PERGAMON

Expert Systems with Applications 24 (2003) 199–211

Expert Systems with Applications

www.elsevier.com/locate/eswa

What makes expert systems survive over 10 years—empirical evaluation of several engineering applications

Jukka K. Nurminen*, Olli Karonen, Kimmo Hätönen

Nokia Research Center, P.O. Box 407, FIN-00045 Nokia Group, Finland

Abstract

This case study analyzes eight expert system applications that have successfully been in industrial use for a long time. We have personally been involved in the development of these applications and are thus in a good position to analyze what is important for a successful application and what kind of mistakes can be made. Since the development of the applications started in 1986–1990 and some of them are still in use we are able to observe what has happened to those applications during their lifetime. Our key observations are related to the scope of the applications, to the trade-off between usability and automation, to the role of human experts in the use and development of expert systems, on the technical solutions used, on aspects of the operation of the expert system and on the similarities between expert systems and information systems. The key findings are expressed as 20 hypotheses for successful expert systems. The support of each application to the hypotheses is discussed.

Keywords: Expert systems; Industrial applications; Lessons learned; Success factors; Implementation issues

1. Introduction

In late 1980s there was a period of high activity on expert systems, with a lot of commercial interests and expectations. As very few of the overdimensioned expectations were met the interest declined (Harmon, 1995) but now the activity seems to be increasing again. The goal of this paper is to look back at some expert systems that were developed at that time in Nokia Research Center, discuss their lifecycle, and see where those systems are today. This experience allows us to extract several lessons learned that are important for the success and longevity of expert systems.

In this paper we would like to emphasize the aspects of practical, industrial usage of the expert system technology. The applications are real-life, deployed applications that have been in production use within Nokia or its partners for years. We have personally been working on the development, deployment, and use of these applications.

We feel that this is a good time for a post mortem analysis since now over a decade has passed since the applications were first deployed. It is thus possible to review what happened to those applications, what aspects were important for the future development and maintenance, and which applications or features were able to stand the test of times. Application descriptions in the literature commonly discuss either prototypes or solutions that have been in use for a short time. Such descriptions illustrate the possibilities of new technologies but they fail to cover issues that become visible after a longer period of serious use.

Grandon Gill (1995) statistically studied the status of the early successful expert systems to see what had happened to those systems in a decade. Only about a third of the systems that were initially reported as successes were still in use. The reasons were found to be individual and organizatorial rather than technical or economical. The important issues highlighted in that study were: coordinating ES development with the IT and business strategies of the firm, understanding the task to be performed by the system, recognizing the legal implication of systems, identifying user concerns and expectations, and managing developers and development responsibilities.

Guimaraes, Yoon, and Clevenson (1996) tested several hypotheses for expert system success using a sample of 130 expert systems at E.I. DuPont de Memours and Company, Inc. Most of the systems were very small and developed by one person. The study found support for seven of the nine hypothesized determinants of ES success. The important factors were: problem importance, developer characteristics, shell characteristics, and with less confidence: end-user characteristics, ES desirable impact on end-user jobs, and user involvement. The two factors that did not find

* Corresponding author. Tel.: +358-7180-36442; fax: +358-7180-36855.
E-mail address: jukka.k.nurminen@nokia.com (J.K. Nurminen).

0957-4174/03/$ - see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0957-4174(02)00149-5

support were problem difficulty and managerial support. Another very similar statistical study (Guimaraes, Yoon, & O'Neal, 1995) focuses on manufacturing expert systems withing IBM.

Duchessi and O'Keefe (1995) analyzed six cases (three successes, three failures) within three companies focusing mainly on engineering applications. The success factors found in their study include management support, demonstrable business benefits, problem urgency, degree of organizational change, organizational support, and users' personal stake in the system.

Millett and Powell (1996) focused on the organizational aspects. They identified measures for the success of expert systems and used these measures to evaluate and benchmark the performance of one anonymous company. The critical success factors found in that study include: the type of application, importance of the system to business strategy, an early, key, successful development, existence of project champion, existence of an information technology strategy, and organizational culture. Main factors contributing to failure were found to be poor project control, insufficient resources for maintenance, and financial constraints.

Stylianou, Madey, and Smith (1992) performed a field survey among expert system shell users to identify the most important features needed for expert system shells. The most critical characteristics found in that study were embeddability, rapid prototyping, backward chaining, explanation facility, ability to customize explanations, linkages to databases, and documentation compressiveness and readability.

Other authors have focused not only on expert systems but AI research and applications in general. Hayes-Roth (1997) discussed from his subjective perspective different fields of artificial intelligence, tried to find the most successful areas, and presented practical issues for real-life use. Allen (1998) also uses a subjective perspective to discuss the opportunities of AI underlining the connection between AI research and real world needs. Doyle and Dean (1996) summarize the results of the Strategic Directions in Computing Research AI Working group. Among a variety of topics the paper highlights the need to integrate AI research and solutions to other research disciplines and computing systems, collaborative problem solving, and usability of the computers.

The problem with all case-based research is that the results are specific to the applications and companies studied. However, the similarity of results from two companies, IBM (Guimaraes et al., 1995) and E.I. DuPont de Memours and Company, Inc. (Guimaraes et al., 1996), indicates that the conclusions from one organization can be applicable to others. A problem, especially with the statistical studies covering a large group of systems, (Grandon Gill, 1995; Guimaraes et al., 1996, 1995; Parpola, 1995), is that the tested hypotheses are quite abstract. Studies with smaller sample or narrower focus give more concrete, practical advice.

Measuring and collecting detailed data of application development and lifecycle without disturbing it is problematic. One suggested way is to retrospectively look at the data available and consider what kind of hypotheses could be tested with the data (Fenton, 2001). This is the empirical approach we have followed here. This paper complements the previous studies by giving support to some earlier observations, refuting some other ones, and suggesting a set of new observations.

