Research Report 18/98

Developing a Language for Participation -

Project Language as a Meeting Place for Users and Developers in Participatory Software Development

by

Yvonne Dittrich

Department of
Computer Science and Business Administration
University of Karlskrona/Ronneby
S-372 25 Ronneby
Sweden

ISSN 1103-1581
ISRN HK/R-RES—98/18—SE

Developing a Language for Participation - Project Language as a Meeting Place for Users and Developers in Participatory Software Development

by Yvonne Dittrich

ISSN 1103-1581
ISRN HK/R-RES—98/18—SE

Copyright © 1998 by Yvonne Dittrich
All rights reserved
Printed by Psilander Grafiska, Karlskrona 1998

Developing a Language for Participation

Project Language as a Meeting Place for Users and Developers in Participatory Software Development

Yvonne Dittrich
Department of Computer Science and Business Administration
University of Karlskrona/Ronneby
Soft Centre 37225 Ronneby
+46 457 78783
Yvonne.Dittrich@ide.hk-r.se

ABSTRACT
During participatory development different professional groups with different professional languages meet. They have to communicate about the future software in a profound way. To enable that, a common way to talk about the future software has to be developed, relating concepts of the use context and concepts of software development. An example of the development of such a project language is given and the relevance of this for design is argued for. To support the development of a project language a toolkit is provided in which methods are compiled that respects the creative side of ordinary language.

Keywords
Ordinary language, participatory development, communication.

INTRODUCTION

In participatory software development, different professional groups meet in order to design, develop, and finally use software. Usually, these groups come from different professions with different practices and skills. To really participate in development the users somehow have to understand about technical possibilities and the design alternatives. To be able to develop software that is supporting the users, developers have to be able to discuss about the embedding of the software in the work practice. The issue to support users to participate in design and means like mock-ups, rich pictures, or prototyping is widely discussed in the participatory design community. (See for example [10], and [23]) However many of the methods focus implicitly or explicitly [20] on the early phases of development. The outcomes of the participatory design phases are handed over to software developers working in a more or less traditional way. The problem then is, if and how the results of such a process can be communicated to the developers.

Based on an example of a student project at the Hamburg University I argue for the importance of the development of a common language as a framework of reference for the articulation of the future users as well as the design of the software. This common language allows to express the (designed) relation between the future software and the work practice developed. The development of a project language is argued to be part of the design and development work. The participation of the developers in the participatory design process therefore is as important as the user's.

Already in traditional software development a project specific language evolves over time; specific terms for example about details of design are coined, a special way to anticipate the use context and its relation to the software is developed. For participatory development equal opportunities for all participants to contribute have to be cared for in order to prevent a domination of one perspective, which is in most cases the developers'. A project language that is developed through and provides the means for communication and crossing of the different perspectives can serve as a meeting place for users and developers.

In order to support equal opportunities of contribution a 'project language toolkit' is provided that compiles methods to support the language side of software development while respecting the creative aspects of ordinary language.

This project language toolkit is introduced in the last section. To provide a background, in the next section the language side of software development is discussed. After a first clarification, of what is meant by language, an example for a project language is provided. Already in a small scale student project the development of project specific terms and concepts can be observed. The development of a project language goes hand in hand with a corresponding design. After this, the following section discusses different approaches to address the language side of development and communication. The specific approach in this article is argued. It is argued that language itself can not be subject to design. Therefore the toolkit focuses on the facilitation of the communication that is necessary for language development and methods to mirror and capture the language under development. The article concludes with a summary and shows future issues in research about language and software development.

Throughout the article the argumentation builds on the design perspective of software development. [11] The development of software which is embedded in work practice can not just be regarded as the construction of a technical artifact. Together with the computer application the use and the embedding work practice is developed. Software development for socially embedded systems has to take that into account.

The contribution of the future users as specialists of their work practice is required as much as the computer science competence of the developers. As the use context is changed through the introduction of computer applications, the co-development of work practice and technology requires an evolutionary approach. The construction of prototypical versions and their evaluation in use have to be intertwined. [11]

In Christiane Floyd's context in Berlin and Hamburg methods and tools have been developed, to allow for participation also on a level of architectural design and data modeling. [29, 30] Therefore a close cooperation between software developers and users is required. This cooperation consists to some extent of a joined anticipation of the future computer application and the changed work practice. It relies heavily on communication and therefore on language and other symbolic means, the more as the product itself can be regarded as a symbolic artifact.

