Published in: G. van der Veer, A. Henderson, S. Coles (eds.): DIS'97 Designing Interactive Systems: Processes, Practices, Methods, and Techniques, Conference Proceedings, Amsterdam, The Netherlands, August 1997, pp.205-213, 1997.

On the Inevitable Intertwining of Analysis and Design:
Developing Systems for Complex Cooperations

Anita Krabbel, Ingrid Wetzel,
Heinz Züllighoven
Computer Science Department,
Hamburg University,
Vogt-Kölln-Straße 30,
22527 Hamburg, Germany,
{krabbel, wetzel, zuelligh}
@informatik.uni-hamburg.de

ABSTRACT
Developing interactive software systems requires the well known tasks of analysis, design and construction. In the context of work settings with complex cooperations these tasks and their relationsship undergo drastic changes. Analysis and design have to be accomplished at different levels of complexity, the heterogeneity of users envolved needs to be handled and the presentation of anticipated changes incorporating the organizational context goes beyond proven (object-oriented) techniques like prototyping.

The article claims that complex cooperations require a close intertwining of analysis and design. It is accomplishable by application-driven documents usable in different stages of the development process. Based on a document-driven evolutionaly approach examples of such document types - like Cooperation Pictures and Purpose Tables - are given. They are discussed based on expericences from projects in different application domains.

Keywords
Cooperative Work, Evolutionary Design, Participation, Object-oriented Design

1 Introduction
Traditionally, application software development has focused on the workplace. This was reflected both by the software product and its development process. Early, mainframe-based systems automatized individual work routines, like money withdrawal or money transfer in a bank. Designing interactive systems of that type meant identifiying those work processes which occurred frequently at a workplace and which could be performed in a standardized way. This was called requirements analysis. What followed was a sequence of development steps, often called phases, which followed in some way the well-known and often criticized assumptions underlying the classical waterfall model (cf. eg. [Floyd 1987]). The subsequent software systems showed little application-oriented features or ways of handling. They were geared towards „data processing" and, with there entry fields and menu selections, can be characterized as „windows on data". If more than one user had to work with the system, this was dealt with through technical concepts like time-sharing or read-write locking, whithout any explicit model of cooperation.

With the advent of graphic workstation computers and PCs, there was a shift from implementing work routines to supporting everyday work at the individual work place. This change is reflected in the words by Alan Kay (1977) "do not automate the work you are engaged in, only the materials". If, however, application software has to provide appropriate and flexible components to support the various ways of working, this called for a different development process. Most software development methodologies and software technologies of the eighties were geared towards analysing, desiging and implementing interactive software in an application-oriented way. This meant first of all that the users played a more important role - participation or user involvement became the key word of the day. It became obvious that technological and application domain related issues had to be brought together and that the evaluation of all design results were critical to the overall sucess of application software.

As to the product itsself, this changed attitude in software development showed as well. The electronic desktop became the predominant design metaphor. A user would find a set of useful objects of work like documents and folders plus some more or less generic software tools like word processors and graphic editors to cope with the various tasks.

We, like others, quite successfully adopted a variation of object-orientedness for constructing application software as a product and combined it with an evolutionary

development strategy based on prototyping [Floyd 1984] [Bürkle, Gryczan, Züllighoven, 1995]. In our approach, we focus on a close relationship between the tasks and concepts of the application domain and the software model. This is captured in underlying design metaphors distinguishing those objects which are *worked upon*, i.e., materials, and those which are the *means of work*, i.e., tools. These software tools and materials are offered to the user together with „small automata“ at his or her electronic workplace. Building interactive systems for the individual workplace was done with a good level of quality, usability and acceptance in various industrial software projects.

Along with the success of these projects came further demands and problems. Soon, users and user management called for computer support not only for the individual work place but for those tasks which can only be handled through the cooperation of different persons. This demand was neither surprising nor new. Obviously, it is reflected for years by areas like distributed systems or CSCW. Still, we feel that it poses a new set of problems to application software development which will lead to different requirements towards analysis, design and implementation. Looking at current (object-oriented) development strategies, we can see that, although they claim to cover this area of software for cooperative work as well, they have little to offer with respect to the intrinsic problems of analysing and designing these systems. While mainstream object-oriented technology ignores these problems (cf. [Meyer 1990]), development strategies like [Jacobson 1992] or [Booch 1991] addressed them from the viewpoint of distributed technical systems; [Reenskaug96] focuses on modeling the different aspects or roles of entities within cooperative work.

In this paper we will explain why cooperative work is harder to understand, analyse and support than individual work at a workplace. We will show, why and how application-oriented concepts combined within an evolutionary strategy can help to overcome these difficulties. The key concept here is an intimate intertwining of analysis and design. A set of documents will be presented which smoothly support these development activites. As this is the focus of our paper, we will only hint at the design characteristics of the interactive systems themselves. This was [Wulf, Gryczan Züllighoven, 1996] [Lilienthal, Züllighoven, 1996] and will be subject of other papers.

Section 2 of this paper addresses the specific problems with analysis and design in the context of complex cooperations. Section 3 then presents our document-driven approach by giving examples of useful document types and evaluating their exploitation in our projects. Section 4 summarizes the results.

