STEPS - a Methodological Framework for Cooperative Software Development with Users

Christiane Floyd, Guido Gryczan
University of Hamburg
Department of Computer Science
Vogt-Kölln-Straße 30
D-2000 Hamburg 54
Email: floyd@informatik.uni-hamburg.de or
gryczan@informatik.uni-hamburg.de

Abstract: The paper presents the basic concepts of STEPS (Software Technology for Evolutionary and Participative System Development) a methodological framework to software development focussing on software in its relation to the work tasks of users. STEPS was elaborated by the authors and their colleagues at the Technical University of Berlin until 1991 and is now continued at the University of Hamburg.
This approach concentrates on the methodical support for developers and users and their cooperation in software development. Learning and communication in software development and use are seen in relation to the technical aspects of software construction. Software products are viewed as families of versions to be (re-)designed, (re-)implemented and (re-)evaluated in successive development cycles. Design replaces production as an overall metaphor for software development. Mutual learning in design as well as the extensive use of prototyping facilitate the embedding of software systems in meaningful work processes in the user organization.

1. Motivation for STEPS

The present contribution is concerned with the cooperative development of software to be fitted into user work processes. Here, software developers provide tools for the work of users and media for the communication between users. Thus, development work relies on cooperation both amongst developers as well as between developers and users, and the technical concerns for constructing high-quality products are inherently tied up with communication and work - social processes which define the very nature of the problems to be dealt with.

While software developers are directly involved in producing programs, they contribute indirectly to profound changes in the user organization[1]. Therefore, software developers have to understand user work processes, the potential role of the computer as an artifact, and the scope for design available taking account of the actual situation and the interests of all people involved.

Users, on the other hand, are faced with a transformation of their working lives. It brings about re-organization of work, the need for new skills in using computer programs, and far-reaching changes in competence. Depending on the functionality of the software, the mode of human-computer-interaction allowed for, the division of tasks amongst humans and the delegation of functions to computers, this may imply either impoverishmant by increased taylorization and humans being reduced to operators of the computer or enrichment from gaining a renewed understanding of the work in hand and of the potential of the computer to support it.

The development of STEPS started from the realization that existing specializations in computer science and related fields fail to provide an adequate theoretical and methodical basis for software projects of this kind. Since the authors of the present paper are rooted in software engineering, the background of STEPS is best explained by reference to the world view embodied by that discipline[2]. In [3] and [4] it is argued that Software Engineering as a research programme views software development as production, proceeding ideally with the help of formal methods on the basis of fixed requirements and being divorced from use. It also suggests the possibility of obtaining objectively correct comprehensive models as a result of requirements analysis. These ideas are so fundamental in the discipline that many specialists forget that this is *no more than a view*, pointing out some useful aspects while obscuring others.

Viewing software development as production was a standpoint adopted consciously in order to be able to control large software projects. It helps greatly to improve the treatment of computer programs as formal objects, but does not provide a platform for discussing the relevance or adequacy of programs as tools or media in use. In particular, the traditional view separates the consideration of software products from

1) the use context, i.e. the network of human activities where the software product is to be employed,

---

[1] In [1] software developers are characterized as "agents of change".

[2] This has been elaborated in many textbooks and articles, see for example the superb compendium given in [2].

2) the processes of learning and communication carried out by developers and users which lead to insights into the desired functionality and the potential use of the emerging product.

While incorporating many useful elements of the software engineering tradition, STEPS transcends the software-engineering world-view in important ways. In particular, it views

- *Software development as design* to be carried out in processes of learning and communication amongst responsible humans, developers and users, with software construction and use interleaved;
- *Methods as situated*, to be selected, tailored and applied as needed in actual project settings, rather than as rule systems prescribing plans or courses for action; in particular, STEPS itself is not considered to be a method, but a framework which offers a repertoire of methods and allows for other methods to be embedded;
- *Software quality as human centered*, relating use quality, product quality and process quality.

As a methodological framework STEPS rests on a version-based cyclical project model reflecting the cooperative work between developers and users and on notions for the dynamic coordination of cooperative work. It focusses on software requirements as unfolding in the context of organizational tasks and human activities, on modelling as a means of creating an explicitly agreed basis for anticipating the use of the product, and on human computer interaction as the technical starting point for computer supported work design. STEPS offers methods of its own and allows for the incorporation of existing methods, to the extent that they can be tailored to the needs of cooperative work in the specific project situation.