However, even speaking the same basic language – English or German or any other – work languages and professional languages differ as do the professional backgrounds. During the common development process the different perspectives have to be related. A common language has to be developed in order to achieve a mutual understanding.

LANGUAGE, PRACTICE, AND DESIGN

As I hinted at in the last section, the communicational problem can not be solved through just translating the differing concepts. Our language is part of a special practice, a special way of relating to and of acting in situations.[1] It is inseparably related to that practice. The meaning of terms and utterances depends on their use in a specific pragmatic context. It can not be separated from that context.

Both users and developers have their own professional language that is closely related to the different work practices. Already telling about the work is another practice and requires different concepts, a different language.[2] To really understand the other's language it would require to take part in the other's work practice.

If users and developers speak different languages, how then can they understand each other? Pelle Ehn, who also uses the Wittgenstein's language games to conceptualize the difficulties in understanding between users and designers, emphasizes the necessity of tangible design artifacts like mock-ups and prototypes providing hands on experience for the users and allowing them to articulate concrete proposals and feed back. [10] These artifacts serve as 'boundary objects' bridging the gap between the language games of the use context and the language games of design. In the example described in the next section, it becomes visible that not only common design artifacts are used, but a common language is developed.

Language itself is not only a fixed set of vocabularies and a set of rules. It is means *and* product of human interaction with the surrounding. We encounter in our everyday life a lot of situations that challenge us to express new aspects of life or new perspectives on well known areas of discourse. To communicate about that, we search for words, try out terms, express ourselves in odd ways and we ask the people we are talking with to contribute their understanding.[3] In the ongoing communication new expressions are tried out, and evaluated regarding to their contribution to a mutual understanding. With this development of language the pragmatic context develops as well.

Software design and development is a special kind of practice, a design practice. A not yet existing computer application has to be anticipated, represented, and finally implemented. Therefore software development relies heavily on language and other symbolic means like graphical notation, programming languages, or mock ups. To anticipate the relation between a future work place and the software under development the cooperation between developers and future users is necessary part of that practice. A project specific form of design practice and a corresponding project language has to be developed in order to co-develop the computer application and the embedding work practice. This development is depending on communication. Users and developers have to 'talk themselves together'. The result can be regarded as a project language based on a common practice and common design artifacts.

To illustrate what such a project language might look like and how it relates to the evolving software the following section gives an example.

[1] This understanding of language can be related to Ludwig Wittgenstein's concept of language games. For detailed argumentation see [8] or [9].

[2] See [16] for more details. The new way of talking about the work may result in a change in the work practice even before the software is developed. [21, pp. 99]

[3] This understanding of language is based on Wilhelm von Humboldt's language philosophy. [17]. He describes language as having two aspects: It can be described as 'ergon', a set of rules an a vocabulary, a dead mass that we encounter, when we want to express ourselves. And it can be regarded as 'energeia', as the perpetual work of using oral and written signs to express our thoughts and perspective. Both aspects depend on each other. We use the already developed system as a base for everyday communication, but contribute to its 'maintenance' through using it in new ways, developing it according to actual needs. This way it is kept 'alive' as an up to date means of communication in a changing world.
However, it is also possible to relate that perspective on language to Wittgenstein's ideas. (See [8, pp. 172ff].)

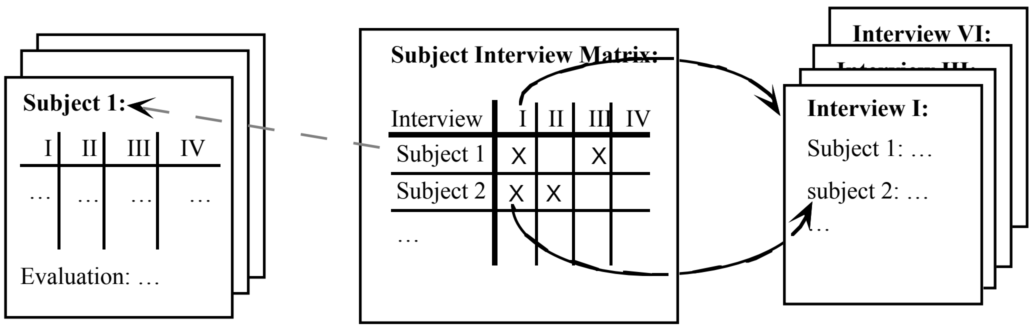

Figure 1: Subject-Interview-Matrix

DEVELOPING A LANGUAGE FOR DESIGN

During the winter term of 95/96 Christiane Floyd and I taught together a course on software development in the computer science department at the University of Hamburg. Part of the course involved the development of a 'project environment' to support the planning, design and evaluation of qualitative interviews. The students used a Web server, developed a structure to categorize the documents in an appropriate way, designed pattern documents, and used common gateway interface (CGI) scripts to implement support for the evaluation of qualitative interviews according to [24].

