In: Floyd, C., Züllighoven, H., Budde, R., Keil-Slawik, R. (eds.): *Software Development and Reality Construction*. Berlin: Springer, pp. 86-100, 1992.

Software Development as Reality Construction

Christiane Floyd

Reality = Community

/v. Foerster 73/

1. Introduction

I would like to present a view of software development as an insight-building process in terms of multiperspectivity, self-organization and dialogue, drawing on epistemological ideas that have emerged from the discourse in Rational Constructivism.

I have come to this view in the course of my recent research on epistemological foundations of software development[1], which was motivated by many years of preoccupation with software development methods in my research, teaching and practical work. It was during this period, first in industry and since 1978 at the Technical University of Berlin, that I began to question the validity of the established models of thought in software engineering as the sole foundation for our work as computer scientists. I gradually became convinced that we need to arrive at a sufficiently rich understanding of software development if we want to facilitate it with methods in a meaningful way.

My doubts apply, in particular, to the following basic assumptions of the discipline: its view of software development as the *production* of program systems on the basis of fixed requirements; the separation of production from use and maintenance; the division of production into linear phases; the almost exclusive use of intermediate results in the form of documents; the view of methods as rules laying down standardized working procedures to be followed without reference to the situation in hand or the specific groups of people involved; and the one-sided emphasis on formalization at the expense of communication, learning and evolution. The resulting critique of software engineering has been elaborated in a number of papers [2].

I will gladly concede that I know of scarcely any author today who still accepts these assumptions without reservations. They are looked on rather as ideals that can only be

---

1 This research was funded by a grant from the Stiftung Volkswagenwerk enabling me to spend my sabbatical term in Palo Alto from September 1987 to March 1988.

2 cf. /Floyd 81/, /Floyd, Keil 83/, /Floyd 84/, /Floyd 85a/ and /Floyd 87/

approximated in practice. There have, however, as yet, been few efforts to develop conceptual alternatives to the established tradition of thought in software engineering.

One of my reasons for questioning the validity of this tradition were the glaring contradictions between what it postulates and the reality of software projects in both industrial and academic settings, despite the fact that many of these projects were ostensibly conducted along traditional lines. I by no means wish to contend that the production view of software development is irrelevant. But it does seem to me to hold only in part – for specific, well-defined partial goals – failing to do justice to software development as a whole.

Another reason for my doubts about the validity of the established view was its failure to take into account the quest for quality[3] in software development. Ultimately, it neglects to provide any sort of foundation for human-oriented system design.

It seems to me a richer view is needed here. And it was this realization that induced me to seek for suitable epistemological foundations for software development. Such foundations must help us to understand specific, communally sustained, coordinated processes of cognition in which different domains of reality meet. In these processes, abstract — and at the same time highly complex — results are obtained, and the emergent technical reality of the software is interwoven with the social reality of its production and use. These processes take place against a background of social conflicts, changing needs and limited resources and lead to insights regarding the desired software and its use.

The soundest alternative thought model available, to my mind, is Peter Naur's view of programming as "theory building"[4], which has been very influential in shaping my own ideas. However, Naur says little about what theory building consists in. And he does not account for the interpersonal nature of communal theory building.

As I see it, software development is, first and foremost, a specific instance of *design*. By design I understand the creative process in the course of which the problem as a whole is grasped, and an appropriate solution worked out and fitted into human contexts of meaning. In Naur's words: "Software development is an activity of overall design with an experimental attitude" (/Naur 74/).

To establish foundations for design, I draw here on epistemological insights that have emerged from the constructivist discourse, notably on the work of Heinz von Foerster and

---

3 A term coined by Don Knuth in his reflections on the errors he made in the development of the TEX system (/Knuth in this volume/).

4 cf. /Naur 84/ and /Floyd 1.1 this volume/

Gordon Pask[5]. Their ideas are, I feel, particularly conducive to understanding the process of software development as design.

In the following section, I begin by outlining the postulates of the established software engineering tradition, subsuming them under the general term *production view* of software development. My aim is to show that these postulates are of a perspective nature. They were *invented* from a particular viewpoint to highlight a particular facet, necessarily obscuring others. Which brings us straight to a key insight of constructivism.

