Blekinge Institute of Technology
Research Report No 2001:04

Design for change

by
Yvonne Dittrich, Olle Lindeberg, Ingela Ludvigsson, Lars Lundberg, Bengt Wessman, Wolfgang Diestelkamp, Marie Tillman

Department of Software Engineering and Computer Science
Blekinge Institute of Technology

Design for Change

Yvonne Dittrich*, Olle Lindeberg*, Ingela Ludvigsson, Lars Lundberg*, Bengt Wessman**, Wolfgang Diestelkamp***, Marie Tillman****

***Department of Software Engineering and Computer Science**
Blekinge Institute of Technology
Softcenter
S-37225 Ronneby, Sweden

****** Europolitan AB**
S37180 Karlskrona, Sweden

*****DieData**
wolfgang@bahnhof.se

1 Introduction

This report collects preliminary results, ideas and reflections of a research project 'Design in Use of Database Applications'.[1] The subject of the project is to explore the deployment of a flexible database server to provide maintainable and changeable computer applications to support a work practice underdevelopment. The idea is to develop software that can be easily adapted to a range of changing requirements, without necessarily recompiling the program or creating a new database. Maybe even a technically interested user can tailor the program using a special interface.

The report title 'Design for Change' indicates a change in perspective that is necessary when considering the development of a highly tailorable application. Software development is often perceived as producing a 'solution' for a 'problem'. A 'Design for change'-perspective puts the support of a work practice in the focus, that is expected to change in anticipated as well as in unexpected ways. The changes that can be foreseen can be taken into account during development, so that the application can be easily extended or that part of the application can be implemented in a way that allows for tailoring.

The idea of change is not a new one. Already 1980, Lehman argued that every program embedded into a social context produces the pressure for further development by changing the embedding practice (Lehmann 1980). Evolutionary process models like Steps (Floyd et al. 1989) organise the co-development of work practice and technology as a learning process through which developers and future users together design and develop the software and its context. Bødker uses the concept 'Design in Use' to describe how users creatively embed a given application (Bødker 1999), adapt it to their needs and – when possible – tailor it in co-operation with software experts or themselves (Trigg & Bødker 1994).

---

[1] The project takes place as a co-operation between Europolitan, DieData, and Blekinge Institute of Technology (BTH). Within BTH the main responsible is the research group UODDS (Use Oriented Development and Design of Software) but also the research group RISE (Research in Software Engineering) is involved. The project is founded to 50% by the industrial partners (Europolitan AB and DieData) and to 50% by KKS (The Knowledge Foundation).

Emphasising flexibility and changeability in design does not aim at replacing the method and tools develop to mediate the evolution of software and the co-operative design. It rather looks at what is required from a technical design to support for these processes. What aspects of the software will be stable, what part of the model is probable to change. The resulting software will not provide flexibility for all circumstances. 'Design for Unanticipated Use' (Robinson 1993) in this sense is not possible. But it might allow adapting the software to specific changes that are inherent in work practice that should be supported.
To provide such specific flexibility as part of a software system requires to combine sound knowledge about the use context with a careful design of interface and tailoring interface and often a more sophisticated implementation.

Diedata provided such a sophisticated piece of technology with the FLEX database system (Diestelkamp& Lundberg 1999). The system allows changing the database scheme during runtime. It does so by saving the definition of the database scheme as well as the data in a relational database. Of course there are disadvantages of such an approach: Storing and accessing data is more complex and requires more time. To make use of a flexible database requires a flexible interface as well which might lead to a less usable design. The data tables are not readable as tables of a conventional database. Changes might be more difficult to design and to test. If the system is not a stand-alone application it has to supply interfaces to other – not that flexible – computer applications. These interfaces might be more complex and less maintainable because of the additional layer of abstraction. And last not least users and developers have to cope with one layer of abstraction more than in development and use of normal applications.

Europolitan's software development provides a realistic scenario for the deployment of techniques to provide flexibility. The telecommunication world is rapidly changing with new types of services like GPRS, UMTS, convergent billing and service providers. This plus the lack of standard systems supporting the telecom-industry puts a lot of requirements on the IT-systems and the development of them. The software architecture/design must be flexible and easily changeable and new types of functionality must be easily added to existing systems.
The application that is subject to the co-operation computes payments based on contracts and triggered by certain events.[2] The software is supporting a strategic business area. Today only payments based on a certain event can be handled automatically. The business practice requires to base payments on other events as well. Other aspects of the computation, that today a hard coded should be subject to manipulation as well.
The software that is used at the moment had turned out to be too cumbersome to change. Beside specific restrictions in the interface that are worked around by changing the database directly, the adaptation of today's program to new types of contracts and payments is not possible. They have to be handled manually. The decision to replace the system had been taken before the co-operation with the university started. But would not a new system run into the same difficulties? Implementing a good support for today's practice would probably be outdated very quickly.
On the other hand, the criteria a payment can be based on do not change that frequently. Between these bigger changes that require a change in other systems as well the users should be able to recombine the different criteria to define a new type of contract and payment.

[2] To protect the business interest of our industrial partner, we do not tell about the character of contracts and we have changed other terms. In any other respect we keep our presentation to what we got to know from our interview partners and from the other sources of our research.

The database storing the information about the contracts does not only serve as an administrative tool, it provides also the interface to the batch program that computes the actual payments, triggered by incoming events.
As Europolitan's IT department develops and maintains software in close contact with the users and their departments, the contract handler project allows us to study a whole range of issues relevant for the development of flexible systems:

- What kinds of flexibility do the users need? How can they be provided? Does a well-designed interface solve the problem? Do we need possibilities to tailor the application? Or does a change require programming anyhow? Who is able to, and who can be allowed to do changes in a system when errors are critical for the whole business?
- Are flexible interfaces still usable?
- What about the interaction with other programs?
- How does the deployment of flexible technologies and meta-modelling techniques influence the development process? How does it interact with development tools and technologies used by the software developers?
- How does such a change in paradigm change the development, use, and maintenance of software? And how does it change the relation and intertwine of these practices?

This report is meant to recapitulate what happened during the first year of the joined project, and to collect preliminary results, ideas, and reflections. It does not have the concise structure of an article.
In the next section we tell the history of the project.
The section thereafter analyses from Europolitan's point of view the efforts and gains in the research co-operation.
Section 4 summarises a reconstruction of the development and change history of the program that is today used to administrate contracts and compute payments.
In section 5 we present the results of two students who did a workplace study and designed an interface and a related tailoring interface together with the users. They analysed, what different kinds of flexibility were needed by whom in what context? And they looked into different ways to provide them.
Section 6 summarises the project activities regarding (meta) modelling and flexibility. Several prototypes have been implemented as part of the project we compared the different solutions regarding flexibility and other criteria.
The last section attempts to compile these results into a first sketch of a methodological toolkit consisting of checklists, hints, methods and tools that can be used or adapted. It is meant to support software developers who struggle with a project where a tailorable program might be useful.

Co-operation between industry and academia is not always an easy attempt. Different cultures, different goals, and different rationales meet. A fruitful co-operation requires mutual tolerance and respect. A specific problem in this project was that the meetings had to take place in a mixture of English and Swedish, as one of the participants does not speak enough Swedish. Despite these difficulties, the intermediate resumée is positive. A lot of the above questions could not be formulated without a realistic application. To bring academic experiments to bear in industrial settings require such a co-operation. On the other side the industry can learn a lot from the university about new methods for developing systems.

Comments, feed back and own experience are welcome. Please contact one of the authors!

2 Project structure and timeline

How to organise such a co-operative research and development project? None of the participants had experiences of co-operation between university and industry before starting the project. We therefore decided on a very situated way of planning and co-ordination. We agreed on co-ordination meetings between the responsible persons at the university and at Europolitan every six weeks. Diedata was involved in other projects at the university and in student projects, so we did not expect formal meetings to be necessary. Beside we met around thematic issues in different combinations. For the project start up and at several project meetings, the whole project met, discussed what had happened so far and decided on the next steps.

