THE MENTOR PROJECT MODEL: A MODEL FOR EXPERIMENTAL DEVELOPMENT OF CONTRACT SOFTWARE

Kristine Stougaard Thomsen
Computer Science Department
Aarhus University, Denmark,
on leave from
Mentor Informatik A/S
Fredens Torv 6, DK-8000 Aarhus C

July 1992

Abstract

The Mentor project model supports experimental development of contract software. The application area is interactive information systems, i.e. systems closely integrated with user work practice.

The Mentor project model is a spiral model with iteration of activities such as (re-)design, estimation and negotiation, development and evaluation of prototypes. End-users are actively involved in design and evaluation. Repeated estimation and negotiation activities based on a calculation model for estimating system extent ensure that growth in extent is made visible and is subject to explicit decisions balancing use quality of the system with cost and schedule. The project model defines how to share the cost of experimentation between customer and supplier.

The project model is in current use and has been successfully used in several large projects during the last 6 years. This paper contains a synthesis of the practical experiences gained through these projects

and relates the project model to ongoing discussions of system development methodologies. The model contributes to the discussions by focusing on contract software and by giving elaborate suggestions for how to prototype large information systems with many users.

1 Introduction

In this paper, an *interactive information system* is considered to have the following characteristics. It is structured around a database that models a part of reality relevant for some administrative purpose of an organization. The system typically has several users who update and query the database as an integrated part of their daily work routines.

The term *supplier* is used to denote the system development organization that is responsible for designing and programming the system in cooperation with users. By *customer*, is meant the organization that orders the system and negotiates resources like time and price with the supplier. That is, it is assumed that there is a formal contact between supplier and customer concerning the development process and the product to be delivered. By *users*, is meant end-users, that are to use the system as a tool in their daily work. In many in-house system development projects, the same distinction between supplier, customer and users can be made, at least informally. In fact, there is a trend, at least in Denmark, towards formalizing these roles also in in-house system development.

Traditional waterfall models for system development, e.g. (Royce, 1970), consist of a document-driven, linear sequence of phases. Such models have proven insufficient for development of interactive information systems (Boehm, 1988). Project models that allow requirements to be gradually unveiled in an interaction of development and evaluation, are better suited. However, prototyping methodologies have shown to give other problems, including difficult project management and control, (Boehm e. al., 1984) and (Alavi, 1984). This results in difficulties using prototyping to large systems with many users, and in particular to contract software.

During the last 6 years I have been working in a Danish consulting company, Mentor Informatics Ltd., where we try to meet this challenge in practice. We have developed the Mentor project model which is a spiral model that combines experimental system development or evolutionary prototyping

with intensive use of user documentation and repeated negotiation. Negotiation is based on a calculation model that makes growth of system extent visible. The cost of experimentation is shared in a well-defined way between customer and supplierand. Explicit decisions are made on how to balance use quality, schedule and cost of the system. The model is therefore well suited for contract software. Moreover, the model includes a division of the system into subsystems and establishes autonomous working groups of users and system developers for each subsystem. This makes the model well suited for large systems with many users.

The Mentor project model has been successfully used in several large projects. The two most important projects were the development of an educational administration system (ESAS) and a financial system (ØSE) for the Danish trade schools. There are 115 such schools with a total of approximately 2000 users. The schools differ in size and are in some degree specialized for different trades. The customers in these projects were two different departments in the ministery of education. The project effort of the ESAS project was 220 person-months, and 60 users were actively involved in the development. At present, ESAS includes 239 screens and 143 reports. The project effort of the ØSE project was 115 person-months, and 40 users were actively involved. ØSE includes 105 screens, 45 reports and (approximately) 30 batchjobs. The platform and tools used in both projects were mini-computers running UNIX, C and ORACLE. The ESAS and ØSE projects will be used as examples throughout this paper.

This paper is a combination of can experience report and a contribution to the ongoing debate on system development methodologies. The Mentor project model is described in some detail in Section 2 to give an impression of the practical implications of experimental development of large systems with many users in a context of contract software. At the same time, the description is intended to be sufficiently abstract to be generally useful. Section 3 discusses experiences with use of the model. Section 4 contains a broad discussion of various project models and their relative advantages and disadvantages as described in literature. The Mentor project model is related to these ongoing discussions. Section 5 concludes the paper.

2 The Mentor Project Model

2.1 Division into Subsystems

The Mentor project model is based on a division of the system into subsystems. Each subsystem is treated as a separate sub-project, and explicit interfaces are designed between subsystems.

The division into subsystems is performed at an early stage of the development process. There is no well defined method for performing this division. However, we have found it useful to base the division on criteria given by the work practice of the users. That means defining a subsystem as a meaningful unit seen from a user's perspective, containing functionality related to a logical set of tasks in the work situation.

As an example of subsystem division, the ESAS system for educational administration was divided into 10 subsystems: Student administration, schema and absence registration, examiniation planning, community home administration, personnel administration, building and rooms administration, material, equipment and furniture administration, activity planning, data exchange (with other schools) and system administration.

Such a division enables focused user involvement in system units that are meaningful to the users. Moreover, it eases incremental implementation of the system in the use organization, e.g. by implementing one subsystem at a time.

Of course it is also important to minimize the dependencies between different subsystems, since explicit interfaces must be designed between subsystems. These interfaces should be kept as simple as possible, but not at the expense of a convenient division according to a user's perspective.

2.2 Involved Actors

In the project model, we have several working groups: the managerial user group, the specification groups and the design groups.