The present paper centers around the underlying concepts of STEPS and the project model. In contrast to earlier work (see [22]) it does not aim at clarifying the philosophical underpinnings of our work, but rather aims at discussing possible directions for tool support.

2 Perspectivity in Software Design

Perspectivity is constitutive in all human knowing and doing. In recent years, the term has been widely used in psychology and sociology, on one hand, and in system development literature[1] on the other hand.

By "perspective" we mean here a class of related views on relevant aspects of an area of concern from a common view-point. Each perspective unfolds against a background which is hidden and is selective in terms of the aspects it brings to the fore and the priorities it sets. In the literature, the term "perspective" is used in two distinct ways which are both relevant here:

1)Perspectivity arises implicitly from our individual subjectivity as shaped by our culture, our beliefs and our life experience. We can never leave our perspectivity in this sense, we can, however, aim at elucidating it to some extent in communication with others. Clarifying and crossing the individual perspectives of those who participate in design is of paramount importance for arriving at a common understanding, an intersubjectively shared platform for development work and for design decisions that reconciliate between conflicts of interest amongst those involved. While perspectives are always inherent in our thinking, we can attempt to clarify them, allow them to interact and gain deeper insights from their interrelation. STEPS aims at supporting such processes.

2) The term perspective is also used to denote explicit and named positions adopted consciously by individuals or groups and related to common tasks, formulated interests or specific paradigms. Examples are all notions of perspectives given in [1] as well as the proliferation of ever new perspectives for human computer interaction. STEPS also relies on adopting explicit perspectives in this sense. It distinguishes the "use perpective" held by those who interact with software as a tool or as a medium from the "development perspective" held by software developers. Furthermore, there is no one single use perspective, but a plurality of perspectives related to functional roles of different users, to collective interests and to individual tastes and priorities.

Different perspectives on one and the same software system may be in conflict due to misunderstandings or clashes of interest. If we take working with users seriously, these conflicts must be acknowledged and dealt with rather than smoothed out in a quasi-objective system

[1]This definition is related to but not identical with the one given in [1]. For the practical use in this paper there is little difference, however. A comprehensive treatment of perspectives and metaphors used in connection with Computer Human Interaction is given in [3].

analysis. Thus dealing with multiperspectivity becomes a basic prerequisite for cooperative work.

3. Subject-Orientation in Software Development and Use

Generally, STEPS reflects a human centered notion of software development and use, viewing them as processes of learning, work and communication. Since the possibility for theses processes to unfold cannot be taken for granted, but depends on the responsible action of human subjects with a suitable scope for choice, we will first introduce these ideas and discuss them in connection with related views from the literature.

We have adopted the term "subject-orientation" from [6] as the title of this section because the phrase is suggestive in two ways:

1) it refers to human beings as subjects of their work processes, subject-ness is associated here with competence and responsibility.

2) it emphasizes, that a conscious orientation is needed in order to enable human beings to feel and act as subjects of their work processes.

These issues have been of fundamental importance throughout the age of industrialization but they take a distinct flavour in connection with computing. Here, work design reflects specifically the view of human beings in relation to computers embodied in system design.

Traditionally, computer science starts from equating humans with computers. This is explicit in the use of the computer metaphor for human cognition prevalent in artificial intelligence and implicit in the established assumptions of fields like software engineering and human-computer interaction. As a consequence we find perspectives for software development and use reflecting the equation of human beings with machines since they concern themselves - only - with rule guided rational thinking and predictable functional human behaviour[1].

3.1 Subject-Orientation in Software-Use

STEPS aims at supporting the development of subject-oriented information systems as discussed by Nurminen (in [5]). While he starts from analyzing the use situation, in order to enable responsible work, where the use of information systems results in computerized tasks as inseperable parts of daily work, the contribution of STEPS is to promote development processes that help to achieve this kind of use situation.

---

[1] A deeper treatment of these issues can be found in [5] and several other contributions in [7], in particular [8].

It is therefore necessary to intertwine development and use of new information systems. Consecutive cycles of development and use will result in a mutual learning process, where developers learn about users´ work procedures and users learn about possibilities and restrictions of computer technology.[1] Therefore cooperation among users and developers and cooperation within the developers team is not only necessary, but mandatory for a projects success.