The rest of this paper is structured in the following way. In Section 2 we provide a more detailed overview of the applications. Section 3 discusses our experiences and the lessons learned. Section 4 summarizes the most important observations of this study and suggests ideas for further research.

2. Applications

Starting from 1986 Nokia Research Center had a group focused on knowledge technology, which was later merged with the Software Technology department. The mission of the group was to develop knowledge-based applications for Nokia internal use and for key customers. The emphasis was strongly on practical applications; very little basic AI research was done.

Table 1 lists the most influential and widely used ones of these applications, some of which have been in use for close to 15 years. The table also shows the domain, the lifetime, the software and hardware environment (which may have changed during the lifetime), and the current status of each application.

As can be seen from Table 1, five applications deal with planning types of activities, two with diagnostics, and one is a decision-making system. All of the applications except one (MATIAS) work on the engineering domain.

2.1. NICAD

NICAD (Karonen, 1987) was a knowledge-based system for designing plants and other complex entities. Its first target application was the design of the soda recovery boilers for paper industry. A boiler plant contains thousands of components and hundreds of kilometers of pipes. The design of a new unique plant can take over 30 person years.

In NICAD, the components of the plant (e.g. valves, pipes) were modeled as frames using an object-oriented approach. Each frame contained attributes that described the needed features of the corresponding component. Most of the attributes were geometric but there were also other types such as materials and standards.

Each attribute could be associated with a calculation rule that tells how the value of the attribute depends on other attributes. The local dependencies between attributes defined global dependency graphs. These graphs were

Table 1
The main expert system applications developed at Nokia Research Center in 1986–1990

Name	Domain	Lifetime	Platform	Status
Planning and configuration				
NICAD	Intelligent CAD for engineering applications	1986–today	KEE, Symbolics → C, Sun, Microsoft Windows	Since 1989 continued within an independent company design power (http://www.dp.com) by the name Design++
RFT	Electronic CAD system for mobile phones	1987–today	Lisp, HP → C++, Microsoft Windows	Intelligent GUI in use as part of the Aplac simulator package. Since 1998 continued within an independent company Aplac Solutions Corporation (http://www.aplac.com)
Nokia planner	Project planning	1987–today(?)	Lisp, Apollo → C, Microsoft Windows	Since 1992 continued within an independent company ViSolutions by the name Visual Planner. Merged with ICL 1996
CabCon	Cable configuration	1989–1996	Xi Plus, Genesis, NICAD → C	In trial use in Nokia Cables until 1996
DXCON	Telephone exchange configuration	1990–1997	SQLWindows, Microsoft Windows	In production use in Nokia Networks until ~1997
Diagnostics				
DMG	Diagnostics of radio links	1988–1992	Lisp, apollo, Sun, HyperExpert, PC	In production use in Nokia Networks until ~1992
DXLib	Documentation management	1990–today	Lisp, Sun and C++, Microsoft Windows → www	In production use in Nokia Networks. DXLib model is still part of product shipment.
Decision-making				
MATIAS	Loan application analysis	1986–today(?)	Xi Plus/GURU, PC	In production use at OKO bank Finland

two-directional and NICAD behaved like a 3D spreadsheet. On one hand, the dependencies were used in the calculation of the value of an attribute. On the other hand, if you changed a value, the resulting modifications were propagated in the opposite direction in the dependency graphs.

The whole plant was described as a hierarchical product structure. E.g. a power plant consists of the boiler and the building, the boiler consists of the furnace and the economizer, etc. The product structure contained also calculation rules for the numbers of components (e.g. the number of support pillars), which changed dynamically during the design based on the dependencies.

The calculation rules were expressed by means of a simple language, including Lisp functions and keywords like right, left, behind and above: *object C is to the right of object A by 10 units and under object B by five units.*

NICAD was developed initially in the KEE expert system environment but was later ported to C and Sun. The current version of the tools, under the name Design++, runs on Windows NT platform. Interfaces to ODBC compliant databases and APIs for C, C++ and Visual Basic are supported.

In 1989 a separate spin-off company, Design Power Inc., located in Silicon Valley, was founded to handle the commercialization and further development of the tool.

2.2. RFT

RFT was a design support system for analog and especially radio frequency (RF) circuits. It was developed mainly for the needs of Nokia Mobile Phones. RFT is discussed in greater detail in Ketonen, Lounamaa, and Nurminen (1988).

RFT was in a pilot usage but the complete system never reached a wide user space. One of the reasons was the Lisp implementation language, which tied the system to an expensive engineering workstation. Later the intelligent user interface of the system was reimplemented in C++ for the Windows platform. The user interface is still used today together with the APLAC simulator.

RFT was based on three main techniques. First, the graphical user interface allowed the direct manipulation of circuit diagrams. This was the most successful part of the system and is still in use today.

Secondly, a rule-based layer was used to hide the details of the tools from the users. A number of separate tools were needed for RF design at that time. Since each tool had their own relatively unfriendly user interfaces the idea was to create a knowledge-based layer that would take care of transforming a user-specified problem to the appropriate

form for each tool. In this way the user would be able to work with familiar schematic circuit diagrams. The symbolic manipulation at the knowledge-based layer would take care of the details, constraints, and complex syntaxes of the underlying tools. Although this concept worked well on the pilot cases it was not flexible enough. A further problem was that hiding too many details was confusing to the user.

The third focus area was the design synthesis. The idea was to collect design rules from experienced engineers and thus allow the distribution of the knowledge to more inexperienced persons and to geographically different locations. This turned out to be extremely difficult. Creating a prototype system was relatively easy but updating and maintaining the design rules became a problem. Additionally, automated design synthesis was not flexible enough since, as observed also in Simon (1995), the goals and constraints are changing during the design process. The creativity of the user cannot be substituted by an automated solution.