Semistructured qualitative interviews were used for another assignment during the course. Their evaluation is based on the extensive development and use of documents derived from the transcriptions of the interviews: In each transcript relevant subjects are identified. The sections of the interview where a subject showed up are put together. The thematic index of the transcript is the base of an evaluation of the corresponding interview regarding the single subjects. As there are often more than one interview, in a second step the subject related evaluations of each interview are related to each other. These connecting and comparing evaluation are the foundation for the final report of the case study. To allow for orientation, which subject showed up in which interview is marked in a subject-interview-matrix. Reviews are necessary throughout the evaluation process to compensate for individual biases.

We decided to develop some support for the document handling and development during the evaluation process as part of the project environment.

The subgroup responsible for this small scale software development project took part in the other assignments as well. The development was organized in a participatory way. Halfway through the course we planned some reflection on our own language. We collected terms that were new for the participants.[1] We identified different groups of terms:

The concepts, the students contributed with came on one hand from the area of technical implementation, the Web infrastructure and tools: WWW, pages, links, pointers, HTML-constructs, URL, Browser are just a few examples. These concepts were new for some students as not all of them studied computer science as a major, the others had to acquire them on another level.

Another set of concepts originated in the social science methods to be supported through the project environment: interview evaluation, protocols, to transcribe, subject-interview-matrix, interview guidelines among others were related to the use context.

A third set of concepts the students were to my surprise totally unaware of. Nonetheless I found them in the documents they produced themselves. These concepts were specific interpretations of terms from the technical implementation context as well as the use context: 'document' referred to the project specific documents made available on the project server, 'document types' related to a project specific categorization of documents like 'protocols', 'background material', or 'interview guidelines'. 'evaluation documents', and 'pattern documents' described the specific support already developed.

During the second part of the project additional concepts from the area of technical implementation – Pearl, CGI-scripts, references – and from the use context – subject areas, thematic index, interpretation of an interview, subject related evaluation, comparing evaluation – showed up.

Meanwhile the project specific concepts developed as well. Especially the 'subject-interview-matrix' changed meaning and became relevant. As mentioned above, this matrix is a table relating subjects to the interviews in which they showed up. It is used in 'paper based' evaluation as well. In the project environment the web based implementation of the subject-interview matrix (Fig. 1, center) became a central navigation tool. The headings of the interview dimension were realized as

[1] The analysis focussed on the concepts. However, the concepts are to be regarded related to their usage, their pragmatic context. We understood them as pointers to the broader language-practice context.

links to the subject oriented evaluations of each interview (Fig. 1, right); behind the marks identifying a specific subject appearing in an interview were links to the related part thereof; the headings of the subject dimension were realized as pointers to dynamic documents which extracted parts from the different interview specific evaluations and compiled them into one document leaving room for a connecting and comparing evaluation regarding one subject (Fig. 1, left). The term 'subject-interview-matrix' was used for the document as well as the navigational means it provided.[2] The change of meaning for this term and the design and implementation of the corresponding functionality can not be distinguished. The development of language has to be regarded as part of design.

Even during such a small project a special project language had thus developed. This language consisted of project specific interpretations of concepts from the area of technical implementation as well as concepts originating in the use context. The term 'document type' for example was used for the categorization of the project specific documents. The term 'subject-interview-matrix' is an example of a special interpretation of a concept from the use context and its implementation using a different technology.

Surprisingly the participants were not aware of these concepts. As they took part in their creation, they were not 'problematic' for them at all. The terms they recognized as new were the already established terms from the various domains which they had to acquire during the project.

The even participation in design and development was probably due to the small size of the group, a quite common background, and a very intense cooperation; it was not necessary to discuss the usage of terms as a base for the development. In this specific case the development of a common language was unproblematic. This is not always the case. In bigger groups with a more pronounced division of labor, it becomes a problem.