In section 3, I introduce some important elements of constructivist thinking and relate them to our present subject. I refer specifically to the school of thought labelled Radical Constructivism. Of interest to us are the insights it provides concerning the emergence of knowledge in different areas. Our task will then be to explain, in the light of these insights, the specific types of cognitive processes that are important for software development.

In section 4, I attempt to elaborate a suitable concept of design for software development. Design is not primarily tied to predefined goals, but is governed by the quest for quality.

In section 5, I then go on to unfold the design space. It consists of the interlaced domains of reality – application, methods and means of implementation – that are *constructed* during design. Unlike the production view, there is no assumption here of a phase-specific, temporal transition from one domain of reality to another; we are dealing rather with a vibrant structure of ever new and ever finer distinctions, "dancing", as it were, in time.

In section 6, the cognitive processes taking place in design are characterized as a web of decisions linking together the domains of reality important for design. The suitability of a design decision is determined through its evaluation. Where feedback is permitted from the evaluation to the design process, we have *closure*, the results of design again forming the basis for its further development. Successful design is marked by a stabilization of the web of design decisions through revisions.

What we have said so far also applies to design processes carried out by individuals. In section 7, though, we go on to look at communal design implemented in terms of dialogically organized cooperation. This allows for a conscious dialogical orientation of design in which the "I and Thou" of software development is acknowledged in the basic relations "I develop software with you" and "for you" and. This orientation directly incorporates the element of responsibility in our technical work.

---

5 cf. /von Foerster, Floyd/ this volume

In the final section, I examine the way this view affects training, project organization as well as the development of methods and tools for software development.

What emerges, in general terms, is a view of design as consisting of interlocking, living processes that are sustained by us, that may atrophy, degenerate or unfold. Their unfolding presupposes both sufficient autonomy of design and the ability and willingness on the part of those involved to engage in multiperspective reflection.

2. Software Development as Production – a View and Its Limits

In this section, I wish to outline what are to me the essential issues and show how I set about tackling them. A practicable approach to epistemological inquiry in our domain has been shown by Winograd and Flores[6]. The established view, the authors argue, only appears to be self-evident to us as long as we remain within the rationalisitic tradition, which is characterized by them in terms of its postulates and underlying assumptions. (Actually, we are not only concerned here with the rationalistic tradition as a theory about our way of thinking, but also with the realistic tradition as a conception of reality.) This tradition, however, like any other, involves a certain "blindness" by obstructing our view to its underlying assumptions.

Winograd and Flores translate the notion of rationalistic tradition into more concrete terms by deriving from it assumptions relating to different subject domains. Related specifically to the domain of software development, it justifes the following assumptions:

- There is a given reality "out there" which we come across during software development. By analyzing the facts of this reality, we obtain requirements for the software.
- The essential task of the software developer is — starting from the problem defined in that reality — to find a correct solution in the form of a program system.
- It is possible to separate the production of software from its use. Software engineering is concerned with the production of software on the basis of fixed requirements.
- Software production is based on models representing reality. Models should map reality precisely.

---

6 in /Winograd, Flores 86/

- The whole process is largely independent of individuals. For one and the same problem, different developers should arrive at the same results. The developers should be interchangeable.

- Communication should be restricted and regulated via fixed interfaces. The division of labour can be worked out on an ad hoc basis. Subject to technical feasibility, any desired parts of the production process can be automated.

- The developer's responsibility covers — only — proper construction of the product in accordance with the requirements specification. Any ethical considerations that go beyond this are quite separate from the technical aspects of the work.

The view of software development reflected in these assumptions has been instrumental in bringing about impressive advances in programming methodology, in generating controllable models for project execution, and in promoting the development of tools on this basis. It allows us to understand important aspects of software development prior to initiation of the development process, to subsequently assess more or less completed projects - or to "fake" a rational design process as suggested by Parnas[7] .

It fails, though, to offer any help in understanding the software development processes actually going on in a given situation — processes relating to the emergence of insights into the functionality, implementation and usability of programs. Indeed, the production view is misleading, suggesting as it does that we can (must) proceed from fixed requirements and can derive a program system from these (ideally, in accordance with fixed rules).

