems to be followed, she suggests that plans could be seen as *resources* to be relied on. Related is Reinhard Keil-Slawik's design guideline: *avoid all unnecessary sequencing* of predefined work steps in [K-S 92]. The common theme here is that formalized procedures should not be imposed, but be available for flexible use in cooperative work.

Closely connected are the perspectives and metaphors used in design, ranging from *machine*, to *tool*, *media* and so on. An excellent overview has been given by Susanne Maaß and Horst Oberquelle [MO 92]. They show, how different perspectives embody views on how humans should interact with computers. Examples of specific metaphors include the *tools and materials metaphor* first elaborated by Reinhard Budde and Heinz

Züllighoven in [BZ 92], and the *subject-oriented approach to information systems* put forth by Markku Nurminen [Nur 92]. I suggest, we should investigate how formal approaches can be combined with using such metaphors.

## 5 Formal models in situated use

Since all operational form rests on reductionist models of entities carrying informational attributes, another borderline to explore is the relation of formal models to reality. This involves various levels: How do we decide which aspects are relevant for being included into a model? How do our modelling concepts shape how we think about the world? What kind of being do we ascribe to modelled entities? What claims do we associate with models? How are different models connected? How do models become operational in use?

The underlying assumptions of modelling can be expressed with reference to various philosophical schools dealing with ontology and epistemology. This is particularly relevant in connection with global data models in enterprises. It also has direct bearings to how computer based systems are used to control technical and social processes.

Heinz Klein and Kalle Lyytinen address in [KL 92] the *social process of model building*, in particular data modelling in organizations as one of negotiation. Data models appear like laws, codifying the informational work basis in a specific manner. Modelling involves making choices. It also involves the power and authorization for making these choices. Thus, it is *essential who participates* in the processes leading to formal models.

Naur gives a careful account of what he calls the mystique of formal models [Nau 92, pp. 468-479]. He addresses specifically the relation between what is to be modelled, the *modellee*, the *concepts used for modelling* and the *modelling process*.

Starting from a different angle, Brian Cantwell Smith discusses *the relevance of formal models for use* in his thoughtful paper on the limits of correctness in computers [Smi 85]. He contrasts formal correctness (i.e. logical consistency) with suitability or adequacy in the situation. Since formal approaches do not help us with suitability, the connection between the model and its use remains unresolved. Only humans can take the responsibility for bridging the gap.

A basic distinction made by several authors is between modelling *structure and behaviour as such*, and modelling *with a view to human use*. In the former case we adopt the stand that attributes for formal entities can be described in a context-free manner, while in the latter, the entities to be modelled are perceived in the context of the work to be performed with them. If we want to use models, like plans, as resources in situated activity, this requires an understanding of their relation to use.

Therefore, Michaela Reisin understands a *reference theory* in software development as centering around the *use-meaning* of software. The use-meaning must be anticipated in cooperative design processes involving developers and users [Rei 92]. Similarly, Budde and Züllighoven regard programs as *things to use* and formulate their object-oriented concepts in terms of how things are used in work processes [BZ 92]. Use-oriented modelling rests on understanding the human activity to be supported.

We can identify various semantic dimensions of modelling in software development. This requires a holistic approach, relying on different notions of semantics. That is, we need to concern ourselves with *individuals forming personal meaning*, with *groups relying on conventional meaning* in their respective technical language, and with the relation of these two *to the formal meaning* specified for the attributes and operations in formal models and implemented in programs. Since embedded programs dominate in practice, we are faced with the emergence of insights on software issues on the part of the people involved as constitutive for the formal artifacts to be developed.

We can also distinguish different ways in which auto-operational form becomes effective in use. Is it supportive or controlling? Is it informative or instructive? Is it simulative or does it directly affect reality? I don't know a good classification here, but we need to work in this direction, as we populate the world with auto-operational entities of various kinds. Perhaps we need something akin to the speech-act theory in the philosophy of language: an operational model-effect theory that would provide us with conceptual categories for discussing how computers interfere with human activity in situated use.

In these fields, I see important avenues for joint research combining the formalist and the human activity perspective.