2 Complex Cooperations and Problems with Analysis and Design

In the following we first illustrate the characteristics of complex cooperations by presenting small examples from software projects we have done (and are still doing) in two different domains, namely banking and hospitals. On this basis we will outline, why cooperative tasks show a new level of complexity to the software developer. Then we state a set of requirements for the development process in order to overcome these difficulties.

The background of the mentioned projects is briefly as follows: In the hospital domain the aim of one of our projects was to support a hospital in introducing an integrated Hospital Information System. We were involved in two project "stages": the first comprising the analysis of the situation, a statement of the different requirements, a rationale for assessing different systems in the market and the selection of one appropriate candidate. The second, still ongoing project stage is devoted to the configuration and use of this system in the light of changing demands.

In the banking sector, we have been involved in different projects (eg. [Bäumer, Knoll, Gryczan, Züllighoven, 1996]), but we will concentrate on those three projects, which focus(ed) on the credit business. In all three projects the aim was to develop an object-oriented interactive system which supports the different tasks of granting a credit; the customers, however, being either private persons, companies or municipalities.

2.1 Characteristics of Complex Cooperative Tasks

What are the main characteristics of complex cooperative tasks that we find in both domains?

- A multitude of people from different occupational groups cooperate within a single task.
- Almost every tasks calls for situated action.
- Cooperative tasks require coordination.

Obviously, this is no new discovery. What we will show are the implications for the software development process and the techniques and means used.

Starting with the hospital domain, a good example is the „admission of a patient“ in a hospital which arises a lot of times each day. It involves adminstrational and organizational tasks as well as medical and caring work with the patient. A brief description is given in box 1.

The admission of a patient to the hospital is usually initiated by a call of a general practitioner or the central hospital bed registrar. The calls are received either by the admission office or the senior physician on duty. Each morning the admission office provides a list with all available beds. The senior physician makes himself knowledgeable about it. The allocation of beds is made by the senior physician and the admission office in close cooperation.

When a patient arrives at the hospital he usually brings with him an admission sheet from the general practitioner on which the diagnosis is stated. The patient signs an admission contract and is being questioned regarding his personal data. He receives his admission contract and stickers on which are printed his personal data and walks to the nursing unit. The admission of patients usually occurs in the morning.

On the unit he is questioned further by a nurse. She fills in a sheet about his physical condition and starts the patient's record. She enters his name on several tables for overviews of bed usage, telephone numbers, diagnosis and treatments. She passes on a menu card of the patient to the kitchen.

The responsible resident physician examines the patient and fills in a physician's order sheet. The nurse copies all the doctor's orders into the patient's sheet. For the (routine) examinations ordered in the course of admission she fills in the order entry forms, hands them to the physician for his signature and delivers them to the corresponding department e.g. X-ray department. She labels the blood tubes for the blood tests and also fills in order sheets. If the patient was previously admitted to the hospital she calls the archive and orders the old patient record.

When the arrangement with the functional departments have been accomplished the patient goes or is brought to the corresponding department e.g. X-ray department. The radiology technician orders the old X-ray bag from the archive if the patient was admitted before. The result of the examination is dictated on a recorder by the radiologist and typed directly on the order entry sheet by the secretary of the chief physician. Sometimes in the afternoon the nurse picks up the sheet from the X-ray department, hands it to the resident physician to read and sign, and after that files it into the patient record.

In the afternoon the admission officer assembles a physician portfolio and a patient portfolio. The physician portfolio contains the discharge form and further patient stickers and is sorted into the patient record. The a patient portfolio remains in the admission office and contains documents for billing purposes. Also data is sent to the hospital controlling department.

Box 1: Admission of a patient to the hospital

As apparent from the description the first characteristic of complex cooperative tasks in an application domain is that they are accomplished by a *multitude of people from different occupational groups often with highly varied fields of activity*. In the hospital domain the admission of a patient requires the cooperation between people even in different organizational units like the admission office, nursing unit, functional departments, laboratory, archive, kitchen, gate, secretary of chief physician and the administration. They all have to work together because their different skills are necessary to get the task done.

The introductory example from the banking domain is „the granting of credits to private customers“ which forms a major business in e.g a savings bank (This example is taken from [Wulf, Gryczan, Züllighoven, 1996]). Credits are granted by Customer Advisors. How the actual process of granting a credit is done, is mostly up to the customer advisor and the specific case at hand. There may be general rules within the branch of a bank and there are some legal restrictions ("Four-Eyes-Principle").

Box 2 gives an overview of the activities in case of typical credit granting task.

The work procedure for granting a credit distinguishes between customers that are known to the bank and new customers. Based on the information available for a customer, the advisor will usually prepare and conduct the consulting talk with the customer which will lead to a decision about the credit. For known customers, information is already available. It can be gathered from various files kept in the bank. Based on this information, a customer advisor will usually prepare and conduct the consulting talk with the customer which will lead to a decision about the credit. The allocation of customers to advisors is regulated through bank-internal assignment rules; for example based on an alphabetical scheme for customer names.

The form for granting a credit is filled by the advisor partly while talking to the customer and partly afterwards. In addition, a protocol of the talk is made. Ordering the payment of the credit can either be done by the customer advisor or by another bank clerk. Usually, a specific account has to created.

