# Talking Design

## Co-Construction and the Use of Representations in Software Development

Yvonne Dittrich, Kari Rönkkö
Yvonne.Dittrich@ipd.hk-r.se, Kari.Ronkko@ipd.hk-r.se
University of Karlskrona Ronneby, Department of Software Engineering and Computer Science

### Abstract

*Software development differs from other design work insofar as the object to be designed is not visible. Representations play an important role. Even as they only describe aspects of the later software, they mediate the common design work. Software engineering literature focuses on persistent representations, documents, diagrams, mock-ups, or similar things. Our article puts 'talking design', where the software is represented in utterances, sounds, and enactment, in the centre. With the help of concepts from the CSCW discourse, we conceptualise what is happening here; the collaborative object for the design talk is not given, it has to be collectively constructed. Software development can be regarded as routine co-construction. In our case the protocol of that design meeting seemed to serve as a reminder for the participants rather than as in itself representing the design decided upon. The design meeting, we focus in this article, was part of a distributed software development project, with a larger project situated in Ronneby, Sweden and a smaller one in Oulu, Finland. If important parts of design are collectively constructed during such meetings, what does that imply for co-operation, co-ordination and division of labour in software development projects? How can a common practice be developed among distributed work groups?*

**Keywords:** Co-construction, representations, talk, software development as work

**BRT Keywords:**

# Introduction

Software development is in most cases a highly co-operative type of design work. Software is developed by project groups in often very intense co-operation. CSCW is concerned with co-operative work and possible computer based support for it. In this article discussions from both areas are related to make sense of what happened during a design meeting among software developers. From a software development point of view, this can be regarded as conceptualising the own subject with the help of categories from CSCW in order to understand it better. From a CSCW perspective, the representation of the software development discussion can be seen as further background material from the field, supporting the relevance of the conceptualisation. Being interested in software development as co-operative work, with the goal founding the development of methods and tools in a better understanding of the practice of software development, we are

reflecting these two perspectives.

In this article we put the use of representations as means of design and media of communication in the center. We argue that, despite the emphasis on documents and other lasting design artifacts in software engineering, the more flexible and fleeting means of representation used in face-to-face communication play an important role in co-operative design. The shared understanding achieved through these means provide the background for the interpretation of the more permanent documents.

The concepts from CSCW help us relate these findings to the discussion about the character of co-operative work and its dynamics. Co-construction is a major aspect of co-operative design rather than an exception in the working distribution of labor. It puts forward the question of how to support co-construction, perhaps even for distributed projects.

During the period of January- June 1998 we were able to observe a software development project that took place in Ronneby/Sweden and in Oulu/Finland. Both teams consisted of students doing a project course in their respective software engineering education. Both project courses are designed to provide realistic settings to allow the students to experience and reflect on software development under industrial conditions. The students develop software for real customers and are supervised by an experienced software engineer acting in the role of a 'Head of Department'. In our case the project was successful; a running program satisfying the requirements was delivered, and both projects contributed to it. However, the co-operation did not work in a satisfactory way. Despite the fact that the two subprojects were only loosely coupled, the distribution of tasks and the mutual expectations and sub-goals were not clearly decided upon and communicated. Half way through the project the communication almost broke down. With joint efforts it was improved. In the end, both subprojects contributed to the finished product. The possibility to generalise from student projects could of course be discussed. However, even in real life industrial settings written and oral representations are used for co-operative design. In this respect, student projects differ less from real world projects than for example regarding organisational issues and budgeting.

The next section revisits the discussion around the use of documents and representations in software engineering. Then we put a specific meeting in Ronneby in the centre, a design meeting where the overall architectural design of the software was collectively constructed. What took place during that meeting was a joint construction, a co-construction (Badram 1998) of what the future software should look like on an overall level. The developed understanding could not be communicated alone through the documents produced.

Here we see one of the main problems regarding the spatial distribution of software development – or "software engineering in the wide"; software development can be regarded as routine co-construction. The problems of the co-operation between the two teams, and our observations within the Ronneby team, showed how much this co-construction, as well as the everyday co-operation and co-ordination, depends on a shared space that allows for spontaneous as well as planed face-to-face meetings. Software development practice is developed using – and hence depends on – face-to-face communication. Software engineering in the wide requires a changed practice as well as adapted technology.