The timeline in figure 1 gives an overview of what had happened during the first year. The different parts are shortly described thereafter. Some of the topics are taken up later in the report as well.

Europolitan's Prestudy

Before the actual pre-study started, a preparing work was carried out to clarify the parts to be analysed in the pre-study. During the preparing activities the end users took part in workshops and interviews carried out by two students of the study program 'Human, Computer, and Work' at the university. When the pre-study actually started it could be concentrated mostly on those parts that was expected to be implemented in the project. There were several decisions to take considering what to be implemented in the first iteration and what parts that had to be postponed for the next iteration. The end users as well as the 'buyer' of the system participated in meetings twice a week where the functionality of the system was prepared. The goal was to obtain a picture, as clear as possible, of what the system would achieve at the end of the project. Another issue to clarify was the dependencies to other systems, both the systems we had a dependence on and the systems that had a depend on our program. This preparation made it possible to start the implementation of the system almost immediately when the project started.

Tailorabiliy in Use: A Bachelor Thesis

To explore the use side of tailorability and flexible software, two students of the study program 'People, Computer, and Work' studied the work practice around the use of today's system and explored the design of a tailoring interface. The thesis contributed to the analysis of the meaning of flexibility for this specific application (see Section 5). The students organised two design workshops together with the users – one focussing on the interface design and one on a design of a tailoring interface. Their mock ups provided a first idea, how tailoribility could be presented to the users.

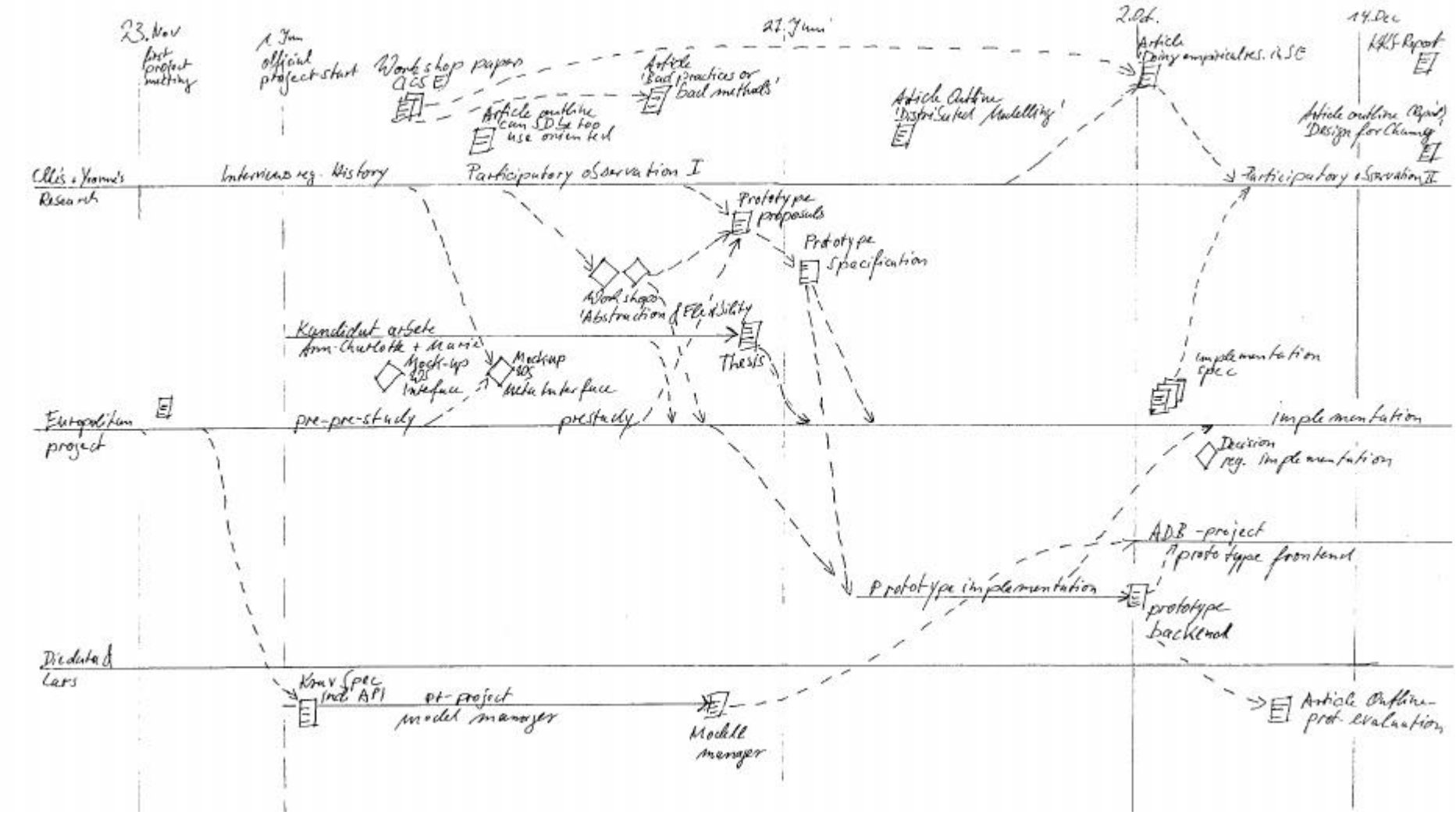

Figure 1: Timeline of the Project in 2000.

A Model Manager for the FLEX Database System: A Small Team Project

To complement the FLEX database system with interactive model manager and to provide even a general interface to create and administer the context of the database a small team project was set up. Five second year students from the Software Engineering program designed and implemented a user interface to the FLEX database system. The resulting program even contained an application program interface (api) so that the functionality to manipulate the data and the data model would be available from within an application, that would implement a domain specific model and an application specific inter face. Even as the program was not used later in the project, the ideas were reused in the front-end prototype implemented by another student project during Fall 2000.

Reconstructing the History of the Program

To understand the kind of changes that might be required of the new contract handler, we tried to reconstruct the history of the development and change of the software that is in use today. Key members of the former project were interviewed and project documents were evaluated. The analysis of the program and its source code contributed to the understanding of the problems with the existing system. Section 3 summarises the result of this reconstruction and some further reflections.

Participatory Observation 1

During spring 2000 one the researchers accompanied the of the ongoing pre study of the development project. This kind of empirical research is called 'participatory observation' in the social sciences. The researcher takes part in the setting he observes and uses his experiences and observations to reflect on the situation in a scientific way. The pre study was driven by the IT department but also users was part of the group working with the pre-study. A number of meetings with this group took place. There had been also a number of working meetings where some of the members of the pre study group met with different peoples from other parts of the organisation. The reconstruction of the history and the workshops described below can be regarded as part of the empirical research as well.

Developing the Research Methodology: Co-operative Method Development (CMD)

Subject of the research part of the project is the development and the deployment of technical possibilities to implement flexible and tailorable applications and the implications of the deployment of such technologies on the software life cycle and software development processes. Use, maintenance and (further) development change if software can be adapted by the means of a user interface.

Existing approaches focussing on quantitative evaluation of methodological changes in the software development process or deploying ethnographic methods to understand the work practice of software development were not applicable in this context. In reflection on former projects and the starting research in the context of this project the researchers participating in the project developed a specific interpretation of action research, we tentatively call co-operative method development. (See also (Dittrich 2000) (Lindeberg 2000), (Lindeberg & Rönkkö 2000), (Dittrich 2002).)

