# Computer Supported Coordination

Peter H. Carstensen

This Ph.D. dissertation was made by Peter H. Carstensen at Department of Computer Science, Roskilde University, Denmark. The work was conducted in the period from January 1993 through December 1995, and defended in public in March, 1996.

# Table of contents

# Preface and acknowledgments

This dissertation documents the research activities conducted during my Ph.D. work. The work began in January 1993 and lasted for three years. The work was supervised by associate professor Finn Kensing, Department of Computer Science, Roskilde University.

During the work I have been employed at Systems Analysis Department, Risø National Laboratory, where I, most of the time, was associated to the Esprit Basic Research Project COMIC lead by Tom Roddon, Lancaster University, and having Kjeld Schmidt as project manager at the Risø site.

The work is related to the research field called CSCW (Computer Supported Cooperative Work). The central overall questions regards what characterizes cooperative work, and how we by means of computer systems can support the cooperation, e.g., by providing better communication facilities, provide improved monitoring and awareness possibilities to the actors, and reducing the complexity of the coordination activities to be conducted by the involved actors.

The work presented has mainly been concerned with contributing to the establishment of a conceptual framework for understanding the basic mechanisms used when mutually interdependent actors coordinate their distibuted activities. To establish such a framework, several different activities have been conducted. These include analyzing existing CSCW-applications, conducting a field study of coordination of cooperative work, and designing a computer-based coordination mechanism.

Much of my thesis work has been published elsewhere, either as technical reports, reviewed conference proceedings papers, or journal papers. These papers have been used as input for the present dissertation. Some chapters are revised versions of one of these papers, or an integration and elaborated discussion of the themes addressed in these papers. Other chapters are completely new in the sense that the theme or content have not been published elsewhere. A list of all the papers published elsewhere can be seen in Appendix A.

During the work, I have collaborated, or in other ways communicated, with a lot of supportive people. This dissertation would definitely have looked differently, if I had been excluded from having detailed discussions on ideas, concepts, prototype sketches, etc. with a large number of bright people. I have also learned a lot from writing papers together with others. I would like to take the oppotunity to thank the following for their support and collaboration:

I am indebted to Finn Kensing for supervising the thesis work. I have learned a lot from our discussions on ideas, drafts, etc., and from his constructive comments to papers and preliminary versions of the chapters presented here. When I complained being too busy and stressed, Finn was also the person who taught me, that the person to blame was me. Decisions on what to be involved in is taken by me and nobody else. Simple, but true.

Two of my colleges and close friends should have a very warm and special thank: Thanks to Kjeld Schmidt and Carsten Sørensen. I am indebted to both of you. I have written several papers together with both of you, and you have taught me a lot about doing scientific work. We have had a lot of great fun together, and an abundance of fruitful discussions on almost all the topics addressed in this dissertation. Thanks to Kjeld for introducing me to the CSCW community and for previous collaboration on the Work Analysis (cf. Schmidt and Carstensen, 1990) that opened my eyes for the usefulness of sociologically inspired approaches within software development. Kjeld is the visionary person establishing the ideas of coordination mechanisms that are central in this dissertation. I am grateful to Carsten for his good guidance on writing papers and doing thesis work. Carsten has supported me by reading and commenting on a draft version of this dissertation, and he has done a great job in helping me polishing my poor English.

Also warm thanks to Tuomo Tuikka for a fruitful collaboration on papers on reuqirements for support of coordination of software testing, to Thomas Albert for implementing the prototype, BRaHS, that is presented in this dissertation, and to Liam Bannon for his involvement in the evaluation of the prototype.

I had a series of good discussions with Anker Helms Jørgensen in the early phases of my thesis work on how to combine the areas of CSCW and

HCI (Human-Computer Interaction). Although the HCI aspects ended not playing a major role in the final approach taken, I would like to thank Anker for his support.

Jesper Simonsen and Morten Nielsen supported me by providing a long series of good, relevant, and constructive comments on a draft version this dissertation.

I have also written papers in collaboration with a number of other people: Thanks to Hans Andersen, Henrik Borstrøm, Monica Divitini, Betty Hewitt, Birgitte Krogh, and Carla Simone. I appreciate all our discussions in relation to these papers. They have been fruitful and informative, and I have learned a lot from each of you.

During the work, a lot of people have been working for periods in our group at Risø: Apart from Kjeld and Carsten also Hans, Monica, Betty, Birgitte, Carla, Tuomo, Morten, and Steffen Herskind have been associated to the group. Thanks to all of you for a long series of good discussions at our monthly meetings. We addressed many interesting topics, both within what is addressed here and other areas of relevance. And we had—and still have—a lot of great fun too!

Much of the research presented here could not have been conducted without the invaluable help of numerous people at Foss Electric in Hillerød. The stay at Foss Electric gave me a great opportunity to ask thousands of questions concerned with their work. Thanks for letting me fumble around. A special thank to Marianne Malmstedt, Carsten Paludan, Ole Pflug, and Jørn Ørskov.

The Man-Machine Interaction group at Risø, where I work, has been supportive, both by funding my Ph.D. work, and by allowing me time to write up this dissertation. A special thank to the group manager Leif Løvborg for his suppport.

The work has been partially conducted within the Esprit Basic Research Project COMIC. Thanks to all the colleges involved in the COMIC project. The meetings with this group of people was my first encounter with a larger part of the European CSCW community. It was great.

The Danish Science Research Council (SNF) and the Danish Technical Research Council (STVF) have partially funded the work reported here through their support to the Centre for Cognitive Informatics (CCI), and

through the support from SNF to the CSCW work at Risø (the KOMEK project).

Thanks to the Øresunds group for good support through valuable comments to papers and presentations at the Dragør seminar in August 1994, and to the reviewers at the following conferences: COPE IT' 93, HCI International' 93, IRIS' 94, HCI International' 95, and IRIS' 95. And to the reviewers of The Scandinavian Journal of Information Systems, and The CSCW Journal.

Last, but definitely not least, a warm thank to my wife, Helle. She has supported me tremendously in many ways, basically by being a safe base pleasantly making me aware, that there is indeed a life besides that of working on a dissertation.

Although it would have been impossible to write this dissertation without the great support from all the people mentioned above, it is, of course, clear that all the errors, mis-interpretations, mistakes, etc. in this dissertation is my personal responsibility. I am the only one to blame!

Peter Henrik Carstensen
Risø, December 1995

# Part I:
# Introduction and case

## 1. Introduction

A general trend in modern work settings seems to be that the work becomes more and more complex. Complex in the sense that it is characterized by complex problem solving and decision making activities, rule interpretation, cooperative work processes, etc. The demands for flexibility, production time, complexity in products, etc.—as it is for example reflected in the trends of “concurrent engineering” (Helander and Nagamachi, 1992)—are increasing. Also the structure of the context of most work settings are becoming more and more differentiated, and is characterized by an increase in the speed of the changes (Ciborra, 1993). Many activities and situations to be handled in modern work settings often have an inescapable aspect of contingency (Suchman, 1987).

The increasing complexity of the work activities to be conducted, the situations to be dealt with, and the structures to be handled, requires involvement of several or many actors in the work processes. Since individuals have limited capabilities and capacities, the work arrangement to conduct the work becomes cooperative. Cooperative work arrangements emerge in response to different requirements and must serve different generic functions such as augmentation of capacity, differentiation and combination of specialities and techniques, mutual critical assessment, and combination of perspectives (Schmidt, 1994c).

In the literature several approaches for when to define a work setting as cooperative exists (e.g., Ehn, 1988; Johansen, 1988; Schmidt, 1991a) spanning from, in the one extreme, viewing almost all work as cooperative, to, in the other, only consider work as cooperative if all actors are aiming at the same common goal.

When several actors having different competencies, perspectives, strategies, etc. are involved in a cooperative work arrangement they become mutually interdependent in their work, i.e., “cooperative work occurs when multiple actors are required to do the work and therefore are mutually dependent in their work and must coordinate and integrate their individual activities to get the work done" (Schmidt, 1991a, p. 305). Mutual interdependence means not only sharing resources, but also that the involved actors mutually rely on the quality, feedback, etc. produced by the other actors, i.e., no matter how the division of labor is organized, the actors involved will be interdependent and need to interact with each other.

In order to get the work done, they have to coordinate, schedule, integrate, etc. their individual activities. The actors have to coordinate their work along the salient dimensions of who, what, where, when, how, etc. (Strauss, 1985). The need for coordination can be handled by use of several means, from purely *ad hoc* communication and coordination in the one extreme to completely pre-programmed and rigid work-flow systems in the other (Schmidt, 1994c). One of the central researchers in the organizational theory, Mintzberg, has an even simpler approach to the needs for coordination. He argues that as soon as two persons work together, coordination must be achieved "across brains", and that every organized human activity gives rise to "two fundamental and opposing requirements: the *division of labor* into various tasks to be performed, and the *coordination* of these tasks to accomplish the activity" (Mintzberg, 1983, p. 2).

When relatively few actors are involved, or the complexity of the work, or its coordination, is low the actors can achieve the required coordination by means of modes of interaction and conventions from everyday social life such as talking, gesturing, monitoring the situation, etc. (Mintzberg, 1983; Schmidt, 1994c). Several studies indicate that actors in these situations are extremely good at handling the complexity of coordinating by means of ad-hoc modes of interaction (cf. e.g., Harper *et al.*, 1989b; Heath *et al.*, 1993). Problems will, however, often emerge in highly complex work when, for example, the cooperative work setting includes many geographically distributed actors; when there are a large number of intertwined activities, actors, or resources; when different areas of competence with different conceptualizations and goals are involved; or when the work is carried out over a long time-span (see for example Carstensen *et al.*, 1995b). In addition to this, new technology (e.g., communication technology) provides a new potential for cooperation introducing new problems as well.

Mintzberg argues that the fundamental ways in which organizations coordinate their work can be described as five mechanisms: mutual adjustment, direct supervision, standardization of work processes, standardization of work outputs, and standardization of worker skills. These are ordered, i.e., when the work becomes more complex the "favored means of coordination shifts from mutual adjustment to direct supervision to standardization, preferably of work processes, otherwise of outputs, or else of skills, finally reverting back to mutual adjustment" (Mintzberg, 1983, p. 7).

To summarize: The need for cooperation emerge due to the need for more resources (capacities, competencies, perspectives, etc.) than one actor can provide. Furthermore, from the construct of a cooperative work setting raise the need for coordination, the actors have to coordinate their inter-related distributed activities. Opposing requirements appear: Increased complexity calls for more actors (i.e., more division of labor), but more division of labor calls for more coordination.

This dissertation is based on the assumption that more and more work settings will be characterized by complex cooperative work arrangements, and these will require complicated coordination activities. It thus becomes relevant to discuss how computer-based technology can be used to support the activities concerning coordination in complex work settings. This is the central topic in this dissertation.

The primary goal is to contribute to the development of a conceptual framework for supporting designers in analyzing and designing computer-based mechanisms supporting actors in cooperative work arrangements in coordinating their distributed activities. The second goal is to formulate preliminary, general, normative statements for: 1) how can complex cooperative work settings be analyzed in order to establish a basis for designing computer-based coordination support systems, and 2) which kind of computer-based tools supporting coordination work will it be relevant to provide.

My work has taken a design oriented approach to the CSCW research field. I have approached CSCW as a discipline concerning the problems of how to conceptualize, design, and construct computer-based systems supporting the coordination aspects of a complex cooperative work setting (Bannon and Schmidt, 1989). The emphasis has been on understanding cooperative work as a distinctive form of work (Schmidt, 1991c) with “the objective of designing adequate computer-based technologies” (Bannon and Schmidt, 1989, p. 5). The approach taken is systemic, inspired by, for example, Simon’s ideas of inner and outer environments (Simon, 1981; Simon, 1983), the Cognitive Engineering tradition (e.g., Rasmussen, 1986), Checkland’s ideas of soft systems (Checkland, 1981), and my own work on work analysis (cf. Schmidt and Carstensen, 1990).

So, what is meant by cooperative work and its coordination? This will be discussed in details in chapter 4, but let me here briefly introduce the approach applied:

Cooperative work is handled by a cooperative work arrangement. The central characteristic of a cooperative work arrangement is not that it is sited in one organization, that the actors share resources, or that the actors are physically located in the same room, etc. Instead, a cooperative work arrangement is constituted by the field of work, i.e. constituted by “the part of the world that is being transformed or otherwise controlled by the cooperative work arrangement [...] all cooperative work is based upon interactions mediated through the changing state of a common field of work” (Schmidt, 1994c, p. 15). This further implies that “‘cooperative work’ does not imply any notion of shared goals or conviviality, but rather people engaged in work processes related as to content” (Bannon, 1993, p. 11). A cooperative work arrangement is often established across organizational boundaries.

The actors in a cooperative work arrangement are handling task and activities that are closely related, intertwined, and interdependent, and, at some level, characterized by having a common objective with respect to the purchaser of the services provided. The term ‘coordination’ covers activities to be conducted in order to handle needs that arise because intertwined tasks and activities conducted by individual interdependent actors have to be coordinated, meshed, scheduled, integrated, etc. This is similar to what Strauss calls ‘articulation work’ (Strauss, 1985).

In this dissertation the term coordination is used in a broader meaning than the connotations usually implied. Coordination activities cover aspects such as scheduling, meshing, and allocating resources, negotiation of resource allocations, monitoring work activities, resolving inconsistencies, etc. I furthermore include activities concerning the establishment of means supporting coordination activities. For example, development of a form supporting the coordination of distributed test activities, or refinement of a classification scheme used to classify software bugs in a distributed cooperative work setting, will be considered coordination work (both examples are discussed in chapter 6). A simplification could be to state that, all 'extra activities', or overhead costs, arising because no omniscient and omnipotent actor exists (Schmidt, 1994c, p. 10) are regarded as coordination.

**co•or•di•nate** (*adj., n.*), *adj., n., v.*, **-nat•ed, nat•ing.** —*adj.* **1.** of the same order or degree; equal in rank or importance. **2.** involving coordination. **3.** *Math.* using or pertaining to systems coordinates. **4.** *Gram.* of the same rank in grammatical construction, as *Jack* and *Jill* in the phrase *Jack and Jill*, or *got up* and *shook hands* in the sentence *He got up and shook hands.* —*n.* **5.** a person or thing of equal rank or importance; an equal. **6.** *Math.* any of the magnitudes that serve to define the position of a point, line, or the like, by reference to a fixed figure, system of lines, etc. **7. coordinates,** articles of clothing, furniture, or the like, harmonizing in color, material, or style, designed to be worn or used together. —*v.t.* **8.** to place or class in the same order, rank, division, etc. **9.** to place or arrange in a proper order or position. **10.** to combine in a harmonious relation or action. —*v.i.* **11.** to become coordinate. **12.** to assume proper order or relation. **13.** to act in harmonious combination. Also, **co-or'di•nate.** [1635-45, co- + (SUB)ORDINATE] —**co•or'di•nate•ly, co-or'di•nate•ly,** *adv.* —**co•or'di•nate•ness, co-or'di•nate•ness,** *n.* —**co•or•di•na•tive, co-or•di•na• tive,** *adj.*

**Figure 1-1:** Examples of usual connotations of the term 'coordination' (RandomHouse, 1987, p. 447).

The approach could be accused to be too naive, and a simplification of what should be considered relevant. Many aspects are not explicitly addressed, for example the social structures in a work setting, the psychosocial work environment (e.g. Keller, 1994), the sociocultural aspects of work, or the power structures in the organization of the systems design (e.g. Ehn and Kyng, 1987). The approach is, however, chosen to have a pragmatic approach reflecting, that I consider CSCW a design discipline. It is not to deny the importance of the aspects excluded. It is rather to simplify the approach enough to actually make it useful for my purpose.

The rest of this chapter will first discuss the research question addressed in this dissertation in further detail. Then the research approach is described. Section 1.3 will discuss what different readers can gain from reading this dissertation and introduce the structure of the rest of it.

## 1.1 The problem addressed

As argued above, much of everyday work is so complex and demanding that it needs to be done as a cooperation among several, mutually interdependent, actors. This implies an increased need for coordinating distributed, complex, and interdependent activities. Involvement of many actors—often separated in time and space—implies that the common means for coordination activities (gestures, talk, mutual awareness, etc.) become insufficient in many situations.

It seems reasonable to consider how computers can be used to support the coordination activities in complex cooperative work settings. To do so analyst and software designers needs tools supporting them

(1) in approaching and understanding the coordination aspects of complex work settings,

(2) in identifying which aspects of the cooperation and coordination activities are relevant to consider computer support of, and

(3) by providing structures and guidelines for design of computer-based mechanisms supporting the coordination activities.

The purpose and idea in this dissertation is to contribute to the conceptual framework for understanding the mechanisms mutually interdependent actors use when coordinating their activities. The framework might later develop into a more general theory of cooperative work and its coordination. To be useful for analysts, the framework must be applicable when addressing actual real-life work settings. And to be useful for designers, the framework must be applicable when considering design of computer-based support. Analysts and designers might not necessarily need the same framework, but seeing the framework as supporting 'the transition from an analytical approach to a construction approach', I assume that a common framework supporting both will be useful. The framework will, therefore, be discussed from both an analytical and constructive perspective.

When studying cooperative work activities it becomes clear that communication is the basic means for coordination. It could then be obvious to 'only' develop more sophisticated computer supported communication channels. The framework should, however, also address the structure of interaction in order to be able to reflect how the structure of the cooperative interaction can be supported. That is, to let the computer tools go deeper into the structure of the coordination activities in order to provide a more sophisticated support. A basic assumption is that by aiming at a deeper and more conceptual understanding of what characterize coordination work we will be able to build better and more useful coordination support tools. Since the allocation of functionality between the human actors and the (computer-based) coordination mechanism probably will be changed during the (re-)design process, the conceptual framework should not define how the functionality should be divided up.

Phrased differently we can say that, coordination activities are (and be in the future too)so complex, that they need to be supported by tools. These tools can be designed by applying several strategies. A first strategy is to let the actors handle it themselves since human actors are good at adapting to new situations and handle coordination on an *ad hoc* basis. A second is to automate (‘amplify’) already existing means (artifacts). A third is to support what we can immediately observe in cooperative work settings, i.e., support personal communication and interaction. And a fourth strategy is to provide support tools that are based upon, and thus reflects, a conceptual understanding of what characterizes coordination activities. It is assumed that potentially the last will result in the most useful support although it must be recognized that in all cooperative work settings some of the coordination will be handled in an *ad hoc* manner, and that in many settings the actors will constantly invent and use new means for coordination, i.e., we cannot base coordination support on preprogrammed rigid workflow-like systems only.

Besides the contribution to the establishment of a conceptual framework for coordination mechanisms this dissertation also have a more construction oriented approach. Namely to, based on empirically studies, discuss general requirements for computer-based coordination mechanisms, discuss the usability of the conceptual framework for analyzing cooperative aspects of complex work settings, and to discuss the applicability of one of the commonly used software development paradigms (the object-oriented) for modeling the coordination aspects of work.

The what has been conducted, and approach for doing this, will be described further in the following.

## 1.2 Research approach

The research reported in this dissertation has basically been empirically driven, and has had a design oriented approach to CSCW. The central set of problems in CSCW is seen as the problem set related to: Which aspects of activities concerning coordination of cooperative work are relevant for designing computer-based support systems? How can we conceptualize these aspects? And what should characterize a good and useful computer-based coordination support system?

I consider myself a member of a construction discipline (design of computer-based systems), who has recognized a lack of knowledge in our design and construction methods, frameworks, and techniques. To overcome some of these I have entered other disciplines to search for inspiration to improve our conceptual understanding of some of the characteristics of the work settings that we build systems for.

First let me briefly mention the activities that has been conducted during the dissertation work.

### 1.2.1 What has been done

The aim has been to contribute to the establishment of a conceptual framework for coordination mechanisms, and discuss its usability as a tool for analyzing cooperative aspects of complex work settings. Furthermore, general requirements for computer-based coordination mechanisms, and how these can be achieved, have been discussed. To fulfill these aims a number of activities have been conducted:

First, a preliminary understanding of essential characteristics of complex cooperative work and its coordination was established through literature studies. This was, for example, literature on complexity (e.g., Simon, 1973; Simon, 1981; Woods, 1988), theories of cooperative work and its coordination (e.g., Strauss, 1985; Malone and Crowston, 1990; Schmidt, 1991b), and ethnographic studies of work (e.g., Suchman, 1987; Harper *et al.*, 1989b).

A first version of a conceptual framework was then established. This framework addressed mechanisms supporting coordination activities of cooperative work by providing and mediating the information structures needed for the coordination, and by prescribing and stipulating the flow of either the work itself or the coordination work. This was done in collaboration with several colleagues and was first described in relation to the Esprit project COMIC (cf. Schmidt *et al.*, 1993; Simone and Schmidt, 1993). The first version was established through analyzing existing CSCW applications (Andersen *et al.*, 1993), a critical walk-through of existing frameworks and analysis methodologies (Carstensen and Schmidt, 1993a), a re-analysis of earlier field studies (Simone and Schmidt, 1993), and relating to literature on similar topics (e.g., Malone and Crowston, 1990; Johnson, 1992; Bogia *et al.*, 1993b).

To test, improve, and refine the conceptual framework, I conducted a field study of a group of software designers at Foss Electric A/S, a Danish manufacturing company. The study explicitly addressed how distributed activities were coordinated in a complex, cooperative work setting (cf. Carstensen, 1994; Carstensen and Sørensen, 1994a; Carstensen *et al.*, 1995b).

The analysis of the data from this study has been used for several different purposes: To discuss how different artifacts can be considered mechanisms supporting coordination work (Carstensen *et al.*, 1995b), to improve and refine the conceptual framework of Coordination Mechanisms (Simone and Schmidt, 1994; Schmidt *et al.*, 1995), to reflect on the use of a conceptual framework when analyzing coordination activities (Carstensen, 1995b), and to discuss how the notion of ‘organizational context’ can be described in terms of inter-related coordination mechanisms (Schmidt *et al.*, 1994; Schmidt *et al.*, 1995).

After analyzing the data from the field study, overall requirements for computer support of the coordination of software testing was promoted (Carstensen and Sørensen, 1994b; Carstensen *et al.*, 1995c). These requirements were related to the specific situation of coordinating software testing at Foss Electric, but they were also generalized and transformed into general requirements for coordination mechanisms.

Since the object-oriented paradigm is becoming ‘the norm’ in modern software design and the coordination mechanisms observed at Foss Electric had several characteristics identical to those of objects in the object-oriented paradigm, an experiment was conducted. The idea behind this experiment was to test the usefulness of object oriented analysis methods for describing real life coordination mechanisms. A re-analysis of one of the coordination mechanisms observed at Foss Electric was conducted by means of an object-oriented approach (Carstensen *et al.*, 1995a).

Finally, the implementation of a horizontal prototype of a system supporting software testers and designers at Foss Electric in distributed registration of software bugs and automatic routing of relevant information on the bug to other actors involved (Carstensen and Albert, 1995). This was done in order to qualify a discussion on overall requirements for coordination mechanisms, and illustrate preliminary ideas for how these can be reflected in the user interface.

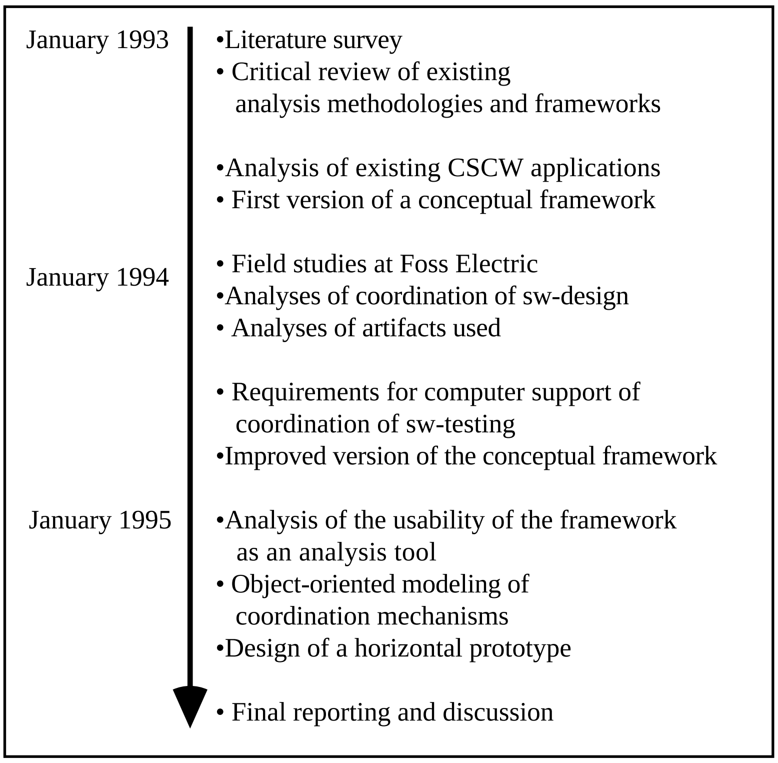

January 1993
January 1994
January 1995
•Literature survey
• Critical review of existing analysis methodologies and frameworks
•Analysis of existing CSCW applications
• First version of a conceptual framework
• Field studies at Foss Electric
•Analyses of coordination of sw-design
• Analyses of artifacts used
• Requirements for computer support of coordination of sw-testing
•Improved version of the conceptual framework
•Analysis of the usability of the framework as an analysis tool
• Object-oriented modeling of coordination mechanisms
•Design of a horizontal prototype
• Final reporting and discussion

**Figure 1-2:** An overview of the most important activities conducted during the dissertation work. It should be noticed that the actual process have included much more iteration and parallelism than what is illustrated here.

As it can be seen from the description above this dissertation attempts to span from contributing to a framework characterizing coordination work, then use the frame as an analysis tool, and finally illustrate and discuss how the framework can be used for designing computer-based coordination support.

### 1.2.2 The approach applied

As mentioned earlier, the research reported in this dissertation has basically been empirically driven. Furthermore, CSCW has been considered a design and construction oriented discipline, i.e., the goal has been to improve the development of computer support of cooperative work. Apart from mainly being empirically driven the work reported in this dissertation has had a qualitative approach. The work was an iterative process of analyzing existing CSCW systems, conceptualizing findings, collecting data through observations, interviews, etc., applying the conceptual framework for analyzing findings, establishing requirements for computer support based on the lessons learned from the qualitative analyses, refining the framework, and applying the conceptual framework for designing computer support.

As argued by, for example, Keyser (1992) and Siemieniuch (1992) field studies are required in order to obtain a coherent understanding of how computer tools can support product development in a manufacturing setting.

As mentioned earlier, a field study of a group of software designers and testers involved in a large instrument design project were conducted. Most of the ideas, concepts, requirements, etc. discussed in this dissertation is explicitly or implicitly related to the findings from this field study. The study focused on software design, especially activities concerning the coordination of the complex process of testing, diagnosing, and correcting software errors. The aim of the study was to analyze and characterize cooperative work in a large work setting where a number of participants with different perspectives, competencies, etc. had to deal with the complexity and uncertainty of testing a software product and determining when the product was acceptable.

The field study was exclusively based on qualitative data collection techniques. Qualitative interviews (Patton, 1980), observations, study of project documentation, and participation in project meetings were the primary techniques used to collect the data. A total of approximately 16 interviews have been conducted, and I participated in 6 project meetings.

Approximately 50 man-hours were spent observing the design and test work. The documentation studied were mainly a requirement specification (about 300 pages), design sketches, a pile of bug reports (the binder contained more than 500 different bug forms), spread-sheet based work plans, and short descriptions of organizational procedures (2-5 pages each).

A triangulation of data sources (Beraneck, 1994) was conducted by comparing observational data, interview data from different actors with different perspective, and data from document inspection. To further strengthen the validity of the findings a triangulation of collection methods (interviews, observation, and document inspection) was applied. During the analysis of the field study I had several meetings with the designers. At these meetings I confronted designers, managers, etc. with my findings and interpretations of their activities. These meetings led to changes and refinements, both in my understanding and analysis description, and in the designers understanding of their own activities. I, furthermore, compared the results with other analyses conducted at groups at Foss Electric working closely together with the designers I studied (cf., e.g., Borstrøm and Sørensen, 1994). Triangulation has been done both through use of multiple analysts (Beraneck, 1994) and through participant consultation (Coolican, 1994).

The approach applied in the field study was conducted by following some the ideas in and using the concepts of Work Analysis (Schmidt and Carstensen, 1990). Here the analysts apply a systems approach, and describe the functions offered by the system and the conditions under which they must be provided. The approach is inspired by, among others, Simon's understanding of artificial worlds (Simon, 1981), Checkland's ideas of soft systems (Checkland, 1981), and the tradition of cognitive engineering (cf., e.g., Rasmussen, 1986). The field study was also inspired by the ethnographic approaches (for example Bucciarelli, 1987; Bentley *et al.*, 1992a; Hughes *et al.*, 1993). The assimilation and understanding of the nature of the work studied has been achieved through frequently shifting perspective. It is while attempting to abstract and conceptualize an observed phenomena, or when searching for examples (real life phenomena) of the conceptual structures that the understanding of the essential characteristics of the work setting has been achieved. In establishing a model for user-developer communication Kensing and Munk-Madsen (1993) argue for similar shifts between the concrete and the abstract in order to obtain an overview of and to understand a domain of discourse.

Data was collected during interviews, attendance in meetings, etc. by means of tape recording, and taking notes. All tapes were transcribed afterwards. The data analysis was based on abstractions and conceptualizations of the patterns found in the data, followed by attempts to use these conceptualizations on other parts of the data material. The process of shuttling among data exploration, pattern generation, theoretical consideration, and pattern fill-in, suggested by Andersen *et al.* (1992), has partly been used as inspiration. During the analysis and interpretation of the phenomena, observations, statements from actors etc. the Concept of Coordination Mechanisms (previously also named Mechanisms of Interaction, cf., Schmidt *et al.*, 1993; Schmidt and Simone, 1995) has been applied—and improved in an iterative process. The concept is introduced in chapter 5.

The findings from the field study was also used for establishing requirements for computer support of the coordination of software testing at Foss Electric, and as a basis for designing a prototype for such a system. Both the requirements and the prototype design have been evaluated in two ways: First, these were presented to and discussed with some of the testers and designers at Foss. This led to a number of changes in the requirements and to a list of suggestions for improvement of the prototype. Second, both the requirements and the prototype design have been presented to, and discussed with, experts and colleges within the area of Human-Computer Interaction (HCI) and/or CSCW, for example Liam Bannon, University of Limerick, Kjeld Schmidt, Risø, Carla Simone, University of Torino, Carsten Sørensen, Risø, and Tuomo Tuikka, University of Oulu. These sessions have functioned as a kind of heuristic expert evaluation (cf. Nielsen, 1993). Qualitative input from these sessions have implied several changes to the requirements and the design.

### 1.2.3 The nature of the results presented

A qualitative approach based on a single field study offers the obvious strength of providing rich and detailed data, enabling a deep understanding of the conditions under which work is performed. It does, however, present a major limitation in terms of promoting statements of general validity. Mason (1989) calls this “The Fundamental Tradeoff in Observational Research”. He argues for a distinction between, on the one hand, richness of worldly realism, and on the other hand, tightness of control. Qualitative research approaches as applied in the field enables us to capture richness of worldly realism. Laboratory experiments which are conducted in a restricted fashion with the purpose of testing the effect of a very limited set of parameters do not provide such as richness, but have the advantage of tightness of control over the parameters.

McGrath (1984) gives a similarly detailed discussion of research strategies to be used when studying groups. He illustrates the ‘strategic dilemma’ as he calls it as illustrated in Figure 1-3. McGrath concludes that “field studies gain realism at the price of low generalizability and lack of precision. Laboratory experiments maximize precision of measurement and control of variables, at the price of lack of realism and low generalizability. Surveys have high generalizability but get it by giving up much realism and much precision” (Ibid., p. 30).

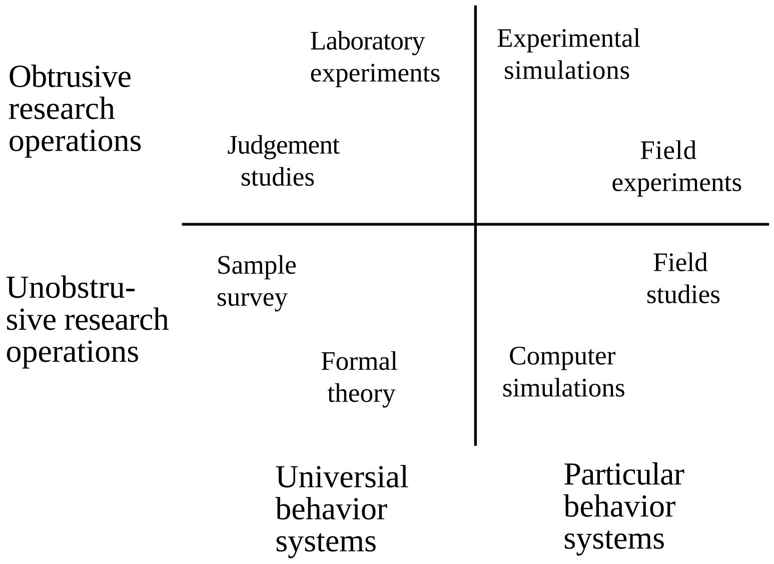


**Figure 1-3:** An illustration of 8 research strategies, their approach and limitations (Adapted from McGrath, 1984).

I have chosen the relevance and realism parameters as the essential. Orlikowski (1993) presents a thorough argumentation for using a grounded theory approach (Glaser and Strauss, 1967) combined with ideas from the innovation literature for studying and developing general frameworks for use of CASE tools. Following Yin (1989) Orlikowski calls the technique “analytical generalization” to distinguish it from typical statistical generalization, and she continues, “here the generalization is of theoretical concepts and patterns. This generalization is further extended [...] by combining the inductive concepts generated by the field study with insights from existing formal theory” (Orlikowski, 1993, p. 310). Although I have not used grounded theory, both the approach and the nature of the phenomena studied and reported here are very similar to those reported by Orlikowski. Flyvbjerg (1992) has a long and thorough discussion of the generality of findings from case studies. He sets up five “misunderstandings” regarding case studies. No. 2 of these states that “you cannot generalize based on a single case. That is why the case study cannot contribute to scientific progress” (Flyvbjerg, 1992, p. 138, my translation). Based on Giddens (1984) and by means of examples of the work of for example Freud, Marx, and Darwin, Flyvbjerg rejects the misunderstanding and states:

> “Correction no. 2: You can often benefit from generalizing based on a single case, and the case study can very well contribute to the scientific progress via generalization as a supplement or alternative to other methods. Formal generalization is over estimated as a source to scientific progress, whereas the ‘power of the good example’ is under estimated” (Ibid., 148-149, my translation).

Mason (1989) argues that the purpose of research must be to provide both the richness of detail and relevance of research problems studied, as well as a certain rigor. I do not believe, that one empirical effort necessarily needs to encompass both aspects, but I do recognize that since the results reported in this dissertation are drawn from a single field study, it might within certain aspects be difficult to make claims concerning the generality of the findings.

In order to characterize a research contribution Sørensen (1993) provides a useful structure: A two times two matrix with approach (theoretical or empirical) on the on dimension and the result (analytical or constructive) on the other dimension. My contributions filled into such a matrix can be seen in Figure 1-4.

| | Analytical result | Constructive result |
|---|---|---|
| Theoretical approach | **1**<br>• Approaches to coordination work | **2**<br>• Conceptual framework |
| Emperical approach | **3**<br>• Field study descriptions<br>• Conceptual framework<br>• Reflections on the usability of OO for modeling coordination | **4**<br>• Conceptual framework<br>• Reflections regarding work analysis<br>• Requirements for, and design of, coordination mechanims |

**Figure 1-4:** A matrix for characterizing research contributions presented in this dissertation. Based on a similar figure from Sørensen (1993).

In terms of this matrix, I will argue that the research contributions presented in this dissertation falls into several of the squares, although most of it is empirically driven, and thus belonging to the lower squares. The descriptions from the field studies can naturally be placed in square 3, whereas the reflections on requirements for computer-based coordination mechanisms and on how a conceptual framework can be used to support a work analysis must be placed in square 4. Also the conclusions from object oriented modeling of a coordination mechanism is regarded as a square 4 contribution. The literature survey of approaches to coordination work is a square 1 contribution, whereas the contribution to the establishment of a conceptual framework for coordination mechanisms can be regarded as belonging to both square 2, 3, and 4.

Hence, this dissertation is mainly having an empirical approach and the results are both analytical and constructive (normative).

## 1.3 My personal background

Before providing a guide for how to read this dissertation let me briefly introduce relevant aspects of my personal background. I have since 1984 been involved in work on usability and user interface design (cf. e.g., Beyer *et al.*, 1986; Carstensen, 1986). This work has mainly addressed methodologies for designing and evaluating user interfaces, but has also addressed how to identify 'relevant' requirements for computer based systems. From the late 1980'es, I have, in collaboration with Kjeld Schmidt, been involved in establishing a conceptual framework to support work analysis of complex work settings (cf. e.g., Schmidt, 1988; Schmidt and Carstensen, 1990; Carstensen and Schmidt, 1993b; Carstensen and Schmidt, 1993a). The intention behind this framework has been to provide a systemic functional oriented approach to understanding a work system. 'Functional' must be understood in its broad sense, i.e., in terms of the services a system provides to its purchasers. The framework was based on a critique of the traditional (procedural oriented) approaches, and was inspired by several well established traditions: Simon's ideas of distinction between inner and outer environment (Simon, 1981; Simon, 1983); Chekcland's ideas of having several perspectives on soft system (Checkland, 1981); The understanding of complexity and the systemic approaches to work systems in the tradition of Cognitive Engineering (e.g., Rasmussen, 1986; Woods, 1988; Woods and Roth, 1988); The early work of Hammer (Hammer and Sirbu, 1980; Hammer, 1984); The critique of the AI-tradition (e.g., Roth and Woods, 1989); etc. This is elaborated a little further in section 5.1.

Apart from the academic basis, it might also be worth mentioning that I, before starting at Risø as a researcher, have worked seven years as a programmer and systems designer, followed by seven years in R&D laboratories and a development support center. Here I mostly worked with research, teaching, and consulting within the area of Human-Computer Interaction.

## 1.4 How to read this dissertation

This dissertation is, of course, mainly thought of as reporting what I have conducted and achieved. It is, therefore, basically written for CSCW researchers. I have, however, attempted to address other possible readers too. Very briefly I expect the following types of readers to gain from reading (parts of) this dissertation:

- Researchers within the field of CSCW can be introduced to a conceptual framework (towards a theory) for central aspects of cooperative work, i.e., introduced to an attempt to model the structures coordination is based on. (especially chapter 4 and 5). Furthermore, they can be introduced to a field study of a type of cooperative work rarely reported in the CSCW literature, namely a study of work which is conducted over a long period of time, and characterized by long periods of none of very little interaction among the actors (chapter 3 and 6).
- Researchers within the software engineering field can gain from being introduced to central characteristics and conceptualizations of complex work that should be taken into consideration when designing complex application systems, designing systems software, or when developing new methods for analyzing, conceptualizing, and designing computer-based systems (especially chapter 4, 5, 7, 9 and 10).
- Furthermore, is it believed that system analysts and designers can get inspiration to include a new and relevant perspective when analyzing work settings or modeling systems to be used for cooperative activities. The concrete and detailed example of a work analysis (chapter 3, 4.3, and 6) is expected to be a source of inspiration for system analysts and designers.
- Finally, I hope that consultants—in for example a development support center—can get inspiration to improve the frameworks and methods currently used. For example by being introduced (in chapter 10) to a discussion of how one of the trendy software engineering paradigms (the object oriented) supports—and does not support—the designers need for understanding and modeling coordination aspects of systems, or from seeing the example of a computer-based coordination mechanism presented in chapter 9.

The dissertation is structured into four parts:

Part I (including this chapter) introduces the research field (chapter 2). Since much of the following is based on a field study an introduction to the work setting studied at Foss Electric is given in chapter 3.

Part II addresses aspects of the problems related to modeling coordination work. Cooperative work and its coordination is discussed in chapter 4 and the conceptual framework of coordination mechanisms is described in chapter 5. An example of a coordination mechanism identified in the field study is described in chapter 6. Part II is concluded (chapter 7) by some reflections on how a conceptual framework can be used to support the analysis of the coordination of complex cooperative work.

Part III discusses how computer support of coordination work can be organized. Chapter 8 identifies requirements for computer support of some of the coordination activities studied in the field study. In order to illustrate the ideas of computer-based coordination, chapter 9 provides a description of BRaHS, a horizontal prototype system. Chapter 10 illustrates and discusses how this mechanism could be modeled by means of object-oriented modeling techniques. Part III is concluded by a set of general recommendations for how computer-based coordination mechanisms should be designed (chapter 11).

Part IV concludes the dissertation. Chapter 12 discusses which lessons can be drawn from the work concerning establishment of a conceptual framework, doing work analysis of coordination work, and how computer-based support of coordination work should be designed. Chapter 13 discusses directions for future research in this field.

It is suggested that all readers read the Part I. For readers mainly interested in theory and conceptualizations relevant to CSCW, chapters 4, 5, 7, 12, and 13 will probably be the most interesting. To readers interested in work analysis and how this can be done in practice, chapters 5, 6, and 7 are the most central. If the readers interests are more in the direction of concrete design of systems supporting coordination work it is suggested to read chapter 5, 6, 8, 9, 10, and 11.

As mentioned earlier much of the content of this dissertation is based on papers that I have previously (co-)authored:

- Some parts of the descriptions in chapter 3 have previously been published in Carstensen and Sørensen (1994a).
- Some of the approaches discussed in section 4.3 are previously discussed in Carstensen (1995a).
- The findings described in section 4.4 have previously been discussed in Carstensen *et al.* (1995b).

- Chapter 5 aggregates findings reported in many different papers and reports on the concept of Coordination Mechanisms. These are cited in the chapter.
- The characteristics of the coordination mechanisms described in section 5.7 and 5.8 are also discussed in Carstensen *et al.* (1995b) and in Schmidt *et al.* (1995).
- Most of the analysis of a real-life coordination mechanism given in chapter 6 have previously been published in Carstensen (1994).
- Parts of the discussion of overall requirements for work analysis methodologies in section 7.1 have been published in Carstensen and Schmidt (1993b), and the discussion on the usability of the Concept of Coordination Mechanisms as a mean for analyzing coordination work has been briefly started in Carstensen (1995b).
- Chapter 8 on requirements is partly based on Carstensen and Sørensen (1994b) and Carstensen *et al.* (1995c).
- The prototype introduced in chapter 9 has been more thoroughly described in Carstensen and Albert (1995).
- The experiment of object oriented modeling of a coordination mechanism and its results described in chapter 10 have previously been published in Carstensen *et al.* (1995a).

A complete list of all papers I have considered relevant produced during the work with this dissertation can be seen in Appendix A.

# 2. Related research

"Collaborative work is the core of our society, wrought with difficulties and benefits. It is clear that technology can change group work, and there is a good possibility that it can result in major enhancements to productivity. But, there is a lot of work to do before we fully understand how to accomplish that. Trial and error from creative system builders is too slow a discovery process. What is required is a better understanding of the nature of group work, the extent of the possibilities of the design space of technology features, and evaluation of systems in use that leads to a theory of computer supported co-operative work, which in turn can help us direct subsequent invention of new ways to do group work" (Olson *et al.*, 1993).

As a research field, the field of CSCW (Computer Supported Cooperative Work) is relatively new. A generally accepted definition has not yet been established (cf. Bannon, 1993). There is still a lot of confusion and debate of which overall research questions should be addressed. This chapter is by no means an attempt to solve this. I will present what I consider the most relevant dimensions for characterizing the field, and briefly introduce the approach taken by some of the central players in these dimensions. The aim is to relate my own position in the "map of CSCW" to other traditions and approaches.

The literature contains, of course, many more or less different definitions of what CSCW is, what groupware is, what characterizes groupware technology, etc. Since many publications in the CSCW field contain their own definitions it is more or less impossible to provide an overview. It exceeds the scope of this dissertation to go into detailed discussions of these definitions, but a brief introduction to some of them might be relevant:

According to Grudin (1991) groupware, or CSCW systems, is defined very differently by different researchers. Some will consider technology providing access to shared files as CSCW systems (Crowley in Ensor, 1990); others will consider e-mail systems as CSCW systems (e.g., Kraut in Ensor, 1990); whereas for example Bannon and Schmidt argue, that

CSCW systems must be based upon, and reflect, an understanding of the cooperative aspects of the work to be supported, i.e., e-mail that do not recognize any roles beyond the receiver and the sender is not to be considered a CSCW system (Bannon and Schmidt, 1989). E-mail and facilities providing shared file servers, etc. should be seen as enabling technology.

The introduction of the CSCW field has also been used to argue for a paradigm shift in software development. Grudin (1990) suggests that CSCW can be an opportunity to stress the “importance of ‘workplace democracy’—engaging the users or workers meaningfully in the design process [... although] this time-consuming, labor-intensive approach may not be equally appropriate for all development projects ” (Grudin, 1990, p. 101). Several have defined cooperative design as cooperative work and used this to argue for seeing the tradition of Participatory Design (PD) as an aspect of CSCW (e.g., Kyng, 1991). But, as argued by Bannon this seem to add confusion to both fields rather than defining the field of CSCW: “While certainly various forms of user involvement are important to development of successful CSCW systems use of [PD] techniques or ideas does not automatically signify any focus on cooperative work [...] Nor, in many cases, are PD researchers interested directly in computer support for the design practices they are proposing” (Bannon, 1993, p. 11). Like Bannon, I think it is a mistake to consider PD as part of CSCW or vice versa. I would rather think of PD as a tradition that has developed and introduced a number of approaches and techniques which, of course, should be applied when designing CSCW applications.

Hughes *et al.* (1991) argue that CSCW should be seen as a paradigm shift. Instead of seeing sociology as having a service role in CSCW, CSCW should be seen as posing a challenge for sociology: “A new theoretico-empirical terrain is being formed, as much in the sociology of work and organisations as elsewhere, and the interdisciplinary confrontations invoked in CSCW can be a formative influence” (Hughes *et al.*, 1991, p. 321). They then try to relate this to systems design without clearly defining what is meant by the term design. I do not find the paradigm shift discussion central to what is discussed in this dissertation, but like the PD approach, this approach contains several useful propositions, for example techniques for analyzing cooperative work, and a deeper and more coherent understanding of the magnitude of aspects of a work situation that are important to understand compared to traditional computer science.

Schmidt and Bannon have discussed useful approaches to CSCW in several papers. This resulted in a definition focusing on the need for understanding the nature of cooperative work in order to establish a foundation for designing information systems to support the work: "CSCW should be conceived of as *an endeavor to understand the nature and requirements of cooperative work with the objective of designing computer-based technologies for cooperative work arrangements*. [...] in the conception of CSCW proposed here—as a research area devoted to exploring and meeting the support requirements of cooperative work arrangements—CSCW is basically *a design oriented research area*. This is the common ground. Enter, and you must change." (Schmidt and Bannon, 1992, p. 11). The idea of considering CSCW primarily a design discipline fits, as argued earlier, with my approach. This approach calls for an attempt to establish a better conceptual understanding of what characterizes cooperative work and its coordination.

The approaches mentioned above are only examples, but it should be sufficient to illustrate that CSCW is not a well-defined research field yet, and it might never become one.

The rest of this chapter will provide a brief introduction to some of the topics and approaches included in the CSCW literature. Apart from the CSCW field and other related areas this dissertation has also been inspired by, and related to, the more traditional Computer Science and Information Systems literature. I have, for example, prior to the dissertation work conducted a long critical review of analysis methodologies within Computer Science and the office automation tradition, and discussed the applicability of these methodologies for modeling cooperative work (Carstensen and Schmidt, 1993a). Whenever relevant, this literature is referenced, but this chapter will not attempt to provide an overview of these fields.

The rest of this chapter will, of course, reflect which parts of the literature I have looked at, and this implicitly reflects the relevance of the literature. This might explain an eventual "lopsidedness" in the descriptions. I have, furthermore, been involved in the Esprit Basic Research project COMIC for three years, and have therefore been heavily influenced by work conducted by CSCW researcher from Lancaster University, University of Manchester, GMD in Bonn, University of Milan, University of Limerick, and Risø.

## 2.1 Dimensions characterizing the field of CSCW

CSCW is a new research field. A generally accepted definition and frame for how to classify the works within the field has not yet been established. In this section I will present a preliminary set of dimensions characterizing the CSCW field. The main purpose is to provide a frame for the introduction to the field given in the following sections.

Inspired by Johansen (1988), CSCW technologies are often characterized by means of a two-times-two matrix having synchronous vs. asynchronous communication (time dispersion) on the one axis, and having distributed actors vs. non-distributed actors (geographic dispersion) on the other (cf. e.g., Baecker, 1993).

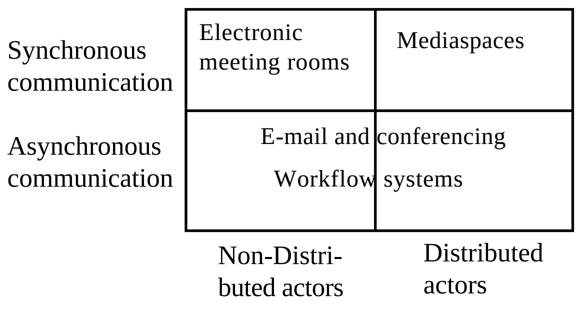


**Figure 2-1:** An illustration of Johansen's two-times-two matrix. For illustration purposes a few examples have been inserted.

The two-times-two structure is very useful for characterizing CSCW applications, although many of these have different facilities falling in different boxes. The structure is, however, insufficient to characterized the research field of CSCW. It contains, for example, no structure for placing all the field studies of cooperative work conducted within CSCW, and all the work on applying theories from related fields (e.g., communication theory) in order to improve our understanding of cooperative work is excluded too.

Inspired by the structure used in a CSCW paper collection edited by Ronald Baecker (1993), the matrix for classifying CSCW-systems established by Johansen (1988) and by the overviews of the CSCW field provided in Bannon (1993), Grudin (1991), and Ljungberg (1994), I will suggest the following—non-orthogonal—dimensions to characterize the

CSCW-field: 1) Studies of cooperative work; 2) Evaluation of CSCW systems in use; 3) Conceptualization of cooperative work; 4) Development methodologies; 5) Design of CSCW systems for synchronous use; 6) Design of CSCW systems for asynchronous use; And 7) Design of architectures and platforms for CSCW systems design. The dimensions are visualized in Figure 2-2 below.

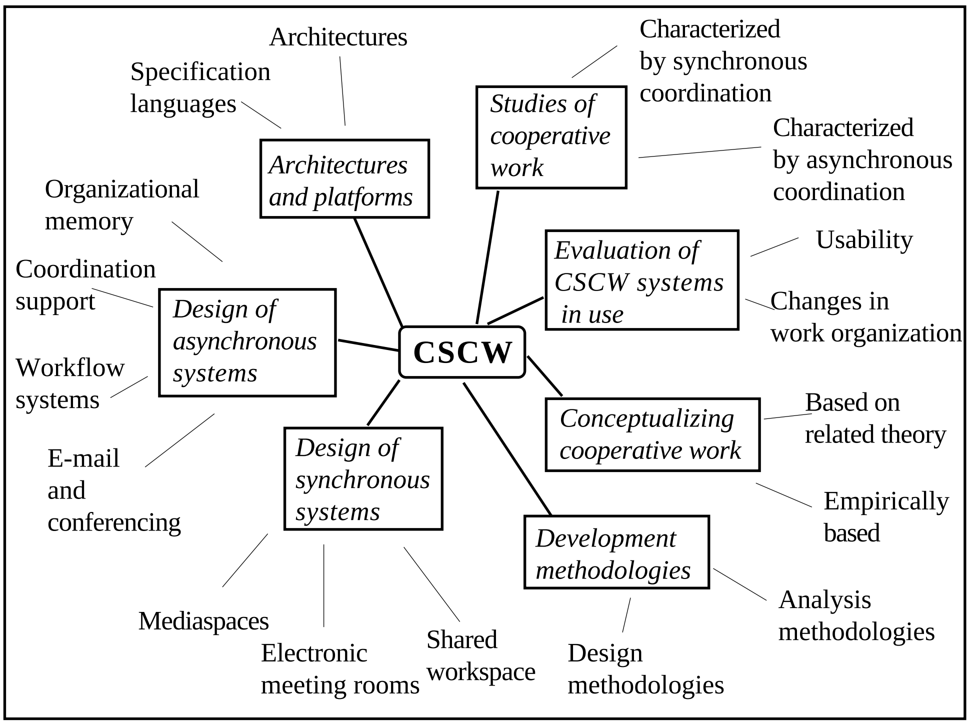

CSCW
Architectures and platforms
Architectures
Specification languages
Studies of cooperative work
Characterized by synchronous coordination
Characterized by asynchronous coordination
Evaluation of CSCW systems in use
Usability
Changes in work organization
Conceptualizing cooperative work
Based on related theory
Empirically based
Development methodologies
Analysis methodologies
Design methodologies
Design of synchronous systems
Mediaspaces
Electronic meeting rooms
Shared workspace
Design of asynchronous systems
Organizational memory
Coordination support
Workflow systems
E-mail and conferencing

**Figure 2-2:** A visual overview of the seven central dimensions suggested for characterizing the research field CSCW. Each box contains one of the non-orthogonal dimensions. The "topics" related to each of the dimensions should be regarded as examples of typical themes related to the dimension.

As mentioned, the seven dimensions are not orthogonal. Most of the research within CSCW could be related to several of the dimensions (categories). The purpose of the dimensions has, however, mainly been to impose a structure for introducing parts of the literature within CSCW!

The following sections each provide a brief introduction to some of the approaches and works for each of the dimensions. I have chosen to mention a specific body of work in the dimension (section) where I found it most apparent. This will, of course, often be debatable. Those naturally belonging in several dimensions are only mentioned in the most apparent section.

## 2.2 Studies of cooperative work

The CSCW literature includes a lot of reports from field studies aiming at providing a deeper understanding of the cooperative aspects of real-life work or other important characteristics before designing software support.

Some of the most referenced are the studies of control work at the air traffic control center outside London (cf. e.g., Harper *et al.*, 1989b; Hughes *et al.*, 1993). The central findings in these studies were that harmony and good social relations among the air traffic controllers, and awareness of each others strength and weaknesses among the controllers were important for the handling of the work. Furthermore, the flight strips used were analyzed as a communication medium and a means for mutual awareness of current state of affairs. The flight strip analysis has later been used as input for discussions of computer support of the work, and how ethnographic studies can be used to inform systems design (Sommerville *et al.*, 1991; Bentley *et al.*, 1992a).

At the London Underground some interesting studies have been conducted of how actors in a control room are sensitive to, and monitor, activities within the local work area, while participating in different activities (Heath and Luff, 1992). Heath, Luff and associates have conducted similar studies of work in a dealing room in the City of London (Heath *et al.*, 1993). These studies illustrated the same awareness as the control room study, but here the dealers were sensitive to general information provided (‘open cries’) too. This often led to focused collaboration between dealers. Examples from the control room and the dealer study have, furthermore, been used to argue for a more flexible and more dynamic perception of what constitutes “the work context” (Heath *et al.*, 1995).

Suchman and Wynn (1984) have studied clerical workers and their use of procedures in what is usually called procedural work. Their overall finding is that “operational significance of a given procedure or policy on actual occasions is not self-evident, but is determined by workers with respect to the particulars of the situation at hand” (p. 152). Suchman follows this statement in her famous book on plans and situated action by illustrating that making errors, improvisation, handling of exceptions, etc. are typical characteristics of human activities, that most work is characterized by situated actions, and that plans thus are resources for actions only (Suchman, 1987).

Most of the empirical studies in the CSCW field analyze work settings involving many interdependent actors, but they primarily focus on the work of relatively few. And the domains investigated are most often characterized by a high degree of monitoring and regulating (often time-critical) activities among the actors (cf. the examples given above). Some ethnographic studies of the cooperative aspects of work conducted over a long time span do exist. For example Anderson *et al.* (1993) reports a study of designers designing a feeder for a photocopier. They especially address how the characteristics of the 'organizational life' affects the work and thus must affect the design of support tools, etc. Our own studies (Sørensen *et al.*, 1994; Carstensen *et al.*, 1995b) have also addressed design work conducted over long time spans. Results from this will be discussed in the following chapters.

A more quantitatively oriented survey of cooperative work in software development has been conducted by Kraut & Streeter (1995). They collected data from 65 different projects focusing on means used for coordinating the activities. Their conclusion was that the informal communication needs to be nurtured, and that tools and methods must provide support for inter-personal communication to be supportive. They do, however, also conclude that "direct contact between organizational members is not a panacea for coordination problems in large software projects [...] Thus, a large project will also need formal communication" (Ibid., p. 79).

The work mentioned in this section can all be regarded as contributing to one of the other dimensions of CSCW, namely the conceptualization of cooperative work (discussed further in the next section). Although it contains very detailed descriptions, most of the work related to the air traffic control study is, however, characterized by lack of conceptualizations and abstractions. This seems to be against the ethnomethodological tradition. Heath and associates have through their studies of London Underground definitely contributed to a better conceptual understanding of cooperative work. They have are to be honored for the focus on awareness as an important factor in cooperative work. Suchman and associates' thorough studies have contributed to our conceptual understanding of work by explicitly addressing the situated nature of all kinds of work (or rather human action in general). And our own work at Risø have primarily focused on how to conceptualize central aspects of cooperative work.

So, most of the work categorized as "studies of cooperative work" here could also be considered as contributing to "conceptualization of cooperative work" discussed in the following.

## 2.3 Conceptualization of cooperative work

A number of models, theories, conceptual frameworks, etc. for understanding and conceptualizing cooperative work and its coordination have been suggested. The basis for these are findings from field studies, application of theories from related fields, or concrete experiences from modeling cooperative work settings. The most relevant of these will be thoroughly discussed in chapter 4. This section will briefly introduce some of these models, theories, conceptual frameworks, etc.

Strauss was the first to explicitly distinguish between work and what he calls 'articulation work'. His work was based on, among others, empirical studies of medical workers (Strauss *et al.*, 1985). Strauss argues that in order to handle the underlying interdependencies among the cooperating and mutually interdependent actors, a set of 'second order activities' handling the articulation of activities and resources is needed. The actors must, in often complex ways, articulate the plurality of tasks making up their totality, and the relations of actors to tasks, i.e., mesh tasks, actors, and organizational structures (Strauss, 1985). The approach is later outlined into a seven dimensional theoretical scheme for studying the articulation of project work (Strauss, 1988).

Gerson and Star (1986) have used the work of Strauss to define articulation work as consisting of all the tasks involved in assembling, scheduling, monitoring, and coordinating all of the steps necessary to complete a task. Leigh Star has later refined their approach further and established a concept of Boundary Objects (Star, 1989), i.e., objects that are supportive for a collaborative course of action. The conceptualizations provided by Gerson and Star are based on findings from field studies and attempts to model aspects of the work observed.

Inspired by Strauss, Gerson and Star, and others, Schmidt, myself, and others have established a conceptual framework for understanding cooperative work and its articulation (Schmidt, 1991b; Schmidt and Bannon, 1992; Schmidt, 1994c). This approach identifies a distinction between field of work, work arrangement, and the work environment (Carstensen and Schmidt, 1993b; Schmidt, 1994c), and it suggests a set of conceptualizations (objects of articulation) that are essential when modeling articulation work (Schmidt *et al.*, 1993; Schmidt and Simone, 1995). The conceptualizations suggested in the framework is derived from re-analyses of existing field studies, studies of existing CSCW applications, and new field studies. Also, theories of complexity and work modeling from the Cognitive Engineering tradition (e.g., Rasmussen, 1986; Woods, 1988) have inspired the development of the conceptual framework.

Malone and Crowston have also recognized the need for a set of concepts based on “fruitful interdisciplinary connections concerning coordination” (Malone and Crowston, 1990, p. 357) supporting design of computer-based coordination support. They argue for a narrow definition of coordination focusing on managing interdependencies. Based on a discussion of different kinds of interdependencies they conclude that these are the unique aspect of coordination, and this is what is to be supported. Malone and Crowston base their suggested conceptualizations on experiences from designing a basic language for designing CSCW applications (Lai and Malone, 1988), and on attempts to combine and apply theories and frameworks from related research fields, e.g., linguistic (they reference Winograd and Flores, 1986; Flores *et al.*, 1988), philosophy and rhetoric (they reference Conklin and Begeman, 1988; Lee, 1990), or economics (with references to unpublished work Murrey of Turoff).

Winograd and Flores (1986) have developed a perspective on cooperative work inspired by ideas from linguistics. This ‘language / action’ perspective is used for understanding the different ‘speech acts’ going on in cooperative settings. The speech acts are described and modeled in terms of declarations and authorizations, requests, offers, acceptances, rejects, and promises (Winograd, 1992). The perspective was a primary basis for the design of The Coordinator (Flores *et al.*, 1988). The approach is, as mentioned, inspired by language/action having a language philosophical basis, and conversation analysis derived from generalizations of empirical findings. The work by Winograd and Flores have also been used for establishing ideas of how to analyze and design computer support for cooperative work (Kensing and Winograd, 1991). The work has thus contributed to the dimension of CSCW I have called ‘Development methodologies’ (cf. section 2.5).

Since people do not strictly follow rules and procedures we should instead model cooperation and cooperative work in terms of goals. This is the overall conclusion in an approach to cooperative work and CSCW systems suggested by Ellis and Wainer (1994). They offer, based on this overall idea, a functionally-based taxonomy for understanding the kind of support CSCW systems provide, or should provide. The dimensions in the taxonomy are related to the support of 1) access to and sharing of data space, 2) synchronization of activities, 3) communication, and 4) how to handle the entire CSCW system. Ellis and Wainer do recognize that most systems are, and should be, a mixture of the categories. The taxonomy is established from a theoretical basis.

Many other approaches could be discussed here, especially if approaches from related fields such as organizational theory or social psychology are included. Mintzberg, for example, has established a model for understanding the coordination going on in organizations in terms five different “coordination mechanisms” (Mintzberg, 1983). Mintzberg will be briefly discussed in chapter 4, but apart from this, other approaches from sociology (e.g., Sharrock and Anderson, 1986; Button, 1993), organizational theory (La Porte, 1975a; Thompson, 1983), cognitive engineering (e.g. Rasmussen *et al.*, 1994), etc. have been considered out of scope.

## 2.4 Evaluation of use of CSCW systems

One way of learning about cooperative work and how to computer support it is, of course, by studying and evaluating how existing CSCW applications are used in real work settings. To illustrate this I will briefly mention two well-known studies:

Based on involvement in several projects evaluating or designing applications Grudin has identified five factors contributing to groupware failure: (1) actors were requested to do work they did not perceive benefit for, (2) led to violation of social taboos and political structures, (3) did not open up for the range of exceptions required, (4) were so complex that it was almost impossible to learn from experience with previous systems, and (5) design process had failed due to lack of intuitions for multi-user applications (Grudin, 1989; Grudin, 1990).

As a counterpart Grudin has later published discussions on why groupware succeeds (Grudin and Palen, 1995). The previous findings are reviewed and compared with new studies of using groupware tools for scheduling meetings. It is concluded that improvements in the technical infrastructure, an expanded functionality, the improvement of the user interface, and the “peer pressure” for using the systems in modern groupware tools and their use patterns have resulted in an improved adoption to the tools and an increased success rate. This has attracted a critical mass to use the systems.

Another important and often referenced study is Orlikowski’s famous study of the introduction and use of Lotus Notes in a large office in a services firm. Her interest was to “investigate whether and how the use of a collaborative tool changes the nature of work and the pattern of social interactions” (Orlikowski, 1992, p. 362). Based on a long series of interviews and observations it was concluded that the way people act towards and relate to new technology depends mainly on their understanding of it. And since people have no understanding of the collaborative nature of groupware, the technology will be interpreted as personal software. Thus “a central aspect of implementing groupware is ensuring that prospective users have an appropriate understanding of the technology” (Ibid., p. 368). A second overall conclusion was that the structural properties (policies, norms, etc.) influenced the utilization of the groupware technology. Work settings based on competition and individual rewards will have few and insufficient norms for sharing and cooperating. Without some “shared norms groupware will likely be used primarily for advancing individual productivity” (Ibid., p. 368).

## 2.5 Development methodologies

The penetration of CSCW as a research field has implications for people discussing how to organize software development projects. The CSCW center at Lancaster University has, based on lessons learned from a series of field studies, argued for the use of ethnographers in the software development process (Sommerville *et al.*, 1991; Bentley *et al.*, 1992a; Hughes *et al.*, 1992; Hughes, 1993). The basic arguments are that the kind of information ethnographers can provide is very important for understanding the cooperative (and social) aspects of working life, and that ethnographers can substitute users in a ‘user-centered’ design.

The same series of studies also pinpointed some of the limitations of the ethnography (Shapiro, 1994). The conclusion here was that ethnography used isolated will provide as simplistic a picture as other approaches. Instead Shapiro argues for hybrid work forms including ethnography, cognitive science, participatory design, computer science, etc.

The studies of how existing CSCW applications are used (e.g., Grudin, 1989; Orlikowski, 1992) have also provided input for how to organize the development of CSCW applications (cf. previous section).

As mentioned the work by Winograd and Flores on the language/action approach (cf. section 2.3) has also been used by Kensing and Winograd for establishing ideas on how to analyze and design computer support for cooperative work (Kensing and Winograd, 1991). The result is an exemplification and discussion of how a language/action perspective can be used for structuring and analyzing observations of the flow of work and the conversations encountered. Furthermore, it is briefly discussed how techniques from the Scandinavian participatory design tradition (e.g., Bjerkenes *et al.*, 1987; Andersen *et al.*, 1990) can be used for re-designing the work and the computer systems.

Holt (1985; 1988) discuss how a graphical language based on ideas from Petri Nets can be used to build systems supporting coordination, or build “coordination technology” as he calls it. Holt mainly discusses a notation for specifying coordination technology, and provides only few ingredients for a development methodology, apart from introducing “a new formal graphical ‘language of plans’, called diplans, whose newness lies in the meaning it brings to consciousness, rather than in its encoding of meanings already there” (Ibid., p. 124).

Finally, some of my own findings from the field study at Foss Electric have been used for reflecting upon requirements for support of the analysis of complex cooperative work (Carstensen, 1995b). These requirements will be discussed further in chapter 7.

A concluding remark in this section might be that very limited literature seems to explicitly address methodologies for analysis and design of computer-based systems supporting the cooperative aspects of work. Although this is not exactly the core of this dissertation, conceptual frameworks for modeling cooperative work and methodologies for designing CSCW applications are closely related. Chapter 7 will therefore include some methodological reflections too.

## 2.6 Design of CSCW systems for synchronous use

I have chosen to distinguish between CSCW systems intended to be used in a synchronous manner and systems to be used asynchronously. Of course, many systems provide facilities for both types of interaction. This section will discuss approaches for CSCW systems for synchronous use, and section 2.6 will relate to interesting approaches for designing CSCW systems for asynchronous use.

One of the prevalent approaches to design of systems to support synchronous interaction and cooperation is the idea of establishing a shared workspace. The idea is, based on lessons from early What-You-See-Is-What-I-See systems (e.g., Stefik *et al.*, 1987), to provide the users with all their individual and familiar tools. These should be integrated in a computer-based system by which the users can communicate, interact, cooperate, share workspaces (like documents and boards), etc. dynamically over space constraints (Ishii and Miyake, 1991). The goal is to provide a "seamless collaboration media" (Ishii *et al.*, 1994). The illustration systems typically contain shared workspaces where the actors can see each others manipulations, video cameras so the actors can see each other, voice transmission allowing oral communication, etc. Some more simple systems provides only some of the facilities, e.g., shared whiteboard facilities (e.g., Pedersen *et al.*, 1993), or video conferencing systems (e.g., Isaacs *et al.*, 1994).

Another tradition concerns collaborative writing, i.e., systems that enables a number of users distributed in space to work on the same document at the same time, e.g., GROVE (Ellis *et al.*, 1991), or ShrEdit (Olson *et al.,* 1990). Research in this field has focused on, for example, what characterizes collaborative writing processes (e.g., Posner and Baecker, 1993), how to support awareness in synchronous writing (e.g., Baecker *et al.*, 1993), or how to support that co-authors offer and make use of information to and from each other (Beck and Bellotti, 1993).

A third approach is the idea of establishing a media space integrating video and audio technology with network technology so that face-to-face communication can be supported although the actors are distributed in space. Xerox's RAVE system (Gaver *et al.*, 1992) is the most well-known. This system has been used for a number of studies resulting in discussions on, for example, control mechanisms in this kind of systems (Dourish, 1993), how privacy influences the use of, and can be handled in, media spaces (Bellotti and Sellen, 1993), ideas for supporting awareness (Gaver, 1991), and how to establish a joint frame of reference for the interaction in such systems (Gaver *et al.*, 1993).

## 2.7 Design of CSCW systems for asynchronous use

As for the synchronous systems a long series of systems and design projects exists. Only a few, representing some central approaches, will be mentioned in this section.

Some of the first systems (approaches) that were called CSCW were e-mail and conferencing systems. Engelbart was one of the first to discuss these ideas (cf., Engelbart and Lehtman, 1988). E-mail and conferencing systems have existed and been used for several years, and a lot of experiences from using them exists (e.g., Murrel, 1983). They will not be discussed further here.

Some CSCW systems are labeled 'organizational memory', i.e., "the record of an organization that is embodied in a set of documents and artifacts" (Conklin, 1992, p. 133). Here focus is on how the data (or artifact) oriented paradigm for recording information can be expanded to include relevant recording of the 1) context in which the result have been produced, and 2) the process followed when producing these results. The central idea is to build systems supporting traceability, reuse of ideas, establishing consensus, etc. An example of such systems is Answer Garden (Ackerman and Malone, 1990) that provides facilities for routing questions within an organization to relevant experts, or gIBIS which support discussing, establishing, and keep track of the design rationale for a software design project (Conklin and Begeman, 1988). A comparable system is EGRET (Johnson, 1992) that supports ‘exploratory group work’ (dynamic and unpredictable) in software engineering by providing a structure of schemes describing tasks, specifications, etc. EGRET keeps track of the variations by offering functionality for listing registered deviations from the schemes.

Another predominant approach is the idea of developing work flow systems that aims at automating or supporting the flow of tasks, data, actors, and recurrent or non-recurrent events in order to improve the efficiency and effectiveness of organizations (Cruse, 1992). In this approach (or paradigm) organizations are seen as networks of intertwined flows of work rather than of physical entities and structures (Winograd, 1992). The basic units used for modeling the work thus become the workflows. DOMINO (Kreifelts *et al.*, 1991b) is an example of a system in which office processes are modeled in terms of information flows including concurrent flows and processes and alternatives courses of activity. The Coordinator (Winograd, 1986) mentioned earlier is a workflow modeled in terms of ‘conversations’ (requests and offers, agreements, assertions, and assessments). So, although The Coordinator is not a typical workflow system it is often labeled as such.

