INDIVIDUELT STUDENTARBEID

IMT3101 Objektorientert systemutvikling

Jon Langseth

Table of contents

1. EL-Kontakt Vision

1.2. Positioning

1.2.1 Problem Statement

The problem of:	In a medium-to large electronics superstore, it is necessary to utilize multiple suppliers of goods and services. In such a service, access to information like contact numbers, addresses, user names, passwords and customer ID's may quite cumbersome, requiring personnel in an order-placement or service/support situation to utilize multiple sources of needed information, many of which are poorly maintained. Often the information is only known by a few persons, and in the absence of these persons, key information may be unavailable.
Affects	sales, management and service personnel of medium-to-large electronics superstores.
the impact of which is	reduced productivity, time spent handling orders, or service requests, is greatly increased by the need to search for correct information regarding the supplier of goods and/or services.
a successful solution would be	building an application that centralizes information, and thereby making information accessible and easily maintainable.

1.2.2 Product Position Statement

For	Elkjøp Gjøvik and Lillehammer
who	Needs a simpler system of maintaining a contact database, tailored for order-management and service/support work.
The El-Kontakt	is a standalone contact-database
that	is tailored for the needs of Elkjøp, and gives a single point of access through a simple GUI to needed information.
Unlike	current multiple, separate electronic and paper-based lists
our product	eliminates the search time needed to locate the needed information. Through our solution, all required information is located at the users' fingertips, without the need to support a myriad of standards.

1.3. Stakeholder Descriptions

1.3.1 Stakeholder Summary

Name	Description	Responsibilities
Elkjøp Gjøvik and Lillehammer	Customer, contractor	Contractor, and thereby supplies the market need, approves funding, monitors the projects progress. Assumes responsibility for assuring that the product meets the requirement presented.
Sales personnel of Elkjøp Gjøvik and Lillehammer	Users of system in an order-management and maintenance situation	Is responsible for providing real-world examples of needed information, developing and approving Use-case scenarios. Will be the end users of the product.
Service personnel of Elkjøp Gjøvik and Lillehammer	Users of system in when handling service requests and handling support	Is responsible for providing real-world examples of needed information, developing and approving Use-case scenarios. Will be the end users of the product.
Division managers and store management	Suppliers of information	The managers and management are responsible for providing the real-world data, both during development, and after deployment.

1.3.2 User Environment

Application will be used in conjunction with already existing sales/order management application (POS). The system will not with current needs have to be integrated into existing store-management/POS software, but should be able to be updated from formatted Microsoft® Excel™ documents. The expected usage will be short sessions, performed daily, as the user makes inquires regarding single orders/service request. Software platform is Microsoft® Windows™ NT and 2000 based, running on low-performance computers. Most systems have the Java2 runtime platform installed. The likelihood of this system-base being upgraded in the near future is negligible. The application will be required to be able to run on more than one computer at the time, and will be accessed by up to five users simultaneously. Database storage in the form of a centralized SQL server is not available, but there exists an operational LAN to which all workstations are connected.

1.4. Product Overview

1.4.1 Product Perspective

At the time being, there exists no proper competitive software. The closest competitor would be contact-management software like Microsoft® Outlook™, although these close-to competitors do not fulfil the need for simple access to multiple elements of the information. The system is to be regarded as stand-alone, as it will not be integrated into other solutions, nor will it provide interfaces for external systems.

1.4.2 Assumptions and Dependencies

The application will be developed for the available Java2 runtime platform, and targeted to the installed version of that software (1.4.0) on the Microsoft Windows NT operating system. The system will assume that the data-storage will be performed as files on a centralized, mountable network disk/share.

1.4.3 Needs and Features

Need	Priority	Features
Ease of use	High	The application must be simpler to use than current solutions, to be viable.
Responsiveness	Medium	The application needs to be able to present data, and respond to changes quickly, as the primary focus of the application is presenting stored information to user.
Completeness of information	High	The system must cater for all elements of information currently in use. Where there is ambiguousness in whether an informational element deserves a separate storage/view, alternate storage points must be presented.