In our project the developers and the users worked close together, both took part in doing and evaluating interviews as well. This way developers had experience from the use context as well. That is a quite unusual situation. How can the development of a project language in a more 'normal' situation be supported?

SUPPORTING THE CREATIVE SIDE OF LANGUAGE

The relevance of the users' professional language for software development has been recognized for a long time, even outside the participatory design community: It is in most cases regarded as an important source for requirements engineering. Stamper and his group for example developed based their method of software development on the analysis of semantic and discourse structure of the language of the use context. (See for example [32].) Other approaches try to support the communication with the users about the requirements specification by using a subset of natural language that can be analyzed by a program. The resulting formal representation can be subject to proofs of completeness and ambiguity. (See [15] for an especially elaborated example.)

Both ways address the communication and language problem between users and developers with an emphasis on control and formalization. Formalization of language categories have thoroughly discussed in coordinator debate. Winograd [33, p. 193] and Grudin [COOD, p.58] pointed out, that software always formalizes categories, explicitly or implicitly. But as 'categories have politics' (Suchman, [33, p. 177], it is important *not* to control or formalize the language that serves as a means to decide on these formal categories. Even the formalization of ordinary language concepts of the use context implements an implicit design decision.

Ordinary language is a highly optimized means to communicate about and adapt to a changing environment. Its ambiguity, its vagueness, its openness for change allow for creative interaction and anticipation. To constrain its creative side by control or formalization would interfere with the creativity necessary for design.

The above described project language is a good example in this respect. Even as we expected the development of language to take place, we could neither anticipate what that language would look like nor with in what kind of computer support the project would result: Of course the professional terminology of the use context as well as the concepts and notations of methods and technologies from the developers side is known. However, already the work language of users and developers is related to local contingencies, personal background and so on. Moreover as I mentioned above, language is both medium and product of interaction between individuals and their surrounding. The development of a project specific language and project specific concepts is to be regarded as a creative achievement in situ.

This creative side of language is – as the example of the 'subject-interview-matrix' showed – part of the design and development of software. From that point of view, the development of a language to talk about the future use and the implementation can be regarded as creating the means of development. Design of the future software and development of a project language go hand in hand. The project language provides a common framework of interpretation for the different symbolic and non symbolic design artifacts [19] in the context of this specific project. Without such a framework the communication and common theory building [26] would be impossible.

The problem then is, how to support the creative side of language, without controlling its outcome. One approach focuses on facilitating the communication and allowing

[2] The subject-interview-matrix had been evaluated regarding the support for software development as group work. [7]

the handling of communicational obstacles.[3] Another way makes use of and supports the reflective aspect of language. The developing concepts and terms are represented in written or graphical form in order to feed them back into the process. The development and reflection of the common language and its relation to design becomes a subject of the process itself.

In the context of Christiane Floyd's group, quite a few methods have been developed addressing specific aspects related to communication an project language. However there is not one right way to handle communicational obstacles or to represent and mirror the developing concepts. Compiling different methods into a project language toolkit allows for a project specific answer to the problems at hand.

A PROJECT LANGUAGE TOOLKIT

The reason for providing a toolkit is to support a flexible situated way to care for the language side of software development. The means and methods provided each relate to a specific area of problems. However it is up to the specific project to identify the actual problem and to decide how, when, and in which combination to apply the proposed means and how to adapt them if necessary. Moreover, I do not claim completeness. There might be other methods fitting for specific contexts. The structure of the toolkit then might help to relate them to the problems at hand.

The toolkit consists of two parts: The first is meant to support handling obstacles in communication. The second compiles methods and notations that support the explicate (re)construction of concepts, both from the application domain as well as regarding the invention of new concepts for the future software. Communication – talking listening, reflecting, arguing, and so on – is the medium through which language develops. Whereas the (re)construction of concepts refers to a special aspect of language and other use of symbols. There oral or typographical representation allows to make an expression subject to reflection. To represent the language side of the development process supports a more reflected and conscious development of language without determining it.

Part I: Communicational Obstacles

The precondition for the development of a common language between developers and users is a working communication between them. That means: both sides are able to contribute their practices and their concepts; there is a mutual acknowledgment of the different skills; developers and users have to feel free to try out concepts they are unfamiliar with and make errors doing so. But reality is not necessary so.