The *managerial user group* is a group of representative managers from different kinds of use organizations. Representatives from the customer organization also participate. The managerial user group must accept each specification and design of subsystems.

For each subsystem, a *specification group* is established. The specification group consists of selected managerial users and end-users, together with some system developers from the supplier organization. At least some of the developers will continue to be involved in the further development and programming of the subsystem.

For each subsystem, a *design group* is also established. The design group consists of selected people from the specification group, supplemented with more end-users, together with all the system developers that are involved in the development of the subsystem.

2.3 System and Subsystem Life Cycle

Each subsystem has its own time schedule and life cycle. Coordination has to take place between subsystems e.g. concerning design of interfaces between subsystems and concerning requirements for a subsystem originating from other subsystems.

Each subsystem has a general life cycle as illustrated in figure 1.

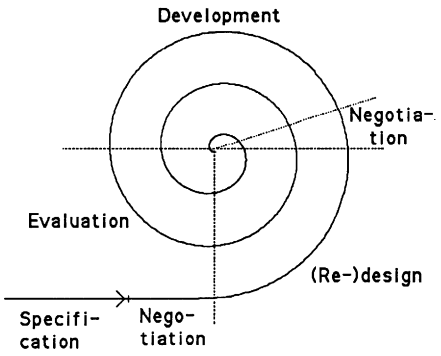

Figure 1: General subsystem lifecycle

Initially, the subsystem is specified, and time and price is negotiated based on the specification. Then a number of iterations on the following activities is carried through: (re-)design, renegotiation of time and price based on the (re-)design, development of (the next) version of the subsystem and evaluation of the subsystem through intensive user test.

The experimental process is controlled by limiting the number of iterations, and gradually reducing the amount of experimentation involved to ensure

convergence of the process and eventual accomplishment. Moreover, the extent of the system is closely monitored by means of re-estimation at well-defined points of time followed by renegotiation of elements of the contract between supplier and customer.

In the center of the spiral, the subsystem is implemented in the use organization and new spirals (not shown) of maintenance may be initiated.

The number of iterations in the spiral may vary from subsystem to subsystem and depends on the degree of uncertainty about the requirements for the subsystem. The more uncertainty, the more need for experimentation and thus the more iterations. In the extreme case of no uncertainty, the subsystem can be developed in one iteration, reducing the development model to a specification based waterfall approach without experimentation. The number of iterations is usually determined in advance, but may be changed as a result of negotiation during the process.

Our practical experiences have been based on up to four iterations, but the most common situation is three iterations, where the redesign in the third iteration is limited to corrections of errors and minor inconveniences. The unfolded spiral of our typical subsystem's lifecycle is shown in figure 2.

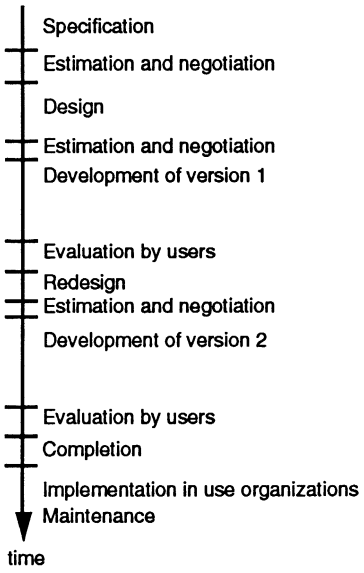

Figure 2: Example of unfolded subsystem

In the following, each activity is briefly described to give an impression of the practical implications of the model. All the estimation and negotiation activities are treated separately in Section 2.4.

Specification

The specification group makes a specification of the subsystem. The specification document is a draft user manual with sketches of screens and reports and their intended functionality. Some introducing sections will usually also be included, describing the relation between the different applications and between the applications and the tasks of the use organization. The specification remains open ended, containing questions and issues to be discussed in further detail at later stages of the development process.

The specification has to be accepted by the managerial user group.

Design of Version 1

After specification and negotiation, the design group takes over. This group produces a detailed design. Each screen and report is designed in detail, and background jobs are described. In addition, the underlying data model is designed. Finally, it is determined which system components should be included in the first version of the subsystem. The first version of the subsystem covers only the most necessary functions for a realistic evaluation, and only one (or few) of each typical kind of function. This usually means 60-80 % of the designed functionality.

The design document is a revision of the specification document, still shaped as a draft user manual, now supplemented with diagrams of the underlying data model.

The design document has to be accepted by the managerial user group, if there are substantial changes, compared to the intention of the specification.

Development of Version 1

Now, after a phase of re-negotiation, the system developers develop the first version of the subsystem. They keep in contact with the users in the design group during the development phase, at least by having a half-way-through meeting, where the whole design group evaluates the applications developed

so far. Some minor corrections are made immediately, and the meeting usually causes many small adjustments.

The development activity usually takes 2-4 months depending on the size of the subsystem.

Evaluation of Version 1

When the first version of the subsystem has been developed, it is installed in a test environment, and evaluated by users. The users in the design group are involved in the evaluation, together with new users, that have not been involved so far.

The purpose of the evaluation is to see if the design matches the use situation, and to discover inconveniences and errors. A number of evaluation reports are made by the users during the evaluation. The reports document errors and suggestions for change and extension.

The evaluation typically lasts one or two weeks, depending on the size of the subsystem. The customer and users are responsible for planning and coordinating the evaluation activity.