1.4.4 Alternatives and Competition

The alternative would be to continue using the non-structured approach of multiple storage formats, non-standardisation and unsynchronized storage. This solution is routinely used, and the day-to-day users have developed the skill of quickly locating required information. The problem is the time needed to switch between systems, the lack of synchronisation between the different users lists, and the daunting task of updating information.

1.5. Other Product Requirements

The system must use a GUI, based on a list of entry’s, and a tab-separated area for segmented display of information.

A complete response from entry-selection in a list, to display and editability of the entry, must be performed in less than 0.5 seconds in 90% of all cases.

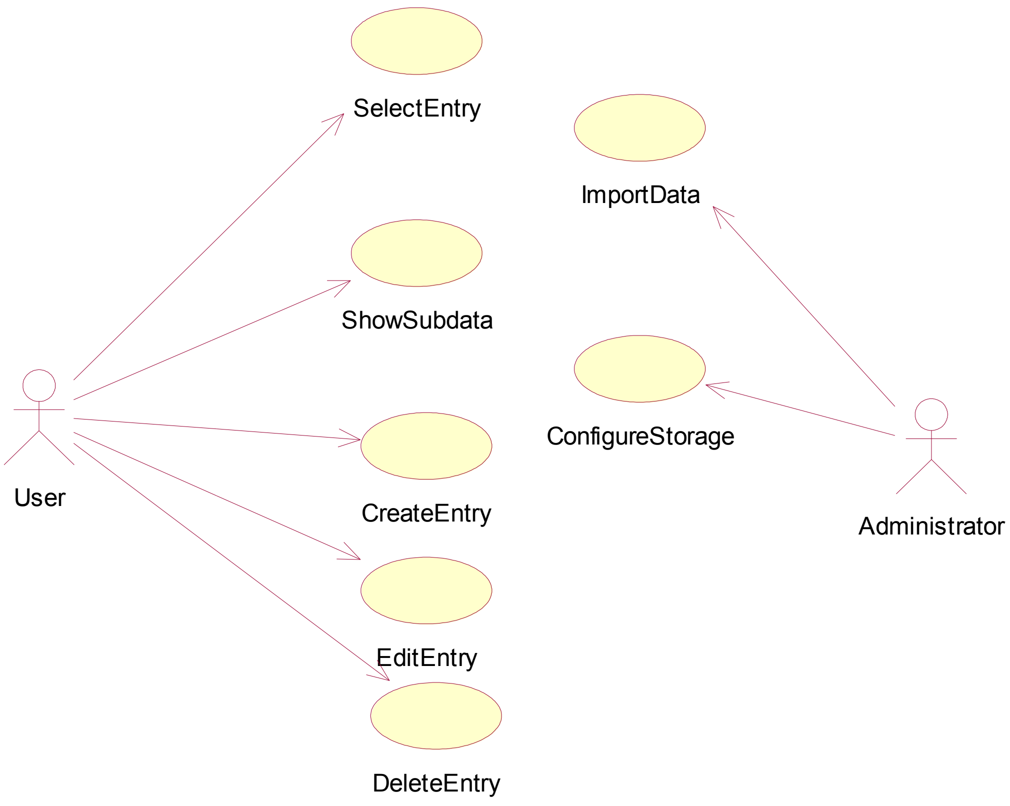
The creation of a new, blank entry, must be completed within 0.5 seconds from entering and accepting the entry’s name. The dialogue for entering the name (primary information) must be displayed within 1 second from requesting a new entry.

Switching between tabs must be performed in less than 1 second in all cases.

The system must store an updated entry immediately after storage is requested, or the entry is closed/hidden. The storage must be completed in less than 2 seconds. Storage must use a locking mechanism, to avoid simultaneous write operations.