Using concepts and methods from group therapy to improve communicational competence

Poor communicational abilities are often a problem even if the participants are willing to communicate. That influences the development and use of software. Only if all participants understand and agree on the solution, they can contribute to the whole in a consistent way. Therefore it is necessary to recognize design conflicts and solve them through symmetrical communication. In our societies we do not necessarily acquire those skills in a normal upbringing. Jürgen Pasch showed in his Ph.D. thesis, that even in pure computer scientist development groups, the ability to communicate and solve conflicts improves the quality of the solution considerably. [28] For the interdisciplinary development groups in participatory projects this applies even more. Pasch proposes methods from family therapy and supervision to help developing a symmetrical communication within the development group. [28, p. 167 ff]

Qualification of participating users regarding technical possibilities

Users often lack knowledge about technical possibilities. The result is, that they can not take them into account regarding the development at hand. The have to rely on the computer scientists' knowledge and biases. Specific, custom tailored qualification can help to develop the necessary competence. Such workshops should not focus on the teaching of computer science methods and notations, but on the technological possibilities and their integration into the work context There have been a lot of experiences around such workshops recorded in the context of participatory design. (See for example [14, 22].) An important issue is that the qualification has to be coordinated with the needs of the project and focus on practical, relevant aspects. The teachers should come from outside to support the development of an independent and critical competence. If there is a strong hierarchy between the different users, putting together status homogenous groups for the qualification sessions should be considered.

Ordinary language documents for design

Concepts and notation of the software development methods used by the software developers tend to dominate in the communication. This leads to a model monopoly of the developers. [4] The users are forced to express their contributions using categories that are not their own. To counter that situation development documents that are meant for discussion with the users

[3] Such means can be regarded as contributing to 'caring for a communicative culture' (german 'Gestaltbildende Projekttechniken', [12]), a set of organizational, communicational, and planning means developed by Christiane Floyd as part of her methodological framework for evolutionary and participatory software development, STEPS. [13] Guidelines to support of the communicational side of development are provided as well as means to provide a organizational frame building a sound base for the cooperation: Taking advantage of the different perspectives involved by communicating and crossing them; what organizational issues are important as a fundament for cooperation; how to build and maintain a network of mutual information and integration; about the importance of a common project language; defining roles and enacting them; practicing constructive criticism.

should be written in ordinary language, using the concepts from professional language and work language of the users. [5] gives an example of a set of documents related to a specific development method. This relation allows an easy 'translation' of use scenarios into class design. Other approaches are presented in [6]. A detailed glossary has to be developed together with these documents to allow feed back, if the developers are to understand not only the work practice but also the relevant concepts. Design related terms and the developing project language should be captured as well. An author-critique-cycle during the development of these documents helps ensure the mutual understanding.

Situation profile and discursive opening

Differences in status and power between the participants can bias the communication. Users whose perspectives are important for the development might not dare to speak up. In his Ph.D. Thesis about discursive requirements determination [1], Urs Andelfinger proposes based on Habermas' 'Theory of Communicative Action' a 'discursive opening' of the situation: A situation profile is developed presenting the participating actors, their roles, interests, values, and criteria of success, the interaction and communication culture and perhaps relevant aspects of methods, notations, and tools used for development. Related to this situation profile problems that might bias the communication and the implication on the project and its outcome are pointed out. The discursive opening is achieved through presenting the situation profile to the project group and discussing it. The group itself has to decide upon measures to change the situation.

Part II: (Re)Construction of concepts

This second part compiles methods that support the development of a project language by making the concepts and their development explicit for the group itself. From a language perspective, analysis and design methods that include a special notation can be regarded as providing a framework for the reconstruction and construction of concepts in a way that is fitting for their computer based modeling.[4] These methods do not reflect the biases they impose on what can be described and how it can be described. They are normally not fitting for a participatory process. Formal or semi formal notations require a training in mathematics or computer science to be applied. The methods compiled here try to avoid both disadvantages. Instead of biasing the language already in an early stage, the objective is to make it explicit and provide it as a resource for the further process. They start from the professional language and work language of the users. Of course these methods can be complemented with more technical or formal notations, or by other means of representation like rich pictures, mock ups, or prototypes.

Reconstruction of concepts

Methods for reconstructing concepts allow the explication of individual or group specific concepts as a basis for a discursive development of more intersubjective and conventional concepts. These conventional concepts are not to be mixed up with results of analysis. They can be regarded as a background for the design and development of software. Especially important for participatory design is the possibility to represent more then one perspective, more then one interpretation and use of concepts. Even if the implementation of software requires one interpretation that can be modeled in a formal language, this should be designed considering the variety of meanings that might exist in the use context.

