Research Report 19/97

How to make Sense of Software -
Interpretability as an Issue for Design

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—98/19—SE

How to make Sense of Software. Interpretability as an Issue for Design

by Yvonne Dittrich

ISSN 1103-1581
ISRN HK/R-RES—98/19—SE

Printed by Psilander Grafiska, Karlskrona 1998

How to make Sense of Software
Interpretability as an Issue for Design

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

ABSTRACT
In the context of CSCW – especially through ethnomethodological work place studies – the stability of particular work practices and therefore the ability to design software that fits with continually evolving work practices is questioned. This challenge for software development has been called 'design for unanticipated use'. Using the concept of interpretability, I attempt to answer this challenge.

A semiotic perspective on computer applications as formal symbol manipulation systems is introduced. A case study involving three alternative ways of using a computer application shows how users make sense of such symbolic machines. Wittgenstein's concept of language games is used as a 'figure of thought' to relate practice, language, and the use of symbolic machines. The development of an interpretation, fitting the implemented symbol manipulation and supporting the specific understanding of the task, remains crucial for competent use. Interpretability is introduced as a quality of computer applications. In order how to support the user in developing her own interpretation, a concept for help systems is described.

Keywords
computer supported cooperative work, interpretation, language games, usability, software development.

INTRODUCTION
Questions about the design of usable software for groups is at the heart of CSCW discourse. As Grudin states the problems often are not the lack of technical means but their interaction with cooperative activities in the use context. [9] Here we have to face the 'Designer's Dilemma' [2]: Through the implementations of technology to support a task we change the network of communication and activity by which this task is achieved. These arguments point to a fundamental challenge for the design of software: How to refer to the relation between computer application and the task at hand? How to design software for a specific use situation.

In the context of HCI a number of concepts have been developed to describe and reason about that relationship: For example the task adequacy of a computer application describes how far a computer application supports the fulfillment of a given task and transparency summarizes the qualities which allow the user to understand the relationship between the computer application and the task at hand.

In the context of CSCW and through ethnomethodological studies, which focus on use of computers, these concepts are questioned: For example people do not use computer applications the way they are designed. [21] The task is not fulfilled in a uniform anticipatable way. People adapt their plans and their typical way of doing things to the contingencies of the situation at hand. Plans are resources for situated action. [24] What is perceived as a task changes and requires an ongoing 'articulation work' to maintain an understanding of what is to be done under what circumstances. [7][1]

The relationship between the computer application and *the* task becomes problematic as the task is not fixed. Task adequacy, transparency, and similar concepts can no longer be regarded as qualities of the computer application, but have to be regarded as attributes of a special use situation. (See e.g. [16]) The designer is left in a problematic situation: She can not be sure about the usability of her product even if some hints, such as using design metaphors or guide lines for consistent interface design could be followed.[2]

The discussion about consistency as a design quality demonstrates this dilemma: Jonathan Grudin put forward a case against consistency and argued that consistency has to be subordinated to the task adequacy of the software. [8] For the use of metaphors the same can be claimed. However, as task adequacy can not be regarded as an attribute of a computer application, no other criteria than the evaluation of the specific use situations can be used as guidance for design. This is one of the major arguments for an evolutionary development that regards the use of preliminary versions as part of the codevelopment of work situation and computer application. [6] The application of

[1] It is not surprising that these phenomenon show up in the context of CSCW: The variation of interpretation and use is not a problem when I am using my PC alone in my office. It just becomes visible and an issue when a group is using the program together. Then an at least compatible interpretation and use has to be developed.

[2] Some Authors distinguish between usability, describing use oriented qualities of software that are independent of a specific use situation and the usefulness of a computer application in such a situation. (E.g. [1].) However, it sounds a bit strange for me to talk about usability independent of an at least imagined use context. It is like talking about hammers without knowing about nails. So I do not use that distinction here.

design possibilities such as the use of metaphors, consistency, its purposeful neglection, depend on the concrete project and must be evaluated in relation the use situation.

However, even with an evolutionary approach the further development of the use context, or the use in another context is not considered. The challenge of the requirement for 'Design for Unanticipated Use' [21] reaches deeper. Just as we can not anticipate the specific use situation we can not design software that fits completely with the work situation. Nonetheless, people manage to use programs and make sense of software. How can we support them in bridging the gap between the application and their use situation?