The production view highlights one important facet of software development, eclipsing others. In my opinion, it obstructs our view of design. The pervasive nature of design and thus the key role occupied by it is also recognized by other authors. Winograd and Flores, for instance, have given their above-cited book the subtitle "A New Foundation for Design".

Questioning the rationalistic tradition necessarily involves examining other epistemological approaches. I shall confine my attention here to constructivist ideas. They fit in perfectly with the facet I have chosen to focus on: *software development as design*.

3. Entering into the Constructivist Discourse

Studying constructivist approaches was no easy matter for me. They call for a radical process of rethinking which I can only accomplish step by step. In addition, the available primary

---

7 cf. /Parnas, Clement 1985/

literature on the subject is heterogeneous and chiefly concerned with domains outside my own particular sphere of interest. This is due to the fact that the key insights have been derived from a variety of sources, including biology and developmental psychology, and subsequently applied to the social sphere. But retracing this process is not our concern here. Nor are we able, within the present context, to explore just how far these insights can be applied to other spheres.

Radical Constructivism as a school of thought has grown out of Cybernetics and is closely related to chaos theory. What we are dealing with here is the *emergence* of phenomena, in our case with cognition as the emergence of insights. Since there is no one overall, clearly elaborated constructivist position, but rather a variety of related approaches — differing quite considerably in detail — that blend into a common mind (cf. /Bateson 79/), one can justifiably speak of a constructivist "discourse" [8]. The various authors are involved in the ongoing scientific discussion within their own particular disciplines, but they go beyond the "what" of their respective subject to look at the "how" of the emerging insights, discovering as they do related patterns within the different disciplines.

It would be asking far too much to expect me to give a proper introduction to constructivism within the present context. Nevertheless, I shall attempt to outline at least its essential fundamentals in order to be able to refer back to them later on.

According to constructivist views, our cognitive faculties are ultimately rooted in the biological nature of the human being and his co-evolution with all other living beings. Bateson, an important pioneer of constructivism, postulates the essential unity of mind and evolution (/Bateson 79/). Maturana considers cognition to be inherently tied up with life (Maturana, Varela 80/).

Mind, in this sense, not only characterizes the individual human being, it is also encountered in other living systems. Mind is also invariably related to something. Thus, mind characterizes the way communities interact with respect to common concerns. For example, a group of people interacting in a project when developing software might exhibit mind. Here, deeper insights emerge by careful contrasting and coordination of individual contributions.

In constructivism, a distinction is made between epistemology and ontology (/v. Glasersfeld 87/) in a subtle way. Constructivism teaches us that we construct what we know; it makes no mention here of being. In constructivist thinking, cognition is not concerned with images mapping a given reality; instead, we construct knowledge in such a way as to make it fit our

---

[8] This is reflected in the German title of an excellent survey on Radical Constructivism, cf. /Schmidt 87/

purposes. Von Glasersfeld suggests to talk about the *viability* of our knowledge rather than about truth or falsehood. This notion seems quite natural in connection with design.

An essential element here is the introduction of the *observer*. Cognition is invariably tied to an observer (cf. /von Foerster 84/ and /Maturana, Varela 80/). Observers can only perceive what they are in a position to perceive. Communication between observers takes place in consensual domains that are mutually accessible.

Also tied up with this is the concept of perspectives. A *perspective* is the totality of assumptions about relevant aspects of a specific subject domain from a common viewpoint. Perspectives are not necessarily tied to individuals. People adopt different perspectives at different times. Between two or more individuals, common perspectives are formed (/Pask 76/, see also /Bråten 78/).

Perspectivity necessarily entails *blindness*. I cannot see what I cannot see from my perspective. It is impossible to eliminate this blindness. An important prerequisite for the emergence of deeper insights is self-reference (see below) and the interaction (crossing) of perspectives.

Constitutive elements of cognition are *distinctions* and *indications* (/Spencer Brown 69/). Complex cognitive processes consist of webs of interlocking distinctions, each relating to different perspectives, and their recompositions into a coherent whole (/Pask 76/).