# Representations in Software Development

In developing a shared understanding of a not yet existing and inherently invisible artefact of high complexity, (Brooks, 1986) representations play an important role, in meetings and face-to-face communication as well as for asynchronous communication. It is no surprise that representations have been considered important for software development by most of those involved with it, independent of what backgrounds researchers and practitioners come from. In the 'Software Engineer's Reference Book' (McDermid and Roock 1991), which can be regarded as compiling a mainstream perspective on software development, the whole software development process is described as transformation of representations of the future software until the final representation, the executable program, is ready. The different representations are taken to mark milestones in the development process, interfaces between different phases that are thought to be internally largely independent of each other. This so-called Waterfall Model was questioned by Parnas and Clement in their article 'A rational design process – why and how to fake it' already 1986: Software developers don't work that way. The authors proposed nonetheless to produce a documentation of the product, as if it were developed in such a rational way, supplemented by the history of argumentation and change around the design. This would help the developers to communicate the reasoning behind the design and allow reconstructing the design decisions later on.[1]

Naur added with his article 'Programming as Theory Building' (1985) another perspective on the role of documents. Here he claims that the development of a theory, relating the design of the interface and the software itself to its anticipated use, plays an important role in software development. Such theories, he claims, surface in the communication about the design and guide the actual work, but they cannot be communicated through documents only. If that is true – and Naur provides some argument for it – then what is the role of various types of representations and documentation during development?

In the context of participatory and evolutionary development the role of documents and other design artefacts are discussed from a different perspective. Common artefacts such as mock ups, prototypes or scenarios should serve as boundary objects between language and practice of developers and language and practice of users. (Ehn 1993) Moreover, design artefacts can support the development of a common practice and common language between users and developers. (Dittrich 1998) Other authors emphasise the same mediating qualities of design artefacts and documents in-between developers. (Keil-Slawik 1992) The tools & materials approach by Züllighoven et al. builds a whole development methodology on the use of documents as means for the developers to make sense of the work practice the software should be embedded in as well as to design and develop the software itself. (Bürkle et al. 1995) The conceptualisation of the documents and other design artefacts changed in that context: they are no longer regarded as interfaces transferring information between phases and groups but as common artefacts mediating the developing process, as necessary means of co-operative development. Using a concept from Bardram (1998), this aspect of the use

[1] The waterfall model was criticised also from a usability perspective: Lehman's article about 'Software, Life cycle, and law of program evolution' added to the discussion by recognising that the result of this description, the finished software, changes the situation it is derived from. Boehm's spiral model (1988) and evolutionary models like in STEPS by Floyd et al. (1989) indicate the further development.

of documents can be called co-construction: the common object is defined in relation to the problem to be solved and the situation at hand.

Based on our analysis of design talk in the following section we propose to consider not only the distinction between oral and written representations, but also their use regarding different aspects of the co-operative design work. During an empirical study about the co-operation in a distributed software development project we had the opportunity to videotape and analyse a design meeting. In this design meeting the collaborative construction of the overall architecture was proposed, negotiated and agreed upon. The resulting document of that meeting can only be regarded as a reminder for the participants. Other means were used to represent the future software in order to make it visible for the group and to discuss the overall design: drawings on the white board providing an easy to change medium for representations, and enactment were used to represent runtime properties of proposed solutions. These means to represent the future software during the design meeting were necessary for the software engineers to reach a shared understanding of the common object, which in this case is the overall software architecture.

In the following section we shortly describe the observed project and the study. Then we introduce you to the design group. We analyse in detail two minutes of the two hour long meeting, a passage we call the bridge talk. Thereafter we will discuss what that implies regarding co-operation and communication and the use of representations in software development as well as computer support for this kind of work.

# The Bridge Talk

## Background and Methods

The bridge talk took place in the context of a ‘big team project’ as part of the undergraduate education in software development in Ronneby. ‘Big team project’ means ca. fifteen third year students in software development, three economics students and two students studying in a program called ‘people, computers, and work’ are working together in as realistic conditions as possible. Each student has a budget of 700 hours to spend on the project. The budget of this specific project was 13300 person hours. The students are acting as a software developing company with an industrial partner as a customer; the supervisor plays the role of a head of department; negotiations, contracts and commitments are up to the group itself as well as the organisation, the planning and the fulfilment of the contract.