This article tries to answer this challenge by referring to the observation of unanticipated use. It argues for a perspective on computer applications as symbolic machines that have to be interpreted to be used (section 2). How then is such an interpretation developed? How do users relate the symbolic machine to their task at hand? A case study shows that such an interpretation depends not only on the computer application but also on the users' perception of their task (section 3). Three student groups used the same computer application for the same assignment in different and unanticipated ways. The interpretation and the usage developed by each group can be related to their corresponding understanding of their task. Wittgenstein's concept of language games is used as a figure of thought, in order to deepen the understanding of what has been observed (section 4). The symbolic perspective introduced in section 2 allows me to make use of the observations regarding the design of computer applications (section 5). Making use of the concept of interpretability, design possibilities that support the users' development of their own interpretations are discussed (section 6). As an example, a new concept for help systems allows users not only to understand the software but to discuss and capture their own interpretation. The article concludes by putting forward some related research questions.

As the reader might have recognized, I am writing this article as a computer scientist. Even though I use qualitative social science methods and base my arguments in philosophy of language the purpose is to improve the development of computer applications, technical artifacts that constrain certain ways of use and open up others. To achieve that I have to relate different scientific perspectives. Where I recognized the need I explicate the necessary changes in perspective.

COMPUTER APPLICATIONS AS SYMBOLIC MACHINES

Consideration about the interpretability of software benefits from a perspective on computer applications as signs or sign systems. From a computer science perspective this is nothing special. Computer applications can be regarded as the implementation of formal symbol manipulation, i.e. mathematical calculus on a technical device.[3] From a theoretical computer science point of view this means that every computer application can, in principle, be described in a mathematical notation. Focusing on the use of computer applications, other qualities of mathematical calculus become relevant: A calculus consists of finite sets of signs and rules which can be used to form and transform an infinite number of patterns of signs. In her book about the development of the idea of mathematical calculus Sybille Krämer points out, that symbols in a formal calculus only relate to other signs or to the rules of transformation. The application of the calculus relies solely on this internal meaning.[4] Additionally, the application of such symbolic machines depends on typographic symbols that are manipulated in a schematic way: Typographic symbols are unambiguously distinguishable so that the rules can relate to their attributes and their arrangement. [11] From the use point of view, the signs become independent of the temporal context. The manipulation of the typographic symbols follow a schema. They are only meaningful for producing the result. The manipulation of numbers in typographical multiplication are meaningless if taken out of this specific context. However, the implementation of a schema is independent of who or what is implementing it. It does not matter if it is a man, a woman, or a computer. Using formal language we distinguish between the manipulation of the symbols according to the rules and their interpretation. As long as we are manipulating the symbols according to the symbolic machines we do not have to refer to their possible interpretation. The laws of multiplication are applied to numbers, but are independent of what the figures stand for. For example, for calculating the price of groceries or developing some new weapon.

Using such a symbolic machine or its implementation on a computer depends on developing of an interpretation of the symbols and their manipulation regarding the situation at hand. Although the calculus rigorously defines the relations between the signs, it does not imply how they are interpreted. The recontextualisation of the decontextualised manipulation of symbols is, as I will argue, a creative achievement by the users. The following section describes a case study that highlights the development of such interpretations.

THE TALES OF THE THREE STUDENT GROUPS

To establish a concrete example of the use and interpretation of a symbolic system, three student groups using the same computer application were observed and interviewed. They were asked to perform a requirements analysis for the same fictional example in the context of their software development course.[5] As a method they

[3] Winograd and Flores give a comprehensible description of the hierarchy of abstract machines where each level can be regarded as an implementation of the next higher and an abstraction of the next lower one. The lowest is the actual hardware, the highest the application as perceived by the user. They also discuss the limits of that perspective. [27]

[4] Therefore applying a formal calculus the signs are transformed in a regularity, that otherwise is the property of machines. Regarding that history, the development of the mathematical model of a computer before any real computer existed is not surprising. [25]

[5] The case study took place in the winter term 94/95 at the University Hamburg, in the context of the course 'Einführung in die Software Technik'. The students had to work in groups on all the assignments. For more information about the case study see [4].

used task nets, a graphical notation to describe the relationship between functional roles, tasks, and accessed objects, that could also be used to design the embedding of the future computer application. They began to work on their assignment by using only pen, paper, and boards. After two weeks they began to use the task net editor, a graphical editor that allowed them to draw the objects used for the modeling. Additionally it implemented some functionality related to the rules of how to apply the task nets.

The case study was carried out according to [12]: The students were observed and taped three times: during their first work meeting for the task net assignment, the first time they used the computer tool and one of the last meetings working on their task nets. They were interviewed just before they started to use the editor and a second time in connection with the last observation. The tapes of the observations and the interviews were transcribed. The detailed evaluation in [4] is based on the transcriptions, additional documents concerning the editor, and the handouts for the software engineering course related to the assignment.