The last approach is a characterized by aiming at supporting the coordination and cooperation going on. The idea is to make models of the work processes and then specify different kinds of support for these. Focus has often been on planning and structuring the work, and then specify models of the work in specification languages, rules, PERT-charts, etc. (Swenson *et al.*, 1994). Such systems are usually shells or environments that can be ‘programmed’ for a specific situation. An early example is Officetalk (Ellis and Nutt, 1988). It was intended to support cooperative management and preparation of office documents. It contained facilities for handling in and outgoing mail, defining different forms to be used, etc. A newer example is ConversationBuilder (Kaplan *et al.*, 1992b) which provide “appropriate mechanisms for the support of collaboration rather than specific policies. Policies can be built out of mechanisms, if the right mechanisms are provided” (Bogia *et al.*, 1993a). The conversation for action theory (Winograd and Flores, 1986) has been used as a basis, and

ConversationBuilder is basically based upon an understanding of both work and coordination work, and that these on the one hand are intertwined and on the other hand need to be supported ‘orthogonal’ to each other. ConversationBuilder have active support of “*orthogonality*. By this we mean searching for ways of supporting tasks while remaining in some sense orthogonal to the work of the tasks themselves” (Kaplan *et al.*, 1992a, p. 1). A third interesting system is Regatta (Swenson *et al.*, 1994). It provides support for planning work processes. This is done by modeling the communication required to coordinate the relevant tasks. The modeling is intended to be done in an iterative process by the users themselves: “process plans can be created and modified by the end user allowing users to experiment and find the best processes” (Ibid., p. 16).

Most of the systems discussed in this section can, of course, be used for synchronous interaction too, and could thus have been categorized in section 2.6 as well. A central characteristic for the systems mentioned here is, however, that they contain facilities for asynchronous interaction.

## 2.8 Architectures and platforms for designing CSCW systems

Several of the ‘platforms’ specified for supporting the development of CSCW systems are mentioned above. Both The Coordinator, ConversationBuilder, and Regatta can be defined as platforms for building CSCW applications.

Some have attempted to specify a language including structures that specifically supports the development of CSCW applications. The most well-known example is the specification language OVAL (Malone *et al.*, 1992) which is intended to be a general notation for expressing coordination work. The basic primitives (objects, views, agents, and links) are general and at a very low level. We have earlier critiqued OVAL for not reflecting the relevant primitives used when coordinating work in actual work settings at a proper semantic level (cf., Schmidt *et al.*, 1993), i.e., the primitives do not reflect the conceptualizations of the essential components in coordination work at a level ‘natural’ to the user.

Others have discussed what kind of requirements the nature of CSCW application sets up for the basic software (operating system, database system, communication protocols, etc.). Some of these have discussed the implications for the ‘distribution technology’. Cooperating CSCW application users will be separated geographically in the future. This will have implication (Rodden and Blair, 1991).

Shen and Dawan (1992) have explicitly addressed control aspects of CSCW applications, and others have discussed how architectures supporting CSCW must be organized (e.g., Bentley *et al.*, 1992b; Jeffay *et al.*, 1992). Based on these discussions and ideas attempts to build basis platforms for CSCW systems exist (e.g., Trevor *et al.*, 1993).

## 2.9 The position taken

To place my own work in one of the categories used for structuring this chapter is very difficult, exactly as it has been for many of the mentioned works. If one category should be chosen for placing the work presented here it would be ‘Conceptualization of cooperative work’. The basic idea has been to contribute to a better conceptual understanding of what characterizes cooperative work, especially its coordination.

Several of the activities conducted in order to fulfill this idea would, however, naturally belong to other categories. I have spend much effort in studying cooperative work, establishing requirements for computer support, and designing a horizontal prototype. So, taken individually many of the papers I produced forming a basis for this dissertation would be placed in other categories, such as ‘studies of cooperative work’ (Carstensen, 1994; Carstensen and Sørensen, 1994a; Carstensen *et al.*, 1995b), ‘evaluation of CSCW systems’ (Andersen *et al.*, 1993), ‘development methodologies’ (Carstensen and Schmidt, 1993a; Carstensen and Schmidt, 1993b; Carstensen *et al.*, 1995a; Carstensen, 1995b), and ‘design of asynchronous systems’ (Carstensen and Sørensen, 1994b; Carstensen and Albert, 1995; Carstensen *et al.*, 1995c).

As mentioned earlier, I consider myself belonging to the information systems tradition, and I consider CSCW—and thus what I have done—as mainly design oriented, i.e., the aim is to improve the design of computer-based support systems.

Hirschhiem and Klein (1989) identifies four prototypical paradigms for information systems development: The functionalist paradigm concerned with explanations of social order and integration, need satisfaction, and rational choice. It focuses on how elements in a social system interact to form a whole. The social relativism paradigm is characterized by seeking "explanation with in the realm of individual consciousness and subjectivity, and within the frame of reference of the social actor as opposed to the observer of action. From such a perspective roles and institutions exists as an expression of the meanings which men attach to their world" (Ibid., p. 1201). The radical structuralist paradigm aims at transcending existing social and organizational arrangements focusing primarily on economic power. The neohumanism paradigm focuses on barriers to emancipation, especially ideology, power, and social constraints.

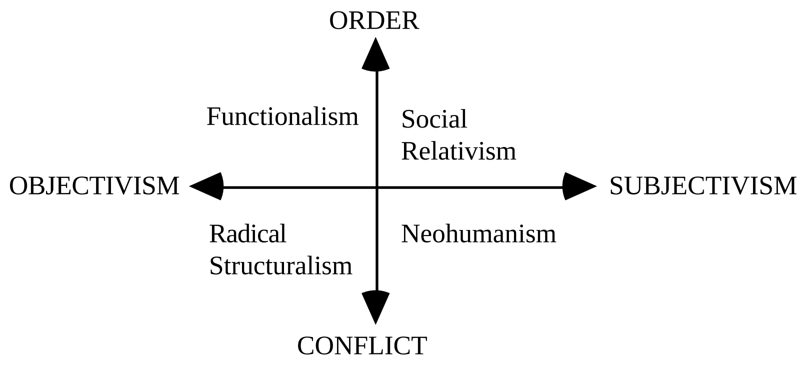


**Figure 2-3:** The four paradigms for information systems development according to Hirschhiem and Klein (1989).

It is not obvious where in this matirx to place the work I have conducted. Most of the work should probably be placed in what Hirschhiem and Klein call functionalism: The work on establishing a conceptual framework (cf. chapter 5 and 7), the work on requirements for a computer-based system (cf. chapter 8), the work on modeling and designing a computer-based system (see chapter 9 and 10), and the normative recommendations for design of coordination mechanisms discussed in chapter 11 can all be regarded as concerned with explanations of social order and integration, need satisfaction, and rational choice, i.e., functionalism according to Hirschhiem and Klein. There are, of course, aspects that can be seen as radical structuralism is this. The nature of the field study conducted could, however, be categorized as both functionalism and social relativism. Here the approach addresses both the individual social actor and the social system as a whole.

In the terms of Dahlbom and Mathiassen (1993) I consider myself and my approach as mainly belonging to what they call the 'Intervention School' (in opposition to the Systems Construction and the Systems Evolution approaches). As the name says the approach is characterized by intervention:

> "It is our job to look for structured domains of information usage, where stable forms of information or stable procedures for processing information exists or can be established. The identification of relevant structured domains requires technical competence, but the activity is carried out as an integral part of evaluating different proposals for a new management system. The new computer system is viewed as one important part of a wider organizational system" (Ibid., p. 117-118).

The development of a conceptual framework (described and discussed in chapter 5 and 7), and the recommendations presented in chapter 10 are intended to be contributions within an intervention oriented approach. I consider the ideas presented in this dissertation as concerned with design of computer-based systems, but the ideas of computer-based coordination support will only make sense if they are viewed as an important part of a wider organizational system. The same goes for the field study findings reported: The abstractions and conceptualizations should mainly be made by actors having some technical competence, but they are only relevant and useful if they are understood and used in relation to a broader context.

According to Dahlbom and Mathiassen, the intervention approach is a natural next step in a thesis-antithesis-synthesis process going from construction to evolution to intervention. Thus, in the intervention approach the designer is still basically an actor solving problems by means of approaches from the construction and evolution paradigms.

# 3. The Foss Electric work setting studied

"You see: Because you are working 'down here' in one corner of the code you only encapsulate yourself, and don't feel responsible for the overall structure. If the project don't have an extremely good coordination and a strong person to ensure that everything is coordinated, the functionality 'disappears between the modules'. No one remember it. That's the problem when we are as many software designers as in the S4000 project. It is problematic to consider too many designs and designers. That's why I say that two is fine, but four is a mess."
(Software designer at the S4000 project)

As mentioned in chapter 1, much of the discussions and reflections presented in this dissertation departure from a field study conducted at Foss Electric. This chapter provide a general introduction to Foss Electric and the S4000 project which has been specifically studied. The study mainly addressed a group of software designers working in the S4000 project. This chapter will provide a descriptive approach. The aim is only to introduce the field study before its results are discussed later.

A large scale manufacturing project is a very complex human activity involving a multitude of people with different areas of competence. A huge amount of decisions are to be made. The actors are mutually interdependent and they must coordinate their activities, allocate resources, schedule future activities, etc. The S4000 project was certainly no exception from this. The instrument to be developed was very complex, the project organization was composite, it included many actors with different perspectives, and it required a lot of coordination activities. During the project the actors invented and adopted several new mechanisms to support them in handling the coordination.

Because this was a study of design and test work, the field of work was in a number of ways different from, for example, work consisting mainly of monitoring and regulation activities. The work studied was neither time- nor safety-critical. It had a very important constructive, as opposed to analytical, element since the ultimate goal was to specify an instrument which can be manufactured within a broad range of constraints. Hence, the focus was on cooperative aspects of a design process carried out in a large scale setting, involving people with different areas of competence over a long time span. Design and implementation of the first version of S4000 lasted approximately 2 1/2 years and involved more than 50 actors. Apart from obtaining a general understanding of the work performed, the main objective of the field study was to identify and characterize artifacts aiming at reducing the complexity of the coordination work activities.

The analysis was conducted during the design of version 2 of the system. The field study and the preliminary data analysis lasted a total of six months and was exclusively based on qualitative data collection techniques such as qualitative interviews (Patton, 1980), observations, study of project documentation, and participation in project meetings. As mentioned in section 1.2.2 16 interviews were conducted, and I attended approximately 6 project meetings and a number of informal *ad hoc* meetings. Approximately 50 man-hours were spent observing the development process, and I have had 5 meetings with the some of the software designers later where we have discussed my analysis and ideas for how to use the findings. The approach was inspired by perspectives promoted in several other research efforts, for example Bucciarelli (1987). The analysis was furthermore organized according to the guidelines in the Work Analysis (Schmidt and Carstensen, 1990), and the conceptual framework provided here was used for modeling the work observed.

First the company is introduced. Then the S4000 project, especially the software group, is characterized. The chapter is concluded with a brief introduction to the aspects that made the S4000 project so complex and demanding. An introduction to Foss Electric and the S4000 project similar to the one given here can be found in Carstensen and Sørensen (1994a). Carstensen (1994), and Carstensen *et al.* (1995b) also contains brief introductions to the field study.

## 3.1 Foss Electric

Foss Electric is a Danish manufacturing company developing, producing, and marketing equipment for analytical measuring of quality parameters of agricultural products. Equipment for measuring quality parameters of agricultural products is a highly specialized field. The research, development, and production is localized in Denmark with subsidiary companies in England and Germany. Sales, service and distributors are spread all over the world. The Foss Electric corporation employs approximately 700 people.

The products manufactured are used for measuring the compositional quality of milk (the fat content, the count of protein, lactose, somatic cells, bacteria, etc.), the composition and micro biological quality of food products, and for measuring grain quality. The measurement technologies are typically infrared, fluorescence microscopy, or bacteriological testing. The customers are most often laboratories, slaughterhouses, dairies, etc.

There are only a few companies in the marketplace and Foss Electric is in their highly specialized field the largest in the world. Thus, when designing and producing new instruments, they mainly compete with themselves. Due to the degree of high market specialization, the few competitors on the market, and an increasing centralization of laboratories, the innovation towards new, better and faster measuring techniques is one of the most important strategic goals for the company. Research and development are essential activities.

Foss Electric has implemented concurrent engineering (cf. e.g., Helander and Nagamachi, 1992) yielding integration between manufacturing functions throughout the development process. Hence the organization is very much structured in terms of projects. These projects typically includes specialists with competence in fine mechanics, hardware and software design. In some projects also specialists in optics and chemistry are involved. The development from idea to final product involves a number of intermediate products: (1) A product concept defining the overall architecture and interaction between the involved technologies; (2) a few functional models (mock-ups); (3) five to ten prototypes of the instrument used for verifying detailed ideas and designs; and (4) a test series of five to ten instruments in order to test manufacturability of the product.

The field study concentrated on one of the large projects Foss Electric has recently accomplished, the System 4000 (S4000) project.

## 3.2 The S4000 project

The objective of the S4000 project was to build a new instrument for analytical testing of raw milk (See Figure 3-1). It was the first ‘system’ Foss Electric produced, i.e., an instrument in which several instruments are integrated and can be plugged in and out. Compared to previous instruments for testing milk, the S4000 system introduced measurement of new parameters in the milk (e.g. urea and critic acid) and the measurement speed was to be improved significantly compared to previous products.

**Figure 3-1:** The S4000 system being tested in the Quality Control department.

The S4000 was the first product with an Intel-based 486 PC build-in. The configuration, control, and operation of the instrument should be made via a Microsoft Windows user interface, i.e. a graphical user interface using mouse and keyboard. The software complex contained more than 200.000 lines of C-code. The software was organized in approximately 25 modules distributed in 15 different application running on the Microsoft Windows platform. The instrument consisted of approximately 8000 components grouped into a number of functional units, such as: Cabinet, pipette unit, conveyer, PC, other hardware, flow-system, and measurement unit. More than 50 different people were involved in the project, which lasted approximately 2 1/2 years (for version 1 of the S4000 system).

The core personnel, involved in the design included a number of designers from each of the areas of mechanical design, electronics design, software design, and chemistry. Added to this was a handful of draughtpersons and several persons from each of the departments of, production, the model shop, marketing, and top management.

### 3.2.1 The overall course of the software design for the S4000

The facilities to be provided by the S4000 instrument was originally considered to be a an aggregation of three to four existing instruments. It was therefore expected that the software to a large extent could be ported from these instruments. Furthermore, the complexity of designing the software required for the Windows interface was underestimated. Therefore, the software group originally included only three to four designers. This group started out by making an overall architecture design, and by specifying the required functionality in some detail. This was done using a preliminary requirement specification and specifications from previous instruments as a basis. During this work it was realized that the software design would require more effort than originally estimated. More designers were successively included in the group. After approximately a year, the software project manager was taken off the project, and an external consultant was requested to review the plans and to define a proper level of ambition for the software design. During this work, some of the ideas of working cycles, use of bug reports, etc. (discussed later) were invented. After the review, the plans were changed and some of the software facilities were postponed for version 2. The architecture design was revised and frozen, and the group started to work on the detailed design, implementation, and test. This, and the rest of the development work, was organized in short controllable cycles (see section 3.2.3). During the last 15-17 months the software design group had a stable size of approximately 7-8 designers, each having rather clear responsibilities related to the design and implementation of one or more specified modules. It is interesting to notice, that there was no software group manager during the one and a half years. It is, furthermore, important to notice that the new ‘structured’ way of organizing the work was new to the designers. From previous projects they were used to being mainly on their own, i.e., they could plan, organize and document their work as they found it most useful without obeying certain pre-specified rules or specification standards. Previously, the coordination required between the involved software designers was basically handled in an *ad hoc* manner without use of standardized procedures.

The development of version 2 of the S4000, which was the work I studied, lasted approximately 9 months and was organized in 5 working cycles (platform periods). Three to six software designers were involved in this project, each having quite clear responsibilities related to a number of modules. First phase in this project was a negotiation between the software design group and people from the marketing department on what to include in version 2. When this was done one of the software designers decided for each new facility (or correction) which software module it should be placed in. Since each of the designers were responsible for one or more modules it was then clear who was to correct what. Plans were then made for when this should be done. After this most of the work concerned detailed design, implementation and testing.

### 3.2.2 The S4000 software design group

During the S4000 project a group of between four and twelve software designers were working on designing, implementing, and testing the software. All the software designers had an educational background as software or electronics engineers, or as computer scientists. Most of them had at least five years experience in designing software for instruments. To handle the complexity of the software design work and its coordination the software designers organized the work in different ways which will be discussed later. Different roles were also defined, relating to the organization of the work. These were:

(1) Software designers.

As mentioned between four and twelve software designers were involved in S4000. All software designers were working as designers, i.e., they were responsible for designing, implementing, maintaining, and correcting bugs in one or more of the software modules. Hence, when a development or correction task was related to a specific module the designer 'automatically' became responsible for it.

(2) Spec-team.

The spec-team was a group of three software designers responsible for diagnosing reported bugs and deciding how to handle each of the bugs. The members of the spec-team in the S4000 project were appointed so that all the three main "layers" in the software were represented: One had deep knowledge on aspects regarding the user interface and the used file structures, etc. One was experienced in designing the algorithms used for computing the measuring results. And one was very experienced in developing software interfacing with the network, the hardware, and the external devices to be controlled by the software.

(3) Platform master.

The software design was organized in working periods called "platform periods" (described in details in the following section). At the end of each platform period, the Platform master was responsible for managing and coordinating all the activities involved in integrating the outcome of the working period. He was, among other things, responsible for verifying the corrections of the software made by the designers, i.e., control that the reported bugs had been dealt with. The platform master was always one of the designers in the project, and the role was taken alternately by the software designers involved in the project.

(4) Project plan manager.

In most of the project period the software group did not have a dedicated manager. Instead they appointed one among themselves as responsible for maintaining a project plan spreadsheet. This could be regarded as a rolling project plan for the software development. The sheet contained information on: (1) Which tasks there are to be accomplished and a reference to a de tailed description of the task; (2) the estimated amount of time per module for each task; (3) the responsibility relations between modules and software designers; (4) the relation between the tasks and which working period (platform period) they are planed to be finalized in; (5) and the total planned work hours per platform period for each software designer. The role and function of the project plan spreadsheet will be discussed further in chapter 6. The project plan manager was one of the three members of the spec-team.

(5) Testers.

Testers were the actors involved in the concrete testing of the software embedded in the S4000 instrument. The testers could be affiliated in several different departments at Foss Electric. They had thus a very different background and approach to what functionality the software should provide, and what the most important characteristics of the software were (e.g., usability, stability, correctness, maintainability, etc.). They were typically software, hardware, or mechanical designers involved in the project, or they were employed in the departments of quality assurance, marketing, service, maintenance, etc. Apart from the software designers approximately 15 'external' people were involved in testing the software. Some of these only implicitly, since they were involved in testing the S4000 in general.

(6) Central bugs file manager

At any given point in time of the S4000 project, one of the software designers was responsible for organizing and maintaining the central bug file, a ring binder containing copies of all reported bugs and organized according to their status. Ahead of each integration period the central file manager was responsible for informing the platform master on which bugs had been reported as corrected since the previous platform integration period. The role and function of the bug reports and the binder will be discussed in details in chapter 6.

### 3.2.3 The software design working cycles

During the project the software designers involved realized, that they had severe problems in coordinating and integrating their activities, and in integrating the software modules. They explicitly stated that they needed stipulations for the control and coordination of integration and meshing of the software in the S4000 project. The idea of ‘software platforms’ were invented by the designers themselves to support monitoring and controlling the integration of software pieces and modules.

The software platform is a concept including a number of artifacts, written procedures, conventions, etc. Originally a software platform was just a point in time where all software designers stopped all design activities and started integrating their bits and pieces. Later on, assisting artifacts and organizational procedures were added. The period between two software platforms—i.e., the period in which the software designers designed, coded, and tested their modules—was typically 3–6 weeks. Version 1 of the S4000 system covered approximately 15 platform periods. After a platform period, the developers spent a week integrating the software modules. This integration was managed by the platform master. During this period no designer was allowed the continue the design work until all had approved the integrated software complex. When the integration was brought as far as it was considered possible, and all known problems were written down by the platform master as tasks to be accomplished, the complete software complex was used (released) as platform for the departure of all new design activities. In the late part of the project—after having established a first running version of the total software system—the integration period was reduced to two and half days.

One of the things to be done in each integration period was to appoint one of the software designers as platform master for the next integration period. He was then responsible for collecting all information on changes (new development, redesign, error correction, etc.) made to the software. Together with the project plan manager, he was also responsible for updating the project plan spreadsheet.

## 3.3 Complexity of the S4000 project

From the characteristics given above it should be easy to see that the work conducted in the S4000 project was extremely complex. Handling the required tasks demanded the involvement of many actors with different perspectives. Consequently a coordination effort was needed (Mintzberg, 1983; Schmidt, 1994c).

To illustrate the complexity of the S4000 project work a modified version of Woods (1988) dimensions of complexity is applied including: Dynamism; Many highly interacting parts; Uncertainty; And multiple mutually interdependent actors. Woods do not distinguish between interacting parts and interdependent actors. Instead he includes risk as a specific complexity dimension. I have, however, found the dimensions listed above more relevant for the this chapter. Using these dimensions the complexity of the S4000 project work can be characterized as follows:

*Dynamism* is often caused by the fact that the work situations are characterized by handling a number of concepts, requirements, etc. that are dynamic by nature, i.e., events happen at indeterminate times. This might result in change of the problem to be solved. There was, for example, inherently dynamism in design and implementation of the S4000 software. There was a constant change in the use and design of the mechanical and electronics design. This implied changes to the requirements to the software and to the constraints under which it should be running.

*Many highly interacting parts*: The field of work is constituted of a large number of interconnected parts, components, concepts, etc. A software design failure could have many possible consequences, and conversely, a failure can have many possible causes and fixes. In the terms of Simon the problems were ill structured, and in such situations the actors use heuristics and general strategies in order to reduce the problem space (Simon, 1973). In the S4000 system there was, for example, heavy interaction and interdependence between the hardware and the approximately 15 different software applications, and between the 200.000 lines of code and the mechanical and chemical processes in the flow and measurement system. As one of the interviewed software designers stated:

> “The problem we have right now is that the software architecture is difficult to decompose so much that one designer can handle a component. We are all working

on several components and work on one component involves two to four guys, and perhaps even some of the electronics designers too."

*Uncertainty* exist in many complex work situations, i.e., the actors are often confronted with missing, incomplete, ambiguous, erroneous and contradictory information (Woods, 1988). The actors have to act on their cooperate judgment. In many work situations the problem it-self is not evident (Dery and Mock, 1985). Uncertainty is usually caused by external occurrences, or it can arise through failures, noise, time delays, influence of previous events, unclear requirement, etc. from the field of work. In the S4000 project, methods for measurement of completely new parameters in raw milk (e.g., urea and citric acid) had to be developed. This gave "unfamiliar" requirements to both the chemical, mechanical, and eletronics design of the S4000. And the software controlling the whole instrument, computing all the measurements, and presenting relevant information to the operator, was planned to be implemented as a set of Microsoft Windows applications, a platform on which the software designers had no previous experience.

Highly complex work situations are often handled in a distributed cooperative work arrangement with *mutually interdependent actors*, requiring a number of secondary activities handling the coordination. Cooperative work settings are often not stable and involve a large, varying or indeterminate number of participants (Schmidt, 1990; Schmidt, 1994c). The S4000 project was no exception. The software design was significantly more complex than in the usual projects. One of the software designers phrased it as:

"It has really been problematic that we did not have any guidelines and descriptions for how to produce and integrate our things. The individual designers are used to work on their own and have all the needed information in their heads, and to organize the work as they want to [..] When we started, we were only a few software designers. And suddenly — problems. And ups, we were several software designers and external consultants involved".

The description above clearly illustrates, that the work in the S4000 project was very complex in all aspects. It has earlier been mentioned that the sources of complexity in a work setting can be characterized by analytically distinguishing between: the field of work, i.e., a conceptual understanding of the work processes and objects, and their interrelations; the cooperative work arrangement, i.e., how work is organized; and the environment surrounding and constraining the work arrangement (Carstensen and Schmidt, 1993b). In the S4000 case all three sources can be seen as contributing to the complexity.

It must be concluded that because of the concurrent engineering strategy, different conflicting requirements is exposed and negotiated early in the manufacturing process. So, although this is an overall advantage in relation to the quality of the finished product, it leads to a heavy burden of coordination work in the design process.

Regarding software design and testing, the S4000 project was complex too. There was a strong need for coordination activities in order to handle the software design, implementation, and testing activities. This required the invention, adoption, and use of a number of artifacts, procedures, etc. to support the coordination. This will be explicitly addressed in the following chapters.

# Part II: Analyzing and modeling

The idea of supporting cooperative work by means of computer systems—the very idea!—can be compared with riding a tiger. Cooperative work may seem familiar and tame. And in fact, a plethora of languages and schemes has been furnished that confidently claim to provide reliable models of organizational roles and patterns of communication. The innocence and familiarity of cooperative work is deceptive, however. Cooperative work is difficult to bridle and coerce into a dependable model. And anyone trying to incorporate a model of a social world in a computer systems as an infrastructure for that world is as reckless as a daredevil mounting a Bengal tiger.
(Schmidt, 1991c)

# 4. Cooperative work and its coordination

This chapter discusses different perspectives for approaching and understanding aspects of coordination of cooperative work. First, I will introduce and discuss the inception for the need of cooperation: The complexity of the tasks and activities to be conducted. Then different approaches to coordination are introduced and discussed, and some examples from the field study at Foss Electric are provided. Finally a short introduction is provided of, how the term coordination has been interpreted and used in this dissertation.

## 4.1 Complexity: An occasion for cooperation

The need for cooperation arise from the limited capabilities of single human individuals (Schmidt, 1994c). The introduction of several actors and some division of labor is dictated by the job to be done and the technology available (Mintzberg, 1983). Of course, there might be other arguments for organizing the work cooperatively, e.g., improving social welfare, or having better control of the work. In this dissertation, a basic assumption is that the major source for establishing cooperative work settings is the complexity of the work. The complexity demands several actors to be involved.

This section discuss central characteristics of complexity in work. Many research areas and disciplines have addressed and discussed complexity characteristics from different perspectives. An introduction will be given based on some central approaches: Cognitive Engineering (Perrow, 1984; Woods, 1988; Woods and Roth, 1988; Rasmussen *et al.*, 1994), Human Problem Solving (Simon, 1973; Simon, 1983), and Organizational Theory (La Porte, 1975a; Mintzberg, 1983). This will illustrate the central dimensions characterizing the complexity of a work setting, i.e., pinpoint some of the important factors of the complexity that can be observed in most complex work settings.

Based on a cognitive engineering approach, Woods (1988) has argued, that complexity must be seen as something characterizing the domain (unfortunately without providing an exact definition of the term) and characterizing the problem solving approach applied. Hence, complexity is not a thing per se, but something that must be understood in a situation. In searching for dimensions relevant to understand complexity Woods identifies three factors, and their interaction, that contribute to the complexity of a problem solving situation: 1) The world to be acted on, 2) The actor who acts on the world, and 3) The external representation of the world. Each of these contain aspects that contribute in making problem solving complex. The involvement of multiple actors and technical systems—or 'Joint Cognitive Systems' as Woods calls it—increases the complexity, and so does the nature of the representations and conceptualizations of the world that the actors act upon. The most important complexity dimensions are, however, derived from the nature of the world. Here four complexity aspects—dynamism, many interacting parts, uncertainty, and risk—are central:

> "1. When a world is dynamic, problem-solving incidents unfold in time and are event-driven, that is, events can happen at indeterminate times. This element means that there can be time pressure, tasks can overlap, sustained performance is required, the nature of the problem to be solved can change, and monitoring requirements can be continuous or semi-continuous and change over time.

2. When a world is made up of a large number of highly interconnected parts, one failure can have multiple consequences (produce multiple disturbances); a disturbance could be due to multiple potential causes and can have multiple potential fixes; there can be multiple relevant goals which can compete with each other; there can be multiple on-going tasks having different time spans. In addition, the parts of the world can be complex objects in their own right.

3. When there is high uncertainty, available data can be ambiguous, incomplete, erroneous, low signal to noise ration, or imprecise with respect to the state of the world; the inferential value of data can vary with context; future states and events are not completely predictable. Uncertainty can be due to external occurrences, noise, changes in noise parameters over time, nonlinearities, time delays or the influence of previous events and inaccurate measurements can arise through sensor failures, miscalibrations or mesenteries.

4. When there is risk, possible outcomes of choices can have large costs. The presence of risk means that one must be concerned with rare but catastrophic situations as well as with more frequent but less costly situations. When uncertainty is coupled with risk, situations of choice under uncertainty and risk arise." (Woods, 1988, p. 130).

Representations and conceptualizations of complex work situations often involve a number of conceptual items containing a very rich semantics, i.e., conceptual elements with a large number of possible interpretations. Concepts to which a simple and immediate interpretation are rare. The concepts cannot be systemized into structures and the interpretations are usually overlapping. The work situation is engrossed in a conceptual world characterized by a rich and varied semantics and the situation requires application of different conceptualizations of the domain, i.e., the decision making process requires employment of and transformation between different implementations (Rasmussen, 1985).

In his famous study of accidents in high-risk work settings, Perrow identifies two essential dimensions that can be used for characterizing the potential risk of a "system" (work arrangement) (Perrow, 1984): The dimensions of interactiveness and coupling between the component in the system. The interactiveness spans from linear in one end to complex in the other, and the degree of coupling ranges from loose to tight. Linear interactions is carried out through a series of sequences or steps laid out in a line, i.e., they are expected and predictable, the feedback loops are familiar, it is easy to locate a failure, and even if unplanned interactions occur they are easily visible to the actors. In these situations the number of interacting parts is not an important factor. But if the components serve multiple functions the interactions become non-linear and complex, i.e., there are many unfamiliar feedback loops, it is difficult to isolate a failed component, only a limited understanding of the ongoing processes exists, and unplanned, unexpected, hidden and invisible interactions occur frequently. In these situations the number of components and their connections, which are often unanticipated, become an important factor. Coupling ranges from loose to tight. Loosely coupled systems can have many connections between the components, but they are characterized by the existence of many alternative and flexible methods, the possibility of change in the sequences, and by having very little consumer monitoring. Tight coupled systems are characterized by not having alternative strategies or methods, being very sensitive to slack in the processes, and being highly time-dependent. Tight coupled systems cannot incorporate shocks, failures and pressure without destablization.

When approaching the actor oriented (problem solving) aspects of a complex work situation, the situation can be seen as a process searching for a solution in a given space, a process of reasoning or accumulating information until an answer has been found, or as a process of constraint satisfaction narrowing down the set of solutions until it satisfies all the constraints (Simon, 1983). The situations involve a huge amount of potentially relevant components, i.e., in practice it is impossible to investigate and test all possible solutions and the problem space is too large to systematically search entirely. The actor is confronted with incomplete, ambiguous, erroneous and contradictory information on the situation (Woods, 1988). Thus a complete constraint satisfaction process is impossible to describe beforehand, i.e., the problem is in reality ill structured (Simon, 1973; Simon, 1983). The actors have to act on their judgment by use of heuristics, strategies, etc., and they are usually not able to be aware of the potentialities in their entirety.

From the point of view of organizational theory the complexity of organized social systems can be measured in terms of the number of system components, the relative differentiation of the components, and the interdependence among the components (La Porte, 1975b). A component is defined as a person or group occupying a position within the system. The relative differentiation is defined as the number of different social roles or positions within the system, and the interdependence reflects the degree of reciprocal relationships between the components. When thinking of work settings where the actors have a perceived relatedness—i.e., the actors recognize their connections to other actors—the complexity first of all depends on the degree of interdependence: "When we remember that the basic element in complex social systems is the exchange interaction among people, we see that such a multitude of relationships can become the source of considerable distress for them. Here lies the root of the limitations to the complexity to which social forms are subject" (Ibid., p. 13).

Many work situations must handle decisions or problems that are new or contain aspects the actors have not been confronted with before. Or the problem is not evident: "problems are not objective entities in their own right, but are the product of interpretation. [...] problems do not come with an identifying tag, neither as problems nor as certain types of problems" (Dery and Mock, 1985, p. 107). The complete set of actions relevant to the organizational world is unknowable. The set of possible states or alternatives for achieving a goal is unknowable. Unforeseen situations are a common occurrence (Barber, 1983). Both these aspects affects what we could call the organizational complexity: First, no organizational procedure for handling the problem can be established beforehand (Mintzberg *et al.*, 1976). Actions and decisions will be based on incomplete, ambiguous, erroneous, or even contradictory sets of rules and procedures. The actors have to make their own interpretations. Second, decomposition of the system of organized complexity is complicated or impossible (cf. e.g., Parnas, 1985). According to La Porte, the latter is essential for decreasing the complexity.

To summarize, the causes of the complexity factors of a work situation are:

1) the nature of the organization of the work, e.g., the interdependence among the actors and the interactional complexity (cf. Perrow's approach),
2) the context or environment in which the work is conducted, e.g., the uncertainty of what customers are demanding,
3) the nature of the work itself, e.g., in much design work the number of possible solutions to a problem are indefinite, and
4) the fact that the cognitive capabilities of the involved actors are delimited.

It could be argued that complexity is a characteristic that constitutes a work situation or a phenomenon. Thus the complexity cannot be reduced. However, supporting and (or) redesigning the interdependence and interaction between the involved actors can address a reduction of the involved components and their interdependences, and a reduction of the interactional complexity. This will then reduce the complexity of the implementation (organizational nature) of the work setting. The complexity derived from the nature of the work itself, or from the nature of the environment, might be hidden by choosing an appropriate structure for presenting the (complex) content.

## 4.2 The emergence of 'coordination'

To cope with complexity the involvement of more than one actor is required. The complexity of many work situations require capabilities of the actors that exceeds the capabilities of individual actors. A work setting based upon cooperation must be established. It is impossible to specify and plan all aspects of such a setting beforehand:

> "An arc for any given trajectory—or project—consists of the totality of tasks arrayed both sequentially and simultaneously along the course of the trajectory or project. At least some of the arc is planned for, designed, foreseen; but almost inevitably there are unexpected contingencies which alter the tasks, the clusters of tasks, and much of the overall organization. Hence the arc cannot be known in all its details—except in very standard, contingency-minimal projects—until and if the actors look back and review the entire course which they have traversed" (Strauss, 1985, p. 4).

Schmidt (1990) has identified a set of generic functions, provided by the establishment of cooperative ensembles:

- Augmentation of capacity. The capacity of the individual actors are insufficient.
- Differentiation and combination of specialities.
- Mutual critical assessment. By this the ensembles try to ensure a more balanced decision.
- Confrontation and combination of perspectives, in order to get a more multifaceted and coherent understanding of the problem complex before taking the needed decisions.

The actors become mutually interdependent in their work. They need to coordinate their activities, i.e. mesh, allocate, relate, schedule, etc. activities, actors, and resources with respect to each other. A central characteristic of a cooperative work setting is the underlying and constitutive mutually interdependence of the involved actors. Hereby a set of secondary activities coordinating the distributed activities is required.

The mutual interdependency among the actors has important implications for our understanding and definition of a cooperative work setting. A cooperative work arrangement is not necessarily sited in one organization sharing resources, etc. A cooperative work arrangement is constituted by the field of work, i.e. constituted by “the part of the world that is being transformed or otherwise controlled by the cooperative work arrangement” (Schmidt, 1994c, p. 15). Actors in a cooperative work arrangement handle tasks and activities that are closely related, intertwined, and interdependent. At some levels these are characterized by having a common objective with respect to the purchaser of the services provided. A cooperative work arrangement is often to be understood across organizational boundaries. Since the field of work and the work arrangement mutually constitutes each other, we should attempt to define them through iterations in a dialectic process.

When few actors are involved, or the required coordination activities are characterized by a low complexity, everyday social life modes of interaction are sufficient for handling the coordination (Ibid.). Several studies indicate that actors are good at handling the coordination on an ad-hoc basis in these situations (e.g., Harper *et al.*, 1989b; Heath *et al.*, 1993). Problems emerge when the actors become distributed in time and space, when many different areas of competence are involved, when the activities are intertwined, or when work is carried out over a long time-span. In these situations the actors need conceptualizations of actors, activities, resources, obligations, etc. from the field of work and the work setting in order to handle the coordination. For example plans, work procedures, and classification schemes. The concepts often need to be redefined and negotiated, and symbolic artifacts are introduced in order to reduce the complexity of the coordination, e.g., forms, work schedules, classification structures, etc.

The aim of this dissertation is to discuss possible computer support of coordination. In “order to be able to conceptualize and specify the support requirements of cooperative work we need to make a fundamental analytical distinction between (a) cooperative work activities in relation to the state of the field of work and mediated by changes to the state of the field of work and (b) activities that arise from the fact that the work requires and involves multiple agents whose individual activities need to be coordinated, scheduled, meshed, integrated etc. — in short: *articulated*. This distinction is fundamental" (Schmidt, 1994c, p. 18). The following sections will discuss some relevant approaches to coordination work (or 'articulation work')[1].

## 4.3 Approaches to coordination work

This section will first introduce three approaches to 'articulation work': Strauss who originally described the term (Strauss, 1985), Gerson and Leigh Star who have related it to discussions on computer support (Gerson and Star, 1986), and Schmidt who has elaborated further on the approach (Schmidt, 1994c). After this I will give a brief introduction to Malone and Crowston's coordination theory (Malone and Crowston, 1990) and Mintzberg's understanding of coordination (Mintzberg, 1983).

### 4.3.1 Strauss' approach

Strauss (1985) establishes a conceptualization of the division of labor "in terms of close scrutiny of *work* itself" (p. 1). The aim of the conceptualization is to support research of the nature of work. He describes how projects involve a course of action which entails a division of labor, both in terms of actors and activities. Strauss calls it 'actions' which are made up by many tasks done over time and divided up among many actors or group of actors. He argues that:

> "Since the plurality of tasks making up their totality, as well as the relations of actors to tasks, are not automatically articulated, actors must do that too, and often in complex ways. We call the work of doing this 'articulation work'—a supra-type of work in any division of labor" (Ibid., p. 2).

In contrast to most research on division of labor—which has been concerned with issues of the division of work by various occupations and professions or issues relating of differential distribution of rewards to

1 This chapter does not distinguish between articulation work and coordination work. I will call it 'coordination work', but in each of the descriptions I will use the terminology used by the inventors.

classes—Strauss addresses the tasks necessary for doing the work in the division of labor: “Distribution is where the emphasis has been, and ‘labor’ in both senses of the word—(wo)manpower and work—has meant largely the former” (Ibid., p. 2). Hence, a central aspect of articulation work is concerned with the division of labor.

According to Strauss the division workers (persons, classes, or organizational units) are depended on the tasks to be done. This implies (Ibid., p. 5) that: (1) the division are changing over time during a the project course, (2) the division of labor varies by the type of work; “it is the variation in work, not merely the class of worker, that is the essential ingredient for getting a task accomplished”, (3) understanding the division of labor requires a detailed scrutiny of how tasks are clustered and related to each other. “If either the tasks or the speciality sharing were to be problematic, then there would be a question of who shall do them”, and (4) “none of this arc of work is called into play automatically.” Some actor has to be responsible for deciding, planning, and articulating the various tasks. The relevant aspects of articulation work and the division of work are closely related to the characteristics of the field of work. Strauss considers the “type of work” as the central division parameter. Understanding the tasks distinctively becomes essential, including aspects like:

> “what, where, when, how, for how long, how complex, how well defined are their boundaries, how attainable are they under current working conditions, how precisely are they defined in their operational details, and what is the expected level of performance. (which of those are the most salient dimensions depends on the organizational work context under study, and we cannot emphasize too much that it is the researcher who must discover these saliences.) Two other important questions are: how they are put together in task clusters, and linked together in an organization of tasks. ‘Work’ which constitutes the total arc, or some portion of it, is the ‘decomposed’ (Gerson, 1983), even perhaps in some arcs down to detailed mini-tasks.” (Strauss, 1985, p. 6).

The allocation of tasks is distributed among the actors in a number of ways: they can be imposed, requested, assumed without request or command, delegated, or proffered. Furthermore, they can, of course, be accepted or rejected. But an allocation is never fixed. Revision goes on continuously, usually as negotiation. ‘Articulation’ then is a central part of the work in the division of labor to be done. Having this in mind, Strauss refines the characterization of articulation work:

> "[it amounts to] First, the meshing of the often numerous tasks, clusters of tasks, and segments of the total arc[2]. Second, the meshing of efforts of various unit-workers (individuals, departments, etc.). Third, the meshing of actors with their various types of work and implicated tasks. (The term 'coordination' is sometimes used to catch features of this articulation work, but the term has other connotations so it will not be used here.) All of this articulation work goes on within and usually among organizational units and sub-units.
>
> All workers articulate something (in accord with their positions on the accountability ladder); whether tasks, task clusters, smaller or larger segments of the arc. Understandably, articulation work will vary with various properties of tasks, task clusters and arc segments and phases. [...]
>
> Paradoxically, articulation tasks themselves also require a higher degree of work, with the highest levels of authority—assigned, requested, claimed, imposed, etc.—doing the highest order of integrating. Like other tasks, articulation ones are carried out both simultaneously and sequentially for different portions of the arc by different workers [...] At any higher level there would be some allocation of articulation tasks involving what, who, how where, when, etc. And so each worker is accountable at least upward, while needing to articulate some tasks—and usually some actors' efforts—downward" (Ibid., pp. 8-9)

Having an approach where the division of labor is the focal point could perhaps have led to a conceptualization mainly covering aspects of the division of work addressing topics of power and division according to classes, professions, etc. Strauss has, however, taken a multifaceted understanding of the complexity of articulation work (the work done in the division of labor) when conceptualizing articulation work. Based on his previous work, and analysis of articulation work and the articulation process[3], he outlines a theoretical scheme or model for studying the articulation of project work, that:

> "revolves around: (1) *work process* (discovering and maintaining appropriate resources; devising and maintaining a division of labor; matching tasks and workers' motivations; supervising delegated tasks); (2) *types of work*; (3) *interactional processes* (including negotiating, persuading, educating, manipulating, and coercing); (4) all these elements occurring at every organizational level; (5) and interactionally requiring continual *alignment*; (6) although the specifics of the articulation process

2 Strauss defines an arc as consisting of the totallity of tasks arrayed along the course of a project.

3 Strauss distinguish between articualtion work and the articualtion process. The articulation process regards putting all work elements together and keep them together. According to Strauss this represents a more inclusive set of actions than the acts of articualtion work. For the purpose here I don't find this distinction particularly relevant.

vary according to the properties of projects (including whether they are more or less routinized and more or less complex). (7) In addition, *unanticipated contingencies* inevitably affect the functioning and articulation of these routines” (Strauss, 1988, p. 175).

### 4.3.2 The approach of Gerson and Leigh Star

Gerson and Leigh Star base their approach to articulation work on analyses of the situated nature of office work. Based on studies of office work they state that, “even apparently simple pieces of information such as entries on fixed forms are the result of many negotiations and struggles” and concludes that “since no centralized authority can possible anticipate all the contingencies that might arise locally, office workers always have some discretion in deciding how reconciliation is to be accomplished” (Gerson and Star, 1986, pp. 257-258). The distributed character of the decision making activities and the situated character of the decision situations studied are hence central characteristics of the approach Gerson and Star establish.

Their approach is inspired by Hewitt’s understanding of offices as “open systems” and the “due process” in such systems. Open systems are characterized by concurrency among its components, asynchrony due to the asynchrony of the outside world and the physical distribution of the components, decentralized control in order to avoid bottlenecks, inconsistent information because many internal components and/or external sources are acting, partly isolated components that are not aware of all states of other components (“arm’s-length relationships”), and continuous operation independently of individual components (Hewitt, 1985). The due process then

“is the organizational activity of humans and computers for generating sound, relevant, and reliable information as a basis for decision and action within the constraints of allowable resources. It provides an arena in which beliefs and proposals can be gathered, analyzed and debated. [...] Due process is inherently reflective in that beliefs, goals, plans, requests, etc. exists as objects that can be explicitly mentioned and manipulated in the ongoing process. Due process does not make decisions or take action per se. Instead it is the process that informs the decision-making process” (Ibid., p. 275).

In the office work studied by Gerson and Star the due process problem is how to assure that information systems make adequate provision for recognizing, weighing, and evaluating alternatives from conflicting sources. They call the ensuring during the work process articulation and define that:

> "Articulation consists of all the tasks needed to coordinate a particular task, including scheduling subtasks, recovering from errors, and assembling resources" (Gerson and Star, 1986, p. 258).

Later refined into:

> "Reconciling incommensurate assumptions and procedures in the absence of enforceable standards is the essence of *articulation*. Articulation consists of all the tasks involved in assembling, scheduling, monitoring, and coordinating all of the steps necessary to complete a production task. This means carrying through a course of action despite local contingencies, unanticipated glitches, incommensurable opinions and beliefs, or inadequate knowledge of local circumstances" (Ibid., p. 266).

Based on Hewitt, Gerson and Leigh Star consider all systems as open systems. Thus, we cannot foresee the contingency that might arise, and every system requires articulation. They identify the outcome of articulation work as:

> "Standardized representations of office work and its products, as captured in forms diagrams, databases, or narrative text are thus the result of articulation, the local adjustments that made the work possible in practice." (Ibid., p. 258).

Gerson and Star argue, furthermore, that office work can be described in an idealized form without concentrating on the articulation aspects. But to understand the situated character of situations, we must consider the articulation of concrete situations carefully. Hence, if we wants to build office information systems supporting the problems of the due process, i.e., supporting the articulation of the work, we need to address the due process explicitly, and articulation tasks cannot just be integrated in the work flow.

Where Strauss based his approach to articulation work on the division of labor and the activities handling the division, Gerson and Star base their approach on the fact that all representations of a work situation are incomplete. The work situations are situated in the work context. Office work is distributed and has some discretion in the decision process. The central purpose of articulation work is to overcome the situated character of the work, i.e., handle the due process problem. Although Gerson and Leigh Star take their departure in aspects involving several actors, articulation work can, in their approach, be something done by an individual actor in order to handle a complex situation in which s/he is the only actor involved.

Leigh Star has later refined the approach further and established a concept of objects containing the multiple viewpoints required when several actors having different background and perspective need to articulate their work; the Boundary Objects (Star, 1989). The idea is to identify "objects that are both plastic and coherent through a collective course of action" (Ibid., p. 38). The concept is then used for establishing requirements for distributed artificial intelligence systems. This idea is interesting since we look for an approach that can be used for discussing computer support of articulation work. The examples given do, however, indicate that all types of objects, artifacts, physical places, etc. can be a candidate for a boundary object. The concept is thus to indeterminate to be useful.

### 4.3.3 Schmidt's approach

Although taking a sociological departure Schmidt's approach to articulation work has a more precise purpose than the above mentioned approaches. It aims at establishing a conceptual framework for understanding articulation work to assist design of computer-based systems supporting articulation work. Schmidt and associates[4] have discussed and refined the approach in several publications (Schmidt, 1990; Schmidt, 1991c; Schmidt and Bannon, 1992; Schmidt *et al.*, 1993; Schmidt, 1994c).

The essential characteristics of a cooperative work arrangement—and thereby of the articulation work conducted—is first of all the recognition of the distributed nature of the arrangement, the interdependent actors and activities, and the constantly ongoing reallocations, adaptions to change in requirements, etc.:

> Cooperative work relations emerge in response to the requirements and constraints of the transformation process and the social environment on one hand and the limitations of the technical and human resources available on the other. Accordingly, cooperative work arrangements adapt dynamically to the requirements of the work domain and the characteristics and capabilities of the technical and human resources at hand. Different requirements and constraints and different technical and human resources engender different cooperative work arrangements (Schmidt, 1994c, p. 11).

4 I am one of the assoiciates, and the work reported in this dissertation is one of the important contributions to this work.

Articulation work is approached as a series of on-going activities handling the articulation of the cooperative work in a world of unforeseen contingencies. To establish an understanding of articulation work addressing this contingency we have worked with a very broad definition covering aspects like scheduling and allocation of resources, monitoring, handing over, resolving inconsistencies, reconciling incommensurate assumptions, opinions, and beliefs, and so forth, cf. e.g., our discussion in Schmidt *et al.* (1993).

| *Objects of articulation work* | *Elemental operations with respect to objects of articulation work (examples)* |
|---|---|
| Actors | enroll A, assign B, reserve C; move D, place E |
| Responsibilities | allocate, assume, volunteer; hand over; accept, reject; |
| Tasks | point out, express; divide, relate; allocate, assume, volunteer; accept, reject; order, countermand; accomplish, assess; approve, disapprove; |
| Activities | do; make publicly perceptible, monitor, be aware of; explain, question; |
| Conceptual structures | define, classify, instantiate, relate, exemplify; posit, accept, challenge; |
| Informational resources | copy to A, move from B, transfer; access, block; read, interpret, relate; |
| Material resources | consume, move from A, place near B; name, characterize; procure, deploy, reserve; |
| Technical resources | use, move, place; name, characterize; procure, deploy, reserve; |
| Infrastructural resources | name, characterize; reserve; |

**Figure 4-1:** Our first list of the essential objects of articulation and examples of operations on these, from Schmidt *et al.* (1993), p. 127. The list have been expanded and objects of articulation are now structured into actual and nominal structures (cf. chapter 5).

To cope with the contingency a conceptual understanding of the essential dimensions along which the work is articulated in order to address questions on what, where how, when, who, etc. (cf., Strauss, 1985) is required. Schmidt and associates have attempted to establish a first set of essential dimensions which we then, when discussing computational mechanisms supporting articulation work, developed further and defined as “objects of articulation” and operations on these, cf. figure 4-1 above.

According to Schmidt—and Strauss—a central aspect of articulation work is handling these objects of articulation work. But, as Schmidt states articulation work is always done in a context that needs to be understood and approached too. The essential aspects of the context, when conceptualizing articulation work, are the state of affairs in the field of work, the constraints posed by the environment in which the work is conducted, and the nature of the organizational setting in which the work is conducted.

Although Schmidt recognizes the fact that much articulation work is contingent and has a situated character, he further argues that in most work settings a lot of routine work and work that can be accomplished by individuals exists too:

> “Action to accomplish some goal is not *always* “enormously contingent”! Of course, any action may be construed as enormously contingent in Herakleitos’ sense that every situation is unique. But the action is not necessarily “enormously contingent” to the actors themselves. Contingencies may be more or less complex to deal with, more or less serious in terms of effect, scope etc., more or less frequent, and so forth, and different contingencies may affect the outcome of action and the validity of plans differently. [...] The more distributed the activities of the cooperative work arrangement, the more complex the articulation of the activities of that arrangement” (Schmidt, 1994c, p. 21).

With respect to designing computer systems supporting articulation work Schmidt recognizes that articulation cannot be mediated only via the field of work itself. We need to address the different modes of interactions actors conducting articulation use, and the means for the interaction used. Based on a discussion of different modes and means of interaction, and central characteristics of articulation work analyzed we established a concept of Mechanisms of Interaction supporting articulation work by mediating and stipulating it (Schmidt *et al.*, 1993; Schmidt, 1994c; Schmidt and Simone, 1995). An introduction is given in chapter 5.

### 4.3.4 Coordination Theory

Malone and Crowston (1990) have recognized the need for a set of concepts supporting design of computer-based coordination support. They argue for establishing a "coordination theory" based on inputs from different disciplines. They define coordination theory as "a body of principles about how activities can be coordinated, that is, about how actors can work together harmoniously" and continues:

> "In coordination theory, the common problems have to do with coordination: How can overall goals be subdivided into actions? How can actions be assigned to groups or to individual actors? How can resources be allocated among different actors? How can information be shared among different actors to help achieve the overall goals?" (Ibid., p. 358).

They identify four main dimensions of components of coordination (goals, activities, actors, and interdependencies) and processes associated with these. The processes have to do with identifying goals, mapping goals to activities, selecting actors and assign them to activities, and managing the goal-relevant relationships between activities. Malone and Crowston argue that we need a narrow definition of coordination that explicitly focus on the elements that are unique to coordination. They suggest coordination defined as: "the act of managing interdependencies between activities performed to achieve a goal" (Ibid., 361). Interdependencies can be analyzed in terms of what they call "common objects" involved in the actions that needs to be coordinated. These common objects will thus constrain how the activities can be performed. A first set of relevant kinds of interdependencies include tasks dependent on the product of other tasks, use of shared resources, simultaneity, and domain specific interdependencies. It is suggested that a first use of technology could be just to detect the interdependencies. The second conceptualization provided relates to the level of processes underlying coordination: coordination, decision making, communication, and object perception.

As said by Malone and Crowston themselves, it is a very preliminary sketch of a framework. Up till now it seems to lack structures describing interdependencies between actors. "Common objects" are only used to analyze interdependencies between activities. Furthermore, the definition of coordination is very narrow and excludes several of the aspects of coordination addressed by the previously mentioned approaches.

### 4.3.5 Coordination according to Mintzberg

Mintzberg addresses frameworks for understanding and designing organizations rather than explicitly discuss coordination (Mintzberg, 1979; Mintzberg, 1983). He argues that analyzing or designing organizations can be done by means of combinations of five types of organization structures: Simple Structure, Machine Bureaucracy, Professional Bureaucracy, Divisionalized Form, and Adhocracy. The second essential dimension characterizing organizations is the “coordination mechanisms”. According to Mintzberg, organizations coordinate their work by means of combinations of five coordination mechanisms: “mutual adjustment, direct supervision, standardization of work processes, standardization of work outputs, and standardization of worker skills. These should be considered the most basic element of structure, the glue that holds the organizations together” (Mintzberg, 1983, p. 4). There is a one-to-one relationship between the organization structure to be chosen and the prime coordination mechanism to be used when designing the organization:

- The Simple Structure is based on centralization, and the “strategic apex” of the organization is the most important part of the organization. For this structure the direct supervision should be used as the coordination mechanism. Coordination is achieved by having one person responsible for work of others.
- Machine Bureaucracy is based on job specialization and formalized behavior. The most important part of the organization is the infra- and technostructure. The prime coordination mechanism is standardization of work processes characterized by procedural description of how to conduct the work.
- The Professional Bureaucracy is based on decentralization. The “operating core” is the most important part of the organization. The prime coordination mechanism is standardization of the workers skills and knowledge. Skills and knowledge are standardized by specified training.
- The Divisionalized Form is based on grouping of the organization reflecting the markets. The “middle line” becomes the most impor-

tant part of the organization and standardization of output is the essential coordination mechanism. This requires specification of requirements for the output.

- The Adhocracy based on decentralization, job specialization, and an organic structure. The most important parts of the organization are the “support staff” and the “operating core”. Mutual adjustment based on informal operator to operator communication is the prime coordination mechanism.

As an interesting point Mintzberg observes that the use of the coordination mechanism seems to follow a specific sequence over time. When the work becomes more complex the “favored means of coordination shifts from mutual adjustment to direct supervision to standardization, preferably of work processes, otherwise of outputs, or else of skills, finally reverting back to mutual adjustment” (Mintzberg, 1983, p. 7).

Mintzberg’s categories for coordination work is interesting since it can be used for an overall characterization of an organization or a work arrangement. But in relation to the purpose of this dissertation—to discuss computer support of coordination work—the framework seems to be too general.

### 4.3.6 Summarizing the approaches

All the approaches seem to agree that coordination work often covers a set of very complicated tasks to accomplish. What is perhaps more important is that coordination work can be considered a set of ‘isolated tasks’ that has to be accomplished in order to handle the fact that several actors are involved, i.e., we can *analytically* distinguish between work and coordination work. Strauss, Gerson and Star, and Schmidt *et al.* all make this distinction explicitly. By suggesting a theory for coordination Malone and Crowston implicitly indicates the possibility for analytically distinguishing between coordination and what is to be coordinated. The same goes for Mintzberg when he explicitly characterizes the type of coordination mechanisms used in a specific type of organization.

Furthermore, the coordination of cooperative activities should be conceived of as an inherently recursive phenomenon (Gerson and Star, 1986). The coordination of the distributed activities of a cooperative work arrangement with respect to its field of work may itself be handled cooperatively, i.e., a cooperative work arrangement can take the organization of another cooperative work arrangement as its field of work. We have elsewhere argued that it is the recursiveness that makes the open-endedness of cooperative work arrangements manageable (Schmidt *et al.*, 1995).

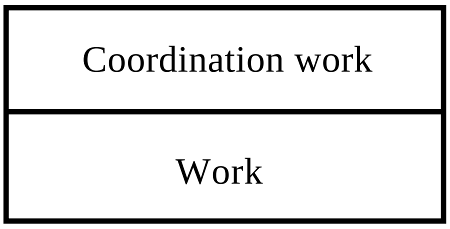


**Figure 4-2:** The approaches discussed in this chapter all  seemsto imply that we can analytically distinguish the tangible work, and the required activities regarding meshing, allocation, naming, relating, scheduling, etc. of activities, actors, resources, etc. We can analytically separate work and coordination work. Note that the model must be understood recursively. Hence, a cooperative work arrangement can take the organization of another cooperative work arrangement as its field of work. This distinction is a basic assumption for the approach presented in this dissertation.

Strauss work departs from discussions on division of labor, i.e., specialization and decomposition are central elements. He also recognizes the need for understanding coordination in terms of scheduling activities and monitoring the processes, etc. Gerson and Star consider the situated character of much work as essential. This then lead to a definition of articulation work which will include the activities on classifying, standardizing, etc. done by an individual actor. The idea of boundary object is interesting, but some refinement needs to be done if the concept is to be used for conceptualizing the central aspects of coordination work.

Not surprisingly I will argue that the approach by Schmidt seems to be a useful approach in relation to work towards computer support of coordination. Based on the ideas of Strauss and Gerson and Star, the approach attempts to identify some essential conceptualizations of the coordination work. Furthermore Schmidt recognizes the distributed and situated character of cooperative work with constantly changes in requirements, allocations, etc. This dualism—on one hand recognizing the contingency and situated character of articulation work, and on the other also trying to establish some conceptualizations of the stabile structures of work—

appears to be a useful approach. Although it addresses computer support Malone and Crowston's theory appears to be a bit to narrow and incomplete, whereas the conceptual framework of Mintzberg is too general to provide substantial input.

## 4.4 Findings from the Foss Electric field study

Chapter 3 gave an overview of the work setting studied at Foss Electric, and chapter 6 will provide a detailed analysis of some of the findings from the field study. For the sake of illustration a few central findings from the field study will be introduced here.

First of all: The designers at Foss spend a lot of effort on coordinating activities related to the software development in the S4000 project. To support this there were at least four types of countermeasures reducing the complexity of coordination work: (1) A project oriented matrix organization was implemented. Projects were organizational units with the project manager serving as a "head of department" and all participants were physically located in the same area; (2) To ensure that the overall goals were met, the project had a whole line of scheduled meetings, some weekly and most twice per month. In the most intense phases of the S4000 project 27 different meetings, involving from 6 to 26 participants, were scheduled; (3) Besides the scheduled meetings a lot of unplanned meetings were held as well. Typically one or two participants who recognized a problem met; (4) The amount of detailed information that needed to be communicated, coordinated, negotiated, etc., required more formalized measures for the daily operation (these will be discussed in chapter 6).

Furthermore, I found that when the number of mutually interdependent actors involved in the software development exceeded the limit of a few, they needed to examine the state of affairs in the field of work. When, for example, the software was designed by more than two to three actors, the designers spent quite some effort on being aware of the redesigns introduced by the other actors. This was impossible to do by means of *ad hoc* modes of interaction only. Managing the work required many intertwined and interdependent activities to be handled, and the complexity of meshing the activities increased tremendously. The actors recognized a need for supportive structures, procedures, forms, etc. New roles taking care of certain coordination activities were needed, and support for structuring and controlling the stipulation of the interrelated activities were required.

The first solution when a coordination problem was recognized was to increase the use of *ad hoc* coordination, i.e., have more formal and informal communication and meetings. In many cases this appeared to be insufficient and ineffective. In these situations different types of mechanisms were invented and used to keep track of the integration or the state of affairs, to schedule relations and dependencies among involved actors, tasks, and resources, etc. These mechanisms were, for example, forms or boards and related conventions and procedures prescribing how they should be used. Some of the mechanisms were invented in the project, some were a result of redesign of previous mechanisms, and others were adopted. Examples of these were: An augmented bill of materials (ABOM) supporting the integration between mechanical design, process planning, and production; The fixed software design working cycle (or working rhythm) including forms and procedures supporting the coordination and integration of the software; Or the bug report form coordinating the activities concerning registration, diagnosis, and correction of software bugs. These and several other examples have been identified and described elsewhere (cf. e.g., Carstensen *et al.*, 1995b). The purpose and function of the bug report form and its relations to the other mechanisms will be illustrated in detail in chapter 6.

The field study indicated that when confronted with an abundance of detailed decisions and activities needing to be coordinated, organizations invent and adopt mechanisms that (partly) mediate and stipulate the coordination of the work. These can be purely *ad hoc* based (e.g., meetings), or they can be more formal and provide standardized structures, classifications, conceptualizations, etc. reflecting relevant structures the field of work and the work arrangement (e.g., project plans with tasks, actors, and deadlines interrelated).

## 4.5 My approach to coordination

In this dissertation the term ‘coordination’ is used in a very broad sense—broader than the usual connotations. Coordination activities cover aspects like scheduling, meshing, and allocating resources, negotiation of resource allocations, monitoring work activities, resolving inconsistencies, etc. This is similar to what Strauss (1985) calls ‘articulation work’. Furthermore, activities concerning the establishment of means supporting coordination activities are considered ‘coordination’, e.g., the refinement of a classification scheme. When certain activities are regarded ‘coordination’ this is always done in relation to a specific field of work. For all fields of work, on which more than one actor works, we can identify a set of activities that are ‘second order activities’, ‘extra activities’, or ‘overhead activities’ with respect to what the work itself requires. These activities arise because more than one actor is required to do the work. These activities are also considered ‘coordination’.

Let us take a short example from the S4000 project: A central activity was to test and correct the software. This activity involved many actors testing and correcting the software. It required planning, distribution of information, meshing of activities, etc. Activities like planning the test activities, scheduling the correction tasks, negotiating the allocations, deadlines, etc., distributing correction requests to the relevant designers, and the development of a bug form and concomitant procedures for its use, are all considered ‘coordination’ in relation to the software testing work. Of course, scheduling the correction tasks can itself be considered a field of work. Then other activities, e.g., negotiating prioritizing of the tasks, would be coordination with respect to scheduling the correction tasks. This is an example of the recursiveness of the “work-coordination work” distinction (cf. section 4.3).

The suggested approach to coordination has two important characteristics: First, it is based on the existence of a given (analytically identifiable) field of work and a related (analytically identifiable) cooperative work arrangement working on the field of work. The field of work and cooperative work arrangement mutually constitute each other. Second, it is based on an analytical distinction between the tangible work and the coordination work related hitherto. Both the “field of work”-”work arrangement” distinction and the “work”-”coordination work” distinction must be regarded as analytical distinctions. In real work settings structures, phenomena, and activities are intertwined.

In discussions with research colleges, I have been met by the viewpoint that the approach is too naive, a simplification, just a simple distinction between value adding work and none value adding work, and the like.

Aspects like the social structures in a work setting, the psychosocial work environment, the sociocultural aspects of work, power structures in the work setting, or a detailed discussion on what value adding work actually covers are not addressed in the approach. The purpose of the approach is, however, to establish a pragmatic and useful approach that can be used to explicitly address important aspects when analyzing cooperative work settings, and when designing computer-based systems to support cooperative work settings. By explicitly addressing coordination we can achieve two things: As analysts we can get a better and more coherent understanding of the phenomena and artifacts we study. And as designers we can ensure provision of the required support for coordination work. Hence, the approach should not be regarded as an alternative to other approaches to systems design, rather it is an enhancement.

# 5. The Concept of Coordination Mechanisms

"A particularly important new service which coordination systems supply to environment builders is a uniform and powerful means to establish the expected relationships to task-interdependence - in other words, the desired patterns of task coordination. To be useful, this must be done in a flexible yet well-integrated manner, with plenty of leeway for the unpredictability of real life. The new capabilities at which coordination technology aims depend on finding and installing appropriate conceptual and structural units with which to express tasks, their diverse relations to each other and to the people who ultimately bear responsibility for them" (Holt, 1985, p. 281).

As argued several places in the previous chapters, the coordination of cooperative work can—when many actors are involved, when the work is distributed in time and space, or when different competencies or perspectives are involved—be very complex. Then everyday social and communication skills are insufficient for managing the coordination. In such work settings it becomes relevant to require different kinds of artifacts. These might be computer-based. They can provide support by prescribing how the work could be conducted, constrain the solution space, mediate relevant information, offer relevant representations of the field of work, stipulate the flow of the work, etc.

This chapter introduces the conceptual framework of Coordination Mechanisms, i.e., a protocol that, by encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, stipulates and mediates the coordination of distributed activities so as to reduce the complexity of coordinating distributed activities of cooperative work settings (cf. e.g., Schmidt *et al.*, 1993; Schmidt, 1994b; Carstensen *et al.*, 1995b; Schmidt and Simone, 1995; Schmidt *et al.*, 1995). The concept has several purposes: It can be used as an analytical concept for understanding how procedures, forms, and other artifacts can be considered mechanisms supporting coordination, and it can be used to support design of computer-based coordination mechanisms.

First, I will provide some background for the establishment of the conceptual framework. Then the conceptual framework is described in some details, and some overall requirements (or facilities) of coordination mechanisms are discussed. An illustration of essential characteristics of the coordination mechanisms identified in the field study is provided. The chapter is concluded by a discussion of an important feature of coordination mechanisms: These are often linked to each other—they interoperate. For example, one mechanism might subscribe to information from another, or trigger a specific action from another.

## 5.1 Background

As mentioned in section 1.3, I have, in collaboration Kjeld Schmidt, established a conceptual framework supporting work analysis of complex work (cf. e.g., Schmidt, 1988; Schmidt and Carstensen, 1990; Carstensen and Schmidt, 1993b; Carstensen and Schmidt, 1993a). The aim has been to provide a framework that has a more coherent and functional oriented approach to understanding a work system than the traditional procedural oriented approaches, i.e., understand a system in terms of the services a system provides to its purchasers. The approach was inspired by several sources: Simon (Simon, 1981; Simon, 1983), Cognitive Engineering (e.g., Rasmussen, 1986; Woods, 1988; Woods and Roth, 1988), Soft Systems methodology (Checkland, 1981), Hammer's early work on understanding the function of office work (Hammer and Sirbu, 1980; Hammer, 1984), the critiques of the AI-tradition (e.g., Roth and Woods, 1989); etc. A number of overall requirements for a framework was established. Regarding requirements for a conceptual framework Schmidt and I have elsewhere argued that:

> "First, the work arrangement must be seen as a dynamic whole (system). No parts (procedures, activities, components, etc.) or perspectives can be addressed and analyzed in isolation. Relevant features must be identified and related to each other. To do this, at least three different approaches are required, addressing 1) characteristics of the outer environment and the relations between the environment and the cooperative work arrangement, 2) the field of work, and 3) the characteristics of the actual cooperative work arrangement itself (i.e. the existing implementation of a socio-technical system).

> To conceptualize the work arrangement as a whole and to address its relations to the outer environment, support for identifying the (abstract) functions offered by the work arrangement to the environment is needed. [...]
>
> Second, the essential characteristics of the field of work must be identified and described. [...]
>
> Third, the cooperative work arrangement itself must be unraveled for several reasons. The types of activities must be identified, e.g. as a list of prototypical tasks and activities. Central questions are: To which extent do the activities reflect characteristics in the field of work or in the environment and to which extents are they caused by the current implementation of the work arrangement? Which activities are difficult and why? When deciding the functional division between humans and computers in the future, this is extremely important.
>
> Also, unraveling the cooperative aspect of the work arrangement is essential. An analytic distinction to overcome the confusion of the concepts of cooperative work must be supported, i.e. distinction of concepts like division of labour, organization, allocation of tasks and responsibilities, profession, collaborative styles, labour market structures, etc.” (Carstensen and Schmidt, 1993b, pp. 578-579).

Hence, the need for an analytical distinction between field of work and the work arrangement was recognized in relation to understanding the interaction between a work system and its environment, or ‘between the inner and outer environment’ as Simon would phrase it (Simon, 1981).

A natural next step was to search for support for understanding the activities and other work aspects going on inside the work arrangement, especially the cooperative aspects of a work setting was to be addressed explicit. From the distinction between work arrangement and field of work, it is easily recognized that actors in a work arrangement are working within the same field of work. Thus, the actors are controlling, handling, transforming, etc. a number of interdependent objects, structures, processes, etc. To do this they have to interact. They have to coordinate their activities. This is done by means of an open-ended set of interactional activities used in mixed ways. According to Schmidt (1994c; 1994b) the purpose of these interactional activities can be to, for example:

- maintain reciprocal awareness among the actors,
- deliberately directing the attention of an actor to certain aspects of the field of work,
- assigning tasks by directing an actors attention to a specific detail or by an explicit request, or

- handing over responsibility for a certain process, task, etc.

These interactional activities can have many different appearances, for example seeing, hearing, marking items, humming, pointing, nodding, doing activities in abnormal ways, or writing notes. It is difficult to structure these different modes by means of certain concepts. Schmidt has abstracted these into three salient overall dimensions of “modes of interaction” (Schmidt, 1994c):

- Obtrusive versus non-obtrusive interactional activities. Some modes are disruptive. Others are conspicuous and do not require much further attention from the actors.
- Embedded in the field of work versus a symbolic representation embedded in an artifact representing the state of the field of work. The symbolic representation provides a higher degree of freedom in manipulating the intimations presented.
- Ephemeral versus persistent interactional activities. Some modes of interaction will disappear without leaving tracks. Others will leave a path to trace.

A fourth prominent characteristic of the modes of interaction is the allocation of the functionality between the actor and the artifact used. The degrees of freedom—or ‘local control’—of the coordination can be understood as a continuum from *ad hoc* coordination in the one end to coordination-based on rigidly prescribed protocols in the other. Figure 5-1 illustrates the continuum.

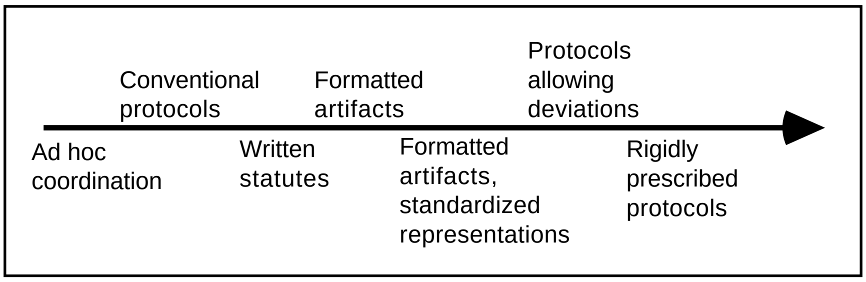

Conventional protocols
Formatted artifacts
Protocols allowing deviations
Ad hoc coordination
Written statutes
Formatted artifacts, standardized representations
Rigidly prescribed protocols

**Figure 5-1:** The continuum of modes of interaction. In one are *ad hoc* based modes that do not involve any pre-specified stipulations. In the middle we have modes including artifacts that somewhat constrains the coordination. In the other end are modes based on formally specified stipulations. The arrow indicates that the formalization increases from left to right.

The nature of the interactional activities and relevant modes of interaction will not be discussed further here. They are, however, important to keep in mind when setting up a concept for understanding and designing mechanisms supporting coordination work.

The interactional activities discussed above can be seen as a category of activities required to manage the fact that the cooperative work is distributed over several actors, in time, and (perhaps) in space. Coordination work (or articulation work, cf. previous chapter) is required. It becomes highly relevant to make an analytical distinction between work and coordination work. When we relate this distinction to the distinction between work arrangement and field of work it appears that what is coordination work for one work arrangement might be the field of work for another work arrangement. The work - coordination work distinction must thus be understood as recursive, as also argued by Gerson and Star (1986).