A key concept here is *self-reference*. While in logic it results in paradoxes, it is fundamental to an understanding of living entities. Self-reference requires operational closure. All this means, in basic terms, is that the results of an operation are themselves elements of its domain of definition. This supplies the conditions required for the recursive application of operations.

In operationally closed and energetically open systems, the system's behaviour is determined by the recursive coupling of operations, with several levels of consideration interacting. System behaviour is stabilized by reference to eigen-values that give rise to system-specific eigen-behaviour. This results in self-organization leading to the emergence of higher orders in systems. This is the essence of the "Order from Chaos" principle formulated by v. Foerster (/v. Foerster 60/ in /v. Foerster 84/).

Living systems are characterized by the fact that, during the course of their existence, they continuously reproduce their own organization ("autopoiesis" is the term used in /Maturana, Varela, Uribe 74/). Autopoiesis takes place in a medium. The autopoietic system and the medium condition one another. I avoid giving here an assessment of how this concept can be

usefully applied also to social systems, as the current discussion about this point is strongly controversial. For example, rather than regarding software projects as "autopoietic beings" using systemic notions as Joseph Goguen does[9], I will confine myself to considering closure at the level of processes.

Self-reference also plays a major part in the emergence of insights. Perspective blindness can be overcome by self-reference. Once I see that I am blind, I can see again. Self-reference is also amenable to mathematical treatment (Varela 75/, /Varela 87/).

An essential and repeatedly emphasized aspect of constructivism is that ethics can never be divorced from the consideration of cognition and action (cf. /v. Foerster 73, /v. Glasersfeld 87/, /Maturana, Varela 80/). This is ensured not by explicitly laying down norms about what to do, but rather by viewing cognition and action as being sustained by us as individuals. This leads to v. Foerster's ethical imperative: *Act always so as to increase the number of choices*. In my opinion, this can be applied directly as a guideline for designing software development projects as well as computer-supported systems.

According to /v. Foerster 73/, acknowledging others involves making a decision. It causes us to emerge from our monologue and enter into a dialogue. Dialogue means adopting the other's perspective. Closure, then, occurs through the other. I see myself through the eyes of the other.

This step towards living in dialogue[10] is of crucial importance in implementing constructivist ideas in our dealings with others. It leads to the view "Reality = Community" (/v. Foerster 73/) selected as an epigraph for this contribution. According to /Bråten 88/, our cognitive faculties as a whole are geared to dialogue.

Well, that was, of necessity, a rather rudimentary and fragmentary crash course in constructivism. I have attempted to convey something of the pleasure I experienced as I gradually came to grasp what it is all about in my dialogue with Heinz von Foerster. I feel at home with these ideas. They tie in with my own everyday experience both privately and professionally, and they seem to me to open up far-reaching prospects for a desirable design of our life in the community.

My subject proper begins with our recognition that the view of software development as production is an invention. Thus, the prevailing view of the subject of the software engineering discipline turns out to be constructed reality. It is useful for understanding certain

---

9 cf. Goguen Part 5 this volume

10 The term is used here in the sense of Buber (/Buber 84/).

aspects of software development, but of no use for others. It is therefore important to contrast it with other views.

4. Software Development as Design

To begin with, I should clarify what I mean here by design. It will not do to simply adopt an existing definition of design. What I shall endeavour to do instead is to construct a concept of design to fit my fundamental concerns. In other words, I would like you to join me in drawing a number of distinctions in order to arrive at a useful characterization of design in software development.

By design, we mean a specific type of cognitive process that is geared to producing feasible and desirable results within a particular domain[11] . The domains in question may differ widely. We normally only speak of design when there are concerns we wish to fulfil, limited resources at our disposal, and different implementation options open to us.

Design in software development is of a specific nature and subject to special conditions. Software is characterized by an interplay of several unusual features, relating to both the nature of the product (cf. /Parnas 86/ and /Keil-Slawik in this volume/) and its embedding in human contexts of meaning (cf. /Ehn 88/, /Reisin in this volume/). Software exhibits an extreme degree of complexity, thus calling for equally complex construction processes. It consists of a uniform, abstract building material, is therefore plastic and, in principle, of unlimited revisability (cf. /Coy in this volume/, Budde, Züllighoven in this volume/). It must be machine-processable, i.e. complete down to the last detail, consistent and formally free from error. It is not amenable to sensory perception and can therefore, in the last analysis, only be evaluated once in use. It creates social contexts for human actions, which are shaped by the technical properties of the product.