Before the money can be transferred, the granting of the credit has to be approved by a second customer advisor. For this approval, various documents may be needed. Sometimes, this is only the tender offered by the bank, in other cases all available data about the customer will be checked. If the second customer advisor approves the credit, the documents are transferred back to the customer advisor in charge.

Box 2 „Granting a credit to private customers“.

As obvious from the example a second important characteristic of complex cooperative tasks is that although they are quite frequent and the individual work steps are well-known, dealing with a concrete task calls for *situated action* (cf. [Suchman 1995]). In our example, most of the activities can be dealt with when appropriate and not in a fixed temporal sequence. Some of them are even optional. The example, by the way, covers the common case, specific situations may call for different or additional activities.

As a third example we present in box 3 the „registration for an X-Ray examination“ again from the hospital domain. It involves a very close cooperation of a few responsible person.

The physician writes the order on the physicians order form and puts the order entry sheet in the nurse's mail basket. A nurse responsible for looking at the basket fills the relevant data on the order entry sheet in order to relieve the physician of the burdon to prepare it himself. Additionally, she enters the test with pencil on the patient's flowsheet. Then she puts the order entry sheet in the physician's mail basket. Seing the order entry sheet in his basket, the physician enters the relevant clinical information, signs it and puts it in the nurse's mail basket again. The nurse carries the order entry sheet to the X-ray department. Now the X-ray departement can schedule the examination under consideration of their work load and the performing physician can check the order.The chosen date is conveyed by phone to the unit.The nurse enters the date of the test in the units calendar to inform the nursing stuff when to take the patient to the X-ray Department.

Box 3: The registration for an X-Ray examination

A further characteristic of complex cooperative tasks is that they always means coordination as well. This means, that a lot of activities within cooperative tasks are performed solely to *coordinate* the other activites. In the CSCW literature this is also called articulation work [Schmidt, Bannon 1992].

2.2 Consequences for Analysis and Design

We have so far outlined situations showing some characteristics of complex cooperative tasks. But what does this mean for application software development? For the developers it means a *new level of complexity* and a problem of *understanding the situation* at hand. Traditionally, the situation at hand was dealt with during the so-called requirements elicitation phase. Here, activities were more focused towards requirements for the *new system* than understanding the *current work sitution*. Looking at the domains of our examples a crucial issue becomes obvious:

Even if the future system is supposed to change current work activities and procedures, the developers have to *identify* and *understand* the tasks which constitute the core business of the application domain. It is important to understand which of the *current tasks* are important in the future and therefore have to be supported by the new system and which tasks will be newly introduced. In addition, it is crucial to understand, where and how the future system will change the way in which tasks are performed. The prerequisite for this is that the current activities have to be understood, which means that developers have to understand *who does what and why*.

But how can developers understand the tasks and activites of everyday work. In most cases, developers are dealing with application areas other than their own, ie. software development. So they have to understand an unfamiliar domain through some kind of *explicit analytical process*. We, like others, have shown, that an object-oriented approach focusing on the objects and concepts of an application domain is appropriate for identifiying and understanding the tasks that are dealt with at the individual work place (cf. [Bürkle, Gryczan, Züllighoven, 1995], [Wirfs-Brock, Wilkerson, Wiener, 1990]). But a so-called behavioral approach of analysing and modeling objects and concepts falls short, if complex cooperative tasks come into play. Here, the objects and concepts necessary for coping with a tasks have to be placed into the context of different *cooperative work processes*. In addition, those items and activities have to be identified which help in *coordinating* the cooperative tasks.

A summary of the main difficulties in *analysing* complex cooperative tasks should give rise to requirements for a different approach to software development in this context:

- The overall situation is highly complex. It is extremely difficult, for example, to get an overview of the various types of banking credits and the different procedures for granting a credit. In a hospital, it is next to impossible to gain an "analytical" understanding of all the potential tasks related to a single patient case. While we can question the possibility of a software developer ever completely understanding the complexity of one of these application domain, it is certainly impossible to have an initial requirements specification stage where all details of the situation at hand can be analysed and modeled adequately. Despite this complexitity, software developers need a process leading from initial to a growing understanding of the sitution on different levels of abstraction.
- Software developers are *no application domain experts*. Even after developing systems in one domain for years they will lack a profound understanding of the many application concepts and the actual work procedures. This holds even for developers who have formerly worked in the application domain. While this is a general problem of application software development, cooperative work makes the situation more critical. Here, an understanding of various tasks and subtasks from the viewpoint of sometimes different professions is required. In a banking project which dealt with customer consulting, for example, some of the developers had started their professional career as banking representatives, even as customer consultants. They saw themselves as domain experts. Despite this background, they simply overlooked to analyse and model some important sub-tasks which were dealt with by backoffice clerks. They had lost the insight into current work processes and a new division of tasks. As a consequence, a more direct contact to the situation at the work places is of crucial importance. A mere analysis of information and data flow among different „data processing units“ based on form sheets or legacy software will be of little help.
- *No single person from the application domain* will understand the overall situation to a sufficient degree in order to communicate it to the developers. This is complementary to the problem just stated. In a workshop with various representatives of a hospital, for example, some doctors remarked during the discussion of cooperation pictures (cf. next section) that they never had an idea about the usefulness of certain form sheets they had to fill and send to the admission office. This example shows the need for a fresh look at participation or user involvement (cf. [Lilienthal, Züllighoven, 1996] [Krabbel, Wetzel, Ratuski, 1996]). It is not only a matter of democracy at the work place or other political or ethical considerations but also a mere factual neccessity to integrate the different parties from the application domain into the development process.