2.3. Nokia planner

Nokia Planner was a project management tool. The initial goal was to automate key project planning and project tracking tasks within Nokia. After the initial prototype was built in Lisp on Symbolics the project focus shifted from automation to visualization. With a graphical user interface running on early versions of Microsoft Windows Nokia Planner provided a graphical user-interface that allowed easy manipulation of activities and their interdependencies as well as resources. An automatic feature was available for scheduling the tasks to minimize the project duration. However, it turned out that the users mostly relied on manual manipulation of the activities since this allowed them more flexibility.

Nokia Planner was 1992 moved to an independent company ViSolutions that continued the development and marketing of the tool under the name Visual Planner. ViSolutions was merged with ICL in 1996.

2.4. CabCon

CabCon was a system for cable configuration (Karonen, 1989). The design of a cable is much more complex than the simple outer sheath lets us believe. The topology (what and how many components), materials, and geometry (also 3D) of the cable must be determined to meet the requirements of the client including transmission capacity and quality, purpose of usage, strength, maximum allowed weight, latest time of delivery, etc. In addition to the mechanical, thermal, and electronical properties of the cable itself, also the overall production situation has an influence on the priority of alternatives. The general goal is to find out a feasible solution involving minimum costs.

The objective of the CabCon tool was to increase productivity and quality of cables design and manufacturing at Nokia. The subgoals were: prompt and high quality responses to customer inquires, lower costs in production, documentation of the design knowledge, standardization of delivered products, and reduced time-lag between design and delivery.

The project started in 1988, and three prototypes were implemented using three different expert system shells, respectively. Exploiting the results of the complementary prototypes, a small production version was implemented in C in 1989. The system was in limited, mainly experimental, use until 1996.

CabCon did not fully satisfy the users' needs for two main reasons. First, the repertoire of cable structures needed is so large that the CabCon rules were not able to generate versatile enough solutions. Secondly, a graphical user interface was missing which made it difficult to update, customize, and manipulate the configurations proposed by the system. These two shortcomings amplified each other: automatically generated solutions were not satisfactory and their manual improvement and fine-tuning was too clumsy.

2.5. DXCON

DXCON was a system for configuring Nokia DX 200 telephone exchanges. Configuration of a DX 200 switching system was a tedious task, and only a few experts were available. Equipping required both brute force and good command of the frequently evolving design rules. DXCON was meant to be an expert tool for design, sales, and production personnel to increase the productivity and the quality of equipping and price setting of DX 200 switching systems.

Some of the equipping phases were totally automated, but for the more sophisticated design tasks interactive tools were considered as the most optimal compromise.

A switching system consists of rack rows, racks, subracks, plug-in units, and cables. A central concept in DXCON was an equipping model, e.g. for each rack type there was a template describing what kind of subracks can be equipped and what are the feasible locations for those subracks. For each cable we had to define its type, source and destination, routing, and length.

In full-scale production use the dominant way was to reuse and incrementally modify earlier complete switching system models created by the end-users themselves or by more senior experts.

DXCON was running on a standard 386 personal computer with a relational database package called SQLWindows by Gupta Technologies.

One of the main problems of DXCON was its speed. Even after several optimization rounds some operations required about half an hour to complete. Complex DB schema, queries with several joins, and big DB size were main reasons for the slow operation.

During its lifetime starting from 1990 DXCON was in wide use and several hundreds of switching system

deliveries were configured with it. Its use was terminated when the importance of the fixed telephone exchange business decreased.

2.6. DMG

Fault diagnosis of complex technical equipment is always a big challenge. One solution for the problem has been model-based expert systems, where a predefined decision system guides the analysis and is able to identify the fault. DMG (Tyrväinen, 1991) was used to create and edit decision trees that were used in the core of the diagnosis system.

The DMG system had two modules: an engineering workstation, where the equipment model was created and where a decision tree was extracted from the model. The tree was then transferred to the end user environment, which was a standard PC with an expert system shell that was able to execute the decision tree.

The system was used at radio link manufacturing. The link equipment was tested and fixed before they left the factory.

2.7. DXLib

DXLib (Tyrväinen, Saarinen, & Hätönen, 1993) was a system for document management and information retrieval from documentation of DX 200 based network elements. Its main components were a text analyzer that produced SGML tagged formatted documents and a browser that linked documentation to the logical model of the element. The objective of the system was to assist human expert in finding relevant information.

Like DMG the system consisted of two modules. An engineering workstation, where a logical model was created, was used to link the model and relevant texts together. The model and the documentation were then transferred to a PC environment, where they could be browsed. The system has been used ever since its first trial in 1993. The model is an optional feature of the electronic network element documentation package that accompanies network deliveries.

2.8. MATIAS

MATIAS (Kontio, 1991) was an expert system for loan application analysis that was developed for OKO bank, which at that time was an important customer of Nokia Data. It contained knowledge about farming loans in Finland. The regulations for such loans including state subsidies were quite complex and MATIAS was meant to speed up the loan decision process.

Containing a few thousand if-then rules divided into multiple modules MATIAS was a large expert system at its time. Most of the knowledge used by MATIAS existed already on documents about the loan policies and regulations. However, a large number of rules was needed to codify this knowledge. The large size of the rulebase resulted into the usual problems of verification and validation of the rulebase as well as difficulties in maintaining the rulebase.

3. Characteristics of successful systems

We have collected our main observations in Table 2. The 'Hypothesis' column contains properties or statements that are characteristic of successful expert systems. These hypotheses are either derived from literature or are based on our own observations.

The 'Source' column provides references to literature that has suggested a given hypothesis. It is difficult to track the ultimate origin of each observation or hypothesis in the literature. Therefore we have mentioned those studies where we have encountered a mentioning of a given hypothesis. Also different authors have formulated their findings in different ways. We have tried to understand what has been the spirit of the observation when listing which authors have discussed each hypothesis.

The 'Evidence of our case study' column indicates general level of support this study gives for each hypothesis. The following columns contain more detailed analysis for each application. A '+' means that the application supports the hypothesis, a '−' means it refutes the hypothesis. An empty value means that the aspect was not relevant or observed for the application or that the conclusion is not clear.