Methods can be understood as providing support for a certain way of doing software development and not lending itself easily to others. On the other hand, as detailed and prescriptive as a method might be, it has to be interpreted and adapted by the software developers regarding the circumstances and contingencies of the concrete situation. The actual implementation has to be designed in use. Learning from participatory design and evolutionary development, method development can be organised as design – implementation

– evaluation cycles, providing space for learning among all participants, practitioners as well as researchers.
As in participatory design, empirical methods can be used to supplement the co-operative effort and to help the researchers to build up an understanding for the practice the other project members are involved in. Results of the empirical studies can serve as input into the co-operative process.
Such an understanding of co-operative method development could be developed into a framework to conceptualise and organise empirical research in software development together with the practitioners involved. (Dittrich 2002)

Workshops on Flexibility

During spring we organised two half-day workshops focussing on flexibility of programs and meta-modelling techniques and how and whether they can be applied on the project. In the first workshop we looked at different techniques and at programs that used these techniques. Excel provides a 'programming language' about fields on a spread sheet and arithmetic computation. The filter tool in a mail system allows implementing rules to have mails sorted automatically by header attributes. To be able to implement rules, the filter tool provides a model about what constitutes an e-mail under a filter perspective. The rules refer to this meta-model. A framework allows to extend certain parts of the class hierarchy easily without necessarily changing the implementation of other parts. We used a graphical editor as an example, inspired by Anders Mørch's work (Mørch 1997) We then applied these ideas on the design of a card game system that allows to define new games easily.
In the second workshop, we discussed the structure of the contract and payment system and possible implementations of flexibility. As we expected to find quite different logics for the contract administration part and for the payment part (frontend and backend) we split into two groups. In both groups we started by listing the different contracts and payments under discussion and structuring them.
For the frontend we identified several groups of contracts handled in similar ways in the user's work practice as well as regarding prioritisation and combination of them. Periodical payments build a group by themselves and are treated independently. Each contract description is of a certain type that is related to a certain type of events that triggers the payment. Contract type are grouped according to professional logic and similar treatment.
For the backend we distinguished payments triggered by one time events and periodical payments. For the latter one needs a trigger function to produce the 'event' that contains the data that allows compute actual amount. The events can be treated similar. A kind of pattern match algorithm – that actually might just be a complex SQL statement – can then be used to select the contract descriptions that fit the same set of data. If more than one description match, the result is sent to a prioritiser that selects the contracts according to which the payments should be computed. This prioritiser can be hard coded or based on a set of rules that can be changed through an interface.

The Backend Prototype

During summer DieData and Europolitan implemented together a prototype implementing part of the backend functionality based on the FLEX database. The backend prototype matches an incoming event to the contract descriptions triggered by the event. The prototype showed that it was possible to handle new kinds of events and corresponding contract types by only changing the database.

The Frontend Prototype

During Fall, a student project implemented what we came to call the frontend prototype. The prototype implements a tailoring interface allowing to define new contract types during runtime, and allows also the administer contract descriptions. Even contract types based on new events could be defined, as soon as a description of the event was added to the database.

The Beginning of the Implementation

The pre-study was completed and the project board accepted in September/October the project proposal. The decision was based on the requirement specification and an implementation proposal. The project started immediately. The project group was reorganised and four more software developers joined the project. Both users and developers are members in the project group, as in the pre study group. The issues taken up in the research co-operation about flexibility and meta-modelling has influenced the project into trying to make the system as flexible as possible while implementing it with conventional tools and techniques. The first part of the project until Christmas has held the time plan.

Participatory Observation II

During fall one of the researchers continued the participatory observation by taking part in the meetings, first in the pre study and later in the project. After the actual programming started the major part of the activities in the project took place outside of meetings. To get an insider's view of these activities the researcher joined the project as programmer implementing a small part of the system. This made it possible to observe how the structuring of the system developed and gave insight into the work practice among the developers.

The Continuation of the Project

During 2001 the team at Europolitan will finish the implementation, test and take the new program into use. We began to evaluate the new program as well as the prototypes regarding flexibility and other use and development qualities. The researchers will further accompany the project. We are also interested in how the possibilities for flexibility will be deployed. We plan a workshop 'project organisation and use orientation' to reflect on the project at hand and to make the measures explicit that made the so far successful project possible.

3 Efforts and gains from Europolitan's point of view

Efforts

The biggest effort is the hours that the resources from IT and other Europolitan departments have contributed to the project. Since the autumn 1999 the research project at BTH has been studying a software development project at Europolitan. Several case studies have been done on the work situation of the end users to this new system. Interviews have been done with people from different departments and consultants involved building the currently used system. Europolitan and Diedata have built a prototype to evaluate a new flexible database. Europolitan has been involved in three different projects with students from BTH, and we have also participated in workshops about metadata, architecture, developing and evaluating the flexible prototype.

Gains

Europolitan has learned new development technologies from the university and will use them to develop flexible solutions and systems that will meet the software requirements both from Europolitan's internal and external customers.

The gains from the project so far are more awareness of the need of flexible design of software architecture and database design. In no doubt the architecture in the software developed by the project at Europolitan have influences from the way the flexible database works. Even though the project did not use the whole concept with the flexible database, new ways to add more flexibility to the design have been used. High flexibility has also a dependency to usability, both for the end users, but also for the developers. It is difficult to see the actual data in this kind of database, which is important when the software is in production. This is one of the reasons why the project chose not to use the whole concept with the flexible database.

Hopefully, the new methods will to add more flexibility of the design that will come out from the ongoing project and will be used in the software development process at Europolitan in the future.

Beside the technical part of the project has the project contributed to a better contact network between the university with both researches and students. From Europolitan's side that is much appreciated.

4 History – or can software development be to use oriented?

To better understand the problems that lead to the decision for re-development, the researchers decided to study the history of the development and the change of the program. The project members from the earlier project that were still available were interviewed, we studied the documentation and we had a look at the interface as well as the source code of the program. To our surprise experts from the use department were involved in requirement analysis, design and development as well as the design and implementation of the changes. The majority of the project team were domain experts or future users. The problems popping up in the use of the program could not be blamed on too little user involvement.

What then could have been the reason? Several issues came up in the interviews:

- **Control:** The semi-automatic way to compute payments before the introduction of the program required a high amount of control and consistency checking. From the economic point of view as well as regarding the reliability of the company for the receivers of the payments, the computations had to be right. When the software was developed the emphasis of control resulted in a too rigour interface. Certain fields in the contract descriptions that in practice change were not allowed to change. A contract description had to be punched in twice in order to double-check the input. These had been requirements from the use department.
- **Inflexibility:** Beside the above mentioned interface inflexibilities, the system hard-coded for example the combination of and prioritisation between different payments triggered by the same event, according to the then valid rules. The combination and prioritisation was embedded in a hard to understand 'spaghetti code program' so nobody dared to change it.
- **The not existing design:** The code in the frontend as well as in the backend was not maintainable. The programmer involved left the company even before the program was taken in use. The efforts to debug and start up operations already showed the difficulties arising from the not existing structure or design of the program. Change requirements to correct the inflexibilities of the user interface for example could not be

implemented. They were worked around by changing the database directly. Other changes – to add new contract types for example – were added in way that touched the existing code as less as possible. All other change requests had to be put down.

Can Software Development really be too use oriented? Perhaps not – but the neglect of the technical design and its implication might have led the problems described above that also influenced the use quality of the program; would the design have been better, the unnecessary inflexibility of the interface could have been corrected easier. The evolution of the program according to the developing work practice would have been possible within certain limitations.

In an article under development, we relate the result of our reconstruction to Peter Naur's classical article 'Programming as Theory Building' (Naur 1985). We argue that as there has been a theory about the use of the application, there has not been a sufficiently elaborated theory about the software as a technical artefact itself. The developers did not make their own conceptualisation of the system. They stayed with the users' concepts that were related to a work practice based on paper, file maker and excel sheets as tools. They neither developed a structure for the software, but related a database and a user interface in an ad-hoc, tinkering way.