So far we have outlined the intrinsic difficulties of analysing and understanding complex cooperative tasks. What are the difficulties of *designing* the future system?

- It is a general but fundamental problem that, at the start of a software project, no one can completely *envision the future system*. As just said, the developers lack a detailed understanding of the application domain. Of course, this was true with application software for individual work places. But here, a desktop approach could overcome some of the problems, as a fairly general set of tools and materials put at the disposition of the user, could be utilized, when and how appropriate. With cooperative work, however, developers have only vague ideas about an appropriate way of organizing future tasks and business processes. Certainly, the superficial and technical concepts of current business process reengineering methods won't help. While it is valuable in the development of work station software to gain inspiration by looking at good examples of available systems, this is almost no option in the area of complex cooperative work. Of course, there is a wealth of research and development in the CSCW area. But this is of interest to acadamia mainly and offers little remedy for the problem at hand, as these systems are mostly not available of demanding as to resources. And software on the market in this area is uninspiring at best.
- Application domain experts may have some ideas about the relevant tasks and related acitivties which could in principle be supported by software. But they usually lack

an understanding of the technical means and potential of computers. Frequently, we were confronted with "visions" of the future system which drew from the year-long use of legacy systems on mainframes with menus and forms or, at the other extreme, from a mix of sci fi movies and game computers. As a consequence, both the development process and the techniques and means to design the future product have to take into account, that the vision of the future system will evolve during a software project. Emerging new ideas will have to be tried out, some will prove inadequate, leading to changed concepts, etc. It is crucial to find a way from the current situation to a common vision which can be shared by all the different parties envolved.

- During the development process, there will be different and conflicting views and interests regarding the future system. In a system supporting cooperation, it is crucial, however, that the different sub-tasks and work processes link smoothly. Given all these different opinions, there cannot be one best solution decreed from "above". Instead, a satisfying compromise has to be found among all parties concerned. Here, the design process becomes a *negotiation process*. Seen from a system developer's view, these negotiations should be based on adequate and fairly realistic models, which are both understandable objects of discussion for domain experts and useful development documents. The development process then has to take into account that revisions of these models due to subsequent insight or changed circumstances will not jeopardize the whole design.

3 The Document-Driven Development Process

Having highlighted the specific problems of developing applications systems to support cooperative work settings, we now describe our approach to overcoming these problems. We will introduce a set of documents, representing the different models of the application domain and the software system, and a strategy of how to conduct and organize the development process itself.

3.1 Reducing Complexity

As a major problem in analysing complex cooperations we have identified their overwhelming complexity in the face of continuously changing demands and requirements. So, first of all we need a means to make this complexity manageable. The key to solution is well-known - abstraction. Accordingly, our approach is based on a set of documents, which allow for different level of abstractions for each of the central analysis and design issues.

3.1.1. From the Individual Workplace to the Overall Situation to the Coordination

A major development task is to analyse, understand and model the work as seen from the various workplaces involved. Here we use well-proven techniques like qualitative interviews together with *scenarios*, a *glossary* and *system visions* (cf. [Bürkle, Gryczan, Züllighoven, 1995]). Scenarios, for us, identify the work tasks at a certain workplace and describe the present way of accomplishing these tasks with different means and objects of work. Glossary entries define the terms and concepts used in the application domain. Focusing on design, systems visions are "future scenarios" that anticipate work situations utilizing the future software system.

Scenarios and system visions capture the perspective of the individual workplaces with their respective views of cooperative work. These individual pespectives frequently lack insight into the nature of cooperative processes. As an example, the nurses of a ward had no idea about the purpose of certain documents they had to prepare and to deliver to the administration. But it is important to get the overall view, ie. to understand which are the main so called joint tasks [Krabbel, Ratuski, Wetzel, 1996] of an organization being accomplished in cooperation across department boarders.

For this reason *Cooperation Pictures* are used (cf. [Krabbel, Ratuski, Wetzel, 1996] and Figure 1). They provide a graphical visualization of concrete cooperations in order to accomplish a joint task. Accordingly, they represent the involved (work) places or roles and the cooperation between them. Places and roles are visualized by symbols with names. In the hospital project, we distinguished between places outside and inside the hospital and certain roles like a chief physician which could not be related to a stable location. *Cooperations* are represented by annotated arrows. In the hospital context we had the delivery of documents by the hospital staff, phone calls, data exchange via computer and the patient making his/her way to the different units of the hospital. The arrows were annotated by pictograms indicating these different kinds of cooperation. It is possible to refine or coarsen these cooperation picture. A more detailed picture eg. could focus on the process of an X-ray examination, were numbers are added to the arrows in order to indicate work sequences for a typical case.