3.2 Subject-Orientation in software development

While understanding the use-situation is of paramount importance for design, the present contribution focusses on cooperative software development, viewed as production in the traditional approach. Associated with the idea of production is the notion of a *software factory* . It suggests that software should be developed ideally like in an assembly line in separable production-stages based on the application of formal methods. The underlying assumption is, that software development is a formal activity quite opposite to human activities like learning or playing games. In contrast to this approach many software developers experience a painful gap between their aquired ability to apply formal methods and the relevance of these methods for the acceptance of the result of their work. Furthermore they learn how crucial a close, social relationship to both colleagues and later users for successful software development is.

We have to consider that computer programs emerge as the outcome of complex human processes of cognition, communication and negotiation, which serve to establish the problem to be dealt with and to anticipate the meaningful embedding of the computer system in its intended use context.[2] This view first of all takes into account, that software development is a *social activity* ([1]) where all participants are involved in keeping with their perspectives. That means, in a social process the problem to be solved is constructed by sharing perspectives.[3]

Moreover, the understanding for the area of concern increases gradually among the project participants. This can be referred to as *theory building* [13]. Theory building happens in a continuous process, enfolded in the totality of activities carried out by the people involved.

---

[1] The focus on the cyclical nature of the development process makes STEPS different from other "evolutionary" approaches (see [2]), which - on the basis of an enhanced waterfall model - allow "going back" to previous phases. The enrichment of waterfall models fail to provide a deeper understanding of design processes, as their focus remains on the production of a single system.

[2] In connection with STEPS this was elaborated in particular in [9] , [10], [11] ,and [12].

[3] (See also [13]) where this idea is applied to data modelling.

Thus, theory building is a *human activity* enabling software developers to cope intelligently with questions and problems as they arise.
Subject orientation means here that all participants in a project are viewed as skilled workers, who alone have the specific knowledge to fullfill their jobs. It is the responsibility of system developers to take this fact into account.

3.3 Towards tool support for subject-oriented, cooperative work

The underlying perspective to software development results, in particular, in distinct concepts for tool support.
The *software factory* is associated with Computer Aided Software Engineering (CASE) Environments. Here we find that users of the environment, i.e. the developers, are confronted with a tayloristic organization of work. They are viewed as mere input devices for a pre-planned procedure, and there is no way on working around the pre-defined work plan. The criticism against software factories is based on the observation, that skilled software workers often come to situations where the pre-defined work procedures do not conform to tasks that have to be accomplished in a specific situation. Although Software Factory protagonists claim that "the factory analogy applies only to the goal of software production, not its industrial implementation" ([15]) they argue that "process information", i.e. information about the development process, is an important characteristic of a software factory. It is exactly this kind of process-information which leads to an alienation of software workers from work processes.
The tool-perspective, on the other hand, is best expressed within the *workshop* metaphor ([16]). Here, software-workers use tools which are designed to serve for a specific task, without making assumptions about the context in which the tool is used. This means especially that no order on the usage of tools is pre-defined. Construction of tools for a tool-workshop is based on the experience of software workers, with a special emphasis on work tasks which can be decontextualized.
As a result, users of tools are always in **control** of the action of a tool. Furthermore they have to be in control of the results - especially computerized - tools produce. These results can be used for other work tasks, whether these are computerized or not. That means, tools do not take actions by themselves, but are driven by human action.
Subject-orientation thus leads to a holistic view, where usage of computerized tools does not eliminate workers freedom of choice in selecting and sequencing work tasks. Its purpose is to take advantage of the capabilities of the computer object, and thereby enhancing the chances for creating challenging jobs. The creation of information systems, i.e. computerized tools, is the creation of such tools and the restructuring of work.

In the following we will outline the methodological framework STEPS which is in line with the subject oriented approach.

4. Cooperative work in Software Development

In keeping with its perspective STEPS aims at supporting software development so as to enhance user competence. It considers software development a learning process for both developers and users and promotes cooperation in design.[1]

As a consequence of this orientation, it has strong connections with other efforts both in software engineering and in participative system development.

STEPS, thus, is a framwork for software development taking account of its connections to work design.[2] The content, organization and conditions of work determine the quality and structure of the software system to be developed. Conversely, the software system has an impact on the working processes. Crucial for judging the quality of a computer-based system from the users point of view, is an evaluation of the quality of their work when using the software. This gives rise to our view of system development, which is not restricted to the product software, but also takes into account the social processes and relations in the context of which it is produced and used.