The observations are divided into four categories: domain, development, operation, and general. The hypotheses and the case study evidence for each of them are discussed in greater detail in the following sections.

3.1. Domain

3.1.1. Narrow scope

Hayes-Roth (1997) suggests that narrowing the scope is important for expert system success. Naturally it is easier to develop a system with a narrow scope. However, the impact of such a system tends to be low because the solution is too limited.

Fig. 1 shows 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

Table 2
Case study evidence for the hypotheses

Hypotheses	Source	Evidence of our case study	NICAD	RFT	CabCon	DXCON	DXLib	DMG	MATIAS	Planner
Domain										
Narrow scope	Hayes-Roth (1997)	Mixed	−	−	−	−	−	+	+	−
Focused objective	Hayes-Roth (1997)	No	−	−	−	−	−	−	−	−
Stability of environment	Hayes-Roth (1997)	No	−	−	−	−	−	−	−	−
High degree of automation	Hayes-Roth (1997)	Mixed	+	−	−		−	+	+	−
High degree of repetition	Hayes-Roth (1997)	Yes	+	+	+	+	+	+	+	+
Small project	Hayes-Roth (1997)	Yes	+	+	+	+	+	+	+	+
Development										
Users prefer usability over automation	Nurminen, Karonen and Hätönen (2003)	Yes		+	+	−	+	+	−	+
Expert systems complement rather than replace human experts	Millett and Powell (1996)	Yes	+	+	+	+	+	+	−	+
Early benefits to experts themselves are important	Nurminen, Karonen and Hätönen (2003)	Yes	+	+	+	+	+	+	−	
AI should be an embedded part of a bigger system	Duchessi and O'Keefe (1995) and Millett and Powell (1996)	Yes	+	+	+	+	+	+	−	+
Simple, straightforward solutions work best	Nurminen, Karonen and Hätönen (2003)	Yes	+	+	+	+	+		+	+
If-then rules considered harmful	Nurminen, Karonen and Hätönen (2003)	Yes		+	+			+		
Lots of custom work	Hayes-Roth (1997)	Yes	+	+	+	+	+	+	−	+
Fast and agile development is important	Duchessi and O'Keefe (1995)	Yes	+	+	+	+	+	+	+	
Knowledge-based applications are based on domain specific application engines and generators	Nurminen, Karonen and Hätönen (2003)	Yes	+			−	+	+	+	
Monolithic applications are sometimes better than knowledge-based applications	Nurminen, Karonen and Hätönen (2003)	Yes		+	+					+
Rules of normal SW development apply	Duchessi and O'Keefe (1995) and Millett and Powell (1996)	Yes	+	+	+	+	+	+		+
Operation										
Move towards mainstream software and hardware platforms	Nurminen, Karonen and Hätönen (2003)	Yes	+	+	+		+	+	+	
Successful systems tend to move out from the company	Nurminen, Karonen and Hätönen (2003)	Some	+	+		−	−	−		+
General										
The difference between expert systems and information systems is small	Duchessi and O'Keefe (1995)	Yes	+	+	+	+	+	+		+

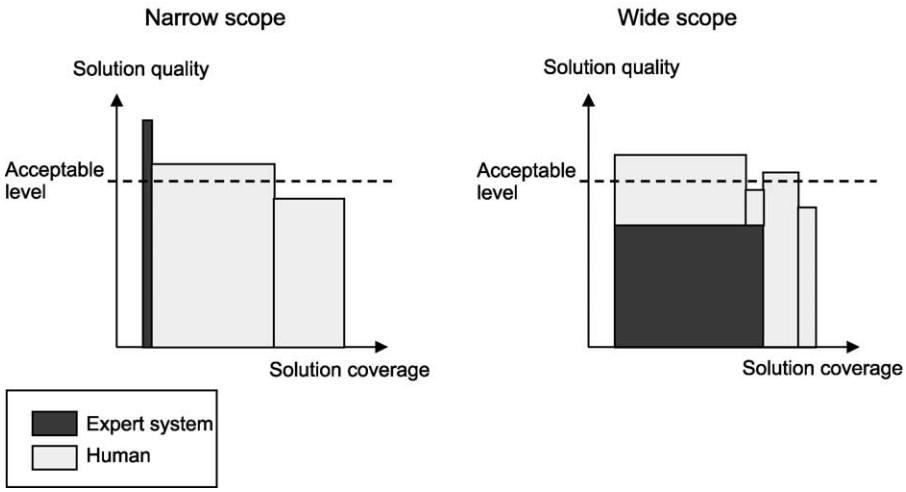

Fig. 1. Narrow vs. wide scope.

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 Fig. 1 shows 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.

Most of the applications of this study had a wide scope. Only the loan decision-making tool (MATIAS) and the radio link diagnostics tool (DMG) had a narrow scope. When the problem domain is large, like for the planning applications, the wide scope allows some support for most of the tasks.

3.1.2. Focused objective

Hayes-Roth (1997) states that focusing on a single objective, such as to reduce time, to reduce cost, or to produce a better design, is beneficial. Our experience strongly contradicts this. In all of our applications we have had multiple goals. Typically we have tried to improve quality, increase speed and save time at the same time.

The need to emphasize multiple goals is especially evident in planning tasks. Simon (1995) observes that design work has multiple goals and constraints and that their priority and importance changes during the design process. Our experience is very much in line with this. For any reasonable complex planning task it is impossible to list all of the objectives in advance. Focusing the design on the optimization of a single criterion tends to create plans that are not feasible in practice.

3.1.3. Stability of environment

Stability of environment is another issue where our experience strongly contradicts (Hayes-Roth, 1997). A stable environment helps the implementation but the assumption of a stable environment is too idealistic. The shortening release cycles of all kinds of products make the environment even more dynamic than what is has been in the past. It is essential that the expert systems are useful in changing environments. Otherwise they would either be outdated or their maintenance would be a constant problem.