The aim of the work presented in this dissertation is, as mentioned earlier, to discuss how the use of computer technology can support coordination work. This can be done by, for example, providing more flexible interaction mechanisms, providing better representations of the field of work that reduces the complexity of the coordination work to be conducted by the actors[5], or providing facilities that delimits the need for coordination.

Inspired by the advanced systemic approach used in the development of the Work Analysis and lessons learned from field studies, empirical studies of existing CSCW-systems, and existing theories we have established a conceptual framework of Coordination Mechanisms, i.e., mechanisms that support the coordination work by mediating coordination information and by stipulating the flow of distributed activities. The framework will be described in section 5.3. Before this let us take a brief overview of what has been conducted in order to establish the framework.

## 5.2 Activities providing input for the framework

The establishment of a framework like the one discussed is, of course, not done ‘over a night.’ Several different activities have been conducted in order to establish the required basis. A lot of literature on themes like work,

5 A better representation might not—literally—cut off any work, but it can make it easier to handle and thereby reduce the effort required by the actors.

cooperative work, studies of work, coordination, complexity, human interaction, use of artifacts, modeling in system development, existing CSCW systems, etc. has been studied. Much of this were discussed previously in chapter 1, 2 and 4. Approaches within the field of CSCW similar to the one provided here are discussed in section 5.6.

Our first version of the concept was described in Schmidt *et al.* (1993). It was based on input from several sources:

1) A number of existing field studies were re-analyzed with respect to coordination aspects (cf. Schmidt, 1994c);
2) A number of existing CSCW systems were analyzed as coordination mechanisms, i.e., they were considered "a standardized symbolic artifact which was publicly available and publicly perceptible, and having a standard format which imposed constraints on the users of it and it could be manipulated independently of the field of work" (cf. Andersen *et al.*, 1993, p. 217 and; Simone *et al.*, 1993);
3) A critique of existing methodologies with the office automation and computer science traditions (Carstensen and Schmidt, 1993a); and
4) An attempt to establish a formal notation to use for specifying computer-based coordination mechanisms.

The next step was to use the framework when conducting work analyses of complex work settings. Mainly three different domains were studied: Writing of technical documentation (Andersen, 1994); Software testing (Carstensen, 1994); and Mechanical design (Sørensen, 1994a; Sørensen, 1994b). Several analyses were conducted illustrating how different artifacts were used as means for coordination (cf. e.g., Borstrøm *et al.*, 1995; Carstensen *et al.*, 1995b). This work was, together with further refinements of the formal notation (Simone *et al.*, 1994), the main input for refining and reconsidering the conceptual framework (Schmidt, 1994b; Schmidt and Simone, 1995; Schmidt *et al.*, 1995), and discussions on how 'real life' coordination mechanisms are linked (Schmidt *et al.*, 1994). Apart from this my field study was also used to discuss overall requirements for computer-based mechanisms supporting the coordination of software testing (Carstensen *et al.*, 1994; Carstensen *et al.*, 1995c), and methodological discussions of the use of a conceptual framework for analyzing coordination aspects of cooperative work (Carstensen, 1995b).

Although the process described above seems to be quite straight forward the truth is, of course, that all these activities have been heavily intertwined and parts of an iterative process.

## 5.3 Coordination mechanisms

As argued earlier, work settings having a high degree of complexity of the coordination work or coordination of the distributed activities, require a certain mode of interaction. This modes often include mechanisms embodied in an artifact containing a protocol that stipulate and mediate the coordination work. This mechanisms can thereby be regarded as reducing the complexity of the coordination activities seen from the actors perspective[6]. Such mechanisms are what I call 'coordination mechanisms'[7]. A short definition:

> *A coordination mechanism is a protocol, encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, that stipulates and mediates the coordination of distributed activities.*

To fulfill this purpose of reducing the complexity of the coordination work to be conducted by the actors, we have identified certain specific characteristics of coordination mechanisms (cf. Schmidt *et al.*, 1993; Schmidt, 1994b; Schmidt and Simone, 1995; Schmidt *et al.*, 1995). These are:

(1) A coordination mechanism is essentially a *protocol*. It is a set of explicit procedures and conventions that *stipulate* the coordination of the distributed activities. The distributed activities are coordinated by executing the protocol. In some situations the actors might, of course, chose not to execute the protocol. Then the co-

6 Although the mechanism reduce the demands on the actors with respect to coordination activities a coordination mechanism can, of course, be seen as a constraint to, or formalization of, the actual work.

7 During the work of developing the conceptual framework, these mechanisms were also termed "Mechanisms of Interaction". The term 'interaction' has, however, too many and wide connotations to be adequate. The same might go for 'coordination', but that is at least well-defined in the context of this dissertation.

ordination mechanism is neutralized, and then later ‘switched back’ to a well specified state.

(2) The stipulations of the protocol are (partly) conveyed by a *symbolic artifact.* This means they are persistent in the sense that they are, in principle, accessible independently of the particular moment, of the state of affairs in the field of work, or of the particular actor. They are publicly available in some kind of ‘physical form’, i.e., a cognitive symbolic structure only existing in ‘the head of the actors’ is not considered a coordination mechanism (but it might, of course, be turned into one).

(3) The symbolic artifact *mediates* the coordination of the distributed activities. The artifact is an intermediary between the actors. It conveys any change to the state of execution of the protocol to other actors by means of changes to the state of the artifact. The representation embedded in the artifact must thus be available, accessible, and manipulatable for the actors.

(4) The symbolic artifact has a *standardized format* that reflects pertinent features of the protocol. It thus provides affordances to and impose constraints on coordination work. By providing a formal structure the artifact “force a specific behavior on a person” (Norman, 1991, p. 34) that makes it easier for others to interpret the information mediated.

(5) The state of the protocol is *distinct* from the state of the field of work. Changes to the state of the field of work are not automatically reflected in changes to the state of the execution of the protocol and vice versa. This allow the actors to reconfigure the behavior of the protocol (locally or permanently) without discarding ongoing work.

Since a coordination mechanism is defined as a set of procedures and conventions supported by an artifact with certain necessary characteristics, computational (or computer-based) coordination mechanisms are conceived of as a special category of coordination mechanisms characterized by a specific allocation of functionality between the human actors and the artifact. A computational coordination mechanism is defined as a computer artifact that incorporates aspects of the protocol of a coordination mechanism so that changes to the state of the mechanism induced by one actor can be automatically conveyed by the artifact to other actors in an appropriate form as stipulated by the protocol (Schmidt, 1994a). The Concept of Coordination Mechanisms can thus be used to characterize mechanisms having a wide range of allocations of functionality between actor and artifact, from almost total reliance on human intervention to almost fully automated computer artifacts. All coordination mechanisms are fundamentally 'social', i.e., they are constituted within a set of procedures and conventions.

The very idea of having pre-defined protocols or procedures as determinants of action has been analyzed and discussed widely in the literature (e.g., Suchman and Wynn, 1984; Suchman, 1987; Rouncefield *et al.*, 1994). Many of these studies indicate that formal organizational constructions in the actual course of work is problematic. They are 'idealizations' when taken as representations of actually unfolding activities. But—as argued by Schmidt (1994b)—the studies have often focused on recovery from error in an administrative agency. They provide only little insight into how standard procedures, defined as pre-defined written stipulations, are applied in routine daily work. So, protocols can convey "affordances and constraints to the individual actor which the actor, as a competent member of the particular ensemble, can apply without further contemplation and deliberation unless he or she, again as a competent member, have accountable reasons not to" (Ibid., p. 94). The actors might have conclusive reasons for deviating from the stipulations. Thus, rather than avoiding protocols, they should be flexible and performative constructs (Bowers, 1993) that can be overruled if necessary.

When addressing artifacts we must consider two aspect with respect to what they represents: The 'surface representation' and the 'internal representation' as Norman calls them (Norman, 1991). Artifact with a surface representation are primarily "systems for making possible the display and maintenance of symbols: They implement the 'physical' part of the physical symbol system" (Ibid., p. 25). Artifacts containing internal representations have symbols maintained internally within the device, unlike for example paper and pencil where the symbols appear only and always on the surface (Ibid.). When describing coordination mechanisms both the surface and the internal representations need to be addressed. The surface representation is usually immediately visible, whereas the internal representations can be ascribed to a protocol that is not incorporated in the artifact, for example forms or check lists used.

An aspect of the definition of coordination mechanisms was, that the state of the protocol is distinct from the state of the field of work. Several field studies describe artifacts supporting coordination of work from which they are separated, for example the flight strip in air traffic control (Harper *et al.*, 1989a) or the bug form used at Foss Electric (described in chapter 6, see also Carstensen, 1994). Other field studies have illustrated how mechanisms directly coupled to the state of the field of work are overruled and used in unintended ways in order to enable reconfiguration of the stipulations without discarding the ongoing work (e.g., Schmidt, 1994b). The conceptual framework should thus be based on an explicit distinction between the protocol and the artifact on the one side and the state of affairs in the field of work on the other.

## 5.4 Towards a first conceptualization of coordination work

In order to establish a basis for designing computer-based mechanisms supporting coordination work, a conceptualization of relevant dimensions in the work is required. As argued by Strauss coordination work is conducted with respect to the salient dimensions of who, what, where, when, how, etc. (Strauss, 1985). Strauss' findings, Schmidt's re-analysis of field studies (Schmidt, 1994c), and findings from the field study as Foss Electric (Carstensen, 1994) all indicated, that coordination work is conducted with respect to conceptualizations (objects) pertaining structures in the actual field of work and the actual work arrangement. An example is the coordination of the software testing activities at Foss Electric (discussed in chapter 6). Here my analysis implied, that when coordinating the activities the actors related to conceptualizations of the software architecture and classifications of modules and bugs (i.e., structures reflecting the field of work), and conceptualizations of work procedures, plans, human resources, and technical resources (i.e., structures from the work arrangement).

Inspired by Strauss 'salient dimensions', the attempt to identify 'basic coordination processes' by Malone and Crowston (1990), the re-analyses conducted by Schmidt, and findings from my field study a first model of the relevant dimensions of objects of coordination work[8] was established. The dimensions were both related to the cooperative work arrangement and to the field of work, and they were furthermore organized according to their status, nominal or actual. The model can be seen in figurte 5-2 including data from my field study at Foss Electric. The dimensions have, of course, evolved, and we have presented the model in several versions during the work on the framework (Schmidt *et al.*, 1993; Carstensen, 1994; Schmidt, 1994b; Schmidt and Simone, 1995).

Regarding the cooperative work arrangement, the following dimensions of objects of coordination work must be included:

- *Roles* indicating general responsibilities for classes of tasks, resources, etc.
- *Actors* participating in the cooperative work arrangement. These can be committed to specific task and roles, participating in ongoing activities, have different roles, cover different capacities, etc.
- *Human resources* which is the potential participants of the cooperative arrangement.
- *Tasks* are operational intentions, characterized by goals to attain, obligations, commitments to meet, etc. These structures include information on the problem, conditions for undertaken the task, etc. Furthermore relations to other structures like actors, informational resources, or material resources is required.
- *Activities* are ongoing unfolding courses of action. Also these structures include relations to other structures like actors or material resources.

Regarding the field of work a list of dimensions of objects of coordination work must include the following:

- *Information resources.* This is documents, letters, notes, files, memos, drawings, etc. which the actors access, refer to, etc. These must also include relation structures to, for example, material resources, actors, or roles (this goes for the following resource dimensions as well).

8 These dimensions have we elsewhere called “Objects of articulation” (e.g., Schmidt *et al.*, 1993).

- *Material resources* being structures containing information on characteristics, components, available assemblies, etc.
- *Technical resources* are structures having information on for example tools, machinery, or software applications.
- *Infrastructural resources* contain descriptions of rooms, communication facilities, transportation facilities, etc. including their status and operational characteristics.
- *Conceptual structures* are the relationships between categories used within a specific community as ordering devices with respect to the field of work. As recognized by Schmidt categories used in coordination work can either be adopted (e.g., used to establish prototypical categories), or they can be applied for classifying events, objects, etc. (Schmidt, 1994b). The former is used to establish a common understanding of aspect of the field of work, whereas the latter is done as to support monitoring of, direct attention to, etc. certain aspects of the state of the field of work.

How the understanding of these dimensions has evolved will not be discussed further here. The refinement can be illustrated by studying the sections on ‘Objects of articulation work’ in for example Schmidt *et al.* (1993), Carstensen (1994), and Schmidt (1994b).

Apart from the “work arrangement”-”field of work” distinction, the analysis of the data from the Foss Electric field study indicated a need for a distinction in order to conceive the complicated dynamics of coordination work.

| Nominal | | Actual | |
|---|---|---|---|
| **Dimensions of coordination work** | **Operations with respect to the dimension** | **Dimensions of coordination work** | **Operations with respect to the dimension** |
| *Dimensions with respect to the cooperative work arrangement* | | | |
| **Role**<br>- Tester<br>- Responsible designer<br>- Spec-team member<br>- Central file manager<br>- Platform master | assign to [Committed actor];<br>responsible for [Task] | **Committed actor** | assume, accept, reject [Role];<br>initiate [Activity]; |
| **Task**<br>- Correction task | relate; allocate;<br>accept; volunteer;<br>reject; accomplish;<br>approve;<br>disapprove; | **Activity** | initiate [Committed actor];<br>done by [Actor-in-action];<br>realize [Task];<br>make publicly; |
| **Human resource**<br>- James, Jones ...... | locate, allocate, reserve; | **Actor-in-action** | initiates [Activity];<br>does [Activity]; |
| *Dimensions with respect to the field of work* | | | |
| **Conceptual structures**<br>(conceptualizations of the field of work)<br>- Bug classifications<br>- Software modulation<br>- Platform periods | categorize; define;<br>relate; | **State of field of work** | classify aspects of;<br>instantiate;<br>direct attention to;<br>make sense of;<br>act on; |
| **Informational resources** | locate; obtain access to; reserve; | **Informational resources-in-use** | show; hide; |
| **Material resources** | | | |
| **Technical resources**<br>- Test machine | categorize; locate;<br>reserve [Task]; | **Technical resources-in-use** | categorize; |
| **Infrastructural resources** | | **Infrastructural resources-in-use** | |

**Figure 5-2:** A model of the essential dimensions of objects of coordination work. Related to these dimensions (column 1 and 3) are examples of typical elemental operations on these (column 2 and 4). Adapted from Carstensen (1994). For illustration purposes are included examples from the coordination of software testing at Foss Electric.

During the coordination work the objects of coordination were often related to one of three stages. They were either *nominal* (potential, but not yet effected), *actual* (ongoing), or *past* (completed). The past-dimension is simple since it is just a track record of the actual-dimension. In figure 5-2 the nominal and actual dimensions of objects of coordination work are incorporated, and typical operations on the objects are listed.

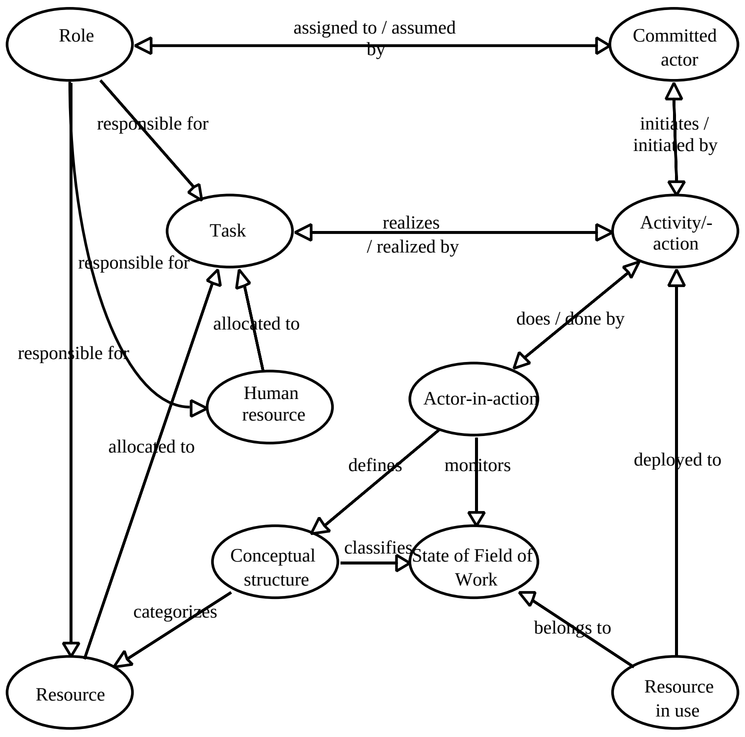


**Figure 5-3:** Dimensions of objects and elemental operations of coordination work. The objects on the left hand side of the diagram are of *nominal* status, whereas the objects on the right hand side are of *actual* status. The 'missing links' between objects can be constructed indirectly, by creating composite operations. Adapted from Schmidt (1994b, p. 28)

The dimensions of objects of coordination work should be considered interrelated, e.g., conceptual structures used for categorizing resources, roles are related to tasks by means of responsibilities, actors are assigned to roles, tasks are related to activities, etc. They have relationships with each other, and together they form a dynamic structure. Conducting one of the elementary operations exemplified will influence the state of the objects in the dimensions. The relationships between the objects of coordination work, and the influence of the basic operations are illustrated in figure 5-3.

The conceptualizations presented in this section have mainly been seen as an attempt to capture the essential aspects of a work setting in a coordination work perspective. The concept of objects of coordination work and the model of these should, of course, also be seen as preliminary tools supporting the process of analyzing cooperative work, i.e., the work analyst could regard the dimensions of objects of coordination work as a framework or checklist to use for classifying his findings.

## 5.5 Overall facilities of coordination mechanisms

As argued earlier a driving force of this dissertation work has been to discuss how use of computer technology can support coordination work. It is thus relevant to draw attention to which facilities existing coordination mechanisms provide to the users (actors). Future mechanisms to be designed should offer similar or improved facilities, no matter whether they are to be based on computer technology or not.

Based on the study of existing CSCW applications, literature on coordination work and CSCW, a re-analysis of existing field studies, etc. we established a first set of required facilities for access and manipulation of computer-based coordination mechanisms in Schmidt *et al.* (1993). Based on this set and the findings in the field study the following overall requirements for coordination mechanisms can be established:

(1) A coordination mechanism must be malleable and controllable by the cooperating work arrangement itself. Similar to plans coordination mechanism are “resources for situated action” (Suchman, 1987). We cannot foresee and pre-program all situations. The actors must thus be in charge of changing the protocol embedded in the mechanism, and of controlling the mechanisms while they are running.

(2) To support the requirements of malleability and controllability the protocol and status of a coordination mechanism must be visible and accessible to the actors, and amenable to modifications by workers while in the process of work.

(3) Since coordination work is intertwined with work activities directed to the field of work coordination mechanisms must, to some extent be integrated in the machinery, support tools, processes, etc. used in the work. Computer-based coordination mechanisms should not be seen as isolated applications, but rather as facilities to be included in the applications already used for conducting the work.

(4) The primitives, structures, concepts, etc. used for expressing the status of the field of work, and used when the actors access the mechanisms must be at a semantic level appropriate to the work context. The primitives, structures, concepts, etc. should be expressed in terms of basic concepts pertaining to the coordination of distributed activities in cooperative work, not in terms of low-level programming primitives or abstractions of the work differing much from the one normally used.

(5) Since the coordination mechanisms are supposed to support cooperative management, they must support multiple actors in modifying the mechanism while being engrossed in the flow of distributed activities. The mechanisms must then have support for propagating information on changes and for managing the inconsistencies that might appear due to changes in the middle of a running process.

To specify this further the following facilities must be provided for:

- Making changes locally to the protocol prescribing the execution flow and to the artifact mediating coordination information. These changes must be temporary, and apply for specific situations only. An example could be that an actor suspend the protocol for a given instance of a work flow, overrule a step in the protocol, or re-start a given instance of a procedure. It is the involved actors themselves that must have access to this facility. There might, of course be situations where only some actors should be allowed to make certain changes to the protocol.
- Making global and permanent changes to the protocol prescribing the execution flow and to the artifact mediating coordination information, for example re-designing the flow of work or adding new

roles in relation to the mechanism. Also this kind of changes must be accessible for the actors involved.

- 'Re-programming' the mechanism. If the actors are going to make permanent changes to the protocol or to the content and function of the artifact(s) used they must be provided with a 'language' for (re-)specifying the protocols and the behavior of the artifact. To make the specification job as easy as possible the components and manipulations provided by the mechanism must correspond to notions and objects of everyday coordination work. A high degree of transparency is required.
- Controlling the propagation of information on changes. A mechanism must provide means for making dynamic reconfiguration cooperatively. Thus means for propagating information of changes should be offered too, i.e., facilities for informing other actors of the changes, and for bringing the mechanism back into a well-specified and consistent state.
- Getting access to the structure of the mechanism. From the requirements above follows that the behavior of the mechanism must be 'visible' to the actors. In order for actors to be able to exercise their control of the execution of the mechanism and re-specify the behavior of the mechanism, the specification of the behavior of the mechanism must be accessible and manipulable to actors and, more specifically, accessible and manipulable at the semantic level of coordination work (Schmidt, 1994b).
- Only partially specify the behavior of a coordination mechanism. A coordination mechanism must include explicit prescribed procedures, but since we cannot foresee the situations the procedures should be under-specified (Suchman and Wynn, 1984; Suchman, 1987). Therefore it must be possible to let certain attributes be left un-specified until a given action is taken by the actors or by another mechanism.
- Keep track of the changes a mechanism has been through. The actors must have access to reconstruct the evolution of the mechanism in order to access and retrieve earlier versions that are possibly more adequate to the current situation or suitable starting points for the redesign of the mechanism.

- Identifying pertinent features of the field of work as represented by the data structures and the functionality of the system in which it is embedded. Coordination mechanisms are local and temporary closures (Gerson and Star, 1986). A coordination mechanism will thus only address certain aspects of the work to be coordinated. Accordingly, a coordination mechanism is to be “conceived of as a specialized [..] device that is distinct from the state of the field of work and yet embedded in an application so as to support the [coordination] of the distributed activities of multiple actors with respect to the field of work as represented by that application” (Schmidt, 1994b, p. 111). This is, of course, especially relevant for computer-based mechanisms embedded in another software application.
- Specify structures in the coordination mechanisms reflecting the structures of the organizational context in which the distributed activities to be coordinated are conducted. Although work mainly is coordinated with respect to conceptualizations of the field of work and the work arrangement, structures in the wider organizational context might also be relevant. For example overall company policies might influence the coordination work.
- Linking coordination mechanisms to each other. The field study clearly illustrated that although the coordination mechanisms used could be considered individual demarcated mechanisms, they have some very important interrelations (Schmidt *et al.*, 1994; Carstensen *et al.*, 1995b). This is discussed in details in section 5.8.

It should be noticed, that the facilities are not distinct categories of facilities. Their respective contributions to the support for access and manipulation are extremely intertwined. So, the facilities must be seen as a tentative frame of reference.

## 5.6 Related approaches

Others have been working with similar ideas of providing computer-based flexible support of coordination work based on a conceptual understanding of coordination. Those considered relevant were discussed in chapter 2. I will very briefly discuss some of them in the light of the required facilities discussed above.

OVAL developed by Malone and associates (Fry *et al.*, 1992; Malone *et al.*, 1992) is intended to be a general framework, including a notation, for expressing coordination work. The basic primitives of objects, views, agents, and links are very general. Consequently, the notation in OVAL becomes very flexible. The specification level is however very primitive, and the basic primitives provided by the notation do not correspond to objects in general coordination work. They rather reflect typical structures in a computer.

The Coordinator (Winograd, 1986; Flores *et al.*, 1988) can be conceived of as providing a certain specification language for the coordination-related interactions between the actors. The Coordinator is an example of a system that incorporates a coordination mechanism. Facilities are provided for specifying and re-specifying protocols. But The Coordinator provides no visibility and control of the mechanism to actors when it is running. Another problem is that The Coordinator uses ‘conversation for action’ as a metaphor for coordination work. Hence, the basic dimensions of objects of coordination work are defined in terms of ‘obligations’.

Another environment for providing support of collaborative activities is the EGRET Framework (Johnson, 1992). EGRET aims to support what is termed ‘exploratory group work’, for example software engineering (Johnson and Tjahjono, 1993). The EGRET Framework supports specific aspects of coordination by giving the users the ability to develop new ‘schemes’ which can include tasks, descriptions, names etc. Only domains characterized by having specific structures (e.g., consensus structures) are supported, i.e., only a few highly specialized domains are supported. The primitives are at an appropriate semantic level, but they are specialized in terms of schemes representing the current state of consensus. Local and temporary changes to the structure, and visualization of these, are supported by the EGRET Framework, i.e., a high degree of control of execution and visibility of changes are supported.

ConversationBuilder (Kaplan *et al.*, 1992b) was developed as a support tool for providing flexible active support for (collaborative) work activities. The flexibility is achieved by providing ‘appropriate mechanisms’ for the support of collaboration rather than specific policies. Visibility and control of the mechanisms are offered. ConversationBuilder offers a flexible environment covering a number of relevant facilities and it offers a set of primitives at an appropriate semantic level. But the system is not based on an understanding of some of the fundamental characteristics of coordination work. The notation for specifying mechanisms is based on a theory of conversation. As for Coordinator, obligations become the central structure, and control is what is mainly supported.

Regatta (Swenson *et al.*, 1994) provides support for planning work processes. This is done by (iteratively) modeling the communication that is required to coordinate the relevant tasks. The system does provide visibility, malleability, and control of the mechanisms developed, but it seems to be based on the basic assumption that an overall 'meta-plan' can be established, so that all other plans become sub-plans to this. Hence linking is not supported. The semantic level of the basic primitives seems to be useful, for example actors and tasks can be specified at a proper level, but several important objects of coordination work are not addressed, e.g., responsibilities, conceptual structures, resources, etc.

## 5.7 Characteristics of 'real life' coordination mechanisms

In order to illustrate the form and function of coordination mechanisms a few examples of the (mainly paper-based) coordination mechanisms identified at Foss Electric will be given here. The only purpose is exemplification, thus some important details might be missing. The mechanisms will be described in greater detail in chapter 6.

In several situations the software designers in the S4000 project realized that *ad hoc* coordination was insufficient and ineffective for managing the coordination of the distributed activities of software development, integration, and testing. Different types of paper-based mechanisms were invented (or adapted) and used to keep track of the integration or the state of affairs, to schedule relations and dependencies among involved actors, tasks, and resources, etc.

The first example is a concept of a working cycles (or working rhythm) called "software platforms". It was invented by the software designers themselves in order to support monitoring and controlling the integration of software pieces and modules. The software platform is a concept including a number of artifacts, written procedures, conventions, etc. Originally a software platform was just a point in time at which all software designers stopped all design activities and started integrating their bits and pieces. Later supportive artifacts and organizational procedures were added, and a platform period was defined at the period used for integration. The period between two software platforms—i.e., the period in which the software designers designed, coded, and tested their modules—was typically 3–6 weeks. After a platform period, the developers spent a week integrating the software modules. When the integration was brought as far as it was considered possible, and all known problems were written down as tasks to be accomplished the complete software complex was used (released) as platform for the departure of all new design activities.

How to handle the integration period was prescribed in organizational procedures. Also the role and tasks of the Platform Master responsible for the integration were specified in procedures and check lists. Apart from the procedures and check lists two artifacts were essential for supporting the coordination: A project plan spreadsheet and a software directory structure.

The project plan contained information on which tasks were to be accomplished, references to a detailed description of the task, estimations per module per task, responsibility relations between modules and software designers, relations between tasks and platform periods, and the total planned work hours per platform period for each software designer. The plan supported the coordination by providing a conceptual structure for scheduling tasks, actors, and deadlines by relating these to software modules. All tasks were thus related to a given platform period. This stipulated how progress in the integration of the software should be obtained, and which actors and modules that needed to be involved when problems, changes in plans, etc. occurred.

The directory structure was developed to support the software integration. The designers had to place their personally tested software in a prespecified directory structure before the deadlines of the integration periods. The structure was distributed and placed on the software designers' work stations. The directory structures and the related software routines reduced the complexity of the meshing of the software by providing a structure stipulating how the software was to be integrated. This was done by providing a scheme for classifying concepts and structures in the software, i.e., the structure established a common standardized conceptualization of the software architecture.

Another example was a paper-based bug report form coordinating the activities concerning registration, diagnosis, and correction of software bugs. In relation to this, a list of not-yet-corrected-problems and a central file containing copies of all registered bugs were established. The bug report forms were used to collect, classify, and manage errors and suggestions. In terms of objects of coordination they mediated and stipulated coordination by means of conceptualizations of tasks, actors, software modules, and responsibilities. The main purpose of the bug report forms, the central file, the problems list, and the procedures for classifying, correcting, and reporting on the problems, was three-fold: ensure that all problems were registered and filed; establish and exhibit a clear and visible organization of the responsibilities to and for all involved designers; and stipulate how a problem was diagnosed, how the correction responsibility was delegated, and how a problem was reported corrected. The problems list was intended to provide all designers with the possibility of being aware of the state of affairs in the total software system.

The form allowed distributed registration and classification of software errors. The standardized format and the classification established a ‘common language’ for reporting on bugs and for classifying the problems, both from a usage perspective and from a software design perspective. Using the form made it possible to distribute the test activities since the need for *ad hoc* communication and coordination was reduced, and by making known problems visible to the software designers it supported the need for awareness of the state of affairs. The most important feature was that it very clearly stipulated the flow of the reporting, diagnosing, correcting, and verifying activities, i.e., when an activity was finalized the following one could be initiated more or less automatically.

The examples mentioned here and several other examples have been identified and described elsewhere (cf. e.g., Carstensen *et al.*, 1995b). The bug report form and related mechanisms will be discussed in detail in chapter 6.

## 5.8 Linking of coordination mechanisms

Coordination mechanisms can be considered local individual mechanisms supporting a demarcated well-specified aspect of the coordination of distributed activities. In the terms of Gerson and Star they are ‘local and temporary closures’ (Gerson and Star, 1986), and no single mechanism will apply to all aspects of coordination work in all domains of work. Hence, coordination mechanisms are to be conceived ofas a specialized device conceptualizing certain aspects of the field of work and the work arrangement.

When analyzing the mechanisms used at Foss Electric it became clear that the different coordination mechanisms (protocols and supporting artifacts) used in an everyday cooperative settings intersect and interlace in multiple ways and that they therefore must be made to interoperate, i.e., they must be linked (Schmidt *et al.*, 1994). The coordination mechanisms represented aspects of the structures in the field of work and the work arrangement by means of different types of links:

- a coordination mechanism may subscribe to policies and other definitions issued by other coordination mechanisms;
- a coordination mechanism may trigger another coordination mechanism into action;
- a coordination mechanism may provide a control mechanism for cooperatively managing changes to another coordination mechanism; and
- ‘foreign’ coordination mechanisms may provide indexing facilities for accessing resources in the wider organizational field;

This conception does not presuppose any single center, nor does it presuppose any well-defined organizational boundary. The context of any particular coordination mechanism (and the cooperative work arrangement using it) stretches as far as the actual links emanating from this particular coordination mechanism (subscriptions, triggers, searches).

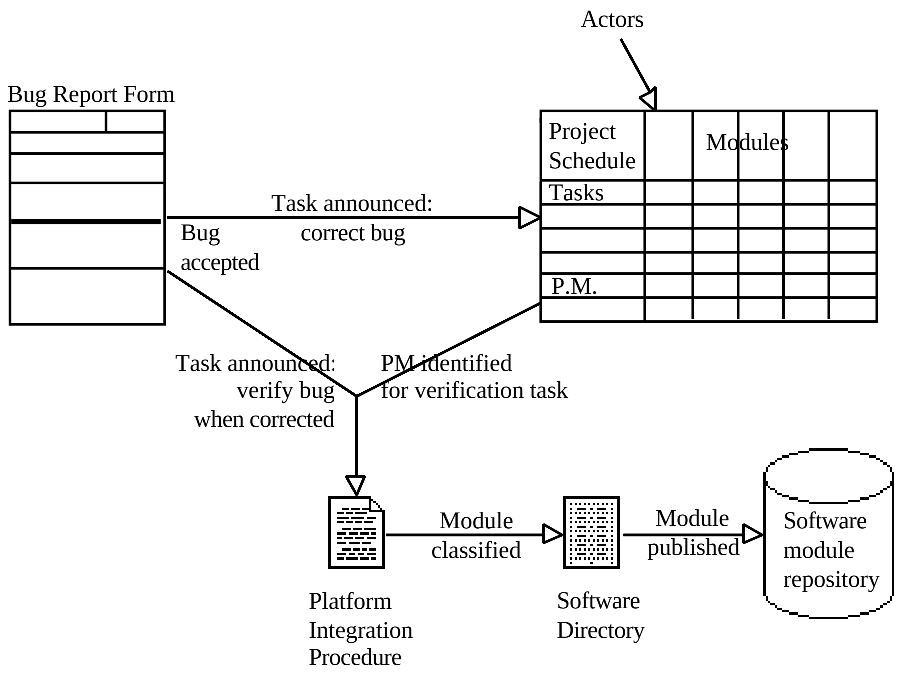

Actors
Bug Report Form
Project Schedule
Modules
Tasks
P.M.
Task announced: correct bug
Bug accepted
Task announced: verify bug when corrected
PM identified for verification task
Module classified
Module published
Software module repository
Platform Integration Procedure
Software Directory

**Figure 5-4:** The interacting coordination mechanisms used when coordinating distributed software testing and correction tasks.
'PM' denotes the platform master, i.e., the actor in charge of the integration of modules and hence verification of corrections at the end of the current platform period. We have previously discussed a similar drawing in Schmidt *et al.* (1994).

As mentioned the field study included several examples of coordination mechanisms linked to each other:

(1) Repositories were accessed by means of a coordination mechanism serving as an index for external clients, e.g., the software directory served as index for the software module repository;

(2) A mechanism triggered the execution of another mechanism when a certain condition occurs, e.g., when a reported bug was accepted, a new correction task was announced and entered in the project schedule. That is, the shift in state of one mechanism (the bug report form) triggered operations on structures in another mechanism (the spread sheet);

(3) Similarly, when a bug was reported being corrected, another new task was announced, namely the task of verifying the correction.

However, this task was pending until the platform master, responsible for the verification, was appointed and until the point in time where the next integration period was going to start. This starting point was specified in the spread sheet.

(4) Links between the mechanisms made it possible for a the instantiation of a mechanism (e.g., a particular bug report sheet) to be incompletely defined. The missing specification could be filled in later by another mechanisms. For example, was a platform period number given in a bug report form an implicit implication of the deadline for the correction task.

Some of the links and interactions between the coordination mechanisms are illustrated in figure 5-4.

The idea of linkable (computational) coordination mechanisms might have some interesting implications. The conception of computational coordination mechanisms as interacting objects may be taken to provide a foundation for creating workflow management systems in a bottom-up manner (Pycock and Sharrock, 1994b). The use of this was clearly illustrated by the Foss Electric case: in their totality the interacting coordination mechanisms constituted a workflow management system, albeit distributed and emerging bottom-up.

# 6. The bug form: An example of a real-life coordination mechanism

“With the number of developers involved, it is extremely important that all problems are registered, otherwise they just ‘disappear’. So, we designed an error and change request form […]. An important derived product then, is a list of problems reported as fixed but not yet tested. Based on the lists and the error forms the platform master can check and then report the problem corrected.”
(Software designer at Foss Electric)

In large software development projects, like the one studied at Foss Electric, the testing of the software is a demanding and difficult task. It often requires and involves many actors who need to coordinate their activities, distribute information on identified errors, be aware of results from others’ tests, negotiate the classification of an error, etc. This chapter reports from the field study conducted at Foss Electric. As mentioned, the field study addressed how the software designers in the S4000 project coordinated their activities in the last stages of the project. The field study and central characteristics of Foss Electric and the S4000 project are described previously in chapter 3.

The chapter will be organized around a description of the use of, what the designers called the “bug form” (or bug report form). The bug form is considered an example of a paper-based coordination mechanism. It will be illustrated, how the use of the form an concomitant work procedures were used in order to cope with the complexity of coordinating certain activities in relation to reporting, diagnosing, and correcting software bugs[9]

9 As my colleague Tuomo Tuikka has phrased it, the metaphor describing a software error as a bug is confusing. An error can cause reactions as if it was a living insect that should be removed, but it is, of course, created by the programmer and should as such be regarded as a software error. The word bug is stuck in our language, and it is probably as difficult to get rid of as errors in software. I will call them ‘bugs’ here since focus is on coordinating their treatment (i.e., focus is on ‘talking about bugs’, cf. Carstensen *et al.*, 1995c) rather than on studying formal techniques for finding them.

in the S4000 project. Other related mechanisms (mainly paper-based) will also be described during the description.

Early in the project, the software designers involved in the S4000 project realized problems in coordinating, controlling, monitoring, and handling the testing activities. They invented and used a standardized bug form that all testers had to fill in whenever they identified an error (a bug). To prescribe the use of the forms, a structured ring binder (being used as a central file) and a set of procedures and conventions for the use of the form were established. Some of the procedures were written down as organizational procedures, others were just conventions developed and refined during the project. A set of roles was defined in order to establish the basis for running the procedures. These were described in chapter 3.

The purpose of the form, the binder, and the concomitant conventions and procedures (hereafter often called ‘The bug form mechanism’ or just ‘The mechanism’) was to support:

(1) a decentralized registration of bugs,

(2) a centralized decision of how to overcome the identified problem,

(3) the correction activities in being handled in a decentralized manner,

(4) providing an overview of the state of affairs (with respect to registered, corrected, verified, etc. bugs), both to the involved software designers and testers, and to the management of the software development, and

(5) a final centralized process for verifying the implemented corrections[10].

The following describes the bug form mechanism by means of the Concept of Coordination Mechanisms described in the previous chapter, i.e., the bug form mechanism will be considered a protocol, encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, that stipulates and mediates the coordination of distributed activities. It is thus relevant to discuss, which functions related to coordination work the bug form mechanism

10 In the early stages of a project, a bug form was often are used as an informal means for communicating ideas, suggestions, recognized problems, etc. This type of use is, of course, relevant, but this chapter will mainly address the form used for registering and handling software bugs.

provides. Hence, certain characteristics of the mechanism become central: The mechanism is based on a publicly available and persistent artifact, the mechanism is symbolic and not coupled in any strong, tight, or irreversible way to the state of field of work, and the mechanism is based on a standardized format. Furthermore, the dimensions of objects of coordination work, along which the coordination is conducted, become essential. Objects of coordination work are the references in the mechanism pointing to components and aspects in the field of work or in the cooperative work setting itself, e.g., references to actors, roles, tasks, conceptual structures, resources of different kinds, etc. The dimensions addressed in the mechanism are discussed at the end of this chapter.

In order to provide some background, I will first present a few aspects of and relations to the software testing literature. This should not be regarded as an overview of the software testing literature, rather it is an attempt to provide a more thorough perspective for the following descriptions. The work context is briefly introduced, and the physical appearance of the form and the binder is described. Then the use and function of the bug form mechanism are described. Also the conditions triggering new activities in the mechanism, and the objects of coordination work reflected in the mechanism, are discussed. In a concluding section, I will briefly discuss how the bug form mechanism reduces the coordination effort required. Later, in chapter 11, the usefulness of the Concept of Coordination Mechanisms for describing phenomena related to coordination activities in real life work settings will be discussed.

## 6.1 Software testing

The art devoted to finding software errors is called ‘software testing’. Myers (1979) defines software testing simply as being the process of executing a program with the intent of finding errors. The understanding of software testing has changed over the last decades. In the early view of programming and testing, you ‘wrote’ a program and ‘checked it out.’ Later testing has been defined as evaluation of software or prevention of problems (cf. e.g., Gelperin and Hetzel, 1988). In modern systems development, software testing is defined as any activity aimed at evaluating an attribute or capability of a system, and determining whether it meets its required results (Hetzel, 1988). Software testing is regarded as a very important aspect of all modern software engineering, which should be an integrated part of all software quality assurance work (Yourdon, 1988).

The field of software testing spans mathematical theory, the art and practice of validation, and methodology of software development (Hamlet, 1988). A broad array of testing methods and techniques are available today, e.g., black and white box testing techniques providing a systematic approach to the design of test cases (Hetzel, 1988; Beizer, 1990) or regression testing (Dalal *et al.*, 1993). Also a fast growing stream of automated and semi-automated software testing tools influences the field today. Neither the stages, the techniques, or the automated tools will be discussed further here.

Most of the software testing literature addresses the use of testing methods and tools or the relationship between the testing and the rest of the software engineering process. The cooperative aspects of the process is only marginally addressed. Some attempts to computer support software inspection and review activities do, however, exist (e.g., Johnson and Tjahjono, 1993; Mashayekhi *et al.*, 1993).

In real life software testing is an extremely complicated activity, and conducting an exhaustive test is in practice impossible (Myers, 1979; Parnas, 1985). According to Parnas software is never faultless, software bugs are rather the norm:

> "The lay public, familiar with only a few incidents of software failure, may regard them as exceptions caused by inept programmers. Those of us who are software professionals know better: the most competent programmers in the world cannot avoid such problems. [...] Software is released for use, not when it is known to be correct, but when the rate of discovering new errors slow down to one that management considers acceptable." (Parnas, 1985, p. 1327).

Despite of all the techniques and methodologies for specific source code testing, black box testing, usability testing, etc., no methodologies exists for establishing a set of unambiguous criteria for a sufficient test strategy in order to ensure that the product is reliable, usable, and correct (Petchenik, 1985). In modern software development projects much effort is required to establish a common understanding among software developers, software testers, and software managers of when a product is acceptable. To cope with the problems, organizations involved in software testing typically apply the strategy of having people with different skills and perspectives test the software. This was the case at Foss Electric too.

Division of labor is required:

> "Effective quality control requires a certain division of labor and responsibilities. In practice, quality is not the only concern, and there is a constant struggle between quality and resource interests. Independence is needed to constantly defend a quality position and to avoid the self-deception in having systems developers evaluate their own products." (Dahlbom and Mathiassen, 1993, p. 170).

The division of labor in a software testing process can either be done so different actors perform different subtasks (detection, diagnosis, correction, etc.), or it can be organized so that each person has the responsibility for all testing activities within a limited part of the program. In any case, the participants will inevitably be mutually interdependent. The former was the predominant at Foss Electric. No matter how the work is organized coordination is needed. Although it is often tried the need for coordination cannot be eliminated just by structuring the software properly (Parnas, 1985). In order to mesh their work results, interdependent actors performing distributed software testing tasks must coordinate and negotiate their work (see e.g., Kraut and Streeter, 1995).

In many software organizations, testing is the poorest scheduled part of programming (Brooks Jr., 1982). If the importance is not recognized properly, the project planning will not include enough time. This causes exstra pressure on the testers, and, of course, the complexity of the testing work and a tight schedule influences the decisions determining whether or not a software product meets its requirements. The case is often, that there is no systematic way to search, no way to judge points selected, and no way to decide when to stop implemented in the organizations (Hamlet, 1988).

## 6.2 The physical work setting

The software design group in the S4000 project included between 5 and 12 software designers during the development. In version one of the project up to 12 designers were involved, and the average number of designer were approximately 8. During the design of version 2 of the instrument 4 - 5 software designers worked in the project. My observations and interviews were conducted over a three month period in the middle of the 10 month long version 2 design period. All the actors I interviewed had, however, also been involved in version 1, and I have considered my findings valid for both projects.

The software designers were physically placed in the same room as the project management, electronic designers, mechanical designers, chemists, and drafts personnel involved in the project. In version 2 of the project approximately 23 people were placed in the room. Apart from these, people from the marketing, quality assurance, and service departments were involved too. None of these actors were working full time in the S4000 project.

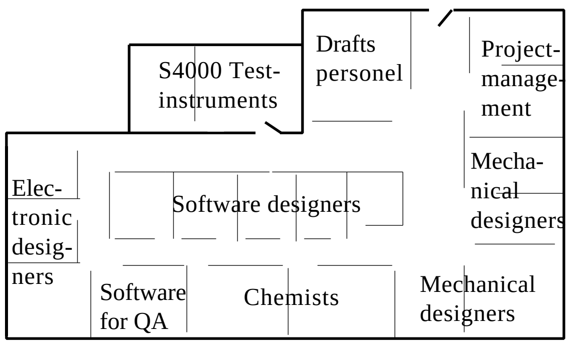


**Figure 6-1:** A sketch drawing of the room in which the designers working on the S4000 version 2 project were placed. Apart from the room containing the two S4000 test instruments it was one big room. Each 'office' was made of movable partition walls only 1.50 meter tall.

As mentioned The software design group included 5 to 12 people. Most of these were engineers in either software design or electronics. One of the designers were a computer scientist. The software design group had, of course, a lot of interaction (on a daily basis) with the other groups and the project management.

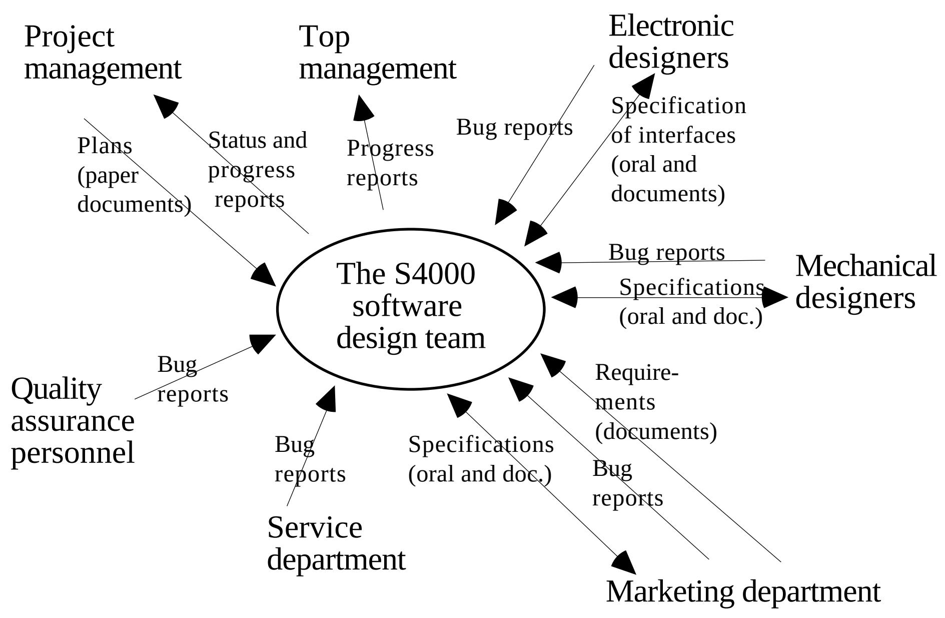

Project management
Plans (paper documents)
Status and progress reports
Top management
Progress reports
Electronic designers
Bug reports
Specification of interfaces (oral and documents)
Mechanical designers
Bug reports
Specifications (oral and doc.)
The S4000 software design team
Quality assurance personnel
Bug reports
Service department
Bug reports
Marketing department
Specifications (oral and doc.)
Bug reports
Require-ments (documents)

**Figure 6-2:** A "context diagram" illustrating the interaction the software design team had with its near context during the S4000 project. The way of drawing the diagram is inspired by the context diagrams suggested in the Work Analysis (Schmidt and Carstensen, 1990) and the communication path diagram suggested by Kensing and Winograd (1991).

If we consider testing and correction of software for the S4000 instrument the field of work, the related cooperative work arrangement must be approached as including testers from the service, marketing, and QA departments too.

## 6.3 The overall facilities provided

I have, together with Kjeld Schmidt, argued elsewhere, that when conducting work analysis a useful perspective is to approach the work arrangement as a system conducting a set of overall funtions in order to fulfil certain requirements from the target domain of the work system (cf. Schmidt and Carstensen, 1990; Carstensen and Schmidt, 1993b). If we think of the cooperative work arrangement dealing with development, implementation, and testing of software for the S4000 instrument[11] the overall functions conducted are:

- *Functional specification* specifying the functionality to be provided by the software. This is mainly done through interaction with people from the Marketing department.
- *Architecture design* specifying which modules the system should include. The work is mainly done by the spec-team through interaction with other relevant designers. The function also includes allocation of resources responsible for the modules.
- *Module interface design* specifying the interaction between the modules. This is done by the designers responsible for the modules to interact.
- *Module implementation* regards writing the code for each module. This, furthermore, includes activities related to correcting a bug reported. This is typically done by each of the designers as an individual task.
- *Software integration* dealing with integrating the modules to each other, and verifying the integration with the hardware and mechanics. This is done by all the designers in cooperation having one of them in charge (being platform master).
- *Testing* regarding the test of the software. The internal test of each module is conducted by the designer responsible for the module. The external (use oriented) test is conducted by the designers themselves and by people from the Marketing, Service, and Quality assurance departments.
- *Diagnosing* the reported bugs concerns the establishment of a hypothesis of, what is the problem and who is supposed to fix it. This is done by the spec-team, often involving one or two designers more.
- *Planning* the work concerns maintaining the work plans, keep track of the progress, and updating the plans. In the S4000 project, this was done by one of the designers involved in the spec-team. The

[11] It should be noticed, that the set of tasks mentioned here includes more than testing and correction of software which is what is mainly addressed in this chapter. The broader perspective is taken in order to provide a more coherent picture of which tasks the actors are involved in.

work plans are, of course, to be coordinated with the overall work plans for the S4000 project in total, i.e., the designer responsible had to interact with the project management in order to handle this function.

Figure 6-3 below contains a “simple” function model drawn according to the recommendations in Schmidt and Carstensen (1990). It includes the must important information areas (problem domains) the actors need access to.

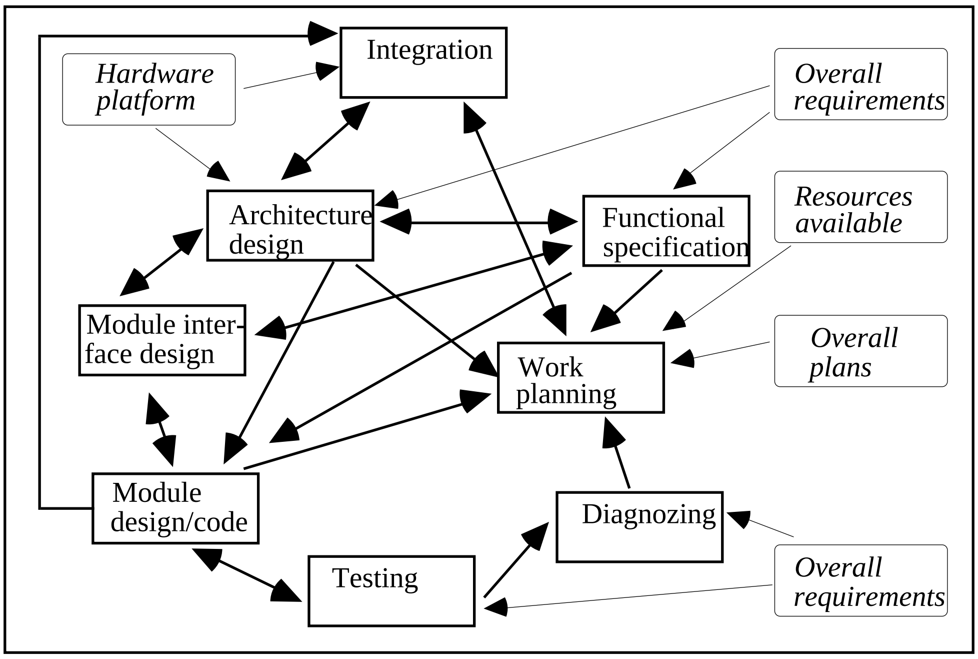


**Figure 6-3**: A function model of the software development work conducted in the S4000 project at Foss Electric. The boxes each contains one main function. The arrows between the boxes illustrates how one function provide input for another function. The rounded boxes with italic text illustrates external information required in order to fulfill a certain function.

Since focus here is on the bug form mechanism, the overall functions conducted will not be discussed further. Many of the activities will be mentioned and discussed during the detailed description of how the bug form is used as a coordination mechanism following.

## 6.4 The overall organization of the work

One of the interesting observations during the study was the overall organization of the design, test, and correction work. The work was organized in what could be called 'working cycles', or 'platform periods' as the designers called it.

During the early phases of the design work, the software designers realized, that they had severe problems in coordinating and integrating their activities, and in integrating the software modules. They explicitly stated that they needed stipulations for the control and coordination of the process of integrating and meshing the S4000 software. One of software designers phrased it as:

> "It has really been problematic that we did not have any guidelines and descriptions for how to produce and integrate our things. The individual designers are used to work on their own and have all the needed information in their heads, and to organize the work as they want to [..] When we started, we were only a few software designers. And suddenly — problems. And, ups!, we were several software designers and external consultants involved".

As the quote indicates, the problems the designers were faced with had their origin in the fact that they were not used to being so many interdependent actors. They were used to being able to handle most interaction in an *ad hoc* based manner only. Some of their problems might thus be considered 'start-up problems' only, i.e., they were not used to, and experienced in, larger collaborative settings.

The concept of "software platforms" and "platform periods" was an important invension. It was used for supporting monitoring and controlling the integration of software modules. Let me briefly sumarize the characteristics of the platform concept: A software platform was originally a point in time at which all software designers stopped all design activities and started integrating their bits and pieces. Later, organizational procedures and other artifacts were added refining how the integration should be organized. The designer worked typically 3–6 weeks on designing, implementing, and testing the software. Then a week was spend on integrating the software modules and components. The result of the integration period was a new software platform on which all new development should be based.

Version 1 of the S4000 system covered approximately 15 platform periods. After a platform period, the developers spent a week integrating the software modules. During this period no designer was allowed the continue design or implementation work until they all had approved the integrated software complex. When the integration was brought as far as it was considered possible, all known problems were written down as tasks to be accomplished, and the complete software complex was used (released) as platform for departure of all new design activities. In the later parts of the project—after having established a first running version of the total software system—the integration period was reduced to approximately two and half day.

One of the things to be done in each integration period was to appoint the platform master for the next integration period. He was then responsible for collecting all information on changes (new development, redesign, error correction, etc.) made, and for ensuring that the software was tested and corrected before it was released. He was also responsible for updating the work plans.

One of the problems the software designer felt the platform concept gave them was, of course, that some designers were more or less inactive during the integration period, and there were no structures for handling major integration problems appearing unexpected in the middle of a work period. From time to time, this type of problem caused a need for rescheduling the platforms, and immediate attainment of an extra platform integration period. Despite these problems the designers considered the establishment of the platform periods concept absolutely necessary, otherwise the work on the software would never be finished.

Apart from improving the concept by adding organizational procedures, the software designers invented a project plan spreadsheet and a software directory structure. The use of these was integrated in the platform period concept.

The project plan sheet contained information on: (1) Which tasks are to be accomplished and a reference to a detailed description of each task; (2) the estimated amount of time per module for each task; (3) the responsibility relations between modules and software designers; (4) a specification of in which platform period the tasks were planed to be finalized; (5)

and the total planned work hours per platform period for each software designer.

| **Task ref.** | **Task title** | **Incl -ude** | **Platf A** | **Platf B** | **Platf C** | **Mod. ABC** | **Mod. BCE** | **.......** | **Mod. XYZ** | |
|---|---|---|---|---|---|---|---|---|---|---|
| *3.12.6* | *Print batch* | *1* | | *Incl.-flag* | | *23* | | | *10* | *33* |
| *6.09 8* | *text* | *0* | | | | | *5* | | | *0* |
| *5.16.6* | *conveyer document* | *1* | | | *Incl.-flag* | | *10* | | *45* | *55* |
| *1.10.4* | *PCIO synchr.* | *1* | | *Incl.-flag* | | | | | *5* | *5* |
| *11.03.2* | *PCIO-message* | *1* | *Incl.-flag* | | | | *4* | | | *4* |
| *11.04.5* | *UI standard* | *0* | | | | | | | | |
| ....... | | | | | | | | | | |
| **Total sum** | | | | | | *23* | *14* | | *60* | *97* |
| **Sum A** | | | | | | | *4* | | | |
| **Sum B** | | | | | | *23* | | | *15* | |
| **Sum C** | | | | | | | *10* | | *45* | |
| | | | **A** | **B** | **C** | | | | | |
| | Designer | I | *104* | *138* | *140* | *1* | | | | |
| | Designer | II | *109* | *124* | *185* | | *1* | | | |
| | Designer | III | *121* | *105* | *0* | | | | *1* | |
| | Designer | IV | *0* | *130* | *141* | | | *1* | | |
| | ..... | | | | | | | | | |
| | **Sum** | | *334* | *497* | *466* | | | | | |
| | **Ideal** | | *112* | *113* | *137* | | | | | |

**Figure 6-4:** An illustration of the project plan spreadsheet. The example shows three platform periods (A, B, and C), three modules (ABC, BCE and XYZ), six different task of which two are postponed (a 0 in Included) and four involved designers. All italics and shadings are my additions. The shadings illustrate the ‘linking’ from a specific module to a specific designer being responsible the module.

Before the deadlines related to each of the integration periods, the designers had to place their software in a directory structure developed especially with the purpose of supporting and enhancing the software integration process. Each piece of software should be tested by the designer himself before placing it in the directory. The directory structure was continuously adjusted, both according to the architecture of the total software system, and to the distribution of responsibility for the individual modules among software designers. The structure was distributed on the software designers' work stations. Furthermore, a set of software routines was implemented. These were used by the platform master for automatically collecting, compiling, and integrating all software modules. In order to support the use of the directory mechanism, a set of check lists, standards, and procedures were established within the concept of platform periods.

## 6.5 Roles involved in the software testing and correction

Let us now return to the description of the bug form mechanism and its use. First a brief repetition of the actor roles that are involved in the work:

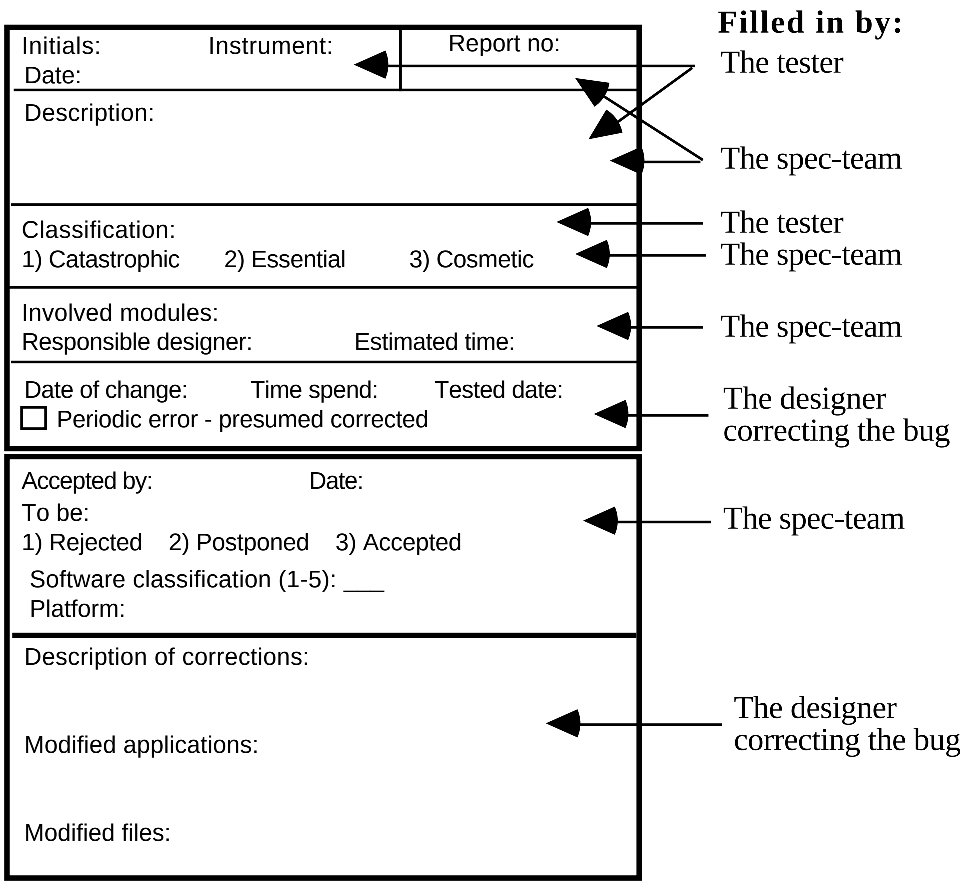
- The software designers responsible for designing, implementing, and maintaining one or more of the software modules, and for correcting bugs.
- The spec-team: three software designers responsible for diagnosing reported bugs and deciding how to handle each of the bugs.
- The platform master responsible for managing and coordinating all the activities involved in integrating the outcome of the current working period (platform period).
- The project plan manager responsible for maintaining a project plan spreadsheet.
- The testers involved in the concrete testing of the software embedded in the S4000 instrument. The testers could be affiliated in most of the departments at Foss Electric.
- The central bugs file manager responsible for organizing and maintaining the central bug file, a ring binder containing copies of all reported bugs and organized according to their status.

This list is a condensed version of the description provided in section 3.2.1.

## 6.6 The bug form and the binder

The bug form mechanism includes two different types of artifacts. A bug report form and a binder in which forms are filed.

The most interesting is the bug report form, or rather forms since there exists one form (plus a copy) for each bug registered. The form is a two pages form (both sides of one sheet of A4-paper) used by all designers and testers involved in testing and developing the software for the S4000 instrument.


Initials: Instrument: Report no:
Date:
Description:

Classification:
1) Catastrophic 2) Essential 3) Cosmetic

Involved modules:
Responsible designer: Estimated time:

Date of change: Time spend: Tested date:
☐ Periodic error - presumed corrected

Accepted by: Date:
To be:
1) Rejected 2) Postponed 3) Accepted
Software classification (1-5): ___
Platform:

Description of corrections:

Modified applications:

Modified files:

**Figure 6-5:** A translated version of the 2 pages bug form invented and used in the S4000 project. On the right side of the figure is illustrated which actors (roles) that fills in the different fields.

The form is filled in step-wise—partly by a tester recognizing a bug, partly by the spec-team diagnosing the problem, and partly by the designer correcting the bug. How the form is filled-in, and the intention of the different fields, will be described in further details in the following sections. This is done by means of a state-transition diagram illustrating possible states of the bug form mechanism, and by a non-formal description of the procedure, how the form is updated, how the routing between the activities is handled, etc.

One of the basic ideas in the bug form mechanism is, as mentioned, that there exists exactly one (original) form filled in for each registered bug. The position of the form is an implicit indication of the state of the bug (registered, diagnosed, corrected, or verified). In some stages of the registering, diagnosing, and correcting process, a copy of the form is inserted in the central file (the binder) in order to keep an overview of the state of affairs.

The second artifact in the mechanism is a ring binder containing all forms (either the original or a copy) filled in and diagnosed by the spec-team (i.e., all registered bugs). The binder is physically placed in the same room as the actors involved in developing the S4000 instrument, i.e., the binder is (intended to be) easily accessible to all the software designers engaged in the project. Some of the involved testers are also engaged in the project as software, hardware, or mechanical designers placed in the same room as the software designers and the binder. Other testers are affiliated within other departments (e.g., the marketing department). They had a more complicated access to the binder.

The purpose of the binder is to provide awareness to all involved designers and testers of the state of affairs in the testing of the instrument. Furthermore, the binder is used by the project management to get an overview of the state of affairs and the progress in the project.

The binder is maintained by one of software designers engaged in the project. He does this through a close interaction with other software designers, especially with the actor being platform master (cf. section 3.2.1).

All bug forms are filed in the binder according to the following categories: (1) non-corrected catastrophes, (2) non-corrected semi-serious problems, (3) non-corrected cosmetic problems, (4) postponed, (5) rejected, (6) corrected but not yet tested, and (7) corrected problems. In each of these entries, the forms are inserted and organized in a chronological order. According to decisions taken by the spec-team and messages from the designers concerning specific problems (bug forms), results from the platform integration, etc. the forms are successively re-filed.

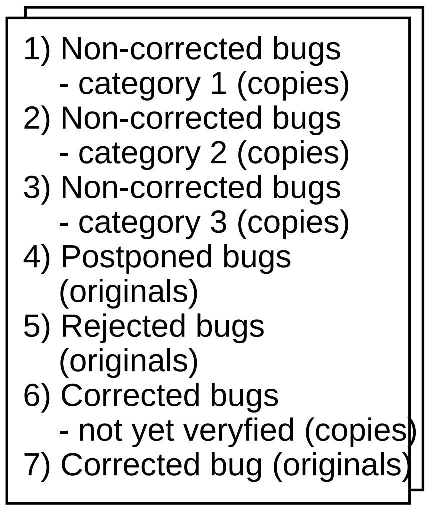

1) Non-corrected bugs
- category 1 (copies)
2) Non-corrected bugs
- category 2 (copies)
3) Non-corrected bugs
- category 3 (copies)
4) Postponed bugs
(originals)
5) Rejected bugs
(originals)
6) Corrected bugs
- not yet veryfied (copies)
7) Corrected bug (originals)

**Figure 6-6:** The seven entries used for filing the bug forms. The headings for the entries are translations of the table of contents in the original binder.

The use and maintenance of the binder will be described further in the following. The following three sections will describe the behavior of the bug form mechanism, and how the mechanism was used in the S4000 project. I have chosen to describe the bug form mechanism from three different perspectives:

First, the bug form mechanism is approached as a ‘box’ (in section 6.7). This description is concerned with, which states the mechanism can be in, i.e., which state can the handling (treatment) of a given bug be in. This is done by means of a state-transition diagram illustrating the possible states of the mechanism, and the possible transitions from one state to the other.

The second description (section 6.8) is organized as a procedural description of the overall procedure used for handling each bug when the mechanism is running. This is organized as a step-by-step description describing what goes on in each step, and the most common deviations from the ‘standard procedure’. This description includes references to the states defined in relation to the state-transition model presented in section 6.7.

Third, section 6.9 describes the bug form mechanism from the perspective of the involved actors and roles, i.e., which information is flowing among the actors (roles), and in what sequence.

## 6.7 A state-transition model for the bug form mechanism

When approaching the bug form as a black box, we are not interested in modeling what goes on inside the mechanism, only in identifying which different observable states the mechanism can take. The state-transition model illustrated below (figure 6-7) reflects the possible different states of the bug form mechanism, and the possible next “legal” steps.

The state-transition model can be considered a model of the mechanism itself, i.e., a model of the possible states (or status) the coordination of the testing activities can be in. It is not a model of the states, procedural steps, or of the activities involved in the testing work itself. Neither is it a model reflecting the different actors involved in the software testing and correction work.

An empty form shifts state when it is filled-in (from initial—BR0—to BR1), and again when it is send to the spec-team (BR2). The state is changed when the problem is classified (BR12, BR3, BR4, or BR7), and again when it is estimated (BR8) and send to the responsible designer (BR5). When the problem has been dealt with a new state occurs (BR6) and, again, when the form is send to the central file (BR9). Before an integration period, the form is send to the platform master and the state changes (BR13). Finally, the state changes when the correction has been verified (BR10), and when the form is filed (BR11).

As it can be seen from this very condensed description the state of the mechanism typically changes when 1) information is added to the form, 2) the form is routed to another actor, or 3) the form is filed.

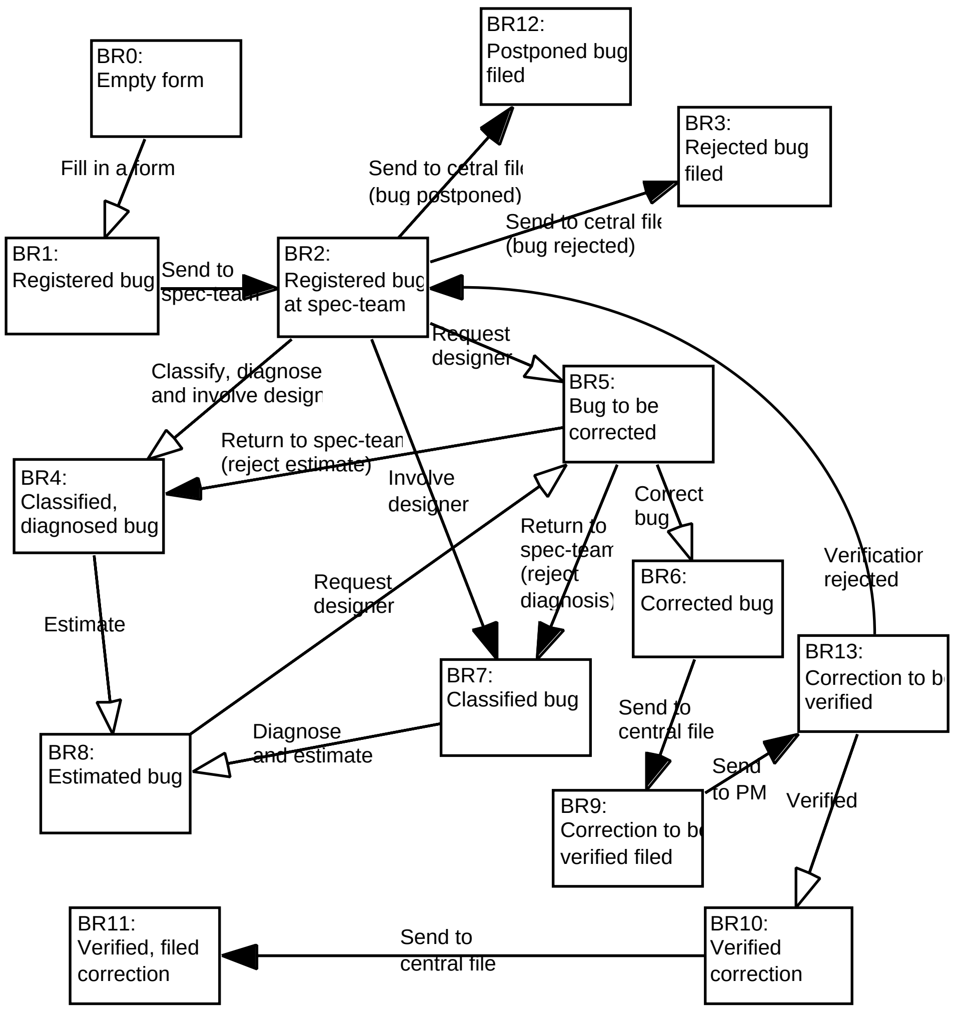


**Figure 6-7:** A state-transition diagram of the bug form mechanism. Each box illustrates a possible state the mechanism. The arrows illustrate possible actions (transitions) that can be taken. Each arrow points at the new state a given action will result in. Arrows with a white head reflects an action that makes changes to the content of the mechanism; here mainly in terms of adding new information to the form. Arrows with black heads reflect changes in who is in control of the mechanism, i.e., allowed to change the state of the mechanism.

When approaching states, and possible transitions from these, it is also relevant to address what triggers the shifts (transitions) in the state of the mechanism. In most cases, shifts in state of the mechanism are caused by decisions taken by one of the involved actors, or by events in the context in which the bug form mechanism functions. The triggering conditions are discussed in the list below, one state at a time. Final states (i.e., states without outgoing transitions) are not included in the list.

BR0: The only transition is the BR0 to BR1. This is triggered by an event in the work: a tester decides to categorize a problem in the software as a bug.

BR1: To follow the procedures, the tester should send the form to the spec-team.