Those participating in a software project are creative in two respects: In designing a computer-based system, they are creating and shaping a product, and, at the same time, the development process itself. These two dimensions are of a reciprocal nature. They are usually, though implicitly, reflected in the course of system development by two classes of different activities: product-oriented and process-oriented activities.

STEPS provides guidance to developers and users for carrying out their cooperation concerning product-oriented and process-oriented activities. It does not claim, however, to furnish generally applicable criteria on how computer-supported work should be designed. To the extent that such criteria are available at this time, we consider them to be within the realm of social theory.

---

[1] As methodical approach to software development, STEPS is closely related to the view of programming as "theory building" presented in [14]. As approach to participative system development, STEPS is related to pioneering efforts in England ([17]) and in Scandinavia. (see, for example, [18],[19], [20], and our study [21]).

[2] In what follows we draw extensively on an earlier paper ([22])

Rather than supplying ad-hoc criteria of our own, we aim at showing ways for how such criteria can be accommodated and put into practice in software development.

This leads to concrete consequences for software development. While traditional approaches tend to advocate the early construction of comprehensive and quasi-objective models relying largely on formal techniques to be used in defining documents, the emphasis here is on recurring to perspective-based evaluation as the supreme guide both in building models and in interpretating constructed models in the context of meaningful human activities.

Therefore, we relate software requirements to the context of user work processes as a whole (see [23]).Treated in this manner, requirements and system functions become distinct domains of discourse, which are relevant throughout system development and are connected actively through design. Requirements are not "given" and therefore cannot, strictly speaking, be analyzed. Their gradual establishment takes place in an interplay of anticipative, constructive and evaluative steps to be carried out by developers and users in interaction.

Owing to ongoing social changes, it is not possible to define the required functions and quality of a software system, which is to be used in working processes, completly at any fixed point in time. On the one hand, requirements evolve because of general changes in society, the economy, technology, law etc. On the other hand, changes in work organization or users qualifications, which are not least an effect of the software system itself, give rise to new and changing requirements. In order to keep pace with changing social and economic interests, software development must be based on the evolutionary development of the use organization and, in particular, of the work context in which the software is directly applied.

On the technical side, therefore, STEPS embodies an evolutionary approach comprising various forms of prototyping and the development of systems in versions. This approach can be visualized in a static project model that allows to choose from a class of situation-specific strategies as needed for the project in hand. While a project model is useful as a map showing salient features of the territory of interest, the actual cooperative process must be initiated and coordinated dynamically in the course of the project itself.

The development, then, consists of an interleavement of some activities to be carried out jointly by developers and users and of others to be carried out by the respective groups on their own. Conventional software development methods for requirements analysis and design have their place, but need to be tailored to the needs of the communicative processes at hand, so as to show multiperspectivity and to allow incremental work.

STEPS embodies a human-centered notion of quality emphasizing the experience with software in use as primary level of concern. We consider it important to give precedence to the specific user-oriented quality criteria, which are derived from the application area and related to the people working there - such as handleability, amenity, intelligibility etc. - over the general technical quality criteria, such as reliability, portability and compatibility which we consider to be necessary prerequisites for meeting user-quality criteria.

In order to make up the specific product quality, it is important to focus not only on the formal aspects of the application area. Informal work factors, differing perspectives, conflicting interests and divergent preferences must be given attention as well. STEPS aims at facilitating the emergence of quality experienced by all participants through cooperative design.

5. A Project Model for Software Development with Users

As a basis for systematic work in software development, STEPS relies on a project model as shown in figure 1. This model serves as a map to guide all participants into the territory of software development by establishing a common understanding of the tasks to be performed. As can be seen, the project model is distinct from conventional life cycle models (Waterfall models) in several ways:

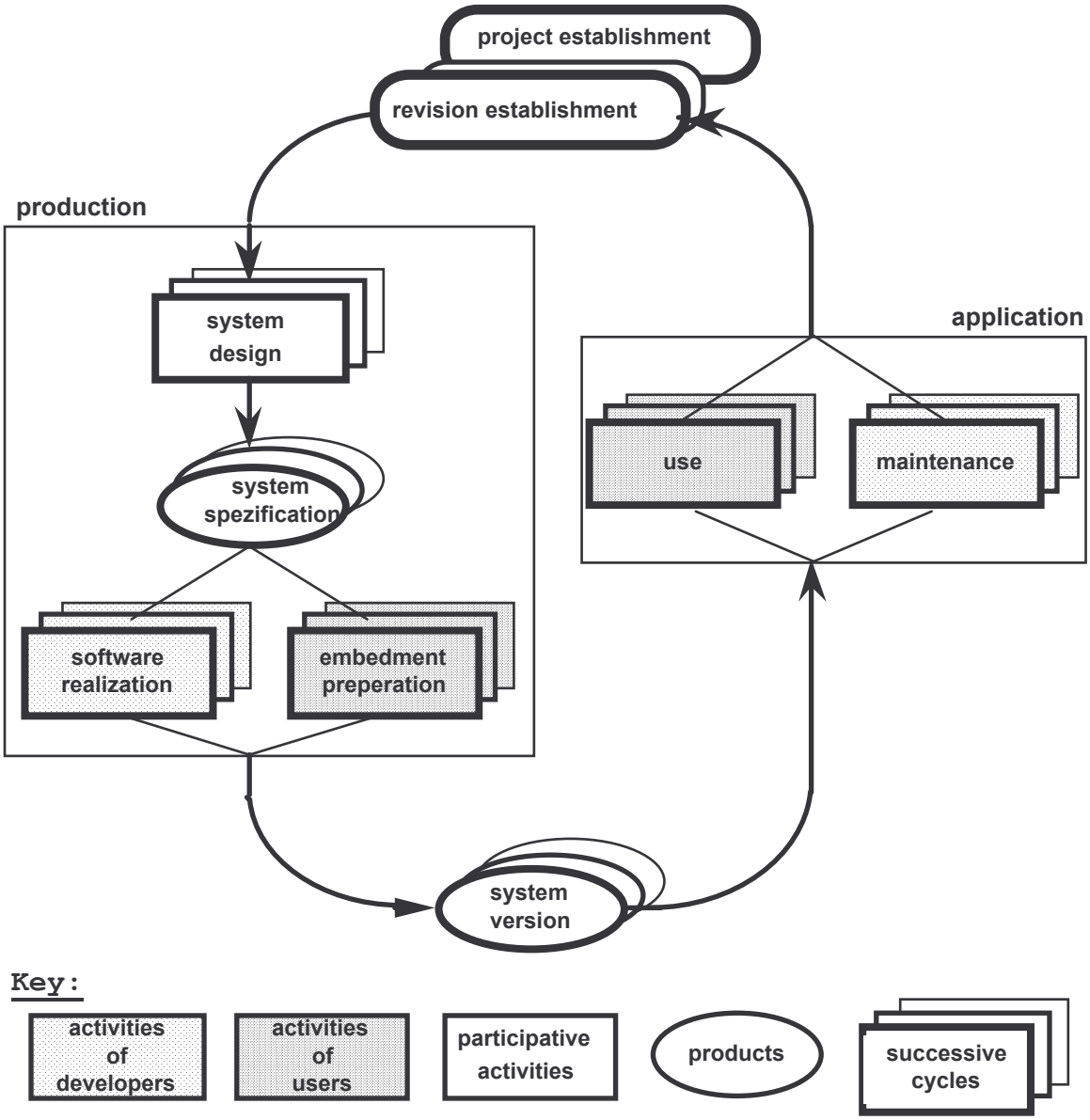

Figure 1: Project Model of STEPS

- It is cyclical rather than linear, portraying software development as proceeding in system versions, all development steps and defining documents therefore being subject to revision.[1]

[1]Due to its cyclical nature one can observe similarities between the STEPS Project Model and the Spiral Model of Software Development as proposed in [24]. This model takes the difficulties into account which arise by using a waterfall-oriented approach. The main difference is the focus on mutual learning processes in the cyclic project model as opposed to project risks in the Spiral Model. Although the Spiral Model takes prototyping as one of

- It combines software production and application, visualizing the tasks of developers and users rather than those of developers only; the development of each technical system version is fitted to an associated re-organization of user work processes; the interplay between these two is anticipated in the coperative design and evaluated in the cooperative revision step.