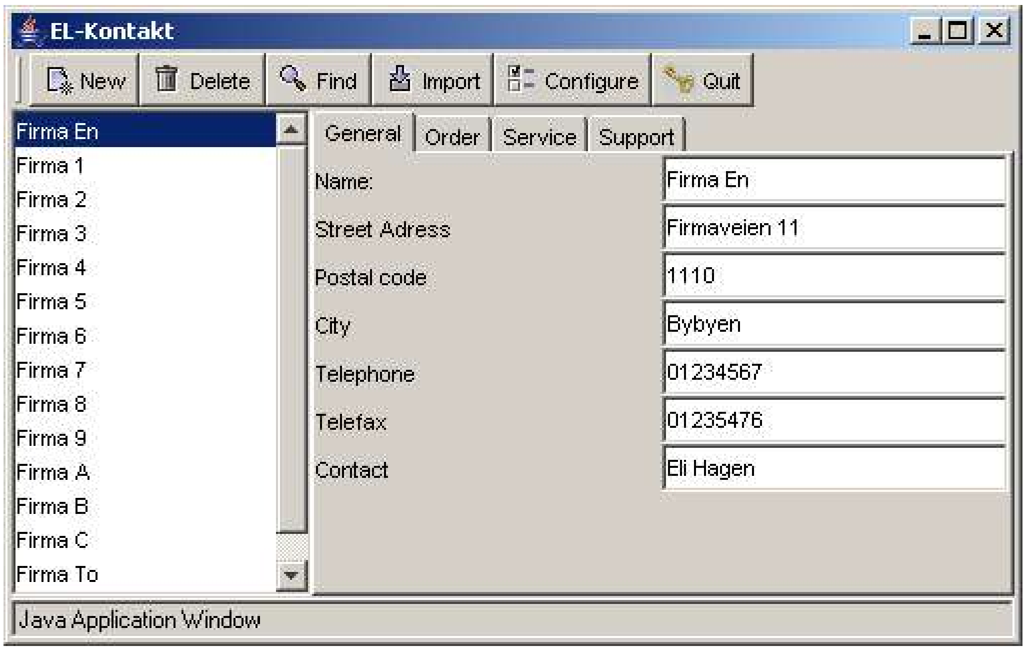
2. Development documents

2.1 Use Case diagram of EL-Kontakt



2.2 Mock-up of application.

To be seen as reference for all following Use Case descriptions.



Name:	Firma En
Street Adress	Firmaveien 11
Postal code	1110
City	Bybyen
Telephone	01234567
Telefax	01235476
Contact	Eli Hagen

2.3 High level Use Case descriptions.

2.3.1 Use Case ID: UC1

Name:	**SelectEntry**
Summary:	User clicks on a name in the list of names, and thereby selects it (see mock-up). This action updates the information in the main area of the application. The currently selected view/tab is updated, so that all fields contain the information associated with the selected name.

2.3.2 Use Case ID: UC2

Name:	**ShowSubdata**
Summary:	User can change between different sections of information. This is performed by clicking on one of the tabs on the top of the main window. This changes the selected view, by displaying relevant fields of information, and updating those fields with information stored about the currently selected name in the name list.

2.3.3 Use Case ID: UC3

Name:	**CreateEntry**
Summary:	User can create a new entry in the system by clicking on the “New” button in the tool bar. This will present a GUI-dialogue window, where the user will enter the name of the new entry. Selecting “OK” in this dialogue, will create a new entry in the name list, sort the list, and select the “General” tab of the main area. Most fields in this window will be blank, as no information is stored, except for the name itself.

2.3.4 Use Case ID: UC4

Name:	**EditEntry**
Summary:	With the currently selected entry displayed in the main area, the User edits data, by selecting the text-field of interest, and changes the data according to familiar GUI-usage. All fields on current view/tab is available for editing, except “Name”, which statically represents the name in the list-view. Storing is automatically performed when user changes view/tab, selects a different name in the name list, or one of the buttons on the tool bar.

2.3.5 Use Case ID: UC5

Name:	**DeleteEntry**
Summary:	The user can delete the currently selected entry completely, by selecting the “Delete” button on the tool bar. This will present a confirmation dialogue to the user, who clicks “OK” to accept, or “Cancel” to abort the deletion operation.

3 Discussion of appropriate methodology

3.1 Preface to discussion

The preceding vision document, use case diagram and descriptions totals as a real software engineering project. As a real, planned project, it makes a good basis for a discussion of potential applicable software engineering methodologies. The only factor that limits its applicability, is its size. The project is relatively small. The need for development staff also seems low, it should be possible to perform the task set, by little more than two developers. This limits the base for a discussion, as it seems that it will become apparent, it will be easy to eliminate some of the more heavyweight methodologies quite early.