During their learning phase all groups recognized the computer application as a sign system to be interpreted: They asked for the meaning of symbols at the interface and answered them correspondingly. 'The dotted line (representation) means you have to put something in there.'[6] They reasoned about how to 'represent' the connection between various nets.

For computer scientists the most striking result of the study was that all three groups used the computer tool in a completely different way and none of the groups used it as intended by the developer. The analysis of the tapes showed that this was due to different ways the task nets were developed by the groups.

Due to the lack of space I will only give one example. Using a special 'copy'-function of the task net editor one could create various representations of the same task net object. This functionality was meant to help keep consistency in a set of drawings describing an interrelated set of tasks on different levels of refinement. The same task net object appeared in different drawings. The name – and other attributes – of such objects were frequently changed. Using the special 'copy'-function and other related functionality these changes could be performed once for all representations. Changing the name of a task in one of its representation would change the inscriptions in all others as well.

One of the groups talked about the task nets as ordered in levels; the task nets could be organized in a way so that each level describing a further refinement of a high level representation of interrelated tasks. Additionally, they thought about the different representations in a hierarchical order. The most abstract representation was the defining one. The other 'levels' depended on this one. When they found out about the special 'copy-function', they interpreted – and used it – as if designed to keep consistency across their various levels of refinement. During the last session using the editor, they recognized that their hierarchical relation between the different representations of one object was not supported by the system: They tried to delete one object with all its representations by deleting the representation on the highest level. However, that was not possible. They finished their work a bit disappointed by the editor.

A second group did not use the possibility of distributing the various refinements on different drawings. Everything was represented in one drawing. For aesthetic reasons they were looking for the possibility of producing symbols of a similar size. They found the 'copy'-function and tried it out. Recognizing the dependency between the resulting symbols, they regarded the function as error-prone. A bit later they found out about another function that was designed to disconnect representations of the same object and used that to get independent objects of equal size. Nonetheless, they regarded the whole idea as bad design.

A third group used the special copy function without problems after some initial difficulties. However they were not satisfied with it: They perceived a special arrangement of objects as a unit and wanted to refine them together. However, that was not supported by the editor. One had to copy and paste the objects one by one. They regarded that as an unnecessary effort.

Each of the groups used the task nets in a specific way; each of them developed additional notions and guidelines – for example the different levels of refinement, the similar granularity of tasks to be modeled on each level –, or changed the meaning of existing terms. However, all of these different ways of using task nets were consistent with the definition of the method, even though each of them resulted in a different interpretation and use of the task net editor. The editor constrained its interpretations. Interpretations that did not fit the implemented functionality were falsified by unexpected states of the application. But it did not force one interpretation either. The competence usage was bound to the ability to develop an interpretation that fitted the functionality of the computer application *and* was suitable for the particular way of using the task nets.

It is not surprising that methods – even related to computer science – are interpreted and used in a way that takes into account the situation at hand. (See e.g. [17], [6], [3]) The point of this case study is that the groups related the same computer application to their different interpretation of notions and guidelines. They interpreted and used the computer application to support their group's specific understanding of the task. Each of the groups made sense of the computer application in a different way.

Interpretation can be seen as a means to bridge the gap between the actual design – that implements the developers' understanding of the use possibilities – and the users' understanding of the task at hand. Support for interpretation would therefore mean support for unanticipated use. The following section introduces Wittgenstein's concept of language games as a figure of thought in order to understand more about the interaction between the user's language and practice, and the symbolic machine in the use situation.

[6] The original German was, 'Wenn das gestrichelt ist, das heißt dann, daß da noch was reinkommt.'

LANGUAGE GAMES AND THE USE OF COMPUTERS

The possibility of big variation in interpretation indicates that the computer application does not have the main influence in its use. The case study showed further that the interpretation could be related to the users' understanding of the task, to their conceptualization and their practice. But how can that relation be understood? How can we understand more about the role of the computer in order to create a better design?

Using Wittgenstein's concept of language games as a 'figure of thought', the different ways to apply task nets and to talk about them can be described as different language games. The usage of the computer application can be described as causing a change in already existing language games. The computer application itself can be regarded as constraining this language game. To explain this step let me briefly introduce Wittgenstein's concept of language games.[7]

In his Philosophical Investigations, Wittgenstein argues for a different perspective on language. Using metaphors, examples, and pictures, he invites the reader to look at the ordinary language as it occurs in everyday situations. He emphasizes the idea that language is intertwined with pragmatic action and both are relying on a common way of life, a common established practice. Terms, and utterances do not carry a meaning around with them, but become meaningful through their use in a special situation, related to a specific context. Wittgenstein uses games, like chess, as a metaphor to describe the relation between language and practice [28, PI 31][8]: Words, like chess pieces, become meaningful through their role in the game. Each move follows a certain goal, relying on the rules of the game as a means to define and achieve that goal. Utterances do the same regarding the rules of language use in a specific situation.