Systems with a narrow scope have a high risk in this respect. As can be seen in Fig. 1 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 from the problem domain. For the wide scope systems this risk is much smaller.

3.1.4. High degree of automation

High degree of automation may seem like a desirable feature. As described in Hayes-Roth (1997) this would cause savings in the operation of the system.

In this issue our evidence is mixed. The narrow scope systems, MATIAS and DMG, provide a high degree of automation. In DXCON and NICAD some parts are highly automated while others require a lot of user involvement. After the planner had created the configuration DXCON took care of hardware details and thus saved work and reduced errors. NICAD automated a lot of the calculation even if human experts made the design decisions. In the other applications the general level of automation was not very high.

We can see several explanations for these observations. First, MATIAS and DMG were intended for non-expert users. All the other applications aimed at improving the performance of an expert user, which seems to be common in engineering applications.

Secondly, increasing the degree of automation requires increasingly more development effort. When the problem

complexity and width increases reaching a high degree of automation may become infeasible goal. In terms of Fig. 1 the question is how much width is sensible to sacrifice to increase automation.

One practical way is to develop some modules with high degree of automation to automate routine tasks and rely on the end-user in the more difficult tasks. Modules containing calculation or detailed but straightforward computation are good candidates.

3.1.5. High degree of repetition

High level of repetition is clearly useful. It allows the benefits from the system to be drawn continuously and thus makes the investment in the system worthwhile.

In addition to the benefits that are gained from using the system the high degree of repetition is also useful for the development. More repetition means more feedback to the development, more need for new features, and more incentive to fund the further development and maintenance.

3.1.6. Small project

Our expert system projects have been small with less than 10, typically 2–3, full-time developers plus part-time experts.

We do not have hands-on experience with the larger expert system projects. With our data we can only conclude that small projects work but the data does not say anything about large project. Our subjective feeling, based on experience with expert system and large-scale software development projects, is that large-scale expert system projects should not be more difficult than any large software projects as long as the expert system parts are developed as compact, well-defined, and reasonably independent components.

3.2. Development

3.2.1. Users prefer usability over automation

Finding the proper balance between usability and automation seems to be an issue in most projects. Since the applications are developed with limited resources it is important to focus the development to the most useful direction.

The experience with MATIAS and with some modules of DXCON indicates that when the degree of automation is high the usability is not considered very important. If the automatically generated solutions are fully correct the user has no need to analyze and interpret the results.

The situation is different in applications where full automation is not feasible. In that case usability is a major issue since users need to control the application, understand the solutions, analyze the trade-offs, and manually adjust the solutions until they satisfy the needs. This seems to be the most common case. Automation is valued but only up to a certain level. The users want ultimately to be in control and understand the solution before they can accept it.

This division also mirrors the difference between expert and ordinary users. MATIAS brought the loan knowledge to the customer agents in bank offices. DXCON had two classes of user: senior experts who defined the component configurations and less experienced expert users who utilized the expertise. In the other applications the users were typically the experts themselves. It seems that in tools that are intended for the experts the usability is very important. The expert can compensate for missing automation by using his experience and competence. The ordinary user is more or less blindly relying on the results of the tool and therefore is not very interested, or competent, to analyze the rational behind the results.

Graphical user-interfaces naturally have a major impact on usability. In fact, the graphical user interfaces seem to be one of the most important results of long-term practical value of the AI boom in late 1980s. Today the graphical user interfaces are commonplace but 15 years ago the expert systems were among the first applications with intuitive user-friendly graphical interfaces.

3.2.2. Expert systems complement rather than replace human experts

A key characteristic of our applications has been that they are intended for expert users such as product designers or electrical engineers. In such a role the tools support, rather than replace, the expert.

According to our experience for any non-trivial planning task it is not possible to create completely automatic solutions. An expert user is essential to handle changing and unexpected needs, multiple objectives, and other similar aspects where the human flexibility and a wide understanding are important. Systems which combine the strengths of both humans and computers, intelligence amplification systems (Brooks, 1996), seem thus to be the most promising direction for practically successful applications.

Millett and Powell (1996) made a similar observation that ‘one reason for success is that expert system has not attempted to undertake the whole task, merely a part of it that is time consuming for staff. This frees experts for more complex tasks not handled by the system’.

Requiring the user involvement may seem like a failure from a puritanist AI perspective but from the practical perspective it solves many problems and results into less complex systems. Requirements for the completeness and correctness of the system can be less strict. The system can consist of a set of basic tools that are controlled by the user. Changes and new demands do not necessarily require a modification to the system but can be handled by the end user. As a result the application development is easier and the applications are more adaptive to external changes.

3.2.3. Early benefits to experts themselves are important

The experts with the relevant knowledge and understanding of the problem are key to expert system success. Typically the experts are busy, which is one reason why it makes sense to develop the expert system. There is often

a conflict between the short-term and long-term priorities when the expert schedules his time between different tasks.

Our experience suggests that one very useful practice to involve the expert is to develop the system in such a way that it offers benefits almost immediately to the experts themselves. It is not enough to satisfy the end-users. It is equally, and sometimes more, important to satisfy the expert to be able to create a good enough system to be delivered to other experts and end-users.

When the experts can directly influence the development their attitudes tend to be more positive towards the new systems. Early participation in development reduces the often-perceived risk that the expert will be replaced by the expert system. Instead active involvement emphasizes the importance of the expert and the tool makes the expert more competent.

Duchessi and O'Keefe (1995) state a more general observation that a system that directly benefits a user fosters operational use. While we agree with their observation we feel that emphasizing *early* benefits and to the experts themselves is important. Otherwise the development easily slows down and stops. At least one of the successful cases that (Duchessi & O'Keefe, 1995) analyzed seems to confirm this.

