Santtu Salminen

Teemu Vidgrén

MANAGING RAPID APPLICATION DEVELOPMENT RISKS WITH A GRAPHICAL MODELING METHOD

Information systems

Master of Science thesis

6.7.2001

University of Jyväskylä

Department of Computer Science and Information Systems

Jyväskylä

ABSTRACT

Managing Rapid Application Development Risks with a Graphical Modeling Method
Salminen, Santtu Harri Severi and Vidgrén, Teemu Olavi
Information systems
August 7th, 2000
University of Jyväskylä
123 p.
M.Sc. Thesis

The controllability of software project risks has always been an area of interest in software engineering. Software project risk management in general has been studied extensively, but the scope of the inspection has traditionally been on linear software development paradigms, for example waterfall approach.

With the emergence of software development environments and tools that allow information systems to be developed significantly faster than before, the rapid application development has gained a strong foothold among information systems development methodologies. However, the controllability of risks in rapid application development projects poses a new challenge. In this thesis, we study the effects of using a graphical modeling method and a CASE tool on rapid application development project risks.

By identifying risks emerging in typical rapid application development projects, a tool for managing these risks was developed. The tool consisted of *Notes Design Language* modeling method and *NotesEdit* CASE tool. The effects of the tool utilization on the identified risks was studied in two pilot projects.

Our study indicates that the use of the modeling method has an effect of reducing the risk exposure of those risk items, that are usually predominant in rapid application development. These risk items are mainly related to the scope and requirements and execution of the project. However, the rapid application development sets some special requirements for the CASE tool.

KEYWORDS: risk management, rapid application development, graphical modeling methods, metamodels

TIIVISTELMÄ

Managing Rapid Application Development Risks with a Graphical Modeling Method
Salminen, Santtu Harri Severi ja Vidgrén, Teemu Olavi
Tietojärjestelmätiede
1.12.2000
Jyväskylän yliopisto
123 s.
Tutkielma

Tietojärjestelmäprojektien riskien hallinta on yksi ohjelmistosuunnittelun tärkeimpiä osa-alueita. Riskien hallintaa tietojärjestelmäprojekteissa on tutkittu runsaasti, mutta useimpien tutkimusten painopiste on ollut perinteisissä tietojärjestelmien kehittämismalleissa, kuten vesiputousmallissa.

Tehokkaampien kehitysympäristöjen ja -työkalujen yleistyessä on tietojärjestelmien kehittämiseen kuluva aika lyhentynyt merkittävästi. Samalla on yleistynyt myös nopeutunutta sovelluskehitystä tukevan, *rapid application development (RAD)* – metodologian hyödyntäminen kehittämisprojekteissa. Nopeissa sovelluskehitysprojekteissa projektien hallittavuus asettaa uusia vaatimuksia myös riskien hallinnalle. Tässä tutkielmassa selvitämme graafisen mallinnusmenetelmän ja CASE –työkalun käytön vaikutuksia nopeiden sovelluskehitysprojektien yleisimpiin riskitekijöihin ja niiden hallittavuuteen.

Tunnistamalla tyypillisille nopeille sovelluskehitysprojekteille ominaiset riskit, kehitimme tutkimuksen aikana työkalun, jonka avulla pyrimme tukemaan RAD-metodologialla toteutettujen tietojärjestelmän kehittämisprojektien riskien hallintaa. Työkalu koostuu *Notes Design Language (NDL)* mallinnusmenetelmästä ja *NotesEdit* CASE –ohjelmistosta. Työkalun käytön vaikutuksia tunnistettuihin riskeihin tutkittiin kahden pilot-projektin avulla.

Tutkimuksemme osoittaa, että mallinnusmenetelmän ja sitä tukevan CASE-ohjelmiston käytöllä on riskialttiutta alentava vaikutus joihinkin nopeissa sovelluskehitysprojekteissa yleisesti esiintyviin riskitekijöihin. Nämä riskitekijät liittyvät teknisen dokumentaation laatuun sekä tietojärjestelmän määrittely- ja suunnitteluprosessiin ja näiden prosessien aikana tuotettavan dokumentaatioon. Nopea sovelluskehitysympäristö ja –metodologia asettavat kuitenkin joitakin erityisvaatimuksia käytettävälle CASE työkalulle.

AVAINSANAT: riskien hallinta, nopea sovelluskehitys, graafiset mallinnusmenetelmät, metamallit

1 INTRODUCTION

1.1 Background

Software engineering is pressed by an increasing demand for producing applications of better quality in less time and with smaller resources. The usability and characteristics of applications are emphasized while at the same time the necessary support for the software design process is often neglected. Furthermore, while new software development approaches such as *prototyping* and *rapid application development (RAD)* have dramatically decreased time needed to develop a software product, the actual design phase has suffered an inflation and the implementation phase easily gets the top priority (Budde, Kautz, Kuhlenkamp, and Züllighoven 1992). Even though the emphasis of software development lies on the finalized product, the development process and tools supporting it must be taken into appropriate consideration.

Software projects are often difficult to manage and too many of them end in failure. Nevertheless, it is claimed that the overall risk of a project can be significantly reduced by identifying and managing the individual risk factors threatening the project. Even with the continuous development of *risk management* approaches, a large percentage of software projects suffers from the realization of risk factors (Barki, Rivard, and Talbot 1993). By identifying these factors we can develop tools and methods to support risk intensive phases of the software project and thus decrease the possibility of project failure (Keil, Cule, Lyytinen, and Schmidt 1998). When successfully applying software engineering risk analysis and management techniques to projects, it is possible to increase the development productivity by 50 percent or more and produce systems with a better quality (Charette 1989). There are additional benefits from good risk analysis and management procedures, as listed by Charette (1989):

- Better and clearer perspective into risks, options, associated tradeoffs, and their effects and interactions between them.
- Consistent view of the problematic situation.
- Explicit identification of project assumptions and confidence that all information has been accounted for.
- Improved credibility of plans, rationale for actions made, inside and outside the organization.
- Better contingency planning, and a better selection of choices available to react to those risks that occur.
- More flexible assessment of the appropriate mix of ways of dealing with risk impacts, allowing for less reactive management, and more pro-active management.
- Better means to identify opportunities and ways to take advantage of them.
- Feedback into the design and planning process in terms of ways of preventing or avoiding risks.
- Feed-forward into the construction and operation of projects in ways of mitigating the impacts of risks that can arise, in the form of responsible selection and contingency planning.
- Decisions compatible with project policies, goals, and objectives ensured.
- Insight, knowledge, and confidence for better decision making, and overall reduction in project exposure to risk.

Our research concentrates on examining how the use of a *graphical modeling method* affects the identification and management of risk factors that become apparent especially in rapid, prototype-based software development projects. Traditionally, graphical modeling methods have been used to reduce risks in projects implemented using traditional development methods (e.g. waterfall approach). The evolutionary development methods (e.g. prototype approach) are also tools for managing these risks. However, as the evolutionary approaches help in managing some of the risks related to traditional approaches, they also bring forward some additional risk factors (Figure 1). We concentrate on studying the graphical modeling method utilization in RAD project risk management.

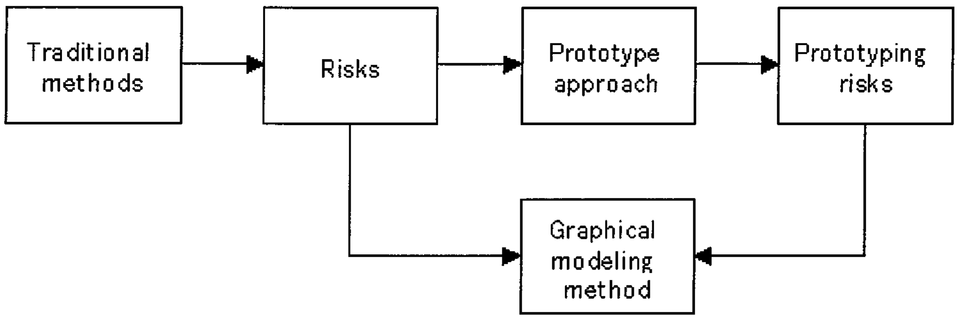

Figure 1: Graphical modeling methods in risk management

We use *Lotus Notes* as an example of a rapid application development environment since evolutionary development methods (e.g. Plan, Do, Check, Act by Larson-Hughes and Skalle, 1995) are typically utilized in Lotus Notes™ application development projects. As Lotus Notes is a visual development environment, the application prototypes and user-interface demonstrations can be quickly developed and the required changes can be implemented efficiently during the prototyping iterations. Additionally, Lotus Notes applications are typically developed for the use of groups of people and this necessitates the utilization of an evolutionary development method. The continuous communication between the developers and end-users can be considered essential for the success of a multi-user application development project (Grudin 1990).

By developing a graphical modeling method for modeling Lotus Notes applications we study how its implementation affects risk factors identified in rapid application development projects.

The main research question addressed in our research can be formulated as: *"How are risks in rapid application development projects affected by utilizing a graphical modeling method?"* This problem can be divided into following sub-problems:

1. What are the primary risk factors experienced during rapid application development projects?
2. Which of the primary risk factors can be controlled by implementing a graphical modeling method in RAD projects?

3. What is the most effective way to utilize a graphical modeling method in order to control risks in a RAD project?

Using previous research and the results from case projects, we create a graphical modeling method for supporting the risk management of RAD projects and a theoretical framework to analyze the constructed method in detail. By validating the technique, its effectiveness and functional integrity with a measurement system, we evaluate the method's usefulness in operational use and find out how the software project risks have been affected.

The methodological approach in this research is both constructive and empirical. At first, a modeling method for modeling Lotus Notes applications is developed and implemented in the MetaEdit+ environment. Then, the method is piloted in two Lotus Notes software development projects. The method is then evaluated with a developed theoretical framework based on related research and previous case project experiences.

Practical implications of this research are the following:

1. A graphical method for modeling Lotus Notes applications.
2. A checklist of risk factors that can be managed using a graphical modeling method.
3. A guideline for implementing the graphical modeling method in order to reduce software project risks when using a prototype approach.
4. Evaluation of the graphical modeling method usability in RAD project risk management.

The developed method is validated and evaluated within the scope of this research and further development suggestions and possible improvements to the method are considered. Additionally, the potential of an integrated code-generator and component repository are taken into consideration.

1.2 Basic concepts used in the thesis

In the following, we present a short summary of the key concepts that are relevant to our research.

A Software project is a process of developing a software product or products for a customer. *Software project risk management* is a discipline for identifying and controlling the risk factors threatening the successful completion of a software project. The social and technological environment in which the developed system is to be installed is referred to as *an object system.*

Rapid Application Development (RAD) is an approach for developing applications rapidly using high-level development tools (i.e. 4th generation programming languages and graphical development environments). Typically the RAD involves utilization of iterative development methods such as *prototyping.*

1.3 Outline of the thesis

Thesis proceeds as follows: In chapter 2 we discuss the area of information systems and software projects in general, and clarify the concepts of prototyping, rapid application development, groupware development, graphical modeling methods, method engineering, metamodeling, and CASE-tools.

In chapter 3 we introduce the concepts of software project risk and risk management, and present risk management strategies by Boehm and Ross (1989), Boehm (1991), and Charette (1989). The principles of risk categorization are also presented according to Keil et al. (1998).

Chapter 4 describes the case projects implemented prior to the modeling method development and the construction of the evaluation criteria using the case project experiences and the risk management principles and techniques put forward by Boehm (1989), Boehm and Ross (1989), Boehm (1991), and Keil et al. (1998). A prioritized list of general RAD project risk factors is presented to support the evaluation of the developed modeling method.

In chapter 5 the NDL modeling technique and NotesEdit tool are described and evaluated using the evaluation criteria described in chapter 4. The developed modeling method is evaluated regarding software project risks by comparing the risk assessment results of the two pilot projects to the risk assessment of the case projects implemented before NDL development. The pilot projects are described and the identified risk factors are analyzed, critical risk items are identified, and the effect of the modeling method is studied. The results are then formalized into a general checklist and guideline for risk management with a modeling method in prototyping projects.

Finally, in chapter 6, conclusions of the research are presented and the most important findings are summarized. In addition, further research areas are suggested and some unanswered topics are listed.

The research outline is described graphically in Figure 2.

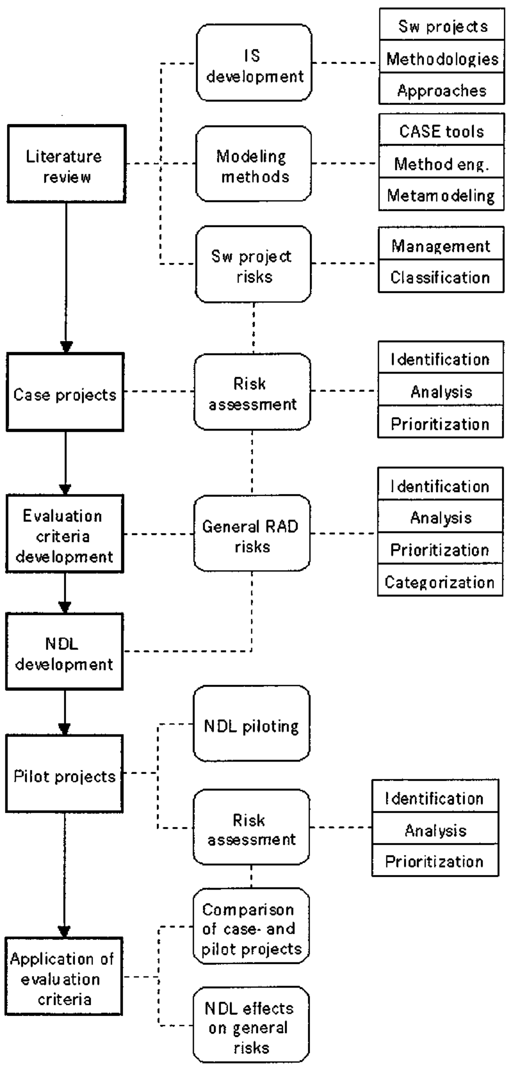

Figure 2: Research outline

The concepts presented in Figure 2 correspond to the chapters of thesis as follows:

- Literature review: Chapters 2 and 3.
- Case projects: Chapter 4.
- Evaluation criteria development: Chapter 4.
- Description of NotesEdit CASE tool: Chapter 4 (more information can be found in Appendix 1).
- Pilot projects: Chapter 5.
- Application of evaluation criteria: Chapter 5.

Summary of the research progress: The literature review was used as the basis for developing the evaluation criteria. Additionally three case projects were conducted to support RAD risk identification and analysis. The NDL modeling method was developed according to the experiences gained from the case projects and the literature. The NDL method and NotesEdit tool were then piloted in two pilot projects and the pilot project risk assessment results were compared to the case projects according to the evaluation criteria. Finally, the constructed method was evaluated and the findings were summarized.

2 INFORMATION SYSTEMS DEVELOPMENT

Software development is probably the most investigated subject in the area of information technology. The internal and external factors affecting the process have initiated a lot of discussion and studies. Software project risks with the traditional software development methodologies have been extensively studied. However, the field of software development is changing rapidly: software projects increase in both size and complexity, project costs are getting higher and available time and resources are scarce. In order to cope with the new requirements of software production, new techniques and tools have emerged.

The appearance of rapid application development (RAD) environments has made it possible to create relatively complex software in a fraction of time spent earlier. RAD environments encouraged the evolutionary approach to software development – prototyping and specifically, rapid prototyping.

While the use of RAD environments helped to increase effectiveness of software development, some insights have not been taken into widespread consideration. The risks associated with the new approaches and tools require a deeper understanding of the risk factors prevalent in rapid prototyping projects. These risks are further studied in chapter 3. In this chapter we attempt clarify the concepts of software project, different software development approaches, and the role of modeling methods in IS development. Additionally, the concepts of CASE tools, method engineering and metamodeling are presented.

2.1 Software projects

A *project* is a process that has predefined objectives that should be achieved within specific resources and time span. Cotterell and Hughes (1995) summarize the key characteristics of a project as follows:

1. Non-routine nature of the tasks involved
2. Planning is required
3. Specific objectives are to be met or specified product is to be created
4. The project has a predetermined time span
5. Work is carried out for someone other than yourself
6. Work involves several specialisms
7. Work is carried out in several phases
8. The resources that are available for use on the project are constrained
9. The project is large or complex

A Software project is a process of developing a software product or products by the developer for the customer. Typical software projects have most of the characteristics of a project concept in general. Therefore many techniques of general project management can be applied to software projects. However, software projects have also some characteristics which differ them from general projects.

A software project is a highly people-intensive effort that spans a very lengthy period, with fundamental implications on the work and performance of many different classes of people (Boehm and Ross 1989). The implementation of the developed software usually leads to major changes in organization, so the specification and development of software product is surrounded by a great deal of social and political tensions.

Another issue distinguishing software projects from traditional engineering projects is that the software products are not physical artifacts and the progress being made cannot necessarily be seen during the project. The estimation and control of a software development process and resources is much more difficult compared to physical artifacts and requires additional measures and progress indicators to be specified.

Software products can also be more complex than typical engineered artifacts. However, despite their complexity, the software products can be considered to be relatively flexible compared to physical products, due to the ease with which changes can be implemented during the development. This means that where the software system interfaces a physical or organizational system, it is expected to change if necessary to accommodate the other components. In many cases this leads to high degree of changes in software systems. (Cotterell and Hughes 1995)

To increase their controllability and measurability the software projects are usually divided into phases which each has pre-specified objectives and outcomes. According to Cotterell and Hughes (1995) individual software projects are likely to differ considerably but will typically have the following nine phases.

- **Project evaluation:** Investigation of project feasibility.
- **Planning:** Formulation of outline plan for the whole project and detailed one for the first stage.
- **Requirements analysis:** Finding out in detail what the users require of the system that the project is to implement.
- **Specification:** Creating detailed documentation of what the proposed system is to do.
- **Design:** Designing a system that meets the specification. The design phase includes designing both the user interface and the underlying functionality of the system.
- **Coding:** Coding the software may include writing code in procedural or object oriented languages, and using 4th generation development environments to support the user interface implementation. System may be built from scratch or tailored from a base package.
- **Verification and validation:** Testing the software to meet its requirements.
- **Implementation:** Installing the system to the production environment includes such things as setting up data files and system parameters, writing user manuals, and training users to operate the new system.
- **Maintenance and support:** Correcting and improving the system. In many environments, most software development is in fact maintenance.

Cotterell and Hughes (1995) emphasize the importance of the early phases of software development projects arguing that the key success factor in any project is having clear objectives. To be able to utilize the objectives effectively, the project staff needs practical tools and measures for testing whether or not the objectives have been met.

Because different stakeholders in a project are likely to have different objectives, a recognized overall project authority is needed. This authority usually comes in the form of a project manager. According to Boehm and Ross (1989) the project manager's primary problem is that the project needs to simultaneously satisfy a variety of parties. The users, the customers, the development team, the maintenance team and the management each have their own desires with respect to the software project. These conflicts are likely to cause problems when taken together and these conflicts are the root of most software project management difficulties both at the strategic and at the tactical level (Boehm and Ross 1989).

When specifying the project objectives and the specific requirements of the system with users, active communication between the development team and the target organization is necessary. Using semiformal specification techniques (e.g. graphical modeling methods) in determining and communicating the specifications among different interest groups can increase the understandability of the specifications and help in creating a sufficient common understanding of the system. Formal specifications can also be used as a basis for application models and technical documentation in later development phases.

Formal specifications and technical documentation also help in justifying the agreed system specifications if users are demanding additional functionality in later development phases. The cost effects of additional functions can be visualized in terms of changes needed to the initial design.

Boehm and Papaccio (1988) have studied the productivity of software development and identified typical cost factors that cause software project budget overruns. They state that the most significant individual factor affecting the software costs is the number of source instructions that must be programmed during the development. The strategies for reducing the amount of coding involve:

1. The use of 4th generation languages, or reusable components to reduce the number of source instructions developed
2. The use of prototyping and other requirements analysis techniques to ensure that unnecessary functions are not developed
3. The use of already developed software products

Other important factors influencing software costs are the selection, motivation, and management of the people, the complexity of the product, and the volatility of requirement specifications (Boehm and Papaccio 1988).

The product complexity management and requirement specifications can both be improved by utilizing formal specification and design methods to standardize the product specification and design practices (Aaen et al. 1992). Utilization of formal modeling methods also emphasize the role of the system design phase and helps the developers in identifying reusable components which can be utilized or developed during the project.

2.1.1 Software development methodologies

In the early ages of information systems development the software developers attempted to design and implement increasingly complex systems, often seeking to make innovative use of computers and information technology. There were no standardized procedures for developing software products. As a result, a significant percentage of these systems was unsuccessful in satisfying the end users' demands, was not timely delivered, or caused significant overruns in the pre-specified budgets and resource plans. The alarming situation of software development made it apparent that the major cause of the difficulties was the lack of a systematic approach to information systems development. (Wasserman 1980)

The *software development techniques* emerged due to the lack of standardization and controllability in software development. The techniques were designed to guide and standardize individual phases of the development process (e.g. design or coding). The techniques consist of a set of instructions and tools that guide the activities of a particular phase of a software project.

Although the individual steps of a project were supported by design techniques, the software development still lacked a framework for overall management of the separate development phases. The emergence of the concept of *software life cycle* made it possible to divide the whole software development project into successive phases and bring together all of the different techniques for software production with appropriate management techniques (Wasserman 1980).

When the individual development techniques were integrated to cover the whole software lifecycle, the sets of techniques appeared as *software development methodologies*. The methodologies were aimed to guide all the relevant activities associated with the development project and hence to introduce discipline for software development, and to standardize both the development process and the developed products.

According to Taylor and Wood-Harper (1996) the major benefits of using software development methodologies are:

1. Consistency
2. Completeness
3. Efficiency

By *consistency* Taylor and Wood-Harper (1996) refer to the consistency of the work practices and the produced artifacts. Using a common methodology, the different teams working on the same project are able to produce compatible components, and the work can be passed from one team to another for completion.

A proper information systems development methodology should ensure the *completeness* of the actions of the development team. Following the steps of the methodology should guarantee that all the necessary views of the system have been considered and that the design and implementation contain all the necessary functionality and attributes. (Taylor and Wood-Harper 1996)

Taylor and Wood-Harper (1996) suggest that if the information systems development methodology is chosen and implemented correctly it should lead to systems with appropriate quality being implemented in the shortest possible time scale. This *efficiency* is most likely missed by *ad hoc* development projects.

The diversifying nature of software projects makes it difficult to choose a suitable methodology for different situations. The methodology implemented successfully in one project is not necessarily the best one for another project with different objectives. The selection of an appropriate methodology is apparently crucial to the success of the project, but even today, there is no commonly accepted framework for supporting methodology selection.

2.1.2 Approaches for software development

Software development process can be examined from a multitude of views. The prevalent view since the beginning of information systems development has been that of the traditional approach. The traditional approach is characterized by the use of traditional methods, such as the waterfall approach. In traditional development methods the process of developing a software product is seen as a linear path from one step to another successive step. These steps are separated by certain goals or markers, which make it easy to evaluate the progress being made. Some traditional methods also contain feedback loops between stages, and guidelines to confine the feedback loops to successive stages to minimize the extensive rework involved in feedback across many stages (Boehm 1988). When all successive steps have been taken, a software product is considered to be ready for implementation.

