Helsinki University of Technology
Systems Analysis Laboratory Research Reports A85, May 2003

MODELLING AND IMPLEMENTATION ISSUES IN CIRCUIT AND NETWORK PLANNING TOOLS

Jukka K. Nurminen

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology

Systems Analysis Laboratory Research Reports A85, May 2003

MODELLING AND IMPLEMENTATION ISSUES IN CIRCUIT AND NETWORK PLANNING TOOLS

Jukka K. Nurminen

Dissertation for the degree of Doctor of Technology to be presented with due permission for public examination and debate in Auditorium E at Helsinki University of Technology, Espoo, Finland, on the 11th of June, 2003, at 12 o'clock noon.

Helsinki University of Technology
Department of Engineering Physics and Mathematics
Systems Analysis Laboratory

Distribution:
Systems Analysis Laboratory
Helsinki University of Technology
P.O. Box 1100
FIN-02015 HUT, FINLAND
Tel. +358-9-451 3056
Fax +358-9-451 3096
systems.analysis@hut.fi

This report is downloadable at
www.sal.hut.fi/Publications/r-index.html

ISBN 951-22-6551-6
ISSN 0782-2030

Otamedia Oy
Espoo 2003

Title: Modelling and Implementation Issues in Circuit and Network Planning Tools

Author: Jukka K. Nurminen
Nokia Research Center
P.O. Box 407, FIN-00045 NOKIA GROUP, Finland
jukka.k.nurminen@nokia.com

Date: May 2003

Publication: Systems Analysis Laboratory Research Reports A85, May 2003

Abstract: This thesis consists of studies of modelling and implementation issues for planning tool development. In particularly it emphasizes issues that are relevant for practical, industrial use of mathematical models and algorithms. Using a circuit design and a network planning tool as concrete examples the thesis analyses the practical issues of the development and maintenance of such systems and, especially, how models and algorithms affect these. An important part of the thesis is observations of the changes in the software products over time and analysis what effect and requirements the evolution has for the models and algorithms

The thesis deals with implementation in a wide sense. In addition to the issues of the implementation of a given algorithm this work discusses algorithm selection, development process, users' role, software evolution and how these affect each other. In particular the thesis tries to answer three questions: (i) What kind of models and algorithms to use in the tools? (ii) How to divide the work between the computer and the user? (iii) How to develop such tools?

The main part of this research has been conducted using the participant observation research methodology by collecting and analysing experiences in the development of two commercial planning tool products. These results have been tied to the model of planning tool implementation that has been developed as part of this work. In particular the focus has been on how technology interacts with tasks, persons, and organizations. The results are also discussed within the context of established research disciplines such as operations research, software engineering, artificial intelligence, and problems solving environments.

One finding of this work is the practical significance of simple algorithms and other building blocks that are easy to implement and enhance. This is often more important than high accuracy or strict optimality. The work also suggests a practical way to measure and compare the implementation complexity of algorithms.

Another observation is how complex planning tasks should be divided between the computer and human expert. Examples how this is done using knowledge-technology, intelligent interfaces, and different kind of mathematical models are discussed.

Practical issues in the development and evolution of algorithms and software are examined in case studies of routing algorithms and of a network visualization software component. These studies highlight the importance of the incremental development approach and discuss its implications to algorithms and software modules.

Keywords: algorithm implementation, modelling, routing algorithms, intelligent interfaces, software evolution, network planning, circuit design

Academic dissertation

Systems Analysis Laboratory
Helsinki University of Technology

Modelling and Implementation Issues in Circuit and Network Planning Tools

Author:	Jukka K. Nurminen
Supervising professor:	Professor Raimo P. Hämäläinen, Helsinki University of Technology
Preliminary examiners:	Professor Pasi Tyrväinen, University of Jyväskylä, Finland Professor Tapio Westerlund, Åbo Akademi University, Finland
Official opponent:	Dr István Maros, Imperial College, UK

Publications

The dissertation consists of the present summary article and the following papers:

[I] Ketonen, T., Lounamaa, P., Nurminen, J. K., 1988. An Electronic Design CAD System Combining Knowledge-based Techniques with Optimization and Simulation. In: Gero, J. S. (Ed.) *Artificial Intelligence in Engineering: Design* (pp. 101-118). Elsevier Science Publishers. Amsterdam, Netherlands.

[II] Ketonen, T., Nurminen, J. K., 1989. A Knowledge-based Simulation Environment for Electronics Design. *Proceedings of the 3rd European Simulation Congress* (pp. 622-627). Edinburgh, Scotland.

[III] Nurminen, J. K., 1990. RFT Design System – Experiences in the Development and Deployment of a Lisp Application. *Proceedings of the First European Conference on the Practical Applications of Lisp* (pp. 183-191). Cambridge, UK.

[IV] Akkanen, J., Nurminen, J. K., 2001. Case Study of the Evolution of Routing Algorithms in a Network Planning Tool. *The Journal of Systems and Software* 58(3), 181-198.

[V] Nurminen, J. K., 2003. Using Software Complexity Measures to Analyze Algorithms – an Experiment with Shortest Paths Algorithms. *Computers & Operations Research* 30, 1121-1134.

[VI] Nurminen, J. K., 2003. Models and Algorithms for Network Planning Tools – Practical Experiences. Systems Analysis Laboratory Research Reports E14, Helsinki University of Technology.

[VII] Akkanen, J., Kiss, A. J., Nurminen, J. K., 2002. Evolution of a Software Component – Experiences with a Network Editor Component. *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering* (pp. 119-125). Budapest, Hungary.

[VIII] Nurminen, J. K., Karonen, O., Hätönen, K., 2003. What Makes Expert Systems Survive over 10 Years – Empirical Evaluation of Several Engineering Applications. *Journal of Expert Systems with Applications* 24(3), 199-211.

Contributions of the author

Papers [I-III] were created in the Radio Frequency Tool project 1987-94 at Nokia Research Center. The author was the project manager of the project and the primary author of papers [I and II]. In paper [III] the author was the sole contributor.

Papers [IV-VII] were created in the Transmission Network Planning Tool project 1992-2000 at the Nokia Research Center. The author was the project manager of the project and the primary author of paper [IV and VII]. In papers [V and VI] the author was the sole contributor.

For paper [VIII] the author was the primary author.

Acknowledgements

This thesis is a result of a long project and over the years there are numerous people who have influenced the work.

The creative environment at Nokia Research Center (NRC), my long-term employer, has been a cornerstone of this work. I would like to thank all of my colleagues especially within the Software Technology Laboratory of NRC for the friendly and intellectually stimulating atmosphere. Over the years the list of my closest colleagues and team members has grown to be so long that it is not possible to list all of them. In any case the cooperation with my colleagues in Helsinki, Budapest and elsewhere in the world has been a most important ingredient of this work. In particular I would like to thank my coauthors Dr. Jyrki Akkanen, Mr. Kimmo Hätönen, Dr. Olli Karonen, Dr. Timo Ketonen, Mr. Attila Kiss, and Dr. Pertti Lounamaa.

The execution of this work has been possible within the scope of two long tool development projects that have been funded by Nokia Mobile Phones and Nokia Networks. The business units have not only contributed to the funding of this work but also provided the highly relevant practical perspective. In particular the cooperation with Mr. Harri Korpela from Nokia Networks has been very important in this respect.

The encouragement and support of Dr. Olli Karonen from Nokia Research Center has been essential for the completion of this work.

I am grateful to Professor Sampo Ruuth who during my masters studies introduced me to academic research and who later pushed me forward with this thesis work. The advice from my supervisor Professor Raimo P. Hämäläinen has naturally been important for the successful execution of this work. I extend my gratitude to the two reviewers, Professor Pasi Tyrväinen of University of Jyväskylä and Professor Tapio Westerlund of Åbo Akademi University.

Finally I would like to thank my family for the joy and happiness that has helped to keep the worries related to the completion of this work in their proper scale.

Helsinki, May 2003

Jukka K. Nurminen

1 INTRODUCTION

This thesis focuses on circuit and network planning problems that have arisen in telecom industry and studies mathematical and software solutions to them.

Planning tasks are frequently so complex that handling them manually is not possible. Companies have thus become dependent on planning and design tools. As estimated by Gartner Dataquest the size of the worldwide system and network management market, which includes network planning tools, was over $7 billion in 2002 (Gartner, 2002a) and the market for electronic design automation was close to $3 billion in 2001 (Gartner, 2002b). These figures do not include the internal tool development within companies to satisfy special needs and to gain competitive advantage.

Efficient tool development is important for at least two major reasons. First, the tools have a crucial role in many businesses. With better tools a company can create better designs with less errors, faster, and with less manpower. Secondly, the amount of development resources available for tool development is limited. A key question is thus how to use the development resources to maximize the tool utility to the users.

There are many answers to the questions depending on the point-of-view. The software engineering people could argue that with better development processes better results can be achieved, computer scientists could argue that better algorithms, better programming languages or development tools are important, operations research specialists could argue that better models are the solution, and so on. All of the above statements are probably true but looking at the problem from one perspective only may not lead to globally optimal solution.

The standpoint of this thesis is that the current state-of-the-art in relevant disciplines is good for most problems in tool development. The essential problem is rather how to select among a multitude of possibilities the appropriate models, algorithms, tools, and processes to be used. For instance, should we use mathematical optimization models, numeric graph algorithms, or expert systems, or a combination of all three in a network planning tool?

Another problem is to decide what features to automate and what level of automation to target. Given a range of possibilities to increase computer support how can the developer choose the ones with the best payback. This question often manifests itself in the division of work between the computer and user. What automated features are most valuable for the users, and what is the optimal level of automation? Obviously it does not make sense to automate tasks that are simple for the users. Likewise it does not make sense to aim for fully automated solutions if the user can rather easily deal with those cases where full automation is highly difficult.

An important component in the development decision-making is time. Tools are seldom developed in one step. Instead they evolve gradually. How to handle the evolution, preserve the flexibility to deal with new feature requests and changing priorities, and ensure that users get the essential features at the right time? Can this be affected by proper technological choices? Are there some aspects of models and algorithms that make some of them more suitable for evolutionary development?

This thesis tries to provide some answers to the above practical problems. In particular, it focuses on three questions of efficient tool development:

1. What kind of models and algorithms to use in the tools?

2. How to divide the work between the computer and the user?

3. How to develop such tools?

In this thesis we are particularly interested in practice: what kind of issues are important for practical work and what kind of solutions seem to work in practice. Such results not only can serve the practitioner but also indicate areas and aspects that require more emphasis in theoretical studies.

The problem area is large and difficult for several reasons. First, tool development is interdisciplinary work requiring expertise from multiple disciplines such as computer science, mathematics, software engineering, and the problem domain of the tool. Secondly, empirical work over a long time period is needed to observe how things work in practice. Theoretical work or small-scale controlled experiments are not able to cover all important issues that arise in practical tool development projects. Thirdly, the problem is hard to divide into separate pieces. The relevant factors form a tightly connected network of relationships where change in one factor affects many others.

A planning tool consists of various parts. Kershenbaum (1993) divides a planning tool into four parts:

- The algorithms part takes care of the computing. It can be further split into subparts for design algorithms and for analysis algorithms. Design algorithms are able to synthesize new solutions or optimize the current ones. Analysis algorithms are able to estimate how a fixed plan will work and summarize the behavior in key performance figures. Besides numeric models and algorithms, the tool can contain also symbolic computing, such as knowledge bases and inference engines.

- The front-end part handles the user activity. In interactive planning tools it typically consist of a graphical user interface that enables the division of work between computer and human expert.

- The database part handles the storage of related data, bookkeeping of the design decisions, and interfaces to external systems that frequently act as sources of input data. The database part often consists of two subparts: the main memory data structures implementing the run-time data model and a secondary permanent storage such as a file system or a database management system.

- The utilities part consists of machine specific interfaces to operating system such as graphics libraries or file system access. The utilities part serves the other parts.