Games as a metaphor does not apply in every respect; there is no predefined goal, and language does not have a fixed set of rules. Language games change. 'There are *countless* kinds: countless different kinds of use of what we call 'symbols', 'words', 'sentences'. And this multiplicity is not something fixed, given once for all; but new types of language, new language games […] come into existence, and others become obsolete and get forgotten.' [28, PI 23] Wittgenstein does not write about the reasons for change of language games. He is not interested in the 'why'. We change the rules while we are playing the games. 'And is there not also the case where we play and – make up the rules as we go along? And there is even one where we alter them – as we go along.' [28, PI 83] However in [28, PI 142] he clarifies his perspective of the relation between language and the world: Language games correspond to a successful way of acting in the material world. So if we change the material world or our way of interacting with it – e.g. through the development of technology –, our language changes too.

[7] To assure that my interpretation is not pure fantasy I relied on [23], the German commentary on Wittgenstein's Philosophical Investigations.

[8] The first part of the Philosophical Investigations is cited by their numbers (PI Nr), the second is cited using the page numbering of [28]

As I argued in the first section computer applications can be regarded as technical implementations of formal symbol manipulation. Wittgenstein describes logic and mathematics as special language games. They differ regarding the way rules are used and the role of meaning. In order to relate them to normal language games, let us have a closer look at rules and meaning related to language games.

In the context of normal language games rules enable mutual meaningful action of the 'players'. In a social setting we develop a special but common way of practice and language. This enables us to communicate and coordinate our action. Even 'breaking the rules' of language and action, acting and talking not to the established practice, becomes a means of communication. The rules of language are based in a regularity developed by a community. Following rules or 'breaking' them meaningfully becomes an elaborated practice in itself.

The meaning of a term is not independent of it's use; it is defined by it. As every term might be used in different language games, it has different but related meanings. Wittgenstein uses his metaphor of family resemblance here: '…for the various resemblances between members of a family: build, features, colour of eyes, gait, temperament, etc. etc. overlap and criss-cross in the same way. – And I shall say: 'games' form a family.' [28, Pi 67, see also Pi 66]

As Wittgenstein used language games as a metaphor there is no clear definition what is to be regarded as a language game. He gives examples of different granularity [28, PI 23]. Therefore, to apply the concept on a concrete stituation means to describe language use and practice in a consistent way as a language game.

The three student groups, trying to understand task nets and using the graphical notation on their example, can be described as the developing three different language games. They tried to make sense of the method, interpreting the symbols and the rules for applying them, adding additional concepts and rules – e.g. one group invented different levels of refinement that should comprise tasks or subtasks of comparable complexity. This development of language and practice was necessary to apply the method in a for them meaningful way. But how does the use of the task net editor, their specific symbolic machine relate to that?

Wittgenstein writes as well about logic. Formal logic and mathematics are a special kind of language game and not an ideal for normal language use. They are unambiguously encircled by explicit rules. [28, PI 100-101] They use symbols that have rigorously defined meanings. Wittgenstein explains in an argument with a fictious opponent the relation between strictly defined concepts, and their normal language siblings: 'The kinship is that of two pictures, one of which consists of colour patches with vague contours, and the other of patches similar shaped and distributed, but with clear contours. The kinship is just as undeniable as the difference.' [28, PI 76]

So why apply that kind of language game that is so different from our ordinary language use. An explanation might be found from the second part of the Philosophical Investigations, although the text does not explicitly point to formal language games: 'What does man think for? What use is it? – Why does he make boilers according to

calculations and not leave the thickness of their walls to chance? After all it is only a fact of experience that boilers do not explode so often if made according to these calculations. But just as having once been burnt he would do anything rather than put his hand into a fire, so he would do anything rather than not calculate for a boiler.' [28, PI 466] We use formal calculi – arithmetic – because experience showed that they are useful in specific situations. They allow a more sound construction of technical artifacts like in Wittgenstein's example, a more rigorous coordination, better control over nature, technology or even over social relations.