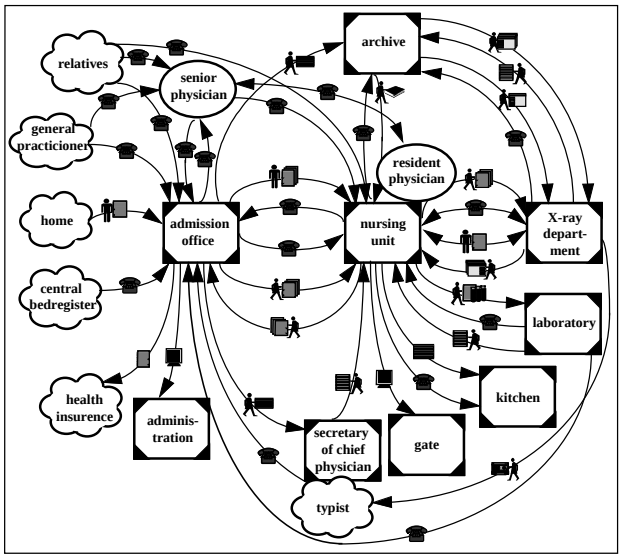

Figure 1: Cooperation Picture „Admission of a patient“ in the hospital domain

As a characteristic of modeling cooperative work, it is necessary to identify and represent the specific coordination

activities. This is not adequately represented in the documents described so far.

As a basis, we have to understand the purpose of the exchanged documents or of the explicit coordination processes. Only then, the concrete actions of a task can be evaluated and consequences for the design of the future system can be outlined. Here, we apply Purpose Tables (cf. [Heeg 1995]) as a representation technique. In purpose tables, we describe the cooperative task devided into Who - does What - with What or Whom - for what Purpose. The focus of this table is to identify the different purposes or implications of each individual task.

Figure 2 shows the registration of a patient for an X-ray examination. A first look might only show the registration of a patient for an X-ray examination. Only by looking at the purposes, it becomes clear that a lot of cooperation and coordination is involved: the nurse is informed about the examination and thus about the treatment of „her" patient. In addition, the entry in the patient's flow sheet makes the registration visible to other physicians and nurses with subsequent medical or nursing consequences.

Single Activities of an Order Entry	Purpose/Implications
Physician writes the order on the physicians order form.	It is documented who ordered the test at what (forensic, quality assurance). To kick on the implementation of the test.
Physician puts the order entry sheet in the nurse's mail basket.	Nurse is alerted that she has to act. She knows what is planned with her patient.
Nurse enters patient's name, other relevant data and the type of test on the order entry sheet.	Nurse prepares the order entry sheet in order to relieve the physician of such burdens.
Nurse enters the test with pencil on the patient's flowsheet.	It is documented for every member of the care team and physicians when the examination was ordered and to which further examinations he is scheduled.
Nurse puts the order entry sheet in the physician's mail basket.	Physician knows that he has to validate the order.
Physician sees the order entry sheet in his basket, enters the relevant clinical information, signs it and puts it in the nurse's mail basket.	The physician that carries out the test knows what to do and that the ordering physician is responsible for the test.
Nurse carries the order entry sheet to the X-ray department.	The X-ray departement can schedule the test and the performing physician can check the order.
Radiology technician chooses a date for the test and conveys it by phone to the unit.	The tests are coordinated within the X-ray department. The nurses know when to take the patient to the X-ray Department.
Nurse enters the date of the test in the units calendar.	Whole nursing stuff knows about the date.

Figure 2: Purpose Table of the registration for an X-Ray examination

3.1.2 Tackling the Complexity of the Development Process

The" abstraction" paradox: In order to get an overview of the overall situation a detailed understanding of the different tasks is mandatory, while one cannot understand the different tasks without an overview.

The obvious problem of the development process for cooperative work is how to solve the *"abstraction" paradox:* an overview of the overall situation can only be achieved by understanding the individual tasks and ongoing activities. We have said that developers don't have this kind of knowledge and that domain experts usually can overlook only those parts which are related to their own work. But it is impossible for the developer to aquire this understanding by an initial detailed analysis of the situation at hand.

The only possible solution of this paradox is to live with it. We try to iterate between an analysis of individual tasks and a synthesis of the different cooperations as part of the overall situation. Well knowing that this is always vague and incomplete at the beginning and can converge only by frequent feedback loops and involvement of domain experts.

Accordingly, we use the different representation techniques for the detailed description of tasks at individual workplaces, general representation of joint tasks and again detailed representation of coordination work. Evolutionary development principles tell us, however, that there is no project state devoted to for example to prepare "all" scenarios or cooperation pictures.

Iteration between the detailed analysis on the task level and an overall synthesis on the level of cooperations seems to be the key to solving the abstraction paradox.

The „design" paradox: One cannot design an application system without analysing the current situation in the application domain, but a vision of the future system is a prerequisite for analysis.

Both developers and users have to understand the current tasks and work processes in order to get a vision of the future system support. But they will not know what situation to look at and to which detail, if the future tasks and the system support is unclear. Here, the intentional intertwining of analysis and design can be a remedy. A preliminary understanding of the potential future situation has to be expressed in models which can be revised or discarded quickly and easily. This is were domain-related documents and prototypes come together (cf. [Bürkle, Gryczan, Züllighoven, 1995]).

The „product" paradox: Building software support for cooperative work, one has to evalute an operative system, in order to really assess its feasibilty and suitability, but it takes an unforeseeable effort to build this system.