Redesign

After the evaluation, the design group meets and redesign the subsystem, based on the evaluation reports and on the experiences gained through the evaluation. The first version is considered an experiment, and it may be quite radically changed between the first and the second version of the subsystem.

General changes are incorporated not only in the system components present in version 1, but also into the screens and reports that were not included in the first version of the subsystem.

The redesign produces a new version of the design document. It may include new screens and reports compared to the previous version, or some screens or reports may have been removed or replaced by others.

Development of Version 2

After another phase of renegotiation, the system developers produce version 2 based on the redesign results. Version 2 also includes the system components omitted from version 1.

Evaluation of Version 2

Version 2 of the subsystem is evaluated by users in the same way as version 1.

Completion

The completion phase is a rudimentary third iteration of redesign, renegotiation, implementation and evaluation. The design group meets or otherwise agrees on what changes have to be made before the system is ready for implementation in the use organization. This time, suggestions for new functionality in the system will usually be postponed, unless they are crucial for the success of the system. That is, the revision is this time expected to be less radical, mainly being minor modifications of screens and reports and correction of errors. In this way the experimental character of the development process is reduced during the process to ensure accomplishment.

The changes to the subsystem are made, and a one day meeting is arranged for the design group to try out the system and check that all needed changes have been implemented.

Implementation in Use Organizations

Now, after a period of test in realistic settings and education of all users, the subsystem is ready to be introduced in the entire use organization. Either alone, if incremental introduction is wanted, or together with other subsystems.

2.4 Estimation and Negotiation Based on a Calculation Model

After each of the activities that may contribute with new or changed functionality of the subsystem, there is a short phase of estimation and negotiation.

The contract beween customer and supplier includes a calculation model or price list for calculating system extent, which is an estimate of the project effort of developing the system. The calculation model consists of a set of different complexity categories for the different kinds of system components,

with an associated extent measured in hours of development effort. The principle is shown in figure 3.

Complexity category	Extent (person hours)
Simple screen	a
Average screen	b
Complex screen	c
Simple report	d
Average report	e
Complex report	f
Batch job	g

Figure 3: Calculation model

These rules are combined with a set of guidelines for what distinguishes a simple, an average and a complex screen, etc.

The extent of a subsystem is obtained by considering each system component described in the specification or (re-)design document. Each component is categorized into one of the categories of the calculation model, and the extent of the system is calculated as the sum of the extents of all its components. The principle of such a calculation is illustrated in figure 4.

No. of system components	Complexity Category	Extent (person hours)
M	Simple screen	M*a
N	Average screen	N*b
P	Complex screen	P*c
Q	Simple report	Q*d
R	Average report	R*e
S	Complex report	S*f
T	Batch job	T*g
Total extent of subsystem		X

Figure 4: Extent calculation for a subsystem

The system extent calculated after the initial specification is usually used for the initial time and price negotiation between customer and supplier.

At later stages, the phases of estimation and negotiation serve to illustrate the consequences of user requirements to the system extent. A new estimate of system extent is calculated by reconsidering each system component and determining its complexity category. Old system components may have changed their complexity category, and new components may have been designed. Note that changes to a system component that do not affect its complexity category, leave the extent unchanged. That is, the extent measure includes room for experimentation within the given complexity category. The new estimate of system extent is compared to the previous estimate, and in case of difference between the two values, a negotiation between customer and supplier takes place.

The result of the negotiation may be that parts of the system's functionality is given low priority and postponed or cut down in order to keep price and schedule unchanged. Alternatively, it may be decided to develop the system as currently designed, and change the price to reflect the increased extent, usually by multiplying the extent with a price pr. hour. In this case the schedule may also need to be negotiated. Maybe the supplier can just add more person-resources to the development process and keep schedule. However, there will usually be a limit to the amount of ressources that can be added to a project without delaying rather than speeding it up, (Brooks, 1975). Therefore, our agreements with the customer usually determine a maximum extent that can be realized within a given schedule. When a system's extent exceeds this maximum, reduction of functionality is needed or schedule must be changed.

As an example, figure 5 illustrates the increase of the extent measure of six[1] of the subsystems during the development of ESAS, the educational administration system for the Danish trade schools. The different columns show the extent measure in person-hours at different stages in the development process. The rightmost column shows the growth of the subsystem as a percentage of the extent estimated titer specification.

The price of the system was negotiated between customer and supplier to reflect the growth, but the price was not a full "extent multiplied with price pr. hour". Because of the original fixed price contract, a compromise was made between the original price and the calculated extent. Schedule was also

[1]The ESAS project started as a fixed price project, but was changed after the first subsystems so that negotiation based on the extent measure took place after each phase that influenced system extent. Therefore, figure 5 only covers the last 6 subsystems.

adjusted because meeting the requirements of the users had higher priority than keeping schedule. Subsystem 1 (not shown in figure 5) was subject to the maximum adjustment of 6 months.

	Specification (person-hours)	Design (p.-hours)	Redesign (p.-hours)	Growth %
Subsystem 4	1032	1176	1284	24
Subsystem 5	2124	2808	3336	57
Subsystem 6	1152	1284	1284	12
Subsystem 7	900	1280	1352	50
Subsystem 8	1368	1536	1776	30
Subsystem 9	240	540	540	125
Total	6816	8624	9572	40

Figure 5: Increase of extent measure

In the ØSE project of developing a financial system to the Danish trade schools, price was calculated as the extent multiplied with a price pr. hour. The price pr. hour was determined in the beginning of the project. The tight schedule of the project, which was determined by a new law for financial management of trade schools, caused the extent measure to be used more for giving priorities and cutting down functionlity than for adjusting price and schedule. Perhaps as a consequence of a broad knowledge and accept of the schedule among the users, growth was actually much less than in the ESAS project, only $10 - 15$ %. The system was delivered on schedule.