It is important to make a note on exactly what this paper discusses, namely; what methodology to use. It seems natural to start that off with asking what is a software development methodology? The best suited, and perhaps most quoted definition, is the definition given to system development methodologies, by Avison and Fitzgerald (1995). A methodology is by this definition «a system of procedures, techniques, tools and documentation aids, usually based on some philosophical view, which help the system developers in their effort to implement a new information system». This is a far more precise, though to many, confusing definition than what you will find in a dictionary. As Cockburn (1999) quotes the American Miriam-Webster dictionary: «a series of related methods and techniques». Although this definition may be complete, it is important to remember that actual tools used, and the standard of documentation produced, is part of the methodology.

3.2 Classification

I am in other words trying to find a suitable toolbox and philosophy for developing the planned system. It seems to be a well established norm, that to select the correct methodology, one has to categorize the problem at hand. This notion is brought forward by both Avison & Taylor (1997), and Cockburn (1999), although the actual systems for classification differ. Avison & Taylor (1997) presents five classifications for system development projects, as follows.

1. Class 1. Well structured problem situations with a well-defined problem and clear requirements.
2. Class 2. Well structured problem situations with clear objectives, but uncertain user requirements.
3. Class 3. Unstructured problem situations with unclear objectives.
4. Class 4. Situations where there is a high user interaction with the system.
5. Class 5. Complex problem situations, combining two or more oh classes 1-4.

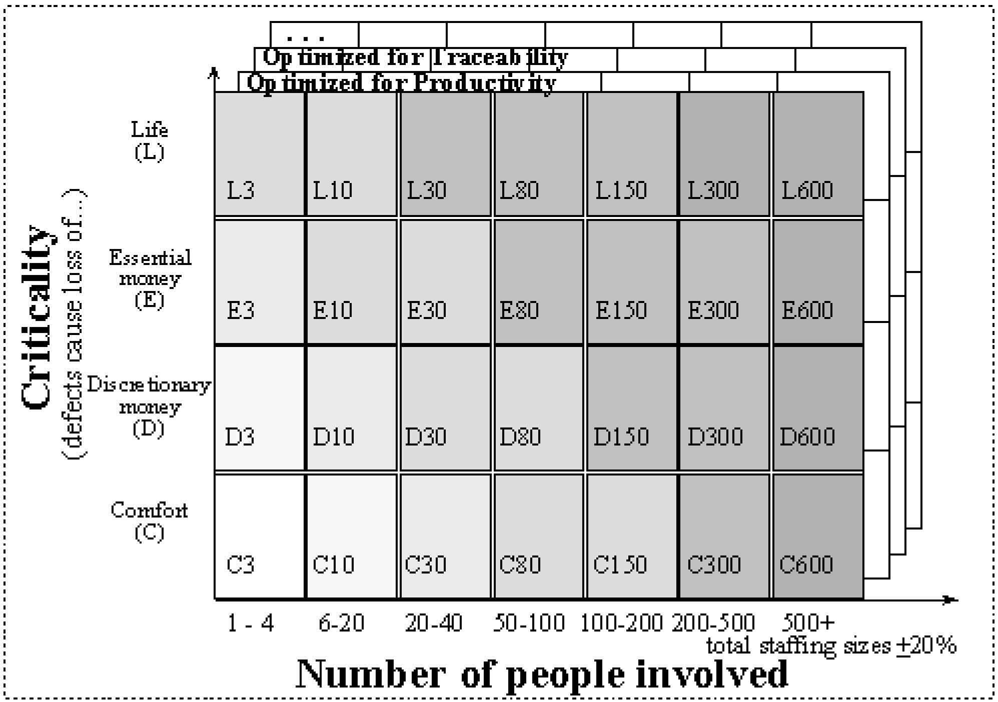
When trying to fit the problem at hand into this system of classification, I'll start by looking at the how structured the problem is. The objective for this system is quite clear, as the task to be completed, is simplifying access to, and centralizing location of currently available information. In essence, it is a question of computerizing a manual task. So it seems that structure alone would select class 1 or 2 of Avison & Taylor's system. Considering the requirements of the system, there are three major requirement areas that need to be satisfied. The actual information that is to be stored in the system, the simplicity of interaction with the user, and the requirements set forth by the software platform in use.