The answer to this problem is again an evolutionary approach, but on a different level. It means to dissect a

seemingly monolothic system, like The Hospital Information System, into a minimal *application kernel* planned in extensions. The kernel has to be operative and has to satisfy urgent needs of the organization. Additionally, it should support tasks of key units or departments which show a high cooperation profile. And it needs to supply basic cooperation and coordination means. Therefore, identifying the kernel and determining extensions of an application system for cooperative work is by no means a trivial task. While we have stressed the priority of domain-related features, technical kernel facilities like openness, extensibility and appropriate database interfaces have to be considered as well.

As an example, Figure 3 shows a sketch of the application kernel system with some extensions of the hospital project.

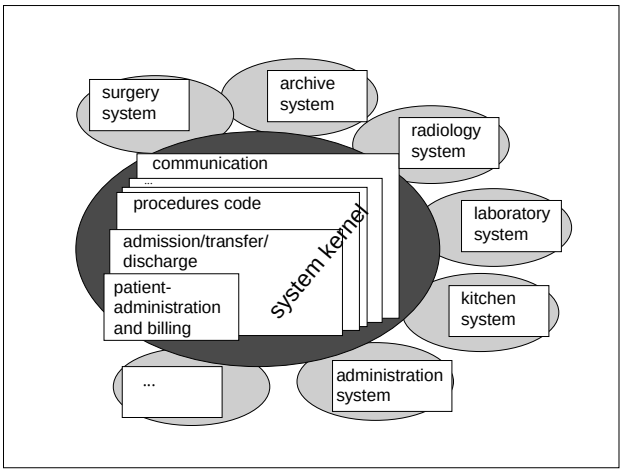

Figure 3: The application kernel (with extensions) and subsystems

3.2 Supporting Group Processes

The second group of problems identified in Section 2.3 had to do with group processes. We have to make sure that the different groups can actively participate in the design process. They have to understand key documents about the current situation and the future system. These document should also support the negotiation processes about work situations, present and future and the different interests and viewpoints involved. So all domain-related representation techniques and documents need to be understandable for very different user groups, easy to apply and stimulative for the communication and negotiation process.

3.2.1 Documents as Materials of Work and Communication Base

Szenario, System Visions and *Glossaries* are written in the professional language of the application domain. So they are obviously understandability for at least the domain experts. Feed back cycles from users will be easy. Moreover, the Glossary makes terms and notions explicit, thereby supporting the formation up of a common project language. The only obstacle is the lacking domain knowledge of the developers. Therefore, they have to write these documents, based on interviews, while the domain experts act as reviewers. In these learning cycles developers quickly catch up the respective professional language.

Cooperation Pictures are used in joint workshops with representatives of all user groups. They serve as a high-level represention of a situation, when prepared in advance and explained by developers or they can be used as a means for actively aquiring knowledge about a joint task by „interactively" drawing and pasting their elements on a wall paper. Due to the use of everyday pictograms and their lacking formality, their meaning and usage can be grasped by almost everyone in minutes.

Cooperation Pictures are also a valuable means for questioning *the adaquacy of the current work processes.* They provoke questions like What are the reasons for a particular work process and What are *the essential cooperation links* to other departments. In the hospital project it was surprising for all participants of an analysis workshop that during a regular admission of one patient in the morning up to 17 phone calls and a series of errands had to be made. Immediately a discussion arose about which of the phone calls or errands were avoidable in general and which could be eliminated by using the future system.

PurposeTables provide a structured document written in the language of the application domain. While users will rarely prepare Purpose Tables on their own, they can use them after a short introduction to analyse and discuss details of work processes and the purpose of individual activities.

Purpose Tables are employed selectively to identify the purpose and dynamics of key coordination patterns. In the hospital project they were eg. used in the decision of whether the signature from the physician would still be required for every X-ray registration. It turned out that this was seen necessary for reasons of quality assurance by the radiologists.

3.2.2 Making the Development Process Manageable

Beyond using domain-related documents it is important to make the development process itsself understandable and manageable.

Our approach is based on the concept of an application kernel with domain-related extensions. Using these, domain experts can enter a discussion about which extensions will support which task; what are the interrelations between the cooperative tasks and extensions; what are the priorities for realizing these extensions. Figure 4 shows an application kernel with extensions indicating the interdependencies between the tasks supported by these extensions.

Both in banking and in the hospital projects, users and user management pretty soon grasped the idea of the application kernel with extensions and used numbers to name the various extensions, which they could relate to familiar cooperative tasks. Even in discussions outside the software project they kept talking about „application kernel" or „extension 2".

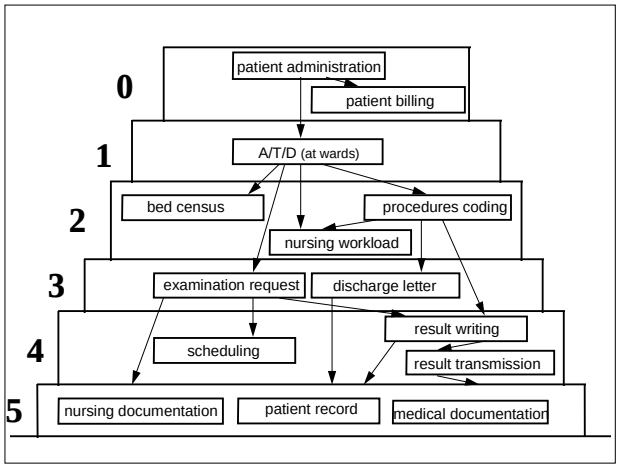

Figure 4: The application kernel with its extensions.