- It refers to a class of possible development strategies allowing the choice of a situation-specific strategy as needed in the project at hand rather than depicting one ideal development strategy to be copied as closely as possible in all projects. A situation-specific strategy consists in planning the development cycles and the prospective scope of the system versions in keeping with the needs of the work processes, with available resources and actual modalities for cooperation.

- It relies on a minimum of pre-defined intermediate products, thus allowing the freedom for choosing the actual intermediate products as needed in the ongoing process; intermediate products are system versions as well as defining documents, they are subject to evaluation and planned revision.

- It avoids suggesting that development steps should be seperated sequentially; rather it assumes that they are relevant throughout all development cycles. They need to be separated and re-connected in system design, and our understanding of them increases in successive cycles of production, application and subsequent revision.

- It incorporates the dynamic coordination of the ongoing project in the form of the project (and version) establishment. In the course of the project, the dynamic coordination continues through reference lines. These, however, are not shown in the model, as they occur as needed during the actual process.

We consider system design, software realization and use to be the most important activities in software development.

---

several possible approaches into account it is predominantly concerned with the development of one software system, as opposed to the development of versions of a software product in the cyclic model.

In *system design*, the quality and structure of the software version and the computer-supported work associated with it are established, evaluated, and determined by the users and developers cooperatively.
Software realization by the developers comprises the design of the software architecture as well as the effect specification, implementation, testing etc. of the system functions.
Use of the software system refers to its employment by the users in the course of their daily work.

For a particular development cycle the project model calls for two predefined products only: the system specification and the system version.
The *system specification*, the result of system design, is produced by the developers. It contains the technical aspects of the system version to be realized and the organizational aspects pertaining to its use.
The *system version* incorporates, in addition to the realized software version, the application computer and all the documents needed for the use and maintenance of the system.

The STEPS approach reflects our view that design-principles such as "top-down development" and "separation of concerns" are thorougly misleading where they are meant to be applied to the temporal organization of the the design process. We find them, on the other hand, useful for providing excellent criteria for structuring the results of design.

So far, we have concentrated on evolutionary system development as the primary prototyping strategy. Evolutionary prototyping is a continous process aiming at fast reaction towards changing needs and wishes of users and the changing environment of the information system. This overall strategy can be supplemented by specialized prototyping techniques, serving specific needs in the development process. They will be introduced in the following.
We distinguish between different *forms of prototyping,* depending on one's current concern in the prototyping process.
Exploratory prototyping should be selected to clarify users' requirements, and can therefore be used during the whole requirements engineering process. With the help of exploratory prototyping, alternative -computer-based - solutions can be worked out. It is especially meaningful if the problem to be solved is unclear. Here, requirements of users and management regarding the system to be constructed can be clarified. Subject to the process of discussion, i.e. learning, are not only kind and amount of computer support, but also which parts of work can and should be computerized. During exploratory prototyping developers gain insight into work and problems of users. Although exploratory prototyping offers great benefits, one should

recognize a potential danger, namely that, at an early stage of development, users and developers commit themselves to a limited number of possible solutions.

Using *experimental prototyping*, a better understanding of certain aspects or parts of a system can be achieved. Emphasis is spent on the techical realization of those aspects. During experiments users clarify their expectations of a computer based information system. Within these cooperative sessions users and developers mainly discuss questions surrounding human-computer interaction and technical questions concerning the prototype. (For an in depth discussion of Prototyping as a technique and related kinds of prototypes see [25] and [26])

The developed prototype serves as an executable model of the software system. As one can see, all kinds of prototyping mentioned above are useful for supporting the communicative processes between users and developers of a system. One cannot underestimate the value of prototypes for learning processes during a project, may they be intended for project establishment, like throwaways, for the discussion of technical questions, like skeletons, or as versions of the target system in evolutionary prototyping.

6 Conclusion

STEPS as an overall approach and all its structural components are outcomes of resarch projects conducted by members of our group. They are derived from practical work in projects in a variety of settings. We have learned, that the development of methods itself is a learning process. Hopefully, STEPS will make progress as a map to guide all participants into the territory of software development, but it will never claim to be the territory itself.
The current research on STEPS concentrates on developing a workshop according to the workshop metaphor with tools for cooperative software development. The ensemble of tools we have in mind will allow to look at materials from a software project based on perspectives.
Specific interest is given to an integration of object oriented techniques. It must be emphasized however that STEPS is not bound to the availabiliy of particular programming paradigms or tools. It allows for a variety of strategies to be selected as needed in the actual project situation.