We observed one of these projects which was co-operating with a small programming project in Oulu, Finland. The programming project teams in Oulu consist of 3-4 students having each a budget of 200 hours. Otherwise the course has the same rational. Working for an external customer allows for realistic conditions. In our case the Ronneby project was acting as a customer. The goal was to learn and teach about software engineering in the wide. In order to evaluate the experience, the project was subject to two case studies, one in Ronneby and one in Oulu.[2] During the software

[2] The Ronneby case study was the basis for a bachelor thesis within the ‘people, computer, and work’-program (Alriksson & Rönkkö 1998). The Oulu study will result in a master thesis of a computer science student (Petman under construction).

development which lasted for 5 months the empirical research-material was collected by interviewing, by observing, audio-taping, and video-taping project members in different meetings, such as: 'Head of Department' meetings, project meetings, inspection meetings, video and telephone conferences. Another approach was to hang around in the project room and take coffee breaks with the team members in order to understand the evolution of concepts and the difficulties in communicating them. We had access to the written communication in-between project members in the form of emails that were cc:ed to us. All documents produced were placed on their common work space, using a BSCW[3] server. We had access to that too.

We paid one visit in Oulu half way into the project to interview the Finnish team, and to share observations with the teachers and researcher in Oulu. The largest part of the empirical research has been done in the first period of the project, their pre-study phase. The performed field study of this distributed software development project has the characteristic of an ethnographical study. We have used Interaction Analysis on the shown field material in this article; the transcribed videotaped meeting.

The customer for our project was a medium sized Swedish software company specialised in strategic gameware. The task of the project was to develop a software system to simulate companies for management education. The co-operation and co-ordination were left to the projects. In Ronneby the project decided to make it a task of the project leader, in Oulu the group considered the project too small to make one person responsible for it. They all took over the responsibility of receiving and answering mails, of co-ordinating and keeping contact. The form of co-operation – Oulu was to play the role of a subcontractor – and the content of the subproject were defined early on. Oulu had access to all project documents. They used the same virtual work space to store their own documents.

Nonetheless the co-operation did not work very well. For example, it was not clear between the groups, who was responsible for the design of the database which was to provide the interface between the parts. Mutual requirements for the interface between the two parts were exchanged at a late state in the project. Documents were difficult to locate in the complex structure designed by part of the Ronneby project according to their local needs. Professional terms such as for example 'project report' had different interpretations in each place due to a differing software development 'culture'. This was not recognised by the students and caused misunderstandings and frictions between the project groups. In the end, the sub system the Oulu group was responsible for could only be finished, thanks to the fact that the students spent additional time on the project and got additional study points for it.

Despite a good technical infrastructure – the students were able to meet in video conferences, they used e-mail and chat, a BSCW was installed to provide a common work space, they could have used telephones – the difficulties in co-operation and co-ordination were resolved very late in the process. The mutual access to an elaborated set of documents did not make a difference here. The video recording and subsequent Interaction Analysis of a design session helped us to understand the source of these difficulties and to reformulate the question; we stopped wondering why they did not communicate and started to admire their success despite the geographical distribution of work.

---

[3] BSCW stands for 'Basic Support for Co-operative Work' an application which has been developed by the GMD in Bonn. (Bentley et al. 1997)

## The setting

The project team in Ronneby, consisting of twenty students, had to split up into subgroups itself. They started out with two design teams of three software engineering students, a quality team consisting of software engineering students, and the MDA students and economic students making up a team of their own. For the implementation phase they split up the group according to subparts of the software. In the Ronneby team the co-operation and co-ordination between the subgroups worked well enough, so that the final product was developed in time.

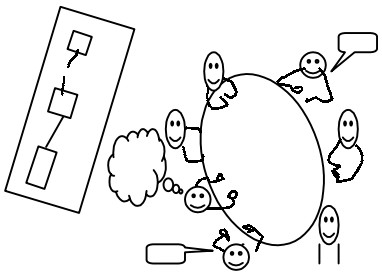

Figure 1.: This is the way they were gathered around a table, with a whiteboard in the room. On the whiteboard (here conveniently laid out on the floor) we can see the game creator represented to the left, the Game engine in the middle and the client to the right. The issue of their discussion in our example is in the space in-between the broken line, that are to say if there is a need for a Bridge.

In the taped session, the two design teams, each of which had separately developed proposals for a common overall design solution, met together with the project manager in order to join their contributions and develop a common design. They met in a smaller group room which was equipped with a whiteboard, and a table with chairs around it (see figure 1). Before our transcribed two minutes, one hour of the meeting had taken place, during which each team presented their design proposal to the other team and they were one hour into a discussion of the common design. In the following analysis of two minutes of this session, we focused on means of representation that served as common references or common artefacts during the design meeting.

## The white board as lasting common ground

One of these means of representation was the whiteboard that was part of the equipment of the room. During the whole session it was used to represent the momentary state of the common design solution. The students used common notations in software engineering: boxes, text, and arrows.