## 6 Formal methods as resources in software development

Naur's radical statement: *There is no right method for theory building* refers to an understanding of "method" as a formalized procedure. Instead, he argues in [Nau 92, pp. 37-48], we should master a repertoire of concepts for modelling to be used as needed in the situation.

This is quite similar to Goguen's argumentation in [Gog 92]. He deals with the idea of error-free programming as treated by the "Dijkstra School". He criticizes the tendency to apply formal methods in a rigid top-down hierarchical manner and argues for using them in flexible non-ideological ways. He also considers bugs in programs as interesting and important in themselves: they define the boundary between what is understood and what is not. Since errors in programming are a major reason for advocating formal methods, it is illuminating how this topic is addressed by different authors.

In formal approaches, errors are defined as deviances between the program and its specification. By contrast, Don Knuth has reported on his own errors in developing TEX in [Knu 92]. The most amazing thing to me is his classification of errors., He analyzes his errors in terms of how he saw them in relation to what would have been desirable. Thus, his error categories range from "blunder, botch" to "quest for quality". There is no way of expressing all of these errors in terms such as "right" and "wrong" with respect to a formal specification. Rather, *errors refer to mismatches between actual and desirable decisions taken during programming*, where no frame of reference is available for judging what is right and wrong.

The connection between errors and formalization is a subtle one. On one hand, formal approaches provide us with conceptual means that enable suitable ways of abstraction and avoid certain categories of errors altogether. Thus, they greatly improve the logical consistency of our programs. On the other hand, they introduce new sources of blunders.

And they give us no hint about quality. Whether or not we make suitable decisions with respect to the context. Whether we tackle the right problem, so to speak.

The formalist notion of errors is based on the idea of considering software development as the construction of a defined product, it is not compatible with the idea of a learning process in design. Here, errors have the constructive role of learning events, a point which is made by several authors in [FZBKS 92], in particular by [K-S 92], as well as by [WF 86] and by many others. In [Flo 87], I argue that errors need to be understood in terms of the human processes where they originate.

This leads to the question on how formalization is embedded in other modes of thinking and being. [Sie 92] gives hints by showing how formal statements need to be communicated in informal ways. This too is a point stressed by Naur repeatedly. In [Bur 92], Rod Burstall gives a fascinating account of the states of mind associated with computing. He concludes that we need to free ourselves from "being entrapped by a limited perspective based on desire for control and exclusive reliance on conceptual thought". Admittedly, this danger of single-mindedness is inherent in whatever approach to system development we adopt. But strictly formal methods introduce yet another level of expertise, control and potential alienation that we have to deal with.

There is by now a wealth of literature dealing with human processes in software development. Basically, it is a question of how insight is to be reached. The dissatisfaction with the original phase models formalizing the software process has given rise to the movement of prototyping and evolutionary development, where the basic idea is understanding and supporting the communication involved with an orientation to feedback and constructive ways of viewing human errors.

Adopting such a view leads us to examine the following issues:

- What is the place of formalization in contexts of multiperspectivity and evolving insight?
- When should we formalize, how and why?
- Is the nature of program verification one of using formal proofs or one of context-oriented argumentation?
- How can we use formal concepts as modelling tools?
- How can we adapt methods cooperatively rather than following rules?

I am passionate in these questions, since they reflect my own area of research. Allow me therefore, to point to [Flo 87] once more and also to our work on STEPS which is a methodical framework accommodating for various approaches. In particular, you can also take a formal approach in STEPS, if you find ways for incorporating this in an overall evolutionary process ([FRS 89], see also [K-S 92]).

In the meantime, I have also gained extensive experience in working with emancipatory approaches in software development. With participatory design, for example, with democratic work organization and with self-organization. It remains an open question to me,

how formal methods could be used as resources in this context. While software developers can and should learn to master formal methods by better education, users cannot. We have to find ways how formalization can be used without interfering in the human process of shaping technology for human use.

## 7 An invitation to continued dialogue

In this paper, I have named some of the issues of interest in treating computing as a human activity, but I have not treated them in depth. There are several reasons for this. One is the scope and time available. If you want to delve into these issues I have raised, be sure to look at some of the books I have suggested. Another reason is the context. I am not primarily addressing the human-activity community here, but those of the formalist community who are interested in cooperating on these questions in a dialogical way across boundaries. Note that in this paper I have not drawn on any philosophical school for founding my arguments.