In this thesis we have mainly been concerned with the algorithms and front-end parts. Figure 1 illustrates how the papers forming this thesis are dealing with the different parts. Notice that most papers are located in the intersections of multiple parts. This is one consequence of the practical background of this work. In a commercial tool all of parts have to cooperate smoothly.

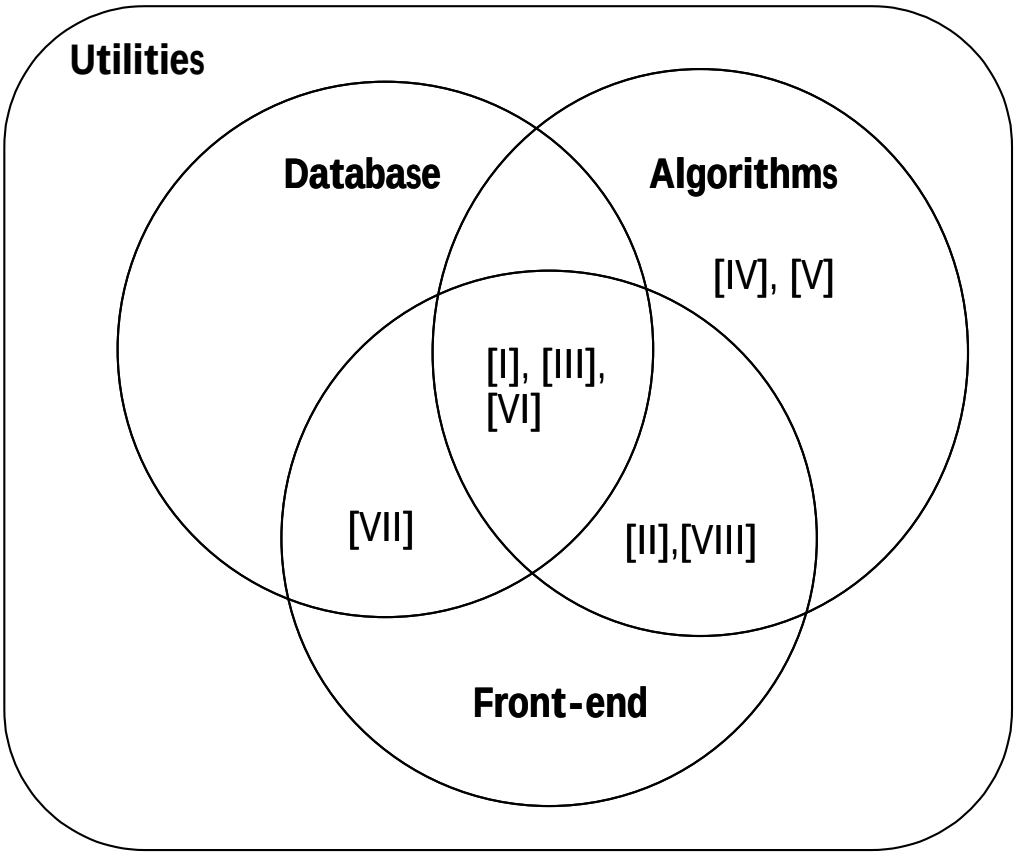

Figure 1 The components of a planning tool and the areas covered by the papers that form this thesis

This summary sets the thesis work to perspective. It is structured in the following way. Section 2 sets the theoretical background for this work by developing a model for planning tool implementation and use, by using the model to divide the research problem into subproblems, and by presenting the used research methodology. Section 3 describes the circuit design and the network planning tools that are used as concrete examples throughout this work. Section 4 gives an overview of the papers forming this thesis, summarizes the main results, and combines them using the model developed in section 2. Section 5 uses a different framework, the state-of-the-art of related disciplines, and discusses how this work contributes to them. Finally, section 6 presents some concluding remarks.

2 THEORETICAL FRAMEWORK

2.1 Leavitt's diamond model for planning tool implementation and use

Leavitt's diamond model (Leavitt, 1965) describes organizations as consisting of four interrelated components: tasks, technology, people, and structure, which refers to the organization as well as to external stakeholders such as competitors. The interdependence between different components is strong. When one component changes it influences the others. For example changes in technology affect the way individuals relate to the tasks they are responsible for performing as well as the organizational

structure. The model has been used as an analysis framework for e.g. information systems (Keen, 1981), IS personnel and their roles (Niederman & Trower, 1993), and telecommuting (Bui et al., 1996).

Leavitt's diamond provides a good framework for structuring the modeling and implementation issues discussed in this thesis. The components of the model can be used to classify different factors affecting the implementation. The interdependences between the components represent the complicated interactions between the factors.

User is an important stakeholder in the software development. However, the user perspective to the software development is very different from the developers. As a result we have chosen to divide the model into two views: one presenting the tool implementation (Figure 2) and one presenting the tool usage (Figure 3). The figures list typical items for each component.

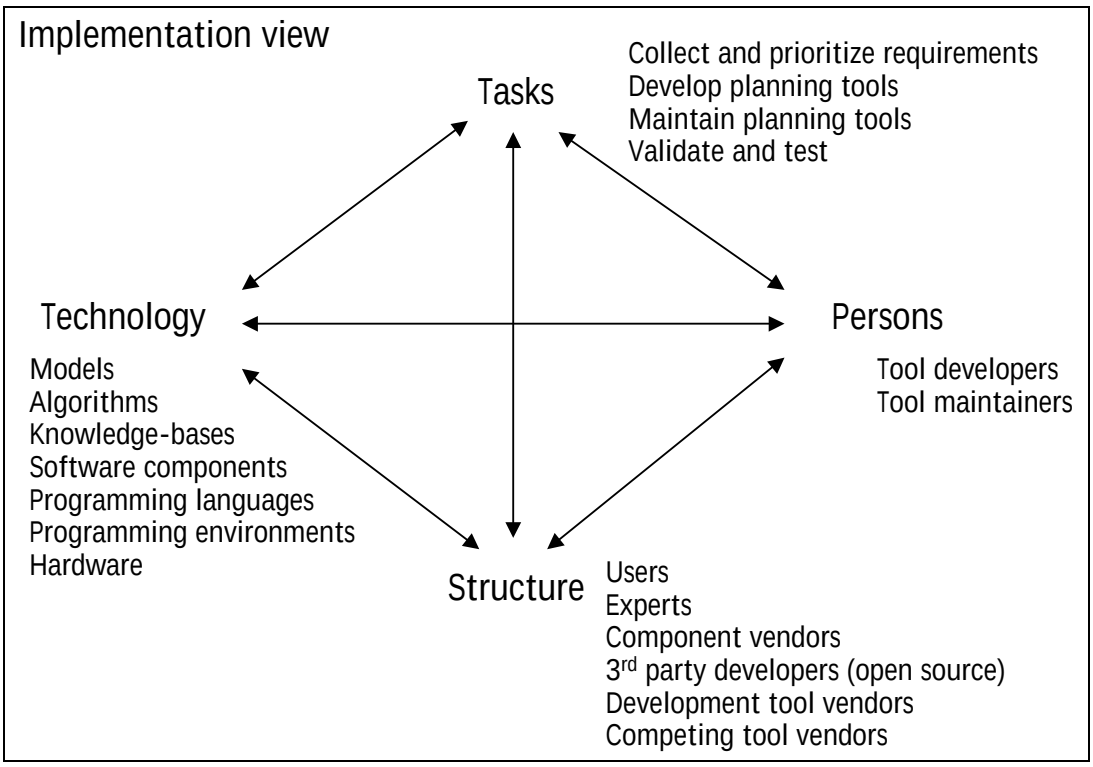

Figure 2 Interactions between implementation issues

Figure 2 shows the application of Leavitt's diamond model to planning tool domain from the implementation point of view. The tasks component consists of requirements management, tool development and maintenance as well as testing and validation. The directly involved persons are tool developers and maintainers. Technology consists of models, algorithms, and knowledge bases, of reusable software components, as well as of software and hardware platforms. Finally, the structure consists of tool users, experts, software component and platform vendors, third party open source developers, and competing tool vendors.

The interactions between the components are numerous. For instance, high-level languages and reusable libraries allow increasingly complex tools to be developed. New needs from the users require

developers to come up with better solutions. In order to implement these solutions new demands arise for the technology. In addition to these examples a number of other interactions are possible.

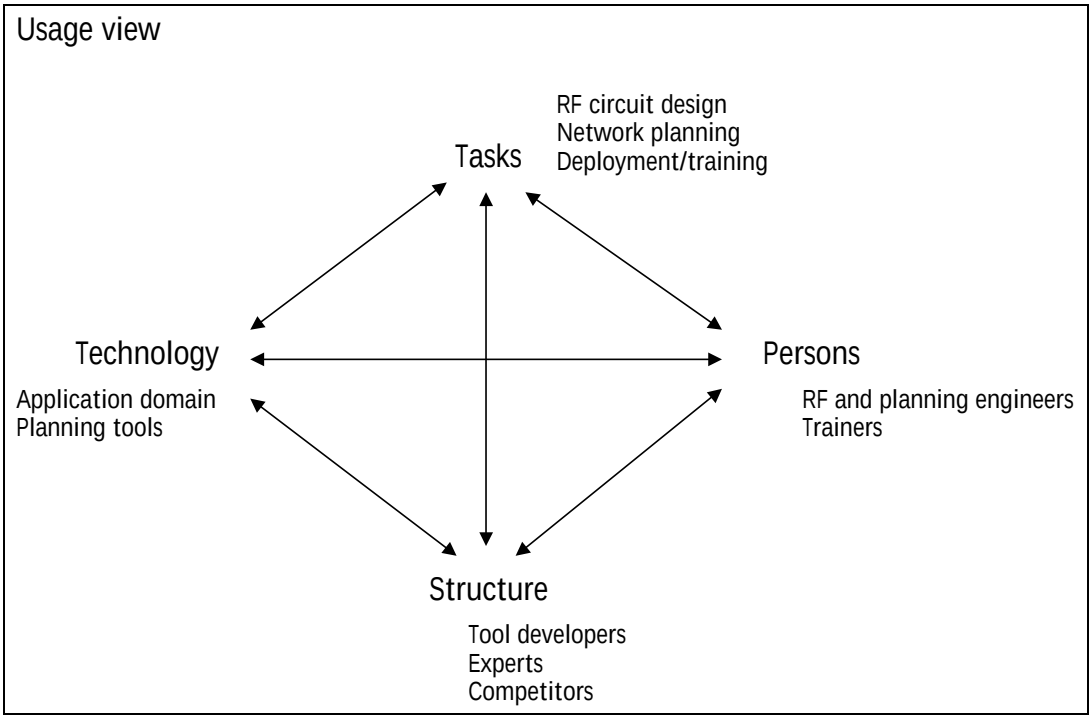

Figure 3 Interactions between usage issues

Figure 3 shows the model from the usage point of view. The essential tasks are related to the application domain, e.g. how to design a cheap and reliable radio transceiver for a given frequency band, or how to route the GSM voice traffic between the base station and the mobile switch. Training and other issues of deployment are also important tasks. The technology component consists of a number of issues of the application domain such as radio frequency circuit technology or the state of the art in transmission communication systems. The focal point of this thesis, planning tools, is seen as a technology from the perspective the engineers using them. The persons involved include trainers and users, RF engineers and network planners, who are technically skilled engineers but who typically do not have an in-depth understanding of the internals of the software tools they use. Finally, structure consists of experts and tool developers as well as competitors.

Like in implementation view the interactions between the components are highly intricate. For instance, interactions between the technology and person components indicate that better tools improve the work of engineers and make it faster and less error-prone. In the reverse direction highly skilled senior experts have different tool needs than junior engineers. Interaction between technology and task components can indicate that better tools allow new tasks to be performed. Changes in the structure, such as pressure from competition, influence the technology as new domain specific technologies need to be taken into use. As a result the complexity of planning tasks can increase which again sets new requirements to planning tools. Other interactions can be traced in a similar fashion.