At the point where this transcription started everybody seemed to be pleased and kind of grasping each other's proposals. It becomes clear that there exist conflicting perspectives concerning the understanding of the white board representation, when the Chief of Software Architecture says[4]:

[4] Transcription notation system developed by Gail Jefferson. The original language in the transcription is Swedish. CSA: Chief Software Architect, PM: Project

CSA: →What! Put something in-between, there is no need for anything in-between, if we,
[if ]we have
PM: [If]
C: [If], if we have TCP IP (0.2)
CSA: No, even if we have RPC, there is no need for anything in-between. (0.4)
V: →No need for anything in-between? (0.2)
CSA: Between ((points with his right hand at the whiteboard drawings, first at the engine then at the client)) engine and the client. Okay we need a communication class, →but that we need in whatever solution we will have. (0.2)
V: ((moves his chair a bit backwards, then leans back in his chair at the same time as he puts up both hands behind his neck)) Hmm ((makes smacking noise)) ye, yes (0.2)
C: →Then we do not need that bridge which we had [then ]
V: [Hm::] (0.9)
V: No eh:: the reason that I wanted the bridge (0.2) is namely: First, this discussion we have now. (0.2) We don't know which technique that is good to have. But it won't ((moves his head like saying no)) be any danger because ((looks in a searching way at the whiteboard)) we will export object type COM, so what technique ((an opening gesture with his arms)), in this way lets say TCP IP or RPC and they won't function. So then we can exchange without digging in the client, (0.2) OR without digging in the engine=
CSA: =Hm::→What you are really saying is, your bridge ((looks towards the "bridge" on the whiteboard)), it could as well be connected with, with the engine. ((V nods his head as though saying yes)) It is just that you have put a COM interface in-between them.
V: Yes ((CSA nods his head as though saying yes, looking at V))
[and because]
CSA: [That, that ]

In the transcript we can see how they achieve a physical sharing of key objects with the help of drawings on the whiteboard. It eases their further development and questioning. In this way, the progress or agreements become within reach to be pointed at and talked about even as time goes by. Since the talk itself did not include pre-programmed questions or answers, the direction of the talk is achieved by the participants and can change with every new turn (Suchman 1987). The white board provides a common memory or a stable ground to their otherwise fleeting, fast and continuously ongoing talk. The participants used it to take care of some of the progress or agreements from their talk. The collaborative corrections, adjustments and agreements in every new conversational turn are in this way negotiated and articulated into a shared understanding of the white board representation.

## Enacting as resource in communication

At the point where the first transcription ended, it seems as if they reached a shared understanding again. However, another project member from the same team as CSA

---

Manager, C, V: Software Engineers.

shows his confusion.

**C:** =But what, the COM interface, COM is an object ((forms his hands like holding the COM object in-between them over the table)), an object that is COM or COM component ((each time he refers to the COM he moves what he is holding in front of himself up and down)), but what ((looking in a searching way at the whiteboard)), what is it that is the COM component? →Is it the bridge ((moves what he is virtually holding in front of himself to the right)) that is the COM component
[or is] it ((now moves what he is holding to the left))

**V:** [no ]

**C:** a part of the game engine, or? (0.1)

C, a member in CSA's team, realised that he did not understand the new turn the conversation had taken. So he forced the conversation into one more explanation-turn by asking questions, hoping that he also would be able to grasp this new understanding the CSA got hold of. As a way of expressing his confusion, he performs an enactment with the help of his hands, trying to mediate what he understands of the COM functionality but somehow not knowing where to put it. The talk continues:

**CSA:** A part of the game engine=

**C:** =→A part of the game engine? ((first looks at CSA then directly back to V as if he is the one to answer the question))

**V:** The whole engine is a COM object and it can in its turn, the file can access new COM objects ((moves his right hand forwards)), you have your game engine ((representing it with an open hand in front of himself)), Schach ((moving fast upwards with his right hand like grasping something of the same size as an apple in the air, holds it)), then they say give me values, ((doing the same movement and grasping, but now with his left hand)) prrt, the value says, give me a company, ((same movement and grasping again as the first time with the right hand)) schach, my company says give me sub-scenario one, ((doing the same movement and grasping again, but now his left hand)) one, then ask for then, now I have these two((holding the two virtual "values" in front of himself, and looking at them)), tell everything to me and then the game engine starts ((doing the movements in turn with both hands, left hand backwards and right hand upwards, again and again)) going
[forwards and backwards there.]