Design thus links different worlds: the social world of the application in question, the technical world of the means of implementation, and the formal world of methods and concepts.

In order to elucidate the meaning of design as we understand it here in all its essential richness, we have to consider the way this concept is used in different contexts. As a rule, we distinguish between design and implementation. We design something. The result is subsequently implemented. Occasionally, we also call the result of a design process "the design", focussing here on the external features of an object. That is too narrow a view,

---

11 In German, the two terms "Entwurf" and "Gestaltung" are used to approximate the meaning.

though. We also speak of design in a broader context when we have in mind the overall process of organizational and technical system development.

Design should be understood here in a processual sense: the results of design are incorporated into the design processes from which they were obtained. Design relates not only to external features, but also to the functionality of the program system under development and its embedding in human contexts of action. Design also includes the provision of suitable tools and methods for the specific software development situation and for project organization.

I should now like to illuminate design in software development from a number of different angles. I apologize in advance for my rather abstract wording. This is due to the fact that I begin by characterizing design in terms applicable to both individual and communal instances of design. I do not make a distinction between the two until later on, in section 7.

5. The Design Space as an Unfolding of Interlaced Domains of Reality

The production view implies looking on software development as a sequence of phases, ideally to be run through linearly. Each phase is concerned with a specific object: first of all, analyzing the requirements of the application; then, on the basis of this, elaborating a specification of the future system, defining what is to be done without prescribing how the system will work; and, finally, deriving the program from this.

It is here that the domains of reality relevant to design implicitly enter into the picture:

- the world of the applications whose concerns are relevant to software development and from which we derive requirements for the software,
- the world of the means of implementation — in our case technical information systems including existing software,
- the world of methods and concepts which we use in the same way as maps to guide us in linking concerns with means of implementation.

The phase model prescribes just one path through these worlds: that starting from fixed requirements and following predefined methods to arrive at the implementation on a given system. Ideally, this path should be followed only once with respect to the overall product software.

The design view involves a process of rethinking here. Temporal progress should be separated from the domains of reality. These are not processed in temporal succession, but are

present and linked at every point in time. Moreover, there are no preordained domains of reality; these are constructed in the course of the design process. This means:

- We do not analyze requirements; we construct them from our own perspective (cf. /Reisin in this volume/). This perspective is determined by our personal priorities and values, by the methods we use as orientation aids, and by our interaction with others constructing requirements from their perspective. Requirements are governed by perspective. In most cases, they reflect differences in perspectives and are subject to temporal change.

- We do not apply predefined methods, but construct them to suit the situation in hand. There are no such things as methods per se — what we are invariably concerned with here are processes of situative method development and application. We select methods and adapt them. What we are ultimately doing in the course of design is developing our own methods.

- We do not refer to fixed means of implementation that only take effect later on when working out the details of implementation decisions. Instead, we construct the meaningful use of means of implementation by testing, selecting or complementing what is already available.

To postulate the existence of a predefined path through these worlds is misleading. It would only be possible to follow such a path if all the relevant decisions had already been taken. But then there is no place for design.

6. Design as a Web of Decisions

Design is *rooted in concerns*. These concerns induce us to set goals that are to be attained with the help of specific means. By drawing this distinction between concerns and goals, I wish to anticipate any discrepancy between what is ostensibly to be attained and what proves to be desirable.[12] Design is normally initiated with a view to set goals, the point of departure being an already established web of decisions linking the concerns considered relevant for attaining the goals set with provisionally designated means of implementation.