The two views in Figures 2 and 3 are not independent. For example, the technology component of the usage view is linked with the task component of the implementation view, as the development of planning

tool technology is one of the key tool implementation tasks. Furthermore, the tool developers belonging to the person component in implementation view are part of the structure component of the usage view. In the reverse direction the tool users belonging to person component in usage view are part of the structure in implementation view.

The interdependence between the two views is a reason for conflicts since the needs are contradicting. For example, from the usage point of view tools should make task execution simple and fast while from the implementation point of view the tools should be easy to implement and maintain. These kinds of strong interactions between different views add one more level of complexity to the model.

All models are simplifications of reality and have their own strengths and weaknesses. A benefit of the above model is that it is simple and still it captures the complicated relationships between different factors. A nice property is that the same model can be used to combine interdisciplinary activities, such as routing algorithm arising from algorithm research, implementation processes from software engineering, problem solving and modeling from operations research. This kind of wider view is easily missed when research is carried out within the domains and frameworks of established disciplines.

An important limitation of Leavitt's diamond model is that time is not explicitly handled. Time is an important component that has an impact on all of the components. At usage view time appears, for instance, as a goal to be minimized for task execution (tasks), as a factor affecting the experience and competence of the engineers (persons), as new developments in application domain that require updates to the supporting tools (technology), and as constant pressure from the competitors to work more efficiently (structure).

At implementation view time affects the release deadlines for the tools and the response times for bug fixing and other maintenance (tasks). It also manifests itself both as advancements of algorithms, models, and software technology as well as a selection criterion for those technologies that are feasible to implement within given time constraints (technology). The time that users and experts are able to contribute to the tool implementation is important (structure), as well as the over time accumulating experience and deepening understanding of the developers (persons).

2.2 Research problem

The essential research problem of this thesis is how to develop planning tools in such a way that the objectives of the organization are best fulfilled. If the organization is viewed as a tool user the main issue is how useful the tools are for the primary task of that organization. If the organization is developing the tools the revenue from tool sales is the key objective. These two perspectives are mixed when the main focus is tool development for company internal use.

This is a large problem and it can be approached in several ways. One way would be a generic approach that looks at the complete problem on an abstract level. In that way the complete model in Figures 2 and 3 could be covered. Another way is to focus the analysis deeply to a single detail. In terms of the above model this means analyzing a single component or, more likely, part of a component. Both of these extreme approaches are valuable, but they also have problems. A general analysis has a risk of being too abstract and too general to be of practical use. The focus of a detailed analysis could be so narrow that it looses sight of the relevant interactions.

The approach taken in this thesis is to divide the problem into pairwise interactions between the technology component and the other components of the model. The focus is on the technology

component, in particular on models, algorithms and rulebases, but this approach allows us to investigate how technology affects other components and how other components influence the technology.

Table 1 illustrates this division. It states the explicit subproblem for each component pair and lists the papers forming this thesis that are dealing with each problem.

Table 1 Subproblem division

Components	Subproblem focus	Papers
Technology & Tasks	What kind of models, algorithms, and SW technology to use in tool development and what effect tool evolution has on these?	[I], [IV], [V], [VI], [VII], [VIII]
Technology & Persons	How to divide the planning tasks optimally between the user and the tool?	[II], [VIII]
Technology & Structure	What is the role of external tools vendors and internal experts/users?	[III], [VIII]

2.3 Research methodology

The subproblems described above can be approached with different research methodologies. Following the classification presented in the survey of Pidd and Dunning-Lewis (2001) the characteristic methodology used in this thesis is retrospective participant observation using exploitative style.

Participant observation that is one of the highly engaged methodologies where the researcher takes part in the activity, records the experiences and, after reflection, draws conclusions from this. Participant observation contrasts the unengaged positivist paradigm which is based on objective and detached observation and analysis.

Exploitative research style takes existing ideas and shows how they can be implemented. The opposite style, developmental research, produces new ideas and proposals but does not necessarily focus on their implementation.

Nandhakumar and Jones (1997) analyzed published information systems research and found that vast majority of research is following the unengaged methodologies. They observe that more engaged research, such as participant observation, would be needed e.g. to avoid the risk that the unengaged researches may miss some important aspects of the people and systems that were being investigated. Their view is that useful research is engaged with the organisatorial world rather than conducted at a rather remote distance.

In a similar fashion Neely (1993) analyzing published production and operations management papers highlighted the risks of the dominant developmental and isolated approach. In particular, the author criticizes that the dominant research paradigm ignores the problems of implementation: such as costs, the difficulty of tailoring general results to specific situations and the simple fact that humans are not rational robots.

Pidd and Dunning-Lewis (2001) referring to analysis of "flagship" OR/MS journals in Europe and USA conclude that the journals are dominated by papers stressing untested theory aimed at problems with very narrow scope. Furthermore, they highlight the need for practice-based papers that consist of honest reflections on engaged work.

Fenton (2001) observes similar issues in software engineering research although he does not underline the engagement dimension. He claims that many software engineering problems arise from the complexity for which small-scale laboratory experiments are not adequate. Furthermore commercial and other interests tend to result into over hyped solutions and strong statements without much proof. One attempt to clarify the situation is empirical software engineering, which tries to create an empirical basis for decisions affecting all aspects of the software life cycle. Fenton (2001) continues by suggesting that since it is very hard to instrument a critical software development project, an important paradigm in empirical software engineering is to look at the data that was available and retrospectively consider the most general and useful software engineering hypotheses that can be tested with the data. By collecting enough such empirical evidence it will be possible some day to draw more general conclusions

Like all research methodologies, participant observation has its limitations. First, it is time-consuming and requires that the researcher is able to contribute to the development project, to extract relevant information, and to reflect the significance of the observations (Pidd & Dunning-Lewis, 2001). Secondly, as an active contributor to the development project the researcher may lose some of the neutrality that an external observer may have (Pidd & Dunning-Lewis, 2001). Thirdly, each development project is unique. The empirical requirement for reproducibility is not possible. Finally, for commercial and other reasons organizations tend to control what information can be publicly released. This can distort the picture.

The above observations indicate that engaged, participant observation type of research is likely to be useful and that only a small amount of such research is currently available. Therefore we have chosen to follow that style in most of this thesis. The notable exception is paper [V] that takes the positivist approach to the analysis of algorithms. Table 2 shows the research methodologies for each of the papers forming this thesis as well as the specific research questions discussed in each paper.

Table 2 Research methodologies used in the thesis

Paper	Specific research question	Research methodology
[I]	How to automate RF-design?	Participant observation, exploitative
[II]	How to hide tool complexity from the user?	Participant observation, exploitative
[III]	How to use Lisp for tool implementation?	Participant observation, exploitative
[IV]	How routing algorithms evolve?	Participant observation, exploitative
[V]	How to choose which algorithm to implement?	Positivist, exploitative
[VI]	What kind of models to use in network planning?	Participant observation, exploitative
[VII]	How software component evolves?	Participant observation, exploitative

[VIII]	What are key success factors for expert systems?	Participant observation, exploitative, hypothesis & evidence

3 INDUSTRIAL APPLICATIONS

This thesis is strongly based on the real life experiences on developing tools for industrial applications covering over 10 years work and issues from mathematical programming to artificial intelligence. By following the development and evolution of the software products over a long time period it is possible to see what kind of practical issues are important for the evolution of the systems, models, and algorithms.

Algorithm implementation and especially the development of commercial software systems require a lot of effort. Additional time is needed before the application is deployed widely enough to get feedback from the users. As a result this research process is very time consuming.

The thesis uses two applications as concrete examples. Papers [I]-[III] and partly [VIII] are from the application area of analog circuit design. The RFT design system was developed mainly for the design of the radio frequency (RF) parts for the mobile phones. Papers [IV]-[VII] are related to the network planning application area and use the NPS/10 network planning tool as an example. RFT and NPS/10 are good example cases because both of them are commercial applications and have a long evolution history.

Additionally, experiences with other development projects which the author has participated, e.g. mixed-integer programming (Lahdelma et al., 1986), software engineering environments (Nokia Research Center, 1997), and network management (Nokia Networks, 2002b), have contributed to the work. The combination of experiences from the development of different applications allows more general conclusions to be draws than the analysis of a single application.

It is also interesting to note that applications developed with different style tend to result into highly similar end products. RFT was developed as an expert system application while the intelligence in NPS/10 was based on graph theory algorithms. To the user both applications look quite similar. A Microsoft Windows graphical user interface (see Figures 4 and 5) allows direct-manipulation of design objects, electrical components and network elements. The user can invoke various tools to evaluate, to modify, or to visualize the design.

The external similarity between RFT and NPS/10 as well as discussions with tool users, indicate that the users are not interested in the underlying models and algorithms. They are mainly concerned with features, user-friendliness, and reliability of the results. The developers are in most cases free to use whatever technologies they feel appropriate as long as the user needs are satisfied. The same user experience can be created in different ways, which restates the key question of the thesis: how should the developer choose the models and algorithms for the tools.

3.1 RFT for analog circuit design

The RFT toolkit for analog circuit design was pilot used by Nokia Mobile Phones for the design of mobile phones in 1988-90. Its successor system, NASSE (TEKES, 1996), based on the user-interface concepts of RFT, is used as a front-end to the APLAC simulator. APLAC and NASSE have been widely used within Nokia for the design of mobile phones and other telecommunication products. Since 1998 Aplac Solutions Corporation (APLAC Solutions, 2002) has marketed the toolkit. In addition to Nokia, companies like LK-Products Ltd. (the world's third largest maker of filter structures for mobile phones), Elektrobit, and VTI

Hamlin are using the toolkit. In 1999 the user base reached 1500 engineers. Figure 4 shows a screenshot of NASSE (currently called APLAC Editor)

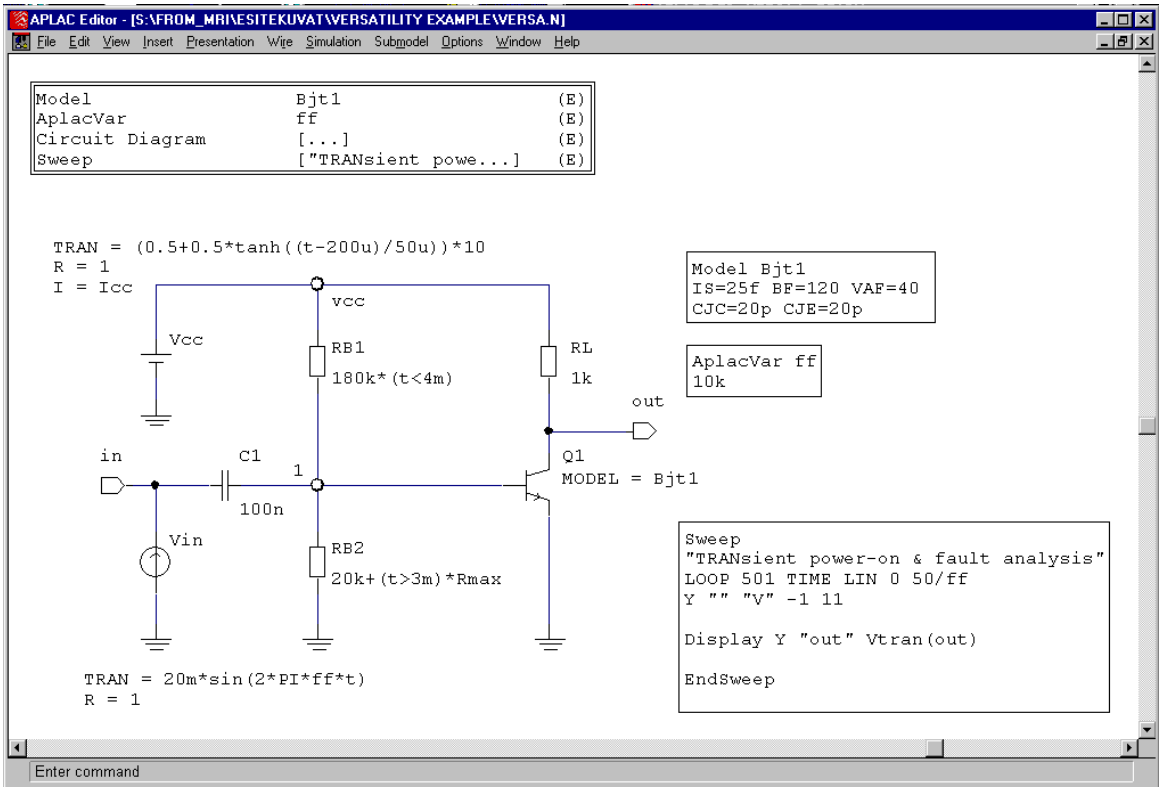

Figure 4 Sample screen of APLAC Editor

3.2 NPS/10 network planning tool

The NPS/10 network planning tool is a commercial product sold by Nokia Networks (Nokia Networks, 2002a). Currently it is packaged with the NetAct tool suite and marketed under the name NetAct Transmission Planner. Figure 5 shows a screenshot of NPS/10. The system has been in extensive internal use since 1993 and commercially available since 1996. According to the last count (January 2000) the system had over 100 active users within Nokia. One measure of the relevance of the NPS/10 development is the list of patents or pending patent applications:

- Bajzik, L., Jaakkola, T., Kodaj, B., Korpela, H., Maarela, A., Nurminen, J., and Oka, L., "Forming a Communication Network," Patent number FI 110746, Finnish Patent Office, 2003.