Essentially, I argue that the stuff that software is made of, is a web of human views. Software does not only embody facts amenable to logical analysis, it also *reifies decisions on how human affairs should be handled*, resulting as compromises in processes of learning, communication and negotiation. This is often soft stuff, as the name software suggests, the validity of our formal models depending on subjective judgements. Inspite of its name, we are *not always* engaged in *computing* in our profession.

Therefore, I plead for a serious concern with theories from the humanities as an integral part of computer science education and for admitting such theories as basis for foundation in computer science research. Furthermore, I propose to create conditions in universities that would enable students to gain experience in communicative software practice during their education and to understand the interplay between the formal and the human activity dimensions of software development.

In 1985, I was engaged in building up a research group at the Technical University of Berlin. Technical and formal aspects of software development were considered there against the background of the human processes giving rise to them. Thus, the students in our department were exposed to formal methods as taught by Ehrig's group on one hand and to reflections on communicative practice in our group on the other hand.

I cannot agree to the overall harmonious interpretation of TAPSOFT given by Ehrig and Mahr. As seen from my side, it stands for conflict as well. TAPSOFT has marked the beginning of a profound change at my former home university with a strong shift of emphasis in the direction of formal methods. Eventually this became so forceful that the values and belief systems inherent in formal methods left no room for other views. I am not blaming this on any individual, it was a qualitative shift in culture. Based on my experience in the years following TAPSOFT, I have come to conclude that formalization issues are not just topics of academic dispute. Only at the surface are we talking about the merits and the limits of formal methods. Underneath, human convictions and scientific power struggle are at play. Decisions must be taken relating to curricula and learning forms. To positions to be created, to criteria for filling them and to money to be allocated for research. There are battles to be fought here, and they are hard, since they involve us.

Within a few years, the tensions became so strong that my willingness for continuing the dialogue came to an end. My continued personal friendship with Hartmut Ehrig was not sufficient as a basis for maintaining a dialogical milieu. So, inspite of my remaining loyalty to the department, I left Berlin and rebuilt my existence in Hamburg. As far as I am concerned, my move was one of professional emancipation.

I remember TAPSOFT'85 as a rare and valuable example for scientific cooperation across boundaries of theoretical perspectives and associated values and belief systems. And I consider my mission here at TAPSOFT'95 to formulate an invitation for joint research. Let us strive for a new dialogue on equal terms, while taking both perspectives seriously. Let us engage in common projects, aiming at bringing out and reconciling the different points of view.

As for maintaining a dialogical milieu: Though we have failed in the long run in Berlin, I wish good luck to our hosts for intensifying their dialogue here in Aarhus.

But dialogue is not confined to taking place within one university site. It is a question of forming alliances and networks of individuals between people from different communities sharing the same concerns.

## Acknowledgements

I would like to thank Cliff Jones for sharing with me at TAPSOFT'85 his experience in putting VDM into practice, and to commemorate with pleasure the contributions made by Rod Burstall and Joseph Goguen to Software Development and Reality Construction. I am very grateful to Dirk Siefkes for a continued stream of discussion throughout the past decade on the issues raised here. The present paper draws to some extent on the work of Uli Piepenburg on "Softwareentwicklung als Semantikerstellung" and that of Guido Gryczan on "Situierte Koordination" to be published in their doctoral theses this year.

## References

[BEK 87] Bjerknes, G., Ehn, P., and Kyng, M. (eds.): Computers and Democracy. A Scandinavian Challenge. Avebury, Aldershot, UK, 1987.

[Bur 92] Burstall, R.M.: Computing: Yet Another Reality Construction. In [FZBKS 92]: 45-51.

[Bød 87] Bødker, S. (1987). Through the Interface. A Human Activity Approach to User Interface Design. Aarhus University, Aarhus, 1987.

[BZ 92] Budde, R., Züllighoven, H.: Software Tools in a Programming Workshop. In [FZBKS 92]: 252-268.

[Brå 73] Bråten, S.: Model monopoly and communication: Systems theoretical notes on democratization. Acta Sociologica, 16(2): 98-107.