BR2: Here are a number of possible transitions. All transitions are triggered by decisions taken by the spec-team and related to the reported bug:

- BR2-BR12 if the correction is postponed.
- BR2-BR3 if the bug is rejected.
- BR2-BR4 if the estimation cannot be done without involving the responsible designer.
- BR2-BR7 if the diagnosing requires involvement of other designers.
- BR2-BR5 if no involvement is required.

Since the spec-team typically have meetings ones a week, we can say that the decisions made by the spec-team are indirectly triggered by an external event: the occurrence of a specific point in time. The decisions taken by the spec-team can also be regarded as being implicitly influenced by another external source. All decisions taken by the spec-team are taken under influence of the constraints and policies directed from the management of the company. A classification will, for example, be closely related to the general product acceptance criteria and the time left before the product should be released.

BR4: The BR4-BR8, BR7-BR8, and the BR8-BR5 are all triggered by decisions taken by the spec-team regarding the classification, the diagnosis, or the estimation.

BR5: All possible transitions from BR5 depend on decisions taken by the responsible designer. As for BR2 the decisions are influenced by the outside world since all decisions are taken within the constraints defined by the general policies.

BR6: BR6-BR9 is triggered when the responsible designer decides to send the form including correction information to the central file manager, i.e., the designer decides, that the problem (the bug) has been solved.

BR7: See BR4.

BR8: See BR4.

BR9: The transition from BR9 to BR13 is triggered when the central file manager decides to send a pile of forms describing corrections to be verified to the platform master. This decision is related to an external event: the occurrence of a certain point in time. The organizational procedures state that, the central file manager should send the forms to be verified to the platform master two days before the next integration period is going to start.

BR10: When the platform master verifies the corrections, he decides on the acceptance of the correction. If a correction is acceptable, he piles the form (BR13-BR10), and when all forms are controlled he sends the accepted forms to the central file manager (BR10-BR11).

BR13: The platform master decides on the acceptability of the corrections. If it is okay see BR10, otherwice BR13-BR2 is triggered.

As it can be seen from the descriptions, all the transitions are triggered by decisions taken and actions made by the involved actors. Some of these actions have a second order triggering condition related to the occurrence of a specific point in time. None of the triggering situations are directly related to the “outside world” apart from these time related events and external policies.

## 6.8 A procedural description of the bug form mechanism

The previous section approached the bug form mechanism as a black box. This section will describe the bug form mechanism in terms of procedural steps conducted, when the mechanism is “running”. Many of the procedural steps described here are described in the organizational procedures established by the software during the S4000 design.

The overall procedure for the mechanism, and how the forms and the binder are updated, are illustrated in figure 6-8 below. The overall procedural steps are:

(1) A bug is recognized by a tester. He fills-in a form and classifies the bug.

(2) The tester sends the form to the spec-team.

(3) The bug is re-classified and diagnosed by the spec-team.

(4) The responsible module (and thereby designer) is identified by the spec-team.

(5) The correction time is estimated by the spec-team.

(6) The required correction tasks are incorporated in the work plans. This is done by one of the members of the spec-team.

(7) The spec-team sends the form (the original) to the responsible designer. This can be regarded as a correction request. The spec-team sends a copy of the form to the central file (the binder).

(8) The bug is corrected and additional correction information is filled-in on the form by the responsible designer.

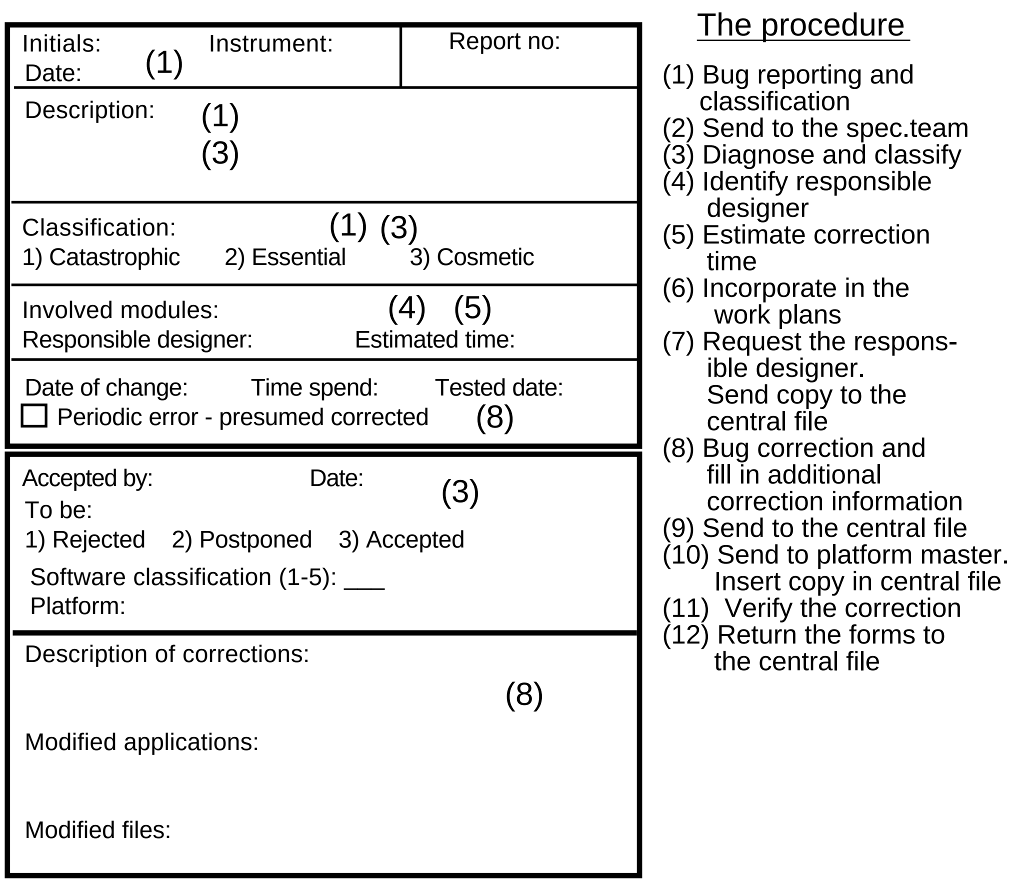
(9) The responsible designer sends the form to the central file.

(10) The central file manager inserts a copy of the form in the central file, and sends the form (the original) to the platform master.

(11) The correction is verified by the platform master.

(12) The accepted forms are returned to the central file manager.

Each step in the procedure is described in further detail in the following. The activities will be related to the use of the forms and the binder. The descriptions will be related to the different states and transitions in the diagram in figure 6-7.



Initials: (1) Instrument: Report no:
Date:

Description: (1)
(3)

Classification: (1) (3)
1) Catastrophic 2) Essential 3) Cosmetic

Involved modules: (4) (5)
Responsible designer: Estimated time:

Date of change: Time spend: Tested date:
☐ Periodic error - presumed corrected (8)

Accepted by: Date: (3)
To be:
1) Rejected 2) Postponed 3) Accepted
Software classification (1-5): ___
Platform:

Description of corrections:
(8)
Modified applications:

Modified files:

The procedure

(1) Bug reporting and classification
(2) Send to the spec.team
(3) Diagnose and classify
(4) Identify responsible designer
(5) Estimate correction time
(6) Incorporate in the work plans
(7) Request the respons-ible designer. Send copy to the central file
(8) Bug correction and fill in additional correction information
(9) Send to the central file
(10) Send to platform master. Insert copy in central file
(11) Verify the correction
(12) Return the forms to the central file

**Figure 6-8:** The bug report form and a 12 steps overall procedure for the use of the mechanism. The numbers in the form indicate in which step of the procedure the field is affected.

Most cases (forms) are handled according to the prescribed procedures. In some cases an actor might, however, choose to deviate from the procedure, e.g., if a tester knows who is responsible for a specific bug, he can contact the designer without involving the spec-team. Some of these deviations are characterized in the following. The description will, however, not attempt to describe all possible situations. A basic assumption related to the Concept of Coordination Mechanisms is that, it is considered impossible to predict all possible work situations, i.e., it will not make sense to attempt to include an exhaustive description of possible deviations.

Although the S4000 project is finished the following descriptions are given in present time.

Bug reporting:

The S4000 software is tested on software simulators and on instrument prototypes. The project has distributed detection, registration, and classification of software bugs. Occasionally testers and software designers engage in preliminary discussions on the interpretation of problems or possible bugs. When a bug is identified by one of the involved testers, a new form is filled-in. The intention is, that the tester fills-in a form for each bug he identifies. It might, of course, be difficult to decide if a problem is “the same” as one identified earlier, or to see if a concrete problem actually is caused by several different software bugs.

The tester fills-in his initials, the date, an identification of the instrument and the software version he has used, and he describes what he did and how the instrument reacted. Finally he fills-in a classification (catastrophic, essential, or cosmetic) of the importance of the problem. Filling in the information is reflected in the state-transition diagram (figure 6-7) as the transition from state BR0 to state BR1.

Some errors are impossible to reproduce, and hence difficult to describe during registration. My observations indicate that in at least 20% of the bug reports, testers were not able to describe the problem in detail. If it is difficult for the tester to fill in the required information, the spec-team member fills-in the form after having discussed the problem with the tester. As one of the spec-team members said:

> “The form is not very user-friendly. We often have to force them *[the testers]* to fill in a form. Sometime they just send in a note describing what they have seen, and we must produce a form.”

The classification of errors as either catastrophic, essential or cosmetic was done according to the tester’s perspective. As one of the spec-team members phrased it:

> “People, depending on who they are, often interprets a catastrophe in a different way than I do. An inconsistency in the user interface might, for example, be a disaster to a marketing guy, whereas it is a cosmetic problem to me”.

An exception to the described procedure is, that the tester personally contacts one of the members of the spec-team (or one of the other software designers), and orally reports on the problem. In this case, the spec-team member or designer will, if the problem is considered relevant, fill in a form.

### Send the form to the spec-team:

Having filled in a new form, the tester sends the form to the spec-team. This is illustrated in the state-transition diagram as the transition from BR1 to BR2. Usually this is done by use of internal mail.

There are, at least, two exceptions to this procedure step. If the tester considers the identified problem to be very important (having classified it as catastrophic), or he decides that the diagnose and correction of the problem cannot be delayed further, he might contact one of the members of the spec-team and discuss it immediately. Furthermore, if the tester knows who is probably going to be responsible for correcting the bug he might choose to contact the designer directly, and discuss the problem with him before handing in the form to the spec-team.

### Diagnosing the bug:

This section describes what is illustrated as three steps in the procedure in figure 6-8: The diagnose and classification of the bug (step 3), the identification of the responsible designer (step 4), and the estimation of the expected correction time (step 5). These steps are all handled by the spec-team and are usually conducted concurrently and intertwined. The spec-team will typically have a meeting ones a week. In periods with very intensive testing activities it might be more frequent.

The spec-team starts out by checking, if any of the incoming bugs are identical. This is done by comparing the description of the incoming forms. If two registered bugs are considered identical only one of the forms is treated.

For each form, the spec-team decides if the described bug can be accepted as a bug. If not, the form is classified as rejected and send to the central file manager (the transition from BR2 to BR3) who inserts it in the “Rejected bugs” entry (cf. figure 6-6). Next step is to decide whether the bug is important or can be postponed. If the bug is postponed, the form is classified as postponed and the form is send to the central file manager (BR2 to BR12). The central file manager files the form in the “Postponed bugs” entry.

The remaining forms are classified as accepted. Then they are classified according to importance. This is done by use of two classification structures on the form. First, the classification made by the tester is corrected (re-classified). Usually the classification is not changed, but if the spec-team disagrees in the classification they might change it. If there is a deviation in the tester classification (and description) and the spec-team classification, the spec-team might choose to contact the tester and negotiate the classification. Second, the importance is filled in in the field for “Software classification” (cf. figure 6-8). This field also indicates in which of the next platform periods the bug should be corrected. The 1-5 classification on the form indicates that the work on version 2 of the S4000 was organized as 5 platform periods.

If the diagnose and the estimation of time required to correct the bug is fairly simple, the spec-team fills-in (writes) the diagnose in the “Description” field of the form, fills-in the “Involved modules”, the “Estimated time”, and “Responsible designer” fields. The responsibility is distributed (i.e., all modules have one software designer associated). Hence, the decision of, who is responsible, is based upon which modules are considered involved. In cases where the diagnosis and time estimation are simple, the spec-team incorporate the correction work in the plans (cf. the following section on ‘Incorporate in plans’) and sends a request (the form) to the designer (BR2 to BR5), cf. the following section on ‘Send request to responsible designer’. The responsible designer is always able to reject the estimate. This is described further in the section on ‘Correct the bug’.

If the diagnose is complicated, the spec-team can choose to call in the designer responsible for the relevant modules (BR2 to BR7). The designer is then involved in diagnosing the problem and estimating the correction time (the transition from BR7 to BR8). When the relevant decisions have been made, the spec-team incorporates the correction work in the plans (cf. the ‘Incorporate in plans’ section below) and sends a request (the form) to the designer (BR8 to BR5).

If the estimation requires only involvement of the responsible designer, the spec-team fills-in the “Description” field of the form, the “Involved modules” field, and the “Responsible designer” field (BR2 to BR4). The spec-team and the responsible designer collaborates on deciding the required corrections time (BR4 to BR8). As for the previous situations, the spec-team incorporates the correction work in the plans and sends a request (the form) to the designer (BR8 to BR5), cf. the section on ‘Send request to responsible designer’.

The diagnosing and estimating work is usually a complicated task requiring involvement of experts in different fields, use of the software specifications, source code, and documentation, etc. A description of this in further detail is, however, considered out of the scope.

#### Incorporate in work plans:

The fact that a bug has been identified, diagnosed, and estimated results in a new task that has to be accomplished. The spec-team requests the designer or manager responsible for the overall project plan to include the new task (cf. the project plan spreadsheet in figure 6-4). In the S4000 project, the project plan manager was identical to one of the spec-team members. Information on the task, the responsible designer, the estimated time, and the platform period are incorporated in the plans by the actor responsible for the overall project plan.

Since the plans are considered “outside” of the bug form mechanism described here, the incorporation does not affect the mechanism, i.e., no transitions are triggered in the state-trasition model. It can rather be seen as an example of a linking (cf. section 5.8) between two different coordination mechanisms supporting the coordination of software development and testing: The bug form mechanism and another mechanism supporting scheduling and allocation of tasks and human resources. A thorough discussion of the observed linking between the coordination mechanisms is given in section 6.11.

A possible exception to incorporating correction tasks in the work plans is, when the problem is corrected immediately by a designer or a member of the spec-team. Then the task will never occur in the plans, and it will not be “officially visible”, that the designer has spend time on this task.

#### Send request to responsible designer:

The updated form containing information on the problem, the classification, the diagnose, the involved modules, and the estimated correction time is sent to the responsible designer. The designer is supposed to consider the form as a request. The request requires either, that the bug is corrected, or the request is rejected (by contacting a spec-team member).

A copy of the form is send from the spec-team to the central file manager. The central file manager inserts the copy in the binder according to the classification on the form (cf. figure 6-6), i.e., in one of the first three entries: “Non-corrected category 1”, “Non-corrected category 2”, or “Non-corrected category 3”.

If the spec-team has made the diagnose and estimation without involving other software designers sending the request is reflected in the diagram as the transition from BR2 to BR5. Otherwise the action is reflected in the state-transition diagram as BR8 to BR5.

An exception is, of course, that one or several of the members of the spec-team personally hand over the form to the designer and/or to the central file manager. This might be done just because it is the easiest way to do it, or it might be caused by the need for additional information, or if it is considered important that the correction work is launched immediately.

#### Correct the bug:

Having received a form, the designer checks if the diagnose, the estimate, and the deadline (the platform period) are acceptable. If the designer considers the estimate as to optimistic (low) he returns the form to (or personally contacts) a spec-team member with a note stating, that the estimate is unacceptable (the transition from BR5 to BR4). The estimate is then negotiated with the spec-team. A basic convention states that “the responsible designer is always right”.

The designer might also disagree on the diagnose, or in the descriptions of which modules are involved, i.e., he might disagree in what is the problem and/or who is responsible. In this case he will return the form to the spec-team with a note explaining, what is the problem in the diagnose (BR5 to BR7). The diagnose and the estimate will then be negotiated with the spec-team.

If the designer accepts the diagnose, and assumes that he can handle the problem within the estimated resources and deadlines, he corrects the bug. That might, of course, be done at a much later point in time.

Since all the 5 to 10 designers correcting the software is placed in the same room, many problems are mainly conducted by an abundance of meetings and discussions. This is, however, quite problematic. One of the designers characterized the problem as follows:

> “The problem we have right now is that the software architecture is difficult to decompose sufficiently so that one designer can handle a component. We are all working on several components, and work on a single component often involves two to four men, and perhaps even some of the electronical designers too. Then we need coordination [..] We have recently started a process where we try to produce more formal documents and agreements about the things we work with, we have not been good at doing this up till now, but now we *have* to do it.”

Having corrected the bug, and tested the corrections, the designer fills-in the fields of “Date of change”, “Time spend”, “Tested date”, and if relevant tip of the “Periodic error” field on the form. Furthermore, he describes the corrections made, and the applications (or modules) and files affected. The addition of information to the form is reflected in the BR5 to BR6 transition in the state-transition diagram.

Finding the origin for a bug, and correcting it, is normally an extremely complicated task requiring intense studies of the source code and specifications, discussion with other designers, etc. Describing the character of this is out of the scope here.

#### Send the form to central file:

When the corrections are implemented and information is added to the form, the designer sends the form to the central file manager. The central file manager is responsible for maintaining the ring binder. He removes the old copy of the form (placed in one of the first three entries, cf. figure 6-6) from the binder and throws it away. A copy of the updated (received) form containing the additional information regarding the corrections is filed in the binder in entry 6 “Corrected bugs to be verified” (BR6 to BR13).

#### Send the form to platform master:

The central file manager sends the form (the original) to the platform master, illustrated as BR13 to BR9. The platform master is supposed to consider the form a request for verifying the described corrections in the next platform integration period.

There might be situations where the platform master decides to reject such a request. I do not have any empirical material indicating that this has happened or could happen, and how it is handled. If such a situation would occur, it would probably result in a situation similar to the one described as the "correction cannot be verified" described in the following section.

Verify the corrections:

During the platform integration periods, the platform master is responsible for verifying the corrected bugs, i.e., verify that each bug is corrected sufficiently without introducing new problems. This is done either by the platform master himself, or he delegates the responsibility to other software designers. As on of the designers put it:

> "Usually we produce a list of all the problems we have identified on a large whiteboard. We then discuss whether this is a problem—an error—or not. Actually, it's the platform master who does that. If it's a real heavy problem you are immediately summoned and asked to correct it."

The verification is also a complex and demanding task out of the scope to describe here. If the verification process results in an acceptance of the correction the form is placed in a pile of accepted forms (the transition from BR9 to BR10).

If the verification process results in a rejection of the correction one of several things can happen: The platform master can, if it is a minor problem, either correct the code himself, or ask the responsible designer to do this immediately. If this is possible the form can be piled in the accepted forms pile (transition BR9 to BR10). If the correction of the bug can be verified, but it has introduced a new bug (or several) the form is piled as verified (transition BR9 to BR10) and a new empty form is filled-in describing the new bug (the transition from BR0 to BR1). Finally, if the correction cannot be verified, the platform master adds a note in the "Description field" indicating the problems and sends the form to the spec-team (BR9 to BR2).

A possible exception to the last mentioned procedure is, that the platform master, instead of sending the form to the spec-team, returns the form to the responsible designer. This requires that the platform master ensures, that a copy of the form is inserted in the relevant non-corrected bugs entry in the central file (done by contacting the central file manager).

Furthermore, it requires that the platform master ensures, that the work plans are updated (cf. the section on 'Incorporate in work plans').

### Return the accepted forms to central file:

Finally the forms piled in the accepted forms pile are send from the platform master to the central file manager. The central file manager removes all the copies of forms placed in entry 6 "Corrected bugs not yet verified" of the binder. The forms (the originals) received from the platform master are inserted in the "Corrected bugs" entry of the binder (BR10 to BR11).

The central file manager can check, if the copies taken out are identical to the inserted ones. Forms that are removed, without a corresponding form inserted indicates corrections that could not be verified. The correctness of this can then be checked by comparing with forms in the non-corrected bugs entry. The procedures do not specify anything about this. The study indicated that whether this check is made or not depends on the work load for the central file manager.

### Monitoring what goes on:

Apart from the twelve overall steps described above, a central activity in coordinating the process of registering and correcting bugs was to establish an overview of the state of affairs. The designers, testers, and managers tried to measure progress of the work, and to be aware of the number of bugs to be corrected, the accumulated estimation of correction time, changes that might affect other modules, etc. The designers tried to be aware of corrections and changes affecting their modules. The spec-team members needed to know the state of affairs before each spec-team meeting. The testers frequently tried to obtain an overview in order to avoid wasting their time on reporting already registered bugs. The platform master needed an overview of corrections to be verified in the next integration period in order to plan the integration work. And management tried to get an overview of the progress of the whole development project.

There were basically three information sources used for monitoring: informal communication, the binder, and the list of bugs not yet corrected. There was a lot of informal communication and discussion among the designers about what kind of corrections and changes they had made. Some of the testers discussed changes in the software with the designers several times a week, whereas others never contacted the designers directly. Even though the designers sat in the same room and were engaged in discussions every day, it was difficult for them to be aware of the state of affairs:

> "Earlier, the channel driver guy and I had a clear agreement. Oral discussions and a sketch drawing were sufficient. But in a project as large as the S4000 we don't have a complete overview of the software complex. Then you are in big trouble when the other guys change their code" (Software designer in the S4000 project).

Testers as well as designers found it very difficult to obtain the necessary overview by consulting the bug form binder, mainly because the forms were only organized according to the seven categories (cf. section 6.6). This made it almost impossible to determine whether the same bug had been reported in several bug forms. It was the intention, that a specification of the corrections should be included in each form. In order to be aware of relevant changes, the designers were expected to browse through all the forms in order to see if anything was of interest. The binder contained approximately 1400 forms at the end of the project!

The third source for getting an overview was a weekly produced list of registered bugs that had not yet been dealt with. One of the designers phrased the problems with this as:

> "Originally the intention was to produce statistics of the number of known-but-not-yet-fixed problems and use this as a management tool. The management hoped to find a decreasing curve on the week-to-week measurement. They didn't. But we realized that as a management tool this can only be used if you have a stable product. We didn't have that."

## 6.9 The flow of the bug form mechanism from a roles and actors perspective

The previous sections included two different approaches: The perspective of the state of the bug form mechanism, and a procedure oriented approach.

It is, furthermore, obvious to approach the mechanism from the perspective of the involved actors, i.e., to focus on the information flow from the view point of the involved roles and actors. Figure 6-9 below illustrates the roles involved in coordinating the software testing and correction activities in the S4000 project, and figure 6-10 is an example of the flow from the actors perspective.

There are, as mentioned in section 6.5, basically six different roles involved in coordinating the software testing and correction activities in the S4000 project. These are the testers testing the software, the spec-team diagnosing the bugs, the software designers correcting the bugs, the project plan manager including the correction tasks in the plans, the platform master verifying the corrected bugs, and the central file manager maintaining the central file in order to keep track of the state of affairs.

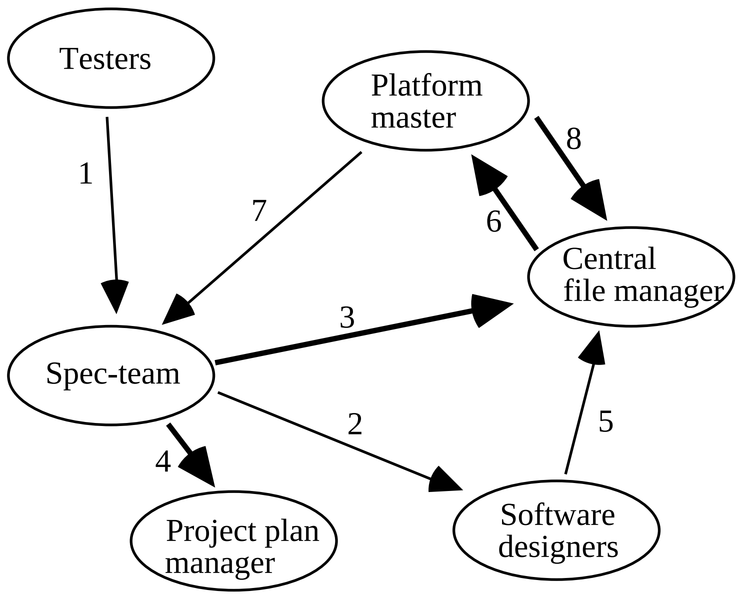


**Figure 6-9:** A visualization of the roles involved in the software testing of the S4000 project, and the information flow between them. The information flow described in the figure concerns only the stipulated (through organizational procedures) flow of the bug form mechanism. The thick arrows (3, 4, 6, and 8) indicate, that the flow is often a bunch of forms sent. The thin arrows indicate, that the forms typically are send one at the time. Other types of information are frequently exchanged between the actors and there might be situations were one of the actors choose not to follow the stipulated information flow. It should be noticed, that one actor might have two or more roles (cf. figure 6-10).

The general flow of information (route of the forms) follows the eight major steps illustrated in figure 6-9:

(1) The testers send forms describing recognized bugs to the spec-team.

(2) The spec-team adds diagnose and estimation information to the form and sends it, as a request, to the software designers.

(3) The spec-team sends rejected forms and copies of accepted forms to the central file manager.

(4) The spec-team sends information on new (correction) tasks generated to the project plan manager.

(5) The software designers add correction information to the form and sends it to the central file manager.

(6) The central file manager sends a pile of forms containing correction information to the platform master.

(7) Forms containing corrections, that cannot be verified, are send from the platform master to the spec-team which then recycle them in the process (starting from flow 2).

(8) Forms containing corrections, that can be verified, are send from the platform master to the central file manager.

As mentioned, actors can have several roles, and several actors can fulfill similar roles. The example in figure 6-10 illustrates this.

James is testing a prototype and reports a problem (1a). During his design work, Jack realizes a problem in the interface between his and one of the other modules. This is reported (1b). John diagnoses problem a, and decides that it should be corrected by Dave (2a) and that Alan should be responsible for b (2b). John, furthermore, sends information on the two problems to the central file manager, Alan (3), and to the plan-manager, Dave (4). Dave corrects problem a, and informs the central file manager (5a). Alan corrects problem b and files the form (5b). Two days before the next integration period, the platform manager, Jack, asks Alan (the central file manager) to check which bugs have been corrected. Alan sends the two forms on problem a and b to Jack (6). During the integration period, Jack checks the corrections. Problem a is okay, and the form is send to the central file manager for final filing (8a).

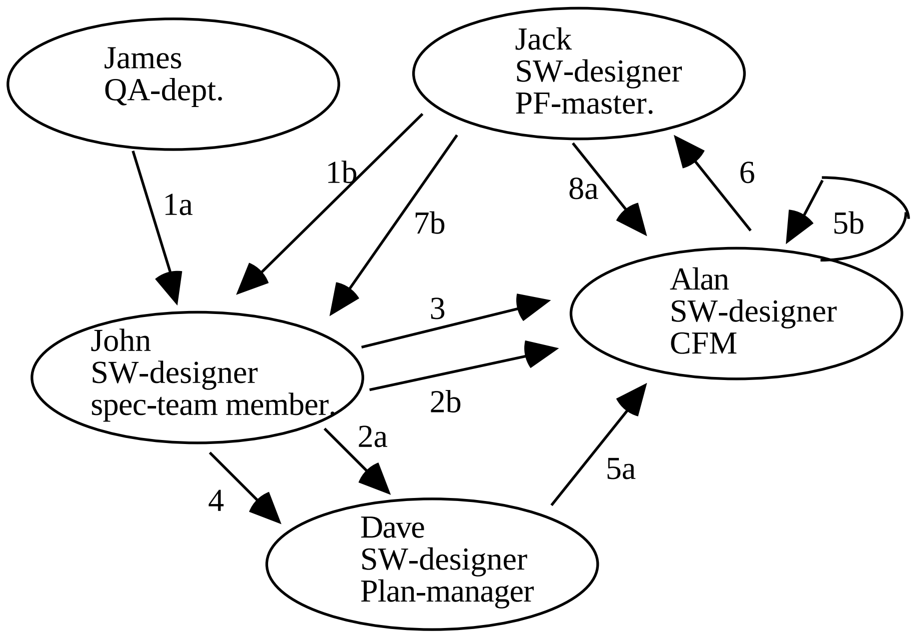


**Figure 6-10:** An example of 5 actors involved in the treatment of 2 bugs. The number of the flow corresponds to those used in figure 6-9. A detailed description is given in the body text below.

Problem b has not been solved sufficiently, and Jack decides that it should be diagnosed further. He therefore adds a note on the form, and sends it to the spec-team for further analysis (7b).

As it can be seen from the example, the actors have several roles. Jack is both tester and platform master in the example, Alan is both designer and file manager, and Dave is designer and plan manager. Thus to a certain extent, each actor  has to be aware of between his roles and the information he receives. In practice, this did not cause any problems.

## 6.10 Objects of coordination work reflected

As previously defined, a coordination mechanism contains information structures relevant for coordinating the work and for monitoring the state of affairs. These structures are 'objects of coordination work' reflecting (pointing at) structures in the actual field of work and the current cooperative work arrangement (cf. chapter 5). This section will briefly discuss, which objects of coordination work the bug report form mechanism governs.

We can consider objects of coordination work the conceptualizations of work the actors use when coordinating distributed activities. An important aspect is which "manipulations" (or operations) to make to these conceptual structures. Both the dimensions of objects of coordination work (cf. section 5.4) and the relevant operations to these dimensions are, thus, relevant to address.

The detailed analysis and conceptualization of the findings concerning the software testing and correction work indicated, that the overall dimensions of objects of coordination of the software testing and correction work contain overall conceptualizations of 1) the software structure, 2) the bugs registered, 3) the classifications of bugs and software modules, 4) work procedures applied, 5) the roles used, 6) work plans, and 7) the human resources available. These are the dimensions along which the coordination of the distributed activities on testing and correcting S4000 software is conducted. The first four can be considered conceptualizations of structures within the field of work (testing and correcting S4000 software), whereas the last three reflect the nature and implementation of the cooperative work arrangement.

The objects of coordination work can be in one of two stages, either nominal or actual (cf. section 5.4). In relation to the description of the bug form mechanism, the relevant nominal dimensions were: Roles, Tasks, Human resources, and Conceptual structures. The relevant actual dimensions were: Committed actor and State of the field of work. These will be briefly described:

The notion and use of *roles* is essential in the bug form mechanism. The description of the protocol stipulating the flow of the testing and correction work is based on a clear definition of roles. All organizational procedures and obligations are formulated in terms of *roles*.

Whenever a decision on either who is to be the next platform manager or who is responsible for correcting a bug is taken an 'assigned to' relation is made from a *task* to a *role* to a *committed actor* (cf. figure 5-3 in section 5.4), and the role is furthermore related to a *human resource*.

When a tester registers a bug, he uses a classification scheme, i.e., in terms of objects of coordination work he uses a *conceptual structure*. Additionally, when the spec-team diagnoses a bug, they relate a specific problem to a software module, and they relate a *task* to a platform period. Software modules can be regarded as conceptualization of certain aspect of the field of work, i.e., in terms of objects of coordination work it is a *conceptual structure*. Similarly, the platform periods can be considered a conceptualization of aspects of the work arrangement, and thus as a *conceptual structure* in terms of objects of coordination work.

| **Nominal** | | **Actual** | |
|---|---|---|---|
| **Dimensions of coordination work** | **Operations with respect to the dimension** | **Dimensions of coordination work** | **Operations with respect to the dimension** |
| *Dimensions with respect to the cooperative work arrangement* | | | |
| **Role**<br>- Tester<br>- Responsible designer<br>- Spec-team member<br>- Central file manager<br>- Platform master | assign to [Committed actor];<br>responsible for [Task];<br>responsible for [Human resource] | **Committed actor** | accept [Role];<br>reject [Role] |
| **Task**<br>- Correction task | allocate to [Role];<br>accept;<br>reject; | | |
| **Human resource**<br>- James, Jones ...... | allocate to [Role] | | |
| *Dimensions with respect to the field of work* | | | |
| **Conceptual structures**<br>- Bug classifications<br>- Software modulation<br>- Platform periods | categorize bug;<br>define bug classification;<br>relate to [Task];<br>classify module | **State of field of work** | make sense of; |

**Figure 6-11:** A model of the dimensions of objects of coordination work used in the bug form mechanism, and the required elemental operations lated to these dimensions. The table is based on the general table illustrated in figure 5-2.

Furthermore, when one of the designers or testers browse in the binder they access a conceptualization of the *state of the field of work.*

The relevant dimensions and the basic operations required with respect to these dimensions are illustrated in figure 6-11 above.

As it can be observed from the table above, only very few structures are placed in the actual stage. The reason for this is, that the bug form mechanism does not attempt to provide information on ongoing activities or actors in action.The purpose of the mechanism is planning and work flow stipulation rather than monitoring ongoing activities.

Another observation is, that the bug form mechanism does not have any relations to resources apart from human resources. Material, technical and infrastructural resources are not reflected at all.

## 6.11 The interrelationhips between the coordination mechanisms

As mentioned several times, the field study illustrated several mechanisms that can be considered coordination mechanisms. One of these, the bug form mechanism, has been described in detail. Other mechanisms mentioned were the platform period concept, the project plan spreadsheet, and the directory structure for software integration.

The platform period concept stipulated, how the work should be organized and which integration activities should be conducted. It, furthermore, defined roles and standard obligations for the designers during the integration periods, i.e., it stipulated how progress in the integration of the software was to be obtained.

The project plan spreadsheet provided a conceptual structure for scheduling tasks, actors, and deadlines by relating development and testing activities to relevant software modules (and thus to responsible actors) and to working periods. The sheet stipulated which components of the software should be integrated in the individual integration periods, and it provided information on which actors and modules to involve when problems, changes in plans, etc., occured.

The directory structure, and the software routines for automatically linking the modules, supported meshing the software components by stipulating how several actors’ pieces were to be integrated. Furthermore, the structure embedded in the directory can be considered a classification scheme for classifying concepts and structures within the software, i.e., the directory structure established a common picture of the software complex architecture that all designers could relate to.

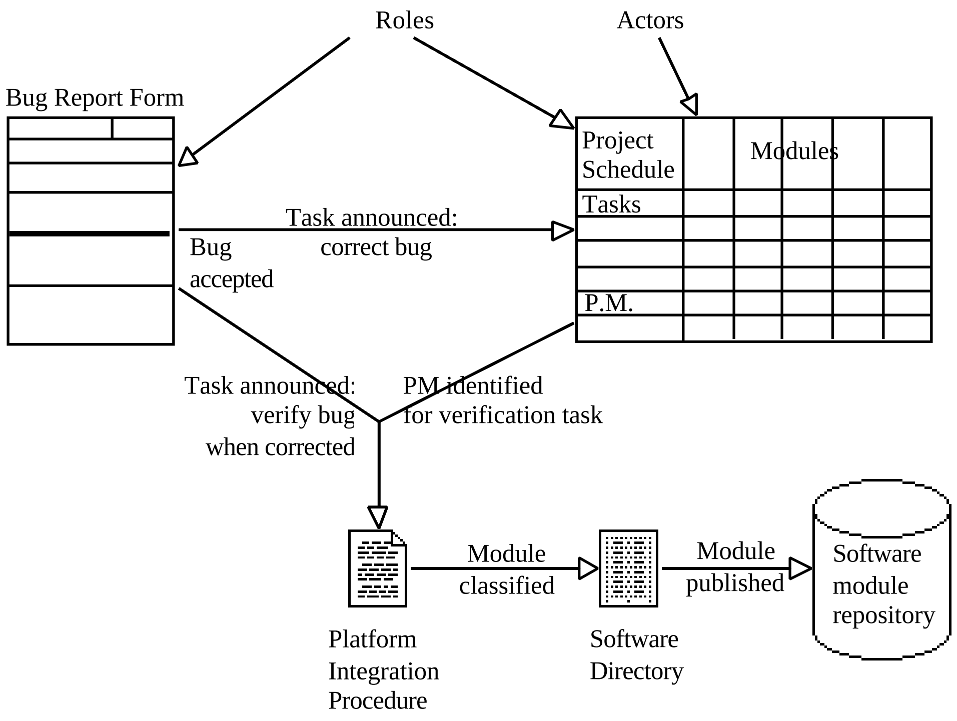

Roles
Actors
Bug Report Form
Project Schedule
Modules
Tasks
P.M.
Task announced: correct bug
Bug accepted
Task announced: verify bug when corrected
PM identified for verification task
Module classified
Module published
Software module repository
Platform Integration Procedure
Software Directory

**Figure 6-12:** The interaction between the coordination mechanisms used for coordinating the software development in the S4000 project. ‘PM’ is platform master. The figure is a reprint of figure 5-4.

Besides being individual coordination mechanisms supporting certain aspects of the coordination of the development of software for the S4000, the mechanisms were heavily interrelated. For example, when a bug was accepted, it “automatically” generated a new task to be invoked in the spread sheet. Another example of the interrelationships between the mechanisms was that, at a certain point in time (specified in one coordination mechanism—the spread sheet), a procedure specified by another coordination mechanism (the platform period concept) was initiated. This procedure included (subscribed to) information from a third coordination mechanism (the bug form mechanism), and it launched the execution of a fourth coordination mechanism (the directory mechanism) linking and compiling the software modules.

The linking between the coordination mechanisms identified in the field study is illustrated in figure 6-12 below. We provide a thorough discussion of linking between real life coordination mechanisms, and the implications with respect to design of computer-based coordination mechanisms, in Schmidt *et al.* (1994). I have, however, only considered it relevant to briefly illustrate the interrelations here.

The study illustrated several types of linking:

- a coordination mechanism subscribing to information from another.
- a coordination mechanism using information from another mechanism as an index for accessing repositories.
- a coordination mechanism triggering the execution of another mechanism when a certain condition occurs, and
- a coordination mechanism "writing" in the information included another mechanism (or requesting a writing).

Another point is worth noticing: The different links between the coordination mechanisms used in a work situation made it possible for a specific instantiation of a mechanism (e.g., a particular bug report sheet) to be incompletely defined. For example, just indicating a responsible software module and a platform period number in the bug form was sufficient since the responsible designer and the actual deadline for the correction task were "added" later by another mechanism, namely the spreadsheet.

## 6.12 The nature of the support provided

This section will briefly condense the most interesting aspects of how the coordination of the software testing and correction work in the S4000 project was organized, and how the coordination mechanism used supported this.

First, some overall characteristics of the coordination of the software testing work in the S4000 are given. This is followed by a discussion of the use of the bug form mechanism, the related coordination mechanisms, and how these supported the required coordination activities.

### 6.12.1 Characteristics of the coordination of the software testing and correction work observed

The field study has clearly indicated, that software testing and correction is a very complex and complicated task requiring a lot of coordination. In order to avoid redundant work, the testers need to be aware of each others work. The study showed, that it was very difficult for the testers to get an overview of reported errors, their diagnoses, correction status, etc. It was extremely difficult for the actors involved in the S4000 project to determine the state of affairs in the software testing at a glance. It was illustrated in the study, that it was difficult for the testers and designers to communicate about the software complex and its status at a given point in time. The state of affairs was “hidden in abstract representations”. An illustration of this was the briefing sessions ending each integration period. Here, most of the time was spend on discussing user interface details such as colors and layout. This was done, despite the fact that all designers agreed, that aspects concerning the structure and implementation of the code computing the measurement results was much more important and essential for the project. The problems of lack of “visible representations” of the relevant aspects of the software complex are similar to problems in the process of developing software reported by Parnas and Clements (1986).

A number of activities were essential for coordination purposes. The most important of these were: Allocating resources, planning and scheduling tasks, monitoring the state of affairs in the development and test process, classifying and prioritizing, distributing information, negotiating requirements, and negotiating priorities. These activities were, in the S4000 project, usually conducted by means of *ad hoc* meetings and discussions, structured meetings, and use of forms, lists, boards. This chapter has described one of these forms, and its related procedures and conventions.

The study, furthermore, illustrated that, the coordination activities within software testing and correction were mainly based upon conceptualizations of structures in the field of work (e.g., the structure of the software complex) and structures reflecting the current implementation of the cooperative work arrangement (e.g., the involved actors, the working cycles, verification procedures, etc.). The conceptualizations was, among other things, used to support the distributed bug registration activities, support the planning activities, monitoring progress in correcting the software, monitoring the state of affairs in general, and to simplify the needed bug classification and diagnosing activities.

Furthermore, aggregations of detailed information of the state of affairs (e.g., the total number of “not yet corrected category 2 bugs”) was used to support the coordination work, especially in order to simplify the required monitoring activities. Several structures for classification and categorization of bugs, corrections demands, and software modules was used. Concrete information from the software testing and development was also used, when the activities were coordinated, e.g., the software code itself, the documentation, or the content of the bug registrations was used when deciding the estimated correction time for a bug.

The most prominent basic coordination functions (or activities) conducted in relation to the conceptualizations were: classification and categorization of bugs and software, monitoring the state of affairs and progress in the processes, allocation of resources, relating resources to tasks (establishing responsibilities), meshing the resources and tasks into work plans, and negotiations on classifications, allocations, obligations, etc.

### 6.12.2 How the mechanisms supported the coordination work

A definition like the one given here, of the basic function of the bug form mechanism would probably only partly be recognized by the software designers involved in the S4000 project. It would not reflect their original intentions, or rather the original description of the problem they had recognized, when they first developed the ideas of the form. When the project had been going on for half a year, the board of directors at Foss Electric, and the S4000 project management realized, that they had no means for measuring the progress of the software development part of the project. In their search for measurement methods they hired a consultant that suggested to perform weekly measurements of how many known problems that remained to be fixed. A decreasing curve would indicate progress. This was very problematic:

> “Originally the intention was to produce statistics of the number of known-but-not-yet-fixed problems and use this as a management tool. But we realized that as a man-

agement tool this can only be used if you have a stable product. We didn't!" (Software designer at Foss Electric)

So, instead of a management tool for monitoring state of affairs in the project, the ideas were refined into a mechanism supporting decentralized registration of bugs ensuring that all registered bugs were remembered. From the field study findings and the comments from the software designers at Foss Electric, it appears to be quite clear, that the most important aspect of the bug form mechanism was, actually, that it reduced the effort required in order to handle certain aspects of the coordination of the software testing and correction process.

It can, of course, be questioned whether a mechanism genuinely "eliminates" the complexity. The coordination to be conducted is as complex as always, but it can appear simpler to the actors through improving the representations of the work domain (cf. e.g., Woods, 1988), by forcing a specific behavior of the actors involved (Norman, 1991), by automating or computer supporting certain activities of the coordination (Malone and Crowston, 1990; Schmidt and Bannon, 1992), or by establishing a division of labor minimizing the need for coordination (cf. e.g., Mintzberg, 1983). The claim here is, that the bug form mechanism supported the coordination of software testing and correction in the S4000 project by providing several of these things:

- The bug form provided a standardized information structure by which all bugs were described. By allowing information to be used for the diagnosis to be included in the form in a standardized prestructured manner, the mechanism made it easier for the spec-team to find the relevant information. The classification of the bugs made it easier for the spec-team to deduce the testers perception of the problem reported. The form can thus be seen as improving the representation of the field of work (the bugs) by establishing a "common standardized language". This makes it easier for the actors to interact. "Standardized languages" are, of course, problematic too. They constrain the actors, and it takes time for the actors to become familiar with them, and they need to be maintained.
- The stardardized format of the form, furthermore, supported the work of reporting, both from tester to spec-team, from spec-team to designer, and from designer to platform master. This is because it forces a specific behavior on the actors. Through a specific surface

representation (Norman, 1991), the form supported the actors: No effort was required for considering which information to include.

- The bug form mechanism also supported the coordination activities by stipulating the work flow for how to handle the reporting, diagnosing, and correcting process. Although it was not completely automated, the pre-specified flow excluded the need for communication and interaction among the actors when handing over the form (and thus the obligations) from one actor to the next. The pre-specified flow (the embedded protocol) afforded support through constraining the actor: He could just apply the pre-specified routing without further considerations.
- Finally, the bug form mechanism was a central tool in the attempt to establish a well-understood and well-defined division of labor. By establishing different roles, and very clearly defining their responsibilities, the mechanism reduced the need for communication and interaction among the actors. All actors in the reporting, diagnosing, and correcting process knew exactely, what their obligations were. When they had dealt with their part of the treatment, they could just pass on the form, and others would take care of the rest. Their need for coordination was reduced.

Coordination activities like monitoring the state of affairs, and negotiation of classifications, allocations, etc. were attempted to be supported by the bug form mechanism too. The establishment of the central file (the binder) including all registered bugs and their current status made it easier to get a coherent picture of the state of affairs. Although the testers and designers found it difficult to achieve an overview from the content of the binder, the benefit was that they only had to search in one place. Furthermore, the standardized information structure in the forms, and the standardized index of the binder (cf. figure 6-6), made it easier for the actors to find the relevant information, for example the classification and status of a bug or the number of not-yet-corrected category 2 bugs. Regarding negotiation of bug classification or resource allocation the bug form mechanism, and its related mechanisms, made classifications and resource allocations visible and accessible to the actors. The classifications and allocations became easier to discuss.

Based on findings in the study, and discussions I have had with the testers and designers at Foss Electric, I will claim, that it would be more or less impossible to handle distributed testing and bug registration involving approximately 20 testers without having a mechanism providing an overall functionality similar to the bug form mechanism. It would require an enormous amount of formal and informal meetings to coordinate all activities and keep all actors up to date.

The purpose of the bug form and the related procedures and conventions was first of all to ensure: That all problems were registered, that responsibilities and obligations were clearly defined and visible to all involved designers, and that the work flow was clearly stipulated. Furthermore, the intention was, that the binder should provide the involved designers and testers with an overview of the state of affairs in the software complex, and support the designers in being aware of activities in modules of which they were not responsible, but to which their modules might have close interaction. Most of these purposes were fulfilled successfully, although the two latter purposes of providing awareness of state of the affairs were only partly fulfilled. Most of the designers found it to complicated to browse the forms in the binder.

Other coordination mechanisms were used in interaction with the bug form mechanism. Regarding how these coordination mechanisms supported the coordination of the S4000 software development we can briefly conclude the following:

- *The platform period concept* aimed at ensuring coherence in the project: All actors had a common understanding of the work and the direction for progress. This was done by establishing specific points in time, and at these points “force” the actors to establish common basis for future activities. This simple idea was recognized as the best solution to an overwhelming problem for the software designers. The concept of platform periods supported the coordination of the software development and testing work by establishing standardized structures, and by establishing a forum for being mutually aware of each others work. More efficient approaches might exist, but the concept proved to be so successful, that it has been developed into a company standard for organizing development projects in the future.

The platform period concept can thus be seen as supporting coordination work by partly automating (stipulating) the flow of the work. It entrenched the division of labor through detailed specifications of the obligations of the different involved actors (roles). The concept, furthermore, supported the actors in monitoring the state of affairs by providing a standardized forum for being aware of what went on and for discussing and negotiating how to proceed. An example of this was the briefing meeting held at the end of each integration period.

- *The project plan spreadsheet* supported the coordination of the software development and testing work by providing a conceptual structure scheduling tasks, actors, and deadlines by relating development activities to relevant software modules and responsible actors. This stipulated how progress in the integration of the software should be obtained, and which actors and modules that should be involved when problems, changes in plans, etc. occurred. The spreadsheet provided a standardized conceptualization of some of the central structures to be related. This standardization facilitated the communication and *ad hoc* coordination by providing a "common language". Furthermore, it supported the actors in being aware of the state of affairs and future tasks, and it facilitated the frequently ongoing negotiations on allocation and reallocation of resources.

  The project plan spreadsheet thus provided a standardized representation of aspects of both the field of work, the work arrangement, and the relationships of these structures. This format can be seen as forcing a specific behavior on the actors dealing with resource allocation, and it, furthermore, made the state of affairs (tasks to be conducted) visible and accessible to all the involved actors.
- *The directory structures,* and the related software routines, reduced the complexity of the meshing of the software by providing a structure stipulating how several actors' pieces were to be integrated. The directory structure, hence, supported distributed software development by providing a classification scheme for classifying concepts

and structures in the software, i.e., the structure established a common standardized conceptualization of the software architecture. All designers could relate to this when communicating and coordinating their activities. In addition to enabling distributed storage and retrieval, the directory structure also provided a set of automated procedures for meshing the elements. Only situations where the architecture of the software complex needed to be updated, required negotiation of the directory structure. In terms of coordination work, the directory mechanism automated some of the integration activities, and established a standardized structure facilitating the integration, and the negotiation of the integration.

The directory structures and the related routines can be seen as an automation of certain tasks related to integrate the software complex. It, furthermore, enforced a standardized structure of the most essential aspects of the field of work, namely the software architecture, i.e., it settled the basic structure of the software which all involved actors had to relate to in their work.

The analysis presented in this chapter has been descriptive only. Reflective and constructive use of the findings and experiences will be presented in the following.

# 7 . The Coordination Mechanisms Concept as a means for analyzing cooperative work

“He who does not lay his foundations before-
hand may by great abilities do so afterwards,
although with great trouble to the architect and
danger to the building”
(Machiavelli, 1514)

The two previous chapters have introduced the Concept of Coordination Mechanisms and applied the concept for analyzing the coordination of software testing and correction in the S4000 project at Foss Electric. Based on the analysis a natural next step towards computer support is to switch into a more constructive approach and formulating requirements and design sketches for a computer-based system. Such an approach will be taken in part III (chapters 8, 9, 10 and 11). I will, however, first contrive some reflections on the usability of the Coordination Mechanisms concept as a means for analyzing, and aiming at understanding, the coordination aspects of a complex cooperative work setting. This will be done in four steps:

1) I have previously conducted a critical review of a number of existing analysis methodologies within the software engineering and office automation traditions (Carstensen and Schmidt, 1993a). First, a brief summary of the conclusions from this will be provided.
2) Although the field study applied a specific framework it can, of course, also be seen as ‘normal’ ethnographically inspired field study (or ‘workplace study’) undertaken with the intention of informing a systems design process (cf. e.g., Hughes, 1993; Plowman *et al.*, 1995). Step two discusses what the study has indicated with respect to overall requirements for methodologies and conceptual frameworks for analyzing cooperative work.
3) Since the Concept of Coordination Mechanisms has been applied in order to analyze the findings, a discussion of the usability of

the concept as a means for analyzing coordination of cooperative work will be provided. The discussion focuses on, what has been gained from using the concept, and what has been left out.

4) Originally a primary purpose of the field study was to provide input for improving, changing, and refining the Concept of Coordination Mechanisms. Much of the lessons learned from the field study have already been implemented in the conceptual framework as it is presented in chapter 5. The last section discusses conclusions regarding the conceptual framework drawn from the field study, and which aspects of the conceptual framework need further development and refinement.

## 7.1 Overall requirements for methodologies and conceptual frameworks for analyzing cooperative work.

The objective of analyzing a cooperative work setting is to provide input for improving a given situation by designing and implementing information systems, by redesigning work, by recommending a retraining program, etc. The basic approach will thus be oriented towards changes or possible “treatment”. The analyst investigates a particular social system of work in order to suggest changes. Accordingly, the analysis cannot take the current behavior of the social system of the cooperative work arrangement for granted. On the contrary, the analyst must uncover the hidden rationale of current practices as well as the accidental choices of the past, the procedures turned rituals, the formalized mistakes, etc. (Schmidt, 1990; Schmidt, 1991b; Carstensen and Schmidt, 1993a). Focus will be on questions such as:

- What is the rationale behind current practice?
- Which aspects of current practices can be interpreted as incidental adaptations?
- Which aspects of current practices are essential and necessary to meet current and future requirements?
- What could be done better, more efficiently, more satisfactory to the actors, and how?
- How could information technology improve the efficiency, effectiveness, actor satisfaction etc.?

The analyst must attempt to unravel the social system of the cooperative work arrangement. While ‘taking apart’ current practices, the analyst has to ‘decipher’ the extensive and shifting networks of cooperative relations ingrained with current practice and identify the rationale of the various patterns of cooperation (Schmidt, 1990). It is, both from my study and other reported studies, quite clear that analysis of complex, cooperative work settings is a hard and—seen from the analyst’s point of view—a never ending activity.

The work settings to be analyzed are normally immensely complex, and it is usually difficult to separate the relevant information, requirements, mechanisms etc. from the less relevant. What are in fact the criterias for defining what is relevant? These complications for the analyst poses some extremely knotty methodological problems. Some of these arise from the fact that the cooperative work arrangement is a complex phenomenon involving multitude forms of social interaction (e.g., within the work process itself, within the organizational setting, or within the social control). Other complications arise from addressing the structure and the function of the work system approached (e.g., the function of the enterprise, or the structure of the market purchasing the products provided).

A first overall requirement then is systematic analytical distinctions overcoming the confusion of the concepts of cooperative work, division of labour, organization, allocation of tasks and responsibilities, profession, management strategy, collaborative styles, labour market structures, class, ideology, etc. This is similar to Strauss’ argumentation (cf. Strauss, 1985; Strauss, 1988).

As suggested ealier (cf. Carstensen and Schmidt, 1993a) it might be useful to make a basic distinction between: 1) the social system of work as a functional system and 2) the social system of work as an arena for human actors. The function oriented perspective approaches the social system of work as a functional system of cooperative relations, i.e., focus is on the social system of work as an instrument meeting the functional requirements posed by the environment. The approach is similar to what others have called “a rational systems perspective” where the behaviour is viewed as actions performed by purposeful and coordinated agents (Scott, 1987). The arena for human actors perspective regards of the social system of work as a system of more or less collaborative arrangements between multiple individuals with diverging interests and motives. This is similar to what Scott calls “a natural system perspective” (Ibid.).

A second overall requirement for a methodology is support of separation and analytical distinction in conceptualizing the cooperative work arrangement as a working system and a social organization. Being a bit more operational we can list a number of more concrete requirements:

- The work arrangement must be seen as a dynamic whole—as a system. All relevant perspectives should be addressed, analyzed, and related to each other. So, at least three different approaches are required, addressing: 1) Characteristics of the outer environment and the relations between the environment and the cooperative work arrangement; 2) The field of work; and 3) The characteristics of the actual cooperative work arrangement itself, i.e., the existing implementation of a socio-technical system.
- The relations to the outer environment—the abstract functions offered by the work arrangement to the environment and the objectives of the work arrangement (cf. Schmidt and Carstensen, 1990)—need to be identified. Conditions, constraints, requirements, etc. under which the functions must be accomplished and characteristics such as time constraints, risks, success acceptance criteria, stability and dynamism in the environment, the frequency of changes in requirements and constraints, the level of interrelations in the requests, etc. should be identified and described.
- The essential characteristics of the field of work should be identified and described in terms of goals, conditions, constraints, sets of rules, heuristics, possible strategies, etc. The complexity should be described in at least the dimensions of dynamism, interacting components and processes, and uncertainty.
- Prototypical activities need to be identified and it should be addressed to which extent the activities reflect characteristics in the field of work or in the environment and to which extents they are caused by the current implementation of the work arrangement. A distinction between the social and the work system should, furthermore, be included.

The overall requirements for work analysis methodologies and conceptual frameworks described above have mainly been derived through a critique of existing methodologies within the software engineering and office automation fields (Schmidt and Carstensen, 1990; Carstensen and Schmidt, 1993b; Carstensen and Schmidt, 1993a).

The software engineering methodologies (e.g., Sutcliffe, 1988; Yourdon, 1989; Mathiassen *et al.*, 1993) presuppose the human-system boundary to be known beforehand, that the field of work can be described exhaustively and unambiguously as a three-dimensional system of data-structures, -processing, and -flow, and that the work can be decomposed into a partial tree structure of procedural processes. The methodologies can only rarely express abstractions or specializations in the conceptual models, and there are no possibilities for separating procedures from routines or algorithms expressing the “correct” logical method (cf. Floyd, 1986). Furthermore, there are no techniques for modeling means/ends relations, causal relation, control structures, structures for mediating co-ordination work, etc. These aspects are only implicitly represented in the models as ‘hidden’ aspects.

The approaches from the office automation tradition (e.g., Zisman, 1977; Ellis, 1979; Sirbu *et al.*, 1981) are also problematic. These provide richer notations for modeling the work. The problem with the procedure descriptions is, however, that only activities and situations that can be described procedurally are covered. The activities are modeled and described, but left un-interpreted in the models and descriptions, and non-procedural decision situations are only implicitly represented as terminators. If exceptions are ‘the norm’ the approaches will fail to produce any usable information. All office automation approaches seem to exclude the issues of unanticipated decision making and problem solving situations.

To conclude this brief review of existing analysis methodologies and frameworks from the software engineering and office automation fields, we can state that, the world modeled is an overly simplified conceptual world when using traditional office automation and software engineering methodologies[12].

12 I have earlier termed this as “The Procrustes Paradigm” (Carstensen and Schmidt, 1993a). In the Greek mythology, Procrustes was known for his own very special method of torture, namely that fitting victims to a bed by cutting off the legs of those too tall to fit or racking those too short.

## 7.2 Requirements for analysis methodologies and frameworks derived from the field study

It is also relevant to provide some reflections based on the experiences from the field study reported on here. Through the process of analyzing the field study I have obtained input for, what a conceptual framework must provide in order to support a 'relevant analysis' of a complex cooperative work setting.

I have elsewhere reported my first conclusions regarding the use of a conceptual framework for analyzing a complex cooperative work setting (Carstensen, 1995b). These can be structured in the following bullets:

- A conceptual framework must be based on an analytical distinction between work and coordination work. The explicit distinction provided by the Coordination Mechanisms Concept proved to be very useful. The fact that cooperative work and its coordination are closely related and intertwined must, of course, be reflected, but an analytical distinction is important. It is the aspects regarding the coordination of the cooperative work that are central. Although we, when designing computer-based systems, need analysis methods that grasps the richness of the work, we also need an approach that restrict aspects not addressing the coordination of work. This is also essential when considering how applications supporting 'conventional' work can be augmented with coordination facilities.
- It must be the nature of the work that constitutes the cooperative work arrangement to be addressed. It is the actors that are mutually interdependent in their work that must be addressed, i.e., actors that are actually working in cooperation, not necessarily placed in the same organizational unit. When conducting our analysis, it made more sense to let the field of work define who and what to address. Otherwise, the actors in the service department would have been considered outside the work setting although they were heavily involved in the software testing and thus closely collaborating with the software designers.

The approach of just cutting (or adding) bits, pieces and slices of your object (e.g., the cooperative work arrangement) until it fits into your model of the world has quite a lot in common with several of the approaches discussed there.

- The coordination of the work studied was first of all based on abstractions and conceptualizations of structures in the field of work and in the work arrangement, e.g., bug classifications, the software architecture, work plans, etc. These structures could be regarded as dimensions of objects of coordination work in the terms of the Concept of Coordination Mechanisms (cf. chapter 5). A conceptual framework should provide structures supporting the analysts in identifying and characterizing the conceptual structures, categories, classifications, etc. used when coordinating. In relation to this the framework should also support identification of relevant functions (or manipulations) on these conceptual structures.

  Conceptualizations and categorizations contain a hidden agenda of discipline and control which needs to be explicitly adressed (Suchman, 1994). However, in order to support the design of computer systems, conceptualizations are required.
- Much of the coordination activities observed at Foss Electric were supported by conventions, procedures, artifacts, etc. A functional approach to such mechanisms used to support the coordination of the work should be included in the framework. ‘Functional’ in the sense, that it must address what function the mechanism serve, for what purpose, and under which conditions the function must be served. This is essential in order to identify candidates for computer-based coordination mechanisms.
- Much of the coordination activities observed in the study were conducted by means of meetings. A functional approach to such meetings might prove useful too. The framework must support the analyst in understanding what function the meeting serves regarding coordination. Certain aspects of meetings will often be candidates for computer-based support, for example in terms of coordination mechanisms or through improved communication support.
- The actors used several mechanisms more or less concurrently to support the coordination activities. These mechanisms were related to, and mutually dependent of, each other. In order to understand how the mechanisms are interrelated we need to address both the structural and dynamic properties of these mechanisms. A conceptual framework should provide techniques for grasping and model-

ing both the structural properties of the mechanisms and the dynamics (and concurrency) of the interaction between the mechanisms.

- In very complex cooperative work settings, certain aspects of cooperation might simply have been avoided due to the complexity of the coordination of the work. The software designers, for example, gave up handling the coordination of the software integration in a distributed, concurrent manner. Instead, they introduced the non-parallel platform period concept. A conceptual framework should support the identification of such situations, i.e., 'see' possible reorganizations of the work that will require coordination, that do not exist today or have been given up.
- The cooperative work studied was carried out over a long time-span. There were often long periods in which the actors had none or very little interaction. They were, however, still mutually interdependent. Hence, the approach and framework to be applied cannot be mainly based on a registration of observable phenomena. Techniques based on direct interaction with the involved actors should be provided too.

The statements listed above are, of course, findings based on one study only, and thus not an exhaustive list.

## 7.3 Advantages and disadvantages of using the Concept of Coordination Mechanisms

Both the previous section on requirements for analysis methodologies and the following (section 7.4) can be regarded as reflections on the usability of the Coordination Mechanisms Concept as a means for analyzing coordination of cooperative work. This section will attempt to more concretely illustrate the advantages and disadvantages of using the concept to support the study, and as a structure for conceptualizing the findings.

Apart from a definition of a coordination mechanism as a "protocol, encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, that stipulates and mediates the coordination of distributed activities", the basic conceptualizations provided by the Coordination Mechanisms Concepts are:

(1) The recursive "work"—"coordination work" distinction;

(2) The “field of work”—”work arrangement” relation;

(3) An identification of some overall modes of interaction among the actors, and prototypical interactional activities;

(4) A ‘continuum of the rigidness’ of the modes of interaction used;

(5) A set of salient dimensions of ‘objects of coordination work’, both with respect to the work arrangement and the field of work, and with respect to its state (nominal or actual); and

(6) A set of elemental operations on the dimensions of objects of coordination work required.

The concept furthermore includes a set of overall statements on which facilities a coordination mechanism should provide.

The analytical distinction between work and coordination work was very useful. It helped explicitly addressing the coordination aspects of the work. It turned out not to be difficult to distinguish when the testers and designers at Foss Electric were conducting actual test, design, implementation, corrrection, or verification work from when they were conducting coordination activities. When observing situations, or making interviews, the information I collected contained, of course, both apsects of work and of coordination work. But, when analyzing the data it was quite easy to make the separations. It is, however, important to notice, that I conducted the study within a field which I am familiar with. It might have been more complicated to handle the analytical distinction within a domain less familiar.

The explicit distinction between work and coordination work also demonstrated usefulness at a seminar at Foss Electric. We[13] presented our findings to approximately 30 designers, project managers, company managers, etc. Several of the participants stated that only very little of what we told them was new to them. But they had never thought of it this way before, and they felt that it immediately provoked considerations on how to reorganize some of their work structures. Examples were using CEDAC board for software testing activities (we have discussed the CEDAC board in details in Borstrøm *et al.*, 1995), and respecifying the role of the spec-team.

13 The presentation was given in collaboration with Henrik Borstrøm (DTU) and Carsten Sørensen (Risø). Carsten and Henrik conducted field studies at Foss Electric in the S4000 project similar to mine within the areas of mechanical design and process planning.

The explicit distinction between the work and its coordination supported a more thorough understanding of the coordination aspects, since the purpose of a number of 'just overhead activities' were explicitly addressed. For example the negotiations of a bug classification, or the allocations of resources for correcting a problem. The recursive definition of the work-coordination work distinction made it easier to understand the work conducted by the spec-team. If I had addressed their work as diagnosing only, a lot of their work would have appeared irrelevant for the software testing and correction. However, when I addressed their field of work to be resource allocation and work scheduling the purpose of many of their activities were easier to grasp.

The disadvantage of distinguishing between "work" and "coordination work" distinction was, that aspects of doing the actual work (testing and correcting software) were not thoroughly addressed. I focused on the coordination, and if the question is, for example, how to support the testers in conducting the actual testing, my analysis will prove insufficient. An analysis of coordination work, as the one I have conducted, must thus be combined with a more traditional work analysis.

This leads on to the "field of work"—"work arrangement" relation. Again, I will claim that this distinction proved useful. It made it quite easy to identify the relevant work arrangement although the arrangement consisted of actors within several different departments at Foss Electric. Since the analysis focused on coordination of cooperative activities, it was important to identify the cooperating actors regardless of the organizational structure. Investigating the testing and correction work clearly indicated the mutual interdependencies among the actors even though theses were in different parts of the organization. For example the spec-team (in the S4000 group) could not do the diagnosis work without indications and specifications of the importance of a specific functionality provided by actors from the marketing department.

The disadvantage of the "field of work"—"work arrangement" relation is that activities made across work arrangement boundaries are left out. For example there was an ongoing activity with the purpose of standardizing the style of commenting software code. If this should have been addressed, the field of work approached should have included (or rather explicitly been) the work on standardizing the commenting style.

The specified modes of interactions (cf. e.g., Schmidt, 1994c) and the continuum of rigidness of the modes applied were only implicitly used in the analysis. The set of modes of interaction were used as a sort of check-list and source of inspiration for where to look and what to look for during the analysis. When observing attempts to be aware of the activities of other actors, situations of directing the attention of an actor to certain aspects, handing over task or reponsibilities, etc. these were naturally considered candidates to be investigated further. Examples were, of course, the use of the binder and the bug forms, and the resource allocation activities. Being aware of the continuum of modes of interaction provoked discussions on costs and benefits of replacing certain modes and mechanisms of interaction with others. A problem here was, that the conceptual framework does not have any guidance for how to compare different modes and mechanisms, and how to ensure that all aspects relevant to a specific mode of interaction have been taken into consideration.

The idea of addressing the dimensions of objects of coordination work was very useful since the coordination of the testing and correction work studied was heavily based upon abstractions and conceptualizations of structures in the field of work and in the work arrangement, e.g., bug classifications, the software architecture, work plans, etc. The support of the data collection process was not particularly good, but it appeared to be a severely useful support for conceptualizing the findings during the analysis process. Although the idea was very useful, the first version of the dimensions was not sufficiently mature, i.e., it was not ready to support an analysis like the one conducted. This will be discussed further in section 7.4 below.

After refining the list of dimensions into the one described in section 5.4 (and illustrated in figure 5-2) it was quite easy to conceptualize the structures required for coordinating the software testing and correction in relatively few dimension. Conceptualizing the coordination work into structures concerning roles, committed actors, tasks, human resources, conceptual structures, and state of the field of work (cf. figure 6-10) provided a simple, but very expressive and coherent view of the activities analyzed. It did, furthermore, deliver a good basis for establishing requirements for computer-based coordination mechanisms, cf. following chapter (chapter 8) in this dissertation and Carstensen *et al.* (1995c). It should, however, be noticed that it is uncertain whether the dimensions of objects of coordination work will prove sufficient for analyzing a work domain having basic characteristics very different from the one I have studied, for example with heavy constrains in relation to safety or time. Further investigations are required to answer this question.

The elemental operations related to the objects of coordination work were not useful. The basic operations seem to need further refinement in order to serve as a guideline, checklist, or inspiration for an analyst. However, just by forcing the analyst to consider which basic operations the actors conducted in relation to the object dimensions mentioned above, the conceptual framework provide implicit support of the analysis process.

The most useful aspect of the Concept of Coordination Mechanisms was without doubt the protocol approach by means of which all artifacts (forms, boards, procedures, etc.) supporting the coordination was addressed. Viewing a form as a protocol, encompassing a set of explicit conventions and prescribed procedures was useful and provided a good understanding of how coordination work was supported and what the overall function of the mechanism was (cf. the analysis in chapter 6). Aspects regarding which coordination information was mediated, how, and to whom it was made explicit, and aspects regarding the stipulation of the work (or its coordination) could be unraveled by “following” the life of instances of the mechanism. Discussions on possible and relevant deviations could the easily be taken with the actors. The protocol approach supported the requirement (listed in section 7.2) of a functional approach to the conventions, procedures, forms, boards, etc. observed.

Another frequently used means for coordination was meetings. To grasp the function of the meetings observed the protocol approach proved insufficient. The meetings often had several (non related) purposes, and could serve as coordination of many different activities. The same goes for support of addressing coordination needs in situations where cooperation had been avoided because of the complexity of the coordination of the work. There were certainly situations like this in the work studied, but the Concept of Coordination Mechanisms did not provide any support for identifying these.

Based on the experiences achieved so far in using a framework like the Concept of Coordination Mechanisms, it can be concluded that the overall impression is positive. The framework provided useful support for structuring and conceptualizing the findings regarding the coordination of the work, but it must be combined with other approaches addressing the actual work conducted.

## 7.4 Improvements of the conceptual framework

As indicated above the field study has been used as input for a process refining the Concept of Coordination Mechanisms. Some of these are already reflected in the conceptual framework as it is presented in chapter 5. The most important of these will be introduced in this section. Furthermore, other improvements will briefly be suggested. These should be regarded as preliminary suggestions for future research, i.e., ideas that has been considered relevant, but out of scope of this dissertation.

The first and most essential problem was that the original definition of a coordination mechanism proved to be much too restricted. In the first version of the concept, a coordination mechanism was defined as an artifact that actively stipulated and mediated the coordination of distributed activities of large cooperative ensembles (Schmidt *et al.*, 1993). This initial definition proved to create certain problems. In fact, when applying this definition to the various artifacts used for supporting the coordination work in the cooperative work setting studied, none of the artifacts qualified as a coordination mechanism according to the definition. The initial definition presumed a specific allocation of functionality between the human actor and the artifact, in the form of activeness on the part of the artifact, that can only be realized by computer-based coordination mechanisms. In the real life work situation studied at Foss Electric, all the mechanisms supporting coordination work were non-active artifacts. Rather, they were forms, schemes, boards, or directory structures containing information relevant for the coordination of the work coupled with a set of related conventions and procedures for its use. The artifacts themselves were thus passive information containers, the stipulation was specified by the conventions and procedures, and it was the actors who were active in making things happen, usually by following the procedures. Hence, a broader definition of a coordination mechanism was needed, or rather a two-leveled definition: An open-minded one to be used for identifying candidates for coordination mechanisms in existing work settings, and a more restricted one to be used when specifying the characteristics of an active (computer-based) mechanism. The definition of a coordination mechanism was changed to the one introduced in chapter 5.

With respect to the dimensions of objects of coordination work the first tentative conceptual framework required several refinements too:

The first version had only an actor dimension, not a dimension of roles. It was, however, quite clear from the field study that the actor dimension needed to be separated into two dimensions: One of ‘actors’ referring to the actual actors involved in the work, and one of ‘roles’ referring to the role an actor has in relation to a given task or activity. In the S4000 project, all the software designers had several different roles which was important to explicitly address when coordinating the distributed activities.

The first model had, furthermore, only one dimension of time, or rather of status, i.e., there were no distinction between actual and nominal status of the objects of coordination. It provided, for example, no distinction between role and committed actor, and tasks and activities were considered as being on the same conceptual level (cf. section 4.3.3). Findings from the study of the software testing and correction work illustrated the need for a separation along the dimension of status. Some tasks were planned along the dimension of resources available in the period where the task must be accomplished, and at any given point in time some activities using named resources were ongoing. These required coordination. For monitoring purposes historical information should be available as well, i.e., the actors should be able to backtrack what had happened with respect to the status dimension. This observation called for a three layered set of dimensions of objects of coordination work: 1) planned or potential, 2) present or ongoing, and 3) past. Since past could be regarded as just a “logging of present” only two were considered required. In order to clarify the different statuses of resources these were developed into the model of the dimensions of objects of coordination work as a distinction between the statuses of ‘nominal’ and ‘actual’ (cf. figure 5-2).