- Akkanen, J., Korpela, H., and Nurminen, J., " Forming a communication network," pending patent application (number 20001312) with the Finnish Patent Office, 2000.

- Korpela, H. and Nurminen, J., "Partitioning of a Communications Network," pending patent application (number 20001314) with the Finnish Patent Office, 2000.

- Akkanen, J. and Nurminen, J., " Protected routing a communication network," pending patent application (number 2001317) with the Finnish Patent Office, 2000.

- Demeter, H, Korpela, H., and Nurminen, J., "Expansion planning for wireless network," pending patent application (number PCT/EP01/09717) with the Patent Cooperation Treaty, 2001.

- Korpela, H. and Nurminen, J., "Method and Apparatus for Node Adding Decision Support in a Wireless Network," pending patent application (number PCT/EP01/09718) with the Patent Cooperation Treaty, 2001.

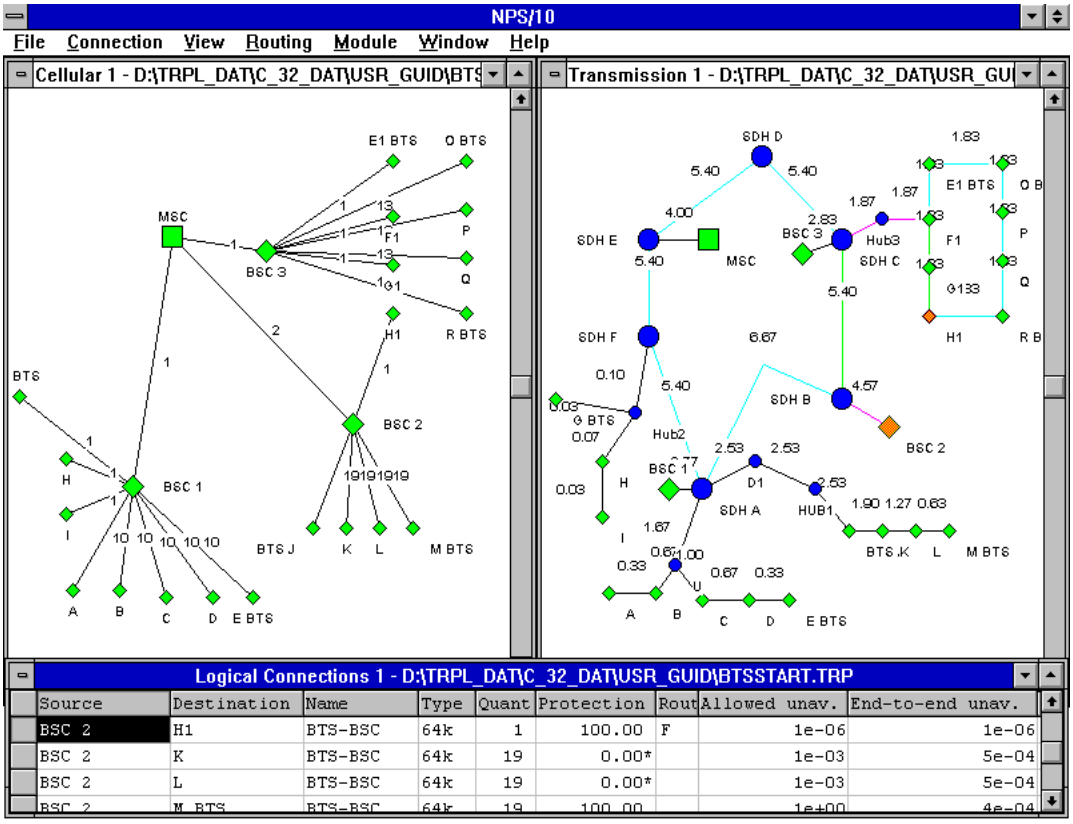

Figure 5 Sample screen of NPS/10

4 MODELLING AND IMPLEMENTATION ISSUES

4.1 The papers forming the thesis

The starting point for the RFT design system [I]-[III] was to hide algorithms and their complexities from the users and create an integrated system. At that time the RF designers used a variety of simulation and optimization tools. By hiding the tools behind the intelligent graphical user interface of RFT the user was supposed to be able to focus on the actual design problems and not on the details of using the tools.

The novel technical approach of papers [I]-[III] was the combination of operations research (OR) and artificial intelligence (AI) techniques. The symbolic computing techniques of AI and the analytical calculations of OR were combined to create an intelligent design system. The role of the intelligence was twofold: to synthesize new designs and to provide an intelligent user interface that hides complex details from the user. Our work attempted to create a practical application by using the most appropriate

techniques for each task. Because improving the interaction with the user was a main target of the work, a graphical user interface was a key component of the application. Also the use of object-oriented programming was a novel implementation technique at that time.

Paper [I] represents the general overview of the RFT design system. It discusses the functional level radio frequency circuit design and describes a design flow, which combines elements from non-linear optimization, simulation, and knowledge-based techniques. The emphasis of the paper is on the design synthesis.

One of the key issues of the paper is that the design problem cannot be solved as a single problem. It needs to be divided into subproblems and the most appropriate solution technique is applied for each subproblem. The techniques that are used to solve the subproblems include non-linear mathematical optimization, heuristic design rules, and interaction with the user.

As a result the system was using multiple solution methods. Even if the paper does not explicitly discuss the pros and cons of different solution methods such thinking was done when the appropriate models and algorithms were selected for the different subproblems. The experience gained by the selection, implementation, and user feedback of the different solution methods was an important trigger to the problem setting of this thesis.

Looking at the paper on hindsight shows that the basic approach of using multiple techniques and integrating AI, OR, user interface, and computer science techniques was a sound choice. In fact, Hayes-Roth (1997) claims that one of the reasons for the success of early knowledge-technology applications was that talented people where developing the systems and applying in an innovative way all kinds of techniques to solve the problems at hand. Also other researchers, such as Doyle and Dean (1996) and Grosz (1996), consider integration and cross-disciplinary efforts very important. They also regard the use of AI as a component in a bigger system as an important direction. This is also confirmed by the observations of paper [VIII].

One of the issues that went wrong was what Allen (1998) calls the microworld mistake. We initially focused on a narrow task and assumed that the same concept could be easily applied to other similar tasks. This turned out to be difficult because of the cost of developing and maintaining the knowledge bases.

The design synthesis part of RFT suffered especially from this problem. Creating a prototype system was relatively easy but updating and maintaining the design rules became a problem. Additionally, automated design synthesis is not flexible enough since the goals and constraints are changing during the design process (Simon, 1995). The creativity of the user cannot be substituted by an automated solution.

This observation is very important for the roles of computer and human problem solver. As discussed in detail in [VIII] it is in most cases useful that the computer supports rather than replaces the human expert.

Paper [II] focuses on the intelligent user interface concept. At that time a number of separate tools were needed for RF design. Each tool had its own rather unfriendly user interface. The idea was to create a knowledge-based layer between the user-interface and the mathematical and simulator tools. The conversion layer would transform a user-specified problem to the appropriate form for each tool. In this way the user would be able to work with schematic circuit diagrams, which is a familiar conceptual level for the user. The symbolic manipulation at the conversion layer would take care of the details, constraints, and complex syntaxes of the underlying tools.

The simpler conversions were based on a one-to-one mapping of the design objects to the tool objects. The more complex conversion handled a one-to-many mapping by converting a single design object to a set of objects for the simulator. In this conversion different object combinations were possible depending on the parameter values used.

In addition to automating the routine task and thus allowing more time for the user for the innovative work, the system also applied a set of verification rules to check that the plan and the resulting problem for the tool did not have any errors. The verification was important since the tools could give wrong results, which might have been hard to find if the input was inconsistent.

This work on intelligent user interfaces has been the most enduring part of the RFT system development. The derivatives of that work are still in use today in the APLAC Editor. As Harmon (1995) notes the intelligent graphical user interfaces are a major use of AI technology even though it is no longer called AI. The area has matured a lot during the past ten years and dramatically changed the way in which computers are used. Graphical user interfaces that have made computers easy to use and intelligent assistants, e.g. the "Wizards" of Microsoft Windows, have become commonplace.

The work described in paper [II] contributes to the discussion of the roles of computers and humans. A hindsight observation of the work in paper [II] is that we tried to hide too much of the details from the user. The straightforward one-to-one mapping from the simulator concepts to the graphical user interface seems to be the most appropriate approach. In that way the user can control how the computer is responding to the input. An attempt to increase the abstraction level and hide more details, such as our one-to-many mapping, is confusing to the user. The result is that the user loses control to what he is doing and the maintenance of the mapping knowledge becomes an issue. As stated by Horvitz (1997) "…the sophisticated machinery frequently does not deliver great value, and can be mimicked by simpler techniques. At times, inelegant application of automated reasoning may even interfere with human-computer interaction."

Paper [III] focuses more on the implementation aspects of the RFT system. The system was developed in Lisp, which was the leading AI development language in 1980s. The main benefits of the use of Lisp were its strong support for the incremental development, which was important since the needs of the design engineers were not easy to extract. Another important aspect was the suitability of Lisp to manipulate the knowledge-definitions, which were defined in separate files.

A further issue discussed in the paper is the integration of AI components with other parts of the application. At that time most AI applications were stand-alone systems. The paper also highlights important issues for large-scale implementation and deployment, such as portability and support. The paper shared our experiences in creating and using an in-house object system and an in-house high-level window system, both of which were important topics at the time of publication. Although today the technical details of the problems are different most of the basic concerns of the paper are still relevant, like the need for integration, problems of portability and compatibility between different versions as well as how to deal with emerging standards.

Later, the use of Lisp turned out to be a burden since it required an expensive engineering workstation. The increasing capacity and popularity of the personal computer prompted us to reimplement the system on a mainstream platform. NASSE, the successor system of RFT, was implemented in Microsoft Windows environment with C++. A single analysis tool, the APLAC simulator (APLAC Solutions, 2002), replaced the variety of previously used analysis tools. Since the user can extend APLAC it can be used as a platform for more complex calculations as well as for optimization. The use of a single simulator

made the interface a lot simpler. Adoption of a clear one-to-one mapping between the simulator and user interface concepts, made the models clearer to the users and easier to maintain.

The relevance of the paper is that it highlighted a variety of tool implementation issues that often deserve little attention. Small details, such as portability, compatibility of different versions, access to support tools etc. can often become critical. One of the lessons on hindsight is that niche approaches, like the use of Lisp, have their drawbacks. Mainstream solutions, even if they were technically inferior, are safer and more future proof.