We use the methodological discussion in ethnography and apply the concept of 'going native' when doing ethnographic fieldwork to software development in co-operation with users. As ethnographers have to become a (temporary) member of the culture and community they study, they have to withdraw and reflect about their participation in order to be able to write about their findings. In a similar way, software designers and developers have to understand the work practice in the use context from a member's perspective, they have to step back and reflect the users' concepts in respect in relation to the possible technical support in order to develop a suitable program.

5 Different kinds of flexibility

What kind of flexibility is needed? And which is possible, provided the given use context and the interaction between the application and other systems. These are the questions to be explored in this section. The project started with the declared intention to look into technical possibilities to provide flexibility and to explore their application in the contract handler. But what flexibility do the users actually need? What are the inflexibilities of today's applications that hinder the use and that hinder the further development of the business practice in this area.

In literature this question is not explored at all. Flexibility is mainly discussed as a lack, as the inflexibility of software systems that enforces a specific organisation of the work practice of the users (Henderson & Kyng 1991) and that disturbs the work practice and its co-ordination (Bowers et al. 1995) and forces users to 'work around' the software.

Tailoring – providing a certain flexibility by means of a user interface – is mainly discussed in connection with general purpose software that is adapted to individual or organisational needs. In our case, already the basic application is a special purpose application. The purpose with adaptation is not to fit the application to a specific context of use, but to provide the possibility for the development of the software according to the anticipated development of work and business practices. If a new type of contract is decided upon the application should be easily adaptable to these changing requirements. Tailoring in our case is not the adaptation of an application to personal needs and preferences, but the further development of a common object of work.

In literature systems to be tailored are mainly stand alone systems: the user interaction of operating systems (Henderson & Kyng 1991), word processors (Trigg & Bødker 1994), cad systems (Gantt & Nardi 1992), or graphical editors (Mørch 1997). In our case the application is embedded in a network of related applications. Its functionality depends on interfaces provided by other programs and it has to provide interfaces to other programs. And last but not least the application is handling payments. Secure operation is important from a financial point of view, but also to provide a competent organisational interface to the receivers of the payments.
What are the actual needs of the different actors in the organisation? How do the concepts and technologies mainly applied by researchers apply in an industrial context? What is suitable for this specific situation?
As part of the research project two students of the bachelor program 'people, computers, and work' studied the work practice of the use context of the existing program and explored the interface design of a tailoring interface. This section is partly based on their work, partly on the pre-study and the discussions during fieldwork and project meetings.
We first introduce the actors involved in the use context of the application. thereafter we take a closer look at the development context and the technical infrastructure. Thereafter we identify three levels of flexibility which are partly motivated by the professional perspective of the user, partly by the perspective of the IT department or and the technical environment.

5.1 Use context

The contract and payment group

The tasks of the users that are supported by today's software are to keep track with the contracts and to make sure, that the right amount of amount is paid to the receivers. The system is used to administrate the information regarding the contracts agreed and to save the conditions necessary to compute the actual amount. Payments that can not be handled by the system are computed manually. The people responsible for these tasks also answer questions from receivers regarding the actual payments. As they are the ones to carry out the work, they take part in discussions regarding the development of this business area.
In the beginning the researchers of us were puzzled by the contradictory statements regarding the frequency of changes and maintenance efforts. Sometimes people talked about weekly changes, sometimes about a few major changes the system had undergone after the first debugging period. The studies of the use context clarified this contradiction. The contract handler is steering the computation of actual payment, which are automatically passed on to the financial system. The correctness of the data and the computation is therefore an important issue. To prevent errors caused by wrong input, the input was constrained in a very rigorous way. The constraints did not always match the actual work practice of the users. For example the list of receivers a contract applies for can not be edited. The real contract in paper form can be between the telecommunication provider and a group of receivers. If a receiver for some reason becomes a member of such a group, the contract holds also for him. As the application could not be modified, these constraints were worked around by changing the respective data in the database directly. This had to be done by a person from the IT department.
Inadequate constraints had been a constant an annoyance for the users. They implied interrupts in their work routine and sometimes meant a delay until the change was available in the database and the payments could be handled by the system.
In discussions around the development of the business practice in this area the proposals had to be put down too often. Part of the contracts and payments that can not be administrated via the software are handled manually respectively with the help of excell sheets. Sometimes the

manual treatment would be too cumbersome; sometimes the data the payment should be based on data that is not available

Strategic business development

Providing mobile communication is a competitive and rapidly changing business. Flexibility in the applications area under discussion is of strategic importance. The groups responsible for the development of this business area are interested in defining new types of contracts and even base contracts and payments on new types of events. The combination of and prioritisation between different contracts that are fulfilled by the same event are competitive instruments as well and should be adjustable from case to case. Neither of that is possible with the old system.

On the other side, the old system provided some regulation, constraining the business to a – hard coded – policy. During a meeting discussing the possibility of a tailoring interface, a member of the use department raised a question in this direction: If nearly everything is technically possible how do we make sure that we only sign contracts that make sense from an economic point of view. If the software is flexible, structures to decide upon a wanted and suitable business practice have to be established organisationally. Policies have to be developed and implemented.

Controlling

The person responsible for the statistics and controlling was interested in the design of the application from a different point of view again. The contract handler provides the information about all payments in this part of the business. He wants to have easy access to this information. Each manually handled payment means extra work.

5.2 The development context

The IT department at Europolitan is not only responsible for development and maintenance, but also for the operation of the different software applications and the infrastructure they provide for the company. The co-operation between IT department and use departments is based on long time relationships. Use problems provide a direct feed back about the use qualities of the software.

The problems with the contract handler were known from the very start. The programmer responsible for the implementation of the existing program left the company even before the program was debugged and taken in use. As the software was not maintainable, adjustments of the interface were not dared by the software engineers that took over. Two bigger changes were done, adding independent modules to the program and changing the back-end algorithm. Otherwise the original program nearly stayed as it was after the testing. The frequent database patching was not acceptable on the long run. Due to the lack of resources and the insecurity regarding the development of the business practice, the IT department hesitated to start a new project sooner.

The request from the use context regarding a higher flexibility has been acknowledged. However as the IT department would also have to firefight in case of errors. The data describing contracts steers actual payment. Errors in input and – even worse – errors in the introduction of new contract types could result in faulty computing that would promote into the economic system. To correct such errors would be a cumbersome job for the software engineer responsible for the operation. Robust operation and a certain control regarding consistency of the input is important as well.

Experience showed that interface design was an important issue for the users. A flexible assembly of data input objects would probably not be accepted by the users. Additionally the project members wanted to use the available technology. Powerbuilder is a rapid prototyping

tool providing an elaborated framework that rationalises the development of standard database applications and the connection to sql databases.
Flexibility was needed, but how much, and which technology to use? And how does flexibility influence usability and robustness?

5.3 The technical environment

Another constraint for the flexibility provided by the system is the (in) flexibility of the technical environment. To compute the concrete payments, the system has to be provided with data describing the triggering events that is produced by other systems. As long as new contract and payment types use the provided set of data only the system itself has to be adapted. As soon as contract and payment should be based on other data, interfaces to other programs have to be created. This would require programming anyhow.

5.4 Different kinds of Flexibility

Bringing the different interests and constraints together, one can distinguish three levels of flexibility that can be regarded as relevant for the project. The levels we identified relate both to different requirements from the use context as well as to differences in the technical difficulties to provide them.