The use of prototyping requires a different view to software development. Evolutionary approaches suggest that the software project is not a linear process consisting of successive development phases but a cyclic process including evolutionary iterations. The stages consist of expanding increments of an operational software product, with the directions of evolution being determined by operational experience (Boehm 1988). These process iterations are repeated until the software product is considered to achieve particular requirements set for it.

Evolutionary approaches are best suitable when using visual development environments, and the system prototypes can be developed in rapid succession. It is also well suited for situations where the end users are not capable to determine the system properties and functionality at the beginning of the development project. (Boehm 1988)

Evolutionary approaches can be considered as a risk management technique since they provide means for controlling many of the risks related to traditional development methods. As the future development directions are determined on each iteration, the project risks can be identified and analyzed actively, and appropriate risk management techniques can be applied correspondingly (Boehm 1988). However, the evolutionary approaches also bring forward additional risks. These risks are further discussed in the subsequent chapters.

The evolutionary approach and traditional approach can also be combined. For example, it is possible to set a traditional phase-division for a software project, but some steps within the traditional process are carried out in an evolutionary fashion using a cyclic model.

Several evolutionary development methodologies exist and they differ primarily by the focus of the methodology. Some examples of the evolutionary development techniques are prototyping, which is focused on the software requirement specification and the spiral model by Boehm (1988) which focuses on the risk management aspects of the software development. Within the range of our study, we concentrate on the prototype-based software development within the evolutionary approach and attempt to isolate the factors that emerge especially in RAD projects.

2.1.3 Prototype approach

A basic problem often encountered with software development project, is the difficult process of requirement specification analysis. There are two major directions in approaching this problem: formalization of activities so that the requirements may be identified by formal means, and the use of experimental methods to assist the collection of necessary information (Budde et al. 1992). One widespread method belonging to the latter category is *prototyping*.

The errors in software requirement specification may lead to significant additional costs in later development phases (Boehm 1991). This brings forward the need for an iterative feedback mechanism (Budde et al. 1992). Prototyping is a common approach in software development when specification requirements are unclear or amorphous, when an application needs to be developed rapidly, or when we need to clarify the problematic nature of the target system.

Krief (1996) defines *software prototyping* as:

> "... a process of building software whose purpose is to attain information about the adequacy and the value of the software design. The prototype is normally used as the precursor to the final software. A prototype differs from the final product in that it is developed more quickly and it is more easily parameterized and manipulated, at the expense of efficiency and performance. The prototype is therefore used to quickly extract information about the software to come."

According to Krief's definition of prototyping, the main role of prototype is to provide the developers with the relevant information needed in order to construct the finalized software product.

2.1.4 The benefits and weaknesses of prototype approach

Prototype approach allows new ideas and concepts to be quickly incorporated into software, without a cumbersome requirements freezing process. Budde et al. (1992) suggest the following beneficial factors of utilizing prototype approach in software development:

- Prototyping provides a communication basis for discussions among all the groups involved in the development process, especially between users and developers.
- Prototyping improves coordination between users and the developers
- Prototyping reduces inhibitions on the part of the users and replaces unrealistic expectations by a more realistic view
- Prototyping enhances the sense of commitment on the user's part and increases motivation to cooperate with the developers
- Prototyping enables us to adopt an approach to software construction based on experiment and experience.
- Prototyping enables us to clarify problems by experimentation.

Budde et al. (1992) also point out some software development situations where prototyping is not an appropriate solution for a specific type of target system

- Systems requiring extensive data protection and recovery facilities
- Systems with very large number of transactions
- Systems that are subject to auditing
- Systems requiring professional maintenance
- Systems that are highly resource-intensive in operation

The use of prototyping sets some requirements for the users due to the required high degree of participation. Budde et al. (1992) sum up the three main requisites:

- Need of a basic knowledge of computers in order to be able to understand the effects of computer use on their own work activities
- Necessary skills enabling them to give an abstract representation of their own work activities as part of requirements analysis
- Need of additional knowledge in order to be able to use problem-oriented languages to support their own work with self-programmed applications

Krief (1996) points out that research in knowledge representation and problem formulation has shown that finding a proper representation of a problem is essential for its solution. By representing a problem graphically, for example with a graphical model, proper resolutions can be taken in order to solve that problem.

However, graphical modeling methods and supporting tools are rarely utilized in projects based on prototype approach. As the prototype itself serves as a model of the target system, the graphical representation of the system is typically considered unnecessary. Additionally, the prototype approach usually requires several modifications to be implemented to the target system during the development. These modifications are also to be implemented in the graphical models to keep the documentation up-to-date. The continuous updating of the models is typically considered time-consuming compared to the benefits acquired from the documentation during the project.

According to Budde et al. (1992) it is difficult to abandon traditional software development strategies (e.g. waterfall approach to software development), because they provide means of formal control on the produced documents and milestones. Graphical modeling methods are supposed to introduce discipline in software development and increase the controllability of the projects (Aaen et al. 1992). Since the graphical modeling methods are typically implemented only in projects based on traditional software development strategies, it is important to study their benefits in projects based on evolutionary approaches.

2.1.5 Prototyping methods

Basically, there are two types of prototypes: S*oftware prototypes* that are used as a functional frame for the final application, and *prototypes* that are used only as a model for the finalized software product. The first approach requires a *transformational* phase where the prototype is transformed in successive steps to a fully functional, efficient program. In the latter case, the prototype is discarded at the end, and a new application being constructed by consulting the developed prototype. The latter method also requires that a functional software product is to be built based on the experiences, feedback and test results from the *simulation* prototype. (Krief 1996)

Budde et al. (1992) present a categorization of three main types of prototypes: *a prototype proper*, *a breadboard* and *a pilot system.* A *prototype proper* is a prototype designed to illustrate specific aspects of the user interface or some part of the functionality of the target system. This type of a prototype is then used to develop the final target system. *Breadboard* prototype is mainly used by the developers to help them clarify construction related questions for the developer team. The constructed prototype can be used as a comparison standpoint when developing the final system. *Pilot system* is a prototype which is employed in the application area itself. Clearly, the level of technical design must be much higher than in the other types of prototypes.

Budde et al. (1992) also categorize prototyping approach according to the goals set to prototyping and the dimensions of the target system covered with the prototype. There are three types of goals for prototyping: *exploratory prototyping* is used when the problem is unclear. In this type of approach the initial ideas are used as a basis for clarifying user and management requirements with respect to the future system. The second prototyping method is *experimental prototyping* where the focus is on the technical implementation. Here communication about software functionality and usability between end users and developers is important. The last category of prototyping is *evolutionary prototyping* which emphasizes the continuing process for adapting an application system to rapidly changing organizational constraints. In this approach, the developers are in a role of technical consultants working in close cooperation with the users to improve the application system.

In addition to the categorization criteria presented above, Budde et al. (1992) present how to divide prototyping into subcategories according to the approach that is taken to the implementation of the prototype system. In *horizontal prototyping* specific levels of the target system are built – usually this means the user-interface. In *vertical prototyping* a selected part of the target system is implemented completely, through all vertical layers. This method of prototyping is usually used when developing pilot systems. It is also possible to divide these approaches into *a functional approach* and *an interactive approach. A Functional approach* means concentrating on the functional components of the target system which is useful especially when the functionality is uncertain. Different functional prototypes can be embedded into simple interactive shells. The emphasis of the *interactive approach* is on customizing the interactive components of the target system according to the user feedback. This kind of approach is useful only when the application functionality is well known or prescribed beforehand. It should be remembered, however, that designing interactive components is impossible without a proper knowledge of the underlying functionality.

Krief (1996) sees prototyping as an interactive and iterative decision-making process between the *expert* and the *developer.* By the term expert Krief (1996) refers to the expert of the application domain as the term developer refers to the personnel developing the application.

First, the expert defines the specifications and requirements for the system together with the developer. Based on the specifications, the developer creates the first prototype, which is then evaluated by the expert according to the requirements. If there are deficiencies or errors in the prototype, the developer creates a new version of the prototype, which is again evaluated by the expert. This iterative process continues until the prototype is ready – either as a model for the future information system or as a functional frame for the upcoming application.

In this research we study the effects of a graphical modeling method implementation on risks of software development projects, where the prototype approach is used for exploratory purposes (e.g. the prototype is transformed to a software product through transformational phases and the objective of prototyping is to clarify the system requirements).

2.1.6 Rapid Application Development (RAD) projects

The emergence of very high level programming languages and 4th generation languages provided the tools for developing applications much faster than it was previously possible. Through the advent of graphical, user-interface centric integrated development environments (IDEs) and the rise of component-based software development, the time of laborious programming of basic system functionality was dramatically lowered. However, this, allowed the developers to engage in unsystematic, *ad hoc* software development. When applying rapid prototyping, it is necessary to consider technical product quality and division of labor in the construction process, and not to give way to purely trial-and-error approach to software development (Budde et al. 1992).

The development environment used in rapid prototyping must allow the developer to quickly change specifications, implement changes rapidly, improve functionality of the prototype, and therefore produce different working versions of the prototype quickly. (Krief 1996)

In rapid application development, creating and utilizing reusable software components is emphasized. If the application utilizes reusable components, it is easier and faster to develop more robust and reliable, and more flexible and easier to adapt to evolving requirements (Nierstrasz et al. 1992). All this requires that the rapid application development environment supports the development and utilization of reusable components. The reuse of existing components during the early phases of development process also increases the likelihood of reuse of later products developed from it (Barnes and Bollinger 1991).

However, the development of reusable components in RAD projects is considered difficult since the applications are developed rapidly without specific design and modularization phases. The lack of comprehensive design also complicates the utilization of reusable components in the development phase. Additionally, the lack of design and modularization usually leads to problems in software maintenance and updating activities. The problems in searching and identifying the correct components for individual development problem are also complicating the reuse of existing software components.

2.1.7 Developing groupware applications with Lotus Notes™

Lotus Notes is a distributed client/server platform for developing and deploying groupware solutions. These solutions allow teams to share information electronically across network. Lotus Notes also supports mobile computing by allowing the usage of information even if only occasional connection to network is available. The focus of Lotus Notes is the capture and communication of the unstructured information that supports and flows within the work process. (Larson-Hughes and Skalle 1995) Generally, applications that are designed to support groups are referred to as *groupware* applications.

Typical Lotus Notes applications include shared document databases that can contain different types of documents produced and shared in a particular organization, team, or project (e.g. memos, agendas, and minutes) and indexes utilized for different purposes (e.g. customer and product indexes). In the late 90's the Lotus Notes applications have also begun to emerge in Internet and extranet solutions due to the Lotus Domino and WWW-technology breakouts. Further information about the Lotus Notes design components and application development in Lotus Notes environment can be found e.g. in Larson-Hughes and Skalle (1995).

Before we start analyzing rapid application development projects implemented using Lotus Notes, we must first specify the reasons why rapid application development is suited when developing software in Lotus Notes environment. Larson-Hughes and

Skalle (1995) present three reasons, why Lotus Notes is suitable environment for rapid application development:

- Since Lotus Notes is a high-level platform for applications, it handles much of the developer's work automatically (for example, network communication, mail requirements and security issues)
- Lotus Notes does not require the developers to know traditional programming languages. Most functionality can be constructed with a simple macro-language
- The results are instant - there is no compiling necessary. When changes are made to the application, it can be used immediately or even simultaneously.

All these features characterize Lotus Notes as a rapid application development environment. Larson-Hughes and Skalle (1995) suggest that evolutionary approach is the most useful approach for Lotus Notes application development. They recommend a PDCA (Plan, Do, Check, Act) -based rapid prototype method suitable for developing applications for the changing requirements of the business processes and the environment. The PDCA-method is a closed-loop problem-solving methodology created at early 1930's by Bell Telephone Laboratories.

In early 1990's typical Lotus Notes applications were quite small document management products built by utilizing basic Lotus Notes functionality. Due to the evolution of Lotus Notes technology and the popularity of Lotus Notes as an organizational Intranet solution, the applications have grown tremendously both in size and complexity. Applications have also a much wider variety of purposes than before; reservation, material management, and inventory systems, as well as other, traditionally RDBMS (Relational DataBase Management Systems) based systems are now feasible to construct by using Lotus Notes / Domino –architecture.

While the development approach has remained the same as before, several problems emerge in software development and maintenance. Poor design of the software and deficiencies in the design documentation lead to problems in system maintenance activities, development and utilization of reusable components, difficulties in dividing the development activities for different teams, and implementation of inefficient design solutions. This necessitates the use of supporting tools in software development projects.

Problems related to information systems specification in general also apply to Lotus Notes application development. The end users are rarely able to specify all the system properties and functions in the beginning of the development project and this typically leads to design changes and implementation of additional properties in the later project phases. As the Lotus Notes applications are typically developed for groups of people, the requirement specification requires additional effort in order to support the group activities appropriately.

According to Larson-Hughes and Skalle (1995) it is difficult for the end user to visualize the functionality and effects of software without actually seeing it on the screen, discussing it and suggesting changes to it. Prototype approach provides means for discussing the system properties with the end users and thus supports the requirement specification activities. However, the role of formalized specifications and design documentation is emphasized also in prototyping projects. As the users suggest changes and additional functionality to the original prototype, the developers must be able to distinguish the originally specified properties as well as to demonstrate the required design changes to the users in order to justify the project schedule and resource plans. One way of demonstrating the structure and logic of the application is the use of graphical modeling methods (further discussed in section 2.2). This also helps the designers to keep up with the changing requirement specifications. As the prototype is a way for the developer to demonstrate the future system properties and functionality for the user, the graphical models are a way to demonstrate the future or current prototype properties and internal structure for the developers as well as for the end users if necessary.

When iterating through the prototype revisions, the use of a graphical modeling method also helps the designers trace the requirement specification changes to the corresponding development ideas. However, the problems of graphical modeling method utilization associated with prototyping projects also apply to rapid application development.

2.2 Modeling methods in information systems development

Development of an information system is always a collaborative process. Communication is needed among the system developers as well as between the system developers and the target organization. According to the Vessey and Sravanapudi (1995) the studies by DeMarco and Lister suggest that on large projects typical system developers spend 70% of their time working with others while Jones reports that team activities account for 85% of the costs of large software systems.

Even though information system development is a collaborative process the knowledge concerning the technical design of the system resides usually in minds of a few IS personnel. The use of graphical modeling methods attempts to utilize sharing of this knowledge with other key personnel. (Gibson et al. 1989)

Graphical modeling method is a method for modeling an object system using a graphical notation.

The basic concepts of graphical modeling are presented in Figure 3.

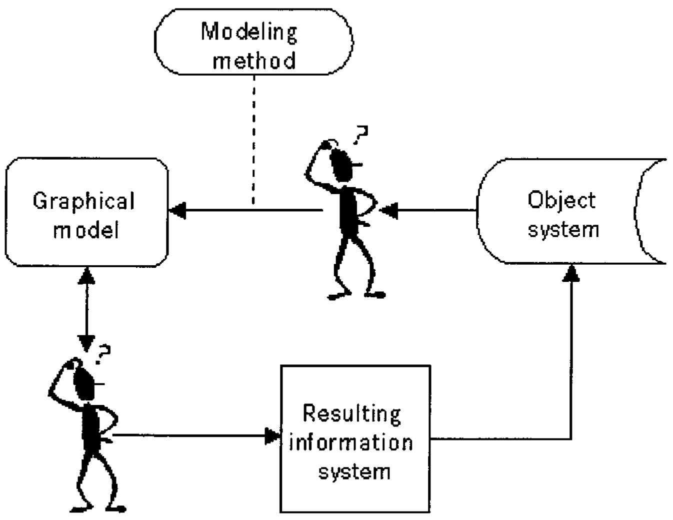

Figure 3: Concepts of graphical modeling

The developers generate the graphical models of the system to be developed in cooperation with the future system users. The models describe the desired system using the rules specified by the selected graphical modeling method. At this point the models can be considered as requirement specifications for the system to be developed. The generated models are updated during the development as necessary in order to ensure the correspondence of the models and the resulting system. When the system is finalized, the graphical models can be used as a part of technical documentation of the generated system. The resulting information system typically manipulates the behavior of the object system.

The graphical modeling methods can be utilized in all the systems development phases. In requirement specification phase the methods can be used in modeling the system to be built. In design phase the internal structure and the behavior of the system can be designed using graphical modeling and analyzing tools. In the development phase the system documentation can be constructed using graphical models. In the maintenance and updating phase the maintenance personnel can be supported offering them graphical representations of the system.

Graphical modeling methods can be utilized in enhancing communication between the developers and the end users as well as in introducing standard work practices in development teams. The application of graphical models in visualizing the system in the early phases of development increases the understandability of the system specifications and enforces standard ways of communication between different interest groups. According to Khosrowpour (1993) the studies of Granger and Pick also indicate that the systems developed with a graphical modeling method and a CASE tool support are better capable to meet the requirement specifications than non-case developed systems.

Formal specifications are an important solution to address software quality problems since they allow several important properties to be analyzed by the means of mathematics and sometimes by computerized tools (Jarke and Pohl 1992, Khosrowpour 1993). Most graphical modeling methods combined with CASE tools enable versatile analysis of the generated models according to specified rules. Analyzing models and specifications generated using graphical modeling methods decreases design errors and prevents inconsistent application specifications.

2.2.1 CASE tools

CASE tools are computer-based products aimed at supporting one or more techniques within a software development method. Typical CASE tool offers graphical editors to help developers model different aspects of a software system. In addition, most CASE tools have capabilities of checking integrity of generated models according to built-in knowledge about method rules. (Jarzabek and Huang 1998)

A number of surveys have been conducted to examine the major benefits of utilizing CASE tools in software development projects. Most of these studies report that CASE tools are perceived to improve software development productivity (Necco et al. 1989, Norman and Nunamaker Jr. 1988) and product quality (Wijers and van Dort 1990, Aaen and Sorensen 1991, Khosrowpour 1993).

Both of these factors include large sets of activities that can be improved with a proper use of CASE technology. Nevertheless, Aaen and Sorensen (1991) state that the benefits of using CASE tools are largely dependent upon the particular circumstances in which the tool is used. Therefore, it is important to study the factors that affect the efficient implementation of CASE tools in a particular type of software development project.

In our study we examine the benefits of CASE tool utilization in risk management of rapid application development projects. CASE tools are rarely utilized in combination with evolutionary approaches since the CASE tends to facilitate the use of structured tools in system design (Khosrowpour, 1993).

According to Aaen and Sorensen (1991) the impacts of CASE tool implementation can be studied with at least three alternative scopes:

1. Productivity improvements in diagramming and document production.
2. Tool impact on software life cycle.
3. Downstream benefits.

Productivity improvements in diagramming and document production refer mainly to the software analysis and design phases although the technical documentation of the later phases can be significantly improved by utilizing reports and diagrams generated with CASE tools. This scope of CASE tool benefits is the most measurable, but probably also the least informative because of its limited view.

Examining tool impact on software life cycle offers a wider perspective on CASE tool benefits. Besides speeding up analysis and design activities, the tools may improve design, e.g., by reducing the error rate or by modularizing the design structure. Profitability of CASE tool implementation can be measured relative to the costs of the software development process or relative to the total life cycle costs of the developed software. Measuring costs relative to the total product life cycle can be expected to provide more relevant findings, since the quality improvements in early phases of software life cycle can be expected to reduce subsequent costs significantly. (Boehm and Papaccio 1988).

In most cases the implementation of a CASE tool seems to warrant the introduction of new methods and work procedures, e.g. project management, configuration management, and quality assurance. According to Aaen and Sorensen (1991) these downstream benefits may contribute significantly to the profitability of CASE.

While Aaen and Sorensen (1991) focus on examining the outcomes of CASE tool implementation, Orlikowski (1993) suggests that the adoption and use of CASE tools should be conceptualized as a form of organizational change. According to Orlikowski (1993) much of the literature on CASE tools focuses on the discrete outcomes, such as productivity, system quality, and development costs, but neglects the social context of systems development, the intentions and actions of key players and the consequences of the implementation process. Examining how CASE tool implementation affects an organization allows us to anticipate, explain, and evaluate different experiences and consequences following the CASE tool introduction process.

2.2.2 Strengths and weaknesses of CASE tools

According to a survey conducted by Aaen et al. (1992) the five main strengths of using CASE tools are the abilities to:

1. Check consistency
2. Improve understandability
3. Automate routine procedures
4. Support software evolution
5. Introduce discipline in software development

Most CASE tools are capable of checking the consistency among various activities of software development. Consistency checking is based on the knowledge about the rules of a particular method that are built in the CASE tool.

Since typical CASE tools offer graphical editors, report generators, and support software documentation, they are assumed to improve the understandability of the developed software product. Especially in large system development projects, the understandability of the software design can be increased, by examining the system at various abstraction levels using appropriate modeling methods and abstraction features of CASE tools.

The productivity of an application development process can be increased using CASE tools to automate routine procedures like coding and report generation. Automating such tasks prevents errors and supports consistency among the generated models and the developed outcome.

Even though Aaen et al. (1992) have identified the support for software evolution as one of the main strengths of CASE tools, Jarke and Pohl (1992) state that CASE tools are ill-suited for the requirements of long-lived information systems with continuously increasing user demands on their functionality. According to Jarke and Pohl (1992) the CASE tools support the life-cycle phases only by improving the methodological consistency and communication within software development teams.

The introduction of a CASE tool in a software engineering organization affects both the development process and the product to be developed (Aaen and Sorensen 1991). Since the CASE tool implementation introduces consistent work practices for the software development teams, the level of standardization of the overall development process and the developed product increases (Aaen et al. 1992, Nunamaker 1989).

While Aaen and Sorensen (1991) and Aaen et al. (1992) consider the standardization of development process as a positive effect of CASE tool implementation, Norman and Nunamaker (1989) criticize the CASE tools for enforcing the development teams to rigid standards instead of supporting teams for continuous improvement. As CASE tools are mostly hardcoded, and there is no representation of the process or the rationales and goals behind their capabilities and actions, it is difficult for the users to describe the desired state of the process. This makes the migration to a new improved process structure complicated (Jarke and Pohl 1992).

The inflexible nature of CASE tools may also impede the tool implementation process. Since most CASE tools are too inflexible to adapt to different work procedures and methodologies used in organization, the software engineering personnel must be trained to use the particular methodologies supported by the CASE tool. Heym and Österle (1993) state that when installing CASE-technology at the IS department of any corporation, the major success and cost factors are adaptation and training for methodological support in using these tools.

Another problem derived from the inflexible nature of CASE tools is their insufficient support for local method development. By local method development we mean an organization's attempts to develop its own methods. According to Tolvanen (1998) organizations prefer to develop their own local variants of third-party methods, or adapt them to their specific needs. Most CASE tools support only fixed, situation independent methods, while the tool users prefer to use local, situation-bound methods.