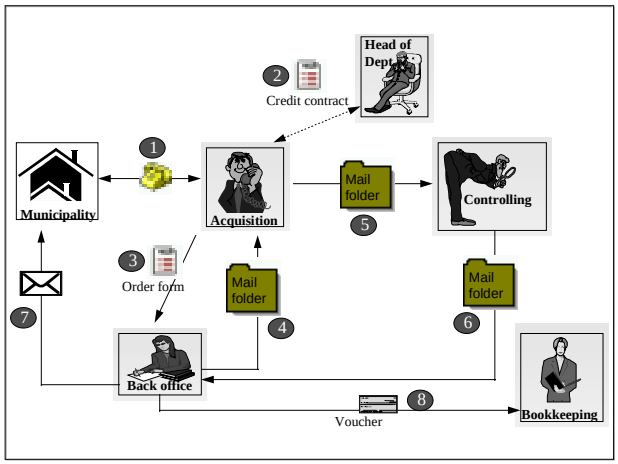

Figure 5: Cooperation Picture for „granting a credit“

3.3 Anticipation of the Future System

While prototyping is very useful in designing interactive systems for individual workplaces, one prototype will not overcome the problem of assessing the interplay of the different parties involved in cooperative work. We at least need distributed prototypes; and frequently, we need different prototypes for the different roles and subtasks involved. In addition, the coordinating activities should not be overlooked, when transfering objects or means of work into application software components. Last, but not least, we have to cope with the high degree of situatedness in many cooperative tasks.

3.3.1 Anticipating Future Cooperation and Devision of Work

While system visions can provide a good overview of the future tasks and the general features of the application system, they have little „analytic“ power, ie. the cannot be used easily to access the future work situation. Here, Cooperation Pictures can help. Figure 5 gives an example from a banking project. This picture initially showed the „status quo“, ie. the current distribution of work and process sequence for a „default“ case of granting a credit. In a workshop the involved bankers discussed with the development team alternatives for both work dsitribution and process, by changing the picture as a basis for What-if questions. Here, an analysis of the Cooperation Pictures indicates nodes of high communication and cooperation „traffic“. In Figure 1 one can for example easily see that the nursing unit and the admission office are focal points of this type. So, the same presentation techniques can be used for analysis and design purposes. Additionally, Cooperation Pictures can be used to indicate different extensions of the system kernel,e.g. by changing arrows being effected by a (next) extension. Eg. errands will get substituted by electronic document exchange.

For developers following an object-oriented approach the glossary entries provide the basis for designing the domain-related class hierarchy. Frameworks and design patterns then are used to achieve a proper fit between domain-related and technical design [Riehle Züllighoven 1995], [Bäumer, Knoll, Gryczan, Züllighoven, 1996].

In the context of cooperative work we have to take care, however, not to overlook the specifics of cooperation and coordination. When analysing existing means and objects of work, it is crucial to understand the role of these things in cooperation contexts. If a decision is made not to transfer a certain entity into the software system, the developers have to provide substitutes, if these entities help to coordinate cooperative activities. Problems of this type can be tackled by using Purpose Tables. They indicate the implications of modeling or omitting certain things or their characteristics. In one banking project, we have discussed the consequences of not modeleing the order sheet as an object of its own right but going straight from an aquistion tool to the credit contract. We found out that the order form was important for indicating the state of the credit granting process as well, at least for the controller.

3.3.2 Designing Support for Future Cooperation

Understanding the implications of cooperative work and activities is the precondition for designing the future system; but it does not automatically lead to an appropriate design of an interactive system. With respect to the individual workplace, we have shown [Bäumer, Knoll, Gryczan, Züllighoven, 1996], [Bürkle, Gryczan, Züllighoven, 1995]. that the Tools & Materials Approach can direct the developers towards this goal. In the context of cooperative work, however, more considerations are needed. First, we have to answer the question, how can we support future cooperation, if this cooperation is going to change while we are building the system or because we introduce a software system with a certain degree of unforeseeable effects on cooperation.

Here, the answer is similar to the one, Tools & Materials provide for the individual workplace: If you cannot foresee the details of future cooperation, don’t implement fixed cooperation and coordination processes into the system. In order to avoid the workflow pitfall, we have proposed the

concept of process patterns (cf. [Gryczan, Wulf, Züllighoven, 1996]). In short, they reify the cooperation and coordination process as documents for the users and make these patterns subject to situative changes and amendments.

Finally, it is again the process of going from the application kernel to the stepwise addition of extensions which puts both domain experts and developers in a position, where the can envision the future system in „stepwise refinements“, going from vague initial ideas to more a growing understanding of what an interactive support can be, as the first building bocks of the system are put into use.

4 Summary

In this paper we addressed specific software development problems related to complex cooperations. Using examples from the banking and hospital domains we started by *characterizing* these applications. Tasks with complex cooperations involve a multitude of different user groups, need a flexible situative execution and include various coordinating activities.