Kelly-Repertory-Grids are proposed by Piepenburg to visualize personal interpretations of concepts as a base for further development [29, pp. 218ff] This method is especially useful for vague, varying and perhaps contradictory individual interpretations. Other visualization techniques he adapts from moderation techniques to mirror the actual stage of the discussion of what interpretation should be the base for the further development. [21, pp. 224ff]

Mathematical lattices can be used to visualize differences and connections between concepts. ([1, 35]) Thereby the relevant categories are introduced by the domain experts, i.e. the users. The lattices can be applied by a single person or by a group discussing the important differentiation. These reconstructions can be used for the design of a databases or the classification of big amount of data on the interface. However, this method might be most suitable for users who are familiar with mathematical or formal thinking.

A *'reference theory'* about the interpretation and usage of relevant concepts represented through a special kind of glossary is recommended by Reisin. [30] That reference theory should be expanded and maintained throughout the development process. It should not only capture the concepts of the work context but also their formal representation if they are to be modeled in the software. For the latter Reisin recommends a semi formal abstract data type like notation.

As Reisin emphasizes, the very representation of the concepts from the work language might change their interpretation: Representing them, making concepts explicit, makes them accessible to reflection as well. As all of the methods are developed to provide a foundation for further design discussions, all of them more or less provide a bridge to the construction of concepts that can be modeled in the computer application. However, as there is no straight forward relation between analysis and design, the construction of computer based concepts can be informed and supported by, but not derived through, the reconstruction of language.

Construction of concepts

The construction of concepts to be modeled in the computer application is often based on the usage of metaphorical concepts that have an ordinary interpretation as well as a formal one that can be implemented on a

[4] For a linguistic describing a knowledge engineering process as such a construction see [34]. Related to data modeling the subject is discussed in [22].

computer. 'Metaphorical descriptions can be used to emphasize some chosen aspects of a phenomenon or object while neglecting others, but never to explain the whole.' [26, pp. 234] Design metaphors mediate between the concepts in ordinary language and their formal language model for the computer application. Aspects of the use context are described as objects, methods, and classes, or as entity and relationships. If one is able to understand their computer based implementation, this kind of metaphors allow for the control of whether a concept is adequately modeled. They do not allow for the estimation of how the future software will effect the work practice.

Using fixed set of metaphors to guide analysis and design
Metaphors capturing the use perspective on a computer application have often more or less unconsciously informed the design. Maaß and Oberquelle identified a whole history of interface metaphors, putting the user in a related role: machines, like compilers, need a 'servant' to provide input and take care of the output, under a system perspective computer and users are more or less comparable components with different roles, and so on. [25]

In user centered software development, metaphors are used that describe the use aspects of the software in a way that regards the user as a competent actor. Züllighoven's software development according to the Tools&Material-Metaphor provides a framework for analysis, design, and implementation focusing on the user as a competent expert. [31] Together with the proposed documents this metaphor allows the discussion of the future design in a way that is meaningful from a use perspective.

Nonetheless, it provides a fixed pattern of recognition in analysis and a fixed set of categories for the construction of the design concepts which imposes a specific perspective on the use situation. This perspective is limited, it focuses on a single user with identifiable tasks allowing a flexible organization of work. Tasks requiring intense cooperation or process control can not easily be supported using this design metaphor. Even if the tasks to be supported allow for the application of such a given set of metaphors, the perception of the work practice as well as the creative side of design is constrained.

Developing design metaphors during the process
In [27] Laura Neumann and Leigh Star shared their experience and reflection in the context of their participation in a large distributed project aiming to build up the infrastructure for an integrated digital library. Metaphors are used to communicate professional backgrounds, interests and anticipations of different professional groups participating in that project. They call this net of metaphors a 'meta-language' [27, p. 232] and a 'boundary Infrastructure' [27, p.237], that allows for communication and cooperation throughout the project. In the same way, as this usage of metaphors helps to design the process [11], metaphors can be used to design the product.