2.5 Parallelism Between Subsystems

The project model encourages parallel or incremental development of different subsystems. The specification groups for the subsystems should have a significant overlap of participants, to ensure consistency, whereas the users involved in the different design groups should be particularly interested and skilled in the work routines affected by the specific subsystem. However, overlap between design groups may be advantagous provided no users are overburdened.

Usually, the development of one subsystem should be started before the others and brought beyond the evaluation of the first version, before detailed design of any other subsystem is started. This makes it possible to take advantage of the experiences from the ftist version of the first subsystem in the design of the following, and thus minimizes the risk of duplicating inadequate design. This is particularly important, if the user interface is subject to experimentation during the development process. Otherwise the final system may suffer from having an incoherent user interface.

Coordination between subsystems is needed in order to design good interfaces between subsystems. Part of the interfaces can be designed at an early stage, whereas others will be needed as the development of the different subsystems progresses. Requirements to a subsystem may also arise as a consequence of the design of another subsystem.

The total system lifecycle can be illustrated by figure 6, which is actually a part of the time schedule for the development of ØSE. Because of the tight schedule, many activities took place in parallel, although it would have been an advantage to gain experiences from evaluating the first subsystem before starting specification of the other subsystems.

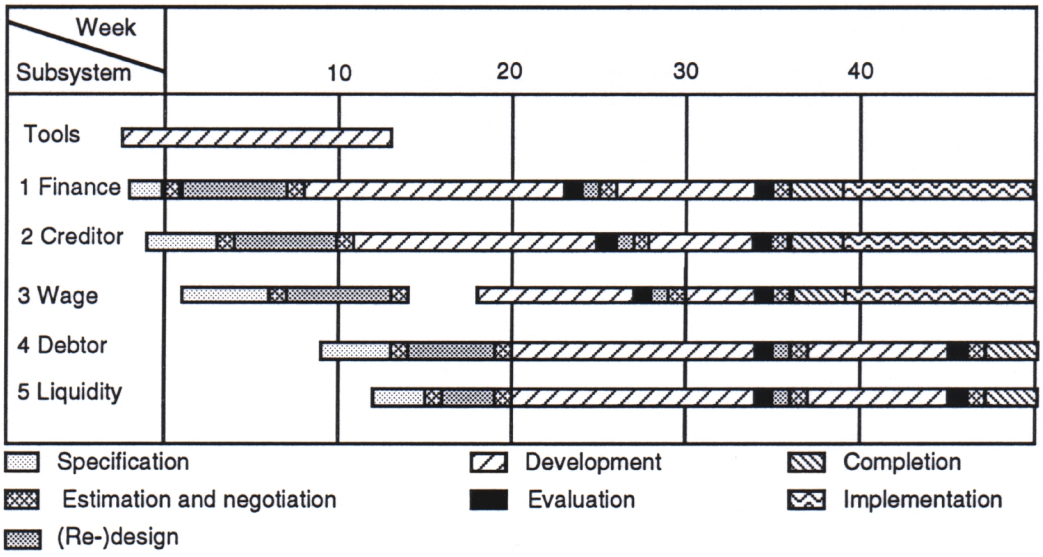

Figure 6: Incremental development of subsystems

2.6 Preconditions for Use of the Model

Two important preconditions for use of the model should be pointed out. The first concerns the development tools and software system architecture, the second concerns the contract between customer and supplier.

Appropriate Architecture and Tools are Needed

During an experimental system development process, both data model, functionality and user interface may undergo substantial changes. In order to make experimental system development possible in practice, these changes must be cheap to realize all the way through the process. This calls for a general software architecture that is independent of the actual data model, functionality and interface so that it can be stable during the process of experimentation. The architecture must also be highly modular in order to isolate each change to a single system component. Besides being a precondition for cheap changes, this is also a precondition for the development of a calculation model like the one described in Section 2.4, which is based solely on "visible" components like screens and reports, i.e. components that are directly meaningful to customers and users.

We have developed such an architecture and a production environment that we use together with the Mentor project model. The architecture is a modular client-server architecture which makes a division of the system into subsystems and an orthogonal division into layers. The layers span the gap between the presentation of the system to the user, and the underlying database system. Each layer encapsulates an important design decision as recommended by Parnas (1972) and Parnas et. al. (1985 a), including all dependencies on hardware and software platform. The architecture is described in detail in (Thomsen, 1992) and compared to the architecture obtained when using fourth generation languages, which are often used for prototyping information systems.

The Contract Must be a Process Contract

In order to make use of the project model, the contract between customer and supplier must be prepared for it. The contract should primarily be a contract about the process by which a system is to be obtained. Secondarily, the contract can contain requirements about minimum functionality and about

schedule. An important part of the contract is a calculation model that enables estimation of system extent and explicit negotiation, when system extent changes radically during the process.

The need for process contracts is also discussed by Grønbæk et. al. (1992).

3 Discussion of Experiences Using the Mentor Model

In this section our experiences of using the Mentor project model are discussed. Focus is on the major themes mentioned in the introduction: experimental development of contract software, prototyping for many users and prototyping of large systems. Each theme is discussed by means of a number of assertions and associated reflections.