How do we use them then? How do we embed them in our everyday's life. An example might be the use of typographic multiplication on a market: Through various action – measurement of the goods, copying of the price per unit – the conditions for the application of that specific formal language game are constructed. Then the formal language game is applied. Symbols are transformed according to the rules of that context independent symbolic machine. The result is then again used in the normal language game of selling and buying goods. It might for example be rounded to do the customer a favor. The formal language game of typographical multiplication 'constrains' the normal language game in so far as it requires a certain input to be applied and its output has to be interpreted regarding the actual situation. However, it does not define the embedding language game.[9]

The use of computer applications is not always as simple as the market place example: There is not only an input and an output, but often a modeling structure and its manipulation, that has to be mapped to the situation at hand. And as users normally do not implement the symbol manipulations they do not know that much about it. Not only did the three student groups have to develop a fitting interpretation, they had to find out about the formal symbol manipulation as well. The signs at the interface and the algorithmic relations between the signs had to be understood and put into relation to their previous perspective on task nets. Together with the interpretation the language game of 'applying task nets to model a work situation' developed into the language game of 'applying task nets using the computer application'. The difference in language games before implied different language games while using the computer application. The task net editor constrained the language game: It required a special kind of interaction, e.g. one can not refine a group of objects at once.[10] Users had to make sense of the special qualities of the model that was built up through that interaction, e.g. the connection between different representations of the same object. Nonetheless the task net editor did not define the interpretation the groups developed, nor the evolving embedding language games.

The development of a language game is not predictable. It has to be regarded as a creative achievement by the users. Additionally, an existing and working language game, with a computer application embedded would change. What does that mean regarding the design of computer applications for single users as well as for groups?

THE FREEDOM IN SIGNS

Before we go on to discuss design issues and a specific example motivated by the case study and its evaluation let us reflect on what we have achieved so far. I introduced a perspective on computer applications as symbolic machines, as decontextualised symbol manipulation mechanisms. To apply a symbolic machine in a concrete situation, it has to be interpreted by its users. The case study showed that this recontextualisation is constrained but not completely defined by the application. To understand how the three student groups made sense of the software, I used Wittgenstein's concept of language games and showed how different interpretations and, by implication, different usage patterns, related to users' task specific language and practice.

The great variation in the observed ways of interpreting and using the computer application could be related to their former language games about using task nets. From the design perspective it is interesting that none of the groups used the tool as the developer intended.[11] So how then are design and use of software related?

The idea that technology is influencing but not defining its use is not new. Orlikowski, for example, argues from a sociological point of view for an 'interpretive flexibility of technology'[12]:'I will use the term *interpretive flexibility* […] to refer to the degree to which users of a technology are engaged in its constitution (physically and/or socially) during development or use. Interpretive flexibility is an attribute of the relationship between humans and technology and hence it is influenced by characteristics of the material artifact (e.g., the specific hardware and software comprising the technology), characteristics of the human agents (e.g., experience, motivation), and characteristics of the context (e.g., social relations, task assignments, resource allocations).' [19, p. 409] However this 'interpretive flexibility' is not infinite. It is constrained by the organizational context as well as by the material character of the technical artifact itself.

[9] For a wonderful example for the creative and purposeful 'breaking' of rules imposed by a computer application see [26]. In that special context the 'breaking' of the rules in special situations becomes a rule itself. Here the restriction of the software allows to communication of specific circumstances by not following them, enabling a special language game.

[10] It influenced as well the cooperative aspect of the language games: One group explicitly complained about the mouse-monopoly. They had a very democratic way of communication. That was not supported by a computer application allowing only one of them to interact with the application.

[11] Dirk Riehle who designed the tool as an illustration for his development of a pattern framework for the design and implementation according to the Tools&Materials Metaphor [20] was still in Hamburg during the case study.

[12] [19] Orlikowski uses the concept of interpretation to describe a more general phenomenon: the meaning assigned to a technology or technical artifact as a whole, the images that influence the imagination about its embedding and use. [19, p. 410]. For a philosophical version see [22].

Whether computer based technology has a higher interpretive flexibility because of its symbolic character – as Orlikowski hinted at [19, p. 421] – is still an open question. According to my argumentation, at least for socially embedded computer applications, I support the notion that the symbolic character is responsible for a special kind of interpretive flexibility: The hardware is not used to transform part of its environment in a material way but to implement a symbolic machine, a mathematical calculus. The result of the application is not a material effect that has to be anticipated, but a symbolic structure that requires interpretation.