3.2.4. AI should be embedded part of a bigger system

All of the applications, except MATIAS, show that expert system modules cannot be handled in isolation. To be useful they have to be embedded to larger software systems. The expert system modules perform key tasks and provide the intelligence of the application but without the other parts the intelligent parts would not be useful alone. Millett and Powell (1996) found that stand-alone systems made users unhappy by requiring existing data to be re-keyed. Two of the three successful expert systems in Duchessi and O'Keefe (1995) were actually database applications where the expert system module was 'icing on the cake'.

The first implication of this is that expert system modules are most useful when implemented in the same environment and same tools as the rest of the application. Often this means the use of a standard programming language like C++ or database centric solutions, like in DXCon. It still makes sense to specify the expertise in special format in definition files that are either loaded to the application or compiled and linked as part of the application.

Another implication is that in the development a lot of effort has to be spent on the non-AI parts. This work involves, for instance, the graphical user interface as well as interfaces to external systems, and can easily exceed the effort needed for the expert system parts. In the project planning it is thus important to allocate enough effort to all parts of the system and avoid the mistake of focusing only on the key expert system parts.

3.2.5. Simple, straightforward solutions work best

If the ambition level is too high it easily happens that the system uses sophisticated techniques and solves limited subsets of the target problem very well, but it is not able to handle the real size problems.

In almost all of the applications in this study the experience was similar. The objectives for the degree of automation were at the start much higher than what was accomplished in the end. The advanced AI modules were pruned and the mainstream technology modules started to dominate the development.

Better results can be achieved when the system focuses on solving the practical problem. Millett and Powell (1996) found that successful organizations were less concerned with pushing the technology and more inclined to concentrate on the application.

A typical sin on the expert system area has been to try to force a certain solution to a problem. Instead it is important to consider which solution approach would the optimal one for a particular problem. Although expert systems are good solutions for some problems, for other ones e.g. the use of optimization and other mathematical algorithms could be better choices.

Developers with a wide understanding of possible solutions techniques have a key role. Naturally finding developers with such broad experience is difficult.

3.2.6. If-then rules considered harmful

The analysis of the applications of this study indicates that if-then rules are not the right way to represent engineering knowledge. This finding is in contrast with the study (Stylianou et al., 1992) that found support for backward chaining to be the third most important feature in expert system shell selection.

MATIAS, the only non-technical expert system of this study, was the only deployed application that successfully used if-then rules. A few of the other applications also experimented with using if-then rules on some parts but these trials did not give good results.

Technical knowledge deals with artifacts and their relationships, formulas, and numeric values that are more easily expressed as classes and objects. Rules are more suitable for describing knowledge on the area of legal reasoning, such as the terms of loan approval.

Another observation is that the maintenance of if-then rulebases is more difficult than for most other knowledge representations. Instead of the general if-then format a special application specific structure is useful for easier development and maintenance. The preexisting solutions available in application development tools are seldom suitable as such. Finding the right structure is very similar to software architecture design, which is a key step for successful software development.

3.2.7. Lots of custom work

The large individual differences between developers (see e.g. McConnell (1993)) are likely to be even bigger in expert system development than in normal software development. The reason for this is that, being in the intersection of AI and

information technology, expert system development requires more creative solutions than normal software development. The task does not only require good software development competence but also good understanding about the applicability, limitations, and usefulness of different solutions techniques. Because many expert systems are pushing the state of the art it is not possible to blindly follow the solutions used in other systems.

3.2.8. Fast and agile development is important

In the engineering domain the expertise is developing on a fast speed. This has two implications. First, rapid development is important to bring the solution into use at the right time. If the development time is too long then the system may not be up to date anymore when it is ready. Secondly, without proper maintenance and adequate tools for it the system very soon becomes obsolete.

Fast and agile development process is thus important. At early development phases flexibility is needed to discover the way to approach the problem and to find the right architectural solution. Also, as discussed above, it is important to bring the tool early into use to ensure the expert commitment and feedback and continue the development in an iterative and incremental fashion. Although Duchessi and O'Keefe (1995) do not explicitly state the importance of this it seems that a user driven phased development has been used at least in the successful systems that they analyzed.

The agile development is another area besides graphical user interfaces where the expert system community has done pioneering work. The incremental, customer driven style of expert system development has only recently become accepted, and increasingly popular, in software community at-large (see e.g. Cusumano and Yoffie (1999)).

3.2.9. Knowledge-based applications are based on domain specific application engines and generators

As shown in Fig. 2 the applications of this study can be divided into three groups

- *Monolithic applications* have no clear difference between knowledge and application logic. Monolithic applications are very close to general IS applications.
- *Knowledge-based applications* separate the knowledge and the processing of knowledge (engine). This is the traditional model for expert system applications. The same engine can be used with different knowledge to create different applications. Maintenance of the application is supposed to be easy since the changes are mainly made to the knowledge part.
- *Application generators* are not intended for end-users but for developers. Their main purpose is to assist in the application development. Since the development and maintenance of a knowledge-base is typically difficult, the generator applications are often essential in making the knowledge bases more robust, easier to understand, and faster to develop. The generators typically work by representing the knowledge to the developer and to the expert in a form that is easier to understand and manipulate. In the application the primary goal of knowledge representation is efficient execution, in the generator it is human comprehension.

This study suggests that the textbook solution to use a general-purpose expert system shell and to concentrate on the knowledge base in the application development does not work. There is a need to create very specialized, domain specific tools for the engine (NICAD) or for knowledge base development and maintenance (DMG, DXLib). Such tools take advantage of the special structure of the knowledge in a domain and make the knowledge more manageable by constraining the knowledge to a more compact, consistent and modular format.

MATIAS was the only application developed with an expert system shell but the experience with it also triggered the development of knowledge engineering tools (Parpola, 1995) to better handle the knowledge acquisition process.