3.1 Experimental Development of Contract Software

Customers are increasingly willing to make process contracts

It is our experience that as information systems are getting more and more integrated into the work practice in the work places, there is a growing understanding that product contracts and waterfall models for system development are inadequate. However, our experience is based on relatively few customers, and many will probably still hesitate to accept process contracts whose deliverables are not well specified in advance. Therefore it is important that a process contract contains a calculation model that defines how to calculate system extent. It is important to note and convince customers that design to cost is still possible, also in process contracts, since the extent measure can be actively used to keep cost and schedule stable.

Mentor Informatics Ltd. are in the process of establishing a quality management system meeting the ISO 9001 standard. It is our conviction, that process contracts and experimental development in the controlled form represented by the Mentor project model will be able to meet the requirements of ISO 9001. Although the formulations in the standards (see for instance ISO 9000-3) do not directly support experimental development, it is explicitly stated that the standard does not enforce use of a specific life cycle model. The primary purpose of the standard is to ensure that all steps

are planed, plans are documented and deviations from plans are documented and justified. According to our interpretation, the standard does not prevent contracts from being dynamic, changing gradually from process to product agreements as the development process proceeds. Neither does it prevent that requirements are developed gradually through experiments. Hopefully, we are right in our interpretation of the intentions of ISO 9001. Otherwise the standard could severely restrict the possibilities for improving software development in the future.

The Mentor model indentifies when system extent increases

The purpose of experimentation is to improve the use quality of the system. In order to control resources, it is necessary to distinguish between use quality, system extent and cost. Improvement in use quality does not necessarily affect system extent. However, experimentation often also leads to new requirements that make system extent increase. Identifying when this happens makes it possible to make explicit decisions on how to weight use quality, schedule and cost against each other. The project model uses the calculation model to make major changes in system extent visible and subject to explicit discussion and negotiation.

Of course this does not eliminate all sources of disagreement between the actors, but the negotiations can now be carried out in a much more direct and transparent way.

Among the problems that still remain is the categorization of system components. However, our experience shows, that it is possible, when there are only a few different categories.

The Mentor model defines how to share the cost of experimentation

The calculation model eliminates many of the tug-of-wars that supplier and customer usually have about whose fault it is that the system does not fulfil its purpose, does not meet the specification or is not delivered on schedule. When the price of the system is based on the extent measure obtained by the calculation model, responsibility for the cost of experimentation is shared between supplier and customer in a well-defined way: Experimentation within the original extent measure of the system is on the supplier, while experimentation that results in specification of new components (screens, reports etc.) or in components changing complexity category, is on the customer.

Our calculation model gives a realistic cost measure

Practical experience in using our software architecture and our calculation model was needed in order to define aproppriate extent measures for the complexity categories based on average experiences. Our extent measures for the different complexity categories were defined early in the ESAS project. They have been used unchanged in the ØSE project and seem to be stable.

In the project of developing ØSE, the financial system to the trade stools in Denmark, the actual project effort was registered as 17.233 person-hours when the system was first installed at all schools, whereas the extent measure for that version of the system was calculated to 16.410 person-hours. We find this relatively small deviation satisfactory and consider it a verification of the usefulness of the calculation model.

3.2 Prototyping for Many Users

Specification groups and design groups work well

As mentioned in the introduction, 60 users were actively involved in the ESAS project, and 40 in the ØSE project. This was of course only possible because the customer and use organizations gave high priority to active involvement of end-users. The schools were willing to pay for it by allowing the involved users to spend a substantial amount of time participating. The customer payed by giving economic compensation to the schools for letting the users participate.

Another important precondition for the successful involvement of so many users, was the division into subsystems, and the associated specification and design groups. These working groups were each sufficiently small to be efficiently working.

The specification and design groups were all relatively inhomogenous, representing both managers and users from the schools. Moreover, the trade schools are specialized for different trades and have varying size, which make their interests differ on many important questions. The specification and design groups are explicitly established to reflect these different interests. The groups were relatively autonomous and were responsible for success of the projects. As a result, they worked responsibly and democratically, although of course not without conflicts and compromises. In some cases, where dis-

agreements were too severe to find a suitable compromise, the result has been design and development of two different sets of functions in the system. For instance, the ESAS subsystem for schema and absence registration includes two different versions of functions for absence registration because of different work practices. Some schools register absence on a weekly basis, whereas others relate absence to individual lectures.

User motivation is high

It is our experience that the users are active and motivated in this kind of development process. There are probably several reasons for this. The close cooperation between system developers and users in the design groups means that the users involved have a direct and immediate influence, which stimulates interest and responsibility. At the same time it gives the users realistic expectations about the coming system, which is of great importance when implementing the system in the use organization. Other reports on prototyping approaches have shown similar effect on user motivation. See for instance (Grønbæk, 1989).

To some extent, we also attribute the high motivation to the division of the system into subsystems. The design groups are established with users who are allready interested in the domain associated with the subsystem. The division of labour and the specialization of knowledge at the workplace can be mirrored in the way different users are associated with design groups for different subsystems.

Finally, specification and design documents are shaped as draft user manuals. This ensures continuity in the way the properties of the system are communicated with the users, and makes it easier for them to relate what they see in the different versions of the system to what they have actually designed.

Few conflicts between users and developers

The expliciteness of discussion about use quality, extent and cost has some positive side effects on the cooperation between users and system developers in the design groups. It makes it clear to the users when to argue with the system developers that some feature should be added or changed, and when to argue with the customer organization that resources should be allocated to make some important but costly extension.