Level 1: Providing a good user interface

The constraints in the user interface should match constraints that are relevant from a professional logic to prevent serious errors. This had not been the case in the old program probably because paper based administration allows to handle exceptions in a different way. On paper it is quite easy to add a receiver to a certain contract, even if this should only be done in exceptional cases. In some cases warnings instead of hard constraints might be a better solution. Flexibility in this respect is subject to research in Human Computer Interaction and Computer Supported Co-operative Work.
To improve the solution for the new system, during the pre-study the different constraints have been re-evaluated. Experience from system operation and maintenance also gave indicators about which constraints have been helpful, and which turned out to be a hinder. Version control for all contracts is implemented.

Level 2: Providing possibilities for new types of contract and payment based on the available data.

Already the old system provided the possibility to administer different contract and payment types, even as they were not addressed as types. The step to call them different types of contract and payment based on the same triggering event, was motivated in the developing professional way to talk about contract and payment. Making contract types available as objects to manipulate is a consistent extension of the existing work practice. During the project the concepts of contract type and different kinds of contract and payments developed in the discussions between users, developers and researchers to distinguish and describe different levels of changes and their implications from a technical point of view.
The FLEX database system can be used to provide this kind of flexibility on the database level. A student project showed the possibility to build an application based on the FLEX database that is capable to provide this flexibility for the user. A tailoring interface allows to pick the available data items from the database and to combine them as conditions for the payment according to a new type of contracts. A semi-automatic generation of an interface is possible – even with Powerbuilder, the development environment Europolitan uses.
To allow changing the rules for prioritisation and combination if several contracts are matched by the same event is also on that level of flexibility. Here no new external data is needed. What is needed is a suitable language to express the prioritisation and the

combination between contracts and the payment according to them and an algorithm interpretation of these expressions.

Level 3:
Providing the possibility to design and administer contract and payment based on other events.

The groups responsible for the development of this business area are interested in defining new types of contracts and even base contracts and payments on new types of events.
This kind of changes is depending on the availability of the data that serves as condition for the contracts and payments. Even as the prototypes showed that the core system could be designed in a way that allowed for this kind of flexibility the interface to other systems had to be implemented 'by hand'.

To map out different interests regarding flexibility and to look at technical constraints based in the interaction between different programs as well as the implementation technology provided a good starting point to find professionally and technically motivated levels of flexibility, that can be used for design discussions.

5.5 But do we want so much tailorability?

From the very beginning questions and concerns regarding the implementation and use of flexibility and tailorability had been discussed. If the constraints are not longer implemented in the software, how can we make sure that out experiments do not cost us too much? Users themselves were doubtful whether they want to take the responsible and be able to implement new contract and payment types. Secure operation is another important aspect to be considered.
The solution that in the moment is about to be implemented realises partly level 2 flexibility for contract and payment based on a certain triggering event and allows to flexibly redefine the combination of and the prioritisation between different types of contract and payment. The tailoring interface is – in the first place – not designed for the users but for the computer scientist responsible for operation and maintenance of the program. Level 3 flexibility is provided in a semi-automatic way; the data that serves to trigger the payment that are based on a different type of event is assembled manually. This allows to implement them immediately but will cost the manual work to collect the data. In the next version the system can be complemented with the new contract and payment types.
It remains subject to further research, how the practice of tailoring and adaptation will develop throughout the lifetime of the new contract and payment system.

6 Meta-Modelling and Flexibility

To construct a flexible system where anticipated types of changes - that in a normal system would need program modification – can be handled without programming is not simple task. It requires a more abstract view and modelling of the system. We have to construct a model of the system that can handle also the changes. One way to do this is to make a meta level shift and letting, at least part of the system, be controlled by meta-data describing the system behaviour and/or data structure.
The next question is 'What tools and techniques we need?' One promising technique – and the incentive for the whole co-operation – is the concept of meta modelling databases, databases that allow to change the structure of the data to be stored, under operation. The aim with using such a database is to simplify the system, without it we might have to implement some of its capabilities as part of the system.

In this section, we describe a fictive system using the FLEX database (Diestelkamp & Lundberg 1999) by putting together the different prototypes to show how a totally flexible system could look like. We also describe the overall architecture of the software under construction in Europolitan as it developed until januari 2001. In this design certain meta-modelling techniques are used. To do so, we begin with presenting first the overall system architecture and a rough conceptual model of contract types. Afterwards we sum up by posing the question that is subject to two evaluative workshops in spring 2001.

6.1 The system architecture

The system is built as two databases and two applications.

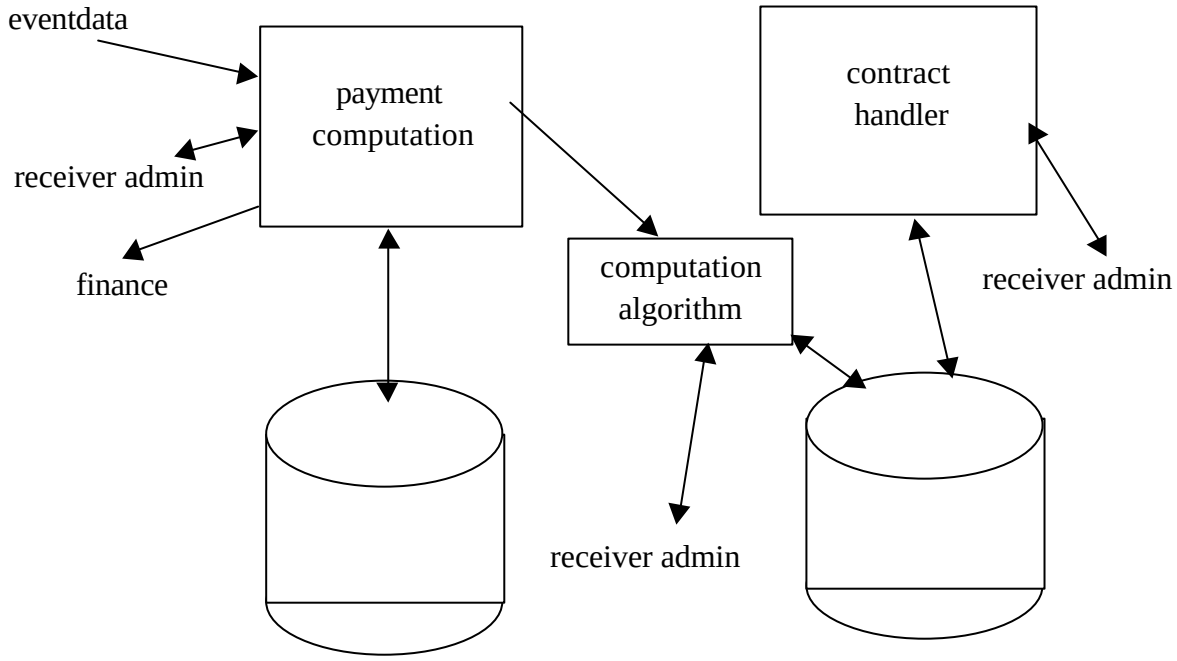

Figure 2

In figure 2 is a rough sketch of the system. The computation algorithm is realised as a stored procedure. It computes and prioritises the payments. The contract handler's database is used to store the data defining contracts.
When a new contract is entered a number of checks is done to assure the validity of the data. Some of them use information from other systems, e.g. when a receiver is entered another database is consulted to assure that he is a possible partner for this kind of contracts.
The payment computation is designed as a batch program processing the datasets describing an event, producing reports to receivers and triggering the financial system. It gets a file with the triggering events, processes them to compute the payments and then triggers the payment. The actual match of the events with the contracts is done in the computation algorithm.
The payment computation's database is split up in three parts; an 'import' table for the incoming events, a 'temporary' table for payments that are processed and ready to be paid, and a 'history' table for payments that had been paid, to allow for corrections. The program takes each event from the import table and first checks all parameters for validity. Most of the checks involve accessing other systems. Thereafter, the actual payment is computed by calling the computation algorithm. If no errors occur, a list of contracts defining the amount to be paid is returned.