While an organization applies information system development (ISD) methods, it creates knowledge about how to carry out ISD projects (Tolvanen 1998). The understanding of the possibilities of methods also increases. Therefore it is essential to be able to improve the development methods according to the experiences of previous projects. CASE tools' support for such development is scarce due to fixed and hardcoded methods.

Meta-CASE tools and so-called CASE shells can provide some solutions for the inflexibility problems of most CASE tools. These issues are discussed further in the next chapter.

2.2.3 Method engineering and Meta-CASE tools

Method engineering is a discipline of designing, constructing and adopting methods and tools for information systems development (Kumar and Welke 1992, Brinkkemper 1995). Heym and Österle (1993) define method engineering more narrowly stating that method engineering is a disciplined process of building, improving, or modifying a method by means of specifying the method's components and their relationships.

The idea behind method engineering is that instead of selecting one of the existing methods according to environmental contingencies, ISD methods should be constructed or tailored to meet a particular ISD project's requirements (Tolvanen 1998).

Method engineering can be carried out based on the needs of the whole organization or a particular project. In case of organization-based method engineering the methods can be engineered to fit the needs of the organization while the level of standardization of work procedures and products are maintained organization wide. When implementing project-based method engineering the organization can improve the method applicability at the expense of work process and product standardization.

When carrying out method engineering an organization must be able to compare existing information system development methods in order to select one as the basis for engineering (Olle et al. 1983). This initiates a need for a common notation to describe different modeling methods and thus support the comparison process.

A common notation for describing modeling methods is also needed when developing methods from scratch. When a new or re-engineered method is described with a common notation, supporting tools can be created by third party, according to these descriptions. Using a common notation to describe modeling methods also supports standardization and formalization of developed methods.

Metamodeling languages serve as common notations for describing different modeling methods. Notations can be textual, graphical or a combination of both. The metamodeling languages support communication during the method engineering process and provide means to formally describe the developed methods (Tolvanen 1998).

A *metamodel* is a model for describing the conceptual schema of a modeling method and metamodeling is a process of modeling a particular modeling method. According to Tolvanen et al. (1996) Brinkkemper presents the relationships between modeling and metamodeling as illustrated in Figure 4.

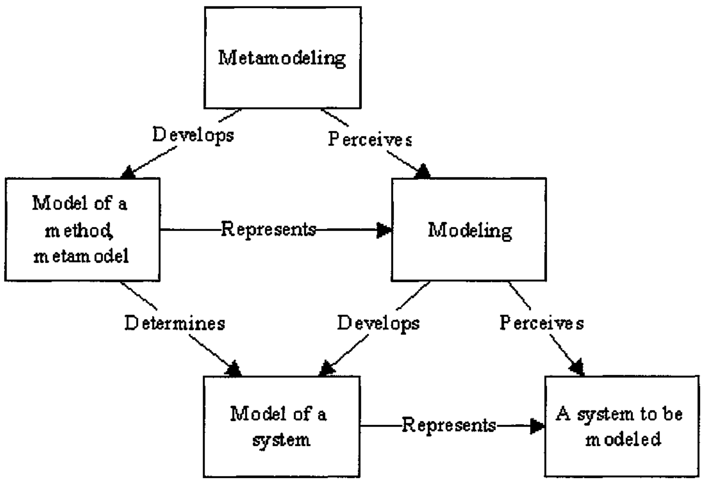

Figure 4: Conceptual schema of modeling and metamodeling (after Brinkkemper 1990).

Configurable CASE tools, meta-CASE tools, or CASE shells are aiming to improve the customizability of CASE tools (Heym and Österle 1993). They support a set of primitives (metamodeling languages), which allow users to describe a method quickly and mechanisms to implement a tool to support the defined method (Tolvanen et al. 1996). The modification of the underlying repository and the specification diagrams are the minimal requirements of meta-CASE tools (Heym and Österle 1993).

To support local method tailoring and development the method knowledge built in meta-CASE tools should be represented in an adequate way so that users can understand, apply, and enhance this knowledge (Heym and Österle 1993).

Heym and Österle (1993) have recognized four concepts as parts of an approach to computed-aided methodology engineering. (1) Methodology representation model for representing the knowledge and rules of the methodology. Representation model consists of (2) several methodology specification diagrams and techniques that specify the knowledge in user-convenient way. As the method is being tested and validated in different development environments, it is important to build (3) mechanisms for integrating the participants' expert knowledge and experiences to the method. In order to customize methods and techniques for specific projects within an organization a (4) versioning concept for managing different method versions is needed. Versioning concept should also support integration of different methods and techniques according to users' requirements.

2.2.4 MetaEdit+

MetaEdit+ is a metaCASE tool supporting a variety of information systems development methods. Since the NDL modeling method described in section 4.1 is implemented in MetaEdit+ meta-CASE tool, we discuss here briefly the basic concepts of MetaEdit+ environment.

MetaEdit+ has a set of predefined methods that can be used separately as individual methods or simultaneously in user-defined method combinations. Since MetaEdit+ is based on metamodels the users can tailor the existing models or create additional modeling techniques and methods by editing the underlying metamodels (Smolander et al. 1991). This makes MetaEdit+ a powerful tool for method engineering.

The metamodeling tasks in MetaEdit+ are carried out using metamodeling languages. The methodology specifications can be developed, analyzed and maintained in MetaEdit+ environment (Smolander et al. 1991). Smolander et al. (1991) present the concepts of MetaEdit+ and CASE shells as described in Figure 5.

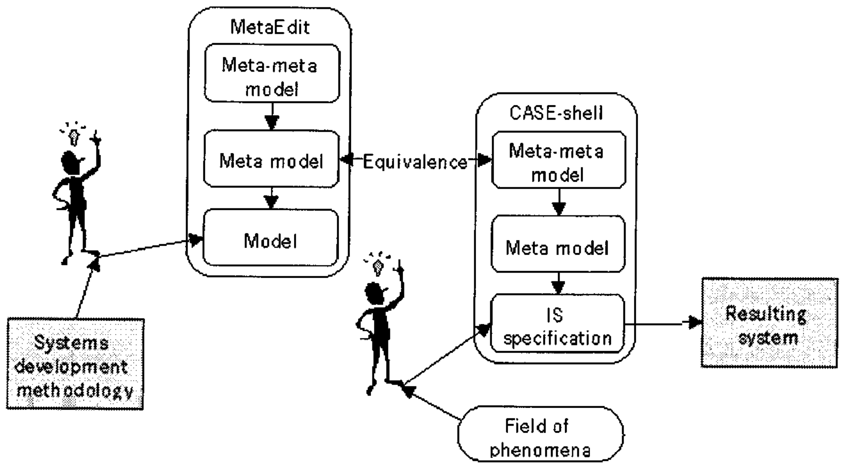

Figure 5: Three levels from meta-metamodel to model (after Smolander et al. 1991)

CASE shells consist of three levels. Information systems developers work on the IS specification level creating representations of the object system. The syntax and the semantics of these representations are defined in the metamodeling level. Metamodels are founded on a modeling language and associated concept structure is specified in meta-metamodel level of the CASE shell. This meta-metamodeling language offers the flexibility of the CASE shell environment. (Smolander et al. 1991)

These three levels are also present in MetaEdit+ though they operate one level higher than in CASE shell. The model level of MetaEdit+ defines the structure and functionality of the modeling language thus corresponding to the metamodel level of a CASE shell. The metamodel in MetaEdit+ is used to specify the methodology specification corresponding to the meta-metamodel level of CASE shell. The multilingual methodology specification support in MetaEdit+ is achieved by utilizing a meta-metamodel to specify the syntax and semantics of various metamodels that can be used to specify the methodology specifications. (Smolander et al. 1991)

2.2.5 GOPRR metamodeling language

In our metamodeling effort we use *GOPRR (Graph-Object-Property-Relationship-Role)* metamodeling language to describe the properties of the developed modeling method. GOPRR has been developed from the OPRR data model specifically to support the metamodeling process. The limitations of OPRR initiated the development of GOPRR language. Lyytinen et al. (1994) report the following limitations of OPRR:

1. No support for recursive structures
2. No support for generalization and specialization hierarchies
3. No support for interconnected methods

The main extension to the OPRR is the graph metatype, which is a collection of all other GOPRR types. Graph metatype makes it possible to specify connections between methods and techniques. The GOPRR also offers additional semantic relationship types between graphs and other non-property types and offers an abstraction mechanism to generalize and specialize non-properties (i.e. object, relationship and role types).

As the name of the GOPRR implies, the language is used to specify the following concepts of modeling methods:

1. **G**raph metatype which is used to manage specifications of different techniques and specify connections between methods and techniques
2. **Ob**ject types (e.g. Class, Process)
3. **P**roperties of the method components (e.g. Class name)
4. **R**elationship types (e.g. Inheritance aggregation)
5. **R**ole types (e.g. Superclass and Subclass in Inheritance type of relationship)

Due to the changes to OPRR semantics and addition of abstraction features, the GOPRR language supports also recursive structures. (Tolvanen 1998)

2.2.6 Supporting RAD with modeling methods

Graphical modeling methods are not usually utilized in rapid application development projects since the development method is characterized by *ad hoc* nature and the applications are developed in rapid succession. The design phase is often neglected in evolutionary approaches as the development methodologies usually emphasize the specification, implementation, testing, and evaluation phases. For example in PDCA (Plan, Do, Check, Act) methodology presented by Larson-Hughes and Skalle (1995) the "Plan" phase only contains specifying the objectives and resources for the development. All the system development activities including design are supposed to be included in the "Do" phase.

Another reason for the weak role of modeling methods and tools in rapid application development projects is apparently the seeming ease with which the applications can be developed. As the applications are developed rapidly and the necessary changes can be implemented with minor resources the explicit designing of the applications is considered being time-consuming and burdensome. Even though the advantages of explicit designing are recognized the necessary methods and tools are rarely utilized.

The inconsistent role of application specifications and the lack of proper application documentation are also due to the rapidity of the development cycles. New features are added swiftly and since the application specifications and implemented components are subject to a lot of changes the documentation process is likely to get omitted.

The lack of precise application specifications and detailed design documentation leads to the problems specified in Section 3.4. These problems can also considered as RAD risks as represented in Table 1.

Table 1: Risks in rapid application development

RISK	SOURCE
Vague application specifications	Inconsistent role of specifications and ad hoc nature of development
Problems in development and utilization of reusable components	Lack of systematic design of applications
Poor maintainability of developed software	Lack of systematic design and documentation of applications

In this study we examine the role of graphical modeling methods and supporting tools (e.g. CASE tools) in eliminating and controlling the risks associated with RAD projects. To be effective in RAD environment the tools must be designed to support rapid changing of the application and the corresponding models. Also the support for the development and utilization of reusable software components must be implemented in a way that the components can be easily manipulated according to the users' requirements.

2.3 Chapter summary

In this chapter we presented the concepts of software project, software development methodology, and different software development approaches. Additionally, the basics of prototype approach and rapid application development were further discussed in order to support presentation of RAD risk management issues in the following chapters.

As we study the utilization of a graphical modeling method and a CASE tool in RAD risk management, the basics of graphical modeling, CASE tools, and method engineering were also presented. The method engineering concepts and the description of MetaEdit+ meta-CASE tool are aimed to support the NDL method and NotesEdit CASE tool description in section 4.1.

3 SOFTWARE PROJECT RISKS

Despite the recent introduction and widespread use of a plethora of approaches, techniques, and tools, it seems that software development efforts still suffer from age-old difficulties of cost overruns, project delays, and unmet user needs (Barki et al. 1993). These difficulties are due to unsuccessful management of the risks related to a particular software project.

Even though risks have been studied from different approaches in a variety of domains, many definitions of risk comprise two dimensions: (1) the probability associated with an undesirable event, and (2) the consequences (usually financial) of the occurence of this event (Barki et al. 1993). These dimensions specify the relative importance of a particular risk. Charette (1989) specifies risk more narrowly as the potential for realization of unwanted negative consequences of an event.

According to Barki et al. (1993) McFarlan states that failure to assess individual project risk and to adapt management methods accordingly is a major source of the problems in software development. The software project risk management strategies aim at identifying the risk factors threatening the project and supporting the selection of appropriate risk management methods.

3.1 Managing software project risks

Software project risk management is a discipline for identifying and controlling the risk factors threatening the successful completion of a software project.

The emerging discipline of software project risk management is an attempt to formalize the risk-oriented correlates of success into a set of readily applicable principles and practices. The objective is to identify, address, and eliminate risk items before they threaten successful software project operation or become the major sources of software rework. (Boehm 1991)

Risk management involves two primary steps, *risk assessment* and *risk control.* Risk assessment involves risk identification, risk analysis, and risk prioritization. Risk control involves risk management planning, risk resolution, and risk monitoring (Figure 6). (Boehm and Ross 1989, Boehm 1991)

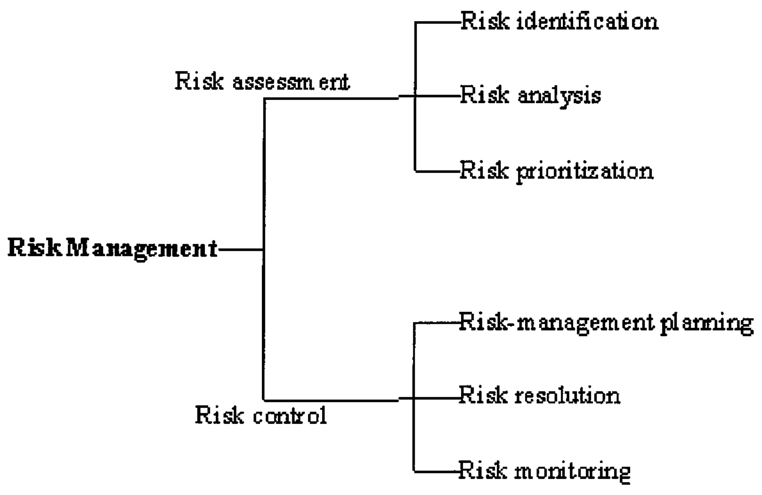

Figure 6: Software project risk management steps (after Boehm 1991).

Keil et al. (1998) emphasize the importance of risk assessment suggesting that before meaningful risk management strategies can be developed the risks must be identified, relative importance of the risks must be established and necessary managerial attention must be focused on the areas that constitute the biggest threats to the project.

Boehm and Ross (1989) divide risk assessment process into three primary activities: Risk identification, risk analysis, and risk prioritization. Risk identification produces a list of risk items that are likely to compromise the project's success. Risk analysis attempts to assess the probability and loss magnitude of each risk item and analyze the interrelations among the identified risk items. Risk prioritization ranks the identified risk items so that the risk control actions can be concentrated on the critical risk items.

We utilize these three risk assessment activities in developing an evaluation framework for analyzing the effects of CASE tool utilization on RAD project risks in chapter 4.

Many of the actualized software project risks lead to rebuilding parts of the object system. Boehm and Papaccio (1990) argue that the one of the key insights to improving software development productivity is that a large fraction of the efforts on software project is devoted to rework. The rework is needed in order to compensate for inappropriately defined requirements, or to fix errors in specifications, code, or documentation. Many software project cost overruns are also due to the assumption that the product is complete when the development is finished and the system is ready for use. The errors and modifications that emerge in the real-world user environment lead to unpredicted costs and delayed delivery times.

The risks related to rework cannot be overemphasized, since according to Boehm and Papaccio (1990), Jones provides research data indicating that the cost of rework is typically over 50% of the total costs on very large projects. When planning the management strategies to deal with risks related to rework costs, a significant insight is that the cost of fixing or reworking software is much smaller in the earlier phases of the software life cycle than in the later phases (Boehm and Papaccio 1990). Therefore additional efforts should be placed in the early phases such as the requirements analysis, specification, and software design.

3.2 Classification of software project risks

Since there is a wide variety of risks associated with a large software project, the project managers need tools for parsing the individual risk factors into a comprehensive risk management strategy. Classifying the individual software project risks in broader categories facilitates the risk prioritization and the planning of risk mitigation strategies (Keil et al. 1998).

Boehm and Ross (1989) divide software project risks in two primary classes:

1. Generic risks that are common to all projects and are covered by standard development techniques
2. Project-specific risks, that reflect a particular aspect of a given project, and that are addressed by project-specific risk management plans.

Whereas the general risks are managed with standard development techniques, the project manager must deal with the project-specific risks.

Keil et al. (1998) have studied the software project risks by assembling three panels of experienced software project managers in different parts of the world and asking them to identify and rank the specific risk factors they considered most important. As a result Keil et al. (1998) identified eleven risk factors that all the three panels viewed as important suggesting the existence of a universal set of risks with a global relevance.

According to their study Keil et al. (1998) present a two dimensional framework for categorizing the risks associated with software development projects. The framework is based on two dimensions: perceived level of control, and the relative importance of the risk. By perceived level of control Keil et al. (1998) refer to the level of control the project manager has on the particular risk category. The relative importance of the risk refers to the significance of a particular risk considering the probability of the risk occurrence and the potential loss associated with the risk item.

Keil et al. utilized the framework by mapping the identified risks into a 2 X 2 grid (Figure 7), which allowed the risk classification into four categories.

Perceived relative importance of risk			
	High	**1** Customer Mandate	**2** Scope and Requirements
	Moderate	**4** Environment	**3** Execution
		Low	High
		Perceived level of control	

Figure 7: A risk categorization framework (after Keil et al. 1998).

Categorizing the risks enables us to choose the proper risk management strategies for each type of risks. Keil et al. (1998) suggest that each of the four quadrants of Figure 7 requires a different risk management approach because of the diversifying nature of risks.

Most of the risks identified by the panelists fell in quadrant 1. Risks in this quadrant have high relative importance, but the project managers usually have little or no control over them. As the name of the quadrant implies the risks associated in this category are derived from the lack of commitment of the senior management and the future users of the system. Keil et al. (1998) state that without a clear charter, or mandate, the project is not viable. Risk management strategies required in this quadrant include creating and maintaining good relationships with customers and promoting the customer's commitment to the project. Prototyping is one possible way of controlling these risks.

Risks in quadrant 2 involve the ambiguities and uncertainties that arise in establishing the project's scope and requirements. These risks have high relative importance because misunderstanding the requirements and poor management of changes can lead to considerable costs and delays in the later phases of the project. Using proper tools and techniques project managers can achieve relatively high control over these risks.

Specifying the exact requirements of the system is not always possible at the beginning of the development project. The utilization of evolutionary approaches makes it possible to specify system's characteristics during the development but emphasizes the risks related to project's scope. Project manager must be able to control the system requirement specification along the development and educate the customer on the impact of scope changes in terms of project cost and schedule (Keil et al. 1998).

The risks in quadrant 3 concern the execution of the project. As examples of the risks in this quadrant Keil et al. (1998) present the issues of inappropriate or insufficient staffing, lack of effective development process methodology, poor estimation, and improper definition of roles and responsibilities. According to Keil et al. (1998) the project management has reasonable control over the risks in this quadrant and hence they are regarded as moderate rather than high-risk items. Risk management strategies in this quadrant include using disciplined development methodologies, defining roles and responsibilities, and developing plans to cope with staffing shortfalls and new technologies.

Risks in quadrant 4 are caused by project environment that exists both inside and outside the organization. The likelihood of the occurrence of these risks is low and naturally the project managers have little or no control over them. Even though the risks in this quadrant are the most difficult to anticipate, the consequences of their occurrence can be significant and dangerous. According to Keil et al. (1998) contingency planning is the most sensible strategy for dealing with environmental risks.

In our study we focus on examining the effects of graphical modeling method and a CASE tool utilization on the risks that fall primarily in scope and requirements (2) and execution (3) quadrants. Keil et al. (1998) suggest that with proper methods and tools the project managers can achieve relatively high level of control over the risks in both these quadrants. This study is conducted to examine the suitability of a CASE tool in assisting the management of these risks.

3.3 Groupware application development risks

Computer applications that are designed to support groups are commonly known as *groupware* (Grudin 1990). Groupware application design and development differs from single user applications in several aspects. Because our CASE study examines the risks associated with rapid application development in groupware environment (Lotus Notes) we discuss here some special characteristics of groupware development projects.

Groupware development project risks are typically related to the factors concerning the technical design of groupware application and the social, political, or motivational factors following the implementation of the developed system.

Grudin (1990) puts forward four problems that emerge in the development phase of a groupware project: The user interface design, support for different roles, social factors, and the study of group processes.

The interface of a groupware application must satisfy a number of individuals within a group. In case of single-user application the users have possibilities to tailor the user interface in some level, but in case of groupware the same user-interface must satisfy all the users.

The users of groupware application must be able to operate with *different roles* and privileges depending on the current situation (e.g. reader-author-editor).

The specification and design phase of the groupware application can include various motivational, political, economic, and *social factors* that usually play only little role in single-user applications. Also the number of people involved in specification and design can be significantly larger. Efficient communication among the users and developers during the specification and design phases is crucial to the successful implementation of a groupware application.

The group processes are often variable and context-sensitive and span a long period of time. The studying of group actions is time consuming and also the change results from introducing new technology can emerge after considerable amount of time. Because each group is different, the observations about the actions of one group are hard to generalize to other groups.

The four groupware application development phase problems presented above are only part of the risks that become apparent in groupware projects. The groupware implementation and the following social and organizational changes are usually the risk factors that the development team has only a little experience on, but that can prevent the application to live up to it's promise. In the following we briefly discuss the risks related to groupware implementation.

As the group members have different preferences, experiences, roles, and tasks, the implementation of a groupware application will never offer every group member the same benefits (Grudin 1990). Some group members may have to do additional work and adjust to different work practices more than others. The group activity is guided by social, motivational, political, and economic factors that are rarely explicit and stable (Grudin 1990). The implementation of a groupware application can interfere with the group dynamics in a way that is harmful to the implementation process.

Since the groupware implementation is a change process, appropriate change management methods should be applied. The social change in organization must be controlled and the management of potential social conflicts must be planned in advance. Active end-user participation in groupware specification and design phases is one solution to manage the risks in implementation phase.

The prototype approach is typically utilized in groupware development projects as it offers tools for active communication between the developers and the end users thus helping in managing the social risks discussed above.

3.4 Rapid application development risks

As discussed before, the prototype approach is a way of managing some of the risks related to traditional software development methods. However, it also puts forward some additional risk items. The utilization of prototype approach in a software development project usually necessitates the implementation of rapid application development since several different versions of the software must be developed quickly. Therefore, the risks related to prototype approach also become apparent in prototype-based RAD projects. Additionally, the RAD itself contains some inherent risk factors that are discussed below.

The RAD approach requires close cooperation between the system developers and the experts of the application domain. Without an ongoing dialog and feedback cycles, the software project starts to live a life of its own. The developers have a tendency of adding new technical features to the application without feedback from the experts as well as the experts typically request new functionality to the application, while the designers implement changes without reflecting the impact to the project itself. This leads to both unclear requirement specification and problems in requirements' traceability. By requirement traceability we mean the ability to specify the requirement changes in a way that makes it possible to later indicate what has actually been changed and why.