Instead of using a fixed set of metaphors like in the above described approaches, metaphors for the display of the tasks and the interaction with the computer are part of the design process itself. Such in situ developed metaphors might fit the specific task better If the developed metaphors are used to guide the design, they can support the conceptualization of the software by its users. Project specific architecture patterns and implementation frameworks would can rationalize the implementation. The construction of the concepts to be modeled in the computer application would then be guided by the project specific metaphors. Metaphors and concepts are both part of the project language. Making them explicit allow for a consistent design fitting the requirements of the use situation.

CONCLUSIONS
The concrete result of the work described in this article is the project language toolkit, compiling a selection of different methods dealing with communicational obstacles and providing support for the development of a project language. It is based on an empirical example underlining the development of a common project language development as foundation for participatory design, and a theoretic discussion about language pointing to its creative aspects. The background for both is an understanding of software development that focuses on the co-development of technology and work practice.

Perhaps because the user's participation had been the most missing aspect in design, the discussion in the participatory design focuses mostly on the user's problems. However, if participatory development is only claimed to be better, but not also easier for the developers, it will not be applied in everyday development. Participatory design and development have to take the software engineer's work into account. In the observed development of a project language the importance of the merge between development related concepts and use related concepts became very visible. In the toolkit I tried to consider the needs of the developers as well as the users'. However, often much more seem to be known about the work of the users where the software should be embedded than about the work of the developers were the participatory design methods are to be embedded.

Software development is itself a skillful practice. Methods, notations, computer aided software engineering are the tools used in this work practice. Seldom the mainstream software engineering discussion or the alternative approaches consider the actual work practice of software developers. (See [2, 3] for exceptions.) More work in this area is necessary to develop methods that support participation of users and development by software engineers.

REFERENCES
1. Andelfinger, U. *Diskursive Anforderungsanalyse und Validierung – ein Beitrag zum Reduktionsproblem bei der Systementwicklung.* Dissertation am Fachbereich Informatik der TH Darmstadt, Darmstädter Dissertationen D17, 1995.

2. Button, G., and Sherrock, W. The Mundane Work of Writing and Reading Computer Programs. ???

3. Button, G., and Sharrock, W. Occasioned practice in the work of software engineers. in Jirotka, M., and Goguen, J. *Requirements Engineering. Social and Technical Issues.* London 1994

4. Bråten, S. Model Monopoly and Communication. *Acta Sociologica* 16 (2), 1973.

5. Bürkle, U., Gryczan, G., and Züllighoven, H. Object-Oriented System Development in a Banking Project: Methodology, Experiences, and Conclusions. *Human Computer Interaction* 10 (1995), 2&3, pp. 293-336.

6. Carroll, J. M. *Scenario-Based Design. Envisioning Work and Techynology in System Development.* John wiley & Sons, Inc 1995.

7. Dittrich, Y. , Heilbrock, J., Knickel, S., Löffler, A., and Savigny, P. v. Einsatz des World Wide Web zu Unterstützung asynchroner Zusammenarbeit in Softwareentwicklungsprojekten. in Krcmar, H., Lewe, H., and Schwabe, G. (eds.) *Herausforderung Telekooperation.* Berlin 1996.

8. Dittrich, Y. *Computeranwendungen und sprachlicher Context – Zu den Wechselwirkungen zwischen normaler und formaler Sprache bei Einsatz und Entwicklung von Software.* Frankfurt 1997.

9. Dittrich, Y. How to make Sense of Software. Interpretability as an Issue for Design. Submitted to the CSCW98.

10. Ehn, P. Scandinavian desgin: On participation and skill in:Schuler, D., and Namioka, A. *Participatory Design. Principles and practices* Hillsdale, N.J. 1993.

11. Floyd, C. Software development as reality construction. in: Floyd, C. et al. (eds.): *Software Development and Reality Construction.* Springer Verlag: Berlin 1992.

12. Floyd, C. STEPS-Projekthandbuch. Universität Hamburg 1992, and Arbeitsunterlagen zur Lehrveranstaltung 'Einführung in die Softwaretechnik.' Universität Hamburg 1994.

13. Floyd, C., Reisin, F.-M., and Schmidt, G. STEPS to Software Development with Users. in: Ghezzi, G., McDermid, J.A. (eds.) *ESEC '89.* Berlin 1989.

14. Floyd, C., Mehl, W.-M., Reisin, F.-M., and Wolf, G. Projekt PEtS: Partizipative Entwicklung transparenzschaffender Software für EDV-gestützte Arbeitsplätze. Endbericht an das Ministerium für Arbeit, Gesundheit und Soziales des Landes Nordrhein-Westfalen, Technische Universität Berlin 1990.