DXCON was the only application that did not support this observation. An explanation for this is that DXCON was a database centric application and stored the knowledge in

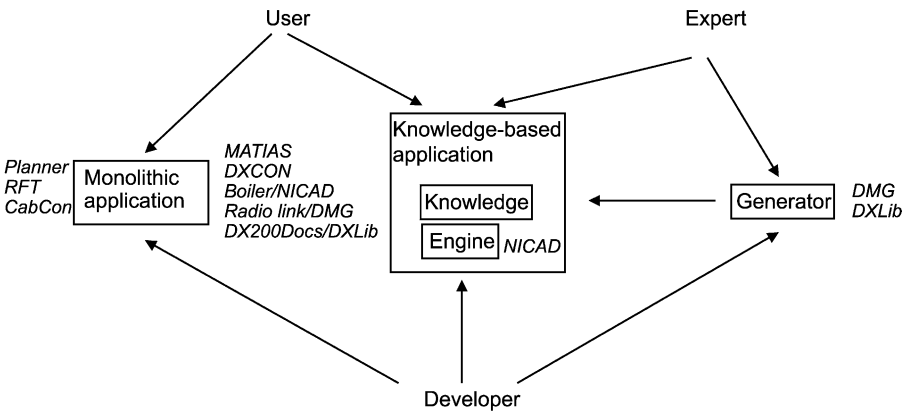

Fig. 2. Applications and their types.

the form of configurations components to relational database tables. This kind of open architecture allowed easy addition of new modules developed with different technologies.

Own development effort is needed to find the right structure for the knowledge but after that initial effort the rest of the development becomes a lot easier.

Developing the engines and generators requires major work. The knowledge-based system creation is thus only partly about the knowledge acquisition. It is also about software development to create suitable tools that can be efficiently used to build the applications. For instance, at the beginning most of the work on the NICAD application had to be spent on developing the engine.

The situation may have changed and today there are more tools that can be used as building blocks for the applications. Rather than expert system tools such software is marketed as domain specific automation tools.

3.2.10. Monolithic applications are sometimes better than knowledge-based applications

For most end-users and applications the difference between monolithic applications and knowledge-based applications is small. A major difference can arise if the normal operation of the system requires that experts dynamically update the rules.

The division has a bigger effect for the development and maintenance of the applications. Knowledge-based applications are often seen as the orthodox way to develop expert systems. Also in our case that was the starting point of all the applications. However, as the development progressed Planner, RFT, and CabCon shifted to the monolithic style. This was mainly caused by the fact that the importance of application specific knowledge turned out to be less important than initially expected.

Knowledge-based application have their price. The development of a knowledge-based application, in our case, required also the development of the engine and generator tools. Reaching the needed generality with such tools requires development effort. Separating the knowledge from the engine also tends to reduce the execution speed.

Our observation is that the selection between monolithic and knowledge-based application depends on the type of knowledge. If the knowledge is quite static and not growing, if the amount of knowledge is not very large, and if its representation with mainstream information technology is easy then a monolithic application can be a good choice. Proper software architecture makes it also possible to modify the knowledge. Naturally it is harder than in the case of a knowledge-based application.

Further characteristics of successful monolithic application seem to be a wide and general application domain, high importance, and expert users.

3.2.11. Rules of normal SW development apply

Our experience suggests that there is very little difference between software development and expert system development. Most of the successful solutions in our applications are very close to normal software development. Likewise the process of developing an expert system and developing normal software is very similar. There are differences in emphasizing the importance of different steps rather than the actual work. Our experiences confirm those of Duchessi and O'Keefe (1995) who stated that factors leading to success in information system and decision support system development are also important for expert system implementation.

The normal software process activities like development, testing, and maintenance easily become very difficult when a larger knowledge base is needed. This is especially problematic if the modularity of the knowledge is low. If the knowledge is stored in rulebases with little internal structure the number of alternative execution paths becomes very high. It is no longer possible to understand how the rules interact, is the rulebase complete, or what happens when something unexpected happens. Testing all of the execution paths is clearly beyond practical limits.

Moreover, the maintenance work easily is very difficult. Maintenance is the biggest consumer of software development resources. Some estimates assume that 70% of software lifetime cost is spent on maintenance (Boehm & Basili, 2001). Millett and Powell (1996) observe that expert systems are no different from conventional systems in that they require maintenance. To us it seems that the maintenance of knowledge bases is even more expensive. First, the expertise tends to change frequently. Secondly, the complexity of expertise and its presentation in knowledge bases makes incremental modifications difficult. Finally, low modularity may result into the need to retest the whole application after each small modification. It can also happen that different experts view the functionality from different angles. It is thus possible that a rulebase that is developed with one expert and is later maintained by another is no longer consistent.

One of the big successes in software engineering has been object-oriented programming and the software components. They have made the software reuse a practical issue. How is the reuse of expertise? For successful reuse the expertise would need to be packaged into some kind of component. Moreover, a mechanism would be needed to control, expand, and modify the behavior of the components from the outside. At least rulebases are quite sensitive to small changes. Already a change in a single parameter value in one rule requires a thorough understanding about the possible effects of the change. Other domain specific formats are potentially better but still there are major difficulties for black-box reuse. Domain specific engines and generators are more applicable for reuse than the knowledge itself.

3.3. Operation

3.3.1. Move towards mainstream software and hardware platforms

The hardware platform tended to shift from specialized computers, Symbolics, to Unix workstations and personal

computers running Microsoft Windows. The applications that were initially developed with Lisp or expert system shells were converted to C or C++.

There were several reasons for this shift. For widely used applications the software and hardware platform cost is essential. Furthermore, with the rise of the personal computer, users tended to prefer to do all of their work on single computer using tools that had a familiar look-and-feel. The mainstream environment also allowed the creation of open interfaces to link the expert system applications with other applications.