Nevertheless, there is no fixed foundation for design, nor is it determined by predefined goals. The concerns may change during the design process. The means of implementation may

---

12 According to the constructivist view, the notion of goals should be treated with caution. In /Schmidt 87/, von Glasersfeldt is quoted that what ever can honestly be set as a goal may be.derived from the need to sustain autopoietic organization /V. Glasersfeld 82/. We mean something similar when we speak here of concerns.

prove inadequate. The predefined goals may turn out to be misleading or be considered no longer valid. To this extent, design may be regarded as *goal-free*. Design creates its own foundations and sets its own goals.

Design calls for an interplay of different faculties: besides a command of methods, what is needed is an awareness of the potential of the means of implementation and a sensitivity to the changing concerns of the application.

Design presupposes a range of options and scope for playfully exploring these options. In other words: design can only take place where a sufficient range of options is available for making the relevant distinctions. Design requires autonomy if a genuine choice is to be made.

Design consists of a web of *design decisions* which, taken together, make up a proposed solution. Not all necessary design decisions are taken consciously. Frequently, it is not until the proposed solution has been evaluated that it becomes clear which decisions will be needed and which consequences failure to take them will imply. The importance of explicit design decisions has been emphasized in particular by Parnas[13] . However, he only considers design decisions as a basis for modular design. Design, of course, involves a wealth of other decisions. They link concerns and means of implementation with a view to attaining the goals considered valid within a specific context. This results in building up complex structures of interwoven decisions. These must be intrinsically coherent and, overall, desirable. Their emergence is *specific to the individual design process*; it is not determined by the given problem. Instead, the problem itself is grasped in the course of the design process. Design is determined by the perspective of those sustaining the design process and by the constraints imposed upon them.

The viability of design is determined by a number of different factors, which frequently give rise to opposing ones: whether the decisions match the concerns; whether the web of decisions covers all elements of the problem that are considered essential; whether meaningful use can be made of the means available; whether the goals set are attained. These distinctions are made by an observer.

Design is thus based on a wealth of correlated distinctions concerning what is "good" (desirable). There emerges here an interplay between proposed solutions and their evaluation. What is "good" is determined in the course of the process by what those involved consider to be "good". The criteria for such distinctions are derived from the concerns relevant to design.

---

13 cf. /Parnas 72/

Design can only fully unfold where decisions that have already been taken can be revised on the basis of their evaluation; in other words, if the results of design again become themselves the starting-point for design. This is how closure in design comes about.

Decision-making, evaluation and closure are interleaved and take place on different levels: in individuals, on an informal level; when developing and checking a proposed solution; during joint critical appraisal; when practically implementing a decision; during testing; during use.

Closure involves admitting our errors and learning from them (cf. /Knuth in this volume/), offering and accepting constructive criticism, abandoning erroneous goals and recognizing changing concerns. Closure means the continuation of design.

Design is successful as a whole if the web of design decisions is stabilized in the course of revisions; in other words, if it withstands evaluation and is acknowledged as "good" by those involved despite changing concerns.

Design is, then, always *multiperspective*, even where pursued by individuals. This is due to the linkage of concerns, means of implementation and methods, to the different evaluation criteria, and to the interplay between design, implementation and use that is an important prerequisite for closure. Design requires multiperspective reflection.

7. Dialogical Orientation in Design

This section focusses on software development with others and for others — the form socially effective software development normally takes. It is to be seen here as *potentially dialogical*. The "I and Thou" of software development is concealed in basic relations such as "I develop software with you" and "I develop software for you". Teamwork in software development can be viewed as a network of this sort of basic relations. The specific network of relations between the people involved, unfolding in time, is constitutive in design..

If we acknowledge this fact and attune ourselves to it, we come to a consciously dialogical orientation. We can decide in favour of accepting the "you" and make provision for it. This means accepting the other, not instrumentalizing him[14]. The essential activity then takes place between me and the other: we develop *jointly*. We look for ways of thinking that will enable us to understand our common reality, and for forms of work that will help us to completely unfold this reality.

---

14 According to /Buber 84/

In the light of the prevailing practice in software development, this may sound absurd. As we know, software development is widely used as an instrument for exercising power and control. Software projects are managed along bureaucratic lines and controlled by means of tools. Teamwork is characterized by rivalry and lack of coordination. Failure to recognize these facts would indeed be absurd. But we are not obliged to regard this an unalterable state of affairs.