The incremental development style has been one of the most enduring contributions of the expert system activity from 1980s. Paper [III] discusses how Lisp fits to fast prototyping. In fact, the development style that we used for RFT development was an early form of incremental style. Contrary to prototyping the early versions of the system were not thrown away and reimplemented. Instead, they were modified according to user feedback and the system thus gradually grew towards the needed functionality. This is the essence of the currently popular incremental development styles.

Paper [VIII] is a more recent and broader look at AI applications that were developed at the same time as RFT. Using a sample of eight engineering expert system tools mainly for planning, configuration, and diagnostics area it presents a set of hypotheses for the success of expert system applications. One of the key findings of the paper is that expert systems should complement rather than replace human experts. From this observation follows a set of others like the importance of usability as well as the importance of committing expert to the tool development by regularly providing improved versions of the tool to trial usage.

Another key observation was that the difference between expert systems and information systems is small, rules of normal software development apply to expert systems development, and that simple, straightforward solutions work best.

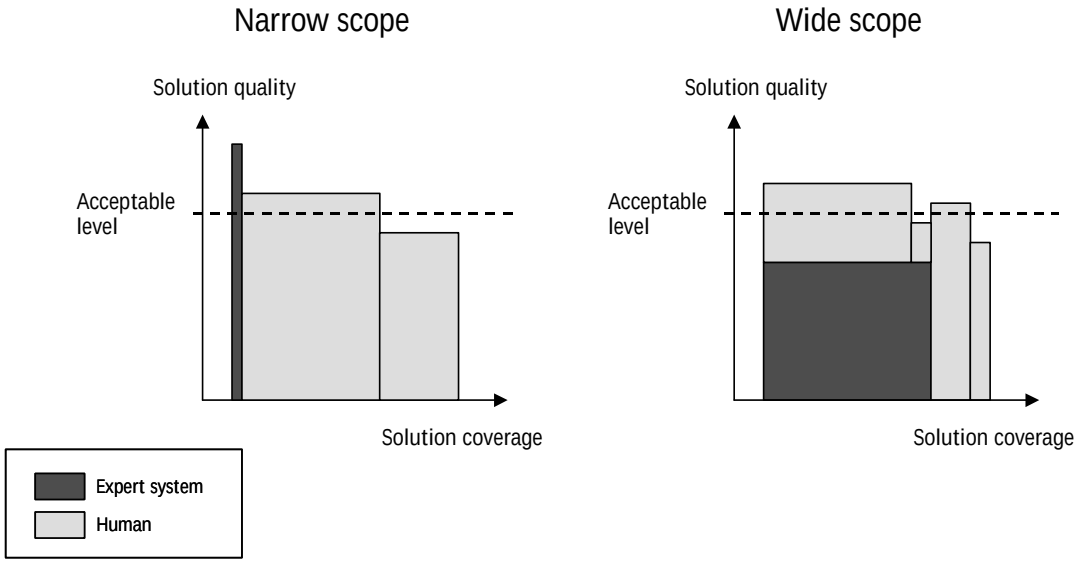

Figure 6 Narrow vs. wide scope

An important finding, that is likely to apply to tool development in general, is that systems with wide scope are better than highly specialized systems. Figure 6 illustrates the difference between narrow and wide

scope applications. With a narrow scope it is possible to develop a system that is able to generate high quality solutions. These solutions can often do things better than average experts. However, it seems that the law of diminishing returns applies: the higher the solution quality is, the bigger marginal effort is needed to increase it. This means that pushing the solution quality above the acceptable level requires much more development than settling with a more modest level.

The drawback of narrow scope is that it is possible to cover only a small part of the problem domain with such systems. Although a large enough number of narrow systems could in theory cover the whole problem domain, in practice there are not enough development resources.

When the scope is wider the level of expertise is typically smaller. In that case the expert system is not targeting complete solutions. Instead, it relies on humans to supplement the draft or partial solutions provided by the system. In this way it typically is easier to cover a much larger area of the problem domain.

A comparison of the two graphs in Figure 6 illustrates that total work for human experts is less when the expert system covers a wider problem area with rough level of detail. At the same time the average quality of solutions tends to be higher. The consequence is that the system cannot work autonomously and a human expert is continuously needed.

A further trouble with narrow scope systems is that they are vulnerable to changes. As can be seen in Figure 6 if the problem domain moves slightly (e.g. with the introduction of a new technology) it can happen that the narrow scope system falls out of the problem domain. For the wide scope systems this risk is smaller.

At first glance our finding of the importance of wide-scope contradicts the influential observation of Moore (1995, pp 20-22) who found out studying the marketing strategies of high tech companies that narrow scope is essential when moving from early markets to mainstream markets. He defined the “whole product” as a *minimum set of products and services necessary to ensure that the target customer will achieve his or her compelling reason to buy* and claimed that high-tech companies frequently failed to reach this level of completion because their scope was too wide.

One explanation to the discrepancy is that Figure 6 does not refer to the product quality. Instead, the solution quality in our study focused on one aspect of the “whole product”; on the quality of solutions produced by the models, algorithms, and knowledge bases of the tool. A number of highly relevant issues, like usability, interfaces, and robustness, were not included. It can be that on the planning tool area the minimum requirement for the automation level is quite low (as this can be compensated rather easily by the human experts).

The observation of value of wide-scope systems is likely to apply not only to expert systems but also to other forms of automation, such as mathematical models. A highly specialized model is likely to create good results but it may require excessive development time. It can also easily become outdated. Different modeling styles were analyzed from this perspective in paper [VI].

Paper [VI] summarizes most of our experiences of NPS/10 tool development and discusses the use of mathematical models and algorithms from the user and developer perspectives. It discusses the user needs and importance of various issues such as the significance of full optimality, the adaptation to changes, usability of the models, and the model and tool development effort.

The paper compares different modeling approaches and analyses how well different approaches, mathematical programming, stochastic algorithms, functional modeling, and special purpose algorithms serve the needs. It concludes by proposing one way to combine the different modeling styles and by discussing when each style would be useful.

Another conclusions of the paper is the importance of incremental and iterative development of the models. The importance of incremental development is supported by studies in the simulation model development (Pidd, 1999; Randell et al., 1999). Incremental methods are getting increasingly popular in software engineering (Jacobson et al., 1999; McConnell, 1997) The main benefits of the incremental process are lower risk, more active user involvement, and better adaptation to changing needs.

Paper [VI] also discusses the roles of the user and the tool and observes again that flexibility is essential for successful planning work to cope with the large number of different aspects. The paper also restates the finding of paper [I] that it is important to divide the problem into smaller subproblems and apply the most appropriate solution technique for each subproblem.

Furthermore the algorithm selection should not only be guided by the easily measurable quantities of speed and solution optimality, but also aspects like complexity of implementation, vulnerability to changes, suitability to incremental development, and usability should be considered.

As observed in papers [VI] and [VIII] incremental approach is essential for tool development. The question is what does incremental development mean to an algorithm and what properties of an algorithm make it suitable for incremental development. The case study of paper [IV] tries to answer these questions. It concentrates on the algorithm evolution and by following the evolution of the routing algorithms in different versions of the NPS/10 planning tool discusses evolution of an algorithm. From the basic Dijkstra's shortest-path algorithm (Dijkstra, 1959) a family of related algorithms has been created to handle increasingly complex routing problems. The lifetime evolution of an algorithm is an important perspective since software evolution and maintenance is expensive. Studies show that the maintenance consumes about 70% of information technology organization budgets (Boehm & Basili, 2001).

The case study in the paper covers a long time span from 1992 to 1998. One of the findings of the paper is that simple algorithms are strong in their capability to be adapted to new uses.

Another conclusion of the paper is that the standard way to evaluate algorithms by comparing running times and closeness to optimality is not adequate. Measures should be developed to estimate and compare the evolution capability, potential for reuse, and the lifetime development cost of the algorithms. Already twenty years ago Ball and Magazine (1981) listed implementation difficulty, flexibility, robustness, simplicity, and interactiveness as important aspects for the comparison of heuristic approaches. It seems that since then very little work about explicit analysis of these aspects of algorithms has been published.

To supplement the qualitative discussion, paper [V] tries to make the concept of algorithm simplicity more explicit and measurable. It studies software complexity measures and their suitability for algorithm comparisons. The paper analyzes to what extent established software complexity metrics (lines-of-code, Halstead's volume (Halstead, 1977), and McCabe's cyclomatic complexity (McCabe, 1976)) are effective in characterizing the implementation and maintenance effort of shortest path algorithms. It is important to notice that the studied metrics work at source code level. Other widely used metrics, such as the function points (for overview see e.g. Garmus and Herron, 2000), measure the complexity from the user point of view. However, they as not applicable to this kind of analysis since the user level functionality, and thus the user level metrics, are the same for all of shortest path algorithms in the study.

The conclusion of the paper is that the complexity metrics do give additional insight that especially allows algorithms from the same implementers to be compared with each other. An interesting observation is that in speed versus complexity dimensions most of the investigated shortest-path algorithms are pareto optimal: an increase in speed required a more complex implementation. This gives evidence to the hypothesis that by a natural selection process the unnecessarily complex algorithms are eliminated over time. Moreover, as the complexity metrics are easy to calculate the paper advocates their use in empirical algorithm studies.

Like paper [IV], paper [VII] analyses the evolution of the planning tool. Its focus is on the graphical component that is responsible for the network data model and visualization. Although software evolution and maintenance have been extensively studied the evolution of reusable components is a much newer area.

Our work supplements the general results reported by Voas (1998) and Lehman and Ramil (2000) by presenting the practical evolution of a software component over a period of ten years. We share many experiences with them like problems with vendor instability, benefits of access to source code, and problems about how the component can be shared between multiple applications. Our more original results are in the area of comparing frameworks with components and what this means for the evolution. Another important practical perspective we discuss is how much effort can in practice be put to the architectural changes and how to implement them in an incremental manner.

A comparison of papers [IV] and [VII] reveals that many evolution issues are common between the routing algorithms and the graphical display component. This observation indicates that there is good potential to apply the results from general software evolution to the models and algorithms as well.

4.2 Summary of the main results

Table 3 summarizes the main results of the papers forming this thesis and shows how they relate to the subproblems defined in section 2.

Table 3 Summary of main results

Subproblem focus	Main results
What kind of models, algorithms, and SW technology to use in tool development and what effect tool evolution has on these?	Combining techniques from different disciplines can result into more optimal practical solutions than optimization with a single technique [I] Real systems are developed in multiple releases over a time span of many years. Maximizing user satisfaction over the tool lifetime should be considered in technology selection and feature prioritisation [IV], [VII] Simplicity, wide applicability, and speed are important for algorithms [IV] Highly accurate or optimal results are seldom needed [IV], [VI] Algorithms have to grow and evolve together with the tools [IV] Implementation complexity is an important, often neglected,

	property of an algorithm. It can also be measured [V] A functional model with good usability for what-if analysis is highly useful. It can be supplemented with more sophisticated algorithms if needed [VI] Domain specific components for product families are useful, but they also have to evolve [VII] It is not possible to foresee all relevant issues. With each release developer insight increases. Models, algorithms, and software modules should be easy to refine and develop further [VII] Simple, straightforward solutions, which the users can easily understand, work best [VIII] Fast and agile development is important [VIII]
How to divide the planning tasks optimally between the user and the tool?	Good tools eliminate routine tasks and trivial errors. This is often all that is needed [II]. Tools should complement rather than replace experts [VIII] Users prefer usability over automation [VIII]
What is the role of external tools vendors and internal experts/users?	Mainstream development tools are a safe choice. Small user community creates problems on niche platforms [III] Commercial general-purpose application generators are not useful. Specially developed application domain specific generators are valuable [VIII] Experts are eager to contribute when they get early benefits for themselves. Such incentives for busy experts are often mandatory [VIII] Expert system development has a lot in common with normal software development [VIII]

Figure 7 collects the main results to the planning tool implementation model. The model can be used to illustrate the close connections between different factors. For instance, simple, fast, and widely applicable models (technology) are good building blocks for solutions that complement expert users but which are still fast and easy to develop and maintain (tasks). Such building blocks fit well to the fast and agile development process (persons). Early access to a working, periodically improving system encourages experts and users to contribute to the development as they realize that tool will help them with their own busy schedules (structure).