So the development of an interpretation is necessary to use a computer application. On the other hand, a symbolic relation can not be determined by the design in the same way as a material relation: The relation between a symbol and its meaning is dependent on creative interpretation by humans. Using it in a different way only requires a reinterpretation – not also a physical rearrangement. The case study is an example of this special kind of interpretive flexibility. It demonstrated in detail how the students, their understanding of the assignment, and their interpretation of the method influenced their interpretation of the symbolic machine and vice versa.

The symbolic and not materially implicated relation between a computer application and its use context, offers a great freedom in *design*. Which aspect of the area of interest is modeled in which way is up to the developer. She can even provide symbols and manipulation about the symbolic machine itself, in order to provide information about what is going on [5] or even to adapt the predefined ways to act to the situation at hand [10].

On the other hand, even if one can anticipate the use during design, there is no way to enforce the anticipation through the design. One can not design away the necessity of interpretation and with it the users' freedom of creative interpretation. We can not define the meaning of our signs in real world situations in the same way as we can control their manipulation inside the computer.

That implies that the question for design has to change. Even as the orientation at the future use is still recommended, total task adequacy, striving for a design that fits perfectly for a specific work place becomes doubtful. In his article 'Design for unanticipated use' Robinson discussed certain design aspects that support more flexible use of applications. He mentions several examples; minimizing enforced sequentiality, providing overviews and supporting peripheral awareness for quick orientation after doing something else, supporting implicit communication through the computer application as well as an explicit discussion about its usage in order to support double level language.

On a more abstract level the question about the relation between task specific support and flexibility of use becomes an issue: What kind and how much support to provide and what to leave to the user? I will take up these questions in the conclusion again.

However, as the case study showed, competent use depends on an interpretation which enables the user to make sense of the defined(!) and hence fixed symbol manipulation. The next section introduces the concept of interpretability to explore how to support the development of a fitting interpretation.

INTERPRETABILITY AS AN ISSUE FOR DESIGN

So far, the case study and its discussion has led to a from a software development perspective somehow pessimistic view. There is no longer a goal, a perfect solution to be reached. Instead of a rule to follow and measurements about deviation from the optimum we have to accept that there is no one best way. According to the case study the relation between the computer application and a specific use situation is a creative achievement of the users. One must anticipate the trade off between usability regarding a specific situation and flexibility to support change and unanticipated use.

The question is: How can we support the user in embedding our (hopefully) good solution according to her needs? How can we support her in developing an interpretation and language games around the computer application? As we can not control the users' perspective on the task or their way of relating the computer application to what they want to achieve, we can only tell about the software itself. As the design is based in anticipation of a use situation, we have as well to explain this anticipation in order to help the users understand not only what is there, but also why it is like it is.

To make that point, I use the concept of 'theory building' in software development introduced by Peter Naur [18]: To develop a program software developers built up a theory, that allows them to explain and reason about the design. This kind of theory is necessary to develop and change the program in a consistent and defensible way. Part of this theory is about how the program should be used in an anticipation of the use situation. [18, p. 256] That means every computer application is 'tuned' for an anticipated use situation, and for a special interpretation. The idea is to explicate the anticipated use so the user can asses how far her situation, her perspective matches. However, mismatches can not be avoided. Explicating the anticipated use also assists the user in interpreting the symbolic machine so that she can use it nonetheless.

Relating these ideas to Suchman's 'Plans and Situated Action' [24] sheds more light on the idea. The anticipation of the use situation can be regarded as a plan which informs the software development. The development results in a technical artifact that introduces a specific arrangement of support and resources as well as constraints into a use situation. The information about the relation between the anticipation of use and the design provides a necessary resource for the situated use of this constraining support.

This explication can be done implicitly providing 'maps'[13] about the application and about the anticipated use possibilities. There are existing techniques that can be used this way: Paul Dourish proposed to use object oriented technology to let the computer application give 'accounts' about its operation. [5] He used copy machine

[13] [John Hughes, personal communication] In a discussion about the topic of the article John Hughes used the metaphor of rough maps provided in tunnels and trains of a subway system to enable the commuters to identify their location and to relate it to their knowledge about the city.

interfaces as an example. In case of an error, the machine should display its actual state in order to enable the user to reorder his copy job. From a computer science perspective this remains a simple example. Nonetheless, it was subject to discussion, on which level of abstraction the machine should be represented.[14] Regarding the task net editor this question would be even more difficult. There is no guideline, what aspect of a computer application should be accounted for, and on what level of abstraction.

Another possibility for such maps are representations on a meta level telling about the operation of the system and making parts of it available for change. An example is Guido Gryczan process patterns that change the character of workflow systems in a fundamental way; the process descriptions that control the transport of documents between users are made available for the one working at a task. She is able to look at the history of a process and can change the description for the future work at it.