If the communication between the application developers and the experts of the application domain is not sufficient, the rapid changes in prototype design can lead to a situation, where software costs soon begin to rise after the system is taken into operational use and the maintenance costs shoot up due to the unplanned changes. Since the changes are implemented rapidly, the documentation tends to get neglected resulting in poor maintainability of the application. Therefore it is important to schedule changes in an effective and controlled way. (Boehm and Papaccio 1988)

The rapid succession of the development iterations and the remarkable amount of design changes implemented in a typical RAD project also affects the quality of design documentation developed during the project. The lack of detailed technical documentation typically leads to additional costs in the later development phases as well as in the system maintenance and updating activities.

The rapid development cycle also leads to problems in development and utilization of reusable components. When implementing applications quickly with no further design and modularization plans, it is difficult and time-consuming to create components that could be reusable in other development projects. Also the fact that the implemented module might not be included in the final version of the software does not encourage the developer to consider the reusability of design. The reuse of components in prototyping requires that the rapid application development environment is designed to facilitate the reuse with minimal effort, for example featuring categorization of components by functionality and clear documentation for each component.

One of the primary risks in RAD projects is the lack of design and modularization in system development. As the prototype revisions are reviewed with the customer in rapid prototyping cycles and the development focus is on the system functionality and user interface, the internal structure of the system tends to become obscure leading to additional work in later development phases and system maintenance. The lack of systematic and modular design also makes it more difficult to share the development work between different development teams. The prototyping risks of vague design documentation and poor maintainability of the developed software are also emphasized in RAD projects.

In RAD projects it is important to set milestones for the project in such a manner that the project progress can be measured and controlled. As a closely related item, clear objectives for project completion must be placed, so that the project will end when these objectives are fulfilled. If this is not done properly, it is difficult to draw the line between original goals and further development goals.

3.5 Managing Rapid Application Development Risks

An important objective for risk management in rapid application development is the improvement of communication between the application designers and the application domain experts (e.g. the end users), as well as among the development personnel and different development teams assigned to the same project. This necessitates the deployment of tools that not only facilitate the feedback between the expert and the developer, but which also organize the process so that transferring specification requirements to the prototype being developed is simple, concise, and effective.

One of the most significant factors influencing software costs is the amount of source code needed to be programmed. Hence, an effective cost-reduction strategy is to employ reusable components when developing software (Boehm and Papaccio 1988). In order to use reusable components effectively, proper facilities must be present in the development environment, for example shared component repositories or libraries and the necessary metadata assigned to each component. A shared component or object repository enables all developers to collaboratively share their own components and utilize the components developed by others. One must remember that the existence of reusable components and development tools enabling the use of them does not eliminate the need to adopt and use effective system development methodologies (Wasserman 1980).

However, in order to be able to create and utilize reusable components in software development, the system must be carefully designed, the possibilities of component generation and usage must be examined, and the necessary design changes must be implemented before the actual system implementation phase.

RAD approach to software development also brings additional challenges to the developers in the form of prototype versioning and configuration control. When creating several successive versions of a prototype, a need for a comprehensive revision control becomes apparent. Technical documentation must be updated and revised according to the changes implemented and measures must be taken to guarantee the integrity of the produced material.

Our study attempts to indicate how the utilization of a graphical modeling method in RAD projects affects the design and documentation quality, and the resources needed for the project. As we study the modeling method implementation effects from a risk-oriented point of view, the actual measurement of the design and documentation quality is not however the primary objective of this research.

3.6 Chapter summary

In this chapter we introduced the concepts related to software project risks. The software project risk management strategy by Boehm and Ross (1989) was described in detail. Boehm and Ross (1989) divide the risk management in two primary functions: Risk assessment and risk control. We utilize the risk assessment steps by Boehm and Ross (1989) in identifying, analyzing, and prioritizing the risks of the case projects described in chapter 4. The NDL modeling method and NotesEdit CASE tool described in section 4.1 are developed for studying the controllability of the identified risk factors using graphical models.

A concept of risk categorization was introduced according to Keil et al. (1998) and two frameworks for categorizing software project risks were presented. The risk categorization frameworks are utilized in analyzing the general RAD risk items identified in chapter 4.

Finally, some risks emphasized in groupware application development and RAD projects were presented, and methods for RAD risk management were suggested.

4 EVALUATION CRITERIA

In this chapter we describe NDL modeling method and NotesEdit tool as well as an evaluation criteria used for analyzing the effects of the method and the CASE tool utilization on rapid application development risks. The evaluation criteria is are applied in chapter 5.

The NDL modeling method and NotesEdit CASE tool are described briefly considering the goals of the method and tool development, the implementation process, and the proposed usage of the NotesEdit CASE tool.

The evaluation criteria description is divided in three sections. First, we present essential theories utilized in the evaluation criteria development. These theories are based on the risk management literature by Boehm (1989), Boehm (1991), Boehm and Papaccio (1989), Boehm and Ross (1989), Keil et al. (1998), and Cotterell and Hughes (1995). Second, we describe the three case projects implemented before the CASE tool development. The experiences acquired from these projects are then utilized in the evaluation criteria, and the NDL method development. The risks related to the case projects are identified, analyzed, and prioritized in order to support the identification of general RAD project risks. Third, the general RAD project risks are identified, analyzed and prioritized using the results of the two previous sections.

By identifying and analyzing the general RAD project risks we support the analysis of the effects of a graphical modeling method and a CASE tool implementation on different types of RAD project risks. This helps us to evaluate the effects of CASE tool utilization in RAD project risk management.

Outline of the development and application of the evaluation criteria is presented graphically in Figure 8 below.

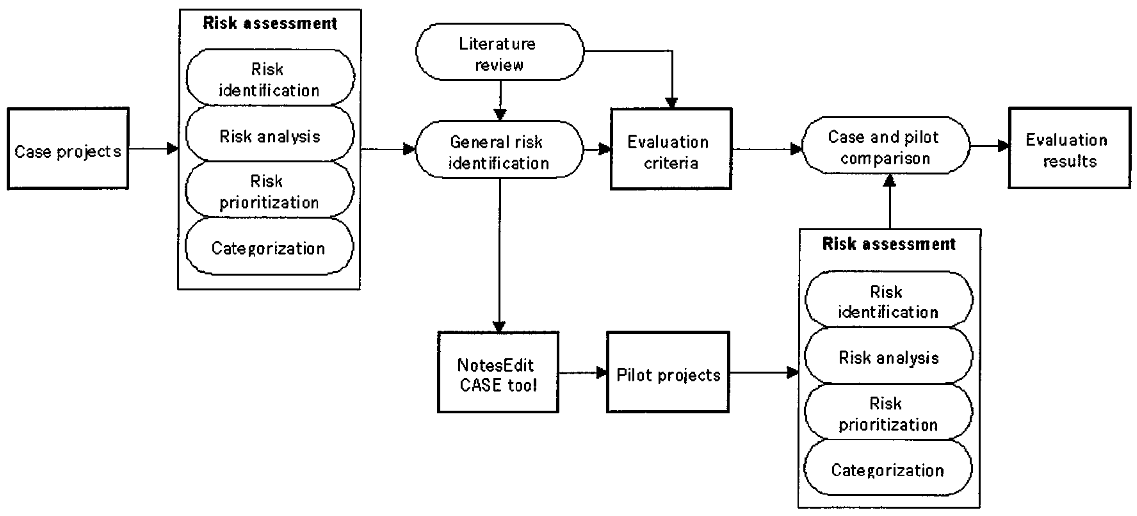

Figure 8: Development and application of evaluation criteria

The general RAD risks identified in the case projects and literature are used as the basis for the NotesEdit CASE too development as well as the development of the evaluation criteria. The NotesEdit CASE tool development, pilot projects, and the case and pilot project comparisons presented in Figure 8 are discussed further in later chapters.

4.1 NDL modeling method and NotesEdit CASE tool description

The goal of the NDL modeling method and NotesEdit CASE tool development was to generate an instrument which can be used to comprehensively model the design of a Lotus Notes application or an information system consisting of multiple Lotus Notes applications. By utilizing this instrument an application developer can replenish the design documentation of the developed system with graphical models describing the properties, functionality, and dependencies among different design objects. These models can be utilized in the system design and development activities as well as in system maintenance.

The NDL method engineering process was implemented by describing the NDL method using metamodels. In our NDL metamodeling effort we applied the extension of Object-Property-Role-Relationship (OPRR) metamodeling language called Graph-Object-Property-Relationship-Role (GOPRR), which has been developed specially for metamodeling (Tolvanen 1998). The GOPRR language was selected due to the available tool support and the previous experiences of the people involved in metamodeling and CASE tool implementation effort.

The NotesEdit CASE tool was implemented by importing the generated GOPRR metamodels to MetaEdit+ meta-CASE tool and describing the notations for the modeling techniques supported by the method. Additionally, some automatic error and warning checkings were implemented.

The NDL method consists of two modeling techniques, *Application architecture model* and *Database model.* The techniques are designed to model the object system in different levels of abstraction.

Application architecture model is used to model the structure of Lotus Notes applications (which applications consist of which databases) and the connections between separate Lotus Notes and Open Database Connectivity (ODBC) compliant databases. Application architecture models can also be used to represent the whole Lotus Notes application architecture of a particular organization. The databases represented in Application architecture models can be exploded to Database models.

Application architecture modeling technique is designed for supporting the structuring of the application consisting of multiple databases. The connections and interfaces between the separate databases can be modeled and analyzed utilizing the application architecture models. Thus, the models can be utilized in enhancing communication among the system developers developing different system modules, as well as in demonstrating the system structure for the end users. Additionally, the models can be used in analyzing the effects of design changes on the application level (instead of database level as in database models). Thereby, the application architecture models can be utilized in the system maintenance phase when updating or modifying the application functionality.

Database model is a technical description of a particular Lotus Notes Database and thus cannot be used to model ODBC compliant databases. Database models consist of Lotus Notes objects (e.g. forms and views) and connections between them. Connection from a particular object to an external database is also allowed. These connections should be modeled accordingly in Application architecture models.

Database modeling technique is designed to support the system design activities as well as the automatic application generation according to database models. The database models generated during the development can be used in enhancing the application design documentation and the analysis of the design changes in database level. The objects, properties and connections presented in Database model are converted to a semi-functional Lotus Notes application frame by utilizing the CodeGenie application frame generator.

Both of the presented techniques can be utilized in the requirement specification phase to formalize the requirement specifications and support the tracking of the specification changes. Additionally, as the system can be modeled according to the requirement specifications, we attempt to support the estimation of the resources needed for the project completion.

4.2 Risk assessment

The main objective of the evaluation framework development is to study the effects of NotesEdit CASE tool implementation on RAD project risks. In order to evaluate the impacts of the tool utilization on the RAD risk factors, we need to implement appropriate risk management strategies to identify, analyze, and prioritize the risks associated with RAD projects. Boehm and Ross (1989) refer to all these issues with the term *risk assessment*.

In our risk assessment effort we utilize the risk assessment steps identified by Boehm and Ross (1989) and further analyzed by Boehm (1991). According to Boehm (1991) risk assessment activities include *risk identification*, *risk analysis*, and *risk* *prioritization*. In addition to these steps, we attempt to *categorize* the identified risks in order to analyze the NotesEdit CASE tool implementation effects on different categories (e.g. RAD risks, risks related to the Lotus Notes environment).

The goal of risk identification is to produce a list of the risk items likely to endanger the project's success (Boehm 1991). In order to evaluate the CASE tool effects on RAD risks we attempt to identify the risk items that become apparent in RAD projects implemented by using prototype approach. The risk identification process is implemented by using checklists and risk identification methods presented in the literature (Boehm and Papaccio 1989, Keil et al. 1998), and by utilizing our own experiences from risk assessment of the case projects prior to the NotesEdit tool implementation. As prototype approach and RAD provide means for controlling some of the risks related to traditional development approaches, we utilize the case project experiences and the prototyping literature in refining the checklists to adapt to the research environment.

Risk analysis assesses the probability and magnitude of the loss associated with each of the identified risk items, and also attempts to identify the compound risks in risk-item interactions (Boehm 1991). Typical risk analysis techniques include network analysis, decision trees, cost models, and performance models (Boehm and Ross 1989).

The importance of a particular risk, known as the risk value or *risk exposure*, depends upon the *probability* of the hazard occurring and its potential *loss*, effect, cost, or impact. The purpose of risk analysis is to obtain some quantitative measure of both these factors to support comparison of the identified risk items in a meaningful way. (Cotterell and Hughes 1995)

By analyzing the interrelations of the identified risk items the overall exposure and cost of each risk item can be specified. As one of the identified risks may contribute to several other risk items, and on the other hand, the exposure of some risk items can be significantly increased due to the realization of several other risks, it is important to carefully analyze the interrelations of the identified risks (Figure 9). The identified relationships must be considered carefully in the risk prioritization phase.

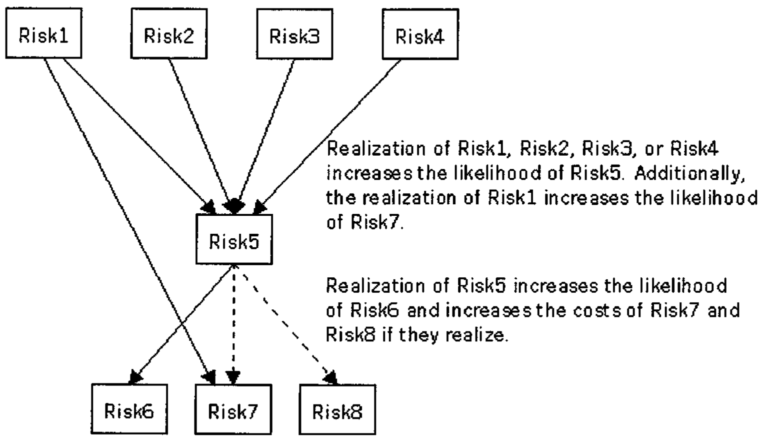
Risk1
Risk2
Risk3
Risk4
Risk5
Risk6
Risk7
Risk8
Realization of Risk1, Risk2, Risk3, or Risk4 increases the likelihood of Risk5. Additionally, the realization of Risk1 increases the likelihood of Risk7.
Realization of Risk5 increases the likelihood of Risk6 and increases the costs of Risk7 and Risk8 if they realize.

Figure 9: Risk interrelations

Good examples of interrelated risks are insufficient or inappropriate staffing and lack of adequate design documentation that both contribute to the probability and scale of the poor maintainability of the developed software.

In our research the risk analysis is conducted by assigning each risk a probability and loss ratings (on scale 1 to 10) that are used to calculate the overall exposure of a particular risk. The probability rating is used to describe the likelihood of the risk occurrence while the loss rating describes the scale of the costs associated with the risk. The overall exposure rating is calculated by multiplying the associated probability and loss ratings thus resulting in values 1 to100.

According to the risk analysis results the identified risk factors must be prioritized so that managerial attention can be directed to the critical areas of the development project. The priority of each risk must be assigned according to the risk exposure. However, the risk probabilities and losses should be calculated considering the interrelations among the identified risk items.

In order to further refine the framework of analysis for the risk factors surfacing in RAD projects we categorize the identified risk factors in a way that supports analyzing the

NotesEdit implementation effects on different risk categories. Additionally, the risk categorization aims to help in identifying the external factors affecting the risk items in the pilot projects.

In our risk categorization we utilize the categorization framework by Keil et al. (1998) presented in section 3.2. Furthermore, we develop our own categorization framework designed to support specifically the categorization of the RAD risks in groupware development (such as Lotus Notes) environment. With the aid of these frameworks, the identified risk factors are set in a more general domain describing certain problematic areas of software development in general, rapid application development, as well as in groupware development.

The first risk categorization framework is presented by Keil et al. (1998). This framework consists of four quadrants: Customer mandate, scope and requirements, environment, and execution. The framework is further discussed in section 3.2. As we place the identified risk factors into the corresponding quadrants with their respective risk exposure ratings, we can see the risk exposure distribution per quadrant. This offers us a guideline to which risk areas are predominantly dangerous in rapid application development projects. Additionally, this allows us to compare the risk exposure to the level of controllability of the identified risks.

Furthermore, we attempt to categorize the identified risk factors by the cause or reason leading to the actualization of the risk. We have identified five main categories that can be considered to describe the reasons for possible realization the identified risk factors. The problems and risks appearing in RAD projects in groupware development environment can be divided to the following categories:

1. General software project risks
2. Risks related to the prototype approach
3. Risks related to RAD
4. Risk related to the product specific environment
5. Risks related to the groupware development

By classifying the risk factors by reason we support the identification of the external factors affecting the risk items in the pilot projects and the evaluation of generality of the research results. For example, if the effects of CASE tool implementation are restricted to product specific environment, in this case Lotus Notes, the generality of the results is poor when considering the RAD project risks in general.

The risk management strategies implemented in this chapter are discussed in more detail in section 3.1. Next sections describe the three case projects representing each step of our RAD project risk assessment effort: Risk identification, risk analysis, risk prioritization, and risk categorization.

4.3 Case projects implemented before NotesEdit development

In this chapter we describe the three case projects implemented before NotesEdit CASE tool development. The experiences gained from these projects were used as a basis for the NotesEdit tool and the tool evaluation framework development. The case projects' risk assessment results are also utilized in chapter 5 in evaluating the CASE tool effects on the identified risk factors.

All the case projects were implemented in Lotus Notes environment using evolutionary RAD approach and relatively similar analysis, design, and development methods. However, the projects were carried out by two different development organizations and by three different development teams.

The next three chapters describe each of the projects as follows:

1. Project's target organization and objectives
2. Project planning
3. Project outcome
4. Project specific risk assessment

The project-specific risk assessment is carried out according to the risk assessment phases and tools by Boehm and Ross (1989). It should be noted that the risk assessment is carried out by the project manager in each respective project, basing on the subjective view of the project manager on the most important risk factors in each project. This approach does not necessarily give us a mathematically and statistically valid basis on comparing the projects. However, we aim to bring up some of the risk factors commonly appearing in RAD projects. The risk factors identified and prioritized by the project managers in each case are then further analyzed in section 4.3, given the methods suggested by Boehm (1991).

4.3.1 Case project 1

In case project 1 the target organization was a large Finnish company manufacturing machinery mainly for foreign markets. The company's sales department had previously collected all technical information on the machines manually by typing the information on sheets and adding technical diagrams and drawings on them, as supplied by the development engineers. The sales department needed to automate the system with a groupware solution that would make it possible to easily share the information to all sales personnel as well as carry the necessary information with them while visiting the customers.

The purpose of the application was to enable easy entering of technical specification for a piece of machinery and to provide this information for those who needed it. The application should also be able to print the required information with a proper layout. The research and development department of the company already had a similar information system, but it did not contain all information the sales department needed. The application was designed to completely replace the old manual, paper based archive system, which was problematic to update and share. The sales department had an information system developed some years earlier by another company, which contained the basic data for the machinery. However, this information system did not include all the necessary information about specific machine parts, which was vital for sales activity.

Project planning

The project was scheduled to last for 6 weeks with 2 programmers, a project manager, and three representatives from the customer. The schedule and personnel resources were estimated to be sufficient because the requirement specifications were very detailed and technically complete, indicating little or no need for specification changes during the project.

The specifications were supplied in the first meeting, containing detailed descriptions of each document form, complete with field descriptions. A layout proposal was also included which complied with the company standards on application development with considerations given to the printable layout of the documents. The specification included a link to an external application so that when the machine information would be entered in the database, the application would retrieve the basic information from the other application. The user then needed to fill in the data for specific parts only. When the basic information would be changed in the other database, the application was required to update the information accordingly.

Project outcome

The project was completed sharply on schedule with no timetable slip-ups. This was due to successful personnel resource assignment and schedule estimation in the beginning of the project. The customer organization adhered to the schedule and no set dates were missed or reassigned. After the final review, the sales department felt having obtained the goals set for the system. The application was taken into operational use and the use of the old manual system was discontinued.

Risk assessment

Despite that the project was successful, the project manager identified the following risk factors to be the top priority risks deserving attention to ensure a successful project completion. The risk factors were:

1. The external database connection would be difficult to implement
2. Some of the key personnel in sales department would have vacations during the project
3. The transition of technical knowledge of the machinery to the developers
4. The selected platform might impose restrictions for the layout setting when printing the documents

When analyzing the risks, two risk factors seem to be interdependent at some level: The vacations of the key personnel in sales department during the project and the transition of technical knowledge to the developers (Figure 10).

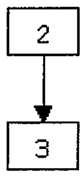

Figure 10: Case 1 risk interrelations

According to the project manager, the absence of the key personnel in sales department during a critical phase of the project can lead to project schedule overrun as the developers do not receive enough feedback from the customer. This lack of feedback and technical consultation can lead to unsatisfactory implementation of key features or misunderstanding in the logical structure of the application. Especially in RAD projects, it is important to have a constant feedback cycle between the developers and customer organization experts. When we asked the project manager to analyze and prioritize the risk items that he had identified above using the risk prioritization by Boehm (1991), we received the risk exposure ratings summarized in Table 2.

Table 2: Case 1 risk analysis

RISK ITEM	RISK OCCURRENCE PROBABILITY	LOSS WITH RISK OCCURRENCE	RISK EXPOSURE
The external database connection would be difficult to implement	5	8	40
Some of the key personnel in sales department would have vacations during the project	7	2	14
The transition of technical knowledge of the machinery to the developers	3	8	24
The selected platform might impose restrictions for the layout setting when printing the documents	6	3	18

The risk analysis and prioritization conducted by the project manager resulted in the following risk list sorted in descending order of risk exposure:

1. The external database connection would be difficult to implement
2. The transition of technical knowledge of the machinery to the developers in order to facilitate as logical application design could not be done sufficiently
3. The selected platform might impose restrictions for the layout setting when printing the documents
4. Some of the key personnel in sales department would have vacations during the project

The external database connection was clearly separated as the top risk item most likely to affect the project outcome. The transition of technical knowledge was set as the second risk factor deserving attention. The restrictions imposed by the technical platform and the key personnel vacations were approximately of the same risk exposure level and therefore set at low priority.

As the requirement specifications were playing a significant role during the project, the risks related to resource shortfalls and exceeded deadlines were not considered significant in this project. Also the social risks related to target organization personnel had only a minor role in this project since the personnel were well motivated due to the successful development projects implemented before this project.

4.3.2 Case project 2

The target organization for case project 2 was a small furniture manufacturer selling its products primarily for domestic retailers, with some foreign retailers and individual non-retail customers. The organization consisted of two primary departments: Combined management and marketing, and production. The developed system was intended for the use of management and marketing department only.

The project's primary objective was to develop an extranet system for online product ordering used by the target organization's retailers. The specifications also included integration to the customer's stock control system that was used for informing the subcontractors about the current stock situation.

The finalized system was to be installed on a server of an outsourced Internet service provider. The selected service provider did not offer the required server environment in the beginning of the project, but had made an agreement with the customer to have the required services running before the project deadline.

Project planning

The initial project organization was based on the requirement specifications received from the customer. The specifications described the system to be developed on a very high abstraction level and contained no technical view since the customer organization lacked knowledge on systems development. The project budget was determined according to these specifications since the customer demanded a fixed price for the project. The project was estimated to last for two calendar months and require approximately 2 part-time programmers.