15. Fuchs, N. E., Schwittner, R. Attempo Controlled English (ACE), The First International Workshop on Controlled Language Applications, Katholieke Universiteit Leuven, 26-27 March 1996.

16. Holmqvist, B. Work and Perspective in: Andersen, P. B., and Bratteteig, T. (eds.) *Computers and Language at Work – The relevance of language and language use in development and use of computer systems.* Oslo Institute of Informatics, Research Report no. 126, ISBN 82-7368-030-4.

17. Humboldt, W. v. *Gesammelte Werke in fünf Bänden.* Edited by Flitner, A., and Giel, K., reprint of the 3. edition, Stuttgart 1995.

18. Jansen, K. D., Schwitalla, U., Wicke, W. *Beteiligungsorientierte Systementwicklung.* Westdeutscher Verlag, Opladen 1989

19. Keil-Slawik, R. Artifacts in Software Design. in: Floyd, C. et al. (eds.): *Software Development and Reality Construction.* Springer Verlag: Berlin 1992.

20. Kensing, F., Simonsen, J., and Bødker, K. MUST – a method for participatory design. in: Blomberg, J., Kensing, F., and Dykstra-Erickson, E. *Proceedings of the participatory design conference 1996* Cambridge Ma.

21. Kilberth, K., Gryczan, G., and Züllighoven, H. *Objektorientierte Anwendungsentwicklung – Konzepte, Strategien, Erfahrungen.* 2. Edition, Braunschweig/Wiesbaden 1994.

22. Klein, H. K., and Lyytinen, K. Towards New Foundations of Data Modeling. in: Floyd, C. et al. (eds.): *Software Development and Reality Construction.* Springer Verlag: Berlin 1992.

23. Krabbel, A., Wetzel, I., and Ratutski, S. Participation of heterogeneous user groups: Providing an integrated hospital information system. in: : Blomberg, J., Kensing, F., and Dykstra-Erickson, E. *Proceedings of the participatory design conference 1996* Cambridge Ma.

24. Lamnek, S. *Qualitative Sozialforschung.* 2 Volumes, 3rd Edition, München 1995.

25. Maaß, S., and Oberquelle, H. Perspectives and Metaphors for Human-Computer Interaction in: Floyd, C. et al. (eds.): *Software Development and Reality Construction.* Springer Verlag: Berlin 1992.

26. Naur, P. Programming as Theory Building. *Microprocessing and Microprogramming* 15(1985), 253-261.

27. Neuman, L.J., and Star, S.L. Making infrastructure: The dream of a common language. in: : Blomberg, J., Kensing, F., and Dykstra-Erickson, E. *Proceedings of the participatory design conference 1996* Cambridge Ma.

28. Pasch, J. *Software Entwicklung im Team – Mehr Qualität durch das dialogische Prinzip bei der Projectarbeit.* Berlin Heidelberg 1994

29. Piepenburg, U. *Semantikerstellung in der Softwareentwicklung- Qualitätszirkel bei der*

Anwendungsentwicklung. Dissertation. Fachbereich Informatik, Universität Hamburg 1994.

30. Reisin, F.-M. *Kooperative Gestaltung in partizipativen Softwareprojekten*. Frankfurt am Main 1992.
31. Riehle, D., and Züllighoven, H. A Pattern Language for Tool Construction and Integration Based on the Tools&Materials Metaphor. in: *Proceedings of the First Conference on Pattern Languages of Programs (PLOP '94), Monticello, Illinois, August 4-6, 1994,* Paper B.
32. Stamper, R. K. Social Norms in System Specification – an outline of MEASUR. In: Jirotka, M., Goguen, J. *Requirements Engineering: Social and Technical Aspects.* Academic Press 1994.
33. Suchman, L., Winograd, T., and others The coordinator debate, Computer Supported Cooperative Work 2(1994), pp. 177-190, and 3 (1995) pp. 31-95.
34. Weingarten, R. *Die Konstruktion von Sprache im technischen Medium.* Habilitationsschrift Fakultät für Linguistik und Literaturwissenschaften, Universität Bielefeld, Mai 1991.
35. Wille, R. Concepts Lattices and Conceptual Knowledge Systems. *Computers & Mathematics with Applications* 23 (1992), pp. 493-515.

1st of October, 1998