Based on these characteristics we outlined *specific problems of analysis and design.* First, analysis of the current work practise is a crucial activity distinct from requirements analysis. Analysing the current work practise, however, is not easy because of the intrinsic complexity of cooperative settings. Furthermore, neither the developers nor the users are able to understand all the relationships and dependencies in full. In design neither developers nor users have a clear picture of the desirable system support. This is due to the lack of successful existing examples as well as to the difficulty to anticipate and communicate about the future system. Additionally, the design of the future system has to be a process of continuous negotiations between the different and often competing groups involved.

In order to overcome these difficulties we presented a document-driven development process. It is based on domain-related document types. Characteristics of our approach are:

For reducing *complexity* we use domain-related document types on different abstraction levels leading from the individual workplace to the overall situation to coordination activities. Preparing these documents we need rapid iterations between analysis on these different abstraction levels.

For supporting the *negotiation problem* we presented documents that can be applied in workshops and group discussions. For making the development process manageable by all parties involved the future system needs a kernel with well-understood extensions according to domain-related criteria.

For solving the *anticipation problem* we use the same documents to discuss potential changes of cooperative work on the basis of a potential system support.

Using the same document types for the manifolded aspects during analysis and design allows a close intertwining between these different activities. This intertwining alone guarantees stepwise understanding and design of well defined system components that match user requirements by making them more explicit.

REFERENCES

Bäumer, D., Knoll, R., Gryczan, G., Züllighoven, H. (1996). Large Scale Object-Oriented Software-Development in a Banking Environment - An Experience Report. In: Pierre Cointe (Ed.): ECOOP'96 - Object-Oriented Programming, 10th European Conference, Proceedings, Springer Verlag, Linz, Austria, July 1996,. pp. 73 - 90.

Booch, G.: Object Oriented Design with Applications, Redwood City, CA: Benjamin/Cummings 1991.

Bürkle, U., Gryczan, G., Züllighoven, H. (1995). Object-Oriented System Development in a Banking Project: Methodology, Experience, and Conclusions. In: Human-Computer Interaction, Special Issue: Empirical Studies of Object-Oriented Design , Volume 10, Numbers 2 & 3, 1995, S. 293-336. Lawrence Erlbaum Associates Publishers Hillsdale, New Jersey, England.

Floyd, C.: A Systematic Look at Prototyping. In: Budde, R.; Kuhlenkamp, K.; Mathiassen, L.; Züllighoven, H. (Hrsg.): Approaches to Prototyping, S. 1-18. Berlin: Springer 1984

Floyd, C. (1987). Outline of a paradigm change in software engineering. In G. Bjerknes, P. Ehn, & M. Kyng (Eds.), *Computers and democracy – a Scandinavian challenge* (pp. 191–210). Aldershot, England: Avebury.

Heeg, G. (1995): Objektorientierte Systeme. In: Fachseminar Objektorientierung, Systems-Kongreß 1995, S. 81-90

Jacobson, **I.** (1992). *Object-oriented software engineering – A use case driven approach*. Reading: Addison-Wesley.

Krabbel, **A., Ratuski, S., Wetzel, I.:** *Requirements Analysis of Joint Tasks in Hospitals.* In: B. Dahlbom et al. (eds.): IRIS 19 "The FutureÓ, Proceedings of the 19th Information systems Research seminar In Scandinavia, August 1996 at L_keberg, Sweden. Gothenburg Studies in Informatics, Report 8, June 1996, pp. 733-749

Krabbel, **A, Wetzel, I, S. Ratuski:, S.:** *Participation of Heterogeneous User Groups: Providing an Integrated Hospital Information System.* In: J. Blomberg, F. Kensing, E. Dykstra-Erickson (Eds.): PDCÕ96 Proceedings of the Participatory Design Conference, Cambridge, Massachusetts, November 1996, pp. 241-249

Meyer, B.: Objektorientierte Softwareentwicklung. München: Hanser 1990

Reenskaug, T. (1996). Working with Objects. New York: Prentice-Hall, 1996.

Riehle, **D., Züllighoven, H**. (1995): A Pattern Language for Tool Construction and Integration Based on the Tools

and Materials Metaphor. In: James O. Coplien and Douglas C. Schmidt (Hrsg.): Pattern Languages of Program Design. Reading, Massachusetts: Addison-

Schmidt, K.; Bannon, L. (1992): Taking CSCW Seriously: Supporting Articulation Work. In: Computer Supported Cooperative Work: An International Journal, 1 (1992) 1, S. 1–33.

Suchman, L. (1995): Making Work Visible. In: Communications of the ACM, 38 (1995) 9, S. 56–64

Swartout, W., Balzer, R. (1978). On the inevitable intertwining of specification and implementation. *Communications of the ACM, 25*(7), 438-440.

Wirfs-Brock, R.J., Wilkerson, B., Wiener, L.: Designing Object-Oriented Software, Englewood Cliffs, NJ: Prentice Hall, 1990

Wulf, **M., Gryczan, G., Züllighoven, H.:** Process Patterns - Supporting Cooperative Work in the Tools & Materials Approach, Information systems Research seminar In Scandinavia: IRIS 19; proceedings, Lökeberg, Sweden, 10 - 13 August, 1996. Bo Dahlbom et al. (eds.). - Gothenburg: Studies in Informatics, Report 8, 1996. S. 445 - 460.