Since the initial specifications received from the customer did not make it possible to start the development of a first system prototype, the first RAD iteration was started from the review phase with a starting meeting together with the customer. The communication between the developer and the customer still lacked common understanding of the detailed system functionality. As it later turned out, the customer was unable to point out all the necessary details affecting the system design and internal functionality. On the other hand the development team also lacked means for specifying these details due to the unfamiliarity of the customer organization. As a result of the meeting a more detailed specification of the system properties and functions was written. The first prototype review date was also set.

Project outcome

The specified system was implemented in three iteration rounds within the determined schedule. However, several additional functions and properties not specified in the initial specifications and the starting meeting were implemented and thus the project resources were slightly exceeded. The requirement specifications were also subject to a number of changes in each customer review. Implementation of the additional properties required several changes in the internal system functionality and external database connection and thus required significant additional resources.

Even though the system was finalized about a week before the specified deadline, the system installation and implementation activities were delayed for several weeks due to the delays in setting up the server environment with the service provider.

Despite the delays in the project schedule the customer was satisfied with the project, and the developed system completely met up the customer's demands. In this project, the prototype approach diminished the risks related to vague application specifications significantly and the development team was able to complete the project almost within the specified resources.

Risk assessment

As the project can be considered to be partially unsuccessful due to the resource and deadline overruns, the project manager was able to identify several risk items threatening the successful project completion:

1. The requirement specifications the project manager uses as a basis for project budget and schedule may not be specific enough.
2. The requirement specification changes may require additional knowledge and programming resources.
3. The Lotus Notes environment can set restrictions for the system implementation resulting in customer dissatisfaction.
4. Users may require several additional properties to be implemented within the project budget since no exact specification for the project scope and objectives is available.
5. The external database connection implementation failure due to network connections (customer had only an ISDN connection to the Lotus Notes service provider) or inappropriate development personnel.
6. Delays in the outsourced Internet service provider's Lotus Notes environment set up activities can delay the project completion.
7. Strict project schedule and the large number of possible design changes needed in each iteration can lead to poor maintainability of the system.

As the project manager analyzed the risk factors identified above, the following interrelations between individual risks were identified. The realization risk 1, vague requirement specifications, is strongly related to the exposure and cost of risks 2 and 4. As the inaccurate specifications increase the possibility and scale of design changes needed in the later development phases and make it difficult to set the project boundaries, the risks can be considered as interrelated. Additionally, the realization of risk 2, changing requirement specifications, is related to the risk 7 as the system maintainability can be reduced due to a lot of design changes implemented in rapid succession. The realization of risk 4 also increases the possibility of poor maintainability since the additional properties must be implemented within the strict project schedule. Therefore the risks 4 and 7 are also interrelated to some extent. The number of necessary design changes is also bound to the project budget and schedule since the additional changes always strain the project's resources and delay the project completion. Risks 4 and 3 can be considered interrelated since the realization of risk 4 increases the possibility of risk 3. Even if Lotus Notes supports all the functionality specified in the initial requirement specifications, the additional functions innovated during the project can exceed the possibilities Lotus Notes offers for the system developers. The interrelations of the identified risk factors are depicted in Figure 11.

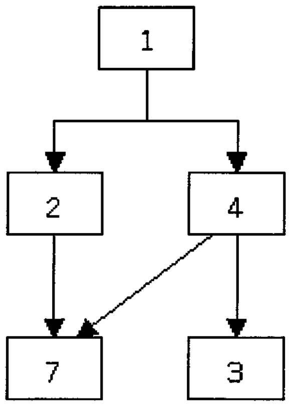

Figure 11: Case 2 risk interrelations

The project manager prioritized the identified risk factors as presented in Table 3. The prioritization was conducted using the risk prioritization framework identified by Boehm (1991). Each risk is assigned with a probability and cost rating that is used to calculate the overall risk exposure. As in the previous case project, the scores are subjectively rated by the project manager, based on the perceived importance and controllability of each risk item.

Table 3: Case 2 risk analysis

RISK ITEM	RISK OCCURRENCE PROBABILITY	LOSS WITH RISK OCCURRENCE	RISK EXPOSURE
Requirement specifications the project manager uses as a basis for project budget and schedule may not be specific enough.	8	9	72
Requirement specification changes may require additional knowledge and programming resources.	6	7	42
Lotus Notes environment can set restrictions for the system implementation leading to customer dissatisfaction.	2	5	10
Users may require several additional properties to be implemented within the project budget since no exact specification for the project scope and objectives is available.	4	7	28
External database connection implementation failure due to network connections or inappropriate development personnel.	3	3	9
Delays in the outsourced Internet service provider's Lotus Notes environment set up activities can delay the project completion.	2	8	16
Strict project schedule and the large number of possible design changes needed in each iteration can lead to poor maintainability of the system.	5	7	35

According to the risk analysis conducted by the project manager, the following prioritized list of the risk items was developed.

1. The requirement specifications the project manager uses as a basis for project budget and schedule may not be specific enough.
2. The requirement specification changes may require additional knowledge and programming resources.
3. Strict project schedule and the large number of possible design changes needed in each iteration can lead to poor maintainability of the system.
4. Users may require several additional properties to be implemented within the project budget since no exact specification for the project scope and objectives is available.
5. Delays in the outsourced Internet service provider's Lotus Notes environment set up activities can delay the project completion.
6. The Lotus Notes environment can set restrictions for the system implementation leading to customer dissatisfaction.
7. The external database connection implementation failure due to network connections (customer had only ISDN connection to the Lotus Notes service provider) or inappropriate development personnel.

As we can see the two risks considered as most important are both related to the vague requirement specification received from the customer in the project beginning. Both these risks also effect the probability and scale of several other identified risk factors. The prioritized risk factors presented above are analyzed further in section 4.3.

4.3.3 Case project 3

The customer in project 3 was a small Finnish company specializing on corrosion resistant and stainless steel machinery part manufacturing with approximately 40 employees. The company products were mostly targeted at domestic markets but some of the products were exported to foreign countries. The company had several subcontractors that mostly mechanized the semi-built parts made by the company.

As the company expanded its subcontractor network and area of expertise, a solution for controlling the information, and sharing it to the employees was deemed necessary. The personnel manager also wanted a company-wide e-mail and collaboration environment for sharing information as a platform for the application. The goals set for the project were ambitious – a complete removal of paper documentation, company-wide e-mail and an extensive information system for controlling orders, quality documentation, reclamations, project documentation, and personnel timetables, as well as installing the required hardware and software enabling the development and use of the information system.

Project planning

The project was scheduled to last for 6 months with a project manager and one full-time programmer. The project started with interviews with representatives from the customer organization. Seven interviews were arranged with key persons in the company: Managing director, personnel manager, account manager, subcontractor manager, production manager, assistant production manager and the production line executive. After the interviews were complete, a diagonal matrix of information, material, and product flows was produced in order to support the spotting of bottlenecks and areas where IS support would be needed. A general system specification was written on an abstract level in a meeting between the developers and the personnel manager.

The installation of necessary hardware and software started in cooperation with the company's network maintenance person. As the network maintenance person had a busy schedule, the installation of the hardware and software was delayed for some weeks. After the server installation, the Internet service provider for the company had trouble opening the TCP/IP –protocol routing for the network causing significant problems with the e-mail traffic. The problems were resolved after the installation of the initial prototype after some active correspondence with the ISP customer support department.

The requirement specification consisted of drafts of each form layout and information content in the application as well as the general concept of the application data flow and user-interfaces. The current paper forms being used in the company were taken as a model for the automated forms. The initial prototype was built rapidly with the document forms being exact copies of the paper-based forms, with no real thought given how to better automate and arrange them to fit better in a collaborative document management system. The software was created from the ground up, without using any previous components as a framework for the development.

The customer representative had a vacation in the middle of the project and the promised error report and development ideas were delivered several weeks later than was originally set. The developers realized that the fixed deadline was approaching and the delivered list of problems in the software was considerable, as the application was quite complex. The errors in the program were fixed quickly and most of the layout change proposals were carried out. The fixed prototype was installed for testing and error reports and development ideas were promised well before the end date of the project.

As the customer was under a heavy workload with orders coming from several customers, the development ideas and error reports could not be delivered in time. As the project deadline was reached, the developer organization saw no other possibility than to end the project.

Project outcome

After the project was terminated at the preset end date by the developer, the customer representatives were highly irritated by their actions: The installed software still contained several unresolved technical problems and the layout of the forms was not what they had expected. Several automation features were missing from the application and there were still problems with e-mail routing, as the service provider had not been able to get their mail server to redirect the company's e-mails without problems.

The customer blamed the developer organization for not fixing the software as requested and for not carrying out the required changes and modifications they wished for. The developer organization on the other hand accused the customer organization not being able to adhere to the set schedule and dates, especially in regard to software error reports and development ideas. The developers emphasized that the end date was fixed in the beginning of the project and the customer did not give the developers enough time to implement the required changes and to correct the errors.

The customer organization did not take the developed information system into operational use, relying instead on the old manual, paper-based documentation and workflow management. The software contained too many errors and design inconsistencies to be used effectively. The project was set on hold to wait for another company to carry out the required fixes and new requirement implementations to the software with an additional price.

Risk assessment

The case project 3 failed because of insufficient risk analysis and the resulting project-critical risk occurrences. The project manager of the development organization identified eight distinct risk factors:

1. The customer and the developer could not create clear requirement specifications
2. The scope of the project was too great compared to the available resources
3. The project schedule was too tight
4. The customer could not present the required feedback on software errors and development ideas within the required time
5. Insufficient people resources in developer and customer organization
6. The lack of component reuse when building the application
7. The Internet service provider for the customer had problems with rerouting the e-mail traffic
8. Poor design documentation and maintainability

As the project manager analyzed the identified risk factors, several interdependencies were revealed. The risk item number 5 – insufficient people resources in developer and customer organization - partially affects the possibility of risk factor 4 realization, the inability of the customer to produce the necessary reports of the software. As the customer organization only had one selected representative, being both busy and on vacation during the project, caused the delay of feedback to the developers. Also, risk item 5 is interrelated to the risk factor number 3, the tight project schedule. If the developer organization had more personnel assets, maybe the project schedule would have been sufficient. With only one programmer and a project manager, the schedule was clearly too tight. The tight project schedule was also due to risk items 4 (lack of customer feedback) and the risk item 2 (too large project scope). The absence of clear requirement specification (item number 1) also prevented the use of reusable components (item number 6) as well as led to shortage in personnel resources (item 5). With a project scope being too large (item number 2), it was difficult to create requirement specifications on a sufficiently detailed level (item 1). The large project scope also had an effect on item number 5, insufficient people resources. The risk item 8 (poor design documentation and maintainability) is effected by a combination of several other risk factors: Absence of clear requirement specification (number 1), project scope being too large (number 2), tight project schedule (number 3), insufficient people resource (number 5), and the lack of component reuse (item number 6). These complicated risk factor interrelationships are more clearly illustrated in
Figure 12 below.

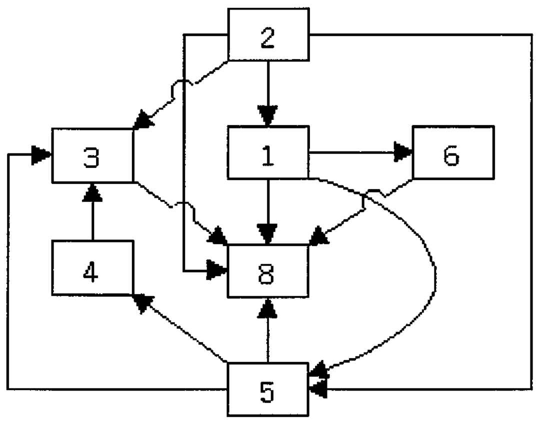

Figure 12: Case 3 risk interrelations

The project manager was asked to rate the most important risk factors in the project on terms with risk occurrence probability and loss with risk occurrence, based on his subjective view. The identified risk factors are prioritized below according to the risk prioritization framework by Boehm (1991). The risk exposure ratings with probabilities and losses identified by the project manager are listed in Table 4.

Table 4: Case 3 risk analysis

RISK ITEM	RISK OCCURRENCE PROBABILITY	LOSS WITH RISK OCCURRENCE	RISK EXPOSURE
The customer and the developer could not create clear requirement specifications	6	8	48
The scope of the project was too great compared to available resources	7	5	35
The project schedule was too tight	6	7	42
The customer could not present the required feedback on software errors and development ideas within the required time	5	8	40
Insufficient people resources in developer and customer organization	5	6	30
The lack of component reuse when building the application	6	8	48
The Internet service provider for the customer had problems in routing the e-mail traffic	2	5	10
Poor design documentation and maintainability	5	9	45

When the prioritized risk items are sorted in a descending order of risk exposure, a following list can be formed:

1. The customer and the developer could not create clear requirement specifications
2. The lack of component reuse when building the application
3. Poor design documentation and maintainability

4. The project schedule was too tight
5. The customer could not present the required feedback on software errors and development ideas within the required time
6. The scope of the project was too great compared to available resources
7. Insufficient people resources in developer and customer organization
8. The Internet service provider for the customer had problems with rerouting the e-mail traffic

The risk exposure estimates made by the project manager indicate that there were several high-priority risk factors affecting the project outcome and warranting immediate action with a risk management approach. Only the risk item 8, external service problems, was ranked very low. When analyzing the project outcome and the identified risk factors, it is easy to find out why the project outcome was a failure. There was not enough attention being paid to the critical project risk factors, no risk management techniques were in use, and therefore no measures were taken to control the probability of risk factors being realized.

4.4 General RAD risks

In this chapter we attempt to generalize the risk items identified in the presented case projects. By risk item we mean the actual risk identified in the development project and the terms risk category and risk type refer to the generalization of these risks. The generalization is conducted by utilizing the general risk identification checklists by Keil et al. (1998) and Boehm and Ross (1989) and by identifying risk factors corresponding to the risks identified in case projects.

The general RAD risks are identified by utilizing software development project risk checklists presented in the literature and the experiences gained by interviewing the project managers of the case projects described in the previous sections. By contrasting the list of subjectively important risk items in each case project and the lists presented in literature, we filter out risks that are not particularly relevant in RAD projects and add some risk items that are perceived as important by the project managers involved in case projects. Additionally, the general risk factors are analyzed, prioritized and categorized in order to support the NotesEdit CASE tool evaluation in further chapters.

4.4.1 General project risk identification

The general RAD risks are identified by utilizing software development project risk checklists presented in the literature and the experiences gained by interviewing the project managers of the case projects described in the previous sections. Boehm and Ross (1989) present a checklist of 10 software project risk items. The list is based on the experiences of several experienced project managers and is aimed at supporting the identification and resolving the most serious risk items threatening the successful completion of a software project. Boehm and Ross (1989) also suggest appropriate risk management techniques for each of the identified risks. The complete checklist of risk items and the corresponding risk management techniques is presented in Table 5.

Table 5: Software risk item checklist 1 (after Boehm and Ross 1989)

RISK ITEM	RISK MANAGEMENT TECHNIQUE
Personnel shortfalls	Staffing with top talent, job matching, team building, key personnel agreements, cross training.
Unrealistic schedules and budgets	Detailed multisource cost and schedule estimation, design to cost, incremental development, software reuse, requirements scrubbing.
Developing the wrong functions and properties	Organization analysis, mission analysis, operations-concept formulation, user surveys and user participation, prototyping, early user's manuals, off-nominal performance analysis, quality-factor analysis.
Developing the wrong user interface	Prototyping, scenarios, task analysis, user participation.
Gold-plating	Requirements scrubbing, prototyping, cost-benefit analysis, designing to cost.
Continuing stream of requirements changes	High change threshold, information hiding, incremental development (deferring changes to later increments).
Shortfalls in externally furnished components	Benchmarking, inspections, reference checking, compatibility analysis.
Shortfalls in externally performed tasks	Reference checking, preaward audits, award-fee contracts, competitive design or prototyping, team-building.
Real-time performance shortfalls	Simulation, benchmarking, modeling, prototyping, instrumentation, tuning.
Straining computer-science capabilities	Technical analysis, cost-benefit analysis, prototyping, reference-checking.

Keil et al. (1998) criticize Boehm and Ross (1989) and Boehm (1991) for the narrow focus of the checklists as they are primarily built upon their experiences from U.S. defense industry in the 1980's and might not be suitable for the needs of typical business enterprises. Keil et al. (1998) also point out that both the organizational and technological landscape have changed considerably in the last ten years.

Keil et al. (1998) have re-examined the risk issue by developing an updated checklist to meet up the demands of today's business organizations. The list contains 11 risk items that have been sorted in descending order, from the most important to the least important, by their relative importance. By relative importance Keil et al. (1998) refer to the combination of risk probability and the cost of risk realization. The prioritized list of risk items identified by Keil et al. (1998) is presented below.

1. Lack of top management commitment to the project
2. Failure to gain user commitment
3. Misunderstanding the requirements
4. Lack of adequate user involvement
5. Failure to manage end user expectations
6. Changing scope / objections
7. Lack of required knowledge / skills in the project personnel
8. Lack of frozen requirements
9. Introduction of new technology
10. Insufficient / inappropriate staffing
11. Conflict between user departments

Some of the risk items correspond to the risks identified by Boehm and Ross (1989), but the updated list also contains several additional items. As we can see, the risks related to computer technology (Real time performance shortfalls, Straining computer science capabilities) included in the list identified by Boehm and Ross (1989) have not been included in the list updated by Keil et al. (1998). Since the information technology has evolved rapidly in the 1990's the risks related to the performance and availability of technology resources are no longer considered relevant. The focus has shifted to the social factors bound to software projects.

However, several corresponding risk items can be identified from the presented checklists. Both lists include several items related to the requirement specification and objectives of the project. Also the risks related to project and target organization personnel are present in both lists.

In our research we support the risk identification by using a checklist which is a combination of the lists identified by Boehm and Ross (1989) and Keil et al. (1998). In addition we have added two risk items (lack of adequate design documentation, poor maintainability of the developed software) that are emphasized in RAD projects based on the experiences acquired from the case projects implemented before the NotesEdit CASE tool implementation (see section 4.3). These two risk items were perceived as important by the project managers in the case projects when estimating the factors effecting the outcome of the project. On the other hand we have excluded some risks that can be relevant to the success of an IS development project, but are irrelevant when studying the effects of a CASE tool implementation on RAD project risks. Our checklist contains the following risk items.

- Misunderstanding the requirements
- Failure to manage end user expectations
- Continuing stream of requirement changes
- Personnel shortfalls
- Unrealistic schedules and budgets
- Gold-plating
- Shortfalls in externally performed tasks
- Lack of adequate design documentation
- Poor maintainability of the developed software

The reasons for selecting the risks specified are presented below. The risks excluded from our list and the reasons for exclusion are presented briefly in Table 6.

Table 6: Risks excluded from the evaluation framework

RISK	REASON FOR EXCLUSION
– Lack of top management commitment to the project – Conflict between user departments	These risks are of social nature and are on the responsibility of project manager. CASE tools offer no significant additional means for controlling these risks.
– Failure to gain user commitment – Lack of adequate user commitment – Developing the wrong user interface	Prototype approach is a tool for controlling these risks.
– Introduction of new technology – Real-time performance shortfalls – Straining computer-science capabilities	CASE tool implementation offers no additional means for controlling technology risks.

It is worth noticing that most of the risks included in our risk identification checklist can be located in the Scope and requirements and Execution quadrants of the risk categorization framework identified by Keil et al. (1998). Risks in these quadrants have high level of controllability and high or moderate relative importance. On the other hand, most of the excluded risks are located in the Customer mandate and Environment quadrants of the risk categorization framework by Keil et al. (1998) and thus have low level of controllability.

The reasons for the inclusion of the selected risk items are described below.

Misunderstanding the requirements

Misunderstanding the requirements is included in both presented lists (Boehm and Ross (1989): Developing the wrong functions and properties, Keil et al. (1998): Misunderstanding the requirements). Accordingly Boehm and Papaccio (1988) emphasize the importance of the early phases of software development since the errors in requirement specification are likely to cause significant costs in later development phases.

The risks related to misunderstanding the requirement specifications were identified in all the case projects described in section 4.3 as presented in Table 7.

Table 7: Identified case risks related to misunderstanding the requirements

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
1	The transition of technical knowledge of the machinery to the developers.	24	2
2	The requirement specifications the project manager uses as a basis for project budget and schedule may not be specific enough.	72	1
3	The customer and the developer could not create clear requirement specifications.	48	1

However the prototype approach itself provides some means for controlling the risks related to requirement specification (see Table 5) as the incorrect requirement specifications can be replenished when the prototypes are reviewed with end users.

Nevertheless the end users tend to focus on the issues related to the user interface and other visible software components, while the logic behind the actions cannot be fully reviewed without complete piloting material. In our piloting projects we concentrate on studying the CASE tool utilization in prototyping projects in demonstrating the logical structure of the prototype and though support the prototype reviewing process.

Another issue related to incorrect requirement specifications is the amount of iterations needed for completing the desired software product. In our pilot projects we study how the CASE tool implementation effects to the amount of iterations and how do the original requirement specifications correspond to the finalized information system.

Failure to manage end user expectations

Failure to manage end user expectations is included in the list by Keil et al. (1998). This risk is especially emphasized in groupware development projects (Grudin 1990). If the developed system fails to satisfy the end user expectations the success of the whole implementation process is threatened. As there are a number of other social factors affecting the implementation process, the failure to manage end user expectations during the system development can lead to additional difficulties in the implementation phase. Most of the other factors threatening the successful implementation tend to be emphasized if the system does not meet the expectations of the end users.

The risks related to the end user expectations were identified in the case projects described in section 4.3 as presented in Table 8.

Table 8: Identified case risks related to end user expectations

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
1	The selected platform might impose restrictions for the layout setting when printing the documents.	18	3
2	The Lotus Notes environment can set restrictions for the system implementation leading to customer dissatisfaction	10	6

The prototype approach is one way of managing the risks related to end user expectations. When the users review and test the system prototypes, the expectations can be actualized, but as discussed before, the users typically concentrate on testing and reviewing the visual part of the system, while the internal logic and the non-visible properties of the system remain unseen. Comprehensive prototyping of a groupware system is also problematic since the benefits and drawbacks of the system implementation can be realized only with an appropriate amount of users.

In our pilot projects we study the CASE tool utilization in end user expectations management in two ways. First, can the CASE tool support the demonstration of the system prototype in such a way that the end users can understand the functionality of the system so that the expectations can be realized before the implementation phase. Second, can the CASE tool support the demonstration of the design changes needed when the users demand for changes or additions in the prototype functionality. Typically, a minor change in the visual part of the prototype can lead to major changes in the internal design structure of the system. The demonstration of these internal changes for the end user can be used to realize the expectations of the transformability of the system prototypes.