This is also a relief for the system developer working in the design group. It makes it possible to be more creative in the design process. A fixed price based solely on an initial specification, according to our experiences often results in a project stressed by sneaking growth, which moderates creativity at least on the part of the supplier.

3.3 Prototyping of Large Systems

Subsystem division gives managable units

The division into subsystems gives a number of relatively autonomous projects of more managable size. Interfaces between subsystems are explicitly designed and encapsulated into dedicated modules in the architecture. Subsystems can be developed in parallel or sequentially, depending on the dependencies between them and depending on schedule requirements.

Robustness and maintainability can be obtained

Some of the problems of prototyping pointed out by comparative studies of prototyping and waterfall models are robustness and maintainability. See for instance (Boehm et. al., 1984) and (Alavi, 1984). As mentioned in Section 2.6, we use a general modular architecture for information systems, described in (Thomsen, 1992). This architecture is independent of the actual data model, functionality and user interface of the system. The architecture is therefore not subject to experimentation during the development process, but is stable to the changes made to the data model, functionality and user interface.

The modular architecture ensures that most small changes in the requirements for the system imply only small changes in the system. Typical changes in requirements can be isolated to a single layer in a single subsystem. Usually even to a single module within the layer.

Later in the system's lifecycle, the modular architecture supports maintenance and portability. More details on the advantages of the architecture are given in (Thomsen 1992). Here it is also argued that the architecture is superior to the architecture obtainied when using fourth generation languages, both with respect to long term maintainability and with respect to the flexibility needed in order to support experimental system development.

4 Comparison With Other Project Models

In this section, the Mentor project model is related to other project models and to literature with critique and suggestions for existing project models.

The Waterfall Model

"The waterfall model" covers a whole class of project models developed during the 1970'es. Another term used for the same is "life cycle model". One of the most influential presentations of the model is given in (Royce, 1970). The characteristics of the waterfall model is its linear sequence of phases. Each phase takes a document produced by the previous phase as inpit, and produces another document as output. The sequence of documents range from initial requirements specifications over design documents at various levels, to documents and code representing the final system.

The waterfall model is well suited to handle the complexity inherent in many systems, and promote the development of a coherent and robust technical design. Moreover, since the specification at an early stage gives a complete description of the task at hand, the process of development can be monitored with respect to its use of resources and its progress. However, the model has its severe weaknesses.

Boehm (1988) characterizes the waterfall model as *document-driven* and argues that it does not work well for interactive end-user applications: *"Document-driven standards have pushed many projects to write elaborate specifications of poorly understood user interfaces and decision support functions, followed by the design and development of large quantities of unusable code."* (*Boehm, 1988, p. 63*). Boehm claims that the requirement that each document should be fully elaborated at some specific level of detail, obscures the high risk areas of system development.

McCracken and Jackson (1982) describe some important reasons for the insufficiency of the waterfall model. They claim that *"system requirements cannot ever be statedfilly in advance, not even in principle, because the user doesn't know them in advance"*. Moreover they stress that *"system development methodology must take into account that the user, and his or her needs and environment, change during the process." McCracken & Jackson, 1982, p. 31*).

The waterfall model has often been used for development of contract soft-

ware. Since the specification is made early in the process, it is often incomplete or even erroneous. Nevertheless, the specification often forms the only basis for estimating the whole project and establishing a contract between customer and supplier with a fixed price and a fixed time schedule. The consequence is, that during the development process, the customer and the supplier will both make efforts to interpret the original specification according to their interests. The supplier experiences being forced to deliver more than paid for, using more internal resources than originally anticipated. This results in internal management problems and in great efforts to limit losses. Losses can be limited by being rigid in the negotiations with both users and the customer and by taking technical short cuts. Clearly this means a system of poor quality both technically and with respect to usability.

It can be concluded that the waterfall model is only well suited in situations with low uncertainty about the problem to solve for the users. A theoretical and philosophical discussion of this theme can be found in (Dahlbom & Mathiassen, 1991, chapters 4 and 5). A more practical and pragmatic view is given in (Andersen et al., 1990, chapter 2).

The Mentor project model includes the waterfall model as a special case, when uncertainty is considered so low that only one iteration of the spiral is needed in the development process. The Mentor model is partially document-driven, and in this respect resembles the waterfall model. However, the Mentor model uses documents with open ended points and revises the documents in an experimental way based on the use of prototypes.

Evolutionary Prototyping

Floyd (1987) advocates a paradigm change from a product-oriented to a process-oriented perspective on software engineering. She uses the term product-oriented to cover waterfall models and characterizes the process-oriented view on software development as follows: *"The process oriented view relies on a cyclic model of (re-)design, (re-)implementation and (re-)evaluation, each cycle leading to a version of the somare system which can be evaluated in the context of work processes."* (Floyd, 1987, pp. 201-202). Emphasis is on establishing a framework for mutual learning between users and system developers. *"The desired functionality of the system will gradually become unveiled as a result of interleaved processes of development and use"* (Floyd, 1987, p. 201.)

Floyd (1984) gives a systematic presentation of different prototyping approaches. One of them is called evolutionary prototyping and is similar to the model described by McCracken & Jackson (1982). Evolutionary prototyping is characterized by vertical prototypes that evolve into a production system. That is, prototypes with selected fuctions implemented in full detail and based on the same platform and development tools as the target system.