Based on the experiences from the field study, a third comment to the first version of the dimensions of objects of coordination work concerned responsibility. These were represented as a dimension of objects in itself. The field study illustrated that responsibilities are always modeled as a relation between two or more of the other dimensions, typically between task and actor. This called for two changes: Responsibilities should not be modeled as an independent dimension of objects of coordination work, and it appeared relevant to establish a general model for how instances of the different dimensions are related to each other. The resulting model of how the objects of coordination work are interrelated is illustrated in figure 5-3.

The last thing changed in relation to the dimensions of objects of coordination work concerned the material, informational, technical, and infrastructural resources. These were modeled as independent dimensions related to the cooperative work arrangement. The study indicated that it would make more sense to relate these to the field of work. This was therefore done as ‘resources’ in nominal status, and as ‘resources-in-action’ in actual status.

So far, this section has described changes to the conceptual framework that has already been implemented. The field study—and a review of what other researchers have reported from similar studies—have, of course, initiated other suggestions for improvements of the conceptual framework. The most important of these are:

- The aspect of time needs to be reflected in the conceptual framework. As it is organized now time aspects of a coordination mechanism is not modeled, i.e., it is not approached explicitly in the analysis. In many work settings, it is a requirement that the coordination is handled in real time (in for example air traffic control). This imposes a series of requirements and specific characteristics on the interaction modes and coordination mechanisms used. These special characteristics of, and the time related demands on, the coordination activities and mechanisms should be reflected in the framework.
- Similar to the time dimension, a dimension of space should be better reflected in the Concept of Coordination Mechanism. Much coordination work is done with reference to certain aspects of the space in which the work is conducted. Characteristics of the space and specifications of how the space influences the coordination activities should be grasped by a conceptual framework for analyzing coordination work.
- As mentioned earlier in this chapter, the basic elemental operations was not very useful as a list of candidates for elemental operations of coordination work conducted at Foss Electric. It only functioned

> as an inspiration. A more coherent list of operations, and a more detailed description of how they manipulate the objects of coordination work, would be useful.

We are—as work analysts—basically interested in understanding the important aspects of coordination work to an extent that is sufficient for informing a design of computer-based support of coordination work activities[14]. Doing this requires at least a good conceptualization of the structures used and the operations and actions required when coordinating the distributed work activities. Other kinds of support are required too:

The conceptual framework does not provide any methodological support, i.e., there is no line of action suggested, and it is not (yet) clear which techniques it would be relevant to use when collecting and analyzing data.

An analysis of the coordination activities within a work setting must be combined with a more traditional work analysis. Work and its coordination is closely related and intertwined. Analysis methodologies must therefore include techniques and conceptual frameworks addressing both aspects of the work.

As analysts we, furthermore, need support for 'an overall abstraction' of the coordination activities addressed. As it is now, there are very little support for understanding what is the function (in terms of coordination) of the meetings observed, or which types of work organization have been avoided in the existing implementation of the work arrangement due to lack of means for handling the coordination required. Means for comparing costs and benefits from applying different modes and mechanisms of interaction in a specific work situation are missing too.

The Concept of Coordination Mechanisms seems to be a good first candidate for providing the required conceptualizations of the structures used when coordinating distributed work activities. Several important aspects are, however, still missing, and there is a big need for methodological support. A lot of further (future) research is called for. As for conducting work studies and work analyses, establishment and refinement of methodological support for this seems to be an open-ended never ending process.

14 Some people—within different research traditions—might disagree with this statement. But it is a basic assumption for the approach I have taken in this dissertation (cf. chapter 1).

# Part III: Towards computer support

# 8. Requirements for computer-based support of the coordination of software testing

"Systems development is a creative activity, and many systems developers see themselves as professional individuals, conscious and proud of their personal working style. The motivation and drive of many systems developers are founded in their practice as creative individuals. Systems development is, however, at the same time a highly cooperative and complex activity. Effective coordination is essential and the limited intellectual capacity of individual systems developer has to be dealt with explicitly."
(Dahlbom and Mathiassen, 1993, p. 163)

The previous chapters have discussed conceptualization of coordination work, and approached the findings from the field study at Foss Electric analytically. A next step is thus, to take a construction oriented approach.

A construction oriented approach could start out by first establishing a set of requirements. The aim of this chapter is to illustrate how requirements for a specific computer-based coordination mechanism supporting a specific work setting could be established and expressed concisely. The area chosen is, quite naturally, the coordination of the software testing and correction process observed at Foss Electric. The aim of establishing the requirements was originally to provide input for refining the Concept of Coordination Mechanisms, and gaining experience regarding using the conceptual framework for supporting requirements specification. This chapter should, therefore, not be read as a documentation of a software-development process. The central approach has not been to come up with new innovations, but to provide an exemplification of requirements for a computer-based coordination mechanism. In real life settings it would, in addition, be essential to address problems such as: How to identify candidates for computer-based coordination mechanisms? How to draw the boundary of the computer systems? How to avoid simple automation of the existing mechanism? etc.

As illustrated in section 6.1 software testing is a complicated task, which, in practice, is impossible to accomplish exhaustively (Myers, 1979; Parnas, 1985). Much effort is required to coordinate the activities, negotiate software acceptance criteria for usability, reliability, capability, etc., and to establish consensus of when the software is acceptable. In large scale software projects, like the S4000 project, many actors with different perspectives and different areas of competence are involved in the testing process. Plans and procedures describing who is testing what, classifying errors, reporting back, handling the corrections, re-testing, etc. are needed. Much research has been conducted in the software testing field (Gelperin and Hetzel, 1988), but very little of this research has addressed the organizational process and cooperative aspects of software testing. An attempt within the field of CSCW to address the cooperative aspects of software testing is the CSRS system supporting collaborative software review (Johnson and Tjahjono, 1993). The idea was to apply an existing hypertext-based environment for building systems to keep track of the decisions taken in a group, on the field of software inspection. Another example is provided by Mashayekhi *et al.* (1993) who discuss how to support distributed cooperative software inspections.

This chapter formulate a set of requirements for a computer-based coordination mechanism supporting the coordination work involved in registering, diagnosing, and correcting software bugs. The requirements are derived from the findings of the field study and from the general requirements defined for coordination mechanisms (cf. section 5.5). Focus has been on setting up a set of requirements for a mechanism that, in a more active manner, stipulates the required work flow, and mediates the needed information among the involved actors. To do this, the requirements address two central aspects: Which conceptualizations of the field of work and the work arrangements should be provided, and which facilities are needed for stipulating the central work flow, in the aspects of coordinating a process of registering, diagnosing, and correcting software bugs. It is not claimed that tools supporting the coordination of software testing should be seen as isolated tools. They should be integrated with other tools for software testing. For the purpose of illustration, this integration has, however, been considered outside the scope here.

Requirements for how the interaction between the mechanims and the actors (users) can run, what the data structures embedded in the mechanism should contain, what browsing facilities should be provided, and which other computer-based mechanisms the mechanism could link to are discussed, both in terms of overall requirements and in terms of more design oriented aspects.

Based on the findings from the field study, a set of overall requirements for computer support of the coordination of software testing is established in section 8.1. Section 8.2 establish a set of detailed requirements for how to computer support the bug form mechanism described in chapter 6. The chapter is concluded with a brief discussion on the generality and usability of the requirements listed.

## 8.1 Overall requirements for support of the coordination of software testing

Many software testing tools already exist in the commercial market. They primarily support the individual test tasks, or provide an overview of the state of affairs by applying a specific set of testing metrics. They do rarely support the coordination of distributed software testing and correction activities.

There are several areas where support for coordination of software testing should be considered. The findings from the field study (see chapter 3 and 6) can be used for establishing an overall picture of the coordination of the bug handling activities. This picture is first of all characterized by the use of a set of conceptual structures reflecting central aspects of the field of work (registering, diagnosing, and correcting bugs), and of the structures of the cooperative work arrangement. The most important structures from the field of work reflected in the conceptualizations are the registered bugs, the classifications of bugs and software, and the software architecture. Structures reflected from the work arrangement are mainly the plans, work procedures, actors, and roles. These structures can be regarded as the dimensions of objects of coordination work along which the coordination is conducted, i.e., the coordination activities are done by means of abstractions and conceptualizations of the nature of the work, not by directly interacting with the objects of the work (e.g., the code).

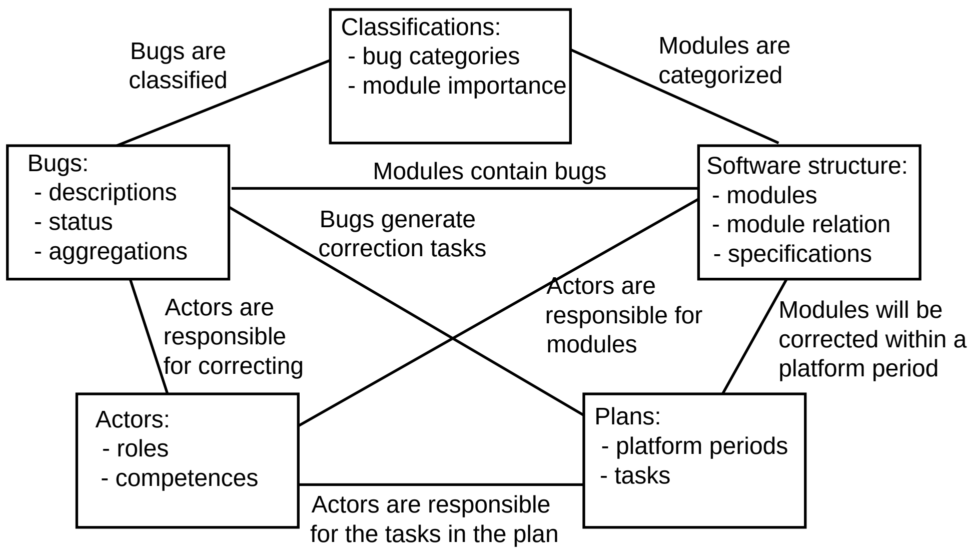


**Figure 8-1:** A model of the conceptual structures used when coordinating the registration, diagnose, and correction of bugs. The lines between the structures illustrate different kinds of relations typically established as a result of the coordination activities. For example: a registration of a bug will result in a relation between the bug and a classification. The three upper conceptual structures are derived from the nature of the field of work, whereas the two structures at the bottom reflects the nature of the work arrangement.

### 8.1.1 Conceptual structures

A first overall set of requirements for computer support of the coordination of software testing includes accessible and modifiable data structures reflecting the conceptual structures. In relation to these conceptualizations a set of basic actions—using the conceptual structures—pops up in many of the bug handling coordination activities. These are first of all *classifying* bugs, tasks, modules, *routing* of information and requests, *monitoring* state of affairs, progress, and *allocating, meshing, and negotiating* the use of resources, etc.. The actions are performed by all the involved actors.

When discussing computer support of the coordination of certain aspects of software testing work, it is obvious to require access to basic operations on the data structures similar to the basic actions mentioned here. If we consider a computer system as consisting of a set of data structures and a set functions, the data structures should be symbolic representations of the above mentioned conceptual structures, and the functions must reflect the activities mentioned as accessible operations on the conceptual structures. So, as it can be seen from the brief discussion above some requirements mainly concern the data structures that must be provided, whereas others concern the functionality.

A general requirement is that all data structures or relations between data structures mentioned must be available and accessible to the actors, and in most cases they should be modifiable too. The most important overall requirements regarding conceptual structures are:

(a) A computer system supporting the coordination of complex software testing processes should provide access to data structures reflecting the architecture of the software complex. Main functionality, relevant classifications, and the relations to other modules should be included for all modules. Furthermore, references to the responsible software designers, relevant documentation, and specification should be available. All the structures must be modifiable.

(b) Access to descriptions of the specific bugs registered should be provided. The descriptions could contain information on originator, symptoms, priority, suggested diagnosis, involved modules, estimations, related responsibilities, correction, etc. The information should be accessible as aggregations with respect to modules, types of bugs, priorities, and responsible designers.

(c) A third type of conceptual structures that should be available is existing plans containing information on the use of both human and technical resources. The relations among tasks, deadlines, actors, software modules, etc. should be accessible. This includes access to a conceptual structure containing information on all involved actors, and to technical and hardware resources involved in the development and testing. Individual characteristics, present and planned workload, etc. of the actors and technical resources should be available.

(d) A central aspect of coordination work is classification of the structures, objects, situations, etc. from the field of work or the work arrangement. This is also a central aspect of coordination of software testing. This implies the need for access to data structures containing information and descriptions of the present classification categories for bugs, software modules, etc. The classification categories should be accessible in all work situations where they are used (e.g., in bug registrations). Furthermore, in order to make classification structures modifiable, support of distributed changes to the classification schemes should be supported. This requires some kind of structured communication channel and/or a mechanism structuring the negotiation process.

(e) The last conceptual structure concerns the organizational context. A computer support system should provide access to structures of information on organizational procedures, techniques, standards, etc. and to structures containing information on the requirements for the software complex (both the existing specifications and the suggested interpretation) used in the test work.

### 8.1.2 System functionality

The most obvious requirement for functional support concerns distributed registration and classification of bugs, and support to the testers in filling in all the information required in order to perform the diagnosis. Handling bugs could be improved by refining the bug classification system. Too often the existing classification structure led to discussions resulting in a re-classification made by the spec-team. A more elaborated classification of the type and importance of bugs would support the testers in providing useful information to the spec-team and to designers. Research within software engineering may provide input for more elaborate classification structures, for example, standard software quality taxonomies (e.g., Boehm, 1981), or a software risk taxonomy. Support for discussions via electronic mail or bulletin boards among testers would also improve the quality of the information registered.

Access  should be offered to the actors that aloows them to specify (and later on modify) a work-flow process stipulating to whom information on the bug should be routed, how the bug registration is made visible to other testers, etc. When an actor has completed his activities in relation to a specific bug form, the work flow system could automatically validate that the required information is registered, pass the information on to the next actor (or group of actors), and notify the receiver(s) that new action must be taken. However, in most situations is it impossible to completely specify all situations which may occur (cf. e.g., Suchman, 1987). The coordination of software testing in the S4000 project was no exception. The actor completing an activity must therefore be able to overrule the routing and redirect the information to somebody else. The protocol stipulating the routing should, furthermore, be based on roles to which actors can be related. This could draw upon research addressing available work flow technologies (e.g., De Cindio *et al.*, 1988; Kreifelts *et al.*, 1991a), but it will not be discussed further.

The coordination work itself is often handled in a distributed manner. Facilities should be provided for a distributed creation and registration of new tasks. Tasks should be established as relations between a task, an actor, and a deadline. Functions for integrating the tasks in the existing plans and for notifying the affected actors are required. The structure for how, and whom, to notify should be available as a modifiable structure describing how this stipulation should run. This, of course, requires a communication structure allowing the actors to send requests, accepts, and rejects to each other. For example, supporting diagnose of bugs primarily implies support of communication among the spec-team members. Without face-to-face communication, the spec-team members would have had severe problems. E-mail based communication between spec-team members, testers and designers could support diagnoses and prevent some of the interruptions in the work derived from the need for *ad hoc* discussions. A communication channel where the actors can negotiate a diagnosis, the resource allocations, a deadline, etc. by means of structured messages and a predefined message flows would be very useful. The structure of the messages and dialogue flow should be modifiable, i.e., tailored to the concrete situation. The coordination of the diagnosis work could also be supported by providing access to information on already reported bugs.

In order to improve the awareness of changes made by others, and to establish a common understanding of the software complex, some support for “viewing” the structure of the software complex should be provided. It was obvious, that the designers had problems in relating themselves to the structure of other designers’ modules although these had an essential impact on how they should (re-)design their own modules. This should be supported by what Henderson and Cooprider call “representation” in their “production technology dimension” of CASE tools, i.e., functionality to enable the user to define, describe or change a definition or description of an object, relationship or process (Henderson and Cooprider, 1990). Improved use of some of the existing specification techniques or CASE tools could decrease the need for *ad hoc* coordination by providing an improved structure of the field of work, i.e., the software system being designed (Mathiassen and Sørensen, 1994).

Furthermore, features that support the testers and designers in making it possible for others to be aware of registered bugs and implemented corrections are required. Both user activated and automatic distribution of this type of information should be available. All actors must, upon request, be able to receive information on the state of affairs. This can be fulfilled through access to the registered bugs, their diagnosis, and status, or through access to information on a specific correction task. Providing designers and testers with browsing and query facilities to a database containing all registered bugs would enable these actors to access aggregated information on reported bugs which have not yet been corrected, the number of a specific type of bugs, the number of not yet corrected bugs in a specific module, the number of bugs a specific designer is responsible for getting corrected, etc. Also access to view the project schedule would be useful for monitoring state of affairs. This functionality could be provided by means of some of the existing project planning tools (e.g., Microsoft Project), and should support requests like: Who is responsible for the UIS-module? Which modules are John responsible for? How busy is Tom during the next integration period? How much time has been spent on correction so far?

When analyzing the coordination of software testing and correction work conducted within the S4000 project at Foss Electric six overall functions to be supported emerged. These were the flow of work, registration and classification of bugs, diagnosing bugs and allocating resources, correction of bugs, verification of the reported corrections, and monitoring the state of affairs. A more elaborate discussion of how the requirements are derived from characteristics observed in the work setting and structured according to these overall functions can be seen in Carstensen *et al.* (1995c). All the requirements mentioned there will, however, also be mentioned in the following.

## 8.2 Requirements for a mechanism supporting the bug handling process

The aim of this section is to present and discuss requirements for a computer-based coordination mechanism supporting the coordination of the distributed software testing activities, i.e., distributed registration of bugs, routing the required information to the actors, and stipulate the sequence of activities and the involvement of actors in the register-diagnose-correct-verify process. The requirements are based on the findings from the field study (especially with respect to the bug form described in chapter 6) and the overall requirements for coordination mechanisms described in section 5.5.

The description is organized top-down, i.e., it goes from general requirements (presented above) towards more concrete requirements and sketches of how a computer-based mechanism could be implemented. Distinguishing in any exact manner between requirement specification and systems design is widely recognized as an impossible task (Andersen *et al.*, 1990). I have, however, as a pragmatic approach to specifying the computer-based bug handling mechanism tried to distinguish between, on the one hand, which functions it should perform on which data, and on the other hand, how these functions are performed, and what mechanism should look like. For simplicity reasons I will, in the rest of this chapter, call the computer-based bug handling mechanism Bug-CM (Bug-Coordination Mechanism).

First step is to delimit the set of overall requirements to address. Next step (section 8.2.2) is a discussion of the basic functionality of the Bug-CM, and how the interaction between the actors and the mechanism should be. Then required structures are discussed (section 8.2.3), and which functions for accessing the data structures should be provided. Section 8.2.5 defines the links from the Bug-CM to “external” mechanisms. It is sketched which concrete data structures, classification types, user interface structures, etc. that should be included. Finally a brief discussion of the protocol embedded in the mechanism should be designed in order to fulfill the requirements is provided.

### 8.2.1. General requirements for a computer-based bug handling coordination mechanism

The requirements established above will be used as inspiration and criteria of relevance for the discussion of the requirements for the Bug-CM in this section. For each of the listed requirements I have considered the relevance of including it as a requirement for the Bug-CM. Aspects which are clearly outside the bug form mechanism are excluded in order to delimit focus. For example, the requirements on providing access for the actors to revise, update, and get an overview of the work plans are excluded in the following discussions.

Before establishing the set of requirements, some general decisions on the allocation of functionality between the involved actors and the Bug-CM should be taken, i.e., decide what is to be handled by humans and what is to be handled by the computer system. An example is the registration of a bug. It is the actor who decides the classification of a bug and enters all relevant information into the system, but it is the Bug-CM that validates that all mandatory information is entered, as well as it passes on the registration to the actor(s) responsible for the next sub task to be conducted.

The Bug-CM should *handle all information*—and aggregations of information—related to the registration of a bug, its diagnosis and correction, and its verification. Furthermore, all *routing of information* between the involved actors (or rather roles) should be handled by the Bug-CM. The Bug-CM is *not making any decisions* regarding whether or not a phenomenon is a bug, how to classify a bug, how to diagnose a bug, which modules a correction will affect, what a correction time estimate should be, how to correct a bug, etc. This is taken care of by the human actors. To phrase it differently: The Bug-CM should not include any ‘knowledge’ used when decisions are taken in the actual work, e.g., decisions on whether new expertise is required, or decisions on how to correct a bug.

The Bug-CM will only contain 'knowledge' of how the work-flow should be stipulated.

Basically, the Bug-CM should support different problems: Ensure that all registered bugs are treated until they reach a final state; Provide an overview of state of affairs that is correct and up to date; And handle a process of distribution (of the registration tasks), compilation (of all the bug forms for diagnosing), distribution (of the registration tasks), and compilation (for verification). These overall requirements result in requirements that actually delimit the degrees of freedom for the actors. This is necessary in order fulfill all three needs.

The decisions on the division of Bug-CM—Human actor functionality implies that basically the Bug-CM should provide four types of functionality:

(1) The mechanism should offer facilities for distributed registering new bugs. The Bug-CM should support the registration so it is ensured that all mandatory information is entered before the registration is completed and passed on. All registered bugs should be filed so aggregations and statistical information on the complete set (or subsets) of the bugs can be generated.

(2) The Bug-CM should stipulate the work flow by routing the information between the actors. When a certain actor has completed his or her activities in relation to the handling of a specific bug, the mechanism should automatically validate that the required information is registered, pass the information on to the next actor (or group of actors), and notify the receiver(s) to indicate, that the specific bug is now at a stage where a new action can be taken. The actor completing an activity must be able to overrule the routing and redirect the information to whoever he or she wants. Furthermore, the protocol stipulating the routing must be based on roles to which actors can be related, and it should be possible to change the actual actor related to a role without changing the protocol stipulating the routing. Receiving a notification can be regarded as receiving a request. Hence, the Bug-CM should provide a facility making it possible to reject a request, i.e., return the request to the originator.

(3) The Bug-CM should support the resource allocation tasks in relation to the diagnosis and estimation tasks. When the spec-team decides on the diagnosis of a bug and on who is going to correct a specific bug, they also handle resource allocation. To be able to do this the mechanism should provide information to the actors on the relations between roles and actors, the architecture of the software complex, the relations between software modules and the responsible designer, the workload of the involved designers, the existing work plans, and the relations between tasks and deadlines, etc. Supporting the diagnosis task—and thus the resource allocation task—requires an improvement of the existing classifications of the importance of bugs. The existing categories are insufficient and are used for several different purposes, e.g., both as an indication of the problem from the perspective of the testers and as an indication of the importance from a software development perspective. The Bug-CM should provide a more sufficient and elaborated set of categories of bugs.

(4) The Bug-CM is required to support the actors in obtaining awareness of the state of affairs regarding the registered bugs. The Bug-CM should provide a series of querying facilities for generating aggregations on reported bugs not yet corrected, the number of bugs with a certain classification, etc. It is important that the actors can monitor the progress of testing activities and the state of affairs in general, i.e., to get an overview of how many percentage of the code is tested, what is the number of man-hours planned for testing, how much time has been spend on testing so far, etc.

The requirements discussed so far reflects by no means whether the facilities should be provided directly by the Bug-CM itself, or whether they can be provided through links to other mechanisms. It is obvious to expect that some of the facilities are provided by the Bug-CM, but facilitated by other (computer-based) mechanisms, e.g., information on the software architecture could be obtained in a data-structure maintained by a CASE tool.

In order to reduce the complexity of the task of specifying the requirements for the Bug-CM, I have excluded supporting certain aspects of the coordination of software testing. The most essential of these are:

- Negotiation structures.

  An important aspect of coordinating software testing and correction is negotiation on classifications, diagnosis, resource allocations, deadlines, etc. A Bug-CM should provide some predefined structures to be used by the actors when negotiating. I have decided not to discuss such predefined structures further here, apart from a requirements stating that all requests sent from one actor to another can be rejected and routed back to the originator.
- Decision support.

  Apart from providing relevant information, the Bug-CM should not provide any decision support facilities. As mentioned, have I chosen to let all decisions on, for example, the classification of a bug, the diagnosis, how to correct a bug, etc. be taken by the actors without any active intervention from the Bug-CM. Decision support is beyond the scope of this dissertation.
- Communication facilities.

  It is relevant to support, for example, the members of the spec-team in being geographically distributed, and then use the mechanism as a means for communicating on the diagnosis of a bug. Such general communication facilities are considered out of scope here. The problem, could be addressed by some of the same means as addressing negotiation (e.g., Flores *et al.*, 1988), collaborative writing (e.g., Beck and Bellotti, 1993), video-conferencing (like the ideas presented in Ishii *et al.*, 1994), etc.
- Access to updating plans.

  A central aspect of coordinating the handling of bugs is, of course, to be able to see the current plans and make changes to these. Here, support for updating the work plans is considered belonging to a separate computer-based mechanism. In this chapter the linking of the two mechanisms will only be addressed through a requirement stating, that information on a task generated by the accept of a bug must be routed to the actor responsible for handling the work plans. An automatic update of the plans combined with a notification of the actor(s) handling (responsible for) the planning would be relevant. This is, however, excluded from the requirements here. Further work on the requirements should involve a specification of the linking (cf. section 5.8) between the two computer-based coordination mechanisms.

Neither the mentioned requirements, nor the list of aspects that are excluded, should be regarded as exhaustive. During the process of refining the requirements, several of the requirements will be specified further, and further decisions on what to include and what to exclude will be taken.

### 8.2.2. The interaction between the actors and the Bug-CM

This section discusses the interaction between the actors and the computer-based mechanism (Bug-CM). This is done by means of step by step descriptions of which actions the actors (users) perform, and what kind of result this leads to in the Bug-CM. The Bug-CM should provide two general facilities. One supporting the registering-routing-diagnosing-correcting-verifying process (figure 8-2), and one facilitating browsing and search activities (figure 8-3, 8-4, 8-5, and 8-6).

For presentation purposes, the description of the types of interaction with the Bug-CM illustrates the ‘typical’ working sequence, i.e., the flow of actions as they would appear in a typical situation without exceptions.

Each row in the tables is a pair of ‘user action—Bug-CM responds’. First column describes the user actions and the next column contains the related reaction from the Bug-CM. The roles and actors are the same as those used to characterize the coordination work previously (cf. chapter 3 and 6). Decisions on the allocation of functionality between the human and the computational artifact are, thus, described as a set of scenarios (cf. e.g., Cambell, 1992; Carroll and Rosson, 1992).

The first process is the registering-routing-diagnosing-correcting-verifying process presented in figure 8-2. The description of the ‘typical flow’ of the registration and correction process illustrated in figure 8-2 does not contain examples of exceptions where, for example, the actor choose to overrule the stipulated workflow and skip some of the steps in the process. It should, according to the overall requirements discussed previously, always be possible for the actor to do so.

As mentioned above, support for browsing and search is the second general facility provided by the Bug-CM. Although this does not provide a similar ‘natural sequence’ as in the previous facility, I have used the same scenario structure in order to describe the browsing and search facility. This is done by presenting four prototypical situations:

(1) An actor (usually a tester or a designer) is interested in obtaining specific information on a specific bug (shown in figure 8-3);

(2) A tester who wants to check if a registered problem is identical to a phenomenon he has just recognized (shown in figure 8-4);

(3) The spec-team searches for relevant information in order to decide who is going to be responsible for a specific bug (shown in figure 8-5); and

(4) A manager or others searching for relevant information on the state of affairs (shown in figure 8-6).

The scenarios presented in figure 8-3 to 8-6 are based on activities observed in the field study. They are all highly contextual activities dependent of the role of the actor interacting with the Bug-CM. From a design point of view, the need for search and browsing facilities might be provided through a generic search and browsing function without distinguishing between types of actors and types of situations.

| **Actions from the actor(s)** | **The responds from the Bug-CM** |
|---|---|
| 1) A tester recognize a bug in the software, decides to report it, and “generate a new bug report”. | The Bug-CM replies by setting up an electronic form containing entry fields for the relevant information. |
| 2) The tester classifies the bug, fills in the fields in the form, and ask the system to “register the bug”. | The Bug-CM validates that all mandatory fields are filled in. If not the tester is requested to do this.<br>If this is OK the registration is filed in the central bug database and a notification is send to the spec-team. |
| 3) A spec-team member asks for the “next new registration”. | All registration information are presented to the spec-team member together with relevant fields to be filled in concerning the diagnosis and estimation. |
| 4) If the bug cannot be accepted by the spec-team, a spec-team member demands to “reject bug”. | The bug is filed as “rejected” in the central database, and a notification is returned to the tester indicating that the registration has been rejected. |

| | |
|---|---|
| 5) If the bug is accepted, but it is decided to postpone it, a spec-team member classifies the bug, describes the reason to postpone it, and demands to “postpone bug”. | The bug is filed as “postponed” in the central database, and a notification is returned to the tester indicating that the registration has been postponed. |
| 6) If the bug is accepted by the spec-team, a spec-team member fills in the classification of the bug, the platform period in which it is going to be fixed, and the responsible module(s). | On the basis of the specified module, a default responsible designer is added to the information. |
| 7) The responsible designer(s) are filled in together with the correction time estimate for each. “Bug accepted” is demanded. | The bug is filed as “accepted” in the central database. The responsible designer(s) and the originator (the tester) are notified. |
| 8) A designer demands a “see correction request”. | All information on the registered bug is presented to the designer. |
| 9) If the designer rejects to do the corrections or cannot accept the correction time estimate he fills in a rejection description and asks to “reject request”. | The bug is filed as “correction request rejected” in the central database and the spec-team members is notified. The spec-team can then handle it as a new registration (cf. entry 3). |
| 10) If the designer accepts the diagnosis and estimate he demands a “accept request”. | The bug is filed as “correction request accepted” in central database and the spec-team members are notified. |
| 11) The designers asks for a “register correction”. | Identification information on the bug is presented to the designer together with fields for registering information on the corrections. |
| 12) The designer fills in the time spend and information on affected modules and files, and “register correction”. | The bug is filed as “corrected” in the central database. |
| 13) The platform master (PM) asks to “see corrections to be verified”. | Information on all corrections to be verified in the next integration period is presented to the PM. |
| 14) The PM demands a “register verifications”. | Information on the next bug to be verified is presented to the PM. |
| 15) If the bug presented cannot be verified the platform master fills in a description of the problem and demands “verification rejected”. | The bug is filed as “not corrected” in the central database, and the spec-team members are notified. The spec-team can handle it as a new registration (cf. entry 3). |
| 16) If the bug presented can be verified the PM demands “verification accepted”. | The bug is filed as “corrected and verified” in the central database. |

**Figure 8-2:** A table of the typical user actions and Bug-CM reactions (each pair in the rows) in the registering-routing-diagnosing-correcting-verifying process.

| Actions from the actor(s) | The responds from the Bug-CM |
|---|---|
| 1) An actor demands “see specific bug”. | The Bug-CM replies by presenting a search template with entry fields for bug registration id., tester id., bug classification, responsible module, responsible designer, etc. |
| 2) The actor fills in the fields to be searched. | The central bug database is searched for all registered bugs fulfilling the search profile. The search template used and the number of retrieved instances are presented to the actor. |
| 3) If the actor is not satisfied with the found number of bug registrations he fills in new information in the fields and demands “search”. | Se the previous Bug-CM reaction. |
| 4) The actor asks to “see next bug”. | The complete set of information on the next bug in the list of retrieved records are presented to the actor. |

**Figure 8-3:** A table of the user actions and the matching reactions from the Bug-CM in a process of searching for information on specific bugs.

| Actions from the actor(s) | The responds from the Bug-CM |
|---|---|
| 1) A tester asks to “see similar bugs”. | The Bug-CM replies by presenting a search template with entry fields for the responsible module and keywords for the bug description. |
| 2) The tester fills in as precise information on the responsible module and the keywords as possible and asks the mechanism to “search”. | The central bug database is searched for all registered bugs fulfilling the search profile. The number of retrieved records are presented to the tester. |
| 3) If the tester is not satisfied with the found number of candidates he or she starts over again by asking to “see similar bugs”. | See the Bug-CM reaction in step 1. |
| 4) The tester demands “see next bug” | The complete set of information on the next bug in the list of retrieved records are presented to the tester. |

**Figure 8-4:** A table of the typical user actions and Bug-CM reactions in a process of a software tester searching for bugs with similar characteristics.

The three queries illustrated in figure 8-5 below are all queries that address information outside the Bug-CM itself. In order to fulfill these requirements the Bug-CM will have to link to other computer-based mechanisms providing the relevant information.

| Actions from the actor(s) | The responds from the Bug-CM |
|---|---|
| 1) A member of the spec-team demands “see roles”. | The Bug-CM replies by presenting a template with entry fields for roles and actors. |
| 2) The member of the spec-team fills in the fields of actor or role and demands “see roles”. | The Bug-CM presents all roles related to the defined actor (or vice versa). |
| | |
| 1) A member of the spec-team demands “see responsibilities” | The Bug-CM replies by presenting a template with entry fields for modules, tasks, and designers. |
| 2) The member of the spec-team fills in the fields of designer and/or module and/or task, and demands “see responsibilities”. | All responsibility relations between:<br>• specified designers - tasks,<br>• specified designers - modules,<br>• specified modules - designers,<br>• specified tasks - designers,<br>are presented. |
| | |
| 1) A member of the spec-team demands “see workload” | The Bug-CM replies by presenting a template with entry fields for designers, tasks, and a deadline. |
| 2) The member of the spec-team fills in the fields of designer and/or tasks and deadline, and demands “see workload”. | The Bug-CM presents the man hours to be spend from now until the deadline:<br>• on the specified tasks.<br>• by the specified designers. |

**Figure 8-5:** A table of three typical user actions - Bug-CM reactions pairs in a process in which the spec-team attempts to establish the required information for deciding on responsibility. The first supports the spec-team member in getting information on, for example, who will be platform master in the next period. The next can be used to get information on, for example, who is responsible for the UI-module or which tasks are James responsible for. The intention of the last one is to provide information on how busy a specific designer is going to be, according to the plans, in a forthcoming work period.

| Actions from the actor(s) | The responds from the Bug-CM |
|---|---|
| 1) An actor demands “see state of affairs”. | The Bug-CM replies by presenting a search template with entry fields for bug status, bug classifications, module, and designer. |
| 2) The actor fills in the fields he consider relevant and demands “search”. Default for all fields is that all are included. | If no fields have been filled in the actor is requested to do this.<br>If at least of the fields have been filled in, the central bug database is searched for all registered bugs. The number of registrations in the database fulfilling the bug status, the bug classification, the module, and the designer specified is presented. |
| 3) If the actor is not satisfied with the found aggregation number he starts all over by demanding “see state of affairs”. | See step 1. |
| 4) If the actor wants to see details on the bugs aggregated he demands a “see next bug”. | The complete set of information on the next bug fulfilling the requirements on the bug status, the bug classification, the module, and the designer specified is presented to the actor. |

**Figure 8-6:** A table of the typical user actions and the matching reactions from the Bug-CM in a process of searching for state of affairs information.

The previous four tables illustrated prototypical situations where an actor search for information in order to establish a basis for taking decisions on how to organize the work. They do not cover all relevant situations and exceptions from these situations. Furthermore, the search situations illustrated have all been based on the same general structure. This structure is illustrated in figure 8-7. In a case where the Bug-CM is going to be implemented, it will be obvious to base the implementation on such a general search structure.

| Actions from the actor(s) | The responds from the Bug-CM |
|---|---|
| 1) An actor evokes the search function. | The Bug-CM replies by presenting a search template with entry fields for bug registration id., tester id., bug classification, responsible module, responsible designer, etc. |
| 2) The actor sets up search profile by entering fields in the search template. | The central bug database is searched for all registered bugs fulfilling the search profile. The search template used and the number of retrieved instances are presented to the actor. |
| 3) The actor asks to "see next record". | The complete set of information on the next record in the list of retrieved records are presented to the actor. |

**Figure 8-7:** A table of the generic user actions and the matching reactions from the Bug-CM in a process of searching for information.

In further work on specifying requirements, the fact that the situations and the need for actors to obtain aggregated information are different could lead to radically different browsing and search functions. Searching for matching records is only the basic starting point we have taken here. A next step would be to provide different types of graphs, links between mechanisms etc. Here, the spec-team member assessing state-of-affairs would probably need other functions than a software tester who needs to get an overview of the software errors found.

### 8.2.3. Data structures required

Overall requirements have been set up, and the required functionality has been illustrated. It is time for a more detailed discussion of which data structures the mechanism should have access to in order to be able to fulfill the requirements, either by having them as internal structures, or through links to other mechanisms. This section will illustrate which structures are required.

The structures along which the coordination of software testing and correction activities is performed were briefly introduced in section 8.1. Using these and the established requirements, results in the following set of structures to be provided: The registered bugs, the bug classifications, the actors and their roles, the tasks, the work plan and platform periods, and the software architecture. These structures are, of course, related to each other in different ways, see figure 8-8. Another work settings might require additional data-structures, for example, references to technical resources such as production equipment etc.

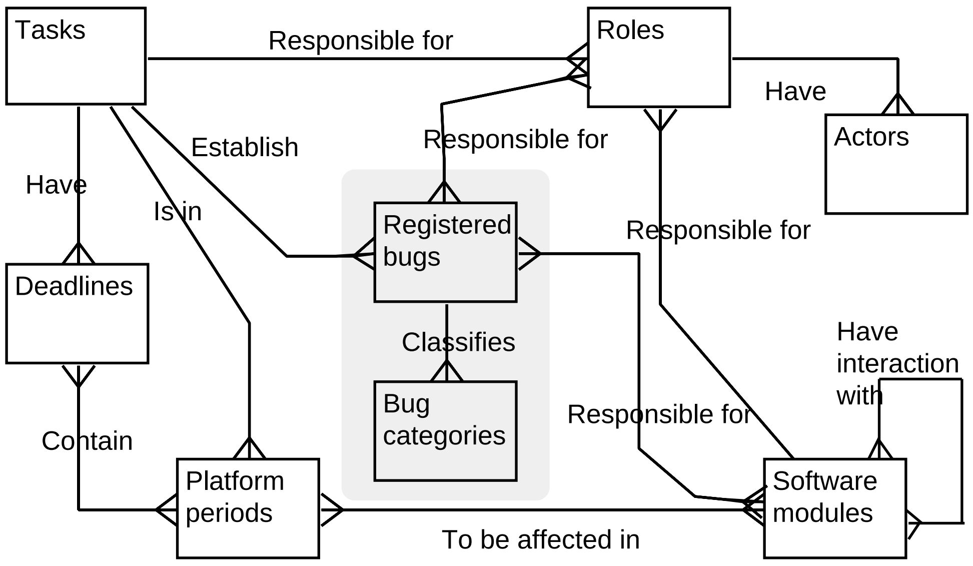


**Figure 8-8:** A simple version of a entity-relationship like diagram illustrating the central structures to be contained or accessed by the Bug-CM. The grayed box illustrates which structures would obviously be inside the Bug-CM. The forks in the relations indicate a many-relation. For example a bug can have exactly one classification, whereas a bug classification can be used for many registered bugs.

In some more details, the data structures include:

- Registered bugs.

  The central database in the Bug-CM should be a file containing a compilation of all relevant information on all bugs ever registered. All information regarding the description of the bug, its diagnosis, its corrections, the classifications and the status of the bug and its correction should be filed here. In order to support search for bugs with specific characteristics, the bug description field and the diagnosis description field should partly be based on selections from predefined sets of situations descriptions.

- Bug classifications.

  As argued previously a generally accepted set of categories for classifying bugs is essential. The set of categories should be substantially improved compared to the categories on the existing form (cf. section 6.6). The classification set will be discussed further in the following.
- Actors and roles.

  Descriptions and information on all possible roles to be included in the software testing and correction work should be available. The roles should be defined including obligations, time limitations, etc. Also information on all the actually involved actors should be accessible, on for example, involvement in different projects, main interests and competence, etc.
- Tasks.

  Access to a database covering all tasks, their estimate, their deadlines, a description, a reference to further specification of goal and acceptance criteria, priority, current status, etc. is demanded, including references to responsible roles/actors, work plans, and originator.
- Work plan.

  A work plan illustrating how all the tasks and roles/actors are related to deadlines (platform periods) is, of course, an essential tool for meshing correction tasks and activities with already defined tasks, and to decide on deadlines for the correction tasks. Such a work plan structure is identical to a combination of the tasks, platform periods, and deadline structures in figure 8-8.
- Software architecture.

  The structure of the software modules, their relation to each other, their importance from a product point of view, etc. are essential when deciding on the diagnosis of a bug, and who to make responsible for the corrections. Information on module name, module priorities, and module status, and references to detailed specifications, other modules related to a specific module, and who is responsible for designing the module is needed.

### 8.2.4. Operations on the data structures

Viewing a computer system as a set of data structures and a set of operations on the data structures, we are now ready to discuss the operations. Previously, I have identified a set of overall functions of coordination activities including: Meshing activities, task, and deadlines; Relating diagnosis and correction tasks to actors and work periods; Allocating resources; Monitoring progress and state of affairs; And negotiating classifications, allocations, deadlines, etc. Some of these have been excluded (cf. section 8.2.1), but they have worked as a criteria of relevance when going through the concrete requirements for the Actor—Bug-CM interaction, leading to the following identification of which operations must be provided:

First, there should be access to *create new* instances of the structures, mainly creating new bug registrations. Creation of new instances of structures outside the Bug-CM should be available in other computer-based mechanisms. Also creation of new bug categories should be available, but some limitation on who have access to this will probably be required. The next basic operation concerns *relating* the bug structure (cf. figure 8.8) to bug classifications, responsible software module(s), and responsible role/actor(s).

In order to validate information entered, and to setup search requests of different kinds, the mechanism should include a *compare* operation, and an *update* of the bug structure. Use of the update operation can then—as a next step—trigger other actions, e.g., a *send notification information* (e.g., pass information or return a rejection to a specific actor/role), or a *notify external mechanism* about an update to be made.

Access to the information registered should include a *read information.* This requires access to *setup query parameters* and *search the database*. Reading information will, of course, require a *present information.* Read information concerns reading a specific (set of) instance(s) of a given structure. This can be organized either as a read from the registered bug database or the bug classification structure, or it can—if the information is placed in an external mechanism—result in a *read request* to the external mechanism.

If access to the above mentioned basic manipulations on the data structures is provided, a basis for establishing of a first version of the Bug-CM exists.

### 8.2.5. Links to external mechanisms

The Bug-CM should have access to other computer-based coordination mechanisms in order to provide the required facilities. These other mechanisms are not established yet, and have not been discussed in this chapter. Figure 8-9 illustrates, through a re-use of figure 8-8, which mechanisms the Bug-CM could have access to (be linked to). The illustration contains only one of several possible decompositions of the mechanisms.

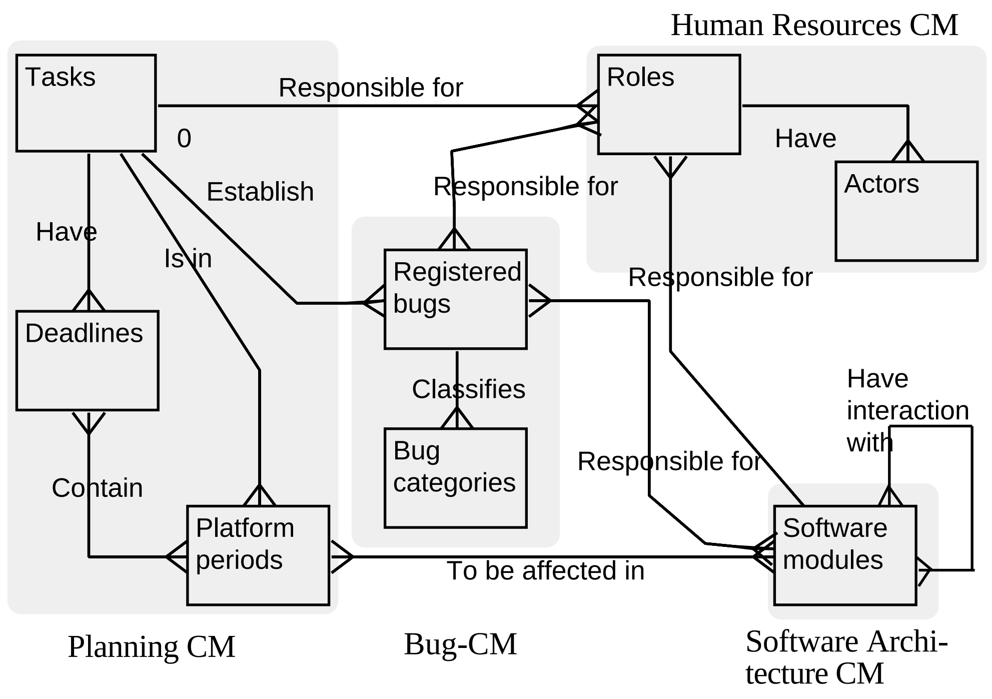


**Figure 8-9:** An illustration of four computer-based mechanisms that, altogether, could support the coordination of software testing and correction work. Each of the gray boxes represent a mechanism. The mechanism described in this chapter is the Bug-CM in the center. The forks in the relations indicate a many-relation.

The Bug-CM has access to three other (computer-based) mechanisms supporting the work plan aspects, the human resource aspects, and the software modulation aspects of the coordination activities. There are two types of access to the Planning CM: One is a *read from* link, i.e., the Bug-CM can read (or request a read of) the content of structures of tasks, deadlines, and platform period information. The other is an *update* link (used for defining new tasks in the plans). This can be implemented either as a write directly in the structures, or as a send update request. The access to information on the roles/actors and their work load is made through reading the content of the structure handled by the Human Resources CM. The same goes for the Software Architecture CM. The Bug-CM can access information on which roles are responsible for which tasks in the current plans both via the link to the Planning CM and via the link to the Human Resource CM. The same goes for accessing information on who is responsible for which modules. Here the information can be accessed via the Human Resource CM or via the Software Architecture CM.

### 8.2.6. Redesign of the bug form artifact

The information processing history contains many mistakes derived from replicating the existing manual administrative systems (Hammer and Sirbu, 1980). Designing a computer-based coordination mechanism based on existing paper-based social mechanisms of interaction, should not mean replicating the existing artifacts and procedures. The bug form mechanism used at Foss Electric as a means of supporting coordination work should therefore not just be given the power of computing. If taken seriously, both artifacts and procedures should be made subject to redesign. The purpose of this chapter is, as mentioned earlier, to demonstrate ideas, not to completely redesign artifacts and procedures used. I will, therefore, only partly sketch one possible solution. The Bug-CM is represented as a semi-structured message system (see e.g., Malone *et al.*, 1987; Herskind and Nielsen, 1994) containing a set of fields to be filled in with further constraints attached through the protocol specified. Designing the actor - Bug-CM interaction the ‘form’ (illustrated in figure 8-11) might result in several screens reflecting different roles interacting with the system in different situations.

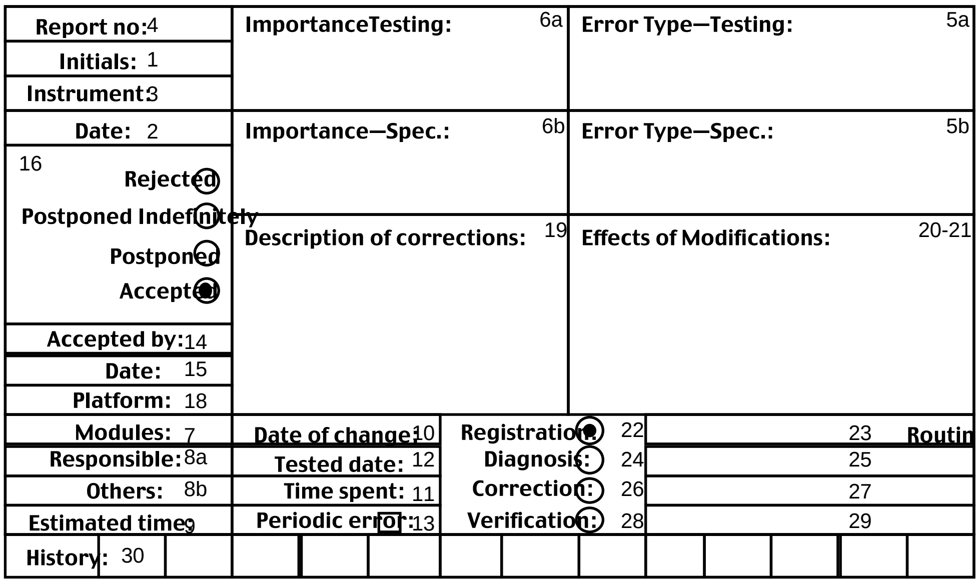
| Re-design of the original Bug Form | |
|---|---|
| 1 | *Concerning the 'Initials' field:* Assuming that the Bug-CM is linked to the Human Resource CM, the actor will at some point have logged into the system, and his or her initials will therefore automatically be inserted as the default value. If not appropriate, the initials can be altered. |
| 2 | *Concerning the 'Date' field:* The date will be inserted, when the instance of the form is made. |
| 3 | *Concerning the 'Instrument' field:* The tester enters the name. If, at a later stage, a repository of pending projects were to be established, this field would naturally be linked to such a repository. |
| 4 | *Concerning the 'Report no' field:* A unique report number will automatically be generated. In the manual system, only reports of accepted software bugs are numbered. This way the last number reflects the total of accepted bugs. In the Bug-CM simple queries will quickly be able to determine this number, even if all reports are numbered. |
| 5 | *Concerning the 'Description' field:* The description of the phenomenon detected is a mixture of classification and free text annotations. The Bug Categories data-structure provides a classification scheme of types of software bugs. Free text is used as a 'other' category. Since both testers and spec-team members add information to this field, it must be split in two. |
| 6 | *Concerning the 'Classification' field:* The classification of the seriousness of the software bug needs serious redesign. The chosen solution is a two dimensional classification (perspective x importance). Using arguments as in the previous field, the classification of seriousness must be split in two. |
| 7 | *Concerning the 'Involved modules' field:* Characterization of the modules involved in testing the bug are to be made by accessing the Software Modules data-structure. |
| 8 | *Concerning the 'Responsible designer' field:* Since the Bug-CM will be linked to the Planning CM, the responsible designer(s) can be assigned as a default by accessing the relationship between roles and software modules. The default value can be overruled by choosing one or more other designer from a list. Because neither this or the estimated time field covers the allocation of more than one involved designer, an additional field is needed. |
| 9 | *Concerning the 'Estimated time' field:* The time is filled in by the spec-team members. |
| 10 | *Concerning the 'Date of change' field:* The date of change of the software is automatically given the default value of the date the designer reports the changes made. |
| 11 | *Concerning the 'Time spent' field:* The amount of time spent is filled in by the designer. |
| 12 | *Concerning the 'Tested date' field:* The date is given the default value of the date the designer reports changes made. This implies, together with the default of the change date field that in all other cases than the changes being made and tested the same day, the defaults must be altered. |
| 13 | *Concerning the 'Periodic error' field:* If the bug is assumed to be periodic, and has been assumed to be fixed, the designer checks in this box. |
| 14 | *Concerning the 'Accepted by' field:* The spec-team member accepting the bug form chooses his or her initials from a list. Initials on more than one person can be added. |
| 15 | *Concerning the 'Accepted date' field:* The field is default assigned the date the spec-team receives the form describing the software bug. |
| 16 | *Concerning the 'To be' field:* The classification of how to proceed further could be linked to the classification of bug categories. Since a fourth category 'Postponed indefinitely' is already used at Foss Electric, although it is not on the form, we have chosen to include it. |
| 17 | *Concerning the 'Software classification' field:* This field is redundant, because it covers the same information as the platform field. |

| | |
|---|---|
| 18 | *Concerning the ‘Platform’ field:* This field is actually linked to the Planning CM. It indicates in which platform the bug should be fixed. Current platform period is default value. |
| 19 | *Concerning the ‘Description of correction’ field:* The corrections made can be classified using the Bug Categories structure. Alternatively, free text can be annotated as an ‘others’ category. |
| 20 | *Concerning the ‘Modified applications’ field:* Classification of which software modules and files have been changed inserted by linking to the Software Architecture CM. |
| 21 | *Concerning the ‘Modified files’ field:* This field is merged with the modified applications field. |

**Figure 8-10:** Suggested redesigns of the fields in the original bug form (see figure 6-6).

If we take a critical look at the original paper-based bug form used at Foss Electric (see figure 6-6), there are several obvious changes which can be made when turning it into a computer-based form. Some of these changes could, with advantage, be made even to the paper-based form. Others are made due to the fact that a set of linked computer-based coordination mechanisms (cf. section 8.2.5) is assumed to be designed. This, for example, results in several fields having default values.

Figure 8-10 reviews each field in the original form and discuss the changes decided. Each change discussed in the figure is introduced by defining the related field in the original bug form as illustrated in figure 6-6. Figure 8-11 shows an initial redesign of the form, representing the design decisions discussed in figure 8-10.



| | | |
|---|---|---|
| Report no: 4 | ImportanceTesting: 6a | Error Type—Testing: 5a |
| Initials: 1 | | |
| Instrument 3 | | |
| Date: 2 | Importance—Spec.: 6b | Error Type—Spec.: 5b |
| 16 Rejected ○ | | |
| Postponed Indefinitely ○ | Description of corrections: 19 | Effects of Modifications: 20-21 |
| Postponed ○ | | |
| Accepted ● | | |
| Accepted by: 14 | | |
| Date: 15 | | |
| Platform: 18 | | |

| | | | |
|---|---|---|---|
| Modules: 7 | Date of change: 10 | Registration ● 22 | 23 Routin |
| Responsible: 8a | Tested date: 12 | Diagnosis: ○ 24 | 25 |
| Others: 8b | Time spent: 11 | Correction: ○ 26 | 27 |
| Estimated time 9 | Periodic error: □ 13 | Verification: ○ 28 | 29 |
| History: 30 | | | |

**Figure 8-11:** Redesign of the Bug-CM artifact. The numbers on the 'form' refer to the changes numbered in figure 8-10. The software tester uses the default values or fills in fields 1, 2, 3, 4, 5a, and 6a. The spec-team applies default values or fills in fields 5b, 6b, 7, 8a, 8b, 9, 14, 15, 16 and 18. The software designer(s) correcting the software error applies default values or fills in fields 10, 11, 12, 13, 19 and 20–21. All actors use fields 22–30. The layout is not supposed to include directions for how the actor-Bug-CM interaction should be designed.

The major changes made in the re-designed 'form' compared to the original form (figure 6-6) are listed, and briefly discussed, in the following. The annotated numbers (e.g., 17) refer to the numbers in figure 8-10 and 8-11. The major changes are:

- Some fields in the form will upon instantiation or triggered by routing, be assigned default values. This can be accomplished because the Bug-CM will be one of a set of linked computer-based coordination mechanisms. Fields 1, 2, 4, 8, 10, 12, 15, and 18 will be assigned default values. The only one of these fields which are given a default value which can not be changed is the report number (field 4). In the manual system, the actors needed to maintain an unbroken sequence of report numbers for accepted software bugs in order to produce statistics. Given the computational power of the Bug-CM system, this is no longer necessary.

- An additional category, ‘Postponed Indefinitely’, has been included for categorizing bug forms. The difference between a bug form postponed indefinitely and one being rejected is that the former is recognized as a software error, the latter is not.
- The fields on software classification and platform (17 and 18) are, on the original form, used for the same purpose and hence merged into one.
- Given the linking to the Software Architecture CM the fields of modified applications and modified files (20 and 21) are represented in the same field on the re-designed ‘form’.
- An ‘others’ field (8b) have been added in order to reflect the practice of using the manual system, where several designers, besides the responsible, can be assigned the task of correcting the bug.
- In order to support the actors in routing the bug reports, fields reflecting the phase and the possible receivers have been added on the new ‘form’ (field 22-29). The semantics of the fields are discussed below.
- The history of the ‘form’ is represented in field 30 of the new form. This provides the actor with the possibility of obtaining an overview of who previously have updated the form and when they have done so.

The description of the software bug found (field 5) and the classification of the importance of correcting the software bug (field 6) both need to be redesigned. First, since both software testers and spec-team members update fields, have they been separated into four separate fields in the redesigned ‘form’. Second, the contents of the two fields need to be redesigned. The description field (number 5) is used for characterizing observed phenomena. It is proposed that a classification structure (contained in the Bug Categories shown in figure 8-8) is used, characterizing software bugs phenomena, e.g., program stopped, window in wrong place, unstable output on tests, etc. Field 6 on classification merges two different criteria in the original form: the perspective and the rating of importance. It would improve the use if we separate, on the one hand, why the tester deems a software bug important, and on the other hand, how important it is to correct the bug. The first could be taken care of by a classification of perspectives or concerns, e.g., maintainability, marketing, stability, safety, usability etc. As for the importance of correcting the software bug, a scale from 1 to 10 could be used. Using a software risk taxonomy to characterize the perspective and a software bug classification for categorizing the observed phenomena should be combined with free text annotations. In classification structures, it is important to make sure that there is an “others” or “miscellaneous” category on each level in order to urge the actor to provide as precise a classification as possible.

The original paper-based form does not present the fields in the sequence in which they are filled in by actors in the work-flow. In principle several criteria can be used to formulate the requirements for the design of the Bug-CM, e.g., present all fields to all actors in the sequence in which they are filled in or present only the most relevant fields as a default based on assessment of which fields are most relevant for each of the roles involved or for each stage in the process. The fields could be prioritized according to the assessed importance and presented in that order. Additionally, criteria regarding the clustering of data elements could be applied. Several data elements could be clustered, for example, all information on actors and roles, information on conceptualizations of the field of work, references to the contents of the field of work, information on decisions made, etc. All these criteria can, of course, be mutually contradictory. The most general solution would be to provide a limited possibility for each actor to create his or her own default views to the data elements. Further discussion along these lines have, however, been excluded here. Here the simple solution is to change it so the approximate sequence in which the fields are updated is reflected in the sequencing of the clusters of fields filled in by testers, spec-team members, designers, and the platform master.

Fields on the stages (22, 24, 26, and 28) have been included in the redesigned ‘form’ in order to provide the actors with an overview of which stage in the process the Bug-CM is at. These fields are also used when routing the form from one actor to one or more others, and hence potentially changing the state of the Bug-CM from, for example, registration to diagnosis. The corresponding routing field (23, 25, 27, 29) shows the possible receivers as a list of roles with the stipulated next step as the default value.

The history of the process is represented by a sequence of fields each containing information on role and date (field 30). The actor can get an overview of which states the Bug-CM has passed through before it arrived on his desk.

## 8.2.7. Towards a protocol for Bug-CM

For each instance of the ‘form’ (identical to each bug reported) a protocol must be stipulated, governing the routing of the ‘form’ through the four main stages: from a bug registered by a tester, to a member of the spec-team diagnosing the bug, to one or more designers fixing the problem, and lastly ending with the platform master verifying that the problem has been dealt with. The following sketch the protocol by discussing general requirements as to how a ‘form’ can be routed.

The process supported by the protocol consists of the four distinct phases: Registration, diagnosis, correction, and verification (see figure 8-12). The standard procedure would, in most cases, be that a role sends the form as a request to the next role in the process. If the receiver chooses to reject the request, for example because of incomplete information, the form is returned to the sender, cf. figure 8-12.

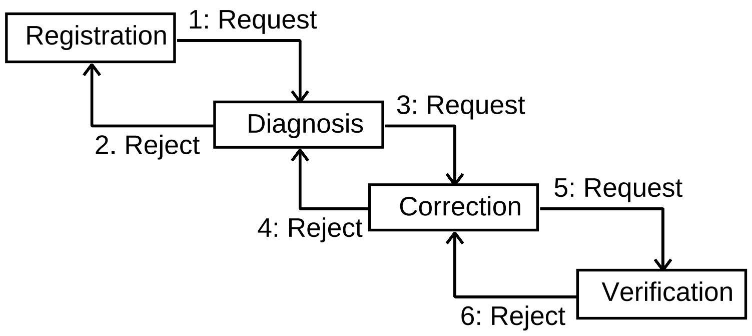


**Figure 8-12:** The standard protocol. In each of the four stages, the actors sends the ‘form’ as a request to the subsequent stage, which in turn might reject it.

A ‘form’ can be sent without any restrictions to roles within each phase, i.e., a spec-team member forwarding to other spec-team members, testers forwarding to other testers. This is illustrated in figure 8-13.

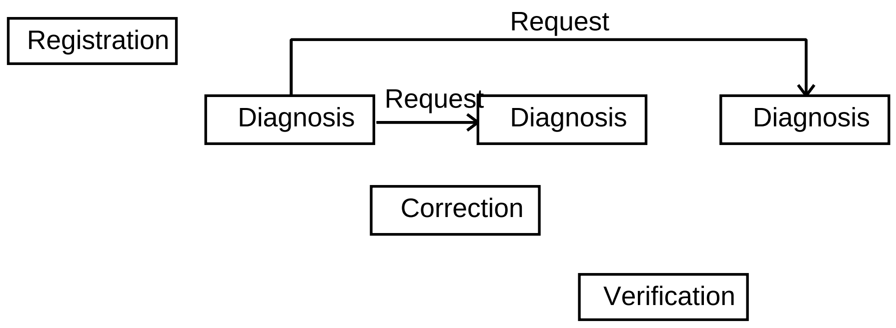


**Figure 8-13:** Within any stage the actors may send the form as a request to others, who in turn can reject the request.

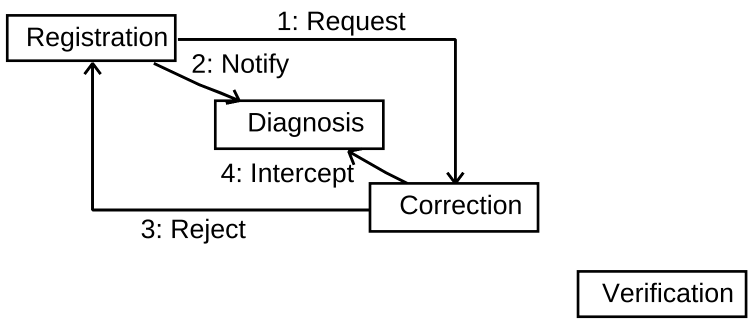


**Figure 8-14:** The strict sequencing of the four stages can be only be broken by the sender skipping a stage downstream (1). The actors in the stage which has been skipped are notified (2). If the receiver rejects, the ‘form’ is passed back to the sender. Actors in the stage being skipped can choose to intercept (4).

A sender (an actor) can choose to skip one stage in the process ‘downstream’. Whenever a receiver is surpassed, he should automatically be notified. A ‘form’ cannot be sent ‘upstream’ unless it has been send as a request, which, if rejected, will be returned to the sender (illustrated in figure 8-14). A surpassed receiver (who has been skipped ‘downstream’) can choose to claim the ‘form’. If, for example, a tester sends a form directly to a software designer, a spec-team member is notified, and he can subsequently choose to intercept the request (see figure 8-14).

The protocol sketched above is an example of how each of the ‘forms’ instantiated in the S4000 could be routed. When it comes to an actual design situation the concrete protocol must be discussed in further details.

## 8.3 Reflections on usability and generality

The aim of this chapter was to illustrate the specification of detailed requirements for a computer-based coordination mechanism. These requirements has been used for sketching a horizontal prototype supporting the coordination of the distributed software testing and correction activities within the S4000 project at Foss Electric. This prototype will be described in chapter 10. The work has, furthermore, been used as input for refining the Concept of Coordination Mechanisms.

The chapter has provided a set of general requirements for computer support of the coordination activities involved in software testing. Based on these, have I outlined a computer-based coordination mechanism supporting the processes of: 1) ensuring that all registered bugs are treated, i.e., all registered bugs are either diagnosed and corrected, or they are explicitly rejected, 2) providing an overview of the state of affairs of testing and correction activities at any given point in time, and 3) distributed registration, centralized diagnosis, distributed correction, and central compilation of verification information. The computer-based mechanism does not provide support for negotiation and communication among the actors, although this is a very relevant requirement.

I have, by presenting a series of interactions between actors and the mechanism, illustrated how a mechanism could fulfill the requirements. The general requirements led to at set of specific requirements concerning the data structures contained in, or accessible to, the mechanism, as well as operations on these structures. In order to outline a computer-based coordination mechanism, subscribing to the general requirements, a number of decisions regarding the detailed requirements and the design, has been made: The structures used for classifying and describing a bug has been redesigned; The design takes into account that several designers is engaged in a specific task; And a major improvement concerning the routing of the information has been introduced.

Another essential requirement for coordination mechanisms is to provide actors with the possibility of assessing the status of the mechanism (cf. section 5.5). The mechanism must offer visibility. Because the Bug-CM protocol stipulates who can send requests to whom at each phase, the actors are provided with various information concerning the state of the mechanism. The Bug-CM provides the actor with information on which software testing phase it is currently in. Whenever a phase is skipped, the actors responsible for this phase is notified, and historical information about the states which the mechanism has been in, and when, prior to being received by an actor, is also provided. This feature also facilitates visibility.

Although the primary purpose of this exercise has been to provide an example, it has also served the purpose of establishing a more general approach to, and concepts for, understanding the process of designing mechanisms supporting the coordination of complex work. It is, therefore, relevant to reflect upon the validity of the requirements in relation to other work settings and situations, even though this problem is far too complex to be discussed thoroughly in this section. It might thus be relevant to relate to other paper-based mechanisms described in field studies similar to mine.

The need for access to conceptual structures reflecting structures in the cooperative work arrangement and in the actual field of work seems to be a general requirement. Other paper-based mechanisms have a similar nature, for example the ‘flight strips’ embodying information on “the goals, intentions and plans of pilots and controllers and their recent actions” (Harper *et al.*, 1989a, p. 10), the ‘augmented bill of materials’ (ABOM) and ‘CEDAC board’ used in engineering design and process planning at Foss Electric (Sørensen, 1994a; Sørensen, 1994b), or the ‘fault report form’ described by Pycock and Sharrock (1994a). Also, some of the basic operations on these structures (e.g. relating structures to each other, routing information, making others aware of changes in the structures, etc.) are similar in the mechanisms analyzed in the mentioned studies: Again is the ‘flight strip’ an obvious example.

The support for defining, applying, and refining different types of classification schemes seems to be a general requirement as well. In the ‘fault report form’ (Pycock and Sharrock, 1994a) classification was essential for the coordination of the work, and the ‘International Classification of Diseases’ analyzed by Bowker and Star (1991) proved to be important for mediating information among specialists. Furthermore, when objects are classified, there seems to be a need for a channel (or structure) through which the classification structure can be negotiated. Andersen (1994) contains an example of how the classification in a 'product classification scheme' is negotiated and refined. Negotiation was one of the requirements excluded in this chapter. It is, however, a central aspect of coordination work, and thus a central requirement for computer-based coordination mechanisms.

The need for stipulation of the work flow seems, not surprisingly, to be a requirement in most of the examples too. Both the ABOM (Sørensen, 1994a) and the 'fault report form' (Pycock and Sharrock, 1994a) included 'build in flow protocols' specified by conventions and organizational procedures. The same goes for the 'Kanban Card' described by Schmidt (1994c). In all these cases, there are also need for actors to be able to deviate from the predefined routing.

Finally, there is the topic of linking mechanisms. An example is the 'construction note' and 'product classification scheme' analyzed and described by Andersen (1994). In much coordination work, activities such as meshing, relating and allocating structures are essential. Mechanisms supporting coordination work should have access (links) to structures within other mechanisms. The need for linking could, in theory, be eliminated by supporting coordination work by one big mechanism. This would, however, conflict with the approach to organizations as multi-facetted and open-ended, and work arrangements as constantly changing open-ended structures (cf. e.g., Schmidt, 1994d).

Like all research work, this dissertation has a delimited focus. Elaborate discussions of methodological issues are excluded, e.g., how an analysis process should be organized, or what is the usability of certain modeling techniques. Chapter 10 will elaborate a bit on the usability of object-oriented methods for modeling coordination aspects of work, and chapter 11 includes some reflections regarding the process of setting up requirements and designing a prototype. I will, therefore, only provide a few methodological reflections here.

Outlining the requirements has, first of all, been characterized by the difficulties in separating the specification of requirements from the design of structures, facilities, and protocols. As in any design process, theories will only facilitate the process. The fundamental design decisions will inevitably be a matter of making design decisions. When reflecting upon the analysis and requirements implicitly and explicitly described in this chapter, it is obvious that there are no clear cut distinction between the requirements for support established and the preliminary conceptual design of a mechanism.

Another problem was related to taking into consideration, how the co-operative work should be supported, when establishing requirements and sketching a design of a computer-based coordination mechanism. Supporting the coordination of distributed cooperative work by means of computer-based coordination mechanisms should, of course, be conducted in consonance with the development of computer support for work itself. The requirements and designs discussed here have not been thought into a general work context.

In relation to the problem of lack of input and considerations regarding the work context, a real-life design situation should certainly also consider, how the requirements, the models and representations developed, and the designs sketched could be evaluated and refined. This must, of course, include a direct interaction with the actors (future users of the system). The requirements and design sketches presented here have been read by some of the software designers involved in the S4000 project at Foss Electric, and the ideas were presented at a meeting where four of the designers participated. At this meeting, the designers agreed to the general ideas and facilities suggested, but we did not have detailed discussions of the functionality, etc. So, although several of the potential future users have read and approved the requirements and design sketches, a much more thorough discussion of functionality, usability, presentation, etc. is, of course, required. This should be based on detailed discussions of illustration prototypes and scenarios with the actors involved in the software testing and correction work.

The requirements and sketches described here are based on several types of information representations. These are a model of conceptual structures, entity-relationship diagrams, actor-system interaction scenarios, verbal descriptions of functionality, an improved ‘form’, and some simple models of the protocol flow. I have only very limited empirically based information on the usability of these. Although it was not explicitly investigated, it was my impression, that the designers at Foss Electric had no problems in understanding these representations. They were, however, software designers and thus used to such representations. I found, personally, the actor-system interaction scenarios very useful. These descriptions illustrated the intended use of the system, and they forced me, as a designer, to carefully consider which types of functionality the system should provide, and how this should be provided. The redesigned form provided a good input for the design of the user interface of the prototype (described in the following chapter). The entity-relationship diagrams proved to simplistic to be used for the database design of the prototype, whereas the considerations and models of the protocol embedded in the mechanism were an important input for the protocol design. Designing the protocol in the prototype without sketches like those illustrated in section 8.2.7 would have been very problematic.

Further methodological reflections are given in chapter 10 and 11, but it is not considered the core of this dissertation.

# 9. A prototype of a computer-based coordination mechanism

"In signs, one sees an advantage for dicovery that is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then indeed, the labor of thought is wonderfully diminished"
(Fredrick Kreiling quoting Leibniz, in an interview in Scientific American, May, 1968)

In collaboration with Thomas Albert, I have developed a preliminary prototype of a computer-based coordination mechanism supporting decentralized bug reporting and routing of the relevant information among the involved actors. The prototype is called BRaHS—Bug Reporting and Handling System. The aim of building BRaHS was to illustrate how the basic components in a coordination mechanism could be expressed in a concrete system, and to establish a basis for discussing how the conceptual framework of coordination mechanisms can be applied in actual systems design. Especially the requirements for visibility, local control, and routing support have been considered in the prototype.