Continuing stream of requirement changes

Continuing stream of requirement changes is also included in both presented checklists (Boehm and Ross (1989): Continuing stream of requirement changes, Keil et al. (1998): Changing scope and objections, Lack of frozen requirements). This risk is emphasized particularly in prototyping projects. As the idea of the prototype approach is to rapidly build software prototypes with only vague requirement specifications, the original specifications are subject to a lot of changes. Thereby, the resources spent to the project tend to accumulate on each iteration and since no exact requirement specifications for the system have been generated in the early phases of development, it is difficult to decide when the system is finalized.

The risks related to the requirement specification changes were identified in the case projects described in section 4.3 as presented in Table 9.

Table 9: Identified case risks related to requirement changes

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
2	The requirement specification changes may require additional knowledge and programming resources.	42	2
2	Users may require several additional properties to be implemented within the project budget since no exact specification for the project scope and objectives is available.	28	4

As the software prototypes are reviewed with end users on each development cycle, the users typically innovate additional properties and functions. Since no exact requirement specification is available, it is difficult for the developer to identify the necessary and "nice to have" properties. This leads to difficulties in stopping to the endless stream of requirement changes since the properties of the finalized system have not been fixed in the beginning of the project.

In our research we concentrate on studying the requirement changes from the technical point of view by analyzing the amount and scale of the change requests and demands for additional properties. We also compare the project results to the original specifications and attempt to analyze CASE tool effects on the project controllability in terms of requirement changes.

Another important issue for study are factors leading to requirement changes in the later phases of the development project. Through CASE tool utilization we attempt to increase the role of requirement specification in the early development phases and thus eliminate part of the requirement changes due to the misunderstandings and communication problems between users and developers in the requirement specification phase.

Personnel shortfalls

Personnel shortfalls is another risk included in both checklists (Boehm and Ross (1989): Personnel shortfalls, Keil et al. (1998): Lack of required knowledge and skills in the project personnel, Insufficient or inappropriate staffing). Also these risks are likely to be realized in prototyping projects since the project's scope is typically poorly specified due to the vague requirement specifications. Because software development is based on continuous user feedback, resources needed for project completion are difficult to estimate in the beginning of the project. Also skills required for the system development are hard to estimate in the beginning of the project since the implementation of additional properties specified in the later prototyping iterations may require unpredictable proficiencies.

Risks related to personnel shortfalls were identified in all the case projects described in section 4.3 as presented in Table 10.

Table 10: Identified case risks related to personnel shortfalls

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
1	Some of the key personnel in sales department would have vacations during the project. The external database connection would be difficult to implement.	14 40	4 1
2	The external database connection implementation failure due to network connections (customer had only ISDN connection to the Lotus Notes service provider) or inappropriate development personnel.	9	7
3	Insufficient people resources in developer and customer organization.	30	7

In our research we study the CASE tool support in estimating the personnel and skills required for the system development. If the system components can be specified in adequate level of detail in the beginning of the project the risk of personnel shortfalls is likely to be reduced. We study the CASE tool effects on these risks by specifying the system in the beginning of the project by using graphical modeling method with a

CASE tool support and by estimating the required personnel resources according to the generated models. This allows us to compare the projects before and after CASE tool implementation and thus evaluate the role of CASE tool in the estimation process.

Unrealistic schedules and budgets

Unrealistic schedules and budgets is a risk related to the estimation of the scale of the project. This risk is included in the checklist by Boehm and Ross (1989). As discussed in the previous chapters, in prototyping projects the vague requirement specification in the early development phases, makes it difficult to evaluate the personnel required for the project completion. This applies also to the schedules and budgets of the project. As the software to be developed is specified only in relatively high abstraction level, it is difficult to estimate the actual time and resources needed for the development. The project budget is typically strongly related to the required resources and as both the schedule and resources are estimations based on the vague requirement specifications they result in risks like exceeded deadlines and budget overruns.

The risks related to unrealistic schedules and budgets were identified in two of the case projects described in section 4.3 as presented in Table 11.

Table 11: Identified case risks related to unrealistic schedules and budgets

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
2	Strict project schedule and the large number of possible design changes needed in each iteration can lead to poor maintainability of the system.	35	3
3	The scope of the project was too great compared to the available resources.	35	6
	The project schedule was too tight.	42	4
	The customer could not present the required feedback on software errors and development ideas within the required time	40	5

Also these risks are studied by comparing the case project schedule and budget estimations to the pilot projects after CASE tool implementation. We also use two test groups to compare CASE tool effects on the estimation process as described in the previous chapters.

Gold-plating

This risk was included in the list by Boehm and Ross (1989). Gold-plating risks can be divided in three categories: Requirements gold-plating, developer gold-plating, and user gold-plating. Requirements gold-plating is a term referring to a situation where the developers overemphasize the performance and complex features of the system while the users are interested in the overall functionality instead. Some of the features the developers and the project marketing personnel lay stress on may be even unnecessary for the end users.

Developer gold-plating refers to the tendency of the developers to try out all the new technical solutions even if they are not actually needed by the end user. Each additional feature increases the time required for debugging, testing, and other processes and though strains the project resources for unnecessary purposes.

By user gold-plating we refer to the situation typical for projects implemented using the waterfall approach, when the users asked about their requirements would frequently reason, “I don’t know if I’ll need this feature or not, but I might as well specify it just in case” (Boehm, 1995). This situation can occur also in prototyping projects if the prototypes reviewed with users include non-necessary properties or functionality due to the requirements gold-plating and developer gold-plating. The users may not actually need these properties, but since they are already partially implemented in the prototype, the users require them to be implemented also in the finalized system.

The risks related to gold-plating were identified in the case projects described in section 4.3 as presented in Table 12.

Table 12: Identified case risks related to gold-plating

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
2	Users may require several additional properties to be implemented within the project budget since no exact specification for the project scope and objectives is available.	28	4

Prototype approach reduces the gold-plating risks as the prototypes are reviewed with the end users in rapid succession and the unnecessary features can be eliminated in the early development phases. However the gold-plating risks must be controlled also among the systems development personnel so that the user gold-plating risks can be avoided. Systematic investigation of the system design is required to prevent the development of unnecessary functionality, and the usage of non-familiar development techniques and technical solutions.

In our research we study the possibilities to support the controllability of the software development by a graphical modeling method and a CASE tool. When the properties and functionality of the system are carefully designed before the implementation, implemented solutions can be controlled by the project manager. As the developers complement the design documentation during the development the project managers are able to observe the development process and stop unnecessary development activities.

We study the CASE tool's role in supporting these activities by measuring the amount and scale of the properties and functions that are not required by the end users in the original specifications, but which, however are implemented in the finalized system. After reasons for the development of these properties are analyzed we categorize them into the following categories: necessary properties not included in original specification, additional "nice to have" properties implemented due to the vague application specifications, unnecessary properties due to gold-plating. The properties implemented due to gold-plating are then further analyzed by specifying the type of gold-plating that led to the implementation of a particular property.

Shortfalls in externally performed tasks

Shortfalls in externally performed tasks include the shortfalls in externally furnished components as well as the shortfalls in other externally performed tasks. These risks are often due to the lack of communication between the development teams and external parties, and the lack of adequate design of system components. Even in prototyping projects the system components are typically developed separately in different teams, departments, or even in different organizations. The components cannot be linked together until certain level of development has been reached. As the requirement specifications of component prototypes are changed due to the user feedback the risk of overall incompatibility increases.

The risks related to externally performed tasks were identified in the case projects described in section 4.3 as presented in Table 13.

Table 13: Identified case risks related to externally performed tasks

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
2	Delays in the outsourced Internet service provider's Lotus Notes environment set up activities can delay the project completion.	16	5
3	The Internet service provider for the customer had problems with rerouting the e-mail traffic	10	8

As the role of designing the components is likely to be increased by an implementation of a graphical modeling method we study the effects of the method and CASE tool implementation on the risks related to the externally performed tasks. The implementation effects are studied by measuring the amount of additional communication needed among the development team and the amount and scale of the changes required to the components due to the component incompatibility. The results of the externally performed tasks are also compared to the results of the tasks implemented by the actual development team.

Lack of adequate design documentation

The lack of adequate design documentation is another typical risk in prototyping projects. Since the prototypes are developed in rapid succession and the properties and functions of the system are subject to a lot of changes, the design of the system is typically left undocumented during the development. If the design is documented during the early iteration phases, the documentation is likely to expire when the changes and additions are implemented in the later phases. If the design of the system is not documented during the development, it is likely to be left poorly documented, if documented at all, when the project is finished. The rapid succession of prototyping iterations forces the developers to focus on the system development activities while the documentation is considered as an additional burden.

However, the lack of adequate design documentation can lead to difficulties in the later development phases as well as in the system maintenance activities. As the person or team implementing the design changes is not always the one that originally implemented the particular design component, it is important to be able to understand the component functionality as well as to analyze the effects of design chances cause in other parts of the system.

The risks related design documentation were identified in the case projects described in section 4.3 as presented in Table 14.

Table 14: Identified case risks related to design documentation

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
3	Poor design documentation and maintainability.	45	3

Since the CASE tool support in documentation activities as well as in analyzing the system functionality is widely recognized (see section 2.2.1) we study the possibilities to implement a CASE tool in prototyping projects. The CASE tool implementation is studied by evaluating the documentation and maintainability of the developed software and by measuring the time spent to the documentation and design activities. The results of the projects prior to and after CASE tool implementation are compared by means of documentation quality, development efficiency and software maintainability.

Poor maintainability of the developed software

Poor maintainability of the developed software is another risk related particularly to prototyping projects. The risks related to the lack of design documentation, gold plating, inappropriate staffing and vague requirement specifications all contribute negatively to the system maintainability. Additionally the prototype approach neglects the actual design of the system and since rapidly implemented and coarse design solutions are typically difficult to update and maintain.

The risks related to software maintainability were identified in the case projects described in section 4.3 as presented in Table 15.

Table 15: Identified case risks related to maintainability

CASE NO.	RISK DESCRIPTION	RISK EXPOSURE	RISK PRIORITY
2	Strict project schedule and the large number of possible design changes needed in each iteration can lead to poor maintainability of the system.	35	3
3	Poor design documentation and maintainability.	45	3

In the pilot projects we study the CASE tool implementation effects on the maintainability of the developed system by analyzing the system design and evaluating the implemented design solutions in means of system maintainability. Additionally we evaluate the effects of the other realized risks on the overall design and maintainability of the system.

4.4.2 General project risk analysis and prioritization

The risks involved in RAD projects have to be analyzed and prioritized to accurately measure the effects of NotesEdit CASE tool implementation on important risk items. The risk analysis is accomplished by inspecting the interrelations among the identified risk factors. The prioritization of the risks is carried out using Boehm's (1991) framework presenting an effective method of risk prioritization - risk exposure calculation. We, however, generalize this technique and prioritize the identified risk factors for a *typical* RAD-project carried out in a groupware development environment, such as Lotus Notes. This approach is somewhat similar to the method presented by Barki et al. (1993). The risks identified in the previous chapter are on a general level and applicable to rapid application development projects in a wider context. The following risk exposure ratings are applicable to RAD-projects *without* the support of a graphical modeling method. In a later chapter, we will examine the effects of a graphical modeling method implementation on these risk factors.

There are some applicable restrictions that we impose on a definition of a typical RAD–project in order to facilitate the use of Boehm's (1991) risk prioritization framework. First, the customer of the project must not exist within the developer organization. Without this restriction, several of the identified risk factors would lose their significance. Second, we assume that the software is not dispensable. By this we mean, that the software has a full lifecycle with a maintenance phase. For example, a dispensable software would be a registration system for an event, that would be discarded right after the event has ended. With these restrictions in place, we can create a generalized prioritization for the common risk items associated with RAD projects. The interrelations among the identified risk factors are presented in Figure 13.

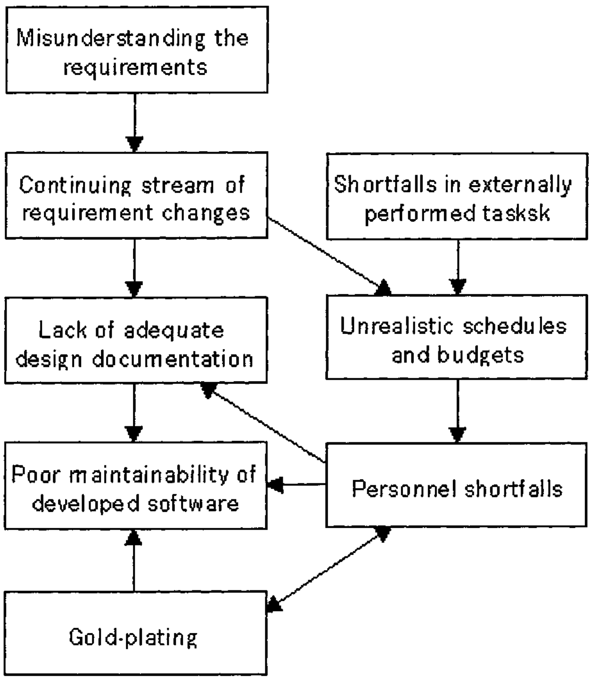
Misunderstanding the requirements
Continuing stream of requirement changes
Shortfalls in externally performed tasksk
Lack of adequate design documentation
Unrealistic schedules and budgets
Poor maintainability of developed software
Personnel shortfalls
Gold-plating

Figure 13: General risk interrelations

As we can see in Figure 13, several of the identified risk factors are interrelated. These interrelations must be considered when assigning the probability and loss values for each risk.

The risk prioritization is carried out by assigning each risk factor a probability of unsatisfactory outcome, P(UO) and a loss caused by unsatisfactory outcome, L(UO). Both probability and loss are scaled from 0 to 10. The resulting risk exposure, RE, is determined by multiplication: RE = P(UO) * L(UO). The inherent problem with this risk prioritization approach comes from the difficulty on making accurate estimates of the probability and loss with an unsatisfactory outcome (Boehm, 1991). When applied to general risk factors, the possibility of making false or misleading estimates increases even further. This, however, acts a rough guideline to the priority of risk items in general and offers a basic agenda for applying risk reduction techniques.

In Table 16 we present the general risk items and the risk exposure ratings of each item.

Table 16: Risk exposure for identified risks

Risk	**RISK EXPOSURE (RE)**
Misunderstanding the requirements	48
Failure to manage end user expectations	14
Continuing stream of requirements changes	35
Personnel shortfalls	23
Unrealistic schedules and budgets	38
Gold-plating	28
Shortfalls in externally performed tasks	13
Lack of adequate design documentation	45
Poor maintainability of the developed software	40

The risk exposure rating associated with each risk is an average of the risk exposures assigned to the case project risk items that were assigned each general risk. The ratings assigned to the case project risks were estimations made by the project manager of each project.

The analysis of these risk factors and their corresponding exposure ratings reveal three top risk items: Misunderstanding the requirements, lack of adequate design documentation and finally, poor maintainability of the developed software. These are followed by three medium priority risk factors: Unrealistic schedules and budgets, continuing stream of requirement changes, and gold-plating. In this study, we focus closely on these top and medium priority risk factors and how the use of a graphical modeling method impacts the risk exposure of these risk factors.

It must be noted that the identified and prioritized risk items are not necessary the most important risks when considering the overall project. Some of the important risk items have been excluded from this research since they have been considered to be irrelevant when studying the effects of a graphical modeling method and a CASE tool implementation. This restricts us to study the risks that are of relatively technical nature.

4.4.3 General project risk categorization

The exposure ratings of the identified general risks summed up within each category of the framework by Keil et al. (1998) are presented in Table 17.

Table 17: Identified risk factors set in the categorization framework by Keil et al. (1998)

1. CUSTOMER MANDATE – Failure to manage end user expectations (total exposure: 14)	**2. SCOPE AND REQUIREMENTS** – Misunderstanding the requirements – Continuing stream of requirement changes (total exposure: 83)
4. ENVIRONMENT – Shortfalls in externally performed tasks (total exposure: 13)	**3. EXECUTION** – Personnel shortfalls – Unrealistic schedules and budgets – Gold-plating – Poor maintainability of the developed software – Lack of adequate design documentation (total exposure: 174)

As specified above, most of the identified risk items exist in categories 2 (scope and requirements) and 3 (execution). This is due to the design and development oriented view of our research. When we attempt to study the effects of a graphical modeling method utilization on the RAD risks, the customer mandate and environment risks have only little importance to our study since the method is mostly utilized in design and development activities. Additionally the risks in quadrants 2 and 3 have high importance and controllability according to Keil et al. (1998).

As we categorize the identified risk factors by the environmental reasons increasing the risk exposure the following table can be generated (Table 18).

Table 18: General software project risks categorized by reason

RISK CATEGORY	RISKS
General software project risks	– Misunderstanding the requirements – Shortfalls in externally performed tasks – Gold-plating
Risks related to prototype approach	– Continuing stream of requirement changes – Unrealistic schedules and budgets – Personnel shortfalls
Risks related to RAD	– Poor maintainability of the developed software – Lack of adequate design documentation
Risks related to product specific environment	None
Risks related to groupware development	– Failure to manage end user expectations

In Table 18 each risk is included in only one category even though some risks would have met the requirements of several categories. Additionally, each risk is included in the category that mostly emphasizes the risk emergency. For example, though the risk 'Continuing stream of requirement changes' could have been included in the general software project risks, it is included in the prototyping risks instead, since the risk is emphasized especially in prototyping projects.

Another noteworthy factor that can be perceived when inspecting Table 18 is that none of the identified risks are actually caused by product specific environment, in this case, Lotus Notes. Even though some of the project specific risk factors identified in the case projects can be considered to be due to the Lotus Notes environment, the generalized risk list contains no risks that are directly due to the project implementation environment. However some of the risks are further emphasized in Lotus Notes environment. For example the risk 'Poor maintainability of the developed software' is included in the prototyping risks, but the poor maintainability is also partly due to the deficiencies of the Lotus Notes environment, due to the lack of shared component libraries and proper design tools. The lack of product specific environment risks suggest, that the results can be applied to groupware development environment on a greater scale, and the results are not necessarily limited by the selection and use of a certain product.

4.5 Chapter summary

In this chapter we constructed an evaluation criteria for evaluating the effects of graphical modeling method and CASE tool utilization on RAD project risks. The criteria are based on a list of general RAD project risk items, which was generated by utilizing the risk management literature and the risks identified by the project managers of the three case projects described in section 4.3. The identified general risks were further analyzed by utilizing the risk assessment techniques by Boehm and Ross (1989) and Boehm (1991).

The evaluation criteria are applied in chapter 5 by comparing the risk assessment results of the case and pilot projects on the general risk basis. The techniques utilized in the comparison were described in section 4.3. Furthermore, the identified general risk factors were categorized using the two categorization frameworks presented in chapter 3 in order to support the analysis of the modeling method utilization effects on different risk categories.

The identified general risk factors and case project experiences are used as basis for the NDL modeling method and NotesEdit CASE tool development in the following chapter.

5 APPLICATION OF THE EVALUATION CRITERIA

In this chapter we evaluate the developed NDL modeling method and the NotesEdit CASE tool described in section 4.1 by utilizing the evaluation framework described in chapter 4. The evaluation is conducted by focusing on the perceived effects of the CASE tool implementation on the identified general RAD risk factors as seen by the project manager, developers and customer representatives. The risk items identified in the case projects implemented before using the NotesEdit CASE tool are used as points of comparison when evaluating the pilot projects. The project manager, developers and the customer give their subjective view on the effects of using the CASE tool, both positive and negative. These remarks are then contrasted to the identified list of RAD project risk factors, and the impact of NotesEdit tool is evaluated and discussed.

The first two sections describe the projects where the tool was piloted and the analysis is described in the following sections. The benefits and drawbacks of the NotesEdit method are presented, the implementation effects on the identified risk factors are analyzed, and the overall usability of the modeling method and the case tool is evaluated.

The two pilot projects described below were both implemented by the same organization, but by two different development teams. In both pilot projects, the project manager was assigned from the developer organization. The work practices and tools used in both projects were similar to the case projects described in section 4.3. However, the project customer was different in each case and pilot project. In addition the pilot project 1 development team was not instructed to the NDL usage, while pilot project 2 staff was offered complete instructions of how to utilize the method. By not instructing the pilot project 1 staff to the use of the tool we were able to examine the project personnel's view of the NDL utilization benefits. In pilot project 1 the tool was not utilized in such development phases where the development team considered it as an additional burden, or the utilization was considered unnecessary, while in pilot project 2 the development team was instructed to utilize the tool in all the development phases.

5.1 Pilot project 1

The customer in pilot project 1 was a medium-sized Finnish company. The main production line consisted of high-technology accessories for large mechanical and hydraulic machines. The company was concentrating on export sales, with a small percentage of domestic subcontracting for large corporations. The IT manager for the company wanted to start a new project to enhance the maintainability of the product information. The current product hierarchy was complicated and each machine part had its own spare parts, each with own information. The main user group for the information system was the sales and marketing department. They were in a constant need of current, updated information of the product line.

The IT department decided to implement the product catalog as an Intranet application, built on Lotus Notes / Domino architecture. This technology platform allowed the product catalog to be online, easily reached by everyone with an Internet browser. The sales organization for the company was very mobile and they preferred a lightweight solution that could be updated centrally. The updates needed to be easily reachable without additional effort by the sales personnel.

Project planning

The project was given a strict schedule. The implementation of the graphical modeling method was estimated to speed up the initial prototype construction so, that the project could be completed within three weeks. A project manager and two programmers were assigned for the project.

The corporate supplied general guidelines on how to implement the system but mostly the designers had free hands on developing the first prototype. The most important requirement was that the system should present the product hierarchy with a visual and clear manner. The implementation of the product hierarchy was designed on the basis of the previous system. The customer and the developers agreed on the set specifications and a model of the application was constructed with the developed modeling method. The first prototype was implemented in two weeks and two days according to the model. After this, the prototype was delivered to the customer for evaluation and the required changes and modifications were documented for the development team. The revisions were carried within two days and the finalized prototype was then installed for the customer.

Implementation of the graphical modeling method

Pilot project 1 was implemented with the support of the developed graphical modeling method. In this pilot, the usage of the method was not instructed or guided in any way. The developers had the liberty of using the method as they saw fit. The requirement specifications were modeled using the database modeling technique described in section 4.1. The created model was revised by the developers and then circulated with the development team. Module implementation was shared in the team based on the created model of the application.

The developers found it easy to implement required functionality according to the model as it included all the necessary information normally found on requirement specifications. The need of communication among the developers was reduced since the constructed model was complete enough to specify all necessary details of the system. The information was more structured and easily adapted to technical Lotus Notes design when compared to a traditional paper-based documentation.

The use of the NotesEdit –tool in pilot project 1 was limited to the requirement specifications. The developers and the project manager agreed that the method best supported the specifications phase. However, when updates and error corrections were implemented and a new version of software was produced, the developers felt that updating the graphical model of the software was not necessary since the developed prototype itself could be used as the basis for the communication. Therefore, the use of the modeling method was not seen very productive after the first version of the software.

Project outcome

The project was completed roughly on time. The only schedule changes were due to difficulties in finding an appropriate time slot that was suitable for both customer and the developer. The deadlines were met and the software product itself was produced according to the set schedule.

The customer was very pleased with the project outcome. The application had been built according to the given specifications and the product hierarchy had been successfully incorporated into it. The developers found the use of the NotesEdit tool to be a significant factor in helping the transformation of requirement specifications into a technical design.