By performing a thorough examination of currently used information, and by doing standard human-computer interface analysis, the two first areas would be simple to identify, and pin down. As described by the vision document, the actual software platform is not likely to change soon. As a conclusion, not only are the objectives clear, but the requirements are also clear. This puts the project into class 1. There is still one factor that has not been considered, and that is the degree of user interaction. This is a system which is developed to increase the performance of the employees of Elkjøp, and to reduce time spend searching for information. Although the application will be used only for short a timespan every session, the degree of user interaction is high, and there must be a high focus on user interaction, usability, and performance. This, as a conclusion, will put the system in class 4.

What the use of the classification presented by Avison & Taylor actually means, will be discussed later. First, let us take a look at the classification system presented by Cockburn. He presents a matrix that divides problem classification into criticality on the Y-axis, and number of people involved on the X-axis. The criticality of a project is defined as what the severity a failure or defect of the system will be. The criticality is divided into four groups, representing projects where a failure will cause respectively, the loss of:

C: Comfort
D: Discretionary money
E: Essential money
L: Life

The X-axis, as mentioned, represents the number of people involved in the project. This axis is also divided into sections, where section separators are set at 4, 20, 40, 100, 200, 500, and 500+. The numbers naturally represents the number of people actively involved in the project, or estimated number of people to be involved. A margin should be taken, Cockburn suggests +/- 20%. To give an example of the use of this matrix, a system for managing a bank-to-bank transfer system, requiring a large range of features, would be assumed to require a development staff of 50 people. This would put the system in an E50 classification, it falls within the range 50-100, and failure will cause loss of essential money.



To utilize this classification, Cockburn talks about large, and dense methodologies. A large methodology, is one where many communication elements are in use, and a lot of focus is put on communication. A dense methodology has elements that tighten control, and thereby ensure correctness and robustness. As projects move along the X-axis, the size of the methodology changes. A project involving 5 people requires less control of communication than a project involving 150 people. As such, the latter project will require a larger methodology. As a project moves along the Y-axis, the change in criticality decides how dense the methodology needs to be. There needs to be more formality and control of documentation and standards with a project of Life-critical nature, than a project developing a new chat-client.

The EL-Kontakt application is one that requires few people involved. Considering only development staff, an assumption is made at 2 persons. Counting the number of people involved from Elkjøp, it would seem that more than 4-5 persons total would be unnecessary. As Elkjøp already perform the task this application is to fulfill manually, it would be wrong claiming that the failure of the application will cause loss of even discretionary money. The class this project falls into will be the one closest to the origin of the matrix, as a C2 project.

3.3 Possible methodologies

Now, what does all this classification actually lead to? What methodologies may be applied to the project specified? Defining the project according to Avison and Taylor, as a class 2 project, means that we should probably follow the recommendations given by Avison an Taylor. We should therefore choose a methodology based on data modeling, process modeling or prototyping. STRADIS, YSM, MerISE, SSADM, SDLC with prototyping, or pure prototyping are suggested as methodologies. I cannot claim to have much knowledge of any of these methodologies, except for SDLC, the systems development life cycle, and pure prototyping.

Defining the project into class 4, would call for the use of socio technical approaches. The prime example provided is ETHICS, Effective Technical and Human Implementation and Computer based Systems. ETHICS introduces a host of stages, with discussions and documentation at every stage, the need to develop a philosophy/ethic for the project, and a great deal of inclusion of the customer. ETHICS may be poorly perceived by the customers management. In such a small project as this, including all the aspects of ETHICS will be overkill, and having to reduce ETHICS actually removes the arguments to use it.

Avison and Taylor's text from Journal of Information Technology, 1997, is quite clearly focused on structured methodologies, and does not mention object oriented approaches. Also, the text was written prior to the establishment of the notion of “agile software development”. As such, none of the agile methodologies are mentioned by A&T. This is where Cockburn's system of classification assists. The fact that EL-Kontakt is a C2 project, gives the criteria that the methodology should be as lightweight and as loose as possible. When it comes to recommendations, Cockburn does not provide any. So we must look elsewhere, or rely on previous knowledge to find methodologies that fit this bill.