'Accounts' of computer applications as well as the users' control over the process descriptions of a workflow system are design ideas using technical possibilities to provide the user with information about the operation of a computer application. However, there is still a lot to discuss about it. Another approach is to provide explicit explanations, e.g. in a help system focusing on explaining the computer application and its intended relation to the anticipated use context.

In the context of the case study a help system for the task net editor was developed using the observations of the case study to explore the requirements:[15] It does not in the first place focus on the anticipated tasks of the user, but on explaining the objects one can manipulate and the functionality provided in the menus. The help texts are organized as a hypertext. As the task net editor was designed according to the Tools&Materials approach [20] we decided on texts about the implemented model of task nets and texts about the manipulating functions. Texts, describing complex task net related concepts like refinement and relating them to the model and its manipulation were added. The symbols of the symbolic machine and their transformations are related to the language game they are developed for.

To support the development of one's own interpretation, the users can add bookmarks and their own text in form of user defined hypertext knots. These texts can be shared among users. The feature allows to present and capture deviating usage and interpretation.

One might argue that this approach would impose the developer's anticipation of use on the users. This argument does not hold for the users we observed in the case study. They formulated design proposals for further development of the task net editor – oriented of course to their own way of developing task nets. In other contexts writing texts that express respect towards the judgment of the users, and the obvious where there is the possibility of changing the hypertext would probably encourage the users' deviating use and interpretation.

The proposed help system points forwards in a new direction. It does not claim to know what tasks the user will perform with the help of the software, nor how he will or should use it. It is designed to support the unanticipated use by explaining the computer application and its relation to the anticipated use, and through this explanation providing a sound base for the users' own interpretation.

SUMMING UP

The major result of the article is a change in perspective on the goal of use oriented software development. Instead of striving for a better and better fit between the software and the changing use context, the point I argue for is to accept a gap between the formal language model of the computer application and the situation at hand. The issue then is to support the user in bridging the gap.

The case study showed that users develop an interpretation of the computer application in order to use it as support for what they perceive as their task. Using Wittgenstein's language games as a 'figure of thought' this can be described as the development of an embedding ordinary language game based on the previous language and practice of the users.

The focus on interpretation as a precondition for competent use is supported by a semiotic perspective on computer applications; they can be seen as formal symbol manipulation implemented by a machine. This symbolic machine is independent of the use context. To apply it in a specific situation it has to be recontextualized by the user; it has to be interpreted.

From a software development point of view this leads back to the contradiction of 'design for unanticipated use'. How can the developer take something into consideration, that she is not anticipating? The symbolic perspective opens up new possibilities; the connection between the implemented formal symbol manipulation and the anticipated use can be provided as a resource for the user to develop her own interpretation. Techniques from literature and an innovative concept for help systems has been discussed as examples how to improve the interpretability of a computer application.

This change in perspective points to a new set of research questions as well:

How to handle the trade off between specific support and flexibility for change? This raises new questions for software development as well as for work place studies. What is relatively fixed, and where should the user be able to put forward her interpretation? How to model for flexible use? Perhaps results from the field of domain specific software development can be used here.

The second set of questions is related to explicit and implicit ways to represent information about a computer application to the user. What aspects are important? On what level of abstraction? How to represent that information? Work place studies about how users conceptualize

[14] Technical embedded systems have the special problem that they are connected – in a very material way – to a machine or a set of machines. So the argumentation in this article would have to be rethought in some aspects to be applied here. However, technical embedded systems are not the subject of this article. Here I use only the idea of accounts about the state of an application.

[15] The following description of the help system is rather short due to the special focus of the article. It is described in [15] focusing on its 'unanticipated use' in the context of participatory design.

software would be helpful as well as ideas regarding orientation and overviews in hypertext.

In the context of development of CSCW–applications supporting unanticipated use would include supporting for a group of users in developing a compatible interpretation of the common application. Features that allow discussing and capturing of interpretations are crucial regarding this issue. The article contributed in pointing out the need and in giving an example how such a support can look like.

ACKNOWLEDGMENTS

I would like to thank the different persons who encouraged me and helped me to formulate the argumentation of this article. John Hughes proposed to write an article after discussing some aspects of the case study and the 'strange' application of Wittgenstein's investigations. He read a short draft of it and gave helpful comments. Ralf Klischewski, from my former department at the University of Hamburg, and Olle Lindeberg, computer scientists here in Ronneby, read and commented early versions of the paper.