The major input for the design of BRaHS was the set of requirements derived from the field study presented in chapter 8. The overall requirements for coordination mechanisms have, of course, influenced the design, but the driving force for the design was the "real-life requirements".

After the design and implementation BRaHS has been analyzed and discussed in terms of the Concept of Coordination Mechanisms. This chapter describes BRaHS in terms of the functionality, and discusses the prototype in terms of the overall characteristics of coordination mechanisms. Overall requirements for coordination mechanisms concern local control, protocol visibility, linking, etc. This chapter discusses how these are fulfilled, and which objects of coordination work that are reflected in the prototype.

In a design, like the one presented here, a lot of decisions have to be made. Most of these are, of course, debatable. In this chapter, I have chosen only to discuss a few of these: When it is obvious to point at "better", but more demanding, solutions. The purpose of the design was to illustrate, rather than to refine, debate, discuss, iterate, etc. until a “perfect design” was established. Especially the user interface could be polished and improved a lot, even with a minor effort. This has, however, not been considered relevant here. Another comment is important: A basic idea in the Concept of Coordination Mechanisms is that coordination support should be intertwined with support of the actual work, i.e., a coordination mechanism should be embedded as an integrated component in the applications used. BRaHS is, for simplicity reasons, designed as a stand-alone application.

The overall functionality of BRaHS is introduced, and a brief scenario is given. The chapter is concluded with a discussion of the central characteristics of the prototype in terms of coordination mechanisms.

## 9.1 The overall functionality provided by BRaHS

BRaHS was designed in order to illustrate ideas. The effort was mainly put into illustrating how the registration of bugs (including an improved classification of bugs) could be handled, how the overall requirements of malleability and local control could be reflected in the user interface, and how a protocol can be made visible and accessible to the users. BRaHS was designed to support distributed registration of bugs, and automatic routing (forwarding and passing on) of information to the next actor (role) in the tester -> spec-team -> designer -> platform master chain. Since the control of the work flow must be in the hand of the user, BRaHS should also provide access to changing the flow. BRaHS, furthermore, aims at providing the users with facilities for getting an overview of which bugs are reported, their status, etc. To fulfill the requirements, the following overall functionality was designed and implemented:

- Log-in procedure where the user identifies himself to the system and specifies the role(s) he assumes when interacting with the system.
- Three windows for registering information about bugs. Both the testers, the spec-team, and the designers fill in information about a bug.
- A window for specifying the classification of a bug.

- Procedures that, when a user indicates that his registration is finished, automatically routes information on the bug to the next actor (role) in the flow.
- A graphical representation of the protocol specifying the information flow, and access for the user to make certain changes to the protocol.
- Facilities to search for bugs associated with certain characteristics, and for browsing information on registered bugs.
- Procedures to select which project is the current, and to define or change which actors, modules, roles, instruments are involved in a particular project.

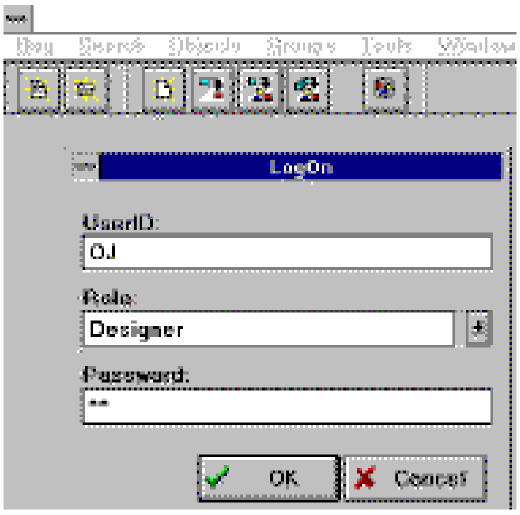
The prototype was implemented as a client/server structure using Borland's Delphi as the application development and runtime environment. It is running on a Windows platform. Each of the listed functions will be described in more detail in the following.

### 9.1.1 Log-in

The log-in window is illustrated in figure 9-1 below. The purpose is to provide access control, and for the system to have an identification of who the user is, and which role(s) he wishes to assume.

The log-in procedure is very simple and primitive. If the system is going to be used in a real work situation this procedure should be redesigned carefully. First of all, a relevant level of security should be reflected, the fact that the user has to specify his role(s) beforehand should be reconsidered. The actors will frequently switch between several roles in many work situations. A designer might, for example, be a tester in one situation and a designer a few seconds later. Thus, explicit switching between roles should be reconsidered. This is only supported through the possibility of indicating more than one role at log-in, and by providing a command button for changing role in the icon button row. This will probably be insufficient and disturbing for actors in real work situations.

After having logged-in the user gets access to a number of menus and functional icon buttons (see figure 9-2). The menus will be described later. The icon buttons give access to open or close a particular project. To create a new project (a new database containing bug reports) is considered a facility that not all users should have access to. This is, therefore, done via another application offering facilities for initiating new actors, roles, protocols, classification structures, etc.



LogOn

UserID: OJ

Role: Designer

Password: **

**Figure 9-1:** The log-in window in BRaHS.

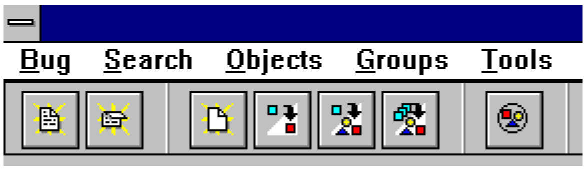
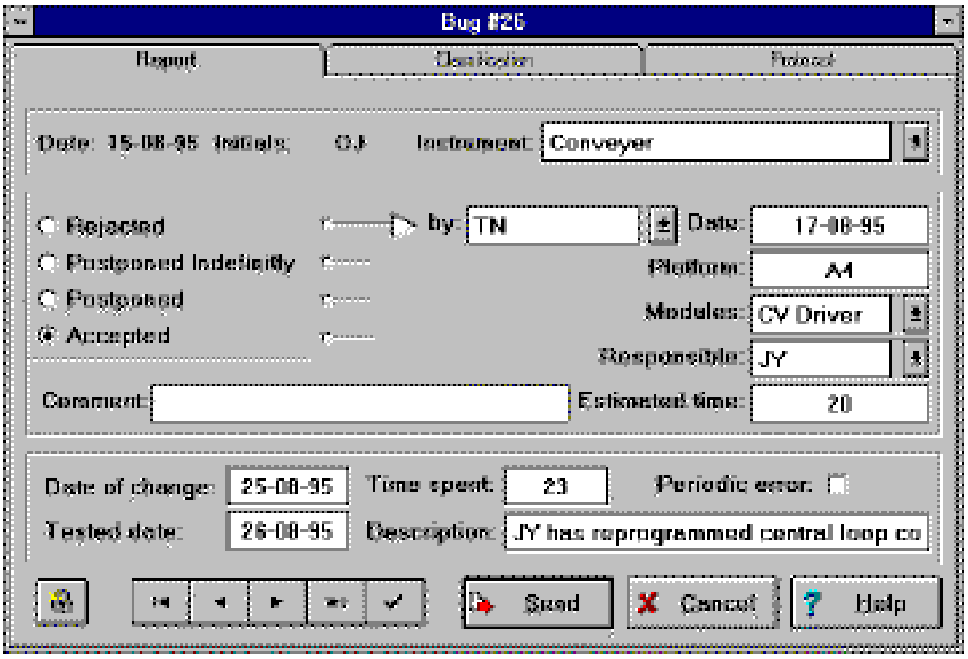


**Figure 9-2:** The main menu, and icon buttons for opening and closing access to specific project databases, create new bug reports, search for specific bugs in the database (three types of search), and for changing the role profile specified.

When a project (and thus a database containing registered bugs) is opened the next four icon buttons give access to functions creating a new empty bug form, searching for a particular bug, searching for bugs fulfilling certain characteristics, and establishing a list of all registered bugs from which the relevant can be chosen. The right most button is used for changing the role.

### 9.1.2 Registration of information on bugs

Registering data about specific bugs is essential, both in the studied testing and correction work and in the prototype. Several different roles are registering data about the bugs: Testers register information the occurrence and importance of a problem. The spec-team register information on importance, deadlines, who is responsible, and estimated correction time. The designer adds information about which corrections have been made, modules and files affected, and time spend on the correction.


Bug #26

Report | Classification | Protocol

Date: 15-08-95 Initials: OJ Instrument: Conveyer

( ) Rejected
( ) Postponed Indefinitly
( ) Postponed
(●) Accepted

by: TN Date: 17-08-95
Platform: A4
Modules: CV Driver
Responsible: JY
Comment:
Estimated time: 20

Date of change: 25-08-95 Time spent: 23 Periodic error: ☐
Tested date: 26-08-95 Description: JY has reprogrammed central loop co

**Figure 9-3:** The first page of the three layered index card used for registering data about bugs. The report is separated into three chunks: one to be filled in by the tester reporting a bug, one to be filled in by the spec-team diagnosing the problem, and one to be filled in by the designer, when having dealt with the problem.

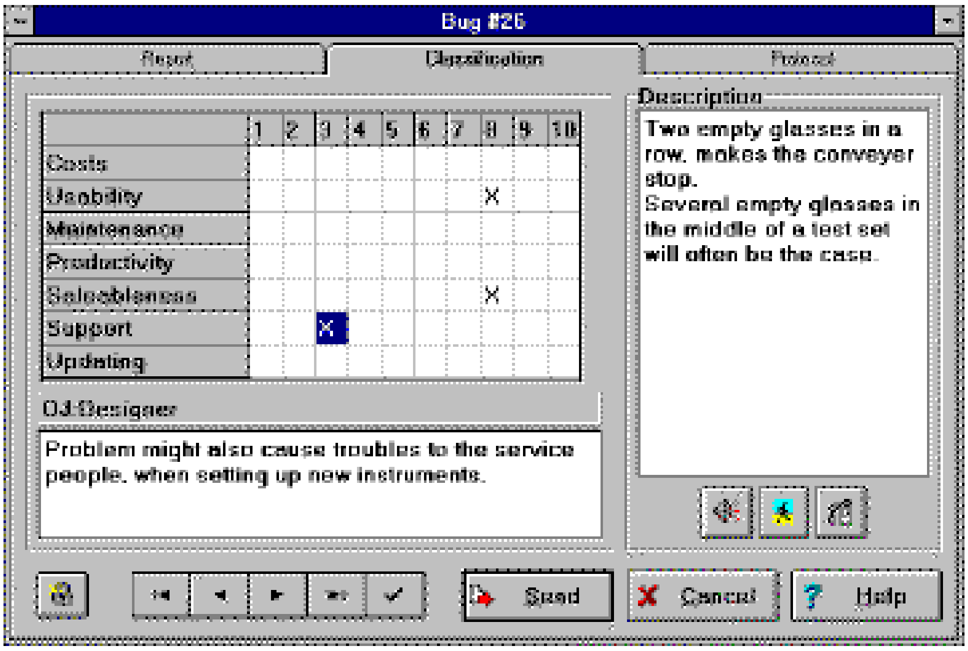
To provide this functionality, BRaHS includes a registration window organized as a three layered “index card” as illustrated in figure 9-3. The content of the registration window is the same for the tester, the spec-team, and the designer, but in each situation fields not relevant are ghosted.

The indexes on the top are used for shifting to windows for classifying bugs or getting an overview of the flow of the protocol. These will be described in the following sections. The entry fields are grouped into three: The upper group is used by the testers to register date, who they are, and which instrument they used when testing. In the middle group, the spec-team enters (or selects from selection lists) whether or not the problem reported can be accepted as a bug, in which platform period it must be corrected, which module has caused the problem, who is responsible for correcting the problem, and the estimated correction time. A general free-text comment can also be entered. The group of fields in the bottom are filled in by the designer after the problem is corrected. He specifies when the bug has been corrected and tested, and the amount of time spent on the correction. Furthermore, he enters a free-text description of the corrections made, and which modules and files are affected. This field also contains an identification of who has conducted the actual correction work. A more sophisticated solution would be to divide the description field into several fields (selection-lists) in which the designer can indicate affected modules, affected files, types of correction, expected implications for other modules, etc.

The buttons at the bottom is used for navigating and controlling the interaction. The lock is used to lock the current record so parallel updates are avoided. The forward and backward keys are used to browse in the selected bug reports. The check mark button is used for saving the registered information. Pressing the Send button indicates that the user has finished his registrations, i.e., the window is closed, and the next actor in the work flow is notified. The Cancel and Help functions should be self explanatory.

### 9.1.3 Bug classification

A proper and detailed classification of the problem is very important. As required in chapter 8, bugs are classified according to two dimensions: One regarding which aspects of the system it concerns, e.g., the usability or the maintenance of the product. The other regards the importance of getting the problem fixed. The window offering the classification is shown in figure 9-4 below.



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Costs | | | | | | | | | | |
| Usability | | | | | | | | X | | |
| Maintenance | | | | | | | | | | |
| Productivity | | | | | | | | | | |
| Saleableness | | | | | | | | X | | |
| Support | | | X | | | | | | | |
| Updating | | | | | | | | | | |

OJ-Designer

Problem might also cause troubles to the service people, when setting up new instruments.

Description

Two empty glasses in a row, makes the conveyer stop.
Several empty glasses in the middle of a test set will often be the case.

**Figure 9-4:** The window used for classifying a bug. The free-text description field on the right side of the window contains a general description characterizing and describing the problem in further detail. Both the tester, the spec-team, and the involved designers can enter as many ‘x’s in the matrix as they wish. For each ‘x’ it will be identified who has entered it (The designer OJ has entered a level 3 for ‘support’). An optional additional comment can be added to each of the ‘x’s. These are shown in the description filed below the matrix.

Due to the overall requirement of malleability both the list of concerns and the importance scale can be re-specified by the users them-selves, i.e., in a project were other dimensions are required the users can specify which dimensions should be used. For example if it is a completely different product from those previously tested. These corrections or expansions can also be made in an ongoing project. Corrections to the classification scheme used in an ongoing projects have been considered rare and important to have control off. Maintenance of the classification dimensions are therefore done through another application (the setup-application) only.

All actors involved in reporting, diagnosing and treating the problem can at any time include a more detailed free-text description of the problem. The navigation buttons are the same as those described in the previous section.

### 9.1.4 Routing facilities

Automatic routing is designed so that, when a user presses Send in order to finish his registration, the system updates the database, and checks the protocol. The protocol contains information on which role must receive the form (the information on the bug) next. If this role is logged-in, the actor receives a message saying that bug number x has been send for him. He can open it immediately and see the information registered so far, or he can decide to look at it later. If the receiver is not logged-in, the system sends him a notification, when he logs-in to the system.

### 9.1.5 Changing the protocol

Visibility of the protocol embedded in the mechanism, and possibilities for the local users to deviate from the protocol, were two essential requirements. BRaHS deals with these by including an index card layer which graphically presents the structure of the protocol, and the possible changes to it.

The idea is to provide the users with access to an overview picture of the protocol in which the roles and flow are represented by icons (cf. figure 9-5).

By clicking an icon (which represents a role), the user can get further information on who has this role for the current bug. If, for example, the user presses the gearwheel icon information on the responsible designer will pop-up next to the icon. Local control is implemented by allowing the user to chose one or more of the allowed deviations (ghosted flows). This is probably a controversial implementation of local control, since the users are only allowed to make certain pre-specified deviations. An alternative solution would be, to let the user draw new arrows between every two icons. The degree of freedom is limited in BRaHS for simplicity reasons. This should, of course, be considered further in a real-life design situation.

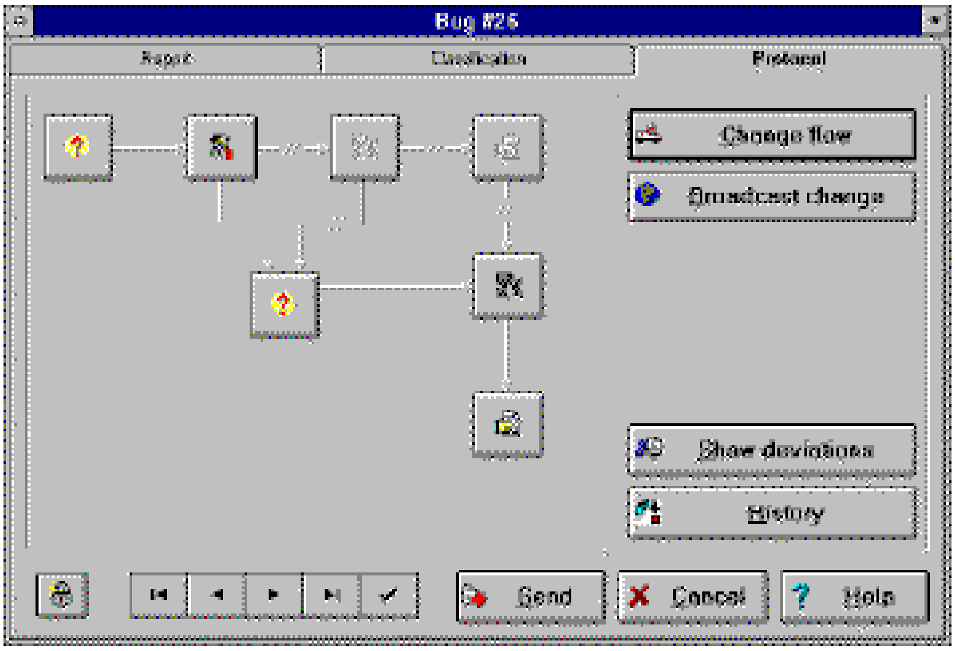


**Figure 9-5:** The window used for getting an overview of the protocol flow, and for making changes to the flow. Each bug form can have its own flow, i.e., there is a protocol for each. Each icon in the diagram represents a role: The question-mark is a tester, the tool-set is the spec-team, the gearwheel is a software designer, and the document is the platform master. The non ghosted icons illustrate the current protocol, and the arrows between them indicate the direction of the flow. The current status is marked by a red ring around the actual icon (invisible in the black/white picture). The 'Change flow' allows for manipulations of the protocol. 'Broadcast change' sends notifications to other actors involved in the treatment of the currrent bug, indicating that the protocol has been changed. 'Hide/Show deviations' removes/ adds the rings indicating the default protocol, and 'History' gives the user access to see how the protocol has previously appeared.

The flow is changed by first pressing the Change flow button. Then the protocol is changed by clicking on the components. A ghosted icon or broken arrow can be made active (non ghosted) and vice versa. The corrections are completed via the check mark button. The consistency is checked, and the protocol is saved.

When a new version of BRaHS is instantiated—e.g., a new project is established—a default protocol is defined at the same time as the actors and roles are defined. All new bugs will be routed according to this default protocol until a user changes it for a particular bug. The default protocol is highlighted if Show deviations is activated. This can be turned off by pressing Hide deviations. Thus the user have access to see both the actual protocol, possible deviations, and the default flow serving as the standard protocol.

The user making changes to a protocol can chose to inform the other actors involved by pressing the Broadcast change button. Then all actors involved in the treatment of the current bug will receive a notification indicating that the protocol for bug number x has been changed. They can then request to see the bug information and thereby get access to see the changed protocol.

All users can browse previous versions of the protocol related to the treatment of a particular bug. If the History button is pressed, a series of new windows pops up. Each of these contain a diagram of an older version of the protocol for the current bug form. These diagrams can, of course, not be changed.

### 9.1.6 Searching and browsing

BRaHS also includes facilities for finding information on specific bugs, or on bugs having certain specific characteristics. Three different search facilities are implemented:

The 'simple search' allows the user to find a particular bug form by entering the bug number. Information on the bug in a three layered index card window similar to those shown in figure 9-3, 9-4, and 9-5 is then shown.

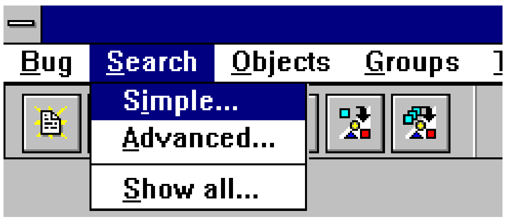
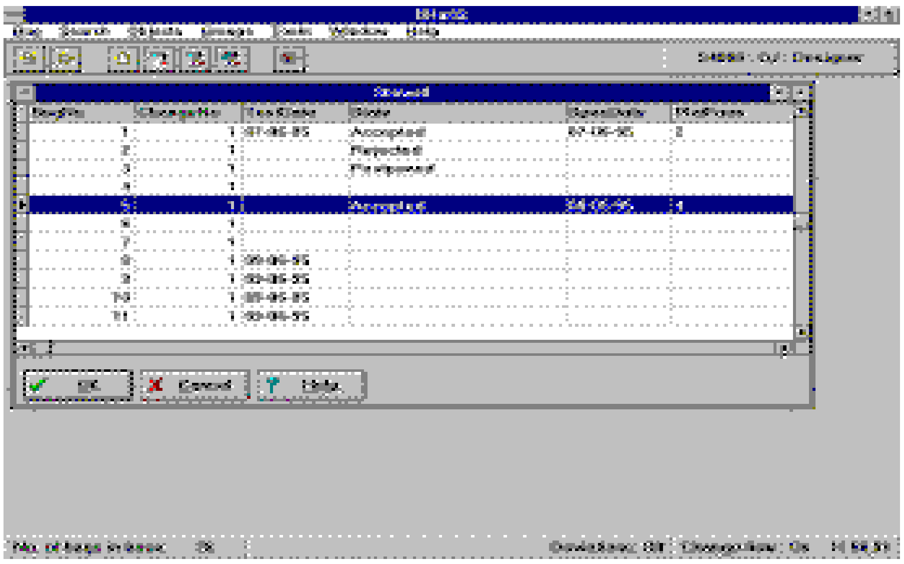


**Figure 9-6:** Three different search facilities are implemented. These can be accessed either via the search menu or via the three right most icon buttons on the opening screen.

Another quite simple search facility is the ’Show all’. This results in a list of all the bugs registered on the current project. For each bug all information registered in the database is listed, i.e., the bug number, an indication of whether the protocol has been changed or not, the acceptance state, the test date, the state of the bug, diagnose date, the status of the treatment, an identification of the tester, the designer involved, etc. An example of such a search is illustrated in figure 9-7 below.

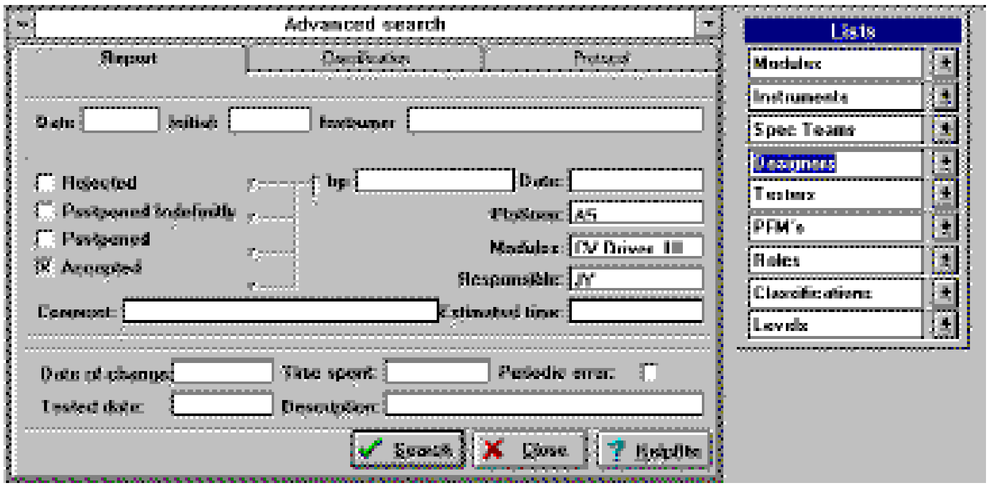


| BugNo | ChangeNo | TestDate | State | [illegible] | [illegible] |
|---|---|---|---|---|---|
| 1 | 1 | 07-06-95 | Accepted | 07-06-95 | 2 |
| 2 | 1 | | Rejected | | |
| 3 | 1 | | Postponed | | |
| 4 | 1 | | | | |
| 5 | 1 | | Accepted | 04-06-95 | 4 |
| 6 | 1 | | | | |
| 7 | 1 | | | | |
| 8 | 1 | 09-06-95 | | | |
| 9 | 1 | 09-06-95 | | | |
| 10 | 1 | 09-06-95 | | | |
| 11 | 1 | 09-06-95 | | | |

No. of bugs in base: [illegible]

**Figure 9-7:** The ‘show all’ search results in a list of all bugs registered on the current project. The user can, by clicking on a specific line, get detailed information on the specific bug. The upper right corner of the screen contains information on the project (here S4000) and on the actual user and his current role(s).

The most complex search is the one called ‘advanced’ in the menu. When this is selected the user gets a window as showed in figure 9-8 in which a search profile can be specified. The profile can include all kinds of combinations of all the fields registered for all bugs. That is, the user can search for bug reports to be corrected in a particular platform period, bug reports for which a specific designer is responsible, bugs which it has taken more than 15 hours to correct, bugs having a particular classification profile, bugs treated according to a certain deviation from the standard protocol, etc.



[✓ Accepted]

Platform: A5

Modules: CV Driver III

Responsible: JY

**Figure 9-8:** In the 'advanced search' all kinds of search profiles for all fields registered for the bugs can be combined. All fields can contain a combination of several selections or keywords. This is the main reason, why all selection lists are shown in an additional window. It minimizes the requirements for how much the users must remember on his own. The search profile in the example will result in a list similar to the one shown in figure 9-7 containing all registered bugs to be corrected in platform period A5 in the CV Driver module by designer JY.

## 9.1.7 Set-up of projects, actors, modules and roles

As mentioned earlier, a new project having its own actors, classifications, module structure, and default protocol must be specified in another application, which the users usually have no access to. Local corrections to the way BRaHS is executed can, however, be made. New actors, modules, and instruments can be added to the lists, or the existing actors, modules, and instruments can be re-specified or removed from the lists. This is done in the Objects menu.

The situation is very often that several actors have the same role, that one actor can have several roles, and that it will be obvious to group the modules and instrument components. Facilities for this are also provided via the Groups-menu.

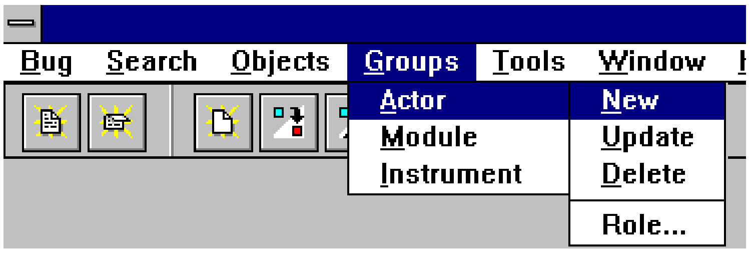


**Figure 9-9:** The menu used to define new, re-specify, or remove groups of actors, or groups of modules, or groups of instruments previously specified. The menu shown here, furthermore, includes a command for relating a role to a (group of) actors.

## 9.2 A scenario for the use of BRaHS

The previous section gave a rather detailed overview of the functionality provided by the system. In order to relate this to a use situation, this section will briefly describe a small simple scenario of a possible use of BRaHS.

Mr. Tester is involved in the test of the software controlling the pipette of System 4000. He discovers the conveyer stops if the pipette meets two empty milk test glasses in a row. He checks the requirement specification and realize that this is not a correct behavior. It should be reported. Mr. Tester logs-in to BRaHS and identify himself as tester. He opens the S4000 project via the ‘Bug’ menu and creates a new bug form by pressing the ‘New bug’ icon. An empty bug registration window is opened (cf. figure 9-3). Mr. Tester now fills in information on the problem. He classifies it according to support as ‘3’ (not too important) and according to usability and salability as ‘8’ (important). He writes a short notice in the comments field on classifications, saying that the conveyer will often contain empty glasses, and two glasses in a row will be quite common (cf. figure 9-4). He presses the Send button, and the window is closed.

Two days later the S4000 spec-team have their weekly meeting. Mr. ST-member logs-in to BRaHS, and searches for bug forms having the status registered. The first on this list is the bug form filled in by Mr. Tester. Mr. ST-member opens it and reads the description. The spec-team agrees, that this is a problem related to the *pip-control* module, which Mr. Designer is responsible for. Thus Mr. ST-member tip of the ‘accepted button', and selects the pip-control module and Mr. Designer from the selection lists.

The spec-team, furthermore, agrees that it is important to have this bug corrected during the current platform period, so Mr. ST-member enters an A4 in the platform field. Estimating the correction time causes problems. The field is left open. The spec-team does not add anything to the classification. The default protocol prescribes that the form on a diagnosed bug should be send to the tester, before it is routed to the designer, cf. figure 9-10. But in this situation, the problem is quite clear. Mr. ST-member change the protocol so that the form is directed directly to the actor responsible, Mr. Designer.

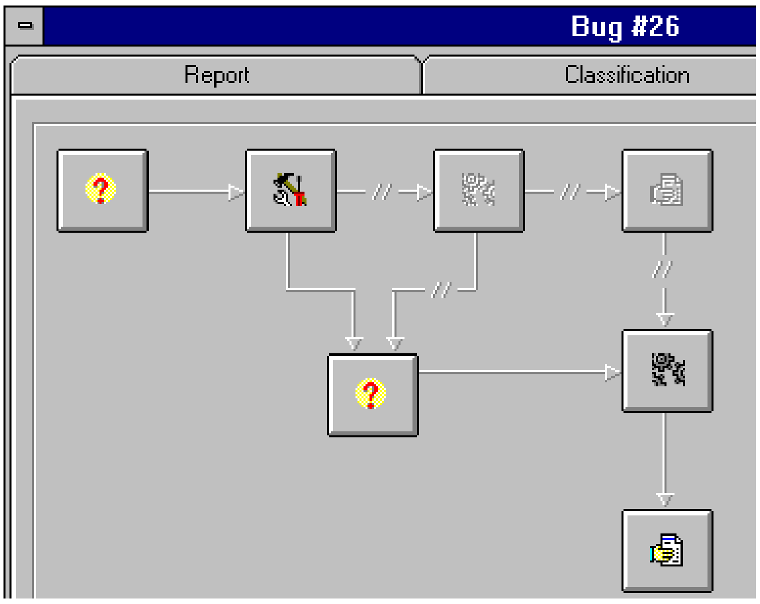


**Figure 9-10:** The default protocol prescribes, that the form on a diagnosed bug should be send to the tester, before it is routed to the designer: The arrow goes 'down' from the spec-team (the tools icon) to the tester (the question mark). This is not needed. Mr. ST-member can change the protocol, so that the flow follows the icons in the upper line, i.e., it goes directly from the spec-team to the designer (the gearwheel icon), and further to the platform master (the hand and paper icon).

When Mr. ST-member presses Send the database is updated, and a notification is send to Mr. Designer. He is a curious person. He opens the bug form right away, and reads the description. He realizes the problem, and that he needs to correct it soon. He estimates the correction to take 12 hours. This is entered and the window is closed. A week later, Mr. Designer has time to correct the problem. When he has finished doing this, he logs-in to BRaHS, opens the actual bug form, enters the date for changing and testing, and fills in that he has spend 23 hours on correcting it. Furthermore, he writes a note saying that only the code in the *pip-control* module has been affected, and that he has changed the stop conditions so that an indefinite number of empty glasses is accepted without stopping the conveyer. The form now contains all the information previously illustrated in figure 9-3. Mr. Designer presses the Send button to indicate that he has dealt with it. The database is updated, and a notification is send to the platform master, Mr. PM. Two weeks later, the day before the next integration period, Mr. PM logs-in to BRaHS and searches for all bugs having the status of corrected, but not yet verified. As a preparation to the forthcoming integration period Mr. PM prints the description of each of bug on the list. He is now ready to check if all the problems have been dealt with properly.

## 9.3 The prototype considered a coordination mechanism

Although ideas and concepts from the conceptual framework described in chapter 5 have provided input, the current design of BRaHS mainly reflects the requirements specified in chapter 8. It is thus relevant to discuss BRaHS in terms of the Concept of Coordination Mechanisms.

### 9.3.1 BRaHS as a protocol

A coordination mechanism is defined as a protocol that, by encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, stipulates and mediates the coordination of distributed activities so as to reduce the complexity the coordination work. A computer-based coordination mechanism is a computer artifact that incorporates aspects of the protocol of a coordination mechanism so that changes to the state of the mechanism induced by one actor automatically are conveyed by the artifact to other actors in an appropriate form (cf. section 5.3). Let us elaborate a bit further on how these characteristics are reflected in BRaHS, and how this affects the use:

(1) BRaHS contains an incorporated *protocol* that stipulates aspects of the coordination of the distributed testing, diagnosing, correction, and verification process. BRaHS encompass a set of prescribed procedures for how the information should be routed from each actor (role) to the next in line. The routing is conducted automatically and supports, thus, the coordination in two ways: First, the information is handed over—and the receiver is automatically notified—and second, the receiver is specified in advance, so that the user do not need to consider who to address the information for. To some extent BRaHS allows the user to overrule the flow (cf. section 9.2.5). These situations will require more coordination by the user. On the other hand, it is an overall requirement for a coordination mechanism that it supports local control.

    Using BRaHS requires that a set of conventions for its use is established among the actors: The users must use the system. Since BRaHS is not directly related to the field of work—the software being tested—there are, amongst other things, no automatic registration of data. It is also important to establish conventions for how to use the classification scheme, negotiate an estimate, etc. The need for use conventions exists for all kinds of artifacts used by more than one actor.

(2) The structures and information in BRaHS are conveyed by a *symbolic artifact*. The PC running BRaHS, the code stipulating the flow, and each instance of the database containing information on registered bugs in a specific project are symbolic artifacts. They are, thus, persistent to changes in the actual field of work accessed, i.e., they can be accessed by the involved actors independently of a particular moment in the work flow, and independently of a particular actor.

(3) To elaborate a bit further on characteristics of a symbolic artifact, we can say that BRaHS is *distinct* from the state of the actual work conducted. Changes to the state of the testing, diagnosing, correction, and verification work are not automatically reflected

in changes the content of the data stored in the databases. And vice versa: Changes to the content of the data stored in the databases, made by one of the actors, will not automatically be reflected as changes to the code or the work processes.

(4) BRaHS *mediates* information relevant for handling the required coordination of the distributed activities. First, BRaHS mediates the information registered for a bug from one actor to others, either as a notification to a particular actor or upon request. Second, changes to the protocol are conveyed to other actors as notifications (if the originator chooses to inform other involved actors, cf. section 9.2.5), or upon request. These are visible as changes to the protocol for the particular bug. It is the users who decide, whether change information should be distributed or not. Information on changes to the state of execution of the protocol will not in all situations be obtrusively mediated to the other actors.

(5) All registrations and presentations of information in BRaHS are based on a *standardized format* of the data and the protocol. The format can, thus, be seen as providing affordances to the coordination work: The pre-specified structure supports the actors in filling in the correct and required information, and helps them to search and browse for specific information. The pre-specified structure will also impose constraints on coordination work by not allowing the user to, for example, define his own module names, etc. while describing the diagnose of a bug. BRaHS has no validation of the content of entry fields as it is implemented now.

### 9.3.2 Objects of coordination work reflected in BRaHS

A central structure provided by the Concept of Coordination Mechanisms is the model of dimensions of coordination work (cf. section 5.4). When the field study was conducted, this structure influenced the analysis and the structuring of data, and a model for the essential objects of coordination work and related operations was defined (cf. figure 6-10). Discussing BRaHS in terms of coordination mechanisms should also include reflections on this structure.

The model distinguishes between nominal and actual objects of coordination work. Nominal covers structures having a potential status, i.e., resources that can be allocated, task that should be conducted, etc. Actual includes structures in ongoing actions or allocated for specific tasks, obligations, etc. This distinction appears in BRaHS as well, for example in the database structure. Some structures in the database can be regarded as nominal structures, for example actors and modules as roles and potential responsible modules. When a relationship structure to specific bug is established (during the diagnosis conducted by the spec-team) the structures become actual, i.e., they become 'committed actor' and 'module to be corrected' (a thorough and detailed discussion of how the database structure reflects the actual-nominal distinction is given in Carstensen and Albert, 1995).

The nominal-actual distinction is also visible in the user interface. When a user establishes a new actor or module group, or he is using the selection lists, the structures used are nominal, but when the user browse or search for specific information on who has which obligations in relation to the next platform period, the structures he is facing are of an 'actual' nature.

If we look at the objects of coordination work identified in the field study (cf. figure 6-10) most of the nominal are reflected in BRaHS: 'Role' and 'Human resource' can be recognized immediately in BRaHS, and 'Task' is implicitly reflected since all registered and accepted bugs can be considered a task to be conducted. Regarding 'Conceptual structures' the modules and instruments could be considered as such. Informational, material, technical, and infrastructural resources are not included in BRaHS. Regarding the actual structures a few of them are identifiable. 'Committed actor' is reflected for all accepted and diagnosed bugs. BRaHS does not register any specific ongoing activities, neither activities related to the testing and correction work itself, nor activities related to the coordination. Structures described as 'Activity' and 'Actor-in-action' are thus not included. The same goes for the different resources. The filed information on registered bugs contains certain aspects of the state of the field of work. These are actual conceptual structures.

### 9.3.3 The fulfillment of the overall requirements for coordination mechanisms

Let us also consider how the following overall requirements for computer-based coordination mechanisms, established in section 5.5, are reflected in the prototype: local and/or temporary changes, global and lasting changes (including a language for re-programming), control of propagation of changes, visibility, partial definitions, the reflection of structures of the field of work, and linkability. Functionality related to these requirements has already been discussed in this chapter, but to complete the picture, let us go briefly through each of the requirements, and describe how these are reflected in BRaHS and consider some potential alternatives:

*Local and/or temporary changes:* Actors should have control of the execution of the mechanism in order to cope with unforeseen contingencies. BRaHS allows actors, roles, and modules to be grouped and reorganized during execution, i.e., without need for a new instance. And the protocol can be redefined individually for each bug registered. It is, however, only changes included in the pre-specified set of deviations that are accepted (cf. section 9.2.5).

A redesign of BRaHS should consider, how the actors can be supported in redesigning a running protocol in a more flexible manner. It would, for example, be obvious to consider, how to allow the actor to connect the nodes (icons in the graphical representation) arbitrarily. Furthermore, a facility through which the user could selected any nodes in the protocol, and restart the process from there, would improve the flexibility.

*Global and lasting changes,* and a language for re-programming: Facilities for actors to specify and re-specify the behavior of the mechanism should be provided. This requirement is partly fulfilled in BRaHS. The actors can (re)specify the module-structure, involved actors, roles, classification schemes, etc. via the setup-application. Through BRaHS itself actors, roles, and modules can be grouped and reorganized. There are, however, no means for identifying completely new conceptual structures, e.g., references to the design specifications used.

An alternative design would have been to: 1) Establish a more coherent set of conceptual structures for software testing including, for example, the software architecture and the platform oriented work plans; 2) Design a set of protocol components providing more flexible building blocks for the users designing the protocols; and 3) Allow the users to save a running protocol as permanent.

*Propagation of changes:* Actors should have access to control the propagation of changes to the specification of the behavior of the mechanism. BRaHS is based on a pre-specified set of possible changes to the protocol. The user has full control of the propagation of the possible changes. Regarding the propagation of information on changes, users of BRaHS can decide to inform all actors involved in the treatment process of a specific bug.

If a more flexible and freely changeable protocol is offered, problems of how to ensure consistency, and how to let the changes propagate, should be reconsidered.

*Visibility:* The behavior (protocol) of the mechanism must be accessible and manipulatable to actors. This should be at a proper semantic level of coordination work. The protocol used by BRaHS is visible and accessible to actors in terms of a diagram consisting icon representations of the involved roles, and arrows indicating the flow (cf. section 9.2.5). Certain pre-specified changes to the protocol are provided as manipulations on these graphical items.

An overall aim of the BRaHS design was to illustrate a possible visualization of the protocol. Several alternatives should be considered. An important question to think of is whether the roles should be 'visible' in the interface. Roles are useful for describing work settings, but actors rarely think of themselves in terms of roles when working.

*Partial definitions:* A coordination mechanism should allow the actors to let attributes to be left un-specified. As BRaHS is designed now, there are no restrictions on what must be filled in by one actor before he is allowed to pass it on to the next. There are, however, no features supporting other mechanisms to infer and specify the un-specified attributes.

In a redesign of BRaHS, it should be carefully considered which fields that has to be filled in during registration, which fields that can be filled in through requests to other mechanisms (applications), and which fields other actors can be requested to fill in.

*Reflection of structures* of the field of work and the work arrangement: A coordination mechanism should reflect pertinent features of the field of work. BRaHS is based on conceptualizations of the field of work, and of structures of the work arrangement (cf. section 9.4.2). These conceptualizations were used by the users themselves, when characterizing their work and coordination work activities.

As mentioned in previously, it would be obvious to improve BRaHS by including conceptual structures reflecting the software architecture, the actors involved, and the work plans.

*Linkability:* Facilities for establishing links to other coordination mechanism within the wider organizational context should be accesible. BRaHS does not have any (computer-based) linking to other mechanism at the moment. There are, of course, links to the work plan schedule, to the module architecture, to actor lists, and to the directory structure supporting the software integration. These links are, however, all based on one or more human actors (users of BRaHS) who manually export information from BRaHS to other mechanisms, and vice versa.

The central ideas in the Concept of Coordination Mechanisms of linking should be included in BRaHS. This calls for a complete redesign, both of the data structures used, and of the way the functionality is partitioned. Linking has not been explicitly addressed in the current design of BRaHS.

## 9.4 Evaluation of the prototype

An open question needs to be answered: How was the prototype evaluated? First of all, this has not been done sufficiently if we consider BRaHS the first prototype of a system that is going to be implemented at Foss Electric. BRaHS has been evaluated with the purpose of getting input for the discussion given it this chapter, not with the purpose of getting input for a concrete redesign (or refinement). The evaluation was conducted as an ongoing iterative process during the design, and it basically contained three different inputs:

(1) During the design the ideas and preliminary mock-ups and sketches were presented at several meetings. The participants (colleges from both Risø and the Esprit BRA project COMIC) were familiar with both the results from the Foss Electric field study and the Concept of Coordination Mechanisms. Ideas were discussed, changes suggested, possible scenarios considered, etc. BRaHS was also compared with another prototype, Gordion

(Tuikka and Sørensen, 1995) which is also based on the field study descriptions of the bug form mechanism.

(2) The prototype was presented to three of the software designers at Foss Electric that had been involved in the S4000 project. This demonstration had two purposes: To get some feed-back on the design of BRaHS from some of the potential users, and to provide input for a design of project support systems which Foss Electric is conducting at the moment. The prototype presented was identical to the one presented in this chapter.

(3) In a two-day heuristic evaluation session (cf. e.g., Nielsen, 1994) Liam Bannon, University of Limerick, evaluated the prototype. Liam Bannon is a highly respected researcher within both Human-Computer Interaction and CSCW. The comments and discussions included aspects like: user interface design, scenarios for possible use (and what problems this could cause), discussions of the basic assumptions behind the design, and discussions of the model of the cooperative work setting hidden in the prototype, etc. The prototype used for the demonstration and evaluation with Liam Bannon was identical to the one presented here.

The data from these three types of sessions have been analyzed and used for identifying things that could be corrected. For example, the database structure were redesigned completely after one of the meetings mentioned in bullet 1 above. Findings from the two last mentioned evaluations have mainly been used as input for the discussion of BRaHS presented in the end of this chapter. More thorough analyses can, of course, be made from such evaluations. In a real-life design situation, a more well-structured evaluation should take place, involving the future users much more actively. This has, however, not been considered relevant in the context of this dissertation, and for providing the required input.

Despite the limited evaluation, I will claim, that BRaHS is an illustrative example of a computer-based coordination mechanism. There are, however, still a long list of questions to consider.

# 10. Object-oriented modeling of a coordination mechanism: An experiment

We needed to grasp, to understand in depth, the problem domain—and we needed to do it fast as quickly as possible. Of course, the situation is less extreme for many system analysts—but even if you happen to be a subject matter expert as well as an analyst, you still need tools to effectively communicate your expertice to others [...] OOA—Object-Oriented Analysis—is based upon concepts that we first learned in kindergarten: objects and attributes, wholes and parts, classes and members. Why it has taken us so long to apply these concepts to the analysis and specification of information systems is anyone's guess—perhaps we've been too busy 'following the flow' during the heyday of structured analysis to consider the alternatives.
(Coad and Yourdon, 1991)

As briefly discussed in chapter 8, the transformation from the observations conducted in the field and the description of these (described in chapter 6) to a set of requirements and design sketches (cf. chapter 8) was quite complicated. There was no simple unambiguous procedure or course to follow when analyzing the findings with the purpose of identifying needs for support and the actual requirements for a computer based coordination mechanism. These problems make considerations on methodological support highly relevant. Although the Concept of Coordination Mechanisms can support the conceptualization of field study findings and inspire the requirement engineering process other means are called for. Following Hughes (1993) it can be stated that, in order to bridge the gap between field studies of cooperative work and design of coordination mechanisms, there is a need for suitable methods and techniques facilitating the modeling activities. Albeit, methodological considerations are not core issues in this dissertation I have, in collaboration with Birgitte Krogh and Carsten Sørensen, conducted a small experiment investigating the question of, what are the possibilities and limitations for modeling computer-based coordination mechanisms using an object-oriented analysis method? To explore this we have applied an object-oriented analysis methodology for modeling the bug form based coordination mechanism identified in the field study, previously reported in Carstensen *et al.* (1995a). This chapter describes the experiment and its overall results. When reading this chapter, it is important to notice, that the experiment described was conducted before designing the BRaHS prototype described in chapter 9. The analysis presented here should be considered a re-analysis of the data and descriptions presented in chapter 6, i.e., the models presented here are analysis models only (cf. figure 10-1). The intention of the experiment was to investigate whether the process and resulting models appear to be useful as input for design of computer-based coordination mechanisms.

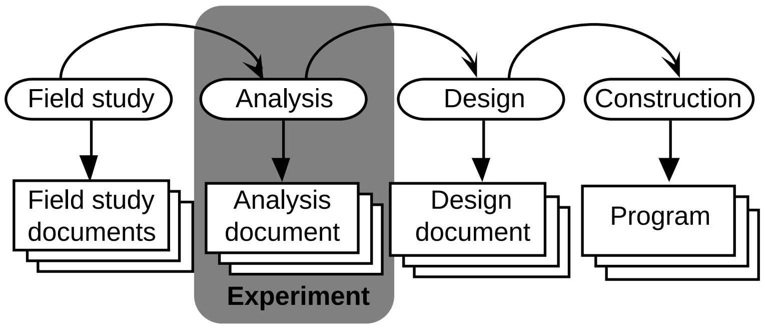


**Figure 10-1:** The scope of the experiment documented here is analytical modelling. The figure is from a forthcoming improved version of Carstensen *et al.* (1995a).

The first obvious question to answer is, of course, why choose an object oriented approach? First of all, applying object-oriented techniques seems to be one of the very important trends within the research field of software engineering these years, and it has become increasingly popular in the software industry too (Monarchi and Puhr, 1992). Although the limitations of using object-oriented techniques for analyzing have been critiqued (e.g., Embley *et al.*, 1995; Lauesen, 1995), the popularity itself makes it relevant to check out and discuss, if the approach can be supportive with respect to building computer-based support of coordination work. One aspect of the critique concerns problems in grasping the procedural and dynamic aspects of the domains modeled (Monarchi and Puhr, 1992). Especially the aspect of dynamism is important in relation to coordination.

We chose a methodology that explicitly claims to address this problem. The methodology chosen is discussed in section 10.1 below.

The second—and in the context of this dissertation much more important—argument for experimenting with object-oriented techniques relates to the very nature of the coordination mechanisms observed, both in my and other field studies. As discussed in section 5.8 and section 8.3, and illustrated in chapter 6, coordination mechanisms can be regarded as individual detached mechanisms each supporting a limited set of coordination activities. These individual detached coordination mechanisms are, however, usually linked to other coordination mechanisms in order to get access to required information, or to trigger an action in another mechanism. The nature of detached, but interrelated, components seems to fit nicely into the basic structures provide by object-oriented methodologies. To phrase it differently: The central techniques of abstraction, encapsulation, association structures, etc. (cf. e.g., Coad and Yourdon, 1991) in the object-oriented paradigm makes it relevant to consider object-oriented techniques for modeling the aspects of coordination work to be computer supported.

Finally, although others have addressed the problem of modeling the coordination work with the purpose of building computer support (e.g., Flores *et al.*, 1988; Malone and Crowston, 1990; Swenson *et al.*, 1994), only few studies have addressed the applicability of object-oriented modeling techniques for modeling the coordination aspects of a real life cooperative work situation.

First, the basic ideas in the object-oriented approaches are briefly introduced. The results from the experiment are described, i.e., an object-oriented analysis model of the bug form mechanism containing system definition, clusters, class structure, events, and behavior diagrams is presented. The chapter is concluded with some reflections on the usability and applicability of an object-oriented approach to modeling coordination mechanisms.

## 10.1 The object-oriented approach

A number of object-oriented methodologies for systems design have been proposed over the last years (e.g., Coad and Yourdon, 1991; Jacobson *et al.*, 1992; Mathiassen *et al.*, 1993). Although they are definitely more complicated to use than intimated in the quote from Coad and Yourdon in the beginning of this chapter, they have become increasingly popular in both industry and academic communities. The methodologies encompass principles and guidelines for the development of systems—which tasks to carry out, which techniques to use, etc.—as well as principles for what representations to produce in the process, i.e., which diagrams and descriptions to make. The object-oriented paradigm is intended to be applicable to all kinds of systems.

There is little standardization in the field of object-orientation, apart from the notion of classes and objects (Monarchi and Puhr, 1992). Some general characteristics can, however, be identified: The elemental concept of the paradigm is, not surprisingly, that of an *object* modeling a phenomenon of the real world. An important underlying principle of object orientation is that of being able to use the same set of concepts throughout these different phases or aspects of the development process. A *class* denotes a set of objects sharing static and dynamic properties. *Structural relationships* between classes are one of the main strengths of object-orientation. Relationships like one kind of vehicle being a specialization of another, or a customer having three bank accounts, are expressed in inheritance hierarchies, and association or aggregation relationships respectively. The expressive means and the resulting products are typically lists of objects and events, diagrams illustrating static and dynamic properties of single classes, and structure diagrams for interrelated objects. All with corresponding textual descriptions. Other key principles are those of encapsulation, coherence, and reusability. In object-oriented methodologies, an object represents an entity, or a phenomenon, in the real world.

The methodology applied in the experiment was the analysis part of the OOA&D method promoted by Mathiassen *et al.* (1993; 1995). Using the classification structure of Monarchi and Puhr (1992), the OOA&D approach is representative of object-oriented methodologies, providing both process- and representation support. A second argument was, that we were familiar with the methodology. Furthermore, a central characteristic of many coordination activities is the dynamics. Activities are often related to several structures in the problem domain. OOA&D is one of the few methodologies that explicitly claims to address the issues of the dynamics within the problem domain.

OOA&D is based on techniques selected from a number of established object-oriented methodologies, together with basic principles for using the various concepts, techniques, and notation forms. The most important principles include 'using objects as a unifying concept' and 'describing the model of the problem domain before the requirements to the functionality'. The former is what basically imbues all object-oriented methodologies; having as key concepts: object, class, inheritance, association, aggregation. The latter is to be seen as a contrast to having functional requirements as the primary objective, i.e., basing the modeling of dynamic aspects on the concepts of event and behavior. This is inspired by Jackson System Development (Jackson, 1983), since no existing object-oriented methodologies seemed to grasp the kind of dynamics of the problem domain, that was sought expressed when developing OOA&D.

Focus, in the analysis part of the methodology, is on building a dynamic model of the problem domain. Among the main activities in the analysis phase is the specification of a 'system definition' stating what elements are of interest in the problem and application domains, including the general functionality of the system. Other central activities are the generation of a 'structure diagram' expressing the static aspects of the model, and 'behavior diagrams' defining the relationships between events for each object class.

## 10.2 An object-oriented model of the bug form mechanism

The idea of the experiment was to build an object-oriented analysis model of the computational aspects of the coordination of software testing in the S4000 project. The aim was on capturing the problem domain characteristics and the modeling process. Specifications of interface or functions was excluded. The fundamental principles of object-orientation include that of using the same basic concepts throughout the development process. It is therefore the 'analysis phase' that determines the essential components in the system, and thus the phase where the applicability of object-orientation comes to test. Hence, only activities regarding modeling in terms of object-oriented analysis have been conducted.

The experiment consisted of the following activities, in consonance with the recommendations of OOA&D: 1) Specifying the 'system definition'; 2) identifying candidates for object-classes; 3) specifying clusters of classes; 4) identifying main attributes for each class; 5) specifying relationships between classes in a structure diagram; 6) identifying relevant events; and 7) specifying the event-behavior for each class. The method does not prescribe, that the model is specified according to a strict sequence. The actual modeling tried to follow this sequence, but it was, of course, a much more discontinuous process than what can be seen from the following descriptions consisting of several iterations each refining the model. The system definition, structure diagram, class descriptions, event lists, and behavior diagrams established are described in the following subsections.

### 10.2.1 System Definition

Inspired by the ideas of Checkland (1981) on defining a 'root definition for a relevant perspective on a system to be analyzed' the OOA&D prescribes to specify a system definition. A system definition is, according to the method, a short and precise piece of natural language text that defines the scope and boundary of the system with respect to a number of crucial factors:

- The conditions under which the system is to be developed and used, if any such are of specific interest in that they, for example, restrain the possibility for user involvement, or consist of areas of conflicting interests.
- The problem and application domain, i.e., an overview of the phenomena (later modeled as objects, hence termed 'the object system') and tasks to be captured and supported by the system.
- The functionality required to support the phenomena and tasks.
- The technology available during development and usage (this was considered irrelevant for the experiment and thus omitted).
- The philosophy, describing the essence and idea behind the system and how it is supposed to form part of the work setting.

These considerations constitute the criteria of relevance when assessing class, event, etc. candidates at more detailed levels later on in the process. The below system definition is defined as a basis for the modeling of the bug form mechanism:

The system is to be used by the testers, software designers, and management as an integrated part of the coordination involved in registration, diagnosis, and correction of bugs. The problem domain consists of software and bugs, software testers and designers, the registration, routing, and monitoring of the bug handling process, and the work plan for scheduling and coordinating these tasks. The functionality required is storage and retrieval of information on who is doing what and when. The systems philosophy is fourfold: automating the routing of tasks, making state-of-affairs of bugs available, supporting the coordination of the tasks, and supporting the actors in organizing the work as they consider most efficient.

### 10.2.2 Classes and Structure

Candidates classes and relationships were identified by going carefully through the phenomena described and observed and searching for things, actors, roles, concepts, resources, events, etc. that could be candidates for an object class. All these were then considered and clustered according to the relevance recommendations specified by the methodology.

The resulting set of relevant classes was subdivided into three clusters: Software, Work, and People. These encompass, respectively, the software that is being debugged and its bugs, the debugging work tasks, and the people performing the debugging tasks. The cluster view on the model is merely for clarification purposes, to give an overview when, as in the current case, there is what can be considered a fairly high number of classes.

More important were the generalization and aggregation relationships, expressing the range of possible static situations in the problem domain. These where identified by considering possible static relationships between the classes, and relationships that occurs due to the dynamic interaction between objects. Figure 10-2 shows the structure diagram

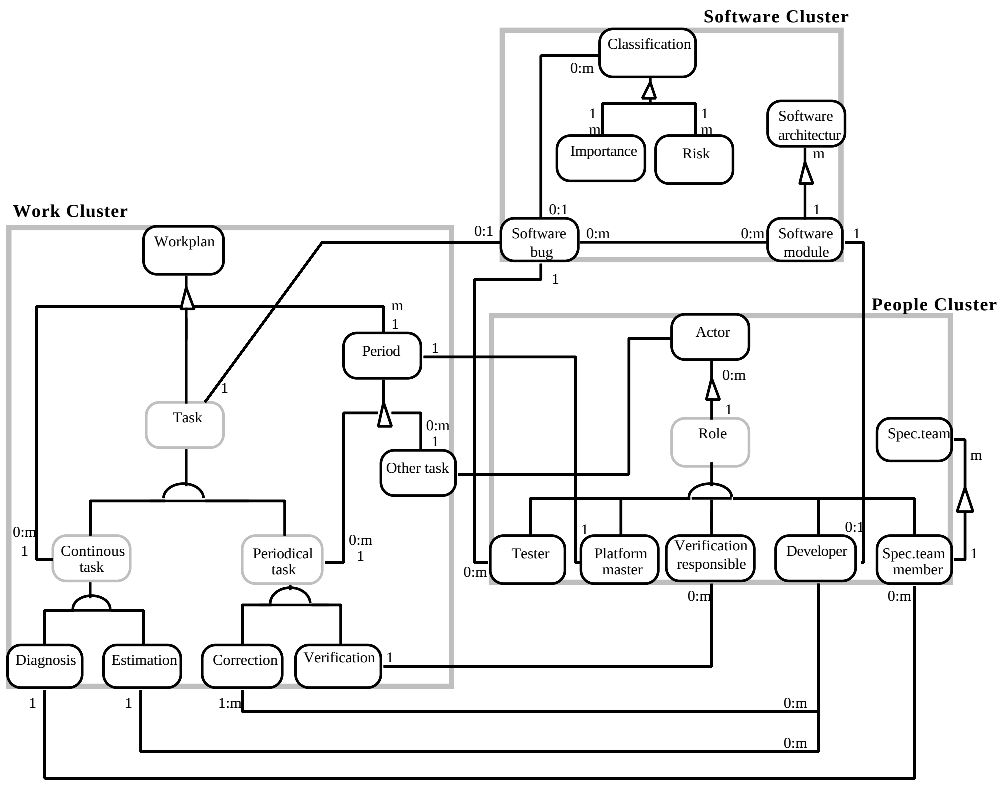


**Figure 10-2:** The Structure Diagram.

The classes are represented as rounded boxes. A grayed class indicates that the class do not have any instances in itself. Triangles symbolize aggregation, and semicircles generalization structures. Lines with no symbols are associations. Numbers and intervals are the cardinalities, i.e., they state how many instances of the other class an instance of one class can be related to. The three large grayed boxes illustrates the three clusters.

The modeling technique illustrated follows the diagramming language suggested by the OOA&D methodology.

Throughout the remainder of the chapter, capitalized words are used for the names of classes and clusters to help distinguishing them from other uses of the same words. Each class and structure is described in further detail in figure 10-3 below.

| The Software Cluster | The Actor Cluster | The Work Plan Cluster |
|---|---|---|
| **SW-architecture:** The software part of the instrument, or the part of it, that is currently being debugged. It is an aggregation of a number of software modules.<br>**SW-module:** Attributes(Developer)<br>A distinguishable part of the SW-architecture. At any given moment it is related to zero or some bugs that have been diagnosed to originate herefrom. Each SW-module has a Developer responsible for it.<br>**Bug:** Attributes(Description, classification, testers initials, module relation, the various information resulting from diagnosis through verification tasks, current state)<br>*The* central class of the system. A bug is related to the tester that found it, and becomes associated with a classification and the suspected SW-module as part of the diagnosis process. At any given moment it is related to one, and only one, of the bug correction tasks and therethrough indirectly with an actor. It is the filling in and updating of a bug objects attributes and the dynamic change in its association with tasks that are the essential dynamic behavior of the system.<br>Classification, Importance, and Risk:<br>Basically these classes constitute the range of possible categorizations of bugs; each classification is a combination of the assessed risk and importance of a bug. | **Actor:** Attributes(Name etc.)<br>An actor is any person who can take on one or more of the roles that are relevant for the bug correction process. Through the roles, an actor can be involved in zero or more tasks of the same or different kinds. For the sake of completeness of the work plan, an actor can also be related to any other tasks that are not relevant to the debugging, but that take up the persons work hours.<br>**Role:** Attributes(Assigned actor and current tasks)<br>A role is taken on by an actor for some time, during which a number of tasks, of the specific kind that role is associated with, can be conducted.<br>**Tester:** The role testing software and identifies bug.<br>**Platform Master (PM):** The designer responsible for a whole platform period; the parts that make up the work plan.<br>**Verification Responsible:** The person who gets assigned to verifying the correction of a bug.<br>**Developer:** An actor that takes on estimation and correction work.<br>**Spec-team:** The small group of people responsible for the initial assessment of the bug, its acceptance and classification.<br>**Spec-team Member:** A member of the Spec-team | **Work plan:** Where information on scheduled work and resources is gathered.<br>**Period:** A scheduling and coordination entity.<br>**Task:** Attributes(Bug, actor, time)<br>Short for Bug Correction Task. Any task is always related to a bug and an actor. A task object models the actual state of a bug, as opposed to the intended state.<br>**Continuous Task:** The kind of tasks that, for scheduling reasons, are *not* necessarily related to specific periods.<br>**Periodical Task:** The ones that *are*.<br>**Diagnosis:** The first treatment of an identified bug. During the existence of such an object, the bug is classified by a member of the Spec-team and associated with the SW-module suspected of containing the bug.<br>**Estimation:** Based on the diagnosis, a correction time is estimated by a developer.<br>**Correction:** One or more developers perform the correction itself. Possible outcomes of this task, apart from the bug being corrected, is that it has to be diagnosed or estimated again.<br>**Verification:**<br>The bug correction is verified, i.e., it is either acknowledged as corrected and 'filed' or it has to be registered all over again.<br>**Other Task:** Relevant only to planning the work because one of the actors is assigned to it. |

**Figure 10-3:** Descriptions of the classes, organized according to the three clusters. Details about attributes considered of no importance to the experiment are omitted.

### 10.2.3 Dynamics

The next of the modeling concerns the dynamic aspects of the problem domain. The basic concepts for modeling these are the events occurring in relation to the system, and behavior of the classes.

According to the methodology, events are the atomic parts of the dynamic aspects of the problem domain. That is, an event is any occurrence in the real world that causes a change in the value of the attributes of one or more classes in the object system, or that requires a change in the structure of the classes, typically an association being established or removed. Similarly, the behavior of a class is identified as an abstract pattern of events that defines the possible course of events for all objects in a given object class.

The first step was to identify all possible and relevant events. Again, all phenomena and descriptions were searched for candidates of events to be included in the list. The resulting list is illustrated in figure 10-4.

Using the list of relevant events as input the next step is to make a behavior diagram for each class, illustrating the constraints on the order in which the events that involve a specific class can occur. Behavior diagrams are diagrammatic representations of finite automatons.

With the event list and the behavior diagrams we can determine equivalent patterns of events that objects of a class has to follow. In the case of the bug form mechanism modeled here, a large number of events are common for a number of classes: They involve changes in more than one type of object. Furthermore, many classes are involved in more than just a few events. These two characteristics tend to yield complicated behavior diagrams. The former makes it difficult to get an overview of the consequences of one specific event. The latter because many events have to be part of the same diagram.

| Event List | |
|---|---|
| • New Actor inserted<br>• New SW-module inserted (and put into SW-architecture aggregation)<br>• New Period inserted (and put into Work plan aggregation)<br>• New Importance inserted<br>• New Risk inserted<br>• New Classification inserted<br>• Actor assigned as Spec-team Member (and put into Spec-team aggregation)<br>• Actor assigned as Verification Responsible<br>• Actor assigned as Platform Master<br>• Actor assigned as Tester<br>• Actor assigned as Developer<br>• Actor resigned as Spec-team Member<br>• Actor resigned as Verification Responsible<br>• Actor resigned as Platform Master<br>• Actor resigned as Tester<br>• Actor resigned as Developer<br>• Actor assigned to Other Task<br>• Platform Master assigned to Period<br>• Developer made responsible for SW-module<br>• Bug registered (a new Bug is instantiated and associated with the responsible Tester)<br>• Bug postponed (no task association happens, bug deleted and/or re-registered later) | • Bug rejected (no task association happens, bug deleted and/or re-registered later)<br>• Bug accepted (a new Diagnosis is associated with the Bug and a Spec-team Member)Bug diagnosed (the Diagnosis is deleted, a new Estimation is associated with the Bug and a Developer)<br>• Bug estimated (the Estimation is deleted, a new Correction is associated with the Bug and a Developer)<br>• Bug corrected (the Correction is deleted, a new Verification is associated with the Bug and a Verification Responsible)<br>• Bug re-diagnosed (the Correction is deleted, a new Diagnosis is associated with the Bug and a Spec-team Member, when the corrector finds the diagnosis was wrong)<br>• Bug re-estimated (the Correction is deleted, a new Estimation is associated with the Bug and a Developer, when the corrector finds the estimation was wrong)<br>• Bug correction verified (the Verification is deleted)<br>• Bug re-registered (the Verification is deleted, a new Bug is registered, when the verifier finds something else was the matter)<br>• Bug deleted (when the information is obsolete) |

**Figure 10-4:** The final events list. For some events more classes are involved than those mentioned in the event name. In these cases further details of classes affected are added in a parenthesis. In other situations a parenthesis is used for a clarifying description. Details about when which attributes are updated are omitted.

Showing behavior models for all classes would require a lot of space. Let us, for illustration purposes, look at one of the essential events and present all the classes involved as an explanatory example of, how the dynamic aspects are modeled: Some of the most crucial events to be modeled were those that had to do with the transition from one stage of handling a bug to another. These were also the single events that involved objects from the largest number of different classes. The event ‘Bug corrected’ is chosen here for the illustration of how the dynamics of the problem domain was modeled. The ‘Bug corrected’ event represented the situation where a correction task was completed successfully, i.e., without the need for re-diagnosis or re-estimation, and the relevant next task—the verification—was to take over. The event involved five classes: Bug, Correction (the prior task completed), Verification (the next task to take place), Developer (who had made the corrections), and Verification Responsible (who had to verify the correction). The behavior diagrams for those five classes are shown in figures 10-5 through 10-8.

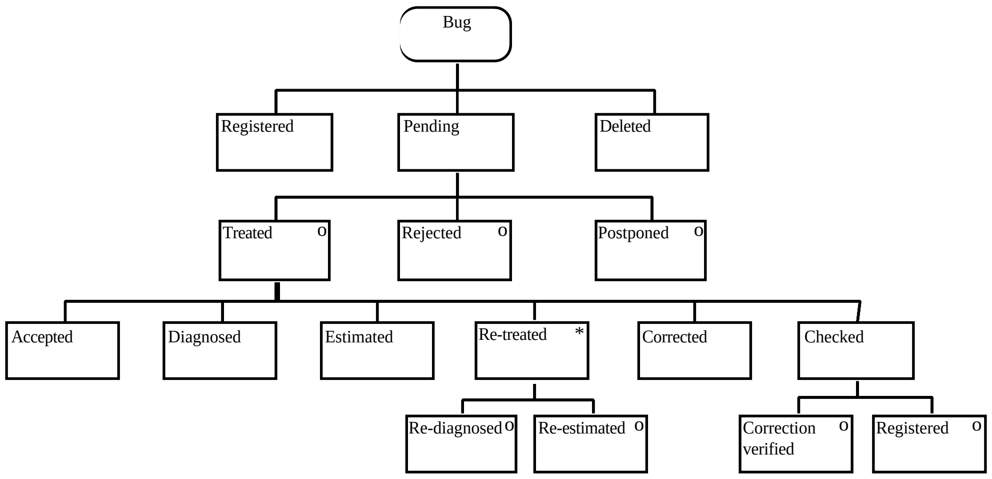


**Figure 10-5:** The behavior diagram for the class Bug. The events with a ‘o’ are part of the same selection. A ‘*’ is an iteration, and a row of events on the same level, that are either marked ‘*’ or not marked form a sequence.

‘Bug corrected’ can be considered to be one, atomic, task. The event was an important part of the coordination happening in the bug correction work. It indicated that one person had completed a task, which then implied that the next person to take over could start his own task on the specific bug. He therefore needs to be made aware of the fact, that he can start.

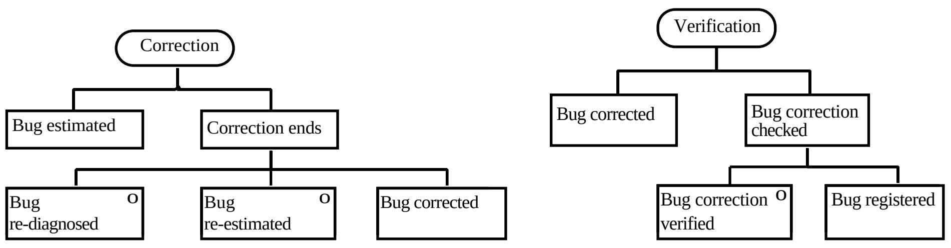


**Figure 10-6:** The behavior diagrams for the classes Correction and Verification.

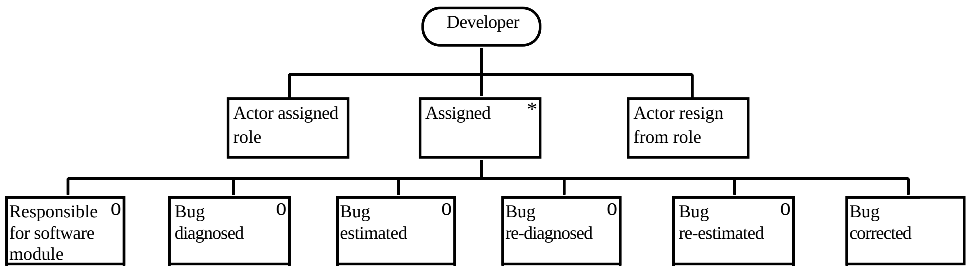


**Figure 10-7:** The behavior diagram for the class Developer.

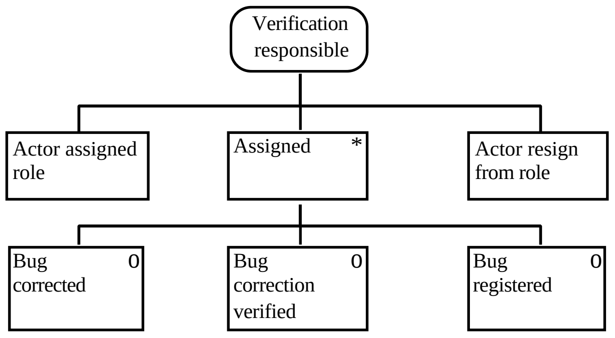


**Figure 10-8:** The behavior diagram for the class Verification Responsible

In terms of how the structure diagram and classes are changed six things happen when the ‘Bug corrected’ event occurs:

(1) Attributes in the Bug object regarding the correction are updated;

(2) The association between Bug and Correction task objects is removed;

(3) The Correction task object is deleted;

(4) A Verification task object is instantiated

(5) The Verification task object is associated with the Bug object; and

(6) The Verification task object is associated with a Verification Responsible object.

## 10.3 The usability of object-oriented modeling techniques

The idea of the experiment was, as mentioned, not to establish a complete object-oriented model for the bug form mechanism, and it was, for example, not used as input for the design of the prototype presented in chapter 9. The intention of the experiment was to explore, what the possibilities and limitations for modeling computer-based coordination mechanisms by means of an object-oriented analysis method are, and would this type of modeling grasp the essential aspects of coordination, and could it improve the understanding gained from the other descriptions made (cf. chapter 6)? This section will discuss, what has been learned in relation to these questions.