**C:** [→And it is the bridge that sits ] and ((repeats V's earlier hand gesture with his right hand, but with smaller movements)) pick up and asks after all these?=

**V:** =Yes, and the bridge goes from using these ((does the hand movements just as above, again and again)), these COM objects, then interprets them to what ever ((kind of throw away things movement with his right hand, repeated faster and faster a few times)) communication channel ((C nodding yes with his head, up and down)) it wants=

As if the body language were not enough, V in his answer added sounds to his enactment, this as a way of injecting even more 'life' in it. It appears that the COM object should be understood as a function, something that the static white board representation does not provide. The functionality was instead illustrated with body movements and sounds. In

order to confirm and unfold earlier enacted contributions in their talk, the developers also imitated each other's earlier enactment's as references to what they were talking about, as if the shared understanding was linked to these earlier objects of enactment.

In another episode later on, enactment is used together with the whiteboard in order to represent dynamics through this otherwise static medium.

## The role of their representing protocol from the design meeting

The meeting continued, until the participants thought they had a common enough understanding of their design object, the collaboratively constructed overall design. The software engineers' work of developing and agreeing on the overall design solution was nothing they considered as special in any way. It was part of their daily, ordinary work, an important meeting of course, but at the same time just another meeting. The actual work of developing a shared understanding and agreement concerning the collaboratively constructed design object was taken for granted and passed unnoticed. This is the kind of mundane work that is not necessarily consciously made note of. Project members would not function or go ahead in the project if they started to consider every such step they made in detail. However, for us as observers it is of interest how they reached a shared understanding of the design object and agreed in their mundane work. If this is the way mutual intelligibility and shared understanding (Suchman 1987) develop in a project, then it also has implications for how the role of the documents should be understood.

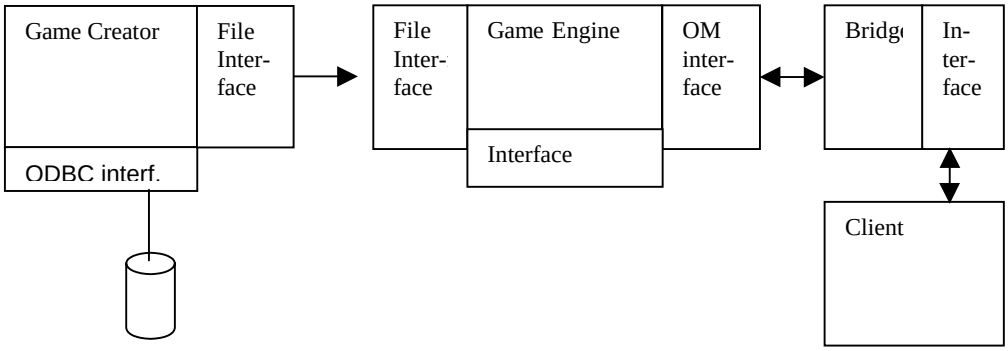


Figure 2: The representation from their protocol.

The protocol from this meeting included the same representation as that which emerged on the white board. There was rarely any describing text connected to it. It seemed to be a memory support made exclusively for the team members present at that meeting. If the framework for interpreting the representations emerged parallel to the 'hard' work of establishing shared understanding, as seen above, then how can project members that were not present at the meeting interpret the protocol and take part in this shared understanding? How was it possible for the non-present project members to collaboratively operate and co-ordinate work that depended on knowledge developed at this meeting? The project on the Ronneby side, where the main part was developed, had the advantage of physically having each other within reach for face-to-face communication. They could establish and re-establish their shared understanding concerning the common product through face-to-face communication; in the corridor or in a discussion during a coffee break, arrange new meetings, by personal interviewing, or perhaps get understanding through overhearing others talk, to give some examples. While the Finnish side in their striving for understanding had to trust the information technology, which turned out to be insufficient.

# Representations and Co-operative Design

During the field study, the bridge talk and the analysis of it have helped us to understand the dynamics of the co-operation and co-ordination within the project group and with the Finnish team. Instead of focussing on the interaction or non-interaction between sub-groups we started to look at the co-operation in a more vertical way. Software development work often is not nicely organised with sub groups taking care of tasks which can be put together afterwards, despite of what is regarded as an ideal. The focus on planning and reporting, estimation and measurements, and defined routines, can perhaps even be regarded as an indicator that especially this aspect of software development work is a rather problematic one. Software development as co-operative design work means to co-construct the 'what', to co-operate in the implementation of it and to co-ordinate this co-operative effort. Oral and written documents, artefacts like mock-ups, plans, and procedures play a role in all these types of work.