Jeanette Blomberg and Bo Helgesson made it possible for me to participate in their post graduate course Work Practice and Technology last term in Ronneby without being a student here. They as well as the other participants of the course gave helpful comments.

REFERENCES

1. Blomberg, J., Suchman, L., and Trigg, R.H. Reflections on a Work-Oriented Design Project. *Human Computer Interaction 11* (1996), 237-265.
2. Button, G., and Dourish, P. Technomethodology: Paradoxes and Possibilities. in *Proceedings of the ACM Conference on Human Factors in Computing, CHI '96,* (Vancouver BC Canada '96), ACM Press, 19-26.
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. Dittrich, Y. *Computeranwendungen und sprachlicher Context – Zu den Wechselwirkungen zwischen normaler und formaler Sprache bei Einsatz und Entwicklung von Software.* Frankfurt 1997.
5. Dourish, P. Accounting for Systems Behaviour: Representation, Reflection and Resourceful Action. in *Proceedings of the Conference 'Computers in Context: Joining Forces in Design,* Aarhus, Denmark 1995.
6. Floyd, C. Software development as reality construction. in: Floyd, C. et al. (eds.): *Software Development and Reality Construction.* Springer Verlag: Berlin 1992.
7. Gerson, E.M., and Star, S.L. Analyzing Due Process in the Workplace. *ACM Transactions on Office Information Systems 4* (1986), 257-270.
8. Grudin, J. The Case Against User Interface Consistency. *Communications of the ACM 32* (1989), 1164-1173.
9. Grudin, J. Computer Supported Co-operative Work: History and Focus. *IEEE Computer, 2,* (1994) 5, 19-26.
10. Gryczan, G. *Prozeßmuster zur Unterstützung kooperativer Tätigkeit.* Wiesbaden 1996.
11. Krämer, S. *Symbolische Maschinen*. Frankfurt am Main 1988.
12. Lamnek, S. *Qualitative Sozialforschung.* 2 Volumes, 3rd Edition, München 1995.
13. Lehmann, M.M. Programs, Life Cycles, and Laws of Software Evolution. in *Proceedings of the IEEE* 68 (1980), 1060-1076.
14. Lilienthal, C. Konzeption und Realisierung eines an der Anwendungssprache orientierten Hilfesystems nach der Werkzeug-Material Metapher. Diplomarbeit am Fachbereich Informatik der Universität Hamburg, Juni 1995.
15. Lilienthal, C., and Züllighoven, H. Techniques and Tools for Continuous User Participation. in Blomberg, J., Kensing, F., and Dijkstra-Erickson, E. (eds.) *Proceedings of the Participatory Design Conference, 13.-15. Nov. Cambridge, MA 1996*, 153-160.
16. Maaß, S. *Transparenz – Eine zentrale Softwareergonomische Forderung.* Bericht des Fachbereichs Informatik der Universität Hamburg, FBI-HH-B-170/94.
17. Mathiassen, L. *Systems Development and Systems Development Methods*. PhD Thesis, Computer Science Department, Aarhus University (in Dennish), 1981.
18. Naur, P. Programming as Theory Building. *Microprocessing and Microprogramming 15* (1985), 253-261.
19. Orlikowski, W. The Duality of Technology: Rethinking the Concept of Technology in Organizations. *Organization Science 1* (1992), 398ff.
20. 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.
21. Robinson, M. Design for Unanticipated Use. in DeMichaelis, G., Simone, C., and Schmidt, K.: *Proceedings of the Third European Conference on Computer Supported Co-operative Work, Milan, 1993.*
22. Rohbek, J. *Technologische Urteilskraft. Zu einer Ethik technischen Handelns* Frankfurt am Main 1993.
23. Savigny, E. von *Wittgensteins „Philosophische Untersuchungen", Kommentar für Leser.* 2nd edition, 2 Volumes, Frankfurt am Main 1994.
24. Suchman, L. *Plans and situated actions. The problem of human-machine communication* Cambridge: Cambridge University Press 1987.
25. Turing, A. On Computable Numbers with an Application to the Entwcheidungsproblem. *Proceedings of the London Mathematic Society, Ser. 2, Vol. 42* (1936-1937), 230-265, corrections: *ibid, Vol. 43* (1937), 544-546. Repr. in: Davis, M. (ed.): *The*

Undecidable. Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions. New York 1965, 289-291.

26. Whalen, J. Making Standardization Visible. ?

27. Winograd, T., and Flores, F. *Understanding Language and Cognition*. Ablex Publishing Corporation 1986.

28. Wittgenstein, L. *Philosophical Investigations*, New Yorck 1958. Cited by the author numbering, if not marked otherwise.