In general the OOA&D method provides good support for specifying the static aspects of the model, e.g. defining clusters, classes, instances, aggregations, and associations. The big problem regards the dynamic aspects of the problem domain to be modeled. These just seems to “disappear” in the model. This is not a problem specific for OOA&D, but it is interesting to notice, since the method explicitly states that event lists and behavior diagrams are included in order to deal with the dynamic aspects of the problem domain. Object-oriented methods, in general, do not sufficiently address “the sequence timing and control of events and processes” (Monarchi and Puhr, 1992, p. 4). The experiment described here confirms Monarchi and Puhr’s observations. There are more object-oriented static models managing structural complexity, than dynamic models handling behavioral complexity. As it is now, the dynamic aspects of object-oriented models are expressed in “attached” formalisms, such as state-transition diagrams, Petri-nets, etc. A closer integration between static and dynamic aspects of object-oriented models is highly needed.

In Carstensen *et al.* (1995a) we concluded the experimented by a list of specific lessons learned regarding the usability of object-oriented approaches for modeling coordination mechanisms. Elaborated from these, the following conclusions should be mentioned:

(1) As mentioned above, the object-oriented modeling improved the quality of the structural specifications. Prior to the OOA&D experiment, entity-relationship diagrams, state-transition diagrams and diagrams of the information flow were the only means applied to express the properties of the bug form coordination mechanism (cf. section 6.7 - 6.9). The cluster specifications, the structure diagram, and the cardinalities of class-connections improved my understanding of which structural aspects of the coordination mechanisms should be computer supported.

(2) Similarly to the requirements for for a proper semantic level when describing the objects of coordination work (cf. section 5.5), the level of detail in which an event is defined must be in an appropriate form to reflect real life coordination. As an example can be mentioned the ‘Bug estimated’ event. It was defined as: “The Estimation is deleted, a new Correction is associated with the Bug and a Developer”. Alternatively, each sub-event could be expressed as a separate event. This would, however, result in loosing the essence of the coordination which takes place when the spec-team estimates the time needed to fix a bug: A developer is notified, takes over, and starts the correction. Similar to the events, a ‘coordination function’ will include several roles described in several behavior diagrams. It is the contiguousness that characterizes the coordination according to the organizational procedure, not the individual sub-tasks. The coordination mechanisms we want to design should provide the facility of automatically conveying, to one actor, changes in the state of the mechanism induced by another actor, as reflected also in the system philosophy, stated in the system definition (cf. section 10.2.1).

(3) Regarding modeling at a proper level, the model must contain classes reflecting conceptualizations used in the real life coordi-

nation. It could, for example, be tempting, instead of having four classes expressing different tasks, only to have a 'Continuous task' and a 'Periodical task' class, or maybe even only a 'Task' class. Then the diversity of the tasks could be expressed through attributes and the behavior of actors. However, given the composite events, mentioned above, this would result in loosing the essence of the coordination mechanism. In the behavior diagram, which even in a selectiont being relevant would depend on the value of an attribute. Furthermore, getting an overview of the consequences of an event like 'Bug estimated' would involve looking at two instances of the same diagram instead of two different diagrams. Both are undesired situations.

(4) Although clustering classes provided a straightforward way of subdividing the structure, the process did not help in designing linking between coordination mechanisms. In chapter 8, four linked coordination mechanisms was identified (cf. figure 8-9 in section 8.2.5). Two of these were identical to the 'Work' and 'People' clusters (named 'Planning CM' and 'Human Resources CM'). The two others were the 'Software' cluster divided into a 'Software Architecture CM' and a 'Bug CM'. The three clusters in the model makes sense from an OOA&D perspective, but the four mechanisms express that, a re-designed bug form mechanism should be able to link to an existing repository containing conceptualizations of the software architecture and modules. Furthermore, the concern of modeling linking between coordination mechanisms resulted in classes which, from an OOA&D perspective, should be expressed as attributes to other classes. The contents of the 'Classification' class could, for example, be defined as an attribute to the 'Software bug' class. This was not done since the 'Classification' class is considered providing a link to an external mechanism maintaining the classification structure. Hence, clustering does not support linking, or designing coordination mechanisms by means of interrelated software agents—as promoted by Divitini *et al.* (1995b)—properly.

(5) As mentioned, a fundamental finding concerned the interactions between actors involved in coordinating their activities. These are very difficult to identify from the established model. When mod-

eling coordination work, the interaction between actors is an important structural property of the problem domain. As argued several places in this dissertation, protocols stipulating the flow of the work, or the flow of the coordination work, are essential. Although all the events related to the procedure of registration, diagnosis, correction, and verification are described in different diagrams, it is impossible to see from the model, how the documents are exchanged among the actors.

(6) Although reflected in the behavior diagrams, complex dynamics is very difficult to model, and the dynamics is difficult to grasp from the models, e.g., in order to assess the 'Bug corrected' event comparison of five different behavior diagrams is required. Coordination work mainly consist of activities meshing symbolic references to the problem domain, e.g., "have actor A fixed bug 7?". This implies, that many of the events, that can be observed, will affect many classes. According to the findings in this and other field studies, this is a very basic characteristic of coordination work. Coordination work is about doing things (provoking events) related to two or more conceptual structures (potential object classes). The OOA&D methodology states, that if there are many mutual events between classes, this implies that the class-model (the structure diagram) should potentially be redesigned:

> "Consider mutual events between several objects: Mutual events point at important dynamic relationships in the model. If there are mutual events, the classes chosen and their structural relationships should be reconsidered" (Mathiassen *et al.*, 1993, p. 122, my translation).

The model could, of course, have been redesigned, but this would break up the intuitive and nice cluster structure. Hence, the principle of redesigning the model, if there are many mutual events between classes, is potentially in conflict with the very nature of coordination work.

A first brief conclusion on, whether it is useful to model coordination mechanisms by means of object-oriented techniques, must be that the OOA&D methodology provided very powerful means of managing and expressing the static complexity of the coordination of the software testing work. It was, however, insufficient with respect to the dynamic aspects. The latter must be concluded despite the fact that OOA&D explicitly aims at providing means for modeling dynamics. Using an object-oriented approach should therefore be combined with other techniques having their strength in modeling the dynamism and concurrency aspects of coordination work, e.g., petri nets.

As mentioned, the object-oriented model provided good support for expressing the static complexity of the coordination work, better than the entity-relationship diagrams used in chapter 8 (cf. figure 8-8) since the association structures were better reflected. The state-transition model (cf. figure 6-7) and the models of the flow between roles respectively actors (cf. figure 6-9 and 6-10) provided a better understanding of the flow of information among the actors than the behavior diagrams. So, the rather primitive representations of the dynamic aspects of coordination applied in chapter 6 and 8 proved to be better than the models provided by OOA&D.

It is difficult to make any claims on whether the process of building object-oriented models adds knowledge and understanding of coordination, or it just models already recognized knowledge differently (cf. e.g., Kensing and Munk-Madsen, 1993). Making claims regarding this would require an experiment where object-oriented models are build without having conducted other kinds of analyses beforehand. My impression, from the process of analyzing the field study as reported in chapter 6 and 8 followed by the experiment reported in this chapter, is that the object-oriented methodology mainly supports a more rigid process of establishing explicit models of already recognized knowledge. This is, in itself, very useful in many analysis situations. And it is important to note, that the claim needs much further investigation.

In assessing the outcome of the experiment, it is relevant to notice, that modeling, like many other human activities, involves a certain element of style. None of the persons conducting the experiment were in any position to claim being an expert, but it was conducted from a solid base in terms of a thorough field study and a some modeling experience, based on other approaches than the object-oriented. OOA&D is only one of many object-oriented methods. The object model part of it is, however, not radically different from most other approaches. The conclusions will thus probably be valid for most of the existing object-oriented methodologies. Wirfs-Brock *et al.* (1990) offers what they call ‘contracts’ and ‘collaboration graphs' which might be relevant for modeling coordination. The concept of 'use cases' (exemplar usages of the system, each one illustrating a particular subset of the functionality), introduced by Jacobson (1992) is another recent and powerful approach to expressing dynamic issues in object-oriented modeling. Both approaches should be investigated further. This has, however, been considered out of scope of this dissertation.

# 11. General recommendations for design of computer-based coordination mechanisms

“The general attitude seems to be that people should wear square shoes, because square shoes are easier to design and manufacture than foot-shaped shoes. If the shoe industry had gone the way of the computer industry it would be running $400-a-day courses on how to walk, run and jump in square shoes.”
(Alan Kay quoted in Durham, 1988).

The three previous chapters have aimed at illustrating and discussing how the findings from the field study could be used as a basis for building a computer-based system supporting the coordination of software testing and correction work, i.e., the work has been constructively oriented. This chapter discusses which general recommendations regarding design of computer-based coordination mechanisms we can derive from these experiments. The findings and lessons learned are not mature, unambiguous, and conclusive. Hence, the following should not be regarded clear-cut normative statements about design of coordination mechanisms. They do, however, represent a series of experiences, which can serve as a basis for preliminary recommendations.

The design process has been similar to many other design processes. Theories, methodologies, guidelines, etc. can only facilitate the process. The fundamental in design will inevitably be a matter of making design decisions. How to organize the development process is important. It has, however, not been essential for the work presented in this dissertation. Useful representations, recommendations on setting up requirements, and a few reflections on evaluating a coordination mechanism design will be given. From the experiences collected during the field study and prototype design work, I do not find it relevant, to go into further discussions on how to organize *the process* of designing coordination mechanisms.

This chapter is structured into four sections related to modeling analysis findings, setting up requirements for coordination support, concrete design of coordination mechanisms, and evaluating preliminary prototypes of coordination mechanisms.

## 11.1 Modeling the coordination aspects of the work

Modeling and representing work that is going to be supported or re-designed is a very complicated process. We need both representations of the work studied and of the system to be designed (Kensing and Munk-Madsen, 1993; Kyng, 1995). It is, however, difficult to prescribe or recommend on which representations are needed (and sufficient) for informing the design. Involved in designing computer-based systems we need representations that can be used for: 1) supporting the designers understanding the basic characteristics of the work, 2) supporting the future users understanding of the abstract descriptions of their work, and in grasping the ideas of the different (re-) design suggestions, 3) helping the designers establishing possible (re-) designs, and evaluate these, 4) facilitate an interaction between the designers and the coming users, and 5) anchoring the ideas and visions established in the project. Mock-ups, different types of prototypes, narratives, and scenarios of future use are some of the most frequently suggested and used representations (cf. e.g. Carroll and Rosson, 1992; Simonsen, 1994; Kyng, 1995).

In this section, I will concentrate on representations related to the first mentioned usage: Representations having the purpose of supporting non-domain experts (the designers of the computer-based systems) in getting a sufficiently deep and coherent understanding of the basic characteristics of the work to be supported. The other usages are extremely important, but is considered out of the scope in the context of this dissertation.

The experience gained from the field study indicates that it is very useful to explicitly model the coordination aspects of the work setting. The analytical distinction between work and coordination work argued for previously proved to be useful, and it is strongly recommended to apply this distinction. Both aspects of coordination work and actual work must, of course, be modeled, but to address the coordination aspect explicitly improves the understanding of, for example, the functions provided by forms, boards, organizational procedures, classification schemes, etc.

Much coordination work is conducted by means of conceptualizations. Both the Foss Electric field study described in this dissertation and other field studies (e.g., Harper *et al.*, 1989b; Edwards *et al.*, 1995) illustrate, that much coordination is organized, and mediated, in terms of conceptual representations of structures from the actual work conducted (the field of work), the work setting (the work arrangement), and the broader organizational context. It is recommended to—in as early a phase in the analysis and modeling work as possible—establish models illustrating the conceptual structures used for coordinating the work. These models should include, at least, conceptualizations of the field of work, and conceptualizations of the work arrangement. Examples of conceptualizations of the field of work are bug classifications and the software architecture within the area of software testing, or airplane identifications and airspace divisions within air traffic control domain. Examples of conceptualizations of the work arrangement are representations of the actors and roles involved, and work plans established. In some situations, it will, furthermore, be relevant to include conceptual structures representing the wider organizational context (e.g., general company policies, external regulations to be obeyed, etc.). The scheme containing and structuring the objects of coordination work (cf. figure 5-2) proved very useful for conceptualizing the central findings. It is recommended to use this scheme for approaching the phenomena observed and for structuring the findings. Using this scheme can help identifying the conceptual structures. It is, however, important to also be aware of structures that do not fit into the scheme, i.e., the scheme should be used for inspiration, not as a filter for what is included in the analysis.

To improve the information in the models containing the conceptualizations, it is, additionally, suggested that relations between the conceptual structures established as a result of certain coordination activities are presented in the models (e.g., relating a role to a task as a result of a resource allocation process). An example of a model of conceptual structures along which the coordination was conducted, and their relationships, was given in figure 8-1. An introduction to, and discussion of, a previous version of the model can be found in Carstensen *et al.* (1994).

It is important to grasp (model and describe) the mechanisms used to support the coordination, and how these are interrelated. The term ‘mechanisms’ covers all types of mechanisms used for coordination, ranging from conventions for how to inform each other, over forms and organizational procedures, to more active and/or rigid mechanisms like work flow systems. The idea can be compared to taking an “activity-oriented approach” as suggested by Sachs (1995) trying to look at whole activities distinct from a particular task. Mechanisms used for coordination of series of tasks should be addressed and modeled. That is, we should aim at achieving a generalized understanding of: What is the purpose of the mechanism, what function does the mechanism provide, who is it provided for, under which conditions is the function provided, and how are the different mechanisms related to each other? The ambition is to establish an overview of which types of coordination work are conducted, and how these are supported by different mechanisms. An illustration of how some of mechanisms identified in the field study were interrelated was given in section 6.11.

Apart from modeling the coordination support as a “network” of related mechanisms, it proved informative to address the behavior (and typical deviations) of the individual mechanisms. This can be done by means of, for example, state-transition diagrams, as illustrated in section 6.7. Through this kind of modeling, the analyst or designer is forced to consider possible states for the mechanism, and the possible subsequent states. Both the “standard flow” of the mechanism and relevant deviations are thereby explicitly considered. Modeling the observed mechanisms supporting coordination by means of state-transition modeling, or similar techniques, is recommendable.

As well as addressing the conceptual structures and mechanisms used for coordination, the field study at Foss Electric exemplified the importance of obtaining an understanding—and thus a description or model—of the working rhythms applied in the work studied. Much cooperative work is organized in working cycles. These can be structured with respect to specific time slots, the tasks to be conducted, work shifts, etc. The rhythm in which the work is organized and conducted is in itself an important structuring and coordination mechanism that needs to be described. One way of describing the working cycles or rhythms is by means of a time line for a full cycle. This can then illustrate which roles are conducting which activities or actions, at which point in time. The activities and actions included should, of course, mainly be those related to the coordination of the work. Such time lines illustrate the rhythm and, to some extent, the sequence of interaction among the involved actors. Time lines can be used for modeling both “here and now” activities, coordination intensive activities (like air traffic controlling), and cooperation conducted over long time periods (like design work). A good example of how a time line can be used for illustrating the interaction among several actors can be found in Kensing and Winograd (1991).

It is, furthermore, relevant to address the structure of the interaction and coordination work from the perspective of the involved actors. There are two aspects of communication that it can be relevant to model:

The first is the interaction between the work system addressed and its context, i.e., modeling the work system as a black box interacting with its surroundings. How this interaction can be analyzed and modeled in terms of overall functions provided by a work system is thoroughly discussed in Schmidt and Carstensen (1990). It is outside the scope of this chapter to explain these ideas further. Another useful technique for modeling the interaction between a work arrangement and its context is the communication path diagram suggested by Kensing and Winograd (1991) illustrating the major paths of interaction, defining their direction, and the media used. This modeling technique has not been used in the analysis presented in chapter 6, but in cases where real time coordination is required, this type of models will improve the general understanding of the work and its coordination.

The second type of interaction to model is the flow of information among the actors within the work system. Setting up a model of the information flow provides important input on the dynamics of the coordination and the interaction between the actors. Such flows will usually model the “ideal” or “normal” flow being the flow present when the organizational procedures, conventions, etc. are followed. It will, in practice, often be impossible to model all possible deviations from this flow, but it is nevertheless highly recommended to establish such models. Both models of the flow organized according to roles and prototypical examples of flows from the viewpoint of the involved actors should be considered.

Chapter 10 experimented with using object-oriented techniques for modeling coordination mechanisms. The conclusion from this was quite clear: The models provided good support for specifying the static aspects of the work in terms of defining clusters, classes, instances, aggregations, and associations. But the dynamic aspects of the work and its coordination modeled were not “visible”—and thus understandable—from the models. It is recommended to model the static aspects by means of some of the traditional modeling techniques within the area of software engineering, for example object-oriented techniques (e.g., Coad and Yourdon, 1991; Mathiassen *et al.*, 1993) or entity-relationship modeling (see e.g., Flavin, 1981). When doing this, it is furthermore recommended, that the basic primitives, structures, concepts, etc. used for modeling the conceptual structures in the field of work, the work arrangement, and the organizational context, and their interrelations are at a proper semantic level. They must be at a level, where it makes sense for the future users of the system. If we model by means of too low-level programming primitives, the future users will not be able to relate them to their daily work.

The dynamic aspects of the coordination need careful consideration. They should be modeled further than what has been recommended so far. The idea of identifying all events and model the dynamics in relation to these as promoted by Mathiassen *et al.* (1993) appears to be useful, but the resulting models must provide a better overview of the dynamics. The experiences accumulated through the work described in the previous chapters have not provided any good suggestions for a technique for modeling these aspects. Holt (1985; 1988) argues for using Petri Net for modeling what he calls “coordination mechanics”, and Käkölä (1993) suggests a language called Role Interaction Net (RIN), based on organizational role theory and Petri Nets for understanding the coordination of work. Some of the flow models (e.g., ICN, cf. Ellis, 1979) might also prove useful here. These approaches have, however, not be considered in the work presented here, and thus need further investigation.

What I have argued above is to consider using a number of different modeling techniques in order to address certain aspects of the coordination work studied. Each of the approaches suggested are very limited. None of the mentioned representations—or any other representations—will provide a complete picture. Even combining all the techniques will only result in a very limited representation of the work. Most analysis and modeling work will, however, be constrained by a limited amount of resources. So, modeling the observed work will always cover certain aspects only. Given this fact, I will argue that the suggested approaches, used intelligently, and combined with a great deal of common sense is the best we can do at the moment. The experience from the field study and the construction oriented activities confirm what Kensing and Munk-Madsen (1993) argue: It is in the process of switching from one approach to another that your understanding is elaborated, i.e., we enlarge our knowledge when we try to conceptualize and model the observed phenomena, or when we attempts to find phenomena that illustrate certain aspects of a model, theory, or framework. The discussion in this section has, furthermore, addressed techniques for supporting the designers in understanding the basic characteristics of the work. Techniques and representations fulfilling the purposes of supporting users understanding the models, facilitating interaction between users and designers, anchoring visions, etc. should be included too. These are, however, not considered relevant here.

## 11.2 Towards useful requirements for coordination support

Setting up requirements for a computer-based system supporting a group of actors in coordinating their distributed activities is a complicated task. Much research has been conducted focusing on how to educe the relevant requirements (e.g., Hughes, 1993), and how to involve users in gathering requirements and specify these (cf. e.g., the May 1995 issue of Communications of the ACM). In general, specification of requirements for coordination support systems should, of course, be based on studies of the work to be supported, and established through involvement of the future users.

This section will briefly discuss a set of overall requirements, that should be considered when specifying requirements for the coordination support of a specific work setting (or work situation). These overall requirements could be regarded as a preliminary “check list” recommended for supervising a requirements specifications process. The bullets on the check list below should not necessarily be covered by a specific requirement specification, but each of the bullets need, at least, to be considered carefully with respect to the actual work setting. The recommendations will mainly be based on lessons learned from the work previously presented, especially the requirements presented in chapter 8, and the overall facilities of coordination mechanisms identified in section 5.5. Since most recommendations have been discussed previously the bullets on the list presented here will only be concisely discussed. When specifying requirements for computer-based support of coordination activities, the following overall requirements should be carefully considered:

- A coordination support system should provide access to data structures reflecting the conceptual structures along which the coordination work is conducted. Structures reflecting the central aspect of the field of work and the work arrangement should be accessible to the users.
- A coordination support system should include a possibility for categorizing and classifying the conceptual structures mentioned above. Classifications plays a central role in establishing a ‘common language’ among the actors (see e.g., Bowker and Star, 1991; Schmidt, 1994c).
- Meshing, combining, relating, allocating, etc. the structures presented in the coordination support system should be supported. Much coordination work results in establishing relations between the structures, e.g., the outcome of an allocation activity is a relation between an actor and a task.
- A coordination support system should support (or at least not exclude) informal communication among the actors. Kraut and Streeter (1995) argue, for example, that computer support of the informal and direct communication is required, and they suggest to provide tools supporting conferences and distributed meetings.
- Structures supporting negotiation among the actors should be provided, either through the communication support mention above or through more structured negotiation mechanisms, like, for example, the Coordinator (Flores *et al.*, 1988).
- A coordination support system should provide facilities for specifying work flows and how information, tasks, etc. should be routed among the involved actors or roles. Coordination of complex work situations is often handled by means of pre-specified work flows (an example is the bug report form described in chapter 6).
- The work flows stipulated by a coordination support system should be malleable. Work situations change, and the actors must have access to change the flow permanently (due to general changes), or temporarily (due to the characteristics of a specific situation).

- In order to support the malleability mentioned above, a coordination support system should provide visibility of the protocol specifying the work flow, and access to manipulate the protocol. The actors should be allowed to experiment until the best processes are found (Swenson *et al.*, 1994).
- Coordination support systems should be flexible and tailorable. Establishing the flows, processes, etc. by which the coordination work should be conducted is often a long incremental process switching from supporting individual work patterns into supporting group work patterns (Bentley and Dourish, 1995).
- A coordination support system should support the actors in being aware of each other. Heath *et al.* (1993), and several others, have illustrated the importance of awareness in order to coordinate work.
- Monitoring the state of affairs is an extremely important aspect of much coordination work. It should thus be supported. This can be done both actively (the actor is informed about changes to the state of affairs), or upon request from the actor.
- A coordination support system should keep track of the history of changes the roles involved, work flow protocols used, classification structures applied, etc.
- A coordination support system needs access to subscribe to information from other coordination support systems. In order to be supportive, the coordination mechanisms studied at Foss Electric had to include (or use) information handled by other coordination mechanisms.
- Last, but definitely not least, coordination support systems should be integrated components in other computer-based tools used within the work setting. Coordination work is always intertwined with the a actual work. Hence, coordination support mechanisms should be integrated within the work support tools used.

As mentioned most of the general requirements listed above have been derived from the work on the Concept of Coordination Mechanisms presented in chapter 5, the field study work presented in chapter 6, and the work on requirements described in chapter 8. Most of the requirements listed are based on experience, only a few of them are based on external references. This is important for understanding the generality of the requirements. The list is not complete, or ordered according to any particular criteria. The list is intended to be a source for inspiration. Using common sense is always required.

The most important input for the requirements specification process described in chapter 8 was my knowledge (from the field study) of structures and processes relevant for coordinating the work, and an overview of technological options (cf. Kensing and Munk-Madsen, 1993). Furthermore, when specifying requirements (cf. chapter 8), the well-known problem of establishing criteria of relevance for what to include was experienced. This will probably be a problem in many requirement specification situations. Further research focusing on establishing conceptual support for specifying the allocation of functionality between actor and system is therefore called for.

## 11.3 Design of coordination mechanisms

Outlining a complete prescription for the design of a computer-based coordination mechanism is impossible. This section should by no means be regarded as a set of prescriptions for how concrete computer-based coordination mechanisms should be designed. The only aim, is to elaborate a bit on, what kind of recommendations can be established based on what has been learned from the process of designing the BRaHS prototype (cf. chapter 9). The following will address aspects of possible choices regarding overall, requirements, database structure, architecture, and user interface.

### 11.3.1 Fulfilling the overall requirements

A set of overall requirements for coordination mechanisms has been discussed both in the previous section and in section 5.5. The most important general facilities relates to visibility and manipulability of the work flow protocol embedded in the system, local control of the execution, access to relevant conceptual structures, support for negotiation, and facilities for monitoring the state of affairs.

The presentation and manipulation of the work flow protocol can be designed in many different ways, and it is hard to draw any conclusions from the protocol design in BRaHS. A few things should be mentioned: It is recommended to present the protocol visually, i.e., by means of interrelated graphical structures like icons, diagrams, etc. In situations where the protocol consists of a rather limited number of 'nodes', a presentation where the user can see the total protocol would be preferable. If there are many nodes in the protocol, a leveled structure like the one used for, for example, data flow models (see e.g., Yourdon, 1989) could be applied. The manipulations on the protocol should be provided as manipulations directly on the graphical structures. This kind of interaction is usually the easiest to learn, operate and control for the users (cf. e.g., Shneiderman, 1987). Apart from BRaHS, an example is Regatta (Swenson *et al.*, 1994) providing a visual process language by which the users can specify the flow processes.

Most of the daily use of a coordination mechanism will not involve making changes to the protocol. It is therefore recommended to structure the user—coordination mechanism dialog (the user interface) so that the protocol is only presented on demand. Otherwise it might confuse the users, rather than support them.

If we think of the computerized part of a coordination mechanism as consisting of data structures and code manipulating these, the data structures should reflect the conceptual structures identified in the field study. A table of the essential objects of coordination work, similar to the one presented in figure 6-10, should be established (cf. section 11.1), and it is recommended to include data structures reflecting all the structures in this table. Please notice: The scheme is a source of inspiration, not necessarily a complete list of all structures to be included. The structures should be visible and modifiable to the users, e.g., information on actors, work plans, tasks, classification structures, etc. should, upon request, be presented to the users, and the users should be allowed to update the information.

The requirement regarding negotiation support should, first of all, be fulfilled by providing channels for informal and unstructured communication and interaction (cf. e.g., Kraut and Streeter, 1995). E-mail, conferencing systems, etc. should be provided in relation to a coordination mechanism. Use of more structured negotiation structures, as for example the semi-structured message system suggested by Malone *et al.* (1987), should also be considered.

Access to monitoring the state of affairs should be provided through several types of facilities. Search facilities should include free combination search with respect to all the conceptual structures represented in the mechanism. It is, furthermore, important to consider which kinds of aggregated information that are required. It is not possible to set up directions for which aggregation types to provide based on the work presented here. The aggregations must be designed through reflections on the coordination work conducted in the actual work setting.

Facilities allowing an actor to actively direct another actors attention to a specific situation should be provided too, for example by letting one actor include comments on a structure in a coordination mechanism, or using a communication channel, to notify another actor on changes in the state of the field of work.

### 11.3.2 Database structure

Although a detailed description of the database architecture used for the BRaHS prototype is considered out of scope of this dissertation, we have, of course, learned some lessons with respect to the design of the basic databases to be used by a coordination mechanism. The most relevant of these experiences were related to the overall requirements established in chapter 8.

The design of the underlying databases for coordination mechanisms should reflect the requirement of providing access to structures containing the conceptual structures, and the overall requirements of visibility and malleability. First of all the database structure needs to reflect the conceptual structures identified and modeled (cf. section 11.1).

Since the protocol should be visible and malleable, a database structure containing the relevant nodes and links in the protocol should be established. The database needs to have a structure containing the originally defined default protocol. This is the structure specifying how all new instances of the mechanism (e.g., one for each bug registered in the S4000 project) should be treated until something else is specified. Furthermore, the database should include a structure defining the protocol related to each instance of the mechanism.

In terms of an entity-relationship diagram, the recommendations can be summarized: In many database designs for coordination mechanisms, the following entity-types should probably be included:

- Structures containing information about the aspects of the work being coordinated, i.e., the information that needs to be mediated by the mechanism (e.g., information on the bugs in a bug form mechanism).
- Structures including information about the field of work, typically conceptualizations of structures worked on (e.g., software modules) and classification structures used.
- Structures reflecting relevant aspects of the work arrangement (e.g., the actors and roles involved).
- Structures containing the default protocol to be used.
- Structures specifying the specific protocol for each of the instances, or events, to be coordinated, and
- Structures containing prescriptions specifying the policies and rules to be applied when manipulating the protocol, or changing the content of the structures (e.g., rules for who are allowed to make changes to the bug classification scheme, and how this should be done).

These listed structures should, of course, be related to each other by applying the normal rules for entity-relationship models (see e.g., Flavin, 1981). An introduction and discussion of the entity-relationship model designed as part of the BRaHS design can be seen in Carstensen and Albert (1995).

### 11.3.3 Architecture

To outline the overall architecture of coordination mechanisms was not an essential issue in the design of BRaHS. It is, however, important to consider, and others have made considerations that are relevant to be aware of.

The architecture used in BRaHS was chosen to be organized as one application. This could be started up from several work stations all having access to the same central database. All the different types of actors should then use the same application (cf. chapter 9). This is a very inflexible structure, since changes regarding the involved roles, the overall policies to be followed, etc. would require changes to the application. It is recommended to chose a more module-based structure. This could, for example, be implemented as a set of small applications (or rather extensions to applications used for the actual work), each presenting the roles involved or the overall coordination activities to be conducted. Divitini *et al.* (1995b) discuss approaching coordination mechanisms as “active artifacts”, and introduce an architecture for designing coordination mechanisms based on ideas of software agents, and Tuomo Tuikka has designed a bug handling coordination mechanism based on these ideas (see Tuikka and Sørensen, 1995). These ideas, however, need much more considerations.

An interesting lesson regarding the architecture was learned from the evaluation of BRaHS: Although the coordination work is described in terms of roles, this might not be a good structuring concept for the application. Hence, when designing the architecture, it should be considered carefully, how the structure should be presented to the users. Both BRaHS and the ideas presented in Gordion (Tuikka and Sørensen, 1995) are based on the assumption, that the actors think of themselves in terms of roles. This is often not the case in the daily work.

If the architecture designed includes building blocks which the users are going to have access to, like for example the CSCW-systems specification language OVAL (Malone *et al.*, 1992), it is, furthermore, important, that these basic building blocks are at a proper semantic level, i.e., the primitives reflect the essential components in coordination work at a level ‘natural’ to the user.

### 11.3.4 User interface

As I have described elsewhere, design of modern user interfaces is difficult (Carstensen, 1993b), and it is far beyond the scope of this dissertation to discuss it in any detail. Furthermore, it should be said, that BRaHS is not very visionary with respect to the user interface design. The user interface in BRaHS should not be considered exemplary in any sense. Thus, only a few overall recommendations, apart from those already mentioned in the previous sub-sections, will be presented.

The interface design of a coordination mechanism should, of course, follow the standards and styles used for designing the other applications the actors are using. Since the coordination mechanisms will be used in parallel with other applications, the dialog should work similarly. The basic heuristics for good interface design—as described in for example Nielsen (1993)—should be applied here as well.

It is important to notice, that this last recommendation can be somehow problematic, and requires a great deal of common sense from the designer applying it. Most of the well-established heuristics within the field of Human-Computer Interaction (e.g., be consistent, provide feed-back, allow undo, user in control, etc.) are based on a single user systems, can therefore be problematic, when being applied on a multi-user situation. There might, for example, be situations involving several actors collaborating via a system where it would be impossible to define what undo from one actor should mean. Hewitt and Gilbert (1993) provides a list of examples of this kind of problems, when applying single user heuristics on user interfaces for CSCW systems.

Another overall requirement for the actor-coordination mechanism dialog is, that it should be tailorable. The user interface should be designed so that the users can smoothly change the system when switching from single user use to multi-user use. Dewan and Choudhary (1991) provides an interesting set of user interface primitives that can be used for designing tailorable CSCW applications. Designing this type of interfaces is, however, extremely complicated, and it will not be discussed further here.

Work flow and protocol structures should be presented and manipulated visually whenever this is possible. It is much easier for actors to overview and understand flow models, if they are visualized by means of graphs or similar visual components. The same goes for the presentation of aggregated monitoring information. Human actors find it easier to monitor graphically visualized information, especially if changes in the state of affairs occur frequently and dynamically.

As mentioned previously, it should be carefully considered, if roles should be used for structuring the actor-coordination mechanism dialog. If this is the case, it should be clearly visible from the interface what the current role is, and a smooth switch between the different roles must be provided.

## 11.4 Evaluation of coordination mechanisms

When talking about test and evaluation of prototypes here, two aspects of the prototype are essential: One is the evaluation of the usability of the coordination mechanism, i.e., how the functionality provided fits the actual needs. The second regards the user interface, i.e., how easy is the system to learn and use.

As mentioned in section 9.1 the prototype (BRaHS) was evaluated through discussions with colleges, an informal presentation at Foss Electric, and a rather unstructured heuristic evaluation session (cf. e.g., Nielsen, 1994) with Liam Bannon as the external expert. This would not have been sufficient if BRaHS was intended to be a prototype of a system to be implemented at Foss Electric.

Regarding evaluation of coordination mechanisms, it should first and foremost be recommended to involve the users much more actively in the design process. This will provide current feed back for the design process. A huge body of literature on, how this can be organized, exists within the areas of Human-Computer Interaction (e.g., Gould and Lewis, 1985) and Participatory Design (e.g. Ehn and Kyng, 1987).

One of the best and most useful techniques for evaluation of the user interface and the functionality provided is ‘think aloud experiments’. I have, some years ago (in Carstensen, 1986), in detail discussed how such experiments can be organized, and what can be obtained from using these. I will highly recommend this technique to be used for evaluation of coordination mechanisms too. It requires, however, a very carefully designed experiment since the scenario should be set up, so it creates a convincing illusion of several interdependent actors involved.

Another obvious technique to be used for evaluating a prototype of a coordination mechanism is heuristic evaluation (cf. e.g., Nielsen, 1994) involving external experts. This proved to be simple and useful for evaluating the BRaHS prototype.

Although it is highly recommended to use the two classical HCI techniques mentioned above, one important aspect must be noted: Both techniques are based on the basic assumption that the user interface basically can be considered a dialog between *one* actor and *one* system. This goes—as illustrated in the previous section—for many of the design heuristics, models of human-computer interaction, etc. that is described within the HCI literature. Coordination mechanisms will usually basically be a means for interaction between *several* actors. When applying techniques from the field of HCI it should therefore be considered, how the fact that the coordination mechanisms is supposed to support several interacting actors influences the evaluation or the design.

The work on designing and evaluating the BRaHS prototype clearly illustrated, that use of scenarios is probably the most important technique for evaluating coordination mechanisms. Almost all discussions, examples, considerations, etc. took their departure in reflections on, how the prototype should (or would) support a specific work situation. Section 9.3 contained a short example of a scenario of how BRaHS could be functioning. It is highly recommended, that scenarios are used as an evaluation technique. Carroll and Rosson (1992) illustrate, how scenarios can be used for designing and evaluating computer-based systems.

The scenarios should be established and described carefully in collaboration with the future users of the coordination mechanism. It is important, that the scenarios include aspects of concurrency and involvement of several interdependent actors, since these situations are very hard to illustrate by means of the other techniques mentioned. The scenarios can then, after the prototype is sketched, support verifying whether the provided functionality appears to be sufficient and well designed.

As for most other aspects of systems development, evaluation is not a question of finding one powerful technique and use it. Depending of the situation, the best way to evaluate will usually be to use a mixture of approaches like: active user involvement in the design process, use of controlled experiments, application of a structured walk through technique, and use of scenarios for potential use of the coordination mechanism. System evaluation is a complex and difficult task, that needs more detailed planning and better accomplishment than the one conducted in relation to BRaHS, and than the very general recommendations given here indicates.

This chapter has aimed at establishing a set of general recommendations based on the experiences gained from my own analysis and design work. The recommendations are thus based on one case only. Further experiments are required, and certain aspects, e.g., regarding the organization of the design process, need more explicit considerations. This has not been possible in this chapter due to the limitations of the scope in this dissertation, and the limited amount of time that can be spend on a project like this.

# Part IV: Conclusion

# 12. Lessons learned

A man, viewed as a behaving system, is quite simple. The apparent complexity of his behavior over time is largely a reflection of the complexity of the environment in which he finds himself....
(Simon, 1981).

The first 11 chapters in this dissertation have aimed at presenting the problem setting and research questions addressed and the results following from the concrete work on the questions. The three short chapters in this final part are devoted to concluding the work. This is done by presenting the lessons learned from the work, and discussing relevant questions for future research.

Several of the previous chapters have already concluded on the results achieved so far: Chapter 5 on the Concept of Coordination Mechanisms presented the results derived from reflections on much of the work presented in the other chapters; Chapter 7 discussed the lessons learned regarding analysis of coordination work; Chapter 11 summarized experiences attained in relation to modeling coordination and the design of a specific coordination mechanism, and several of the other chapters have included sections reflecting on the results presented. This chapter will mainly summarize the lessons learned. The lessons are organized in relation to: The conceptual framework; Characteristics of coordination work; Analysis of coordination work; Requirements for computer support of coordination; And coordination mechanisms design. It should, furthermore, be noticed, that the 32 lessons listed in the following are partly overlapping. The reason being that the experiences gained have, of course, relations to several of the sections in which the chapter is organized.

## 12.1 The Concept of Coordination Mechanisms

The work on the Concept of Coordination Mechanisms should be regarded as basic research. The aim was to establish a set of concepts for conceptualizing the basic aspects of coordination activities conducted in a cooperative work setting. The intendment of the conceptual framework established was to support systems analysts and designers in understanding how procedures, forms, and other artifacts can be considered mechanisms supporting coordination work, and to assist the design of computer-based systems supporting coordination work. The basic research nature of the work implies, of course, that the conceptual framework is still somewhat preliminary.

The framework is based on the assumption that it is possible and relevant to make an analytical distinction between work and its coordination. Another important distinction is the one between the field of work, the cooperative work arrangement, and the organizational context. The cooperative work arrangement is the actors, machines, etc. that cooperatively transform and control the complex of objects, processes, etc. in the field of work. This is all done in a wider organizational context.

The fundamental concept in the framework is the "coordination mechanism" defined as "a protocol, encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, that stipulates and mediates the coordination of distributed activities" (cf. section 5.3). Since the stipulations are conveyed by a symbolic artifact that mediates the relevant coordination information, the state of the protocol is distinct from the state of the field of work. The purpose of a coordination mechanism will (no matter whether it is computer-based or not) usually be to reduce the complexity of coordination work to be conducted by the involved actors.

The conceptual framework includes, furthermore, a basic model of the essential dimensions of objects of coordination work. This model distinguish between objects of coordination work related to the structures of the work arrangement and objects of coordination work related to the structures of the field of work, and between the stages of 'nominal' and 'actual' (cf. section 5.4). Finally the conceptual framework comprehends a set of overall requirements which coordination mechanisms need to relate to in order to be useful to the actors. These are: malleability (re-programmable mechanisms), local control to the users, visiblity of the protocol embedded, and linkability between the mechanisms (cf. section 5.5).

A summary of the results of the work presented in this dissertation regarding the Concept of Coordination Mechanisms has been structured into the following lessons:

*Lesson 1:* The analytical distinction between ‘coordination work’ and ‘work’ distinction proved useful. It is, however, important to notice, that the distinction is recursive, i.e., what can be considered coordination work for one work arrangement might be the field of work—and thus the “tangible work”—for another work arrangement.

The advantage of the analytical distinction was especially that focus was explicitly directed towards coordination aspects of work.

*Lesson 2:* In order to conceptualize the relevant aspects of a specific work situation, the field of work and the related work arrangement must be defined. This is an iterative process based on the fact, that the field of work and the work arrangement mutually constitute each other.

The iterative process supports the analyst in improving his understanding of the mutual interdependencies among the involved actors.

*Lesson 3:* Coordination mechanisms used and invented in real-life work settings seem to be established bottom-up, i.e., since we cannot assume an complete overall view of the organization, and thereby over the work arrangement, the detailed coordination mechanisms can only be established from the bottom—or to put it differently: the mechanisms can only be precisely defined by the actors involved in the work. This has, of course, implications for how computer-based coordination mechanisms should be designed.

*Lesson 4:* Real life coordination mechanisms are often interrelated. One mechanism might, for example, subscribe to information from another. This implies that a top down approach for establishing detailed coordination mechanisms will be very difficult.

We should, hence, as designers provide a structure from which the involved actors can specify mechanisms and relate (link) these to each other.

*Lesson 5:* The Concept of Coordination Mechanisms proved to be a useful framework for an analysis of a cooperative work setting conducted with the purpose of providing input for designing computer-based support, and it provided support and inspiration for the process of designing a specific coordination mechanism. The conceptual framework should, however, only be seen as one of several required approaches for doing work analysis.

The lessons mentioned here, and the approach taken throughout this dissertation, are related only to the coordination aspects of cooperative work. Cooperative work involves a lot of interesting questions related to the tangible work (the essence of the field of work), such as: How does a team collect the information required for conducting their tasks? How are different approaches to a task combined in cooperative work setting? How are decisions made in collaboration? Etc. These aspects were not addressed. The delimited scope, in the work presented here, only addressed aspects related to how mutually interdependent actors coordinate their distributed activities. Due to the limited scope, the temporal and spacial aspects of coordination mechanism are not sufficiently covered in the Concept of Coordination Mechanisms.

The development of the conceptual framework was, apart from the empirical studies, also based on litterature surveys (cf. section 2.3 and 5.6). A brief conclusion from this was that, certain approaches within the sociological and organizational field (e.g., La Porte, 1975b; Mintzberg, 1983; Strauss, 1985; Gerson and Star, 1986) recognize the relevance of addressing work and its coordination distinctly, but it provides no support conceptualizations in order to inform systems design. Construction oriented approaches within CSCW are based on rather rigid non-changeable interaction structures (e.g., Coordinator, see Flores *et al.*, 1988), or flexible structures, but with very atomic concepts at a low semantic level (e.g., Johnson, 1992; Malone *et al.*, 1992). An interesting approach on modifiable work-flows is presented by Swenson *et al.* (1994). Interesting ideas on how to establish flexible representations supporting coordination are presented by Simon Kaplan and associates (e.g., Kaplan *et al.*, 1992b; Fitzpatrick *et al.*, 1995).

The lessons learned regarding the conceptual framework have relevance for both researchers and practitioners. Researchers should recognize that further work on conceptualizing relevant aspects of cooperative work is called for. Furthermore, the relation between the competing approaches of bottom-up establishment of supportive structures versus the top-down establishment, as promoted in, for example, Business Process Reengineering (cf. e.g., Hammer and Champy, 1993), needs to be addressed further. For practitioners, the lessons listed imply that a more detailed and explicit focus on the coordination aspects of work than what is the norm today is required.

## 12.2 Coordination aspects of cooperative work

The analysis conducted in relation to this dissertation provided much input in terms of special characteristics of the cooperative work. These were discussed in chapter 3 and 6.

The field study clearly illustrated that cooperative work often is extremely complex, and it requires a lot of coordination to handle the distributed actors and activities. The actors need to be aware of each other and each others work, and to monitor the state of affairs in the field of work. It was, furthermore, illustrated that it can be hard to communicate about the state of affairs when these are “hidden in abstract representations” like in the software development process observed. Activities such as allocating resources, planning and scheduling tasks, monitoring the state of affairs in the field of work, classifying and prioritizing, distributing information, negotiating requirements, and negotiating priorities, etc. are essential in coordination work. The coordination work is often closely related to conceptualizations of the domain and the aggregations of detailed information. Classification and categorization structures play a central role in the coordination activities.

Summarizing the findings regarding coordination of cooperative work should include the following lessons:

*Lesson 6:* Coordination work is often conducted by means of conceptualizations. The conceptual structures mainly reflect (contain references to) structures of the field of work and structures of the current work arrangement. The representations are usually distinct from abstractions of the field of work itself. A coordination activity will often result in a relation between two or more of these conceptualizations, e.g., an allocation activity results in a responsibility relation between an actor and an informational resource.

*Lesson 7:* The most prominent basic coordination functions conducted, in relation to the conceptualizations mentioned above, are: classification and categorization, monitoring of status and progress in the processes, allocation of resources, relating conceptual structures to each other, meshing the resources and tasks into work plans, and negotiations of classifications, allocations, obligations, etc.

The conceptual structures (cf. lesson 6) and the basic functions mentioned here should be reflected in a system supporting coordination work.

*Lesson 8:* When the complexity of the needed coordination activities increase, the involved actors invent and use mechanisms, for example forms, boards, procedures, working rhythms, etc. These mechanisms stipulate how certain aspects of the work and/or the coordination work should flow, and they mediate relevant coordination information to the actors.

The rigidness of the mode of interaction the mechanism supports spans from the *ad hoc* based to rigid protocols prescribing the flow of work.

*Lesson 9:* Improved rigidness is often considered an advantage. Although all the work was situated, the actors found that more rigid procedures, explicit defined roles, clear-cut agreements, etc. were required in order to cope with the complexity of the coordination. As long as they were only two or three persons collaborating, they could manage by means of *ad hoc* coordination. But as soon as they became more than three interdependent actors, *ad hoc* coordination proved insufficient.

The lessons learned from the field study confirm findings from other similar studies. Artifacts used as means for coordination have been described by for example Edwards *et al.* (1995), Harper *et al.* (1989a), Pycock and Sharrock (1994a), and Gerson and Star (1986).

The lessons have impact on research within the area of CSCW. In CSCW, some approaches claim, that all work is situated and that plans and procedures are only resources for action. Other approaches aim at establishing rigid and pre-specified flows for how work should be conducted. The approach taken and the ideas presented aims at balancing these two rather than going for one of the approaches.

For practitioners the lessons mentioned here imply that systems supporting cooperative ensembles should provide very flexible structures with a large degree of freedom for the users to tailor their systems.

## 12.3 Analysis of coordination

Apart from learning about coordination, the analysis also resulted in a series of experiences regarding the use of techniques, concepts, etc. The work provided input for a better understanding of what should be required from methodologies, techniques, conceptual frameworks, etc. used for analyzing cooperative work. These were discussed in chapter 7, 10, and 11. Basically, we can conclude, that several approaches need to be combined. Apart from this, summarizing the lessons learned regarding support for conducting field studies of coordination of cooperative work should include:

*Lesson 10:* As mentioned in lesson 1, an explicit analytical distinction between work and its coordination proved useful. By explicitly approaching the coordination aspects of the mechanisms (conventions, artifacts, procedures, working rhythms, etc.), a deeper and more coherent understanding of the coordination and the mechanisms was achieved.

*Lesson 11:* Frameworks and schemes for conceptualizing the basic structures along which the coordination work is conducted, and for grasping the basic coordination functions used, are important tools for the analyst. These frameworks should not be considered as prescribing what to look for, rather they should be sources for inspiration, i.e., they are means for more systematic reflection on our observations.

*Lesson 12:* An analysis should be based on several different techniques, typically observation, interviews, document analysis, etc. The field study illustrated, that findings from using each of the techniques supplemented what was found by the others.

This is not new, rather it confirms previous experiences and finding. In general, it can be said, that analytical skills, use of common sense, etc. are basic qualifications for conducting a good analysis, no matter which tools are provided.

*Lesson 13:* Both the static aspects of the domain modeled and the dynamic aspects of the coordination work should be addressed and modeled. The static aspects are important in order to understand the conceptual structures used in the coordination work, whereas the dynamic aspects are important in relation to grasping the very nature of coordination.

*Lesson 14:* To be useful, analysis methodologies must address the informal communication explicitly. The informal and *ad hoc* based communication between collaborating actors is important. The methodologies applied should provide techniques for localizing and characterizing this type of interaction too.

The lessons listed here confirm what others within the software engineering tradition have recognized (e.g., Kraut and Streeter, 1995), and what existing approaches has been criticized for (cf. e.g., the description of the lack of focus on timing and events in the object-oriented techniques in Monarchi and Puhr, 1992). One of the basic modeling guidelines in the object-oriented approaches—reorganize your models in order to avoid events affecting several object classes—might in fact be a fundamental problem in relation to the very nature of coordination work. Coordination work is characterized by consisting of dynamics affecting several structures (cf. section 10.3)

Again, the mentioned lessons have impact on both research and practitioners. Researchers within the software engineering or information systems area need to address the lack of dynamical aspects, or lack of statical aspects, in almost all existing approaches. For practitioners the lessons imply that several different approaches and techniques should be applied when doing work analysis.

## 12.4 Requirements for computer support of cooperative work

Based on the findings from the field study, I have promoted a rather detailed set of requirements for how coordination of software testing and correction should be computer supported (cf. chapter 8). In brief, support for coordination of distributed software testing and correction should include distributed registration of problems, automatic routing of information to the next relevant actor, structures for classifying problems and software modules, and facilities for browsing and aggregating information on problems and their status. Chapter 5 contained a set of general overall requirements for coordination mechanisms including malleability of the mechanisms provided, local control of the execution of the embedded protocol, visibility of the content and protocol structure within the mechanism, and linking between mechanisms. Reflections on what an analyst, designer, or user should consider when setting up requirements are described in section 11.2.

The following lessons summarize what has been learned regarding requirements for computer support of coordination work:

*Lesson 15:* A coordination support system should include data structures reflecting the conceptual structures along which the coordination work is conducted, and provide facilities for making the basic operations related to meshing, allocating, monitoring these structures.

*Lesson 16:* Support of coordination work should include facilities for specifying work flows, and provide facilities for routing relevant information among the involved actors.

As mentioned earlier, the observed actors attempted to increase the use of pre-specified work flows in order to limit the need for *ad hoc* coordination.

*Lesson 17:* A computer-based coordination mechanism should be malleable and provide local control to the users. In order to fulfil this, the protocol embedded in the mechanisms should be visible and manipulatable to the actors. The mechanisms should, furthermore, be tailorable.

This lesson has resulted in one of the most essential general requirements for coordination discussed in section 5.5.

*Lesson 18:* Negotiation should be supported by a coordination support system. Negotiation of obligations, classifications, allocations, etc. is essential in coordination work. Negotiation could be supported by including structures that can be used explicitly for this (e.g., as structured communication channels), or by including a channel for *ad hoc* based informal interaction among the actors.

*Lesson 19:* A coordination support system should offer facilities that enable the involved actors of being aware of each other, and being aware of the state of affairs in the field of work and the work arrangement.

Co-located human actors are extremely good at being aware of each others actions. When computer-based systems are inserted between the collaborating actors, the problem of awareness support becomes essential.

*Lesson 20:* Coordination mechanisms should be linkable. The existing paper and board-based coordination mechanisms all functioned as limited individual and detached mechanisms, but they were usually related to other individual and detached mechanisms. The nature of coordination could be regarded as a network of interrelated mechanisms. Hence, a coordination support system should offer facilities for linking mechanisms together.

*Lesson 21:* Computer-based coordination mechanisms should be presented as extensions to the applications already used by the actors, rather than being designed as individual applications. Coordination work is heavily intertwined with the actual work to be conducted. Hence, a smooth, simple, and invisible transition from the part of the system supporting work to the part of the system supporting coordination needs to be provided.

As for the lessons on the conceptual framework (cf. section 12.1), the requirements work have been heavily inspired by the literature: The attempts to establish architectures and languages for building CSCW applications (e.g., Johnson, 1992; Malone *et al.*, 1992) are based on the basic idea of providing conceptual structures relevant for the actors and some basic manipulations on these. Flexibility and tailorability has also been argued elsewhere (see e.g., Kaplan *et al.*, 1992b; Bentley and Dourish, 1995; Fitzpatrick *et al.*, 1995; Schmidt and Simone, 1995), and the need for informal communication channels was the overall conclusion in Kraut and Streeter (1995). The problems with awareness have been very carefully illustrated by Christian Heath and associates (e.g., Heath *et al.*, 1993; Heath *et al.*, 1995).

From a practitioners point of view these lessons are quite important since they establish a long list of requirements for how a computer system, designed for supporting several interacting and collaborating actors, should be designed. Most existing systems provide only one or a few of the mentioned facilities, for example a work flow structure, or an unstructured communication channel. Almost none of the existing systems are malleable and tailorable to the extent required.

## 12.5 Design of computer-based coordination mechanisms

The work reported in this dissertation included design of a prelimary prototype of a computer-based coordination mechanism supporting the coordination of software testing and correction as it is conducted at Foss Electric. The prototype, called BRaHS, is described in some detail in chapter 9.

The prototype illustrated a number of ideas and provoked a lot of discussions and reflections on design of coordination mechanisms. The lessons regarding the requirements (lesson 15 - 21) affect also the design, and can be considered as lessons on the design of coordination mechanisms. Although the work on the prototype was a rather "quick and dirty" design process, a few central conclusions can be derived from the work. In terms of lessons learned these conclusions are:

*Lesson 22:* It is not clear to what extent flexibility should be provided. Malleability and local control are important. Providing too high a degree of freedom for manipulating the execution can complicate the use of the mechanism too much compared to the benefit. This is a trade-off requiring thorough considerations.

*Lesson 23:* The notion of roles is problematic in the design. Roles are important and essential when characterizing coordination work, but the roles should usually not be explicitly reflected in the user interface.

*Lesson 24:* Supporting awareness by means of computers is complictated. Search facilities for actively finding specific things are not too complicated, but to provide "passive" awareness support of what goes on, and to what extent this should be provided, require a detailed analysis of the awareness needs. Again, this is a trade-off. Too little awareness support might mean too much add hoc interaction, whereas too mush awareness support could mean too many interupts.

*Lesson 25:* Some of the requirements established for coordination support can be achieved by means of existing technology, such as conferencing systems, e-mail. The design should thus also focus on how to integrate these different facilities with the applications used for the actual work, and with the new facilities designed.

*Lesson 26:* The protocol, and the manipulations on the protocol, should normally be presented visually. It appeared to be easier to understand the flow when it was visually represented compared to the verbal descriptions in, for example, the requirements (cf. chapter 8).

The protocol should not have too dominant a presentation in the user interface. To many actor changes to the protocol will be very rare.

*Lesson 27:* To implement a visible and changeable protocol increases the demands on the database design. The database design can benefit from separating aspects regarding the relevant information to be mediated by the mechanism from aspects regarding the default protocol, and aspects regarding the dynamic protocols related to each instance of the mechanism.

*Lesson 28:* A flexible an decomposed architecture should be chosen for coordination mechanisms. Designing a coordination mechanism as one application results in a to inflexible structure. Since the structure of work is often changing, so is the coordination of work. It becomes too problematic to introduce changes if the architecture is organized so that the coordination mechanisms must be re-programmed every time aspects of the work organization are changed. A more flexible structure based on, for example distributed agents reflecting the roles involved, will probably prove to be better.

*Lesson 29:* The well-established heuristics and guidelines for user interface design should be applied when designing coordination mechanisms. It must, however, be considered carefully, because the well-established heuristics within the field of Human-Computer Interaction are usually based on a single user systems. It can therefore be problematic to apply them when designing systems for multiple interdependent users (cf. section 11.3.4).

During the design of BRaHS, several basic questions related to design of coordination mechanisms were left open. The most important of these was, how to decide on propagation of changes to a running protocol: How do we “inform” the rest of the system and the actors using the system? And how do we handle concurrency and interdependencies among instances of the protocols? This has been left open for further research.

The eight lessons mentioned above have massive impact on how practitioners should design coordination mechanisms and organize the design activities.

It has been mentioned several times, that focus in this dissertation has not been on the design process. The work has not explicitly addressed the problem complex of how to organize and conduct the analysis and development work. A few overall lesson regarding how to organize the process have, however, been learned:

*Lesson 30:* The analysis and design work should include use of many different kinds of representations for describing the analysis findings and design ideas. The work has illustrated the need for understanding and modeling both static conceptual structures and their dynamic interaction. The field study indicated further, that the actors were not explicitly aware of their coordination activities. This calls for using several representations in order to get as rich a picture of the coordination as possible.

*Lesson 31:* Much of the coordination work was conducted tacitly, i.e., there is a high degree of uncertainty of, what work is to be supported, and how it should be supported. This uncertainty implies the need for organizing the analysis and development work in a highly iterative manner.

*Lesson 32:* Evaluating coordination mechanisms is very complicated and the commonly used techniques for doing this might prove insufficient. The problem is mainly, that it is difficult to establish situations that illustrate how several interdependent actors can interact by means of a coordination mechanism. To phrase it differently: The work situation that the coordination mechanism should be used in might be impossible to simulate in an experiment or to describe in a session involving the future users.

Lesson 30 and 31 are not surprising. Similar lessons can be found in the computer science and information systems literature. The last lesson should be interesting to researchers within the field of CSCW and information systems. The question of, how to evaluate systems supporting complex cooperative work settings, remains to be answered.

The 32 lessons listed in this chapter should not be read as precise and explicit statements on, how to conceptualize coordination work, how to analyze coordination, how to design a coordination mechanism, or how to organize the development of coordination mechanisms. Rather, the lessons should be seen as an executive summary of what I, as a researcher, an analyst, and a designer, have learned about analyzing and conceptualizing certain aspect of coordination of cooperative work, and about the design of computer support for certain specific coordination activities.

# 13. Future research

"To make a bundle, be a star;
Spread it wide and spread it far.
But if you want to change the sun,
Best begin with Number One."
(Poem reprinted in Weinberg, 1985)

In his entertaining and informative book on consulting, Weinberg (1985) defines "The Harder Law" stating that: "Once you eliminate your number one problem, YOU promote number two" (Ibid., p. 17). I am not in a position to claim, having eliminated problem number one, but the work has definitely promoted (and illustrated) new problems to be attacked. This is along the same line as Heinz Klein, when he, in his informative paper on how to establish a prospectus and workplan for a dissertation states that:

> "When formulating the problems that you plan to attack and thereby defining the scope of your dissertation, keep in mind that there is, indeed, life after your thesis. Hence there is no need to solve all the problems that you are discovering. For a while keep expanding your horizons. But then be MODEST with what can be achieved in a single piece of work..." (Klein, 1989).

Although this dissertation has approached a number questions, some addressed more thoroughly than others, there is, of course, still a long list of open relevant research questions yet to be addressed. I have considered these out of the scope of this dissertation. Several of them are, however, both interesting and relevant to think of as future research to follow the work presented. This chapter will introduce some of these.

As mentioned in the previous chapter, the work on establishing the Concept of Coordination Mechanism has been a basic research oriented activity. Only limited and preliminary conceptualizations existed beforehand. There will, therefore, quite naturally still be a lot of improvement and refinement work to do:

The notion of 'time' in relation to the conceptual framework needs to be explored further. The dimension of time have not, neither in our studies of existing CSCW applications (cf. Andersen *et al.*, 1993) nor in the field study discussed in this dissertation, played a major role. The coordination activities studied have been conducted over a long period of time. The coordination has not been time-critical, or in other ways been closely related to time constraints. The dimension of time is thus not reflected in, for example, the objects of coordination work. Many work situations require that the coordination is handled in real time. This is clearly illustrated in, for example, the reanalysis of the hot rolling mill (Popitz *et al.*, 1957) presented in Schmidt (1994c), or the different control room studies (e.g., Harper *et al.*, 1989b; Heath and Luff, 1991; Fillipi and Theureau, 1993). Several questions need to be addressed: How should time be reflected in the conceptual framework? How should the model of dimensions of objects of coordination work be expanded (with a third dimension?). Which new requirements should be established for computer-based coordination mechanisms due to the fact that some coordination work has serious time related constraints?

Similar to the notion of time, the notion of 'space' (or work context) needs to be explored further. Much coordination work is done with reference to certain aspects of the space in which the work is conducted. Brown and Duguid (1994) have illustrated the importance of the working context when cooperating, both the 'centre' aspects and the 'periphery' aspects. This opens questions to be answered similar to those raised regarding 'time': What does the notion 'work context' include, and how is it organized? Heath *et al.* (1995) are, for example, very critical to the approach taken by Brown and Duguid. Which specific requirements should be raised for computer support?

The completeness and coherence of the dimensions of objects of coordination work should also be studied further. The literature used as input for the work presented here, and the empirical studies carried out, have pointed at the dimensions included in the model illustrated in figure 5-2. Other types of work domains, or differently organized work arrangements, might have coordination mechanisms based on, or including, other dimensions of objects of coordination work. Further field studies, and surveys of literature describing cooperative work settings, are required in order to refine the model of dimensions of objects of coordination work.

As mentioned in chapter 7, the operation lists were not particularly useful as candidates for elemental operations of the coordination work within the work setting studied. A more coherent list including better detailed descriptions of, how the operations manipulate the objects of coordination work is required in order to improve the efficiency of the model as an analysis tool.

Better methodological support for guiding the work analyses is also required. The conceptual framework, as it is now, supports a work analysis by providing concepts for structuring the findings, and by offering inspiration for the analyst. There is, however, no methodological support, i.e., there are no prescriptions guiding the analyst in: Where to start, what kind of observational studies to carry out, what kind of interviews to plan, which types of artifacts to look closer at, etc. I have previously, in collaboration with Kjeld Schmidt, worked on establishing a conceptual framework for supporting work analysis including methodological guidance (Schmidt and Carstensen, 1990; Carstensen and Schmidt, 1993a; Carstensen and Schmidt, 1993b). A systemic approach is, as for the Concept of Coordination Mechanisms, pertained in the Work Analysis methodology. The Work Analysis was used as inspiration for the Foss Electric field study, and have been applied with some success earlier (e.g., Carstensen, 1993a; Simonsen, 1994). It would be obvious, in a future research project, to try to integrate the ideas and concepts of coordination mechanisms into the conceptual framework provided by the Work Analysis, and to expand the methodological support to include prescriptions and guidance for, how to analyze the coordination aspects of the work arrangement studied.

Methodological support raise another interesting and complicated question: What information about the cooperative aspects of a work setting is sufficient and useful for a process designing computer-based support for a cooperative work setting. Although the future users should be involved in the design of systems supporting the cooperative aspects of the work (cf. e.g., Kyng, 1991), we need as analysts and designers, knowledge about the work and the work setting to be supported. A work analysis process can go on forever, and we will constantly expand our understanding of the characteristics of the work and the work setting. A never ending analysis process is, of course, not possible. We therefore need a better understanding of, what type of information is sufficient from a pragmatic point of view. Some analysis approaches have implicitly, by the concepts and techniques they provide, defined a set of criteria of relevans (for example the object-oriented methodologies, see e.g., Coad and Yourdon, 1991). Others have illustrated and discussed, how the ethnographically based methods can be useful (e.g., Hughes *et al.*, 1992; Suchman, 1995). These methodologies do not apply a fixed set of concepts, and the analyses is difficult to abstract from the actual settings. As argued by Lucy Suchman (1995):

> "...by definition normative accounts represent idealizations or typifications. As such, they depend for their writing on the deletion of contingencies and differences. [...] Problems arise, however, when normative representations are either generated at a distance from the sites at which the work they represent goes on or taken away from those sites and used in place of working knowledges." (p. 61).

Both positions are, however, problematic: The methodologies with a fixed set of concepts have too much of a 'Prucustes approach' (cf. Carstensen and Schmidt, 1993a) excluding everything that do not fit. And the ethnographically inspired methods do not seriously address the problem of limited resources and time available for analysis in real life software development. Further work on what is required and useful information for informing design of systems supporting cooperative work is called for. Such work was started in Hughes (1993). It was concluded, that it is needed to "examine the conceptual basis for viewpoints and how these may be used to allow different perspectives from different theoretical perspectives to be contrasted and compared in the development of requirements" (p. 241). There is still much to do.

In relation to the analysis methodology support discussion, the use of object-oriented methodologies should be mentioned. As argued in chapter 10, there are some good arguments for thinking of coordination mechanisms in terms of interrelated object classes. It was concluded that the OOA&D methodology provided good support for specifying the static aspects of the model, but the dynamic aspects of the problem domain modeled disappeared in the model. There is no doubt, that the object-oriented methodologies will become common in software development in the near future. Research on how to better represent the dynamic aspects of coordination work in object-oriented models is therefore highly demanded. A starting point could be, to take a closer look at the ideas for handling behavioral complexity by means of the concepts of 'contracts' and 'collaboration graphs' suggested by Wirfs-Brock *et al.* (1990). Also the possibilities for attaching formalisms like state-transition diagrams or Petri-nets to the existing formalisms need further consideration.

Regarding the construction oriented aspects, it would be obvious to suggest design, implementation, use, and evaluation of a number of prototypes of computer-based coordination mechanisms. The design of BRaHS (cf. chapter 9) provided a lot of input for improving the conceptual framework, and for refining my understanding of requirements for supporting the design process. To establish a sufficient body of experiences, a number of experiments on designing and evaluating coordination mechanisms within a number of different work domains having different basic characteristics are required. For example, what will happen, if computer-based coordination mechanisms are going to be used for coordinating time critical work (e.g., navigating a large tanker), or for coordinating activities between actors that do not have a well-defined common field of work (e.g., a group of university teachers and researchers)? To develop the Concept of Coordination Mechanisms further, a series of practical experiences with design of coordination support systems is needed.

In relation to these practical experiences, it would also be interesting and relevant to look at other systems and frameworks in order to get a deeper understanding of the differences and commonalities among the approaches. To establish a basis for comparison, it could be interesting to have identical coordination support system implemented by means of several “approaches”. The most obvious “platforms” to try out would be OVAL (Fry *et al.*, 1992; Malone *et al.*, 1992), Regatta (Swenson *et al.*, 1994), ConversationBuilder (Kaplan *et al.*, 1992b; Bogia *et al.*, 1993a), and the continuation on the work on ConversationBuilder, WORLDS (Fitzpatrick *et al.*, 1995). The work on the conceptual framework would benefit a lot from thorough comparisons with these (and other) concepts.

Two architectural questions should be explored further: A group around Kjeld Schmidt and Carla Simone (involving myself) has been working on a three layered architecture for coordination mechanisms, having an α-level presenting the running instance of the mechanism, a β-level offering a set of structures for constructing domain specific mechanisms, and a γ-level providing basic building blocks (at an operating systems level) for designing coordination mechanisms (cf. e.g., Schmidt *et al.*, 1993; Simone *et al.*, 1994; Schmidt and Simone, 1995). This architecture needs further refinement, and the ideas of, how to develop it into a agent-based framework (see Divitini *et al.*, 1995a; Divitini *et al.*, 1995b) for design of coordination mechanisms should be elaborated much further.

The second architecture related question concerns the ideas mentioned previously: Coordination mechanisms should be extensions of the applications used in the work domain rather than isolated applications. It should be thoroughly addressed, how this can be organized.

Another very interesting question which has emerged several times during the work presented in this dissertation is the question of how to design the interaction between the human actors and the coordination mechanism. As illustrated in both chapter 11 and 12, the guidelines and heuristics for user interface design established within the Human-Computer Interaction field are only partly valid for design of user interfaces for systems to be used by several interdependent actors. In order to establish a useful set of guidelines, the literature discussing user interface aspects of CSCW systems (e.g., Dewan and Choudhary, 1991; Hewitt and Gilbert, 1993; Manohar and Prakash, 1995) should be carefully studied. The results from this should be related to thorough experiments with some of the other well-established HCI heuristics, as they are presented in, for example, Shneiderman (1987), Nielsen (1993), or Dix *et al.* (1993).

A careful reconsideration of some of the models used for understanding the interaction between users and computer systems (e.g., Norman's Gulf-of-execution / Gulf-of-evaluation model, cf. Norman, 1986) is also required. These models are based on a direct interaction between one actor and a computer system, but, what is the validity of these models, if the computer system is used as a medium through which a number of actors are interacting? This and related questions should be investigated further.

Finally, the question of how to evaluate coordination mechanisms needs to be addressed. As discussed in section 11.4, most of the established techniques for evaluating usability of computer-based systems have problems in addressing the concurrency and user interdependence that characterizes coordination activities. It should be investigated to what extent we can use experimental techniques (e.g., "Think aloud" experiments, cf. Carstensen, 1986), and how experiments evaluating the usability of coordination support systems should be set up. One of the few relevant alternatives to experiments is the use of scenarios. Further elaboration on how scenarios can be used for informing the design of computer-based coordination support systems is required. As argued by Kyng (1995) scenarios must be open-ended, rather than providing algorithm-like prescriptions for the system use. In relation to designing and evaluating coordination mechanisms, it is perhaps even more important, that the scenario techniques to be applied have support of coping with, and illustrating, the concurrency, the user interdependences, the possible time constraints, etc. that characterizes the coordination to be supported. How such scenarios can be organized should be explored.

Each of the research questions mentioned above contain many problems and open questions. It requires a long series of research dissertations to deal thoroughly with any of the listed problems, and they should only be regarded as examples, being interesting and relevant continuations of my work.

# 14. Conclusion

“The more you say,
the more cluttered it becomes,
and the closer you get to the truth”
(my horoscope on the
23. January 1995).

This dissertation has presented the work I have conducted over the last three years. The aim was to discuss what characterizes the coordination of complex cooperative work, conceptualizing certain aspects relevant for analyzing and designing computer-based support of coordination work, analyze coordination aspects of a cooperative work setting, and illustrate requirements for, and concrete design of, computer-based support.

Apart from the current dissertation, the work has resulted in a number of technical reports, conference proceedings papers, and journal papers. A list of the relevant papers produced during the thesis work can be found in Appendix A.

The results of the thesis work is first and foremost a central contribution to the establishment of a conceptual framework. The framework has been established in collaboration with colleges at Risø and University of Milano. The contribution presented is one of several (cf. chapter 5). The Concept of Coordination Mechanisms is a conceptual frame for approaching mechanisms supporting coordination through symbolic artifacts that mediate relevant coordination information among the involved actors, and embed a protocol that stipulates the coordination of distributed activities. The conceptual framework includes a basic model of the essential dimensions of objects of coordination work, separated in aspects related to the structures of the work arrangement and to the field of work. Included in the conceptual framework is, furthermore, a set of overall requirements regarding malleability, local control, visiblity, and linkability of coordination mechanisms.

The most important input for establishing the conceptual framework was the field study conducted at Foss Electric. The software design and testing part of a large scale manufacturing project involving a multitude of people with different areas of competence was studied. The actors were mutually interdependent, and they spent a lot of time coordinating their activities. The field study and data analysis were based on qualitative data collection techniques such as interviews observations, documentation analysis, and project meeting participation. The work setting has been characterized, and a number of paper-, board-, and work-cycle-based coordination mechanisms has been described and analyzed. Especially a form based mechanism for coordinating the distributed activities related to testing and correcting software has been carefully described and analyzed. This was done by means of the concepts of coordination mechanisms. The analysis illustrated, that the coordination activities were mainly based upon conceptualizations of structures in the field of work and structures reflecting the cooperative work arrangement. Activities such as allocating resources, planning and scheduling tasks, monitoring the state of affairs, classifying and prioritizing, distributing information, and negotiating were essential. The field study, furthermore, illustrated how actors, involved in cooperative work settings facing problems in coping with the complexity of the coordination work, invent and use coordination mechanisms. These mechanisms force an extended formalization of the coordination activities by prescribing how the coordination must be conducted and which information that should be mediated to the involved actors. Several examples of such mechanisms invented by the actors have been provided.

Based on the analysis, I have discussed needs for computer support of the coordination of software testing and correction work. A set of general requirements for computer support has been provided. These requirements included support to: Ensure that all registered bugs are treated, route coordination information to all involved actors, provide an overview of the state of affairs of testing and correction activities, and support of a process shifting between being distributed and centralized treatment. The requirements were illustrated by presenting a series of interactions between actors and the mechanism. These illustrated how a mechanism could fulfill the requirements. In order to outline a computer-based coordination mechanism, a number of decisions regarding the detailed requirements and the design were made related to the classification structures.

Based on the requirements, a preliminary prototype of a computer-based coordination mechanism supporting decentralized bug reporting and information routing has been described. The aim was to illustrate how the basic components in a coordination mechanism (artifact, protocol, etc.) could appear in a concrete system, and to provide an example of, how the ideas of visibility, local control, and routing support could be designed. The prototype was introduced and discussed by means of the concepts within the conceptual framework.