References

[1] K. Nygaard: Program Development as a Social Activity. In: K. Fuchs-Kittowski, D. Gertenbach (Eds.): System Design for Human Development and Productivity: Participation and Beyond, Akademie der Wissenschaften der DDR, Berlin (GDR), 1987.

[2] John A. McDermid: Software Engineer´s Reference Book, Butzterworth Heinemann, 1990.

[3] S. Maaß, H. Oberquelle: Perspectives and Metaphors for Human-Computer Interaction In: [7], 233-251.

[4] C. Floyd: Outline of a Paradigm Change in Software Engineering. In: G. Bjerknes, P. Ehn, M. Kyng (Eds.): Computers and Democracy – a Scandinavian Challenge, Avebury, Gower Publishing Company Ltd., Aldershot, England, 1987.

[5] C. Floyd: Human Questions in Computer Sciences In: [7] 15-28.

[6] M. Nurminen: A Subject-Oriented Approach to Information Systems In: [7], 302-312.

[7] C. Floyd, H. Züllighoven, R. Budde, R. Keil-Slawik (Eds.): Software Development and Reality Construction. Springer-Verlag, 1992.

[8] W. Volpert: Work Design for Human Developments In: [7], 336-348.

[9] C. Floyd: Software Development as Reality Construction In: [7], 86-100.

[10] J. Pasch: Dialogischer Software Entwurf, Dissertation, Technische Universität Berlin, 1992.

[11] F.-M. Reisin: Anticipating Reality Construction In: [7], 312-326.

[12] R. Keil-Slawik: Artifacts in Software Design In: [7], 168-188.

[13] H. K. Klein, K. Lyytinen: Towards a new Understanding of Data Modelling In: [7], 203-220.

[14] P. Naur: Programming as Theory Building, Microprocessing and Microprogramming 15 (1985) 253-261.

[15] C. Fernström, Kjell-Håkan Närfelt, and Lennart Ohlsson: Software Factory Principles, Architecture, and Experiments, IEEE Software, March 1992, 36-42

[16] R. Budde, H. Züllighoven: Software Tools in a Programming Workshop,In: /Floyd et. al. 92/, 252-268.

[17] E. Mumford: Sociotechnical Systems Design – Evolving Theory and Practice. In: G. Bjerknes, P. Ehn, M. Kyng (Eds.): Computers and Democracy – A Scandinavian Challenge, Avebury, Gower Publishing Company Ltd., Aldershot, England, 1987.

[18] N.E. Andersen, F. Kensing, M. Lassen, J. Ludin, L. Mathiassen, A. Munk-Madsen, P. Sørgaard: Professional Systems Development, Prentice Hall, 1990.

[19] P. Ehn: Work-Oriented Design of Computer Artifacts, Almquist & Wiksell International, Stockholm, 1988.

[20] G. Bjerknes: Shared Responsibility: A Field of Tension In: [7], 295-301.

[21] C. Floyd, M. Mehl, F.-M. Reisin, G. Schmidt, G. Wolf: OUT OF SCANDINAVIA – Alternative Approaches to Software Design and Development and System Development. Human-Computer Interaction, 4(4): 253-349.

[22] C. Floyd, F.-M. Reisin, G. Schmidt: STEPS to Software Development with users. In C. Ghezzi, and J.A. McDermid, (Eds.), ESEC ´89: 2nd European Software Engineering Conference, Lecture Notes in Computer Science, volume 387. Springer-Verlag, 48-64.

[23] R. Keil-Slawik: Supporting participative systems development by task-oriented requirements analysis. In: K. Fuchs-Kittowski and D. Gertenbach (Eds.): System Design for Human Development and Productivity: Participation and Beyond, Akademie der Wissenschaften, Berlin, GDR, 113 124.

[24] B. W. Boehm: A Spiral Model of Software Development and Enhancement, IEEE Computer, May 1988, 61 - 72.

[25] C. Floyd: A Systematic Look at Prototyping; in: R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllighoven, (Eds.): Approaches to Prototyping, Springer-Verlag, 1985, 1-18.
[26] R. Budde, K. Kautz, K. Kuhlenkamp, H. Züllighoven: Prototyping - an Approach to evolutionary System Development, Springer-Verlag, 1992.