This aspect has been recognised within the software development discourse as well. Floyd emphasised that software development includes the design of the program as the product, the design of the process to develop it, and the design of the use situation. (Floyd 1992) The three design aspects in software development are mutually dependent on each other. The dependency between process and product holds even when the technical aspects[5] of software development are put into focus; for example the overall architectural design of the program indicates how the work can be distributed among sub-teams. On the other hand the actual co-operative implementation influences how the design is interpreted and perhaps changed in order to fit the distribution of work force and competence. The co-development of the design and a shared understanding are therefore not only a step to the final product, but also necessary for a working distribution of labour.

## Dynamics in co-operative work

Artefacts – plans, documents, key racks (Robinson1993), for example – are subjects in the CSCW literature as well. In 'Coordination mechanism' Schmidt and Simone (1996) introduce a way of looking at the interaction of co-operative work and artefacts. The co-operative work is constituted by the interdependency of actors in their individual activities contributing to a common objective. Changes in one of the fields of work imply changes in the whole system. In this way the actors interact through the change in their common field of work, through a common artefact. The co-operation has to be restructured in order to regain stability regarding the co-ordinated activities. Articulation work is needed to restrain the complexity of the interdependent activities.

The co-operative work that is be supported by these co-ordination mechanisms seems to take place in a stable division of labour. Articulation work to reconstruct this division and the necessary co-ordination mechanisms is an exception caused by a break-down. Focusing on stable work situations on co-ordination mechanisms and their redefinition through articulation work makes a lot of sense if one is looking for computer support. However in our example it leaves us with some confusion. Regarding the construction of a software system, the whole talk was articulation work. Within the talk itself you can distinguish work: describing, explaining and arguing for a design solution – and articulation work, -what is the problem we were assigned to solve. Besides this, in

[5] And, as in this article, the designer – designer communication, is put into focus.

our case there did not yet exist any stability in the division of labour. Both the 'what' and the 'how' still had to be constructed together, as well as the co-ordination mechanisms for the further design effort on the different levels.

Bardram proposes in his article 'Designing for the dynamics of co-operative work activities' (1998) to take further dynamics of co-operative work into consideration.[6] He also takes as a starting point, what takes place after problems or break-downs have occurred. He points at the need for addressing the dynamics for understanding co-operative work. Bardram's conceptual frame is based on Activity Theory. Activity Theory identifies a three level hierarchical structure of collaborative activity, which is what co-operative work is called in this school. On the first, the *co-construction* level, the collaborative object is either not stable or does not exist and therefor has to be collectively constructed i.e. co-constructed. Questions like; 'What is the meaning of this problem?', 'How did the problem emerge?', 'Why are we trying to solve it?', 'Who benefits from the solution?' are asked on this level. On the second, the *Co-operation* collaborative level the actors have achieved a common object from their collaborative work. The actors are engaged in balancing and adjusting their own actions to the actions of their partners in order to achieve the common task. The third, the *co-ordination* level is that of routine flow of interaction, where work is done in harmony with surrounding activities. Here the flow of work goes on without questioning or discussing, according to tacitly assumed traditions and norms. In practice collaborative work takes place on all three levels according to the requirements of the situation. Figure 3 shows the dynamics between different levels. The collaboration shifts between the three different levels of co-ordination, co-operation and co-construction. Problems within routine work lead to breakdowns in the co-ordination level and become subject for co-operation, and breakdowns on the co-operation level in turn become subject for co-construction.

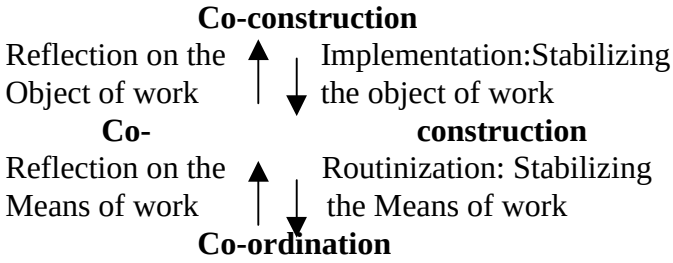


Figure 3: Levels of collaborative activity (Badram 1998, p.92)

## Co-construction, co-operation, and co-ordinated work in software development

In software development projects the routine division of labour is achieved together with the design of the software. Design and development projects do not start out with a given object to work on. The common object is the subject of the construction throughout the whole process. Expertise and routine regarding how to run such projects will of course be acquired over time. Regarding each project, the new product, a suitable way to divide the

[6] Badram argues against Schmidt and Simone. However, we actually rather see them as two ways of conceptualising what is going on rather than contradictions. To discuss the relationship in depth would be interesting but is beyond the scope of this article.

work and the team, and a working co-ordination of the single tasks has to be achieved. Even if there are existing mechanisms and routines for how to achieve the co-ordination they have to be adapted and customised for the project at hand. With the product, the process has to be designed and implemented. The emphasis put on the different aspects of co-operative work might change throughout the project. Independent of the overall process model, during the requirements and design phases the co-construction can be regarded as the main task. During implementation co-operation and co-ordination might be most visible from the developers' perspective. However, as the design is not finished until the software is ready, the division of tasks to be co-ordinated is not stable. Tasks are redistributed, designs are changed because of unexpected technical constraints or evolving requirements. One could say that co-construction is routine work for software developers.