Use of object-oriented techniques is a central trend in the area of software development, and several basic characteristics of the coordination mechanisms studied called for an object-oriented modeling technique. I have therefore presented an attempt to model the bug form mechanism by means of object structures. The experiment should explore, what the possibilities and limitations for modeling computer-based coordination mechanisms using an object-oriented analysis methodology were. The conclusion from the experiment was, that the methodology applied provided good support for specifying the static aspects of the model, but modeling the dynamic aspects of the problem domain was insufficiently supported. These aspects disappear in the models. Formalisms and techniques explicitly addressing the dynamic aspects of the coordination should be integrated in the object-oriented methodologies.

Apart from the contribution mentioned above, this dissertation has provided an overview of some of the relevant literature within CSCW and related areas. A series of discussions have also been presented. These discussions departured from experiences from analyzing the S4000 work setting at Foss Electric, from establishing a requirements specification for computer-based coordination support, from designing a prototype of a computer-based coordination mechanism, and from evaluating the prototype.

Conclusions from these discussions indicate, that The Concept of Coordination Mechanisms appeared to be a useful tool for analyzing coordination aspects of cooperative work. It must, however, be combined with other approaches and the methodological support should be improved. The conceptual framework also gave inspiration to the design of a coordination mechanism. Especially the general requirements proved useful. The basic operations related to the objects of coordination work were, however, not as supportive as expected. The final prominent conclusion was, that sufficient evaluation of coordination support systems requires use of more advanced and complex techniques than what is commonly used in software development. The overall conclusions and lessons learned are discussed in more detail in chapter 11.

A number of obvious, interesting, and relevant continuations of the work presented here have been proposed (in chapter 13). Each of the topics mentioned would require a series of dissertations to cover thoroughly. So, Heinz Klein (1989) was absolutely right: There is, indeed, life after your thesis!

Risø, December 1995
Peter Henrik Carstensen

# Danish summary

Denne afhandling dokumenterer de forskningsaktiviteter jeg har gennemført i forbindelse med mit Ph.D.-studie. Arbejdet blev påbegyndt i januar 1993 og har varet tre år. Studiet har været gennemført ved Roskilde Universitet, Datalogisk Afdeling, med Finn Kensing som vejleder.

I store dele af erhvervslivet kan man konstatere øgede krav til produktionstid, forøget kompleksitet af de produkter der fremstilles eller de services der ydes, større krav til produktkvalitet, etc. Disse krav leder ofte til at flere aktører involveres i arbejdsprocesserne: Større kooperative arbejds-arrangementer opstår, og aktørerne er ofte gensidigt afhængige i deres arbejde. Dermed øges kravene til koordinationen af de distribuerede aktiviteter.

Denne afhandling ligger indenfor forskningsfeltet Datamat-støttet kooperativt arbejde (CSCW - Computer Supported Cooperative Work). Afhandlingen omhandler specielt, hvorledes koordinationsaktiviteter kan understøttes af datamat-baseret teknologi. Der fokuseres på tre hovedområder indenfor dette:

(1) Hvorledes kan vi analytisk forstå koordinationsarbejde med henblik på at udvikle datamatiske støttesystemer?

(2) Hvilke krav må opstilles for datamat-støtte af koordination af kooperativt arbejde?

(3) Hvordan kunne et datamat-baseret system til støtte af koordination udformes?

Arbejdet har i høj grad været empirisk drevet. Der er specielt gennemført et større feltstudie på Foss Electric. Her har det været studeret, hvorledes en gruppe software designere organiserede deres samarbejde i forbindelse med udvikling af et stort software kompleks som skal styre et kompliceret instrument til måling af mælkekvalitet. Softwaredesignernes håndtering af koordinationen af aktiviteterne omkring test og fejlrettelse af software blev grundigt studeret og beskrevet. Derudover er der blevet bygget en horisontal prototype af en datamat-baseret mekanisme som skal støtte koordinationen aktiviteterne omkring test og fejlrettelse.

Et af de primære resultater er bidragelsen til opstilling af en begrebsramme til at forstå og beskrive koordinationsmekanismer. Ideerne om koordinationsmekanismer udspringer fra Kjeld Schmidt's arbejder (se for eksempel Schmidt, 1994c). En koordinationsmekanisme er et artefakt (formular, organisatorisk procedure, edb-system, etc.) som medierer relevant koordinationsinformation imellem de involverede aktører og som har en indlejeret protokol, der stipulerer hvorledes (dele af) arbejdet eller dets koordination skal forløbe. Ideerne i begrebsrammen er baseret på nogle grundlæggende antagelser og konceptualiseringer: Det antages at vi analytisk kan skelne imellem egentligt arbejde og den koordination som er nødvendig fordi flere aktører er involveret. Dette fører til en fundamental forståelse af et kooperativt ensemble som en gruppe aktører der samarbejder i forhold til et givet arbejdsfelt. Et kooperativt arbejdssystem er således defineret i forhold til arbejdet snarere end i forhold til organisatoriske strukturer.

Begrebsrammen kan bruges som støtte for en arbejdsanalyse gennemført med henblik på at bygge datamat-baseret koordinationsstøtte. Det omtalte feltstudie har dels prøvet at benytte begrebsrammen som et analytisk værktøj, dels været brugt som empirisk grundlag for en lang række forbedringer og justeringer af begrebsrammen.

Der er endvidere, i tilknytning til begrebsrammen, etableret en række overordnede generelle krav til koordinationsmekanismer. Disse krav kan benyttes som støtte og inspiration i en design-orienteret process. De overordnede krav handler specielt om modificerbarhed af koordinationsmekanismer, lokal kontrol til de involverede aktører, synlighed af den indlejrede protokol, og faciliteter til sammenkobling af koordinationsmekanismer. Disse, og krav udledt af feltstudiet, har dannet grundlag for designet af en illustations-prototype.

Afhandlingen er grupperet i fire dele:

**Første del** etablerer rammerne for arbejdet. Problemstillingen introduceres og afgrænses: Afhandlingen tager udgangpunkt i et systemisk perspektiv inspireret af Arbejdsanalysen (Schmidt and Carstensen, 1990), og fokus er på konceptualisering med henblik på design af datamatiske systemer til koordination. Aspekter som sociale strukturer, magtforhold i brugerorganisationen, brugerinvolvering i systemudviklingen, mv. er ikke explicit adresseret. Der gives endvidere (i kapitel 2) en oversigt over dele af litteraturen inden for CSCW-feltet. Første del afrundes med en introduktion til Foss Electric casen.

**Anden del** er orienteret imod konceptualisering af de centrale strukturer af koordinationsarbejde og hvorledes koordinationsmekanismer studeret igennem en arbejdsanalyse kan analyseres og beskrives. Først (i kapitel 4) diskuteres baggrunden for, og centrale karakteristika ved, kooperativt arbejde. Kooperativt arbejde opstår bl.a. pga. krav om øget kapacitet, involvering af forskellige specialer, og ønsker om at kombinere flere forskellige perspektiver. Forskellige tilgange til koordinationsarbejde fra sociologi, organisationsteori, og datalogi diskuteres kort. Den konceptuelle begrebsramme for koordinationsmekanismer introduceres sammen med en diskusion af overordnede krav til 'real life' koordinationsmekanismer. Som nævnt omhandler kravene modificerbarhed, lokal kontrol, synlighed af protokollen, og sammenkobling.

I kapitel 6 gennemgåes, hvorledes en fejlrapport-formular, og den tilhørende procedure for formularens anvendelse, kan opfattes og beskrives som en koordinationsmekanisme. Formularen blev udviklet og anvendt af software-udviklerne på Foss Electric. Det illustreres hvorledes formularen, gennem en formalisering af arbejdsforløbet og en mediering af relevante informationer, reducerede aktørernes overhead af koordinationsaktiviteter. Analysen påpeger desuden, at koordination i høj grad baseres på konceptualiseringer af strukturer i arbejdssystemet og arbejdsfeltet.

Anden del afrundes med en diskusion af, hvad der kræves af metoder som skal støtte analyse af koordinationsarbejde. Desuden reflekteres over brugbarheden af begrebsapparatet som analyseværktøj. Helt overordnet konkluderes det, at den analytiske skelnen imellem arbejde og dets koordination er meget anvendelig, at de konceptualiseringer begrebsrammen tilbyder er anvendelige—men skal videreudvikles, at analyse af koordination ikke kan gøres ved hjælp af observationsbaserede teknikker alene, samt at analyse af kooperativt arbejde bør baseres på anvendelse og kombination af flere forskellige perspektiver og typer af konceptualiseringer.

**Tredie del** er konstruktivt orienteret. Først (i kapitel 8) opstilles en serie krav til, hvad en datamat-baseret koordinationsmekanisme til håndtering af koordination af software test- og rettelse skal understøtte. De overordnede krav omhandler: 1) adgang til strukturer som repræsenterer aktører, software moduler, planer, registrerede fejl, og klassifikationsstrukturer, 2) rutening af relevant information imellem aktørerne, 3) faciliteter til at monitorere fremdrift og status, 4) faciliteter til allokering af resourcer, og 5) kanaler til kommunikation og forhandling imellem de involverede aktører. Kravspecifikationen skitserer endvidere, hvorledes interaktionen imellem koordinationsmekanismen og dens brugere kunne forløbe.

En preliminær prototype præsenteres herefter. Der er i prototypen lagt vægt på at illustrere, hvorledes kravene om modificerbarhed, lokal kontrol og synlighed af den indlejrede protokol kunne opfyldes. Prototypen og et tænkt scenarie for dens brug beskrives, og designet diskuteres ved hjælp af begrebsrammen.

Koordinationsmekanismer har visse centrale karakteristika som peger på, at det kunne være frugtbart at modellere dem objekt-orienteret. De er sammenkoblede "isolerede" mekanismer med en selvstænding datastruktur og funktionalitet. Dette sammenholdt med det faktum, at objektorienterede metoder er meget populære for tiden, er baggrunden for et eksperiment beskrevet i kapitel 10. Her modelleres fejlrapport-mekanismen fra Foss Electric ved hjælp af teknikkerne fra OOA&D (Mathiassen *et al.*, 1993). Konklusionen på dette eksperiment er, at OOA&D-teknikkerne hjalp til at modelere de statiske aspekter af koordinationsarbejdet, mens de dynamiske aspekter—som er helt centrale at forstå og modelere koordiantionsarbejde—ikke repræsenteres tilfredsstillende.

Tredie del afrundes med en række refleksioner over, hvad jeg har lært om design af datamat-baserede koordinationsmekanismer. Der gives en række rekommendationer. De vigtigste af disse omhandler, hvorledes de overordnede krav bør udmøntes, hvordan centrale tommelfingerregler fra HCI-feltet (Human-Computer Interaction) kan benyttes—men kun med omtanke, samt hvilke specielle krav egenskaberne ved systemer til støtte af kooperativt arbejde stiller til evaluering og afprøvning af systemerne.

**Fjerde del** er en relativ kort opsamling og konklusion på hele afhandlingen. Først gives et koncentreret resumé i form af 32 "lessons learned". Herefter (i kapitel 13) diskuteres, hvorledes arbejdet naturligt kunne fortsættes. Der peges på områder som for eksempel integration af begrebs-apparatet i en egentlig analysemetode (her er Arbejdsanalysen oplagt), implementering og evaluering af en række prototyper, mere grundige sammenligninger med andre perspektiver beskrevet i CSCW-litteraturen, og undersøgelser af, hvorledes konklusioner om brugergrænseflade design fra traditionel HCI kan anvendes ved design af CSCW-applikationer.

En del af det arbejde, som er beskrevet i denne afhandling, er endvidere blevet publiseret forskellige andre steder. Enten som tekniske rapporter, reviewede konference-papirer eller tidskriftsartikler. Disse papirer har fungeret som input for visse af kapitlerne. En samlet liste over de papirer, jeg har været involveret i at producere gennem de sidste tre år, og som er relevante i denne sammenhæng, er listet i appendix A.

Risø, December 1995
Peter Henrik Carstensen

# Bibliography

Ackerman, Marc S., and Tom W. Malone: “Answer Garden: A Tool for Growing Organizational Memory,” in *Proceedings of ACM Conference on Office Information Systems,* ACM Press, 1990, pp. 31-39.

Andersen, Hans, Peter Carstensen, Betty Hewitt, and Carsten Sørensen: “Aspects, Collage, Active Memory, OVAL,” in *Computational Mechanisms of Interaction for CSCW*, edited by C. Simone and K. Schmidt, Department of Computing, Lancaster University, Lancaster, England, 1993, pp. 217-238.

Andersen, Hans H. K.: “Classification schemes: Supporting articulation work in technical documentation,” in *ISKO ’94. Knowledge Organisation and Quality Management, Copenhagen, Denmark, June 21-24, 1994*, edited by H. Albrechtsen, 1994.

Andersen, Ib, Finn Borum, Peer Hull Kristensen, and Peter Karnøe: *Om kunsten af bedrive feltstudier [The Art of Conducting Field Studies]*, Samfundslitteratur, Copenhagen, 1992.

Andersen, N. E., F. Kensing, J. Lundin, L. Mathiassen, A. Munk-Madsen, M. Rasbech, and P. Sørgaard: *Professional Systems Development — Experience, Ideas, and Action*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

Anderson, Bob, Graham Button, and Wes Sharrock: “Supporting The Design Process Within An Organisational Context,” *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work*, *Milan*, edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, 1993, pp. 47–59.

Baecker, Ronald M. (ed.): *Readings in Groupware and Computer-Supported Cooperative Work*, Morgan Kaufmann, San Mateo, 1993.

Baecker, Ronald M., Dimitrios Nastos, Ilona R. Posner, and Kelly L. Mawby: “The User-centred Iterative Design Of Collaborative Writing Software,” *INTERCHI’93 Conference on Human Factors in Computing Systems, Amsterdam*, edited by S. Ashlund *at al.*, ACM Press, 1993, pp. 399-405.

Bannon, Liam: “CSCW: An Initial Exploration,” *Scandinavian Journal of Information Systems*, vol. 5, 1993, pp. 3-24.

Bannon, Liam, and Kjeld Schmidt: “CSCW: Four Characters in Search of a Context,” in *EC-CSCW ’89. Proceedings of the First European Conference on Computer Supported Cooperative Work, Gatwick, London, 13-15 September, 1989*, 1989. - Reprinted in *Studies in Computer Supported Cooperative Work. Theory, Practice and Design*, edited by J. M. Bowers and S. D. Benford, North-Holland, Amsterdam etc., 1991, pp. 3-16.

Barber, Gerald R.: “Supporting Organizational Problem Solving with a Work Station,” *TOIS*, vol. 1, no. 1, January 1983, pp. 45-67.

Beck, Eevi E., and Victoria Bellotti: “Informed Oppertunism as Strategy: Supporting Coordination in Distributed Collaborative Writing,” in *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, 13-17 September 1993, Milan, Italy,* edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, Dordrecht, 1993, pp. 233-248.

Beizer, Boris: *Software Testing Techniques,* (Second Edition), Van Nostrand Reinhold, New York, 1990.

Bellotti, Victoria, and Abigail Sellen: “Design for Privacy in Ubiquitous Computing Environments,” in *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, 13-17 September 1993, Milan, Italy*, edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, Dordrecht, 1993, pp. 77-92.

Bentley, Richard, and Poul Dourish: “Medium versus Mechanism: Supporting Collaboration Through Customisation,” in *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work - ECSCW’95, 10-14 September, 1995, Stockholm, Sweden,* edited by H. Marmolin, Y. Sundblad and K. Schmidt, Kluwer Academic Publishers, 1995, pp. 309-324.

Bentley, R., J. A. Hughes, D. Randall, T. Rodden, P. Sawyer, D. Shapiro, and I. Sommerville: “Ethnographically-informed systems design for air traffic control,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992,* edited by J. Turner and R. Kraut, ACM Press, New York, 1992a, pp. 123-129.

Bentley, Richard, Tom Rodden, Peter Sawyer, and Ian Sommerville: “An Architecture for Tailoring Cooperative Multi-User Displays,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992,* edited by J. Turner and R. Kraut, ACM Press, New York, 1992b, pp. 187-194.

Beraneck, Peggy: “Qualitative Research for CSCW: Theory Building Beyond Initial Preconceptions and Frameworks,” in *IRIS 17. Quality by Diversity in Information Systems Research. 17th Information systems Research seminar In Scandinavian, August 6–9, Syöte Conference Centre, Finland*, edited by P. Kerola, A. Juustila and J. Järvinen, University of Oulu, 1994, vol. 1, pp. 60-62.

Beyer, Peter, Peter Carstensen, Anker Helms Jørgensen, Rolf Molich, and Finn Hindkjær Pedersen: *Brugervenlige edb-systemer [User-friendly computer systems]*, Teknisk Forlag A/S, Copenhagen, 1986.

Bjerkenes, Gro, Pelle Ehn, and Morten Kyng (ed.): *Computers and Democracy. A Scandinavian Challenge*, Avebudy, Aldershot etc., 1987.

Boehm, B. W.: *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

Bogia, Douglas P., William J. Tolone, Celsina Bignoli, and Simon M. Kaplan: “Issues in the Design of Collaborative Systems: Lessons from ConversationBuilder,” in *Design of Computer Supported Cooperative Work and Groupware Systems. 12th International Workshop on “Informatics and Psychology”, Schärding, Austria, June 1-3, 1993*, 1993a.

Bogia, Douglas P., William J. Tolone, Simon M. Kaplan, and Eric de la Tribouille: "Supporting Dynamic Interdependencies among Collaborative Activities," *COCS '93*, 1993b.

Borstrøm, Henrik, Peter Carstensen, and Carsten Sørensen: *Artifacts Coordinating Concurrent Engineering. A Study of Articulation Work in a Manufacturing Project*, Technical Report, January, 1995. [R-768(EN)].

Borstrøm, Henrik, and Carsten Sørensen: "CAD Models are not Mechanisms of Interaction — Cooperative Aspects of Design For Manufacture," in *Fourth International Conference on Human Aspects of Advanced Manufacturing and Hybrid Automation, Manchester, England, July 6–8, 1994*, edited by P. Kidd, 1994.

Bowers, John: "Understanding Organization Performatively," in *Issues of Supporting Organizational Context in CSCW Systems*, edited by L. Bannon and K. Schmidt, COMIC, Esprit Basic Research Project 6225, Computing Department, Lancaster University, Lancaster, UK, 1993, pp. 49-72.

Bowker, Geoffrey, and Susan Leigh Star: "Situations vs. Standards in Long-Term, Wide-Scale Decision-Making: The Case of the International Classification of Diseases," in *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences, Kauai, Hawaii, January 7-11, 1991*, edited by J. F. Nunamaker and R. H. Sprague, IEEE Computer Society Press, 1991, vol. IV.

Brooks Jr., F. P.: *The Mythical Man-Month — Essays on Software Engineering*, Addison-Wesley, USA, 1982.

Brown, John S., and Paul Duguid: "Borderline issues: Social and Material Aspects of Design," *Human-Computer Interaction*, vol. 9, no. 1, 1994, pp. 3-36.

Bucciarelli, Louis L.: *An Ethnographic Perspective on Engineering Design*, M.I.T., Cambridge, Mass., 1987.

Button, Graham (ed.): *Technology in Working Order. Studies of work, interaction, and technology*, Routledge, London and New York, 1993.

Cambell, Robert L.: "Will the Real Scenario Please Stand Up," *SIGCHI Bulletin*, vol. 24, no. 2, April 1992, pp. 6–8.

Carroll, John M., and Mary B. Rosson: "Getting Around the Task-Artifact Cycle: How to make Claims and Design by Scenario," *ACM Transactions on Information Systems*, vol. 10, no. 2, 1992, pp. 181-212.

Carstensen, Peter: "The Bug Report Form," in *Social Mechanisms of Interaction*, edited by K. Schmidt, Department of Computing, Lancaster University, Lancaster, England, 1994, pp. 187-220.

Carstensen, Peter, Birgitte Krogh, and Carsten Sørensen: "Object-Oriented Modeling of Coordination Mechanisms," in *Proceedings of IRIS'18 'Design in Context', Gjern, Denmark*, edited by B. Dahlbom *at al.*, University of Gothenburg, 1995a, pp. 113-129.

Carstensen, Peter, and Kjeld Schmidt: *The Procrustes Paradigm: A Critique of Computer Science Approaches to Work Analysis*, COMIC Working Paper, Risø National Laboratory, (version 2.0), May, 1993a. [COMIC-Risø-2-1].

Carstensen, Peter, and Kjeld Schmidt: “Work Analysis-Perspectives on and Requirements for a Methodology,” in *Human-Computer Interaction: Applications and Case Studies*, edited by M. J. Smith and G. Salvendy, Elsevir, Amsterdam, 1993b, pp. 575-580.

Carstensen, Peter, and Carsten Sørensen: “The Foss Electric Cases,” in *Social Mechanisms of Interaction*, edited by K. Schmidt, Department of Computing, Lancaster University, Lancaster, England, 1994a, pp. 295-304.

Carstensen, Peter, and Carsten Sørensen: “Requirements for a Computational Mechanism of Interaction: An example,” in *A Notation for Computational Mechanisms of Interaction*, edited by C. Simone and K. Schmidt, University of Lancaster, Lancaster, England, 1994b, pp. 33-80.

Carstensen, Peter, Carsten Sørensen, and Henrik Borstrøm: “Two is Fine, Four is a Mess — Reducing Complexity of Articulation Work in Manufacturing,” in *COOP’95. Proceedings of the International Workshop on the Design of Cooperative Systems, January 25-27, Antibes-Juan-les-Pins, France*, INRIA, Sophia Antipolis, 1995b, pp. 314-333.

Carstensen, Peter, Tuomo Tuikka, and Carsten Sørensen: “Are We Done Now? Towards Requirements for Computer Supported Cooperative Software Testing,” in *IRIS 17. Quality by Diversity in Information Systems Research. 17th Information systems Research seminar In Scandinavian, August 6–9, Syöte Conference Centre, Finland*, edited by P. Kerola, A. Juustila and J. Järvinen, University of Oulu, 1994, vol. 1, pp. 424–438.

Carstensen, Peter H.: *Udvikling af brugervenligt programmel*, DIKU-rapporter, Master Thesis, University of Copenhagen - Institute of Datalogy, 1986. [86/17].

Carstensen, Peter H.: *Functional analysis of the ‘KAVAS work domain’ and requirements for the global functionality of Kaviar*, AIM KAVAS-2 (A2019), 1993a. [UCI 2.1-1].

Carstensen, Peter H.: “Graphical User Interfaces: Easy to use, hard to design,” in *Proceedings of COPE’ IT ‘93, Copenhagen*, Danish Data Association, 1993b, pp. 107-122.

Carstensen, Peter H.: *The Complexity and Articulation of Work — In search for a useful approach for understanding and computer supporting articulation work*, Working Papers in Cognitive Science and HCI, edited by F. Jensager, Centre for Cognitive Informatics, Roskilde University, 1995a. [WPCS-95-8].

Carstensen, Peter H.: “Modeling Coordination Work: Lessons learned from analyzing a cooperative work setting,” in *HCI International ‘95 — 6th International Conference on HCI: Symbiosis of Human and Artifact, Pacifico Yokohama, Japan 9 - 14 July 1995., Yokohama, Japan*, edited by Y. Anzai, K. Ogawa and H. Mori, Elsevier, 1995b, pp. 327-332.

Carstensen, Peter H., and Thomas Albert: “BRaHS: A Computer Based Mechanism Supporting the Coordination of Bug-handling,” in *Demonstrator prototypes of Computational Mechanisms of Interaction*, edited by L. Navarro, University of Lancaster, Lancaster, 1995, pp. 183-218.

Carstensen, Peter H., Carsten Sørensen, and Tuomo Tuikka: “Let’s talk about bugs!,” *Scandinavian Journal of Information Systems*, vol. 7, no. 1, 1995c, pp. 33-53.

Checkland, Peter: *Systems Thinking, Systems Practice*, John Wiley & Sons Ltd., Chichester etc., 1981.

Ciborra, Claudio U.: *Teams, markets and systems. Bussiness innovation and information technology*, Cambridge University Press, Cambridge, 1993.

Coad, Peter, and Edward Yourdon: *Object-oriented Analysis*, (2), Prentice-Hall International, London etc., 1991.

Conklin, Jeffrey: “Capturing Organizational Memory,” in *Groupware’92*, edited by D. Coleman, Morgan Kaufmann, 1992, pp. 133-137.

Conklin, Jeff, and Michael L. Begeman: “gIBIS: A Hypertext Tool for Exploratory Policy Discussion,” in *CSCW ’88. Proceedings of the Conference on Computer-Supported Cooperative Work, Portland, Oregon, September 26-28, 1988*, ACM, New York, N. Y., 1988, pp. 140-152.

Coolican, Hugh: *Research Methods and Statistics in Psychology*, (2), Hodder & Stoughton, London, 1994.

Cruse, D.: “Workflow Process Automation: Beyond Traditional Workflow,” in *Proceedings of Groupware’92,* , edited by D. Coleman, Morgan Kaufmann, 1992, pp. 301-311.

Dahlbom, Bo, and Lars Mathiassen: *Computers in Context — The Philosophy and Practice of Systems Design*, Blackwell Publishers, Cambridge, Massachusetts, 1993.

Dalal, S. R., J. R. Hogan, and J. R. Kettering: “Reliable Software and Communication: Software Quality, Reliability, and Safety,” in *15th International Conference on Software Engineering, Baltimore, Maryland, USA*, IEEE Computer Society Press, Los Alamitos, California, USA, 1993, pp. 425-435.

De Cindio, Fiorella, Carla Simone, Raffaela Vassallo, and Annamaria Zanaboni: “CHAOS: a knowledge-based system for conversing within offices,” in *Office Knowledge: Representation, Management and Utilization*, edited by W. Lamersdorf, Elsevier Science Publishers B.V., North -Holland, Amsterdam, 1988.

Dery, David, and Theodore J. Mock: “Information Support Systems for Problem Solving,” *Decision Support Systems*, vol. 1, 1985, pp. 103-109.

Dewan, Prasun, and Rajiv Choudhary: “Primitives for Programming Multiuser-User Interfaces,” in *Fourth Annual Symposium on User Interface Software and Technology, South Carolina, USA*, ACM Press, 1991, pp. 69-78.

Divitini, Monica, Carla Simone, Kjeld Schmidt, and Peter Carstensen: *A multi-agent approach to the design of coordination mechanisms*, Working Papers In Cognitive Science and HCI, edited by F. Jensager, Centre for Cognitive Informatics, Roskilde University, 1995a. [WPC-95-5].

Divitini, Monica, Carla Simone, and Carla Quaranta Vogliotti: “Coordination Mechanisms in a multi-agent perspective,” in *Demonstrator prototypes of Computational Mechanisms of Interaction*, edited by L. Navarro, University of Lancaster, Lancaster, 1995b, pp. 73-118.

Dix, Alan, Janet Finlay, Gregory Abowd, and Russel Beale: *Human-Computer Interaction*, Prentice Hall, New York, etc., 1993.

Dourish, Paul: “Culture and Control in Media Space,” in *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, 13-17 September 1993, Milan, Italy*, edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, Dordrecht, 1993, pp. 125-137.

Durham, Tony: *Computing Horizons*, Addison-Wesley, Workingham, 1988.

Edwards, Mark B, Dana K. Fuller, and O. U. Vortac: “The role of flight progress strips in en route air traffic control: a time series analysis,” *International Journal of Human-Computer Studies*, vol. 43, no. 3, 1995, pp. 1-13.

Ehn, Pelle: *Work-Oriented Design of Computer Artifacts*, Arbetslivscentrum, Stockholm, 1988.

Ehn, Pelle, and Morten Kyng: “The collective resource approach to system design,” in *Computers and Democracy—A Scandinavian Challenge*, edited by G. Bjerknes, P. Ehn and M. Kyng, Aldershot, Avebury, 1987, pp. 17-57.

Ellis, Clarence A.: “Information Control Nets,” in *Proceedings of the ACM Conference on Simulation, Measurement and Modeling, Boulder, Colorado, August 1979*, ACM Press, 1979.

Ellis, C. A., S. J. Gibbs, and G. L. Rein: “Groupware: Some issues and experiences,” *Communications of the ACM*, vol. 34, no. 1, January 1991, pp. 38-58.

Ellis, Clarence A., and Gary J. Nutt: “Office Information Systems and Computer Science,” in *Computer-Supported Cooperative Work*, edited by I. Greif, Morgan Kaufmann, San Mateo, 1988, pp. 199-247.

Ellis, Clarence A., and Jaques Wainer: “Goal-based models of collaboration,” *Collaborative Computing*, vol. 1, no. 1, 1994, pp. 61-86.

Embley, David W., Robert B. Jackson, and Scott N. Woodfield: “OO Systems Analysis: Is It or Isn’t It?,” *IEEE Software*, no. 7, 1995, pp. 19-33.

Engelbart, Douglas, and Harvey Lehtman: “Working together,” *Byte*, December 1988, pp. 245-252.

Ensor, Bob: “How can we make groupware practical,” in *CHI’90 Human Factors in Computing Systems, Seattle, Washington*, edited by J. C. Chew and J. Whiteside, acm Press, 1990, pp. 87-89.

Fillipi, Geneviève, and Jacques Theureau: “Analyzing Cooperative Work in an Urban Traffic Control Room for the Design of a Coordination Support System,” in *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, 13-17 September 1993, Milan, Italy*, edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, Dordrecht, 1993, pp. 171-186.

Fitzpatrick, Geraldine, Wiiliam J. Tolone, and Simon Kaplan: “Work, Locales and Distributed Social Worlds,” in *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work - ECSCW’95, 10-14 September, 1995, Stockholm, Sweden*, edited by H. Marmolin, Y. Sundblad and K. Schmidt, Kluwer Academic Publishers, 1995, pp. 1-17.

Flavin, Matt: *Fundamental concepts of information modelling*, Yourdon Press, Englewood Cliffs, New Jersey, 1981.

Flores, Fernando, Michael Graves, Brad Hartfield, and Terry Winograd: “Computer Systems and the Design of Organizational Interaction,” *TOIS*, vol. 6, no. 2, April 1988, pp. 153-172.

Floyd, Christiane: “A Comparative Evaluation of System Development Methods,” in *Information Systems Design Methodologies: Improving the Practice*, edited by T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart, North-Holland, Amsterdam, 1986, pp. 19-54.

Flyvbjerg, Bent: *Rationalitet og Magt. Bind I: Det konkretes videnskab [Rationality and Power. Volume I: The Science of the Concrete]*, Akademisk Forlag, Copenhagen, 1992.

Fry, Christopher, Kum-Yew Lai, and Thomas Malone: *Oval*, v. 1.1, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1992.

Gaver, William, Thomas Moran, Allan MacLean, Lennart Lövstrand, Paul Dourish, Kathleen Carter, and William Buxton: “Realizing a video environment: EuroPARC’s RAVE system,” in *CHI ’92 Conference Proceedings. ACM Conference on Human Factors in Computing Systems, May 3-7, 1992, Monterey, California*, edited by P. Bauersfeld, J. Bennett and G. Lynch, ACM Press, New York, 1992, pp. 27-35.

Gaver, William, Abigail Sellen, Christian Heath, and Paul Luff: “One is not enough: Multiple views in a media space,” *INTERCHI’93 Conference on Human Factors in Computing Systems, Amsterdam*, edited by S. Ashlund *at al.*, ACM Press, 1993, pp. 335-341.

Gaver, William W.: “Sound Support for Collaboration,” in *ECSCW ’91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, edited by L. Bannon, M. Robinson and K. Schmidt, Kluwer Academic Publishers, Amsterdam, 1991, pp. 293-308.

Gelperin, D, and B Hetzel: “The Growth of Software Testing,” *Communications of the ACM*, vol. 31, no. 6, 1988, pp. 687-695.

Gerson, Elihu M.: “Work are going concerns,” *Pacific Sociological Meetings*, , 1983, pp. 257-270.

Gerson, Elihu M., and Susan Leigh Star: “Analyzing Due Process in the Workplace,” *TOIS*, vol. 4, no. 3, July 1986, pp. 257-270.

Giddens, Anthony: *The Constitution of Society: Outline of the Theory of Structuration*, Polity Press, Cambridge, 1984.

Glaser, B. G., and A. L. Strauss: *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine Publishing Company, New York, 1967.

Gould, John D., and Clayton Lewis: “Designing for Usability: Key Principles and What Designers Think,” *Communication of the ACM*, vol. 28, no. 3, 1985, pp. 300-311.

Grudin, Jonathan: “Why groupware applications fail: problems in design and evaluation,” *Office: Technology and People*, vol. 4, no. 3, 1989, pp. 245-264.

Grudin, Jonathan: “Groupware and Cooperative Work: Problems and Prospects,” in *The Art of Human Computer Interface*, edited by B. Laurel, Addison-Wesley, 1990.

Grudin, Jonathan: “CSCW: The convergence of two development contexts,” in *CHI ’91. ACM SIGCHI Conference on Human Factors in Computing Systems, New Orleans, April 28-May 2, 1991*, ACM Press, 1991, pp. 91-97.

Grudin, Jonathan, and Leysia Palen: “Why Groupware Succeeds: Discretion or Mandate,” *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work - ECSCW’95, 10-14 September, 1995, Stockholm, Sweden*, edited by H. Marmolin, Y. Sundblad and K. Schmidt, Kluwer Academic Publishers, 1995, pp. 263-278.

Hamlet, Richard: “Special section on software testing,” *Communications of the ACM*, vol. 31, no. 6, 1988, pp. 662-667.

Hammer, Michael: “The OA mirage,” *Datamation*, vol. 30, no. 2, February 1984, pp. 36-46.

Hammer, Michael, and James Champy: *Reengineering the Cooperation. A Manifesto for Bussiness Revolution*, Nicholas Brearley Publishing Limited, London, etc., 1993.

Hammer, Michael, and Marvin Sirbu: “What is Office Automation?,” in *Proceedings: First Office Automation Conference, Atlanta, Georgia, March*, 1980.

Harper, Richard R., John A. Hughes, and Dan Z. Shapiro: *The Functionality of Flight Strips in ATC Work. The report for the Civil Aviation Authority*, Lancaster Sociotechnics Group, Department of Sociology, Lancaster University, January, 1989a.

Harper, R. R., J. A. Hughes, and D. Z. Shapiro: “Working in harmony: An examination of computer technology in air traffic control,” in *EC-CSCW ’89. Proceedings of the First European Conference on Computer Supported Cooperative Work, Gatwick, London, 13-15 September, 1989*, 1989b, pp. 73-86.

Heath, Christian, Marina Jirotka, Paul Luff, and Jon Hindmarsh: “Unpacking Collaboration: the Interactional Organisation of Trading in a City Dealing Room,” in *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, 13-17 September 1993, Milan, Italy*, edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, Dordrecht, 1993, pp. 155-170.

Heath, Christian, and Paul Luff: “Collaborative Activity and Technological Design: Task Coordination in London Underground Control Rooms,” in *ECSCW ’91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, edited by L. Bannon, M. Robinson and K. Schmidt, Kluwer Academic Publishers, Amsterdam, 1991, pp. 65-80.

Heath, Christian, and Paul Luff: “Collaboration and Control. Crisis Management and Multimedia Technology in London Underground Control Rooms,” *CSCW*, vol. 1, no. 1-2, 1992, pp. 69-94.

Heath, C. C., P. Luff, and G. M. Nicholls: “The Collaborative Production of the Document: Context, Genre and the Borderline in Design,” in *COOP’95. Proceedings of the International Workshop on the Design of Cooperative Systems, January 25-27, Antibes-Juan-les-Pins, France*, INRIA, Sophia Antipolis, 1995, pp. 203-218.

Helander, Martin, and Mitsou Nagamachi (ed.): *Design for Manufacturability — A Systems Approach to Concurrent Engineering and Ergonomics*, Taylor & Francis, London, 1992.

Henderson, J. C., and J. G. Cooprider: “Dimensions of I/S Planning and Design Aids: A Functional Model of CASE Technology,” *Information Systems Research*, vol. 1, no. 3, 1990, pp. 227-254.

Herskind, Steffen R., and Henrik S. Nielsen: *Designing Mechanisms of Interaction for Emergency Management*, Working Papers in Cognitive Science and HCI, Centre for Cognitive Informatics, Roskilde University, DK-4000 Roskilde, Denmark, 1994.

Hetzel, Bill: *The Complete Guide to Software Testing*, QED Information Sciences Inc., Wellesley, MA, USA, 1988.

Hewitt, Betty, and G. N. Gilbert: “Groupware Interfaces,” in *CSCW in Practice: In introduction and Case Studies*, edited by D. Diaper and C. Sanger, Springer-Verlag, London, etc., 1993, pp. 31-38.

Hewitt, Carl: “The challenge of open systems,” *Byte*, vol. 10, no. 4, April 1985, pp. 223-242.

Hirschheim, Rudy, and Heinz K. Klein: “Four Paradigms of Information Systems Development,” *Communications of the ACM*, vol. 32, no. 10, 1989, pp. 1199-1216.

Holt, Anatol: “Diplans: A New Language for the Study and Implementation of Coordination,” *TOIS*, vol. 6, no. 2, April 1988, pp. 109-125.

Holt, Anatol W.: “Coordination Technology and Petri Nets,” in *Advances in Petri Nets 1985*, edited by G. Rozenberg, Lecture Notes in Computer Science, edited by G. Goos and J. Hartmanis, vol. 222, Springer-Verlag, Berlin, 1985, pp. 278-296.

Hughes, John (ed.): *Informing CSCW System Requirements*, COMIC, Esprit Basic Research Project 6225, Lancaster University, Lancaster, 1993.

Hughes, John, Dave Randall, and Dan Shapiro: “CSCW: Discipline or Paradigm? A sociological perspective,” in *ECSCW ’91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, edited by L. Bannon, M. Robinson and K. Schmidt, Kluwer Academic Publishers, Amsterdam, 1991, pp. 309-323.

Hughes, John A., David Randall, and Dan Shapiro: “Faltering from Ethnography to Design,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, edited by J. Turner and R. Kraut, ACM Press, New York, 1992, pp. 115-122.

Hughes, John A., Dave Randall, and Dan Shapiro: “From Ethnographic Record to System Design. Some experiences from the field,” *CSCW*, vol. 1, no. 3, 1993, pp. 123-141.

Isaacs, Ellen A., Trevor Morris, and Thomas K. Rodriguez: “A Forum for Supporting Interactive Presentations to Distributed Audiences,” in *CSCW ’94. Proceedings of the Conference on Computer-Supported Cooperative Work, Chapel Hill, North Carolina, October 24-26, 1994*, edited by T. Malone, ACM Press, New York, N.Y., 1994, pp. 405-416.

Ishii, Hiroshi, Minoru Kobayashi, and Kazuho Arita: “Iterative Design of Seamless Collaboration Media,” *CACM*, vol. 37, no. 8, 1994, pp. 83-97.

Ishii, Hiroshi, and Naomi Miyake: “Torward An Open Shared Workspace: Computer and Video Fusion Approach of TeamWorkStation,” *CACM*, vol. 34, no. 12, 1991, pp. 36-50.

Jackson, Michael: *System Development*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

Jacobson, I., M. Christerson, P. Jonsson, and G. Övergaard: *Object-Oriented Software Engineering – A Use Case Driven Approach*, Addison-Wesley Publishing Company, 1992.

Jeffay, K., J. K. Lin, J. Menges, F. D. Smith, and J. B. Smith: “Architecture of the Artifact-Based Collaboration System Matrix,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, edited by J. Turner and R. Kraut, ACM Press, New York, 1992, pp. 195-202.

Johansen, Robert: *Groupware. Computer Support for Business Teams*, The Free Press, New York and London, 1988.

Johnson, Philip: “Supporting Exploratory CSCW with the EGRET Framework,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, edited by J. Turner and R. Kraut, ACM Press, New York, 1992, pp. 298-305.

Johnson, Philip M., and Danu Tjahjono: “Improving Software Quality through Computer Supported Collaborative Review,” in *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, 13-17 September 1993, Milan, Italy,* edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, Dordrecht, 1993, pp. 61-76.

Käkölä, Timo: “Doing by Understanding: embedded systems for understanding coordinated work,” in *Human-Computer Interaction: Application and case Studies. Proceedings of the fifth International Conference on Human-Computer Interaction*, edited by J. Schmidt and G. Salvendy, vol. 2, Orlando, Florida, 1993, p. 973-979.

Kaplan, Simon M., William J. Tolone, Elsa Bignoli, Douglas P. Bogia, and Alan M. Carroll: “Orthogonal Support Aids Collaborative Tasks,” in *Proceedings of Schärding ’92, Austria*, 1992a.

Kaplan, Simon M., William J. Tolone, Douglas P. Bogia, and Celsina Bignoli: “Flexible, Active Support for Collaborative Work with Conversation Builder,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, edited by J. Turner and R. Kraut, ACM Press, New York, 1992b, pp. 378-385.

Keller, Kurt: “Conditions for Computer-Supported Cooperative Work: The Significance of the Psychosocial Work Environment,” *Technology Studies*, vol. 1/2, no. 2, 1994, pp. 242-269.

Kensing, Finn, and Andreas Munk-Madsen: “PD: Structure in the Toolbox,” *Communications of the ACM,* vol. 36, no. 6, 1993, pp. 78-85.

Kensing, Finn, and Terry Winograd: “The Language/Action Approach to Design of Computer-Support for Cooperative Work: A Preliminary Study in Work Mapping,” in *Collaborative Work, Social Communications and Informations Systems*, edited by R. K. Stamper *at al.*, Elsevir (North-Holland), 1991, pp. 311-331.

Keyser, Véronique De: “Why field studies?,” in *Design for Manufacturability — A Systems Approach to Concurrent Engineering and Ergonomics,* edited by M. Helander and M. Nagamachi, Taylor & Francis, London, 1992, pp. 305-316.

Klein, Heinz K.: *The Prospectus and Dissertation Workplan in Information Systems Research*, School of Management, SUNY Binghamton, 1989.

Kraut, Robert E., and Lynn A. Streeter: “Coordination in Software Development,” *Communications of the ACM*, vol. 38, no. 3, 1995, pp. 69-81.

Kreifelts, Thomas, Elke Hinrichs, Karl-Heinz Klein, Peter Seuffert, and Gerd Woetzel: “Experiences with the DOMINO Office Procedure System,” in *ECSCW ’91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, edited by L. Bannon, M. Robinson and K. Schmidt, Kluwer Academic Publishers, Amsterdam, 1991a, pp. 117-130.

Kreifelts, Thomas, Frank Victor, Gerd Woetzel, and Michael Woitass: “Supporting the design of office procedures in the DOMINO system,” in *Studies in Computer Supported Cooperative Work. Theory, Practice and Design,* edited by J. M. Bowers and S. D. Benford, North-Holland, Amsterdam etc., 1991b, pp. 131-144.

Kyng, Morten: “Designing for cooperation: Cooperating in design,” *Communications of the ACM*, vol. 34, no. 12, 1991, pp. 65-73.

Kyng, Morten: “Making Representations work,” *Communications of the ACM,* vol. 38, no. 9, 1995, pp. 46-55.

La Porte, Todd R. (ed.): *Organized Social Complexity: A Challenge to Politics and Policy,* Princeton University Press, Princeton, N. J., 1975a.

La Porte, Todd R.: “Organized Social Complexity: Explication of a Concept,” in *Organized Social Complexity: Challenge to Politics and Policy,* edited by T. R. La Porte, Princeton University Press, Princeton, N. J., 1975b, pp. 3-39.

Lai, Kum-Yew, and Thomas W. Malone: “Object Lens: A ‘Spreadsheet’ for Cooperative Work,” in *CSCW ’88. Proceedings of the Conference on Computer-Supported Cooperative Work, Portland, Oregon, September 26-28, 1988*, ACM, New York, N. Y., 1988, pp. 115-124.

Lauesen, Søren: “Object-Oriented Design in Practice,” in *Proceedings of OZCHI’95, Melbourne*, CHISIG, 1995.

Lee, J. Sibyl: “A qualitative decision management system,” in *Artificial Intelligence at MIT: Expanding Frontiers*, edited by P. Winston, MIT Press, Cambridge, MA, 1990.

Ljungberg, Fredrik: *Computer Supported Cooperative Work - en allmän teoretisk referensram [CSCW - a general theoretical frame of reference]*, Studies in the Use of Information Technology, Department of Informatics, Göteborg University, June, 1994. [5].

Machiavelli, Niccolò: *The Prince*, Translated by George Bull, Penguin Books, London, 1514. [Reprint published by Penguin Books, London, 1985].

Malone, Thomas W., and Kevin Crowston: “What is Coordination Theory and How Can It Help Design Cooperative Work Systems,” in *CSCW ’90. Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, Calif., October 7-10, 1990*, ACM press, New York, N.Y., 1990, pp. 357-370.

Malone, T. W., K. R. Grant, K. -Y. Lai, R. Rao, and D. Rosenblitt: “Semistructured messages are surprisingly useful for computer-supported coordination,” *TOIS*, vol. 5, no. 2, April 1987, pp. 115-131.

Malone, Thomas W., Kum-Yew Lai, and Christopher Fry: “Experiments with Oval: A Radically Tailorable Tool for Cooperative Work,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, edited by J. Turner and R. Kraut, ACM Press, New York, 1992, pp. 289-297.

Manohar, Nelson R., and Atul Prakash: “The Session Capture and Replay Paradigm for Asynchronous Collaboration,” in *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work - ECSCW’95, 10-14 September, 1995, Stockholm, Sweden*, edited by H. Marmolin, Y. Sundblad and K. Schmidt, Kluwer Academic Publishers, 1995, pp. 149-164.

Mashayekhi, Vahid, Janet M. Drake, Wai-Tek Tsai, and John Riedl: “Distributed, Collaborative Software Inspection,” *IEEE Software*, no. 9, 1993, pp. 66-75.

Mason, Richard O.: “MIS Experiments: A Pragmatic Perspective,” in *The Information Systems Research Challenge: Experimental Research Methods*, edited by I. Benbasat, vol. 2, Harvard Business School Research Colloquium, Harvard Business School, Boston Massachusetts, 1989, pp. 3-20.

Mathiassen, Lars, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage: *Objektorienteret analyse [Object-oriented Analysis]*, Marko Aps., Aalborg, 1993.

Mathiassen, Lars, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage: *Objektorienteret design [Object-oriented Design]*, Marko Aps., Aalborg, 1995.

Mathiassen, Lars, and Carsten Sørensen: “Managing CASE Introduction — Beyond Software Process Maturity,” *Proceedings of the 1994 ACM SIGCPR Conference, Old Town Alexandria, Virginia, USA*, edited by J. W. Ross, ACM, 1994, pp. 242–251.

McGrath, Joseph E.: *Groups: Interaction and Performance*, Prentice Hall, Englewood Cliffs, NJ, 1984.

Mintzberg, Henry: *The Structuring of Organizations. A Synthesis of the Research*, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

Mintzberg, Henry: *Structure in Fives: Designing Effective Organizations*, Prentice-Hall, New Jersey, 1983.

Mintzberg, H., D. Raisinghani, and D. Theoret: “The Structure of ‘Unstructured’ Decision Processes,” *Administrative Science Quarterly*, vol. 21, June 1976, pp. 246-275.

Monarchi, David E., and Gretchen I. Puhr: “A Research Typology for Object-Oriented Analysis and Design,” *Communications of the ACM*, vol. 33, no. 9, September 1992, pp. 35–47.

Murrel, Sharon: “Computer Communication System Design Affects Group Decision Making,” in *CHI’83 Human Factors in Computing Systems, Boston*, edited by A. Janda, ACM-SIGCHI, 1983, pp. 63-67.

Myers, Glenford J.: *The Art of Software Testing*, John Wiley and Sons, New York, 1979.

Nielsen, Jakob: *Usability Engineering*, Academic Press, 1993.

Nielsen, Jakob: “Heuristic Evaluation,” in *Usability Inspection Methods*, edited by J. Nielsen and R. L. Mack, Wiley & Sons, New York, etc., 1994, pp. 25-62.

Norman, Donald A.: “Cognitive Engineering,” in *User Centered System Design*, edited by D. A. Norman and S. W. Draper, Lawrence Erlbaum, New Jersey, 1986, pp. 31-61.

Norman, Donald A.: “Cognitive Artifacts,” in *Designing Interaction. Psychology at the Human-Computer Interface*, edited by J. M. Carroll, Cambridge University Press, Cambridge, 1991, pp. 17-38.

Olson, Judith S., Stuart K. Card, Thomas K. Landauer, Gary M. Olson, Thomas Malone, and John Leggett: “Computer-supported co-operative work: research issues for the 90s,” *Behaviour & Information Technology*, vol. 12, no. 2, 1993, pp. 115-129.

Olson, Judith S., Gary M. Olson, Lisbeth A. Mack, and Pierre Wellner: “Concurrent editing: The group’s interface,” in *INTERACT’90 - The Third Conference on Human-Computer Interaction,* Elsevier Science Publishers, 1990, pp. 835-840.

Orlikowski, Wanda J.: “Learning from NOTES: Organizational Issues in Groupware Implementation,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992,* edited by J. Turner and R. Kraut, ACM Press, New York, 1992, pp. 362-369.

Orlikowski, Wanda J.: “CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development,” *MIS Quaterly,* no. September 1993, 1993, pp. 309-340.

Parnas, David L.: “Software Aspects of Strategic Defence Systems,” *Communications of the ACM*, vol. 28, no. 12, December 1985, pp. 1326–1335.

Parnas, D. L., and P. C. Clements: “A Rational Design Process: How and Why to Fake it ,” *IEEE Trans.actions on Software Engineering,* vol. SE-12, no. 2, February 1986, pp. 251-257.

Patton, M.Q.: *Qualitative Evaluation Methods*, Sage Publications, USA, 1980.

Pedersen, Elin Rønby, Kim McCall, Thomas P. Moran, and Frank G. Halasz: “Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings,” in *INTERCHI’93 Conference on Human Factors in Computing Systems, Amsterdam,* edited by S. Ashlund *at al.*, ACM Press, 1993, pp. 391-398.

Perrow, Charles: *Normal Accidents. Living with High-Risk Technologies,* Basic Books, New York, 1984.

Petchenik, Nathan H.: “Practical Priorities in System Testing,” *IEEE Software,* vol. 2, no. 5, 1985, pp. 18-23.

Plowman, Lydia, Yvonne Rogers, and Magnus Ramage: “What Are Workplace Studies For?,” *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work - ECSCW’95, 10-14 September, 1995, Stockholm, Sweden,* edited by H. Marmolin, Y. Sundblad and K. Schmidt, Kluwer Academic Publishers, 1995, pp. 309-324.

Popitz, Heinrich, Hans Paul Bahrdt, Ernst A. Jüres, and Hanno Kesting: *Technik und Industriearbeit. Soziologische Untersuchungen in der Hüttenindustrie*, J. C. B. Mohr, Tübingen, 1957.

Posner, Ilona R., and Ronald M. Baecker: “How People Write Together,” in *Readings in Gropuware and Computer-Supported Cooperative Work,* edited by R. M. Baecker, Morgan Kaufmann, San Mateo, 1993, pp. 239-250.

Pycock, James, and Wes Sharrock: “The fault report form,” in *Social Mechanisms of Interaction*, edited by K. Schmidt, Computing Department, Lancaster University, Lancaster, England, 1994a.

Pycock, James, and Wes Sharrock: “Two comments on MOIs,” in *Social Mechanisms of Interaction*, edited by K. Schmidt, Computing Department, Lancaster University, Lancaster, England, 1994b.

RandomHouse (ed.): *Random House Unabridged Dictionary. The Random House Dictionary of the English Language*, (2), Random House, New York, etc., 1987.

Rasmussen, Jens: “The Role of Hierarchical Knowledge Representation in Decisionmaking and System Management,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 2, March/April 1985, pp. 234-243.

Rasmussen, Jens: *Information Processing and Human-Machine Interaction. An Approach to Cognitive Engineering*, North-Holland, New York, etc., 1986.

Rasmussen, Jens, Annelise Mark Pejtersen, and Len P. Goodstein: *Cognitive Systems Engineering*, Wiley series in System Engineering, ed. by A. P. Sage, John Wiley and Sons, New York, etc., 1994.

Rodden, Tom, and Gordon Blair: “CSCW and Distributed Systems: The Problem of Control,” in *ECSCW ’91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, edited by L. Bannon, M. Robinson and K. Schmidt, Kluwer Academic Publishers, Amsterdam, 1991, pp. 49-64.

Roth, Emilie M., and David D. Woods: “Cognitive Task Analysis: An Approach to Knowledge Acquisition for Intelligent System Design,” in *Topics in Expert System Design. Methodologies and Tools*, edited by G. Guida and C. Tasso, North-Holland, Amsterdam, 1989, pp. 233-264.

Rouncefield, Mark, John A. Hughes, Tom Rodden, and Stephen Viller: “Working with ‘Constant Interuption’: CSCW and the Small Office,” in *CSCW ’94. Proceedings of the Conference on Computer-Supported Cooperative Work, Chapel Hill, North Carolina, October 24-26, 1994*, edited by T. Malone, ACM Press, New York, N.Y., 1994, pp. 275-287.

Sachs, Patricia: “Transforming Work: Collaboration, Learning, and Design,” *Communications of the ACM*, vol. 38, no. 9, 1995, pp. 36-44.

Schmidt, Kjeld: “Functional Analysis Instrument,” in *Functional Analysis of Office Requirements. A Multiperspective Approach*, edited by G. Schäfer *at al.*, Wiley, Chichester, 1988, pp. 261-289.

Schmidt, Kjeld: *Analysis of Cooperative Work. A Conceptual Framework*, Risø National Laboratory, DK-4000 Roskilde, Denmark, June, 1990. [Risø-M-2890].

Schmidt, Kjeld: “Computer Support for Cooperative Work in Advanced Manufacturing,” *International Journal of Human Factors in Manufacturing*, vol. 1, no. 4, October 1991a, pp. 303-320.

Schmidt, Kjeld: “Cooperative Work. A Conceptual Framework,” in *Distributed Decision Making. Cognitive Models for Cooperative Work*, edited by J. Rasmussen, B. Brehmer and J. Leplat, John Wiley & Sons, Chichester etc., 1991b, pp. 75-109.

Schmidt, Kjeld: “Riding a Tiger, or Computer Supported Cooperative Work,” in *ECSCW ’91. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, edited by L. Bannon, M. Robinson and K. Schmidt, Kluwer Academic Publishers, Amsterdam, 1991c, pp. 1-16.

Schmidt, Kjeld: “Computational mechanisms of interaction— Requirements for a general notation,” in *A Notation for Computational Mechanisms of Interaction*, edited by C. Simone and K. Schmidt, University of Lancaster, Lancaster, England, 1994a, pp. 15-32.

Schmidt, Kjeld: “Mechanisms of interaction reconsidered,” in *Social Mechanisms of Interaction*, edited by K. Schmidt, Department of Computing, Lancaster University, England, 1994b, pp. 15-122.

Schmidt, Kjeld: *Modes and Mechanisms of Interaction in Cooperative Work*, Risø National Laboratory, P.O. Box 49, DK-4000 Roskilde, Denmark, 1994c. [Risø-R-666].

Schmidt, Kjeld: “The Organization of Cooperative Work — Beyond the ‘Leviathan’ Conception of Organization,” in *CSCW ’94. Proceedings of the Conference on Computer-Supported Cooperative Work, Chapel Hill, North Carolina, October 24-26, 1994*, edited by T. Malone, ACM Press, New York, N.Y., 1994d, pp. 101-112.

Schmidt, Kjeld, Hans Andersen, Peter H. Carstensen, and Carsten Sørensen: “Linkable Mechanisms of Interaction as a Representation of Organizational Context,” in *A Conceptual Framework for Describing Organizations*, edited by J. Bowers, Department of Computing, Lancaster University, England, 1994.

Schmidt, Kjeld, and Liam Bannon: “Taking CSCW Seriously: Supporting Articulation Work,” *CSCW*, vol. 1, no. 1-2, 1992, pp. 7-40.

Schmidt, Kjeld, and Peter Carstensen: *Arbejdsanalyse. Teori og praksis [Work Analysis. Theory and Practice]*, Risø National Laboratory, DK-4000 Roskilde, Denmark, June, 1990. [Risø-M-2889].

Schmidt, Kjeld, and Carla Simone: “Mechanisms of Interaction: An Approach to CSCW Systems Design,” in *COOP’95. Proceedings of the International Workshop on the Design of Cooperative Systems, January 25-27, Antibes-Juan-les-Pins, France*, INRIA, Sophia Antipolis, 1995, pp. 56-75.

Schmidt, Kjeld, Carla Simone, Peter Carstensen, Betty Hewitt, and Carsten Sørensen: “Computational Mechanisms of Interaction: Notations and Facilities,” in *Computational Mechanisms of Interaction for CSCW*, edited by C. Simone and K. Schmidt, University of Lancaster, Lancaster, England, 1993, pp. 109-164.

Schmidt, Kjeld, Carla Simone, Monica Divitini, Peter Carstensen, and Carsten Sørensen: *A ‘contract sociale’ for CSCW systems: Supporting interoperability of computational coordination mechanisms*, Working Papers in Cognitive Science and HCI, edited by F. Jensager, Centre for Cognitive Informatics, Roskilde University, 1995.

Scott, W. Richard: *Organizations. Rational, Natural, and Open Systems*, (Second edition; First edition 1981), Prentice Hall, Englewood Cliffs, New Jersey, 1987.

Shapiro, Dan: “The Limits of Ethnography: Combining Social Sciences for CSCW,” in *CSCW ’94. Proceedings of the Conference on Computer-Supported Cooperative Work, Chapel Hill, North Carolina, October 24-26, 1994*, edited by T. Malone, ACM Press, New York, N.Y., 1994, pp. 417-428.

Sharrock, Wes, and Bob Anderson: *The Ethnomethodologists*, Ellis Horwood Publishers, Chichester, 1986.

Shen, HongHai, and Presun Dawan: “Access Control for Collaborative Environments,” in *CSCW ’92. Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, October 31 to November 4, 1992*, edited by J. Turner and R. Kraut, ACM Press, New York, 1992, pp. 51-58.

Shneiderman, Ben: *Designing the User Interface. Strategies for Effective Human-Computer Interaction,* Addison Wesley, Reading, 1987.

Siemieniuch, Carys: “Design to product — A prototype of a system to enable design for manufacturability,” in *Design for Manufacturability — A Systems Approach to Concurrent Engineering and Ergonomics*, edited by M. Helander and M. Nagamachi, Taylor & Francis, London, 1992, pp. 35-54.

Simon, Herbert A.: “The Structure of Ill Structured Problems,” *Artificial Intelligence*, vol. 4, 1973, pp. 181-201.

Simon, Herbert A.: *The Sciences of the Artificial*, (Second edition; First edition 1969), The MIT Press, Cambridge, Mass., 1981.

Simon, Herbert A.: “Search and Reasoning in Problem Solving,” *Artificial Intelligence*, vol. 21, 1983, pp. 7-29.

Simone, Carla, Monica Divitini, and Alberto Pozzoli: “Towards an architecture based on a notation for mechanisms of interaction,” in *A Notation for Computational Mechanisms of Interaction*, edited by C. Simone and K. Schmidt, University of Lancaster, Lancaster, England, 1994, pp. 81-131.

Simone, C., M. A. Grasso, and A. Pozzoli: “Coordinator, AWMS, DOMINO, UTUCS, WooRKS, CHAOS, TaskManager, and Lotus Notes,” in *Computational Mechanisms of Interaction for CSCW,* edited by C. Simone and K. Schmidt, University of Lancaster, Lancaster, England, 1993, pp. 171-216.

Simone, Carla, and Kjeld Schmidt (ed.): *Computational Mechanisms of Interaction for CSCW*, COMIC, Department of Computing, Lancaster University, England, 1993.

Simone, Carla, and Kjeld Schmidt (ed.): *A Notation for Computational Mechanisms of Interaction*, COMIC, Department of Computing, Lancaster University, England, 1994.

Simonsen, Jesper: “Designing Systems in an Organizational Context,” Ph.D. Dissertation, Roskilde University, Roskilde, 1994.

Sirbu, Marvin, Sandor Schoichet, Jay Kunin, and Michael Hammer: *OAM: An Office Analysis Methodology,* Laboratory for Computer Science, MIT, Cambridge, 1981.

Sommerville, Ian, Tom Rodden, Pete Sawyer, and Richard Bentley: “Sociologists can be surprisingly useful in interactive systems design,” Manuscript, 1991.

Sørensen, Carsten: *Introducing CASE Tools into Software Organizations*, Topics in Cognitive Science and HCI, Ph.D. Dissertation, Centre for Cognitive Informatics, Risø National Laboratory/Roskilde University, 1993. [2].

Sørensen, Carsten: “The Augmented Bill of Materials,” in *Social Mechanisms of Interaction*, edited by K. Schmidt, Department of Computing, Lancaster University, England, 1994a, pp. 221-236.

Sørensen, Carsten: “The CEDAC Board,” in *Social Mechanisms of Interaction*, edited by K. Schmidt, Department of Computing, Lancaster University, England, 1994b, pp. 237-246.

Sørensen, Carsten, Peter Carstensen, and Henrik Borstrøm: “We Can’t Go On Meeting Like This! Artifacts Making it Easier to Work Together in Manufacturing,” *OZCHI’94 - Harmony Through Working Together, Melbourne*, edited by S. Howard and Y. Leung, CHISIG, 1994, pp. 181-186.

Star, Susan Leigh: “The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving,” in *Distributed Artificial Intelligence*, edited by L. Gasser and M. Huhns, vol. 2, Pitman, London, 1989, pp. 37-54.

Stefik, M., D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar: “WYSIWIS revised: Early experiences with multiuser interfaces,” *TOIS*, vol. 5, no. 2, April 1987, pp. 147-167.

Strauss, Anselm: “Work and the Division of Labor,” *The Sociological Quarterly*, vol. 26, no. 1, 1985, pp. 1-19.

Strauss, Anselm: “The Articulation of Project Work: An Organizational Process,” *The Sociological Quarterly*, vol. 29, no. 2, 1988, pp. 163-178.

Strauss, Anselm, Shizuko Fagerhaugh, Barbara Suczek, and Carolyn Wiener: *Social Organization of Medical Work*, University of Chicago Press, Chicago and London, 1985.

Suchman, Lucy: “Do Categories Have Politics? The language/action perspective reconsidered,” *Computer Supported Cooperative Work*, vol. 2, no. 3, 1994, pp. 177-190.

Suchman, Lucy: “Making Work Visible,” *Communications of the ACM*, vol. 38, no. 9, 1995, pp. 56-64.

Suchman, Lucy A.: *Plans and situated actions. The problem of human-machine communication*, Cambridge University Press, Cambridge, 1987.

Suchman, Lucy A., and Eleanor Wynn: “Procedures and Problems in the Office,” *Office, Technology, and People*, vol. 2, 1984, pp. 133-154.

Sutcliffe, Alistar: *Jackson System Development*, Prentice Hall, Hertfordshire, 1988.

Swenson, Keith D., Robin J. Maxwell, Toshikazu Matsumoto, Bahram Saghari, and Keith Irwin: “A business process environment supporting collaborative planning,” *Collaborative Computing*, vol. 1, 1994, pp. 15-34.

Thompson, Paul: *The Nature of Work. An introduction to debates on the labour process*, Macmillan, London, 1983.

Trevor, Jonathan, Tom Rodden, and Gordon Blair: “COLA: a Lightweight Platform for CSCW,” in *ECSCW ’93. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, 13-17 September 1993, Milan, Italy*, edited by G. De Michelis, C. Simone and K. Schmidt, Kluwer Academic Publishers, Dordrecht, 1993, pp. 15-30.

Tuikka, Tuomo, and Carsten Sørensen: “Architectural Issues in Design of Computational Coordination Mechanisms for Software Testing,” in *Demonstrator prototypes of Computational Mechanisms of Interaction*, edited by L. Navarro, University of Lancaster, Lancaster, 1995, pp. 219-254.

Weinberg, Gerald M.: *The Secrets of Consulting. A Guide to Giving & Getting Advice Successfully*, Dorset House, New York, 1985.

Winograd, Terry: “A language/action perspective on the design of cooperative work,” in *CSCW ’86. Proceedings. Conference on Computer-Supported Cooperative Work, Austin, Texas, December 3-5, 1986*, ACM, New York, N.Y., 1986, pp. 203-220.

Winograd, Terry: “Groupware and the Emergence of Business Technology,” *Proceedings of Groupware’92*, edited by D. Coleman, Morgan Kaufmann, 1992, pp. 69-72.

Winograd, Terry, and Fernando Flores: *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Corp., Norwood, New Jersey, 1986.

Wirfs-Brock, R.J., B. Wilkerson, and L. Wiener: *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

Woods, David D.: “Coping with complexity: the psychology of human behavior in complex systems,” in *Tasks, Errors and Mental Models. A Festschrift to celebrate the 60th birthday of Professor Jens Rasmussen*, edited by L. P. Goodstein, H. B. Andersen and S. E. Olsen, Taylor & Francis, London etc., 1988, pp. 128-148.

Woods, David D., and Emilie M. Roth: “Cognitive Systems Engineering,” in *Handbook of Human-Computer Interaction*, edited by M. Helander, Elsevier Science Publishers (North Holland), 1988.

Yin, Robert K.: *Case Study Research: Design and Methods*, Sage Publications, Beverly Hills, 1989.

Yourdon, Ed: “Software Quality Assurance in the 1990s,” in *Proceedings of the Sixth Annual Pacific Northwest Software Quality Conference, Portland, Oregon, USA*, 1988, pp. 3-33.

Yourdon, Edward: *Modern Structured Analysis*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

Zisman, Michael D.: “Representation, Specification and Automation of Office Procedures,” Ph.D. dissertation, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, PA, 1977.

# Appendix A: A list of papers produced

During the work, I have been involved in writing of a number of papers. These have been published as reports, conference papers, and journal papers. Several of these have been used as input for the dissertation. The list below includes all the papers I have been involved in writing while working on the dissertation. The list is organized according time of publication (the oldest first).

<u>1993:</u>

1 Peter H. Carstensen: “Graphical User Interfaces: Easy to use, hard to design,” *Proceedings of COPE' IT '93, Copenhagen*, Danish Data Association, 1993, pp. 107-122.

2 Peter Carstensen and Kjeld Schmidt: *The Procrustes Paradigm: A Critique of Computer Science Approaches to Work Analysis*, COMIC Working Paper, Risø National Laboratory, (version 2.0), May, 1993. [COMIC Report no. Risø 2-1].

3 Peter H. Carstensen and Kjeld Schmidt: “Work Analysis-Perspectives on and Requirements for a Methodology,” in *Human-Computer Interaction: Applications and Case Studies*, ed. by M. J. Smith and G. Salvendy, Elsevir, Amsterdam, 1993, pp. 575-580.

4 Hans Andersen, Peter Carstensen, Betty Hewitt, and Carsten Sørensen: “Aspects, Collage, Active Memory, OVAL,” in *Computational Mechanisms of Interaction for CSCW*, ed. by C. Simone and K. Schmidt, Esprit BRA 6225 COMIC, Lancaster University, 1993, pp. 217-238.

5 Kjeld Schmidt, Carla Simone, Peter Carstensen, Betty Hewitt, and Carsten Sørensen: “Computational Mechanisms of Interaction: Notations and Facilities,” in *Computational Mechanisms of Interaction for CSCW*, ed. by C. Simone and K. Schmidt, University of Lancaster, Lancaster, England, 1993, pp. 109-164.

<u>1994:</u>

6 Peter Carstensen: “The Bug Report Form,” in *Social Mechanisms of Interaction*, ed. by K. Schmidt, Esprit BRA 6225 COMIC, Lancaster, England, 1994, pp. 187-220.

7 Peter Carstensen and Carsten Sørensen: “The Foss Electric Cases,” in *Social Mechanisms of Interaction*, ed. by K. Schmidt, Esprit BRA 6225 COMIC, Lancaster, England, 1994, pp. 295-304.

8 Peter Carstensen and Carsten Sørensen: “Requirements for a Computational Mechanism of Interaction: An example,” in *A Notation for Computational Mechanisms of Interaction*, ed. by C. Simone and K. Schmidt, University of Lancaster, Lancaster, England, 1994, pp. 33-80.

9 Kjeld Schmidt, Hans Andersen, Peter H. Carstensen, and Carsten Sørensen: “Linkable Mechanisms of Interaction as a Representation of Organizational Context,” in *A Conceptual Framework for Describing Organizations*, ed. by J. Bowers, Esprit BRA 6225 COMIC, Lancaster, England, 1994.

## 1995:

10 Peter Carstensen: *The Complexity and Articulation of Work — In search for a useful approach for understanding and computer supporting articulation work,,* Working Papers in Cognitive Science and HCI, edited by F. Jensager, Centre for Cognitive Informatics, Roskilde University, 1995. [WPCS-95-8].

11 Peter Carstensen, Carsten Sørensen, and Henrik Borstrøm: “Two is Fine, Four is a Mess — Reducing Complexity of Articulation Work in Manufacturing,” *COOP'95. Proceedings of the International Workshop on the Design of Cooperative Systems, January 25-27, Antibes-Juan-les-Pins, France*, INRIA, Sophia Antipolis, 1995, pp. 314-333.

12 Peter Carstensen, Carsten Sørensen, and Tuomo Tuikka: “Let's talk about bugs? Towards Computer Support for the Articulation of Software Testing”, *Scandinavian Journal of Information Systems*, 7 (April), 1995, pp. 33-53.

13 Kjeld Schmidt, Carla Simone, Monica Divitini, Peter Carstensen, and Carsten Sørensen: *A 'contract sociale' for CSCW systems: Supporting interoperability of computational coordination mechanisms*, Working Papers in Cognitive Science and HCI, edited by F. Jensager, Centre for Cognitive Informatics, Roskilde University, 1995. [WPCS-95-7].

14 Peter H. Carstensen: “Modeling Coordination Work: Lessons learned from analyzing a cooperative work setting”, *HCI International '95 — 6th International Conference on Human-Computer Interaction, Pacifico Yokohama, Japan 9 - 14 July 1995., Yokohama, Japan*, edited by Y. Anzai, K. Ogawa, and H. Mori, Elsevier, 1995, pp. 327-332.

15 Peter H. Carstensen, Birgitte Krogh, and Carsten Sørensen: “Object-Oriented Modeling of Coordination Mechanisms”, *Proceedings of IRIS'18 'Design in Context', Gjern, Denmark*, edited by B. Dahlbom, *et al.*, University of Gothenburg, 1995, pp. 113-129.

16 Peter H. Carstensen and Thomas Albert: “BRaHS: A Computer Based Mechanism Supporting the Coordination of Bug-handling,” in *Demonstrator prototypes of Computational Mechanisms of Interaction*, ed. by L. Navarro, University of Lancaster, Lancaster, 1995, pp. 183-218.

## Forthcoming:

17 Peter H. Carstensen, Birgitte Krogh, and Carsten Sørensen: “Object-Oriented Modeling of Coordination Mechanisms” [Submitted for international conference]. Revised version of paper no. 15.

18 Peter H. Carstensen and Carsten Sørensen: “From the social to the systematic. An analysis of mechanisms supporting coordination work in design,” *Computer Supported Cooperative Work. An International Journal,* 1996, [Submitted for publication].