On the contrary, I consider far-reaching changes in the way software development is practised to be imperative if we are to strive for quality and human-oriented system design. I consider such a reorientation to be necessary not only for "humanitarian reasons" — as if the technical work could be performed at least equally well without taking others into account. As I see it, communal design has no chance of unfolding without this reorientation. This also seems to me to be in agreement with Goguen's views on projects as "autopoietic beings"[15].

For this reason, we at the Technical University of Berlin have been working for a number of years now to reorient software development towards a notion of system design that takes account of human needs. These efforts have led to the development of the approach STEPS[16], which has been elaborated in cooperation with other scientists and tried out in academic project situations, in the context of method development, and in practice[17].

I now go on to describe in constructivist terms the ways of thinking and forms of work that have shaped our experience .

"Dialogical orientation in design" is one way of characterizing cooperation both among developers and with users. In our experience, there are such common features. But we must also be able to differentiate between the two cases. We distinguish then between

- dialogical design among developers — by which I mean jointly working out a proposed solution together with others[18], and
- jointly creating computer-supported contexts of action with users.[19]

---

15 Goguen, 5.1 this volume

16 Software Technology for Evolutionary Participative System development, see /Floyd, Reisin, Schmidt 89/

17 PEtS Project, see /Floyd et al. 89/ and /Reisin in this volume/

18 This is theoretically elaborated and empirically underpinned by Jürgen Pasch in /Pasch 88/, /Pasch 89/ and /Pasch 91/.

19 This is theoretically elaborated and empirically substantiated by Michaela Reisin in /Reisin 89/ and /Reisin in this colume/. She, however, speaks not of "dialogical" but of "cooperative" design.

Dialogical design involves working out a desirable tentative solution between myself and others, taking design decisions on a joint basis, and bringing about closure with consideration being given to the perspectives of all parties. This means that, instead of developing my model in accordance with my evaluation criteria — objectifying and enforcing these where possible —, I should endeavour to be receptive to the perspectives of others. Instead of upholding my own model monopoly (/Bråten 73/), to which others must conform, it is up to me to take up all the other perspectives and allow them to interact.

There is little methodological support for this. Most of the methods I am familiar with are of a monological nature (strictly speaking, they postulate a pseudo-objectivity, failing to acknowledge the designer's perspectivity).[20] In a dialogical design process, we must assume that each contribution is of a provisional nature; that cooperating designers entertain different expectations with regard to the process as a whole, setting different priorities and applying different evaluation criteria. In dialogical design, we must also acknowledge existing conflicts and jointly overcome them.

Dialogical design must succeed in weaving together these perspectives in such as way that the web of decisions emerging during the design process is borne jointly. This involves meaningfully alternating between individual and cooperative work, offering and accepting constructive criticism, exploring the consequences of proposed design decisions, evaluating results multiperspectively and revising them jointly — until gradually a stable solution emerges that is endorsed by all parties.

The basic prerequisite for dialogical design is trust. This can only develop where the interests of those involved are taken into consideration. Moreover, it presupposes a willingness on the part of all parties, especially the project manager, to create and maintain a socially supportive milieu.

Dialogical teamwork cannot proceed from any implicit assumptions; it must lay its own foundations for cooperative work. This involves cooperation in establishing the project, in assigning tasks and responsibilities, in synchronizing and coordinating work, and in laying down conventions and standards for work within the team. But it also means jointly working out an authoritative project view of the basic documents and of the concerns, priorities and evaluation criteria valid in each case.

---

20 One exception here is SADT with its concept of viewpoints and the author-critic-cycle (cf. /Ross in this volume/). However, SADT fails to provide means for bringing the different perspectives into interaction with one another.

Dialogical design calls for the conscious development of a project language, linking the relevant domains of reality in a way that everyone is able to follow.

Joint goals must be set and revised during the ongoing process as the respective situation demands.

Cooperative work calls for commonly accessible and jointly maintained "external memories"[21] such as project files and diaries, recording the currently valid foundations of work and jointly taken decisions, so as to enable the decision process to be reconstructed in the case of revisions.