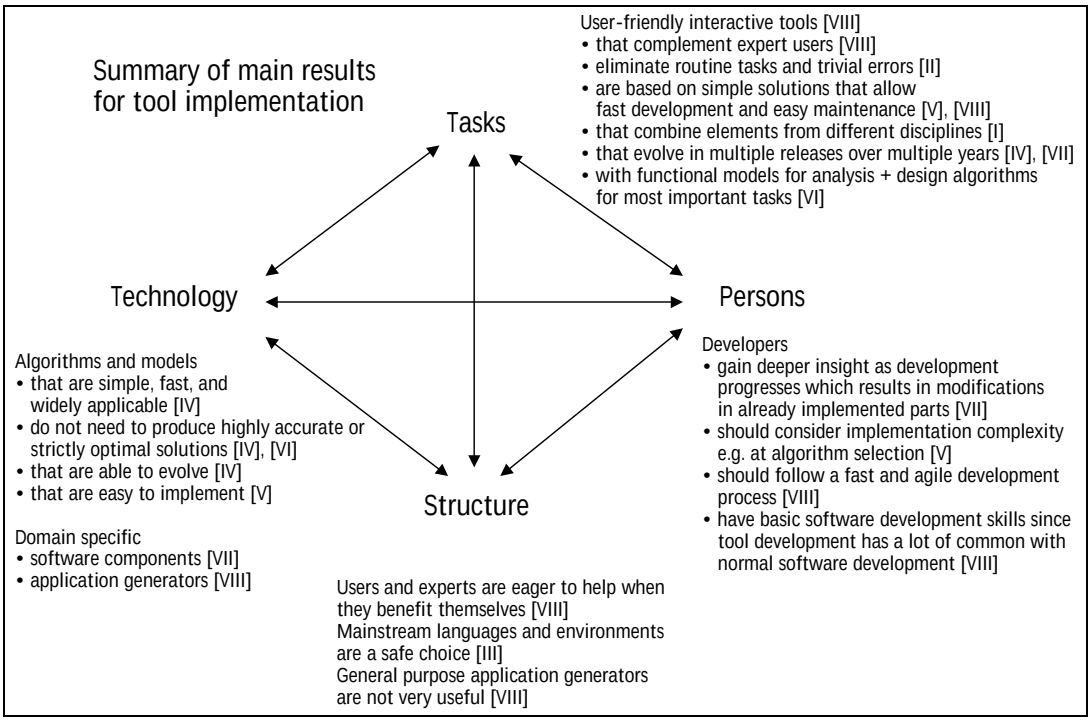

Figure 7 Summary of main results in the tool implementation model

In a similar way Figure 8 presents the main results from the planning tool usage point of view. Notice that Figures 7 and 8 highlight only those issues that in this thesis work have been found relevant for planning tools. Besides these issues there is a multitude of other software implementation and usage issues that are excluded from the figures.

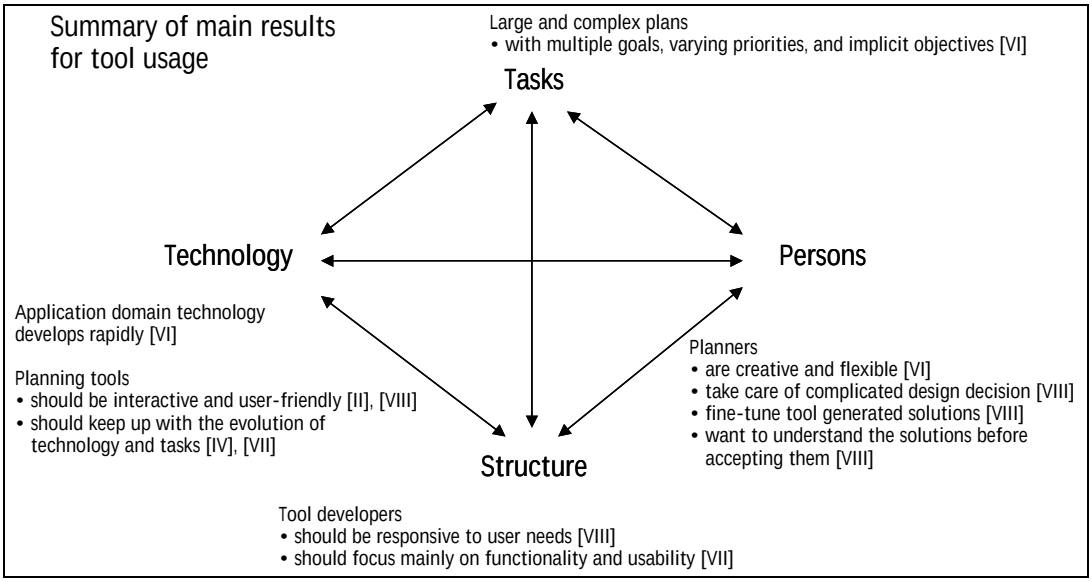

Figure 8 Summary of main results in the tool usage model

5 CONTRIBUTIONS TO RELATED FIELDS

Section 4 presented the papers forming this thesis and tied the results to the model of planning tool implementation and use. Another frame of reference for this thesis is the state-of-the-art of related disciplines. Because of its interdisciplinary nature this work spans multiple fields. In this section we tie the results to the relevant ones and discuss the connections between them and this work.

5.1 Contributions in problem solving environments

Problem solving environment (PSE) is a software system that provides all the computational facilities necessary to solve a target class of problems. The subdiscipline of PSEs was initiated in 1991 in a workshop on Scalable Scientific Software Libraries and Problem-Solving Environments (Gallopoulos et al., 1992). PSEs have their roots on the mechanical and aerospace engineering. Typical applications are the solution of partial differential equations (e.g. Pythia (Weerawarana et al., 1996), Pythia-II (Houstis et al., 2000)), but similar ideas have also been applied e.g. to option pricing (Verykios et al., 2001).

Most typically the PSEs are applied to areas where the solutions are difficult. In these areas multiple algorithms with different strengths are available and one of the key difficulties is to select the right one for the right task. For instance Pythia-II is able to recommend which of the seven partial differential equation solvers in PELLPACK (Houstis at al., 1998) software should be used (Houstis et al., 2000).

As described in Rice and Boisvert (1996), the PSE facilities include advanced solution methods, automatic or semi-automatic selection of solution methods, and ways to easily incorporate novel solution methods. They also include facilities to check the formulation of problems posed, to automatically (or semiautomatically) select computing devices, to view and assess the correctness of the solutions, and to manage the overall computational process. Moreover, PSEs use the terminology of the target class of problems so users can solve them without specialized knowledge of the underlying computer hardware, software, or algorithms. Another definition of PSE is in terms of its components (Rice & Boisvert, 1996)

PSE = user interface + libraries + knowledge base + integration

By 1999 the definition had evolved to be (Rice, 1999)

PSE = natural language + problem solvers + intelligence + software bus

The RFT system discussed in paper [I] was an early example of a PSE preceding by several years the creation of the PSE discipline. RFT corresponds closely to the key features of SPEs defined by Rice and Boisvert (1996). The simulation and optimization modules of RFT provided the advanced solution methods. Intelligent problem formulation for the simulators provided a semi-automatic solution method. The automatic conversions handled the checking of the formulation of problems posed. And finally, the graphical user interface isolated the user from the underlying hardware, software, and algorithms allowing the user to use the terminology of the target class of problems.

Splitting the RFT system into its components results into a formula closely resembling the PSE definition formula of Rice and Boisvert (1996):

RFT = user interface + simulation and optimization libraries + knowledge base + integration.

Likewise the network planning tool, NPS10, can be seen as PSE.

NPS10 = user interface + routing algorithms + integration

The report (Rice & Boisvert, 1996) summarizing the results of a 1995 workshop on the problem solving environment area lists seven major problems for the PSEs to overcome:

1. The current "GUI plus monolithic solver" paradigm is too restrictive
2. Our understanding of the architecture, technologies, and methodologies for scientific PSEs is immature
3. Our ability to achieve easy software evolution and to incorporate large legacy systems is weak.
4. Our understanding of the architecture, technologies, and methodologies for scalable problem solving is immature
5. There is persistent difficulty in creating and managing ever more complex systems
6. The lack of versatile, general systems for complex three-dimensional geometry is a substantial barrier to building software in many science and engineering areas.
7. Our understanding and experience with knowledge bases, expert systems, and similar technologies in scientific computing is premature.

Paper [I] deals with item 1 by providing an example of a system that uses multiple solvers. As a functional system it described one architectural and technical solution to PSEs (item 2). It also touched on item 7 by discussing the role of design knowledge and the problem formulation knowledge.

Paper [II] touched on items 1 and 2 but mainly dealt with items 3 and 7 by studying by means of an example how legacy simulation software could be used as part of PSE.

Paper [III] was on a similar area as paper [II]. While paper [II] was focusing on the use of expert system technology for problem formulation the focus of paper [III] was on the implementation issues.

It is worth noting that the work described in papers [I]-[III] preceded the above list of PSE problems by over five years.

Of the more recent paper, papers [IV] and [VII] analyze software evolution (item 3). They are concrete case studies, which look at the evolution from different perspectives. Paper [IV] is analyzing the algorithm evolution. Paper [VII] is analyzing the evolution of a user interface component. Paper [VIII] again looks at the evolution but this time from the expert system perspective. It also contributes to the item 7 by providing practical experience of the lifetime of multiple expert systems on the technical domain.

Paper [VI] targets item 2 by discussing the roles of different problem solving technologies (mathematical programming, stochastic algorithms, functional modeling, and special purpose algorithms) and considering what would be the optimal use of each technology in a problem solving environment for network planning.

Finally, paper [V] deals with items 3 and 5 by presenting a way to measure the implementation difficulty and, more indirectly, the evolution potential of an algorithm. Rice (1997) identifies the difficulty to measure software quality (accuracy, reliability, efficiency, programming cost) is one major obstacle for PSEs. The use of software implementation complexity metrics as suggested in paper [V] is a promising

way to create such measures which would to some extent quantify the programming cost and, to a less extent, the reliability.

5.2 Contributions in applied operations research

Planning is a goal-oriented activity that aims for the best possible outcome for each planning case. This very naturally leads into considering these tasks as optimization problems. The planning goals are defined in the objective function and the other requirements and limitations are formulated as constraints.

In practice formulating and solving such a mathematical programming problem can be difficult for any reasonably complex task. A multitude of issues, such as implicit constraints and objectives, multiple goals, and the need for new, innovative solutions, cannot be handled without human creativity. Furthermore, even simplified formulations of many problems arising in planning are computationally complex. For instance Johnson et al. (1978) prove that the network design problem is NP complete. Since practically significant problems frequently contain hundreds of nodes heuristic solutions are needed.

An important issue for adding automated support for planning is the appropriate level of automation. Extremes are on the one hand optimization, on the other hand electronic drawing boards. In the former case, the planner has very limited control over the support system and is left passive. In the latter case the planner has full control but little support.

In the thesis work summarized in (Hofstede, 1991) Hofstede came up with a conclusion that it seems better to opt for modesty in modeling. By studying the users of a decision support system for pot plant growers he observed that: “however useful intelligent algorithms are, it is control and flexibility in which the pot plant growers are most interested”. He concludes that this observation should be taken into account in selecting algorithms for computer problem solving more than what was common at that time and called for further research to determine what kind of flexibility is crucial to the user.

Much of this thesis work deals with the same question. Papers [VI] and [VIII] most explicitly discuss this problem by analyzing the planner’s needs and considering how the different models match them. The experience and architecture of the RFT system (papers [I]-[III]), the NPS10 planning tool (paper [VI]), and the expert systems (paper [VIII]) confirm Hofstede’s observation that control and flexibility are very important for planners.