The algorithm that computes the actual payments is the only connection between the two applications. This algorithm is implemented as stored procedure in the contract handler's database. The structuring of the payment computation and the clean interface between the two parts is one of the improvements the project resulted in.

6.2 A conceptual model

During the design discussions we have constructed a conceptual model with four levels of abstraction. The actual data that is stored describes the contracts the payments are based on. There are several types of contacts defining the next level. Some contract types are handled in a similar way; this gives the next level, *contract_groups*. At the top level of the abstraction hierarchy are the *event_types* where we group together contract and payments related to what event triggers them. To help the reader to understand this model, we present it in figure 3 in an object-oriented notation.

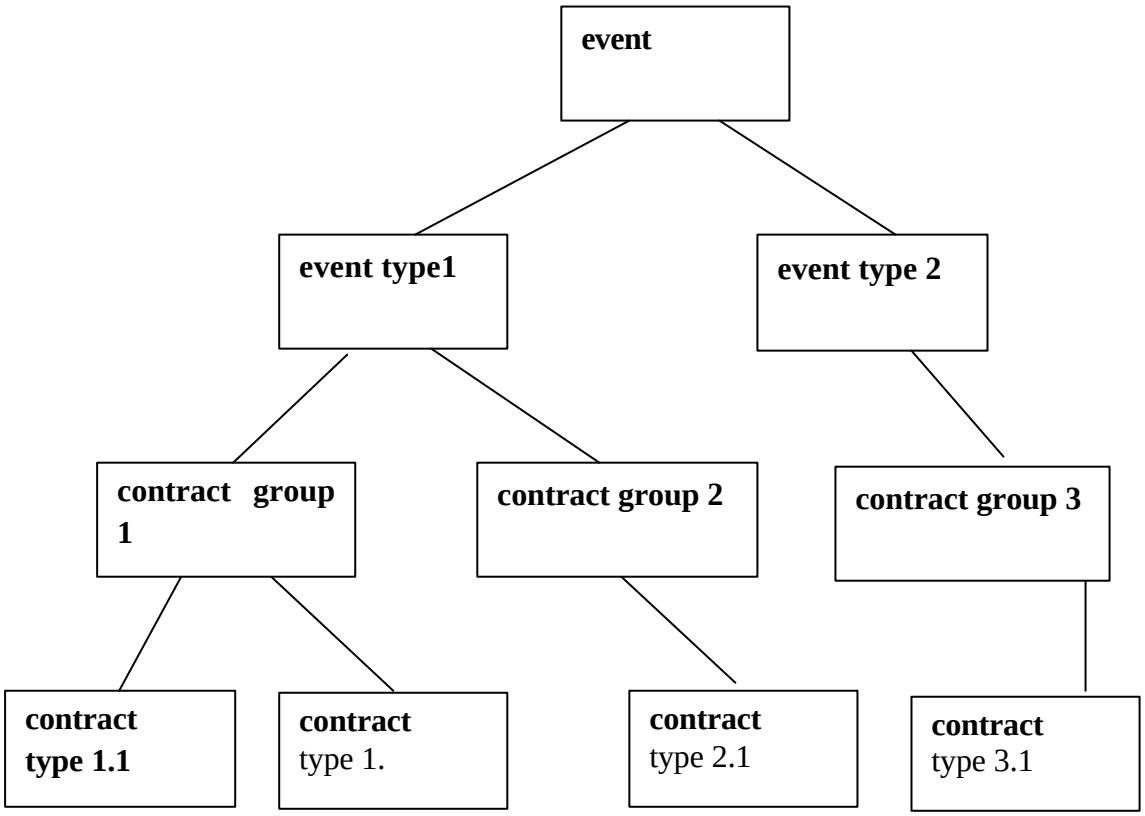

Figure 3: Type Hierarchy

The actual contracts would in an object-oriented implementation be objects belonging to the concrete classes in the bottom line. The rest of the classes would be abstract.
A simplified model – leaving out the group level – was the basis for the prototype implementations. In the system that is now implemented, the event types are not explicitly represented but they structure much of the code. There only exist two event types in the system and in the part of the system computing the payments the code is separate for the different types.

6.3 Meta-modelling and the prototypes

If the conceptual model described above is implemented in a normal and straightforward way the classes become parts of the program and data structure of the system. We would get a

database table named ‘contract type x’ where the actual contracts was stored. In the user interface we would get a hard coded window for entering such contracts. In a flexible system we want to be able to add new contract types without programming, if they can be constructed by putting together parts that already exists in the system. To do this we have to make a meta shift and put the description of what a contract type is in the database instead of hard coding it into the program. When storing actual contracts the program will then interpreted the corresponding type description to find what data is needed for this type of contract. The user interface must be constructed in the same way. When a user wants to enter a new contract he/she first select which type of contract. The program access the database to get the meta description of this contract type and, during runtime, build up a screen with the right fields for the user to enter his/her data in. We also need a tailoring interface where new contract types can be added to the system.
In summer and fall two prototypes that implements part of such a flexible system was constructed. The backend prototype was implemented and realises the computation algorithm. The frontend prototype was done as a student project at the university and implements the user interface of the contract handler part and also a tailoring interface that allows to construct new contract types. With the use of the FLEX meta-modeling database it was possible to make the implementation with full flexibility. Both prototypes are controlled by the meta descriptions stored in FLEX that also handles the storing of the actual data. The prototypes do not implement the full functionality of the system. Some of the parameters in a contract in a sharp system have to be selected and checked in a dialog between the user and the system, to do this other systems have to be. This was to complicate to implement in the prototypes.
A perhaps more serious problem is that the user interface was far from what would be acceptable. The interface is generated based on the parameters a contract type has and does not take professional practices into account. A program that should be used has to provide the possibility to design also the layout of the interface in the tailoring interface.
The evaluation of the prototypes is still ongoing, a tentative conclusion is that they show the possibility to do a flexible system but not how to provide an acceptable user interface. They serve as a proof of concept for the deployment of a flexible database system. During the evaluation other aspects of the meta modelling approach has turned up, some of these are;

- What competencies are needed for developing such system?
- What tools do we need to handle the maintenance and operation of a flexible database?
- If we build a totally flexible system the surrounding non-flexible system will anyway stop us from using the flexibility
- It is not enough to add a contract type in the system, the whole organisational context will be affected.
- Who is permitted to use the ‘meta interface’, who guarantees that the system will function correctly from a business point of view?

6.4 The contract and payment system under development (meta modelling light)

The research co-operation between researchers and the IT department has affected the ongoing development project into trying to make the real system as flexible as possible. The experience with the old inflexible system has influenced the project to think in this direction as well. The result of this is that the new system has aspects of meta modelling. Some things that in most systems would be hard coded has been placed in the database. The user interface has been structured into a set of building blocks where the interface for each contract type is built up of smaller units that can be recombined easily.
The line between what is meta modelling and what is good programming practice is not easy to draw. In the contract handler most of the functionality is controlled by which group a

contract belongs to. This means that adding a new contract and payment type without adding a new contract group is a very small effort. If the new type is too different to all the existing types a new contract group has to be added and this is a major change. Also other meta modelling aspects are used. For example the prioritisation between contracts is defined by a table and can be changed from the administration interface. The difference in flexibility between the system under development and the prototypes turned out to be much smaller than we expected.

6.5 How much 'meta' do we need?

As a result from comparing the prototypes and the system under development we began to put forward a different set of questions: How much meta do we need? Will we really need the full possibilities of a meta model database? Or can we make the system 'flexible enough' using some conventional tools and using just the concepts taken from the meta modelling approach? In our case, it is possible that the system depends to much on other system and their properties to be a good candidate for complete flexibility.