Several comparative studies of the strengths and weaknesses of the waterfall model and prototyping exist. In (Davis et. al., 1988) the models are compared by means of a graphical representation that allows some interesting properties to be compared, all of which are related to the ability and ease of meeting user requirements. The comparison is intended as an aid in choosing an appropriate project model. However, it seems that the comparison is biased, since none of the benefits of the waterfall model are visible in the comparison, and none of the problems of prototyping.

(Boehm et. al. 84) contains a small empirical study, and Alavi (1984) presents the results of an analysis of 12 prototyping projects. Their studies show that benefits of prototyping compared to use of the waterfall model include higher level of user satisfaction and lower cost of systems. On the other hand, the studies also show some problems of prototyping: Project management and project control become more difficult, and the final system may be less robust and less maintainable due to insufficient overall technical design. The result could therefore easily be unforeseen expenses and a system with poor maintainability.

Boehm (1988) characterizes evolutionary prototyping as code-driven. He claims that *"It is generally difficult to distinguish it from the old code-and-fix model, whose spaghetti code and lack of planning were the initial motivation for the watevall model."* (Boehm, 1988, p. 63).

Moreover, it could be argued that a potential disadvantage of relying heavily on the work practice of users is that it may result in conservative systems that support status quo rather than being innovative. However, the risk associated with not being anchored in the existing work practice, when working for changes, is much larger. Without such anchoring, changes in work organization and changes in use of technology will fail to work in practice. This view is common in the Scandinavian tradition of system development. Further discussion can be found in (Greenbaum & Kyng, 1991, chapter 1).

In addition to the critique presented so far, I claim that the area of contract software seems to be a neglected area in the discussions of prototyping

methodologies. The difficulties revealed by the comparative studies suggest that matching prototyping to contract software is a non-trivial task.

The Mentor model resembles evolutionary prototyping in allowing requirements to be vague and incomplete initially. Requirements become gradually clarified through the development and evaluation of a number of vertical prototypes, evolving into a production system. On the other hand, the Mentor model supplements prototyping by requiring specification and design documents to be elaborated. Although not complete and final, these documents help planning and managing the process. In particular, the calculation model, which allows calculation of an estimated system extent based on a specification or design document, supports management of resources and makes the model suited for contract sofmare development.

Mixed Methodologies

Several suggestions for combining the waterfall model and prototyping exist.

Rapid throw-away prototyping in requirements specification as described by for instance Mason & Carey (1983) substitutes traditional requirements specification in the waterfall model by prototyping. However, since prototyping is restricted to the initial phase of the development project, the problem of change in requirements during the development process remains unsolved.

The contingency approach described by Burns & Dennis (1985) suggests that choice between a waterfall model and prototyping should be based on a judgement of two orthogonal properties of the system in question: complexity and uncertainty. Uncertainty is determined by three contingencies: the degree of structuredness, user task comprehension and developer task proficiency. Complexity is determined by four contingencies: project size, number of users, volume of new information and complexity of new information production. In case of low uncertainty and low complexity, prototyping is recommended. High complexity and low uncertainty should lead to choosing a waterfall model, while high uncertainty and low complexity should lead to choosing prototyping. Finally, if both complexity and uncertainty are high, a mixed methodology is recommended. No details of such a mixed methodology are given however, but the authors refer to rapid throw-away prototyping as an example of a mixed methodology. Moreover, they briefly sketch a mixed methodology which they fall “Phased design”. This method-

ology seems to have similarities with the Mentor model, but lacks details and seems to restrict subsystems to be developed in strict sequence. The authors state that prototyping for one user is not difficult; prototyping for many is. However, in their recommendation to use a mixed methodology in case of high uncertainty and high complexity, they do not elaborate further on how to prototype for many users.

The Spiral Model described by Boehm (1988) is a risk-driven iterative model. Depending on repeated risk analysis, iterations of the spiral are initiated to eliminate the main risk factors. A typical cycle of the spiral starts with identification of objectives, alternatives and constraints. In the next step risks are evaluated. The next step depends on the relative remaining risks and could for instance be prototyping or use of a waterfall model on some part of the system or the total system. Final step in the cycle is a review of all products produces by the cycle, including plans for the next cycle. Boehm includes in his paper a prioritized top-ten list of software risk items and suggestions for associated risk management techniques.

The Mentor project model resembles the Spiral model in its iterative and converging approach. The Mentor model is less general than the Spiral model, since it focuses on interactive information systems. This area is dominated by a few of the risk items identified by Boehm: Unrealistic schedules and budgets, development of wrong functionality or user interface, and continual stream of requirement changes. Management of these risks are built into the Mentor model, which can be characterized as a combination of a document-driven and a prototype-driven spiral model. Within its narrower application area, the Mentor model is much more elaborate than the Spiral model. It describes in detail *how* to use prototypes to eliminate risk, not only *when* to use them. Moreover, as pointed out by Boehm himself, the Spiral model does not match contract software in its present form. The Mentor model is particularly developed to match contract software.

Mathiassen & Stage (1990) criticizes both the contingency approach and the spiral model for focusing too much on when to use specifications or prototypes and giving too little advice on how to improve the use of specifications and prototypes in design of software. They suggest to focus on mode of operation (rational or experimental) separated from means of expression (specifications or prototypes). Traditionally, specifications have been associated with a rational mode of operation, whereas prototypes have been associated with an experimental. Mathiassen and Stage suggest that rational approaches

based on prototypes and experimental approaches based on specifications should be considered equally relevant. The Mentor model can be considered an attempt in this direction: Specifications are shaped as draft user manuals that are revised after experimentation with prototypes. Similarly, the model describes how to use prototyping in a rational way that allows planning and management. Use of specifications in an experimental way, faking a rational process, has also been suggested by Parnas & Clements (1985 b).