While Hofstede was mainly focused on the user needs, the work in this thesis expands the perspective by taking into account the implementation effort of the planning tools. As stated in paper [VI] the choice of appropriate model is influenced by the development costs. This aspect is further considered and extended over the system life-time in papers [IV] and [VII] about the algorithm and component evolution.

Visual interactive modeling (VIM) is a management science/operations research direction that integrates mathematical or symbolic models with runtime interaction and real-time graphics displays of the model output to aid decision makers (Bell et al., 1999). Survey studies among model builders (Kirkpatrick & Bell, 1989) and decision makers (Bell et al., 1999) report a lot of positive aspects about the use of VIM:

- VIM increases interaction between the modelers and decision makers resulting into better models
- Increased confidence to the results and enhanced model validation
- Faster decision making time and easier decision implementation

The connection between VIM and this thesis work arises from the fact that both RFT and NPS/10 can be seen as decision-making tools applying the VIM paradigm. In particular paper [VI] deals with these issues and gives support to the three observations above.

While the management science perspective typically deals with strategic decision making problems the main focus of the thesis is on operative, everyday decision-making. It is not possible to develop a dedicated model for each decision. Instead it is important to package the relevant decision making support to a reusable tool. The thesis work thus extends the work on VIM by focusing on issues that are important from the tool development perspective. Such new issues are: the evolution of the decision-making models (papers [IV] and [VII]) and consideration of the implementation perspective in model selections (paper [VI]).

Considering that OR models typically aim at cost minimization, it is surprising that hardly any work is available about the cost of model and solution development. Some papers discuss, as required for instance for the Franz Edelman Competition (see e.g. (INFORMS & CPMS, 1999)), what has been the financial impact of the OR applications. However, even if the reported approaches are definitely useful it does not necessarily mean that they are the best ones when all aspects are considered.

What is missing from the field is comparative analysis of different ways to solve the same problem. The established way to compare algorithm accuracy and running times handles only a small part of the issue. Analyzing the tradeoffs between the model accuracy and the implementation effort would seem to be a good area for OR analysis. The thesis covers this area partly. Paper [VI] discusses the alternative modeling approaches from the qualitative point of view. Paper [V] performs a quantitative comparison of shortest-path algorithms.

5.3 Contributions in software engineering

Software engineering is focused on designing and developing high quality software. Software has an increasingly important role in almost all products, the complexity of the software systems is steadily increasing, the number of people developing software is high, and it is essential that software is developed in an efficient, reliable, and cost-efficient way.

Using the retrospective participant observation methodology this work contributes to the empirical software engineering. Paper [III] analyses the effect of using Lisp and object-oriented programming. Papers [IV], [VII], and [VIII] concentrate on the evolution of the software system viewing it from different angles. The focus of paper [IV] is on the algorithms, paper [VII] deals with the software component, and paper [VIII] analyzes the lifetime of knowledge-based systems.

The thesis work, in particular, papers [IV], [V], and [VI] extend the work on software engineering by concentrating on algorithm selection and on the effects this has on software. This viewpoint is new in software engineering. Algorithm research usually considers them separately from the applications and from their implementation. Some practical guidelines are available. Kershenbaum (1993) states that “given a choice between a straightforward algorithm and a more “clever” one, always choose the simpler one unless there is a substantial difference in efficiency or effectiveness.” The author justifies the statement by claiming that simpler algorithms have less bugs, the testing is easier, and that they are more suitable for reuse. Sedgewick (1995) recommends that “one should first implement the simplest algorithms to solve a given problem.” The author continues that a more complex one can later replace the simple algorithm when it is clear that a more efficient algorithm is needed. Furthermore he states that the two most common mistakes in algorithm selection are to ignore performance characteristics completely

for an easier implementation or to pay too much attention to performance characteristics and ignore the implementation difficulty.

Paper [IV] about the evolution of routing algorithm gives evidence to both of the above statements. It supports Kershenbaum by showing that it is relatively easy to reuse the simple algorithm in more complex tasks. It also illustrates the Sedgewick's statement that it is possible to replace the simple algorithm with a more complex one when the need arises.

Paper [V] targets the two common mistakes stated by Sedgewick and explicitly and quantitatively analyses the tradeoff between performance and implementation effort.

Incremental development is becoming more and more popular software development process. Methodologies like Microsoft's daily build approach (Cusumano & Selby, 1995) have proven to provide good results in practice. The basic idea of a variety of incremental methods is to create a rough prototype application as soon as possible. This skeleton application is later refined incrementally to provide more and better functionality.

Similarly the increasingly popular Extreme Programming methodology (see e.g. Beck, 1999) emphasizes early delivery of the system to the customer and incremental implementation of changes. Extreme Programming underlines the importance of teamwork between developers and customers as a means to achieve customer satisfaction.

The software development methodology is not independent from the model and algorithms involved. As discussed in paper [VI] the incremental development style sets new requirements for model and algorithm development. The algorithms have to be developed incrementally as well. It is important to have a draft algorithm working as soon as possible and refine it later. Such a development process is discussed in detail in paper [IV] for the routing algorithms. Paper [VII] presents similar development process for the network presentation component.

Some parts of the work also contribute to several software engineering areas, such as graphical user interfaces ([II], [III], [VII]), object-orientation ([III]), and software components ([VII]).

5.4 Contributions in software economics

In the discussion of economics driven software engineering, Sullivan et al. (1999) define the central challenge in software development practice to be the making of a sequence of technical software decisions in a way that creates added value, that satisfies the constraints, and that is able to deal with the uncertainties. The constraints typically involve time, money, and intellectual capital. Uncertainties involve the states and evolution of markets and technologies, including competitive threats.

Sullivan et al. (1999) claim further that software engineering research has not fully exploited opportunities to develop explicit guidance for practitioners who are facing such engineering problems. The focus has instead been primarily on technical excellence in a largely economics-independent context.

Another view to the same issues is that information technology components and software engineering cannot be handled separately. Focusing only on great IT components without attention to complementary software engineering practices generally leads to failure (Boehm & Basili, 2000).

An observation of this thesis is that the above thinking applies to algorithms as well. Algorithm research has been mainly focused on the development of technically excellent algorithms. However, as observed

in papers [IV], [V], [VI] it is important to understand that the most sophisticated algorithms may not be necessary for practical success and that sometimes overemphasizing the algorithms can be even harmful.

For software the time-to-market considerations are becoming paramount. Product cycles have dropped from 18 months to 6 months or less (Boehm & Basili, 2000). Tight development schedules also apply to internal tools since they are often mandatory for the business. With such schedules it is not possible to devote too much time on the algorithm development and implementation. In such an environment simple algorithms, which can be incrementally improved, are often more suitable than more accurate or faster ones as illustrated by the case study in paper [IV]. Moreover, the different modeling approaches as discussed in paper [VI] behave differently when the product requires fast and incremental development.

Another important direction in software engineering economics, as discussed in (Boehm & Sullivan, 2000), is value-driven design and especially the strategic value of modularity. The authors state that modularity establishes valuable options for intermediate decision points to improve the system one part at a time, to create variants to address new and changed markets, develop variants quickly and abandon less important or less mature features. This kind of approach has been a cornerstone of the development processes of PC software companies like Microsoft and Netscape (Cusumano & Yoffie, 1999). The implication for models and algorithms is that they should tolerate such rapid changes in direction. Modularity in models and algorithms is largely an unstudied area. Easy implementation of an algorithm is a partial answer to this issue and the metrics discussed in [V] are an attempt to highlight this aspect. Additionally each modeling style in [VI] has its own strengths and weaknesses in this dimension.

An empirical observation for software is that typical life cycle cost distribution is 30% for development and 70% for maintenance (Boehm & Basili, 2001). Although it is not clear if the same numbers apply to algorithms they are a clear indication that the maintenance cost is significant. Research on what kind of algorithms and models are the easiest to maintain would be important. The software complexity measures in paper [V] provide one such indication. However, it would be important to study how different modeling styles affect the maintenance.

An obvious solution to reduce the cost of algorithm development is to use algorithm libraries and other commercially available software products as part of the software systems. In this way the algorithm development work could be saved completely. Unfortunately, this approach may not be as useful as it seems with the first thought. General studies of the use of commercial off-the-shelf software (COTS) indicate that embedding and maintaining such a solution is not easy. Gluing the software component is on the average three times more expensive per line of code than normal software development, post deployment cost of using COTS exceeds development cost, and maintenance is further complicated by the vendor policies of supporting only the latest three releases which results into unsupported platform in less than three years (Basili & Boehm, 2001). It is not clear how closely the mathematical software packages are following the average industry figures.

Boehm and Basili (2000) state that there is a strong need to develop the empirical science underlying software as rapidly as possible. While some results are available on the general software area very few results are available about the economics of using mathematical models and algorithms. One of the contributions of this thesis is that it highlights the need for such research and gives some empirical observations from the area.

5.5 Contributions in routing algorithms

The routing algorithms in paper [IV] extend the work of Suurballe and Tarjan (1984). Our approach is less ambitious than many other models and algorithms available in literature. Much research has been done to formulate and solve the routing problems as multicommodity flow problems (e.g. (Assad, 1978; Balakrishnan et al., 1989; Bertsekas, 1998; Schneur & Orlin, 1998)). Unfortunately, the problem is NP-hard and all these algorithms are computationally feasible only with small networks.

Our approach differs from these in the respect that we are using the simple Dijkstra's algorithm (Dijkstra, 1959) as a building block and create more complex algorithms by using it iteratively. This incremental development approach is in line with Kershenbaum's (1993) and Sedgewick's (1995) recommendations on how to approach a new problem with an algorithmic solution.

5.6 Contributions in artificial Intelligence

Papers [I], [II], and [III] were on the forefront of the knowledge-technology applications at the time of their publication. In particular, the idea to combine numeric and symbolic computing was just emerging and there were not many practical applications that followed such concept.

After the high interest at the late 1980s interest in AI declined. Paper [VIII] is an attempt to highlight what happened to those applications that were developed at that time within our group. The reality is surprisingly bright. Many applications were in real use for years and two applications are still in active use today. Naturally, the applications have been fundamentally changed during the years and would not be called AI applications in today's form.

The long time horizon allows more insightful observations of some of the key characteristics of those applications. The paper complements the observations of Hayes-Roth (1997) and Allen (1998) who are analyzing the state of AI in years 1996 and 1997 as well as the retrospective analysis of expert system success factors of Duchessi and O'Keefe (1995) and Millett and Powell (1996).

Contradicting previous studies, in particularly Hayes-Roth (1997), paper [VIII] suggests that wide scope, multiple objectives, unstable environment, and moderate degree of automation are important considerations for expert system development. The user of an engineering expert system typically is and has to be an expert by him/herself. Expert systems should complement rather than replace human experts. Particular attention should be given to usability, which expert users often consider more important than automation. Part of good usability is that the expert systems are embedded and integrated parts of larger information systems.

Implementation of such systems has a lot in common with normal software development as was also observed by Duchessi and O'Keefe (1995) and Millett and Powell (1996). In fact, we found that most of the systems were ported to mainstream software and hardware platforms during their lifecycle. Specially developed domain specific application engines and generators turned out to be highly useful. General-purpose knowledge representations, in particular if-then rules, did not suit well for engineering applications.

In line with the observation of Duchessi and O'Keefe (1995) we found that fast and agile development is important to cope with the changing environment. Additionally we observed that experts, both as developers and users of the systems, are in a crucial role. Organizing the development in such a way that early versions of the systems gave benefits to the experts themselves, was a key means of getting the experts involved deeply enough in spite of their overload with other urgent business-critical tasks.

6 CONCLUDING REMARKS

This thesis consists of studies of modeling and implementation issues for planning tool development. In particularly it emphasizes issues that are relevant for practical, industrial use of mathematical models and algorithms. This includes the selection, implementation, and continuous development of models and algorithms for tools that support human experts. The work uses two commercial applications, a circuit design and a network planning tool, as concrete examples.