Effects of the modeling method on project risk factors

The project manager and some of the developers of the first pilot project were interviewed as how the risk factors were affected by the usage of the modeling method. The project manager felt that the most significant impact of the method was to control the transformation of the specifications given by the customer into a technical architecture for the application. The formalized model of the application also helped the project manager to assign resources effectively and modularize the application accordingly in order to increase the performance of the development work.

The developers found the use of the modeling method as a stabilizing factor in the prototype construction. The developed prototype had exactly the needed features and no more – thus decreasing unnecessary gold-plating. The developers also appreciated the modularized view of the specification resulting from the use of the graphical modeling method. This reduced problems with synchronizing development work and defining interfaces between modules. The development team also felt that the developed model of the application served as a good base for technical documentation. However, the complete technical documentation could not be derived solely from the model, as the developers did not update the model with changed specifications during the development cycle.

After the project was completed, the project manager pointed out that it would probably have been more productive to use the modeling method across the development cycle and update the model when changes were implemented. This would have resulted in clear technical architecture documentation for both the developer and customer organization.

5.2 Pilot project 2

In pilot project 2 the customer was a large Finnish company manufacturing large machinery for mostly foreign markets. The corporation included several factories, each with a specific product line. The customer organization had a long history of making IT projects and most of the workflow activities in the company were automated using groupware solutions, in this case Lotus Notes architecture.

The next step was to automate the measurement reports in each factory, as specified by the quality standards of the company (ISO 9000). Previously, the measurement data was collected manually and then fed into a Microsoft Excel report. The IT department now wanted to start a project that would automate this process with a workflow application.

Project planning

The project was scheduled to last for two months, with two programmers and project manager assigned to it and a factory representative from the customer as well as a representative from the customer IT department.

The specifications supplied by the customer were exact, as the factory representative had designed a first screen prototype of the application. Each phase of the production was represented by a form consisting of all the needed measurement results. This screen prototype was used by the development team to develop the finalized requirement specifications.

The project was divided into two distinct phases: the first part of the project consisted of creating the application for collecting measurement data at each production phase. After this part was complete, a mechanism for creating the measurement report automatically would be implemented.

Implementation of the graphical modeling method

In pilot project 2, the use of the modeling method was guided, and it was used throughout the development cycle. First, the developers created a model of the application according to the screen prototype supplied by the customer. The developers found it simple to design the specifications for the application using the supplied screen prototypes. After the specification was complete and a graphical model of the application was ready, the team modularized the application according to the graphical model. Each module was developed and the models were revised as changes were being done.

After the project was complete, the models, incorporated with comments on technical design of the product, were given to the customer as a technical diagram of the developed software. This facilitated the customer organization to develop the application further if needed and with different software developers, if necessary.

Project outcome

The first part of the project was completed in time, with the customer and the developer engaging in active discussion about the developed features. Each new version of the software was delivered to the customer for comments. The results of the first phase of the project were approved and the second phase was started as scheduled.

The measurement report was delayed slightly as the customer and the developer had difficulties in specifying the information content of the report. Additional problems surfaced as the customer representatives had vacations and only one of them was available for comments at a time. The report was completed, with some adjustments to its content two weeks late of the original schedule. Though using new technology, no problems were encountered, as the design documentation was kept well up to date with the aid of the tool.

In the final review, the customer presented some small corrections and additions to be implemented. The corrections were added into the graphical model and the application design was modified accordingly. With these changes in place, the results of the project were approved and the project closed. The customer was very satisfied with the result, feeling that the necessary functionality was implemented with some extra features the end users had hoped for. The application was immediately taken into operational use forming the backbone of the quality documentation for the product line.

Effects of the modeling method on project risk factors

The effects of the modeling method were studied by interviewing the project manager and the development team, as well as the customer IT representative. The persons were asked to freely express their view as how they felt the use of the method had affected any general risk factors they have possibly perceived during the project.

One significant advantage of using the modeling method pointed out by the project manager was that the strict screen prototype would not cause ineffective design decisions, as the method allowed the developers to plan the application logic in a structured, team-based environment, using the supplied graphical notation and share the development tasks effectively. In addition, the project manager felt that technical documentation was now much better implemented than in any previous projects, also helping the later phase to be completed successfully.

The project manager also addressed the importance of successful first phase of the project. The outcome of the second phase, the addition of Excel interface, depended largely on the quality of the first phase. Using the modeling method to specify the application logic and interfaces between these two modules greatly lessened the risk of problems with interfacing the Excel module into the existing application design.

The developers found out that the use of the method allowed them to utilize some reusable components made in other projects more efficiently than before. This advantage allowed the software to be developed faster. The clear structure of the developed software also produced some new reusable components, especially in the Excel interface.

A good technical documentation was one of the primary concerns of the IT representative from the customer organization. The updated model of the application was illustrative for the IT representative and she believed that any future development to the application would be made easier with a clear representation of the application design and logic in the form of a graphical model.

5.3 Evaluation of the NotesEdit –tool

In this section, we contrast and evaluate the differences between the three previous case projects implemented without the NotesEdit –tool and two pilot projects, implemented using the developed tool. We concentrate on the effects imposed on different types of risk factors in case projects, as identified by the project managers. The use of categorization frameworks provides us with a more generalized idea of the results of the modeling method, as we contrast the generalized risk items with the findings of the project managers, developers and customer personnel in both pilot projects.

In order to carefully analyze the contribution of the modeling method, we structure the analysis according to the research problem stated in chapter 1. In the first section, we identify the primary risk factors threatening RAD projects using the project manager experiences with the three case projects. In the second section, we contrast the identified risk factors in previous case projects with the risk factors identified in pilot projects implemented with the NotesEdit –tool and attempt to pinpoint the risk factors affected by the use of the modeling method. The second section summarizes all the advantages and disadvantages of using the tool identified during the pilot projects. The final section concentrates on finding the most efficient ways of using the modeling method in order to control important risk factors in a RAD project on a more generalized level.

5.3.1 Primary risk factors in RAD projects

The case projects all had a roughly similar amount of scheduled resources and approximately equal amount of project management experience. Even though each project has its own specific environment (e.g. target organization, developers), by examining the generalized risk factors in order of descending risk exposure, we get a following list (numbers in parenthesis are the average risk exposure for the particular risk factor):

1. Misunderstanding the requirements (48)
2. Lack of adequate design documentation (45)
3. Poor maintainability of the developed software (40)
4. Unrealistic schedules and budgets (38)
5. Continuing stream of requirements changes (35)
6. Gold-plating (28)
7. Personnel shortfalls (23)
8. Failure to manage end user expectations (14)
9. Shortfalls in externally performed tasks (13)

As stated in section 4.3.3., most risk factors are located in the execution quadrant in the framework presented by Keil et al. (1998). In two of the three case projects, most severe risk factors were clearly associated with insufficient level of detail with requirement specifications and lack of adequate design documentation.

5.3.2 Risk factors affected by the use of the NotesEdit CASE tool

In the previous cases, project managers clearly pointed out that the top risk factors in these projects were strongly set in both execution and scope and requirements quadrants in the framework presented by Keil et al. (1998). Both pilot projects demonstrated, that the use of the modeling method had an effect of decreasing the estimated risk exposure for these risk factors.

When relating the effects of the modeling method as seen by the developers, the project manager and the customer representative, three quite notable differences become apparent between the case and the pilot projects: "Misunderstanding the requirements", "lack of adequate design documentation" and "continuing stream of requirements changes" risk factors were top risk items in rapid application development projects carried out without the use of a modeling tool. However, in pilot projects exposure to these risk factors was identified to have been reduced by the use of the modeling method. Two of these risk factors are located in the scope and requirements and one in execution quadrant of Keil et al. (1998) model. On the other hand, when looking at the general software project risk categorization presented in table 19, these risk factors are placed evenly in general software project risks, risks related to prototype approach and risks related to RAD categories. These results do not exclude other effects of the environments specific to each project, but provide an estimate of the effects of a CASE tool implementation in RAD projects.

5.3.3 Advantages and disadvantages of using the NotesEdit CASE tool

The comparison of previous case projects and pilot projects illustrate the usefulness of the developed NotesEdit CASE tool in controlling the software project risks associated with scope and requirements risks and uncontrollable specification changes. According to the project managers in the pilot projects, the tool offered the development team a good medium for developing and maintaining the required level of technical documentation during the project.

Project managers also felt that the use of the tool and the modeling method enabled a more structured form of modularization and division of labor during the programming-intensive phases of the project. The design of the application was also easier to orient towards developing reusable components.

The developers in both pilot projects valued the common platform for discussions that the model offered in team meetings. In previous projects, the amount of technical documentation during the implementation of the software was usually quite small, which made effective teamwork somewhat more difficult, in contrast to the pilot projects.

Representatives of the customer organization also preferred a more visual form of technical documentation in addition to traditional text documentation. For example, further development ideas were easier to specify when the developers and the customer had the same tool for visualizing the design and logic of the software.

Perhaps one of the most important deficiencies of the NotesEdit tool was the lack of linkage between the software and the developed model. When making rapid changes to the software, the developers found it an additional burden to update the design of the model as well and document the changes. The need for an automatic mechanism was seen necessary.

Another feature that was deemed necessary was the need for a way to create a simple application frame from the developed model. Much of the routine programming and laborious testing of basic features could be avoided by using an application frame generator. In conjunction with application frame generator was the need to deposit software components that were used in many different projects with a minimum need for customization in one central location, where they could be easily imported into an existing application and the represented in the model as well. As the development in Lotus Notes environment is highly GUI –oriented, the layout properties of a saved component should also be remembered by the tool, eliminating the need of readjusting the layout of fields and other design elements every time the component is used.

5.3.4 Efficient utilization of the NotesEdit CASE-tool

The advantages and disadvantages summarized in the previous section suggest that a concise way of implementing the modeling method is needed when utilizing it in RAD projects. As it stands now, NotesEdit needs some enhancements before it can fully support RAD development with Lotus Notes. Two important issues need to be remedied: automatic synchronization between the model and the application design, and the ability to generate application frames with design element layout capabilities automatically from models.

These limitations suggest, that the most efficient uses of NotesEdit are, as suggested by the opinions of the pilot project managers, development team and customer representatives:

- Supporting customer requirement specification transformation into application logic
- Enabling efficient modularization of application
- Easing the assignment of developer resources for different parts of application
- Providing means of generating good basis for technical documentation of application

With these considerations in mind, the use of NotesEdit CASE-tool can help control some of the key risk factors surfacing in many RAD projects. With the limited scope of two pilot projects and subjective opinions from the project managers, developers and customer representative, the effects of the modeling method cannot be measured very accurately. Further pilot studies are needed with a larger scope of applications to be developed and with more diverse customer environments, before a definitive guide for using the NotesEdit CASE-tool can be created.

6 SUMMARY AND CONCLUSIONS

In the previous chapters, we have examined the risk factors that are prominent in rapid application development projects. In order to effectively manage some of these risks, the use of a graphical modeling method is suggested. By first constructing a modeling method, implementing it in a meta-CASE tool and testing it in two pilot projects, we have been able to find risk areas that are controllable and risk areas, that are not affected by the use of the tool.

The comparison of the previous case projects and the pilot projects revealed that the use of the modeling method mostly influenced general RAD-project risk factors such as "lack of adequate design documentation" and "continuing stream of requirement changes", as hypothesized in this study as well as "misunderstanding the requirements". These risk factors are located in the "scope and requirements" and "execution" quadrants in the risk categorization framework presented by Keil et al. (1998). These risk factors are characterized by the high level of perceived controllability by the project manager and high level of perceived level of importance, suggesting that the use of the NotesEdit –tool helps manage the critical risk factors from the project manager's viewpoint.

To gain a broader control to the RAD-project risk factors, some issues became apparent during the study: first, the need for two-way communication and interdependence between the graphical model of the application and the application design itself. The project managers found it difficult to revise graphical models while the prototype design was changed. The changes implemented in the code should be automated in the tool so that the model would be updated according to made changes and vice versa.

Second, the application frame generator (ie. automatic code-generation) is essential for developing the prototype frame quickly from the specifications. With the tool developed during this study, the need to code the actual application according to the graphical model was seen as an unnecessary step. For the application frame generator to work effectively, the need for two-way communication between the model and the application design was obvious.

Third, a need for an easy-to-use component repository was recognized. Even though the MetaEdit CASE tool features an object repository in the form of an "repository project", the reusable components should be easy to incorporate into application design.

Finally, the lack of layout design capabilities with the model restrict the quick usability of the method. When components are reused, there should be initial layout settings, eliminating the need for redesigning the visual layout again.

To summarize the most important qualities needed to achieve higher control of general risk factors when using a CASE-tool in a RAD-project:

1. Automatic code generation
2. Two-way communication between the model and the application design
3. Component repository with layout design capabilities

The tool used in the two pilot projects offered the development team a way to effectively share the requirement specifications within the team. Without features presented above, the use of the graphical modeling method helps in sharing specifications among developers, helps in clarifying and conceptualizing the requirement specifications supplied by the customer and aids in understanding complex interrelationships within the application design. The technicality of the method restricted its use with communication between the developer and the users.

6.1 Further research

The use of CASE-tools in RAD-projects has not received very much attention, especially from the viewpoint of risk management. The prototype approach is traditionally seen as a risk management technique in situations, where the requirement specifications might not be clear or are virtually non-existent. However, when using a prototype approach with RAD, some risk factors are emphasized and need additional means of control. In this thesis, the area of using a graphical modeling method with RAD-projects was examined with a risk management approach in mind. Further research in this area is warranted for example by examining the developer-user communication using the modeling method, which proved problematic in this case. In addition, a broader area of researching i.e. inherent and completely new risk factors associated with both prototyping approach and the rapid application development would be areas of great interest.

REFERENCES

Aaen, I., Siltanen, A., Sorensen, C., Tahvanainen, V-P., (1992) A Tale of Two Countries: CASE Experiences and Expectations. In: *Proceedings of the IFIP WG8.2 Working Conference on The Impact of Computer Supported Technologies on Information Systems Development* (eds. K. Kendall, K. Lyytinen, J. DeGross), North-Holland, pp. 61-93

Aaen, I., Sorensen, C., A CASE Of Great Expectations, Scandinavian Journal of Information Systems, Vol.3, pp.3-23, 1991

Barki, H., Rivard, S., Talbot, J., (1993) Toward an Assessment of Software Development Risk, *Journal of Management Information Systems*, Vol. 10, No. 2, pp. 203-225

Barnes, B., Bollinger, T., (1991) Making Reuse Cost-Effective, *IEEE Software*, Vol. 8, No. 1, pp. 13-24

Boehm, B., (1988) A Spiral Model of Software Development and Enhancement, *Computer*, Vol. 21, No. 5, pp. 61-72

Boehm, B., (1989) Theory-W Software Project Management: Principles and Examples, *IEEE Transactions on Software Engineering*, Vol. 15, No. 7, pp. 902-916

Boehm, B., (1991) Software Risk Management: Principles and Practices, *IEEE Software*, Vol. 8, No. 1, pp.32-41

Boehm, B., (1995) Anchoring the Software Process, *http://sunset.usc.edu/TechRpts/ Papers/usccse95-507/ASP.html* (Referred 05/10/1999).

Boehm, B., Papaccio, P., (1988) Understanding and Controlling Software Costs, *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, pp. 1462-1477

Boehm, B., Ross, R., (1989) Theory-W Software Project Management: Principles and Examples, *IEEE Transactions on Software Engineering*, Vol. 15, No. 7, pp. 902-916

Brinkkemper, S., (1996) Method engineering: engineering of information systems development methods and tools, *Information and Software Technology,* Vol. 38, No. 4, pp. 275-280

Budde, R., Kautz, K., Kuhlenkamp, K., Züllighoven, H., (1992) *Prototyping: an approach to evolutionary system development*, Springer-Verlag, Berlin

Charette, R., (1989) *Software Engineering Risk Analysis And Management*, McGraw-Hill, New York

Coad, P., Yourdon, E., (1990) *Object-Oriented Analysis*, Prentice-Hall, New Jersey

Cotterell M.,Hughes, B., (1995) Software Project Management, An International Thomson Publishing Company, London

Gibson, M., Snyder, C., Rainer JR., K., (1989) CASE: Clarifying Common Misconceptions, *Journal of Systems Management*, Vol. 40, No. 5, pp. 12-19

Grudin, J., (1990) Groupware and Cooperative work: Problems and Prospects. In: *Readings in Groupware and Computer-Supported Cooperative Work Assisting Human-Human Collaboration* (eds. R. Baecker), Morgan Kaufmann Publishers Inc. (1993) pp. 97-105.

Heym, M., Österle, H., (1993) Computer-aided methodology engineering, *Information and Software Technology*, Vol. 35 No. 6/7, pp. 345-354

Hillesberg van, J., Kumar, K., Welke, R., J., (1998) Using metamodeling to analyze the fit of object-oriented methods to languages. In: *Proceedings of the 31st Hawaii International Conference on System Sciences*, Volume V, (eds. R. Blanning, D. King) IEEE Computer Society, pp. 323-332.

Jarke, M., Pohl, K., (1992) Information Systems Quality and Quality Information Systems. In: *Proceedings of the IFIP WG8.2 Working Conference on The Impact of Computer Supported Technologies on Information Systems Development* (eds. K. Kendall, K. Lyytinen, J. DeGross), North-Holland, pp. 345-375

Jarzabek, S., Huang, J., (1998) The Case for User-Centered CASE Tools, *Communications of the ACM*, Vol. 41, No. 8, pp. 93-99

Keil, M., Cule, P., Lyytinen, K., Schmidt, R., (1998) A Framework for identifying software project risks, *Communications of the ACM*, Vol. 41, No. 11, pp. 76-83

Khosrowpour, M., (1993) *Computer-aided software engineering: Issues and trends for the 1990s and beyond*, Idea Group Publishing

Krief, P., (1996) *Prototyping with objects*, Prentice Hall, New Jersey

Kumar, K., Welke, R.J., (1992) Methodology engineering: a proposal for situation-specific methodology construction. In: *Challenges and Strategies for Research in Systems Development* (eds. W.W. Cotterman, J.A.Senn), John Wiley & Sons Ltd, pp.257-269

Larson-Hughes, R., Skalle, H., (1995) *Lotus Notes Application Development: Solving Business Problems and Increasing Competitiveness*, Prentice-Hall, New Jersey

Lyytinen, K., Kerola, P., Kaipala, J., Kelly, S., Lehto, J., Liu, H., Marttiin, P., Oinas-Kukkonen, H., Pirhonen, J., Rossi, M., Smolander, K., Tahvanainen, V-P., Tolvanen, J-

P., (1994) MetaPHOR: Metamodeling, Principles, Hypertext, Objects and Repositories, http://www.jyu.fi/~kelly/meta/loppurap/ (Referred 22/11/1999).

Necco, C., Tsai, N., Holgeson, K., (1989) Current Usage of Case Software, *Journal of Systems Management*, Vol. 40, No. 5, pp. 6-11

Nierstrasz, O., Gibbs, S., Tsichritzis, D., (1992) Component-Oriented Software Development, *Communications of the ACM*, Vol. 35, No. 9, pp. 160-165

Norman, R. J., Nunamaker J.F. Jr., (1988) An empirical study of information systems professionals' productivity perceptions of CASE technology. In J. DeGross and M. Olson, editors, *Proceedings of the Ninth International Conference on Information Systems*, pp. 111-118.

Norman, R. J., J.F.Nunamaker Jr., (1989) Integrated development environments: Technological and behavioral productivity perceptions. In *Proceedings of the 22th Annual Hawaii International Conference on System Sciences, vol II, Software Track*, pp. 996-1003

Nunamaker, R. J., Nunamaker J.F. Jr., (1989) CASE Productivity Perceptions of Software Engineering Professionals, *Communications of the ACM*, Vol. 32, Number 9

Olle, T., SolH., Verrijn-Stuart A., (1983) Information systems design methodologies: A feature analysis, North Holland, Amsterdam.

Orlikowski, W., (1993) CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development, *MIS Quarterly*, Vol. 17, No. 3, pp. 309-340

Overmyer, S., (1990) The Impact of DoD-Std-2167A on Iterative Design Methodologies: Help or Hinder?, http://faculty.cis.drexel.edu/~overmyer/papers/2167a/2167pap.htm (Referred 05/12/1999).

Rossi , M., Brinkkemper, S., (1996) Compexity Metrics For Systems-Development Methods And Techniques, *Information Systems*, 21, 2, pp. 209-227

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., (1991) Object Oriented Modelling and Design. Prentice Hall, Engelwood Cliffs, New Jersey

Smolander, K., Lyytinen, K., Tahvanainen, V-P., Marttiin, P., (1991) MetaEdit - A Flexible Graphical Environment for Methodology Modelling. In: *Proceedings of Third International Conference on Advanced Information Systems Engineering (CAiSE '91)* (eds. R. Andersen, J. Bubenko jr., A. Solvberg) Springer-Verlag, pp. 168-193

Taylor, M., Wood-Harper, A., (1996) Methodologies and Software Maintenance, *Software Maintenance: Research and Practice*, Vol. 8, pp. 295-308

Tolvanen, J-P., (1998) *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence*, (Dissertation). Jyväskylä Studies in Computer Science, Economics and Statistics, No. 47, University of Jyväskylä

Tolvanen, J-P., Lyytinen, K., (1993) Flexible method adaptation in CASE – the metamodeling approach. *Scandinavian Journal of Information Systems*, Vol. 5, pp. 51-77

Tolvanen, J-P., Rossi, M., Liu, H., (1996) Method Engineering: Current research directions and implications for future research. In: *Proceedings of the IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering* (eds. S. Brinkkemper, K. Lyytinen, R. Welke), Chapman & Hall, pp. 296-317

Vessey, I., Sravanapudi, A., (1995) CASE Tools as Collaborative Support Technologies, *Communications of the ACM*, Vol. 38, No. 1, pp. 83-95

Wasserman, A. (1980) Information System Design Methodology. In: *Tutorial on Software Design Techniques* (eds. P. Freeman, A. Wasserman), IEEE Computer Society, pp. 25-44

Wijers, G. M., v Dort, H.E., (1990).Experiences with the use of CASE tools in The Netherlands. In *Advanced Information Systems Engineering*, (eds. B. Steinholz *et al.*) pp. 5-20.

APPENDIX 1: DESCRIPTION OF NOTESEDIT CASE TOOL

In this appendix we describe the properties of NotesEdit CASE tool in detail. The tool was developed during this research and it consists of MetaEdit+ metaCASE tool supporting the developed *Notes Design Language* (NDL) method and the *GodeGenie* Lotus Notes application frame generator. Application frame generator is a separate application designed to generate Lotus Notes application frames according to the meta-language descriptions generated from NDL models in MetaEdit+.

NDL method engineering process

The NDL method engineering process was implemented by describing the NDL method using metamodels. In our NDL metamodeling effort we applied the Graph-Object-Property-Relationship-Role (GOPRR), which has been developed specially for metamodeling (Tolvanen 1998). The GOPRR language was selected due to the available tool support and the previous experiences of the people involved in metamodeling and CASE tool implementation effort.