In contrast to the assumptions underlying the production view, dialogical design calls for rich communication between all the parties involved. Information must be continuously collected and disseminated, new viewpoints must be incorporated, and evaluations must be made from ever new perspectives.

The distribution of work tasks must always take place with reference to a jointly upheld overall view, and results obtained individually must be jointly validated. This process can be given technical support in the form of prototyping, the emphasis here being on mutual learning on the basis of preliminary implemented versions.

The measures outlined here are basically geared to facilitating the unfolding of a "Project Mind" and the emergence of a shared perspective.

8. Concluding Remarks

Although I have only been able to outline software development as design in broad terms, I should like to finish off by by showing what the results will be, should we be willing to accept this view and make proper provision for it. It offers us a conscious orientation for our theoretical and practical work.

Applied to training, it will mean equipping students with the knowledge and skills required for design.

As regards the development of methods, it will involve elaborating flexibly adaptable concepts and techniques supporting cooperative work with other developers and users. Also, a cooperative, flexible and incremental work-style will be an important factor in tool development.

---

21 Cf. /Keil-Slawik in this volume/.

Project organization will also promote cooperation in design. This means as much autonomy as possible to enable us to incorporate the element of responsibility; it means the creation of a dialogically oriented working milieu enabling shared perspectives to be formed; a division of labour that allows us to arrive at and evaluate joint design decisions; and finally, the conscious integration of revisions allowing closure in design.

Accepting and implementing this view means designing design. Undoubtedly, we still have a very long way to go before arriving at the sort of societal conditions that will allow this to take place on a large scale.[22] That makes it all the more important — by way of a contribution to desirable social changes — to outline practicable approaches to design, using as a guide our common experience of quality and human-oriented design, and to explore these approaches and prove their suitability.

Acknowledgement

This section is of considerable importance because the ideas set out in this chapter were developed in dialogical processes. In other words, they were made possible and given concrete form by cooperation, conversations and intellectual discussions with a number of people, all of whose names I am unable to list here.

The technical experience on which this contribution is based was gained since 1978 at the Technical University of Berlin in a research milieu organized along cooperative lines, and it has been substantially supported by my co-workers. My special thanks are due to Reinhard Keil-Slawik, Jürgen Pasch and Michaela Reisin and all others who have contributed to our methodological approach STEPS.

Important insights were derived from the research work I did during my stay in Palo Alto in 1987/88 and from the work involved in organizing and holding the conference on "Software Development and Reality Construction". My special thanks go here to co-organizers Reinhard Budde, Reinhard Keil-Slawik and Heinz Züllighoven as well as to all the other organizers and participants.

A considerable help to me in writing this contribution were the valuable discussions I had with Michaela Reisin. I learned a great deal from the continual and conscious crossing of our perspectives.

---

22 The masters of design are to be found in the Scandinavian countries. Their theoretical and methodological approaches — along with strategies for societal implementation of these approaches — can provide us with important insights for our own work (see /Floyd, Mehl, Reisin, Schmidt, Wolf 87/).

For enabling me to make a tentative entry into the constructivist discourse, I am indebted to three scientists for their intellectual guidance, encouragement and kind support. Stein Bråten showed me the door to this world of thought and was an invaluable help in finding my bearings there. My attempts to understand Gordon Pask's theory, and his personal support in these attempts, have helped me to acquire key insights. And last but not least, Heinz von Foerster has been a great inspiration to me and instrumental in shaping the essence of the ideas presented here.

Literature

Gregory Bateson: Mind and Nature - A Necessary Unity. Bantam Books Toronto New York London Sydney 1979

Ernst von Glasersfeld: The Construction of Knowledge - Contributions to Conceptual Semantics. Intersystems Publications. Seaside, California, 1987

Humberto Maturana, Francisco Varela: Autopoiesis and Cognition: The Realization of the Living. D. Reidel Publishing Company, Dordrecht: Holland / Boston: USA / London: England 1980.

Gordon Pask: Conversation Theory - Applications in Education and Epistemology. Elsevier Scientific Publishing Company. Amsterdam Oxford New York 1976.

Heinz von Foerster: Observing Systems. Intersystems Publications Seaside California 1984