[CFR 93] Colburn, T. R., Fetzer, J. H., and Rankin T. L. (eds.): Program Verification. Kluwer Academic Publishers, Dordrecht/Boston/London, 1993.

[CNPRSSS 92] Coy, W., Nake, F., Pflüger, J.-M., Rolf, A., Seetzen, J., Siefkes, D., Stransfeld, R. (eds.): Sichtweisen der Informatik. Vieweg-Verlag, Braunschweig/Wiesbaden, 1992.

[DM 93] Dahlbom, B., Mathiassen, L.: Computers In Context - The Philosophy and Practice of Systems Design. Blackwell Publishers. Cambridge, Massachusetts, 1993.

[EhM 95] Ehrig, H., Mahr, B.: A Decade of TAPSOFT: Aspects of Progress and Prospects in Theory and Practice of Software Development. This volume.

[Ehn 88] Ehn, P.: Work-oriented Design of Computer Artifacts. Almquist and Wiksell International, Stockholm, 1988.

[Flo 85] Floyd, C.: On the Relevance of Formal Methods to Software Development. In Springer LNCS 186 (1986): 1-11.

[Flo 87] Floyd, C.: Outline of a Paradigm Change in Software Engineering. In [BEK 87]: 191-210.

[Flo 92] Floyd, C.: Human Questions in Computer Science. In [FZBKS 92]: 15-27.

[FMRSW 89] Floyd, C., Mehl, W.-M., Reisin, F.-M., Schmidt, G., and Wolf, G.: Out of Scandinavia: Alternative approaches to software design and system development. Human-Computer Interaction, 4(4): 253-349.

[FRS 89] Floyd, C., Reisin, F.-M., Schmidt, G.: STEPS to Software Development with Users. In: Ghezzi, C. and McDermid, J.A. (eds.): ESEC'89, Springer LNCS 387: 48-64.

[FZBKS 92] Floyd, C., Züllighoven, H., Budde, R., Keil-Slawik, R. (eds.): Software Development and Reality Construction. Springer-Verlag, Berlin, Heidelberg, New York, Tokio, 1992.

[Gog 92] Goguen, J.: The denial of Error. In [FZBKS 92]: 193-202.

[Hei 93] Heintz, B.: Die Herrschaft der Regel. Zur Grundlagengeschichte des Computers. Campus Verlag, Frankfurt/New York, 1993.

[KL 92] Klein, H. K., Lyytinen, K.: Towards a New Understanding of Data Modelling. In [FZBKS 92]: 203-219.

[Knu 92] Knuth, D.E.: Learning from our Errors. In [FZBKS 92]: 28-30.

[K-S 92] Keil-Slawik, R.: Artifacts in Software Design. In [FZBKS 92]: 168-188.

[Leh 80] Lehmann, M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 86(9): 1060-1076.

[McC 63] Towards a Mathematical Science of Computation. In [CFR 93]: 35-56.

[MO 92] Maaß, S., Oberquelle, H.: Perspectives and Metaphors for Human-Computer Interaction. In [FZBKS 92]: 233-251.

[Nau 92] Naur, P.: Computing: A Human Activity. ACM-Press, New York, 1992.

[Nur 92] Nurminen, M. I.: A Subject-Oriented Approach to Information Systems. In [FZBKS 92]: 302-311.

[Nyg 86] Nygaard, K.: Program development as social activity. In Kugler, H. G. (ed.): Information Processing 86 - Proceedings of the IFIP 10th World Computer Congress. North-Holland, Amsterdam, 189-198.

[Rei 92] Reisin, F.-M.: Anticipating Reality Construction. In [FZBKS 92]: 312-325.

[Sie 92] Siefkes, D.: How to Communicate Proofs or Programs. In [FZBKS 92]: 140-154.

[Smi 85] Smith, B.C.: Limits of Correctness in Computers. Reprinted in [CFR 92]: 275-293.

[Suc 87] Suchman, L.A.: Plans and Situated Actions - The Problem of Human-Machine Communication. Cambridge University Press, Cambridge, UK, 1987.

[WF 86] Winograd, T., Flores, F.: Understanding Computers and Cognition - A new Foundation for Design. Ablex, Norwood, NJ, 1986.