7 Towards a methodological toolkit for flexibility (instead of conclusions)

In this section we start to collect what we – so far – can tell about how to provide for flexibility in computer applications. We do that in form of a 'toolkit', a set of recommendation, checklists, small tools, 'design patterns' and technical possibilities. We will finish the section with some ideas regarding future work.

The contract and payment system that is subject of the project can be described as an example for special purpose interactive systems. Individual productivity software of co-ordination tools (Mørch & Mehendijew 2000) require a different approach. With flexibility respectively tailoribility we mean in this context the possibility to adaptation of a program to changing work and business practices. We don't consider the adaptation to individual preferences. The embedding of the application in a technical infrastructure adds additional difficulties, but does not change the toolkit systematically.

One of the main lessons learned during the project is one of he basic principles of use orientation: The needs of the users and the developers should guide the design and development, not the fancy of technical possibilities. The following hints, ideas and tools should open up for the creative use and combination of technical possibilities and support the artful integration (Suchman 1994) of work practices and technology.

7.1 How to go about

Flexible programs require a more conscious design and they shift the borders between development maintenance, drift and further development. As change is taken into consideration from the very beginning, an evolutionary approach – iterative development with evaluation of the software in use – offers itself already for the initial development. (See (Floyd et al. 1989) for an example.)

Not only the functionality of the software has to fit with a (future) work practice, also the tailorability has to match anticipated changes and has to be comprehensible from a professional point of view. Co-operation between developers and users is therefore even more important than in projects aiming at less flexible software.

The anticipation of change has to be matched by the technical possibilities to provide flexibility. Analysis of today's work practice, design of the software and experiments with technology and design patterns should be consciously intertwined from the very beginning. This underlines the research challenging the independence of analysis and design and their separation. (Christensen et al. 1998) (Bürkle et al. 1995)

As the basic design constrains the possibilities for changes in further iterations, the analysis/design phase in the beginning is for tailorable systems even more important than for the development of conventional systems.
What is valid for 'normal' domain specific interactive systems is even more important for the development of software with additional requirements for flexibility and tailorability.
The following parts of our methodological toolkit relate to the first analysis/design phase of such an iterative and co-operative development process. That corresponds to the phase of the project and the experience we could evaluate so far. They should be understood as a complement to other methods and tools used for analysis and design.

7.2 Mapping out needs and constraints for flexibility (a structured checklist)

The use context

Most often users' complaints about the inflexibility and rigor of a computer application will trigger the discussion of providing a flexible application. Different members of the use context might use the term 'flexibility' in different ways. To understand the different interests and take the different needs into consideration the following questions provide a starting point:

- Who are the actors involved?
- What do they mean when they talk about flexibility?
- What do they mean when they talk about inflexibility?
- Are there contradictions between the interests of different actors?
- What kind of changes have been experienced in the work and business practices in the past?
- What kind of changes do different actors anticipate?

The result of the investigation guided by these questions can be used to identify stable parts of the applications and the parts that are most likely to change. The way members of the use context talk about the anticipated changes can provide a starting point to develop professional meaningful abstractions, which can build the base for the design of both the normal functionality as well as the flexibility respectively tailorability.

The technical context of the future application

If the software under consideration is depending on or providing interfaces for other computer systems, these interfaces are constraining the flexibility that can be provided as well as the technology to provide it.

- On what interfaces from other system does the software under development rely on?
- What kind of interfaces have to be provided for other systems?
- How do these interfaces look like today?
- How easy can they be changed?
- Can they be implemented in a more changeable way?
- (How) Do the answers to these questions relate to the anticipated areas of change in the use context?

The development context

The technology to be used constrains the possibilities to implement for flexibility and changes the evaluation of the trade-offs discussed in section 7.4. The implementation environment used in Europolitan for example rationalises the interface to a relational database by providing specialised interaction frames relating user interface and table manipulation. Object oriented design is possible, but would in most cases require unjustifiable overhead compared with

using the already implemented features. The development context as well as the implementation technology has to be taken into account when designing for flexibility

- What kind of technology will be used for the implementation? How does this technology relate to conceptual and technical means to implement flexibility?
- What effort (purchases, competence development, consultancy) would be required to deploy a more fitting implementation technology?
- What are possible compromises?

Trade offs

What other functional and non-functional requirements could interact with the request for flexibility? In section 7.4 we collected the trade offs we so far experienced. To consider them from the very beginning will help to decide on implementation technologies and design approaches.

7.3 Exploring means of implementation

Thinking one level more

Tailoring and design for tailorability and change requires to not only model the information to be handled and hot to manipulate it, but also to manipulate the structure ot the information as well as the algorithms. It requires – during development and when tailoring the system – to think one level of abstraction more. During the workshops about abstraction ad meta-modelling we looked at examples to understand how the implementation of 'thinking one level more' might look like. We recommend such workshops for any project team that is not experienced regarding design for tailorability.

Designing a tailoring interface

Beside the normal working interface providing an overview of the object of work and the ways to manipulate it, tailorable applications require an interface presenting the tailorable aspects of the application as well as t possibilities to manipulate them. On the other side the tailoring interface has to make sense from a professional point of view. We don't have any recipe to design such professionally meaningful abstraction and their representation. (Mørch & Mehendijew 2000) Give examples for multiple representations complementing each other and supporting the understanding of the application as well as the tailoring.
We recommend the development even of these interfaces in co-operation with the future users of the application. Mock-up techniques should be used to mediate the co-operation between users and developers especially on this more abstract level. (Ehn 1993)

Flexible user interfaces

If the tailorable aspects of the application effect the user interface, (part of) the user interface has to be implemented so that it can be dynamically generated. There are several techniques that can be used; the interface can be divided in simple building blocks that can be dynamically combined as needed according the meta description of the data to be presented and edited.
A more general technique is to define an internal language to describe an input form and to design a form editor that handles the user interaction. The filled in data is then send to the respective program part that handles it. In any case, automatically generated interfaces might influence the usability of the application as they can not take professional characteristics into account in the way a specifically designed interface can.

Architectural design

Architectural design and tailorability seems to be not yet related in a systematic way. As we only have the experience with one application we do not dare to claim any general conclusion. However, we stumbled over several ways to implement tailorability:

- Designing a language that steers system behaviour. Most tailorable workflow management systems (WFMS) deploy this technique. They have the advantage that the material that is moved around does not change in any way that is relevant for the WFMS. The way things are moved around is subject to manipulation. That can be described by means of formal languages that can be interpreted by the system. The tailoring interface directly manipulates the designed formal language. (Mørch & Mehendijew 2000) Filters in e-mail systems or macro languages use similar techniques on a smaller scale.
- 'Flat' representation of data structures plus meta information. If not only the system behaviour is subject to tailoring, but the structuring of the data respectively the data itself, the above approach does not work that easy. One solution is to separate the actual data and the description of how it is structured. The data itself can be represented as a 'flat' list of key value pairs. The information what kind of data is belonging to an object and how the object should be represented and treated has to be handled via an explicit description that can either be part of the object itself or a corresponding 'type' object. Access to data and manipulation requires then always double access: the meta-description of the data has to be checked in order to manipulate the data in a consistent way.
- Expanding the program. A radical approach to tailorability is promoted by (Mørch 1997). The user can, if he is competent, change and add to the implementation of the program. We started to explore with the help of two students writing their Bachelor Thesis the possibility to use a meta object protocol to implement the expansion of a program without requiring programming skills of the user.
- Component based programming and Product line architectures. The results of this research areas could be adapted if end user tailoring is not appropriate.

Persistence

SQL databases already allow certain changes of the data model during operation. If changes are required that go beyond, the description of the data structures as well as the data itself can be saved in the database. The access of the data requires then to first access the metadata. The technique and performance penalties are described in (Diestelkamp & Lundberg 1999).