The first keyword that springs to mind in such context, is the word “agile”. The concept of agile methodologies, is an attempt at “a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff“ (Fowler, 2003). The idea of agile development is using a methodology that is as lightweight as the project allows. This is also why the term previously was lightweight methodologies when talking about this type of development. Basically, this idea of using the most agile methodology possible, is exactly what Cockburn talks about when he weights projects into the C, D, E and L groups, and adjust communication needs according to number of people involved.

The major names that formed the basis for the Agile Manifesto, and defining what agile development means, where Extreme Programming (XP), SCRUM, Feature Driven Development (FDD), Adaptive Software Development (ASD) and Crystal. Many others also exist in this group, and proof of this can easily be found on the web site of the Agile Alliance. In addition to the “real” agile methodologies, other more heavy ones may be adapted and in some cases reduced or simplified to become agile. It is impossible to examine every single one of all of the methodologies that fall into the group, either directly or through adaptation, and it does not make much sense for a project this small. Therefore I choose to look at, and evaluate only the ones I have at least vague knowledge of.

3.3.1 Extreme Programming

Extreme programming, or XP, is probably the approach that is best known of all the agile approaches. It is actually starting to get “old”, and as such is tried an tested on many projects. The base of XP is to do more communication, more real development, more testing, less paperwork and less administratively. Using a subset of the XP practices (up to twelve) as a toolbox to satisfy the four values of communication, feedback, simplicity and courage, this approach is highly evolutionary, as it views refactoring as one of the most significant elements. Little future planning is performed outside of the current iteration or cycle, the focus of planning is for the task at hand, and the testing cases for the current task. Although XP focuses on doing more of what developers think is fun, it still adds quite a bit of overhead into the process. Each bit of functionality must be developed according to a user story, previous work may need refactoring according to the new functionality, and there is a high need for customer and user interaction in the process. XP seems to me as a great way to go if the project is dynamic and the number of people involved is in the 5-50 range.

3.3.2 RUP/dX:

The Unified Process is absolutely an adaptable methodology, and even though it starts out as anything but agile, it is quite possible to strip the methodology down into lighter solutions. What RUP introduces that is most important to maintain is standardization of document formats, and sane suggestions onto progress and management of the software development process. The use of RUP phases and workflows, and the use of UML as the documentation language may be seen as the base of an agile RUP approach. Still, for a project to use a process that is completely RUP compliant, there needs to be some size or criticality in the project. It would make little sense to assign roles and create all artifacts used in a RUP project that involves little more than a short week of work.

dX is in my eyes a nice paradox. In many discussions about system and software development RUP and XP are displayed as competitors, and incompatible solutions. It is often claimed that a project can be defined as either a RUP project, or as an XP project, and that the selection normally is unambiguous. It is therefore very interesting to note that there exists an adaptation of RUP that actually is an implementation of XP. Or rather, a version of XP that is also RUP. The same flexibilities and limitations continue to be valid for dX as they are for the two parents. The methodology is agile and suitable for lightweight development, but still requires an amount of management that makes it potentially too heavy for a very lightweight project.

3.3.3 Feature Driven Development, FDD

The entire FDD process is driven by the need to satisfy the need of a set of features, or to give certain functionality. The process is divided into five sections: Develop an overall model, Develop feature list, Planning, Design by feature, Build by feature. The first three sections are linear, and form the basis for the actual development performed in the more iterative design and build by feature sections. By creating a solid model of the system up front, and identifying the needed features, this methodology allows for a great deal of focus on the actual implementation of the system. This is also a process that should be manageable for even a minuscule team of developers.