The design talk we have analysed here is an example of such routine co-construction. With the talk the participants developed a shared understanding of what the software on an architectural level should look like. Through the arguments the rationale behind the design, the advantages and disadvantages and the risks were shared, too. Thus not only the foundation for the division of further design and implementation were laid, also the background to co-ordinate the co-operative tasks was developed; which other tasks might be influenced if I have to change my design, and whom should I contact. For this co-constructive task only the white board, talk and enactment were used as representations. The representation in the protocol showed the attained solution, but did not tell about the 'why's and 'how's of it. Through reading only, the result of the co-construction, the shared understanding of what should be build, could not be communicated. Within the Ronneby team this was solved through everyday communication going on unrecognised and undocumented. A subgroup that did not take part in the product design but designed the quality assurance routines and quality criteria for the different documents was introduced into the co-operative design and implementation without any great problems. The Finnish team however was left behind. Despite being able to read all the documents and follow changes, with the help of a common BSCW workspace, they did not participate in the co-construction and its result in a satisfying way.

Different representations and different forms of representations serve different purposes. To some extent this has been acknowledged in literature. Process-related documents – plans, review routines, progress reports for example – help to organise and co-ordinate the co-operative work. Product-related documents – stating requirements, design documents on different levels, documentation, test specifications and protocols – describe attributes of the software under construction. We propose adding another dimension for distinction regarding the medium of the representation: In our example talk and enactment, two fleeting mediums, were used for highly interactive co-construction. White board drawing, a less fleeting medium, was used to keep certain aspects of the object under construction present during the meeting. On the other hand it is a flexible enough medium to not restrain the evolving and sometimes meandering discussion. Written or computer-based documents yielding very permanent representations were used to document results, to structure the design, and to organise the common work over a longer time. Beside this, the different mediums were interwoven; oral communication was used to explain documents; computer based drawings like the 'result' of our talk helped the participants to remember what was going on during the design meeting. Without being able to prove it, it seems that co-construction mainly takes place in fleeting mediums whereas more permanent representations are used to support co-

operation and co-ordination. If that thesis holds, the role of documents for software development has to be reconsidered as well as the possibilities for distributed software development and its support by CSCW applications.

# One Conclusion and a Bunch of Questions

The main result for us is that software development has to be regarded as routine co-construction.[7] Therefore it is not enough to consider the co-ordination aspect of it in order to reason about computer support. CSCW often focuses on communication, co-operation, and co-ordination between people or between groups. Often the object of the co-operation, the goal to be achieved, is assumed to be given. Software development starts out with a very vague goal compared to the needs of users; an idea of a computer application, a satisfied customer. What that means in terms of the software to be developed is unclear. The co-construction of the common product and the design of a co-operative process to achieve it are major parts of the routine work and not exceptions or break-downs that have to be resolved.

We shared an example of such a design meeting where a important part of the co-construction took place, and we were able to observe the impact of not taking part in the co-construction and not being properly informed about the results. How in the normal everyday practice co-construction, co-operation and co-ordinate work interweave was not our focus and is still an open question.

We had the possibility to see how software developers used talk, body language, enactment, and a whiteboard to co-construct an overall design of their program. According to our observation the co-construction within software development takes place above all in face-to-face situations, either during coffee breaks, in front of a screen or a document, or during meetings. Distant co-operation in such tasks seems difficult but necessary; co-operation and a working distribution of labour, and the co-ordination of the defined tasks very much rely on it. A lively discussion via a videoconference device is very difficult, and video conferencing requires a special set up. The technique itself also has its drawbacks as described for example by Heath and Luff (1993). Spontaneous interaction would be difficult. Often such a dilemma is answered by asking for new, and better technology, or alternatively the conservation of today's practice. However, we all know examples of distributed co-constructive practice; the scientific discourse or the organisation of conferences for instance. Besides the development of well-adapted technologies, for distributed Software Development different practice is to be developed, too.