7.4 Evaluating Design Solutions

'Design for change' can mean many things and flexibility carried through all the way from the user interface to the database design is only one solution. A well structured architecture allowing easy changes, a tailoring interface requiring the competence of a system administrator or a partly flexible solution as chosen for the contract and payment system under construction are other ways to provide for change. Deciding for one solution should be based on a systematic evaluation. Using change scenarios to estimate the work required to adapt the system, and the evaluation of a set of trade offs are the means we tried out to be able to compare the design of the prototypes and the design of the system under implementation.

Change scenarios

To construct relevant change scenarios, we used several sources:

- Historical changes, that took place in the application domain, that caused significant trouble regarding computer support or technical infrastructure.
- Anticipated changes, that have been discussed during the development and that have been taken into account for design
- Unanticipated changes, that pop up during development. In a quickly changing business practice that will be the case
- The system design itself; inventing changes that are especially easy or especially difficult to implement based on the design under discussion. The probability of such kind of changes can then be estimated by the domain experts.

The evaluation should be based on an architectural design of some kind. What kind of design representation is used depends on the implementation technique.

Trade offs

Beside the change scenarios we evaluated the designs regarding a set of trade offs. The following ones seem to be relevant to be taken into account when considering a flexible system:

- Implementation effort versus change effort and productivity gain. A flexible solution requires often a more thorough design and perhaps advanced implementation techniques. The tools used in the development organisation might not support a flexible design easily. Meta modelling techniques are more difficult to handle then straight forward modelling. Is this additional development effort worth the gain in time for maintenance and change and the gain in productivity for the users? 'In between' solutions (section 6.4) might provide a sound compromise.
- Performance versus flexibility. Meta modelling techniques often require additional database access and a higher complexity on the architectural level. How does that influence the runtime performance of the program? How important are speed and space for the application at hand?
- Usability versus flexibility. Does a flexible solution provide a user interface that is acceptable?
- Flexibility versus robustnes. If software is changed by programming, the modifications are usually deliberated before they are implemented and their effects on both the external and internal behaviour of the system are carefully planed and tested. With a meta-model system this changes; some types of modifications can be done without any programming, directly through some sort of administrator interface or tailoring interface. This provides the risk that changes are done without the proper deliberation. We also have a problem with the understanding of the system. The running system in normal mode is partly controlled by the meta-data and partly by the actual program code. The effects of tailoring might not be easily to estimate, and to control.

  For individual productivity toolkits, and for workflow management systems, the result of errors in tailoring will be fed back to the user before they get to trigger errors in other system. In a system like the one under consideration errors might be promoted quite rapidly, corrections might require cumbersome follow up in other systems.

7.5 Open Questions

Tailoring and architectural design

This seems still and unsystematised area. What kind of pattern solutions exist for tailorability? What are the possibilities and constraints of different ways to implement flexibility?

Flexibility and usability

The issue of professionally meaningful abstractions and their relation to the implementation of a tailoring interface and of flexible interfaces is still waiting to be explored.

Flexible flexibility

Can we provide a 'good enough' flexibility? The implementation technology for flexible persistence, as an example, requires to design the whole database of an application as 'meta data plus data'. Often only part of a data model has to be handled flexible. Can we apply the technology specifically for that part? Can we have a temporal flexible system, where we after the prototyping phase, when the requirements from the use context become more stable, (semi-) automatically to a normal database? Can we go back and forth a flexible and an inflexible implementation?

Flexibility and organisational context

Openness in the technical infrastructure allows for a greater openness regarding work and business practices. This openness in its turn requires a procedure to agree on, how to do what to do and to coordinate it. It is no longer the rigor computer application that can be blamed, when unwanted requests are put onto the use department.

Flexibility and software development methodology

We are still exploring what changes there are regarding project organisation, the relationship between original development, maintenance and operation, the relationship between design and implementation and so on.

Acknowledgements

to colleagues and to the students who participated with their exam and project work.

References

Bowers, J., G. Button & W. Sharrock (1995), Workflow From Within and Without, Proceedings of the 4th European Conference on CSCW '95, pp 51ff.

, S. (1999), Computer Applications as mediators of design and use – a developmental perspective. DAIMI PB-542, Computer Science Department, Århus University.

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

Christensen, M., A.Crabtree, C.H. Damm, K.M. Hansen, O. Lehrman Madsen, P. Marqvardsen, P. Morgensen, E. Sandvad, L. Sloth & M. Thomsen (1998), The MAD Experience: Multiperspective Application Development in evolutionary prototyping, Proceeding of the ECOOP 1998.

Diestelkamp, W., & L. Lundberg (1999), Promis, a Generic Product Information Database System. R. Y. Lee, (Ed.): Proceedings of the ISCA14th International Conference, Cancun Mexiko, April 7-9, 1999.

Dittrich, Y. (2000), Beg, Borrow, and Steal - but what? and what for? In: Beg, Borrow or Steal: Using Multidisciplinary Approaches in Empirical Software Research, workshop at the 22nd ICSE conference.

Dittrich, Y. (2002), Doing empirical research in Software Engineering – Finding a path between understanding, intervention and method development, *in* Y. Dittrich, C. Floyd, R. Klischewski, Social Thinking – Software Practice, The MIT Press, Cambridge, USA, to be published.

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

Ehn, P. (1993), Scandinavian Design: On Participation and Skill. In: Douglas Schuler, Aki Namioka (eds.): Participatory Design: Principles and Practices Hillsdale ,New Jersey 1993, pp 41ff.

Gantt, M., & B.A. Nardi (1992), Gardeners and gurus: Patterns of co-operation among CAD users. In: Proceedings of the CHI '92 , ACM 1992, 107-117

Henderson, A., & M. Kyng (1991) There is no place like Home: Continuing Design in Use. In J. Greenbaum & M. Kyng, Design at Work, Lawrence Erlbaum Associates, pp. 219-240.

Lehmann, M.M., (1980), Programs, Life Cycles, and Laws of Software Evolution. In: Proceedings of the IEEE 68 (1980), 1060-1076.

Lindeberg O. (2000) What are Software Practices Studies Good For? In: Beg, Borrow or Steal: Using Multidisciplinary Approaches in Empirical Software Research, workshop at the 22nd ICSE conference, 2000.

Lindeberg, O., & K. Rönkkö (2000), 'Bad Practice' or 'Bad Methods': Software Engineering and Ethnographic perspectives on software development, in Proceedings of the IRIS 23, University of Trollhättan/Udevalla.

Mørch, A.I. (1997) Evolving a Generic Application into a Domain-oriented Design Environment. Scandinavian Journal of Information Systems 1997.

Morch, A. & N. Mehandjiev (2000), Tailoring as Collaboration: The mediating Role of Multiple Representations and Application Units, Journal of Computer Supproted Cooperative Work

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

Robinson, M. (1993), Design for Unanticipated Use. in: DeMichaelis, G., Simone, C., and Schmidt, K. (Eds.): Proceedings of the Third European Conference on Computer Supported Cooperative Work, Milan, 1993.

Suchman, L., (1994), Working Relations of Technology Production and Use, Journal of Computer Supproted Cooperative Work 2: 21-39.

Trigg, R. & S. Bødker (1994), From Implementation to Design: Tailoring and the Emergence of Systematization in CSCW, Proceedings of the CSCW '94, ACM-Press, New Yorck, pp. 45-55.

Design for change
by Yvonne Dittrich, Olle Lindeberg, Ingela Ludvigsson, Lars Lundberg, Bengt Wessman, Wolfgang Diestelkamp, Marie Tillman

ISSN 1103-1581
ISRN BTH-RES--04/01--SE

Printed by Kaserntryckeriet AB, Karlskrona 2001