A model of planning tool implementation has been developed and used to structure the results. The main part of this research has been conducted using the participant observation research methodology by collecting and analyzing experiences in the development and maintenance of the two tools. These results have been tied to the planning tool implementation model. In particular the focus has been on how technology interacts with tasks, persons, and organizations. The results have also been discussed within the context of established research disciplines.

An observation of the thesis is how complex planning tasks should be divided between a computer and a human expert. Discussing the circuit design application, papers [I] and [II] suggest one way to do this by using knowledge-technology and intelligent interfaces coupled with simulations and human decision making. Paper [VIII] confirms this with a larger sample of engineering expert systems.

This theme is discussed and generalized further in [VI] which focuses on the end-user needs of planning systems and on the usefulness of different kind of models to satisfy the needs. The routing algorithms and their use in the network planning tool [IV] is one representative case that illustrates the evolution of shortest path algorithms to meet changing needs over a six year period.

Paper [VIII] suggests that a modest level of expertise in a system on a wide area is often better than narrow systems. There is little need for highly accurate or optimal results since the experts in most cases need to review and modify the plans [VI].

The other main theme of the thesis is the importance of implementation. While all of the papers forming this thesis are discussing real applications, the case studies in papers [IV] and [VII] in particular look at the lifetime evolution of algorithms and software components. A common conclusion of both papers is that the user focused, incremental and iterative development is a useful development approach.

Incremental development was briefly discussed already in 1990 in paper [III]. At that time such a development process was far from its current popularity. The importance of incremental development was discussed further in papers [VI] and [VIII], and papers [IV] and [VII] illustrated it with concrete examples. Papers [IV] and [VI] highlighted that incremental development sets new requirements to models and algorithms.

Routing algorithms are discussed in papers [IV] and [V]. Paper [IV] presents and analyses several derivatives of the Dijkstra's algorithm, which are used in the network planning tool. Paper [V] analyses different implementations of routing algorithms. The new dimension in the analysis is the use of software complexity metrics to measure the implementation complexity. The results suggest that the software complexity measures are useful tools in algorithm comparison.

The main conclusion of the thesis is that models and algorithms are part of a bigger picture. Implementation, integration with other components, lifecycle of the software system, and meeting the user

needs decide the ultimate success. Properly selected models and algorithms, their incremental development, and suitable division of work between the user and computer are important success factors.

7 REFERENCES

Allen, J. F., 1998. AI growing up: The changes and opportunities. *AI Magazine*, Winter 1998, 13-23.

APLAC Solutions Corporation, 2002. APLAC. Computer program. www.aplac.com.

Assad, A. A., 1978. Multicommodity network flows - A survey. *Networks* 8, 37-91.

Balakrishnan, A., Magnanti, T. L., Wong, R. T., 1989. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37(5), 716-740.

Ball, M., Magazine, M., 1981. The design and analysis of heuristics. *Networks* 11, 215-219.

Basili, V. R., Boehm, B., 2001. COTS-based systems top 10 list. *IEEE Computer*, May 2001, 91-93.

Beck, K., 1999. *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Bell, P. C., Anderson, C. K., Staples, D. S., Elder, M., 1999. Decision-makers' perceptions of the value and impact of visual interactive modelling. *Omega, International Journal of Management Science* 27, 155-165.

Bertsekas, D. 1998, *Network optimization: continuous and discrete models.* Athena Scientific, Belmont, MA.

Boehm, B., Basili, V. R., 2000. Gaining intellectual control of software development. *IEEE Computer*, May 2000, 27-33.

Boehm, B., Basili, V. R., 2001. Software defect reduction top 10 list. *IEEE Computer*, January 2001, 135-137.

Boehm, B., Sullivan, K., 2000. Software economics: a roadmap. *Proceedings of the Conference on the Future of Software Engineering* (pp. 321-343). Limerick, Irland.

Bui, T., Higa, K., Sivakumar, V., Yen, J., 1996. Beyond telecommuting: Organizational suitability of different modes of telework. *Proceeding of the 29th Annual Hawaii International Conference on System Sciences* (pp. 344-353). Maui, Hawaii.

Cusumano, M. A., Selby, R. W., 1995. *Microsoft secrets.* Free Press, New York.

Cusumano, M. A., Yoffie, D. B., 1999. Software development on internet time. *IEEE Computer*, October 1999, 60-69.

Dijkstra, E., 1959. A note on two problems in connexion with graphs. *Numerische Mathematics* 1, 269-271.

Doyle, J., Dean, T., 1996. Strategic directions for artificial intelligence. *ACM Computing Surveys* 28(4), 653-670.

Duchessi, P., O'Keefe, R., 1995. Understanding Expert Systems Success and Failure. *Expert Systems with Applications* 9(2), 123-133.

Fenton, N., 2001. Conducting and presenting empirical software engineering. *Empirical Software Engineering* 6, 195-200.

Gallopoulos, E., Houstis, E. N., Rice, J. R., 1992. Future research directions in problem solving environments for computation science, Computer Science Dept. Purdue Univ., http://www.cs.purdue.edu/research/cse/publications/tr/92/92-032.ps.gz.

Garmus, D., Herron, D., 2000. *Function Point Analysis: Measurement Practices for Successful Software Projects*. Addison-Wesley.

Gartner, 2002a. NSM Market Dynamics: 2002-2006 Forecast. Gartner Dataquest Market Statistics (SWSI-WW-MS-0116).

Gartner, 2002b. 2001 Worldwide Electronic Design Automation Market Share. Gartner Dataquest Market Statistics (SWTA-WW-MS-0114).

Grosz, B., 1996. The importance of integration for AI. Position statement on strategic directions in computing research. *ACM Computing Surveys* 28(4).

Halstead, M., 1977, *Elements of software science.* Elsevier North-Holland, New York.

Harmon, P., 1995. Object-oriented AI: a commercial perspective*. Communications of the ACM* 38(11), 80-86.

Hayes-Roth, F., 1997. Artificial Intelligence - What works and what doesn't? *AI Magazine*, Summer 1997, 99-113.

Hofstede, G., 1991. Interactive planning system design, a model and an application. *Proceedings of the First International Conference on Expert Planning Systems* (pp. 175-180). Brighton. UK.

Horvitz, E.,1997. Compelling intelligent user interfaces - How much AI is enough. Position statement for panel discussion. *Proceedings of 2nd International Conference on Intelligent User Interfaces* (pp. 173-175). Orlando, Florida.

Houstis, E., Rice, J., Weerawarana, S., Catlin, A., Gaitatzes, M., Papachiou, P., Wang, K., 1998. Parallel ELLPACK: a problem solving environment for PDE based applications on multicomputer platforms. *ACM Transactions on Mathematical Software* 24(1), 30-73.

Houstis, E., Catlin, A., Rice, J., Verykios, V., Ramakrishnan, N., Houstis, C., 2000. PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software* 26(2), 227-253.

INFORMS, CPMS, 1999. Franz Edelman Award Papers. *Interfaces* 29(1), 1-130.

Jacobson, I., Booch, G., Rumbaugh, J., 1999. *The Unified Software Development Process*. Addison Wesley Longman. Reading, Massachusetts.

Johnson, D., Lenstra, J., Rinnooy Kan, A., 1978. The complexity of the network design problem. *Networks* 8, 279-285.

Keen, P., 1981. Information systems and organizational change. *Communications of the ACM* 24(1), 24-33.

Kershenbaum, A. 1993, *Telecommunications network design algorithms.* McGraw-Hill, New York.

Kirkpatrick, P., Bell, P., 1989. Visual interactive modelling in industry: results from a survey of visual interactive model builders. *Interfaces* 19(5), 71-79.

Lahdelma, R., Nurminen, J., Ruuth, S., 1986. Implementations of LP- and MIP-systems, Helsinki University of Technology, Systems Analysis Laboratory, A18.

Leavitt, H., 1965, Applying organizational change in industry: Structural, technological and humanistic approaches. In March, J. (Ed), *Handbook of Organizations*, Rand McNally, Chigaco, Illinois.

Lehman, M., Ramil, J., 2000. Software evolution in the age of component-based software engineering. *IEEE Proceedings on Software* 147(6), 249-255.

McCabe, T., 1976. Complexity measure. *IEEE Transactions on Software Engineering* 5(4), 308-320.

McConnell, S., 1997. *Software Project Survival Guide*. Microsoft Press. Redmond, Washington.

Millett, D., Powell, P., 1996. Critical success factors in expert system development: a case study, *Proceedings of the ACM SIGCPR/SIGMIS Conference* (pp. 214-222). Denver, Colorado.

Moore, G., 1995*. Inside the tornado: marketing strategies from Silicon Valley's cutting edge*. HarperCollins Publishers. New York.

Nandhakumar, J., Jones, M., 1997. Too close for comfort? Distance and engagement in interpretive information systems research. *Journal of Information Systems* 7, 109-131.

Neely, A., 1993. Productions/operations management: Research process and content during the 1980s. *International Journal of Operations and Production Management* 13(1), 5-18.

Niederman, F., Trower, J., 1993. Industry influence on IS personnel and roles. *Proceedings of the 1993 Conference on Computer Personnel Research* (pp. 226-233). St Louis, Missouri.

Nokia Networks, 2002a. Nokia NetAct Transmission Planner. Computer program. www.nokia.com/networks/product_catalog/pc_product_highlights/1,5567,,00.html?prod_id=ES11008.

Nokia Networks, 2002b. Nokia RoofTop Router Management System. Computer program. www.wbs.nokia.com/solution/netmanage.html.

Nokia Research Center, 1997. KISS design tool. Computer program.

Pidd, M., 1999. Just Modeling Through: A Rough Guide to Modeling. *Interfaces* 29(2), 118-132.

Pidd, M., Dunning-Lewis, P. 2001. Innovative research in OR/MS? *European Journal of Operational Research* 128, 1-13.

Randell, L., Holst, L., Bolmsjö, G., 1999. Incremental system development of large discrete-event simulation models. *Proceedings of the 1999 Winter Simulation Conference* (pp. 561 – 568). Phoenix, Arizona.

Rice, J., 1997. Future scientific software systems*. IEEE Computational Science and Engineering* April-June, 44-48.

Rice, J., 1999. A perspective on computational science in the 21st Century. *Computing in Science & Engineering* 1(2), 14-16.

Rice, J., Boisvert, R., 1996. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering* 3(3), 44-53.

Schneur, R., Orlin, J., 1998. A scaling algorithm for multicommodity flow problems. *Operations Research* 46(2), 231-246.

Sedgewick, R. 1995, *Algorithms in C++.* Addison-Wesley, Reading, Massachusetts.

Simon, H. A., 1995. Problem forming, problem finding, and problem solving in design. In Collen, A. & Gasparski, W. (Eds.), *Design and systems: general applications of methodology*, Transaction Publishers, New Brunswick, 245-257.

Sullivan, K., Notkin, D., Fuggetta, A., Favaro, J., 1999. First workshop on economics-driven software engineering research. *Proceedings of the 1999 International Conference on Software Engineering* (pp. 699-700). Los Angeles, California.

Suurballe, J., Tarjan, R., 1984. A quick method for finding shortest pairs of disjoint paths. *Networks* 14, 325-336.

TEKES, 1996. RF Assistant (Nasse). In: *The Electronics Design and Manufacturing Technology Programme 1991-1995: Final Report*, TEKES, 52-53.

Verykios, V., Houstis, E., Tsoukala, L., Pantazopoulos, K., 2001. Automating the analysis of option pricing algorithms through intelligent knowledge acquisition approaches. *IEEE Transactions on Systems, Man and Cybernetics*, Part A, 31(6), 573-586.

Voas, J., 1998. Maintaining component-based systems. *IEEE Software* 15(4), 22-27.

Weerawarana, S., Houstis, E., Rice, J., Joshi, A., Houstis, C., 1996. PYTHIA: a knowledge-based system to select scientific algorithms. *ACM Transactions on Mathematical Software* 22(4), 447-468.