3.3.4 SCRUM

The Scrum process is based on an incremental waterfall model. The additions to this ancestry are most notably the notions of “sprint” and “scrum”. A sprint is a period of 30 days, in which the development team works on implementing a feature. The system requirements and functionality is documented in the initial phases of development, just as the introductory phases of the incremental development model. To the model, Scrum adds the requirement that the “backlog”, consisting of use-cases and requirement specs for a given feature, must be “expanded” by the team before entering a sprint. During the 30 day sprint, developers sit down 15 minutes every 24 hours, at a Scrum. This is a meeting, detailing what has been done recently, and what needs to be done shortly. The Scrum methodology also places focus on the importance of prototyping, and says that “useful product functionality is delivered every thirty days as requirements, architecture, and design emerge, even when using unstable technologies”. Scrum seems to be a methodology that should be manageable for a small team. What might limit the use of Scrum in a small project, is the fact that a sprint is 30 days. A lot of functionality in a small system can be developed during 30 days.

3.4 Selection of methodology for EL-Kontakt

From the presentation of classification and possible methodologies, it should be possible to make a suitable choice. As mentioned in the preface to the discussion, it is simple to exclude the heavy methodologies as options for the project at hand. Using heavy methodologies such as SSADM or ETHICS will not make sense for such a small project. Too much time will be spent planning, creating documentation and managing the information generated regarding the project development itself.

By being such a small project, methodologies like RUP, XP and dX can also easily be discarded on much of the same argumentation. As the actual requirements are largely clear, and both an object system model and required functionality can be easily deduced, it will be unnecessary to use effort administrating a large set of roles, artifacts an a library of documentation. The benefits of simple communication are to a great extent naturally present when working with a development team of less than four persons, and there i little to be gained by stressing communication, as RUP and XP do.

Still, the form of communication introduced by Scrum may be very valuable also for such a small team. The daily powwow, or scrum, makes sure that all development is in sync, and up to speed, when dividing tasks. But as mentioned, Scrum has a limitation in the length of the sprint. Though it is possible to reduce the length of a sprint, the benefits of using the sprint/scrum combination is lost when the sprint is shorter than a week. EL-Kontakt is an application that should be possible to realize in a very short time, with complete development from initial vision to finished product ought to be feasible within the span of a month. This gives the conclusion that Scrum is less than optimally suited.

This should narrow the selection down quite drastically. I have intentionally avoided mentioning the more structured methodologies suggested by Avison & Taylor. I have done this primarily because of the platform available and programming language of choice, in combination with the nature of the project. By choosing Java as platform an language, a strong focus is put on object orientation. Also, the nature of the tasks to be completed would be easily mapped into an object oriented viewpoint. It is therefore natural to avoid the structured methodologies, and focus on the object oriented ones. It must not be forgotten that a structured methodology may be altered to fit an object oriented approach, but trying to find a lightweight and agile methodology makes this form of adaptation undesirable, when more modern and directly object oriented methodologies exist.

Conclusion

I have three conclusions as to what methodology to choose for EL-Kontakt. The first, and least likely to be applied, is pure prototyping. This is a development method that can easily started, requires little administration, and the product is directly available for the customer to evaluate. But pure prototyping will most likely lead to making too many prototypes, an undesirable situation that makes the project drag along for a far longer time than necessary. The second, and in my view, the best choice, is FDD. Setting the system object model early should not be a problem, finding all required functionality should be a simple task, and by documenting this basis for development, all that is left is planning, and the actual development work. Using the FDD gives the project documentation, structure, and a plan to follow, without too much authority and methodology requirements. The FDD ought to be quite simple to understand, even with its wealth of planning documents and diagrams. These plan documents seem daunting at first look, but are basically tables telling what is to be done, and how complete is it.

The last of my conclusions is the one that is most probably going to be used. In this solution planning is disregarded, or performed ad-hoc, and the system design will never be influenced by strategic, long term decisions. Naturally what I am thinking of is not really counted as a development methodology, but the “code and fix” approach is very much practiced in projects of this magnitude and criticality. For small systems, the approach often works quite well. Even though I conclude that this is the approach that is most likely to be applied, it is by no means the one I recommend. It would naturally be of great value to both developers and the contractor, Elkjøp, to actually be able to see what should be, is, and will be performed, to have a plan, and a set of documentation. So my preferred choice for this very small project is the Feature Driven Development methodology.

Jon Langseth, 2004
References available by request.