Tolvanen (1998) suggests the following set of tasks that GOPRR related metamodeling must follow:

1. Identification of the techniques in the method
2. Identification of the object types
3. Determination of properties for each object type
4. Determination of relationships
5. Determination of roles
6. Allocation of properties to relationship types and roles
7. Determination of metamodels for individual techniques
8. Determination of linkages between separate techniques
9. Determination of the representational part of the method
10. Analysis and evaluation of the method

Since Tolvanen (1998) reports of several successful metamodeling efforts conducted applying the tasks specified above (cf. Tolvanen and Lyytinen 1993, Rossi and Brinkkemper 1996, Hillegersberg et al. 1998), we decided to use the list as a guideline for our metamodeling process. The results of each phase are reported below.

Identification of the techniques in the method

The NDL method consists of two modeling techniques, *Application architecture model* and *Database model*. The techniques are designed to model the object system on different levels of abstraction.

Application architecture model is used to model the structure of Lotus Notes applications (which applications consist of which databases) and the connections between separate Lotus Notes and Open Database Connectivity (ODBC) compliant databases. Application architecture models can also be used to represent the whole Lotus Notes application architecture of a particular organization.

By hiding the internal structure of the databases represented in Application architecture models, the connections among different components of Lotus Notes applications can be visualized on a higher level of abstraction. The databases represented in Application architecture models can be exploded to Database models.

Application architecture modeling technique is targeted at supporting the structuring of the application consisting of multiple databases. The connections and interfaces between the separate databases can be modeled and analyzed utilizing the application architecture models. Thus, the models can be utilized in enhancing communication among the system developers developing different system modules, as well as in demonstrating the system structure for the end users. Additionally, the models can be used in analyzing effects of design changes on the application level (instead of database level as in database models). Thereby, the application architecture models can be utilized in the system maintenance phase when updating or modifying the application functionality.

Database model is a technical description of a particular Lotus Notes Database and thus cannot be used to model ODBC compliant databases. Database models consist of Lotus Notes objects (e.g. forms and views) and connections between them. Connection from a particular object to an external database is also allowed. These connections should be modeled accordingly in Application architecture models.

Database modeling technique is designed to support the system design activities as well as the automatic application generation according to database models. The database models generated during the development can be used in enhancing the application design documentation and the analysis of the design changes in database level. The objects, properties and connections presented in Database model are converted to a semi-functional Lotus Notes application frame by utilizing the CodeGenie application frame generator.

Both of the presented techniques can be utilized in the requirement specification phase to formalize the requirement specifications and support the tracking of the specification changes. Additionally, as the system can be modeled according to the requirement specifications, we attempt to support the estimation of the resources needed for the project completion.

Identification of the object types

Application architecture modeling technique supports two object types: *Application* and *Database*.

Application object type is an abstract concept representing an application, or an information system that can be considered as one entity. Application consists of one or more Lotus Notes or ODBC compliant databases.

Database object type is used to represent Lotus Notes or ODBC compliant databases that are components of applications. *Database* object can be exploded to a Database model representing the technical design of a specific database.

Database modeling technique consists of nine object types. Additionally *Database* object type specified in the Application architecture modeling technique can be utilized in Database models to represent external database connections.

In Database models five of the object types (*Form*, *View*, *Agent*, *Navigator*, and *Database*) have a graphical representation and the rest five (*Field*, *Action*, *Button*, *Column* and *Code*) are not visible entities as they are embedded as properties in the object types. Hiding the component objects (objects that are properties of other objects) allows us to represent the design of a whole database in one graph without reducing the readability of the graphical representation.

Since the Database modeling technique is designed to support automatic application generation, all the object types represent the physical building blocks of Lotus Notes databases.

Form is one of the visible object types in Database models. It represents the Form component in Lotus Notes databases. *Form* objects can contain (as properties) one or more *Field*, *Action*, *Button* and *Code* objects.

View object type is used to represent the View component in Lotus Notes databases. *View* objects have a graphical representation in database models and they can contain one or more *Column* and *Action* objects and one *Code* object representing the document selection formula for the view.

Agent object type represents the Agent component in Lotus Notes databases. *Agents* are visible objects in Database models since they usually tend to have a number of relationships with other objects. *Agent* object can contain one *Code* object.

Navigator object type represents the Navigator component in Lotus Notes databases. *Navigator* objects are visible in Database models and they can contain one or more *Button* objects.

Code is an object containing one specific program entity written in LotusScript or Lotus @Formula programming language. There is no actual Lotus Notes component corresponding to *Code* object type, since *Code* objects can be considered as being any piece of program code in a Lotus Notes application (e.g. field or column formula, LotusScript program associated to agent). *Code* is a non-visible object and cannot exist alone in Database models thus it is always contained in another object as a property. The purpose of *Code* object type is to support sharing of program code between other design components.

Field, Action, Button and *Column* object types all represent the corresponding Lotus Notes objects according to the object type name. They are all contained as a property in one or more visible objects in Database models and are therefore referred to as *component objects*. Component objects can contain only *Code* objects.

It is worth noticing that even though the component objects do not have graphical representations they can have relationships with other object types. The issues related to relationships between object types are discussed further below. The properties and relationships of the component objects can be examined using the reporting capabilities (e.g. role and relationship tables) of the CASE tool supporting the NDL method.

Determination of properties for each object type

The object types used in NDL method have two kinds of properties: properties representing the attributes of Lotus Notes components and properties used to support the modeling process, documentation and automatic application frame generation.

The *Application* and *Database* object types used in Application architecture models have the properties presented in Table 1.

Table 1: Properties of Application architecture object types

OBJECT TYPE	**PROPERTIES**
Application	Name Type
Database	Title Type Server Filename ReplicaID

None of the *Application* object type properties exist in the actual Lotus Notes applications since the object type is an abstract representation for multiple Lotus Notes databases considered as one application or information system. However, all the properties of *Database* object type represent the properties of Lotus Notes databases.

The properties of the object types representing Lotus Notes objects were determined by selecting properties of corresponding Lotus Notes components needed for application frame generation. After determining the relevant properties, *Comment* property was added for each object type to support object documentation. *Comment* property is a text string used to describe an object.

In Table 2 we present the properties of Database modeling technique object types. The object types representing Lotus Notes objects (i.e. the object types having the same properties as the corresponding Lotus Notes components) are displayed using italicized typeface. *Comment* properties are not included in Table 2 since they are common for all object types.

Table 2: Properties of Database modeling technique object types

OBJECT TYPE	PROPERTIES
Form	Name Alias Type Defaults Fields (list of Field objects) Actions (list of Action objects) Buttons (list of Button objects) WindowTitle (Code object) QueryOpen (Code object) WebQueryOpen (Code object) QuerySave (Code object) WebQuerySave (Code object) QueryClose (Code object) WebQueryClose (Code object)
View	Name Alias Columns (list of Column objects) Code (Code object) Style Options Type RowSpacing
Navigator	Name Buttons (list of Button objects) InitialViewOrFolder (View object)
Agent	Name WhenShouldThisAgentRun Code (Code object)
Field	Name Datatype ComputingMethod HelpText IsShared Value (Code object) InputValidation (Code object) InputTranslation (Code object) HideWhen (optional Code object)
Action	Title Position IncludeIn Code (Code object)
Button	Label Code (Code object)
Code	Name Comment Type Value

Some of the identified properties can actually be considered as events of Lotus Notes components (e.g. QueryOpen and QuerySave). The NDL modeling technique handles these events as properties since the *Code* objects (i.e. executable program code) associated to the events can be assigned to the particular property to be executed when the event occurs. Implementation of these programs as individual *Code* objects allows us to share program code between different object types.

The variation of the property names referring to same kind of a property (e.g. Title and Label properties of *Action* and *Button* objects) are due to the diversity of Lotus Notes component property names. All the properties of object types representing Lotus Notes objects are named after the corresponding Lotus Notes component properties in order to ease the familiarization of a Lotus Notes developer in using the metamodel.

Determination of relationships

The relationships used in NDL method are used to describe a variety of connections between different object types. All relationship types can exist in both Application architecture models and Database models although connecting different object types. The relationships are used to represent the composition and functionality of applications and to support the generation of Lotus Notes application frames.

Each relationship type has a direction and a particular set of object types that can exist in each end of the relationship. Since not all the object types are visible in graphical models the relationships referring to the hidden component objects must be created by drawing a relationship by using the container object (i.e. the object having the component object as a property). When the relationship type and the source and the destination objects are selected the CASE tool presents the list of the component objects suitable for the specified type or relation. The identifiers (i.e. names) of associated objects are displayed in each end of the relation to indicate the hidden objects linked to the relationship.

All the relationship types supported in NDL method are presented in Table 3.

Table 3: NDL method relationships

SOURCE OBJECT TYPE	**RELATIONSHIP TYPE**	**DESTINATION OBJECT TYPE**
Application	*Aggregation*	Database
Form	*Aggregation*	Form
Action	*Compose*	Form
	Open	Database, View, Navigator
	Run	Agent
Button	*Compose*	Form
	Open	Database, View, Navigator
	Run	Agent
Field	*DbLookup*	Column
	DbColumn	Column
Column	*Display*	Field

Aggregation relationship is used to represent the "consists of" type of relationship, such as "*Application* consists of *Databases*". The *Form – Form* aggregation refers to the subform concept of Lotus Notes environment where one form entity can consist of one or more other forms referred to as subforms.

Compose relationship type represents the creation of new Lotus Notes document by using a particular form component in Lotus Notes database. Documents are always created using a specified form and the name of the form connects the form to the document. Forms can also be referred to by using the alias property. By representing the document creation as a relationship in Database models the changes in Name and Alias properties of the *Form* objects can be reflected to the referring *Action* and *Button* objects which prevents outdated name references. The Compose relationships can be automatically transferred to the actual program code implementing document creation by CodeGenie application frame generator.

Open relationship type is used to describe the opening of a Lotus Notes database or a particular view or navigator component in a specified Lotus Notes database. As in case of Compose relationships the changes in relevant properties of the associated objects can be reflected accordingly. According to the properties of associated objects the relationship can be transferred to the corresponding program code in CodeGenie application frame generator.

Run relationship represents the execution of an agent in a specified Lotus Notes database. The agent is referred to according to its name property. The Run relationships can also be transferred to the corresponding program code in application frame generator.

DbLookup and *DbColumn* relationships represent two different ways of referring to data in Lotus Notes databases. Both relationship types refer to the corresponding Lotus Notes @formula language functions (@DbLookup and @DbColumn). The *Field* object in the source end of the relationship is used to store the return value of the function and the *Column* object specifies the view column where the value or values are retrieved. As the @DbLookup function needs a search key for looking up the specific value, an additional *Code* object is linked as a property to the DbLookup relationship. The *Code* object is used to store the search key that can be a text string, a set of program code returning a text string or name of a particular *Field* object referring to the value of the field. The DbLookup and DbColumn relationships are transferred to field formulas in the application frame generator.

Display relationship type is used to specify the composition of views i.e. which fields of which forms are displayed in which view columns. The Display relationships can be transferred to column @formulas in the application frame generator.

Determination of roles

The roles are used to represent the roles of the objects associated in a particular relationship type. Typical examples of roles are the subclass and superclass roles used in object oriented methods.

In the NDL method the relationship type and direction are used to describe the purpose of the relationship and the roles of the associated objects. Since no additional information can be offered by specifying different role types for each relationship type, all role types are referred to as Source and Destination according to the direction of the particular relationship. As some of the object types bound to relationships are not visible in graphical models, the associated object identifiers (e.g. names) are displayed in each end of the relationship instead of the role types.

The cardinality of a relationship type defines the minimum and maximum number of role type instances a relationship type can have. In NDL method the cardinality varies according to the relationship type. The cardinalities of NDL relationship types are represented in Table 4.

Table 4: Cardinalities of NDL relationship types

Relationship type	**CARDINALITY**
Aggregation	1, M
Compose	M, 1
Open	M, 1
Run	M, 1
DbLookup	M, 1
DbColumn	M, 1
Display	M,M

The cardinalities of the relationship types are specified according to the possible relationships between objects in Lotus Notes environment. Lotus Notes environment also sets some additional restrictions which are not implemented in the modeling method due to the complexity of the rules associated with these issues (e.g. in a Lotus Notes application a subform component cannot include another subform, but the NDL method does not restrict successive form aggregations). These restrictions should be handled with the reporting facilities of the CASE tool supporting the NDL method.

Allocation of properties to relationship types and role types

Information related to the connections between object types is presented as properties of relationship types. In the NDL method the DbLookup relationship is the only relationship type having additional properties. The @DbLookup function in Lotus @Formula language requires a search key for looking up the data from Lotus Notes databases. In the NDL method the search key is defined by attaching a *Code* object as a property for the DbLookup relationship.

Since all the NDL relationship types have the same role types (Source and Destination), no additional properties have been allocated for the role types.

Determination of metamodels for individual techniques

The metamodels of the individual techniques represent all the possible connections between the object, relationship and role types in the specified techniques. Also the cardinality of the connections can be specified. (Tolvanen 1998)

The NDL metamodels were developed using GOPRR metamodeling language. Metamodeling was an iterative process in which prototypes of different techniques were developed, tested and compared. After determining the individual techniques for the method, the metamodels of selected techniques were replenished according to the test results and the CASE tool support was implemented to support further testing. To support the CASE tool implementation some additional information about the metamodel components (e.g. datatypes of object types) was attached to the metamodels as free-form descriptions.

Despite the expressive power of the GOPRR metamodeling language, all the requirements for the methodology could not be described in the metamodels. However the requirements not specified in metamodels were attached to the method documentation as a list of error checking rules and warnings that can be utilized when implementing CASE tool support for the methodology. As most of the typical CASE tools have capabilities of checking the generated models according to specified rules, the necessary error checking and generation of warning reports can be managed by the CASE tool supporting the NDL method.

Determination of linkages between separate techniques

As the Application architecture modeling technique and the Database modeling technique are used to model the same object system on different abstraction levels, there must be rules to ensure the integrity of the models generated using different techniques. These rules are utilized when implementing the model integrity checking in the CASE tool supporting the method. The model integrity must be checked before application frames can be successfully generated.

The Database models are used to represent the internal structure of the databases specified in Application architecture models. As the objects in Database models can have relationships with objects in external databases (i.e. in different Database models) these relationships should also exist in the associated Application architecture models as connections between specific *Database* objects and in all the relevant Database models as external database connections.

The object types bound to the external relationships are determined by the relationship type. For example, if a *DbColumn* type of relationship is specified between two database objects in Application architecture model, the external connection in the Database model representing the source database must be modeled as a *DbColumn* type of connection from a *Field* object to a *Database* object representing the destination database. Accordingly the relationship must exist also in the Database model representing the destination database as a *DbColumn* type of connection from *Database* object to a *Column* object. This enables the connections between objects in separate Database models. An example of a *DbColumn* relationship is presented in Figure 1.

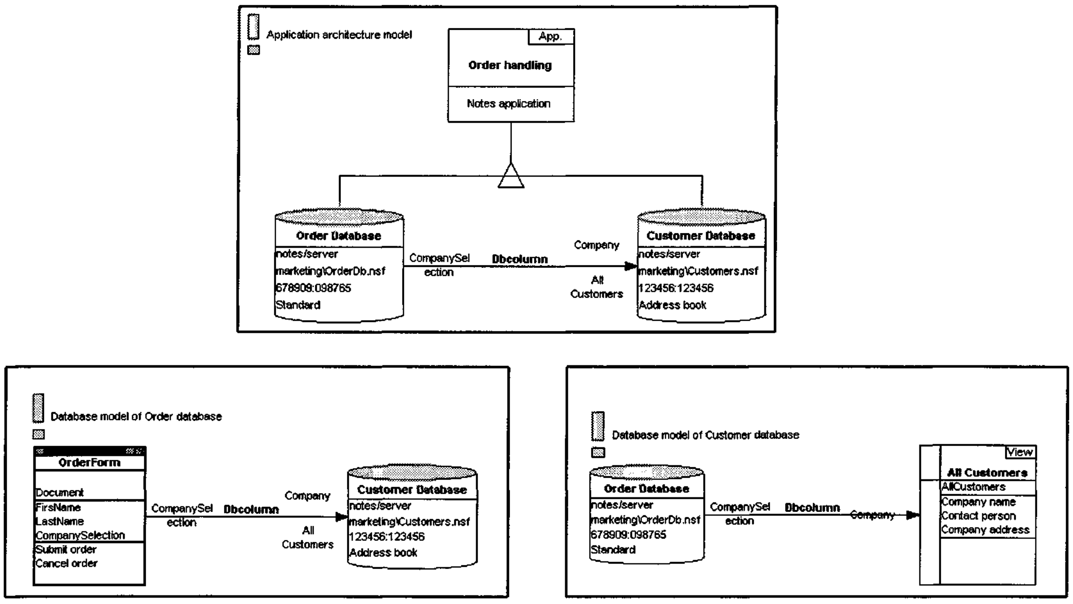

Figure 1: Representation of DbColumn relationship to external database

All the relationship types except *Display* relationships can exist in external connections. The restriction of the *Display* relationship use is due to the functionality of column component in Lotus Notes environment.

Determination of the representational part of the method

Since the NDL method is designed for use with a CASE tool support, the specification of a graphical notation for the method is necessary. The graphical notation includes the representation of the object types, relationship types and role types. The representation aspect in a modeling tool also includes the specification of dialogs, menus, and toolbars, but since these issues have only little to do with the actual modeling method we decided not to describe them in this context.

The NDL models are represented as graphical diagrams where each object type has a unique symbol. The object identifier is displayed at the top edge of each symbol to distinguish the individual objects. Since all the relationships have a direction, the relationships are represented as arrows from the source object to the destination object.

The relationship type is displayed above the arrow. If the objects bound to the relationship are component objects (not visible in graphical representation), the object identifiers are displayed in each end of the arrow (Figure 3).

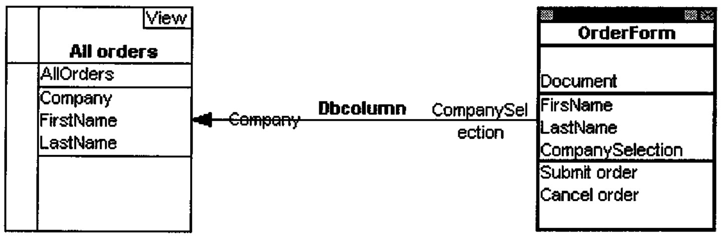

Figure 2: Graphical representation of DbColumn relationship

Aggregation relationship type representation differs from other relationship types since the common notation of aggregation is used. The aggregation relationship is represented with a line-triangle combination presented in Figure 3. The presented representation of aggregation relationship type is used for example in Object Modeling Technique (OMT) (Rumbaugh et al. 1991) and Object Oriented Analysis / Object Oriented Design (OOA/OOD) (Coad et al. 1990) methodologies.

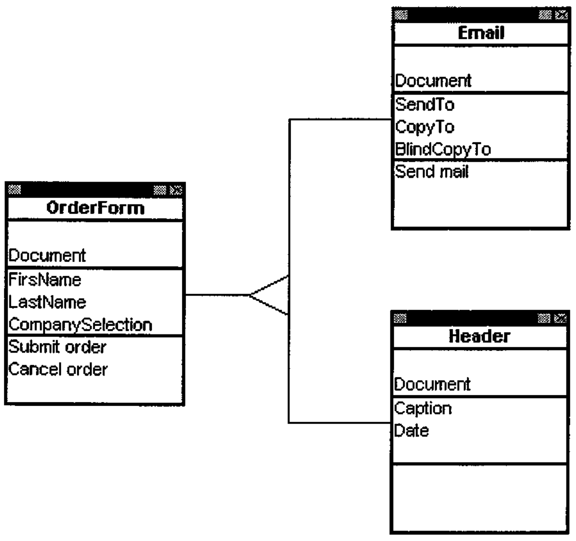

Figure 3: Graphical representation of Aggregation relationship

The exclamation mark is used as a symbol for additional textual information embedded in graphical models.

Analysis and evaluation of the metamodel

In this section we discuss the limitations of the generated metamodels and the GOPRR metamodeling language. As the selection of the metamodeling language was due to the available tool support rather than the properties of the language we do not discuss the alternative language choices in this context.

Majority of the identified limitations become apparent in the Database modeling technique metamodel being due to the complexity of the Lotus Notes application design rules. As the Lotus Notes environment has several property-related restrictions (e.g. if Field.ComputingMethod value is 'Computed', you cannot specify Field.InputValidation property), the description of all possible property-rule combinations would have caused the metamodels to become too complex. However the property-rule dependent restrictions can be implemented as error-checking reports in most typical CASE tools.

Another property-related restriction in the generated metamodels is the representation of unique properties. The Lotus Notes environment emphasizes the need to model unique properties (e.g. Form cannot have two Field objects with same Name property). The need of uniqueness rules exists especially in Database modeling technique where the object property lists are used in several object types. Modeling these situations would require additional constructs in the metamodeling language.

A major limitation of the whole NDL modeling method is its narrow focus on the factors concerning the Lotus Notes application development. Even though the main goal of the NDL development was to support Lotus Notes application development the method still lacks connections to the real-world environment often modeled using Business Process Re-engineering (BPR) models, Information flow charts, and Entity relationship models. As one of the method development objectives was to support the requirement specification phase of application development, integration to a technique used to model the organization is necessary. Additionally the method also lacks means for demonstrating the application functionality as it is completely focused on the internal design structure of the application. However in RAD projects the demonstration of the application behavior with a graphical model is not necessary relevant since the demonstration can be better implemented using an application prototype.

Application frame generator description

The application frame generator is separate application which acts as a "translator" between the metalanguage produced by MetaEdit and Lotus Notes environment. Within the scope of this research, the application frame generator was not developed to its full extent, only a technology demonstration was built which proved the technical feasibility of the concept. The developed generator was named *CodeGenie.*

CodeGenie is essentially an application that reads the metafile produced by MetaEdit and creates the specified design components into a Lotus Notes database. CodeGenie was developed with Microsoft Visual C++ 6.0 using the Lotus Notes C API version 4.6.2. The technology demonstration that was developed was built as a simple console application, without a graphical user interface. A simple report was built using the MetaEdit report generator which produces a metalanguage description of a specified database in the application architecture model. The demonstration report and application frame generator only supported form –design components.

In order to support the graphical modeling method in its full potential, the application frame generator and report engine must support all design elements in the method. The developer also needs a way to control the dependant changes in the Lotus Notes database and the corresponding model. The developer should be able to control the updates per design component basis. This can be accomplished with a graphical representation of the design structure where the designer can designate components that are to be updated. The tools for comparing two databases are also a necessary so that the designer can analyze the differences between two databases, for example two different versions of a prototype.

The implementation of a component repository is also possible to integrate into the application frame generator. The lack of layout design within the graphical modeling method makes this a viable option because in this method, layout information can be saved with the components saving the designer the work of constructing the layout again. The developer simply selects the design components he wishes to save into the repository and supplies necessary information about the use of the component.