What might efficient computer support for distributed software development look like? In our case, the students used the BSCW developed to define a common workspace for their documents. That did not work in a satisfying way. While that might be due to the negligence of the distant group regarding the design of the structure of this common workspace, sharing documents is certainly not enough. How can existing technology be used to support the co-operation in a suitable way? How must the work practice be adapted to allow co-operation across long distance? Perhaps the bottleneck of digitalisation might turn out to influence the development of a design feature – like structurally necessary columns in an example from architecture (Tellioglu et al. 1998). A

[7] With 'routine' in this place we don't mean the 'coordinated activity', like Bardram, we mean the normal everyday work practice.

typographic practice of co-construction might also lead to a better documentation.

This spring term we will have a second trial regarding software development in the wide. Again a big team project in Ronneby will co-operate with a smaller programming project in Oulu. We will share our experiences and observations with the new students. The results will teach us a second lesson. Perhaps some of the questions raised above can be answered, perhaps new one will be added.

# Acknowledgements

We wish to thank Monika Alriksson and Timo Petman, who conducted the field work together with us. We jointly tried to make sense of what we were observing. Conny Johansson and Antti Juustila made the co-operative project possible and supported us as critical discussants. The project members with a rare openness allowed us to spy. We thank our colleagues, especially Sara Ericsén, for support and comments on drafts for this paper.

# References

Alriksson, M., Rönkkö, K.: *Softalk. Communication in distributed software development,* Bachelor Thesis, Institute for Computer Science and Business Administration. University of Kralskrona Ronneby 1998.

Bardram, J. E (1998): 'Designing for the Dynamics of Cooperative Work Activities' in *Proceedings of the Computer Supported Cooperative Work CSCW 1998,* (Seattle, November 1998) ACM Press.

Bentley, R., Horstmann, T., Trevor, J. (1997): 'The World Wide Web as enabling technology for CSCW: The case of BSCW', *in Computer-Supported Cooperative Work: Special issue on CSCW and the Web*, vol. 6 (1997).

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

Brooks, F. P. Jr. (1987): 'No Silver Bullet- Essence and Accidents of Software Engineering.' *Computer*, 10-19.

Dittrich, Y. (1998) *Developing a language for participation. Project language as a meeting place for users and developers in participatory software development.* Technical Report, University of Karlskrona Ronneby, 1998.

Ehn, P. (1993): 'Scandinavian Design: On Participation and Skill.' in Schuler, D. and Namioka, A. (eds.): *Participatory Design: Principles and Practices* Hillsdale,New Jersey 1993, pp 41ff.

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

Floyd, C. (1992): 'Software development as reality construction' in Floyd, C., Züllighoven, H., Budde, R. and Keil-Slawik, R. (eds.): *Software Development and Reality Construction.* Springer Verlag: Berlin.

Grinter, R. E. (1998): 'Recomposition: Putting It All Back Together Again' Activities' in *Proceedings of the Computer Supported Cooperative Work CSCW 1998,* (Seattle, November 1998) ACM Press.

Heath, C. and Luff, P. (1993): 'Disembodied Conduct: Communication through Video in a Multi-Media Office Environment.' in Baecker. . R. M. (ed.): *Readings in Groupware and Computer-Supported Cooperative Work. Assisting Human Collaboration*, San Mateo, CA: Morgan Kaufmann, 837-841.

Keil-Slawik, R.(1992): 'Artifacts in Software Design.' in in Floyd, C., Züllighoven, H., Budde,

R. and Keil-Slawik, R. (eds.): *Software Development and Reality Construction.* Springer Verlag: Berlin.
McDermid, J. and Roock, P. (1991): 'Software Development process models.' In John McDermid (ed.): *Software Engineers Reference Book*, pp 15/3-15/36.
Naur, P. (1985): 'Programming as Theory Building.' *Microprocessing and Microprogramming* 15(1985), 253-261.
Parnas, D. L. and Clement, P. C. (1986): 'A rational design process: How and why to fake it.' *IEEE Transactions on Software Engineering* SE-12, 251ff.
Petman, T. (under construction): *Communication Matters*. Master Thesis, University of Oulu.
Schmidt, K. and Simone, C. (1996): 'Coordination mechanisms: Towards a Conceptual Foundation of CSCW Systems Design', *Computer Supported Cooperative Work*, vol 5.
Schmidt, K. and Sharrock, W. (1996) (guest editors): *Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design*. vol. 5, No.4, 1996.
Suchman Lucy, (1994): *Plans and situated actions*, Cambridge University Press.
Tellioglu, H., Wagner, I. And Lainer, R. (1998): 'Open Design Methodologies: Exploring Architectural Practice for Systems Design' Broadening Participation: Proceedings of the Participatory Design Conference Seattle WA. USA, November 12-14 1998.