In summary, the Mentor model belongs to the category of mixed methodologies. We find that specifications and prototypes supplement each other as means of communicating between system developers and users. We also find that the combination offers an appropriate compromise between the need for experimentation and the need for planning and management.

5 Concluding Remarks

The Mentor project model for experimental development of contract software within the area of interactive information systems has been introduced. The project model describes a spiral of iterations of design, negotiation based on estimation, development of prototypes and evaluation of prototypes. It is a model for experimental development of large systems with many users. The users are actively involved in design and evaluation of prototypes. The model represents a mixed methodology, combining use of specifications and prototypes, and it has proven useful in practice.

In order to make use of the project model, the contract between customer and supplier must be prepared for it. The contract should primarily be a contract about the process by which a system is to be obtained. An important part of the contract is a calculation model that enables estimation of system extent and explicit negotiation, when system extent changes during the process. This calculation model should be based on experience with appropriate tools and software architecture.

Acknowledgements

I am grateful to all my collegues at Mentor who share with me the experiences that inspired this paper. Thanks are also due to Susanne Bødker, Joan

Greenbaum, Kaj Grønbæk, Morten Kyng, Lars Mathiassen, Peter A. Nielsen, and Randall Trigg who commented on earlier drafts.

This work was supported by The Danish Natural Science Research Council, grant no. 11-8385.

Bibliography

Alavi, M., (1984): "An Assessment of the Prototyping Approach to Information Systems Development", Computing Practices, Communications of the ACM, 27(6):556-563.

Andersen, N. E. et. al., (1990): "Professional Systems Development", Prentice Hall, Cambridge.

Boehm, B., (1988): "A Spiral Model of Software Development and Enhancement", COMPUTER, 21(5):61-72.

Boehm, B. W., T. E. Gray & T. Seewaldt , (1984): "Prototyping versus Specifying: a Multiproject Experiment", IEEE Transactions on Software Engineering, SE-10(3):290-303.

Brooks, F. P., (1975): "The Mythical Man Month - Essay on Software Engineering", Reading, MA: Addison-Wesley.

Burns, R. N. & A. R. Dennis, (1985): "Selecting the Appropriate Application Development Methodology", Data Base, Fall 1985, pp. 19-23.

Dahlbom, B. & L. Mathiassen, (1991): "Struggling with quality - The Philosophy of Developing Computer Systems", Department of Computer Science, Chalmers University of Technology and the University of Gøteborg, August 1991 (Draft to be revised and published in the fall of 1992).

Davis, A. M., E. H. Bersoff & E. R. Comer, (1988): "A Strategy for Compating Alternative Software Development Life Cycle Models", IEEE Transactions on Software Engineering, 14(10): 1453-1461.

Floyd, C., (1984): "A Systematic look at Prototypes", in: R. Budde et. al. (eds.): "Approaches to Prototyping", Proceedings of the Working Conference on Prototyping, Springer-Verlag, Berlin-Heidelberg-New York-Tokyo, pp. 1-18.

Floyd, C., (1987): "Outline of a Paradigm Change in Software Engineering", in: G. Bjerknes et. al. (eds): "Computers and Democracy", Avebury, pp. 191-210.

Greenbaum, J. & M. Kyng (eds.), (1991): "Design at Work", Lawrence Erlbaum.

Grønbæk, K., J. Grudin, S. Bødker & L. Bannon, (1992): "Improving Condition is for Cooperative System Design - shifting from a product to a process focus", in A. Namioka & D. Schuler (eds): "Participatory Design", Erlbaum Associates, Hillsdale N.Y.

Grønbæk , K, (1989): "Rapid Prototyping with Fourth Generation Systems: An Empirical Study", OFFICE: Technology and People, 5(2): 105-125.

ISO 9000-3: Quality management and quality assurance standards - Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software, ISO, Schweiz, 1991.

Mason, R. E. A. & T. T. Carey, (1983): "Prototyping Interactive Information Systems", Communications of the ACM, 26(5):347-354.

Mathiassen, L. & L. Stage, (1990): "Complexity and Uncertainty in Software Design", in: Proceedings of the COMPEURO 90 Conference held in Tel Aviv, Israel, May 7-9, 1990.

McCracken, D. D. & M. A. Jackson, (1982): "Life Cycle Concept Considered Harmful", ACM SIGSOFT, Software Engineering Notes, 7(2):29-32.

Parnas, D. L., (1972): "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, 15(12): 1053-1058.

Parnas, D. L., P. C. Clements & D. M. Weiss, (1985 a): "The Modular Structure of Complex Systems", IEEE Transactions on Software Engineering, SE-11(3): 259-266.

Parnas, D. L. & P. C. Clements, (1985 b): "A Rational Design Process: How and Why to Fake It", in H. Ehrig et. al. (eds.): "Formal Methods and Software Development", Lecture Notes in Computer Science, No. 186, Springer-Verlag, Berlin, pp. 80-100.

Royce, W.W., (1970): "Managing the Development of Large Software Systems: Concepts and Techniques", Proc. WESCON August 1970, pp. 1-9, or Proc. ICSE 9, Computer Society Press, 1987, pp. 328-338.

Thomsen, K. S., (1992): "A General Software Architecture for Information Systems", DAIMI PB 402, Computer Science Department, Aarhus University.