It also turned out that the platform support for expert system tools was not adequate. The companies providing specialized AI platforms, Lisp programming environments, and application generators were working on a niche market. The volume of the market was not large enough to guarantee the high level of robustness, development speed, and support that is needed for serious products. Moreover, most of the expert system environments were closed systems and did not easily allow the use of normal, mainstream development tools such as version control systems, makefiles, debuggers, etc.

Even if porting the application to new environments caused extra work this approach was, in fact, very productive. The developer-oriented environments and application generators allowed a fast start and enabled early end-user involvement. When the user needs were understood and the appropriate solution found in this way, it was relatively straightforward to reimplement the relevant parts of the solution in another environment.

Another benefit of the platform change was that it was a good opportunity to improve the speed and robustness of the application. When the needs were clear it was possible to choose architectural solutions that allowed efficient and future-proof applications to be written.

3.3.2. Successful systems tend to move out from the company

Our data gives some support to the observation that successful systems often lead into a spin-off company that focuses specially on the further development and maintenance of the expert system or the platform tools. Even if the balance is even, three cases supporting and three refuting the claim, the applications supporting the claim are the most successful ones in terms of user base and longevity.

There can be a result of numerous factors in the operating environment and in strategy of our company that explain this observation.

Expert systems, in their role as support tools, improve the operational efficiency but are not the primary products of a larger company. Their benefits for the main business are indirect and hard to quantify. Therefore, with increasing pressure to focus on the core business, the development and maintenance of such system may not satisfy the profit requirements.

Application development requires that a lot of effort is spent on monolithic applications or on domain specific engines and generators. It thus becomes attractive to reuse the platform in order to divide the platform development cost between several applications. Such a generalization, which e.g. happened in NICAD, results into the creation of application specific environments that can be used as a basis of a product family. In our case the monolithic application covered a wide domain (project planning) and important special area (analog circuit design) and therefore had a large market and good business potential.

A dedicated spin-off company may be in a position to handle the further development and marketing of such tools. While the mother company is mainly interested in the applications, the spin-off company can focus on the expert system technology, develop a variety of solutions around the platform, and market them to different customers. It also seems that the platform technology is more stable and easier to maintain than the actual applications.

3.4. General

3.4.1. The difference between expert systems and information systems is small

Duchessi and O'Keefe (1995) expected that many of the factors that lead to successful implementation of other systems are also important in expert system implementation. According to our experience many of the organizational and system development issues are common between expert systems and information systems.

First, it is important to understand that the applications are essential, not the technology. Secondly, it is important to develop the applications to solve the right problems at the right time. Millett and Powell (1996) state that it is important that an expert system project tackles a major business problem at an early stage. Duchessi and O'Keefe (1995) found that problem urgency and growth of the underlying application area are important factors for expert system success. Thirdly, the management support, matching the solution to the organizations needs, and right organization support are important (Duchessi & O'Keefe, 1995).

Our experiences support all of these observations. For the expert system development it would be important to utilize all the accumulated experience of information system development. Actually, we feel that one of the mistakes of expert system development may have been to view it as a distinctive area and develop own processes, tools, and methods independently from the information systems. Potentially, better results would have been achieved if the problems had been approached from the information system perspective and the special expert system techniques would have been utilized only when appropriate.

4. Conclusions

The domain of engineering applications is complex: technology is changing, tasks are open-ended, and multiple, often implicit, goals are common. Traditional, standalone expert systems have problems to cope with these challenges.

Contradicting previous studies, this study 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. In fact, 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.

Fast and agile development was found to be important to cope with the changing environment. 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.

The problem with empirical evaluation is that a single study provides only a piece of evidence. The analysis of multiple systems and the long time period increase the confidence of the observations of this study. However, numerous factors, like the culture of our development group, may bias the results. Moreover, it is not clear how sensitive the results are to the type of industry, application domain, or point-of-time.

The findings of this study are based on observations about eight successful applications. It would be interesting to deepen the analysis and look for more fundamental reasons behind these observations. The development of such theory would give more confidence to the observations and provide a general framework where the results of case studies could be positioned.

References

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

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

Brooks, F. P. (1996). 1994 ACM Allen Newell Award acceptance speech. *Communications of the ACM*, (March).

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

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

Duchessi, P., & O'Keefe, R. M. (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.

Grandon Gill, T. (1995). Early expert systems: Where are they now? *MIS Quarterly*, *19*(1), 51–81.

Guimaraes, T., Yoon, Y., & Clevenson, A. (1996). Factors important to expert systems success—A field test. *Information and Management*, *30*(3), 119–130.

Guimaraes, T., Yoon, Y., & O'Neal, Q. (1995). Success factors for manufacturing expert system development. *Computers and Industrial Engineering*, *28*(3), 545–559.

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

Hayes-Roth, F. (1997). Artificial Intelligence—What works and what does not? *AI Magazine*, (Summer), 99–113.

Karonen, O. (1987). Intelligent CAD. *Norwegian Artificial Intelligence Society (NAIS) Seminar*.

Karonen, O. (1989). Experiences with different application generators in cable configuration (in Finnish, original title *Projekti jossa jouduttiin vaihtamaan kehitintä). BLANKO'89 conference*.

Ketonen, T., Lounamaa, P., & Nurminen, J. K. (1988). An electronic design CAD system combining knowledge-based techniques with optimization and simulation. In J. S. Gero (Ed.), *Artificial intelligence in engineering: design*. Amsterdam: Elsevier.

Kontio, J. (1991). Matias: Development and maintenance of a large but well-defined application. *Expert Systems with Applications*, *3*(2), 241–248.

McConnell, S. (1993). *Code complete*. Redmond: Microsoft Press.

Millett, D., & Powell, P. (1996). Critical success factors in expert system development: A case study. *Proceedings of the 1996 Conference on ACM SIGCPR/SIGMIS Conference*, 214–222.

Parpola, P (1995). *Object-oriented knowledge acquisition*. Licentiate Thesis, University of Helsinki.

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

Stylianou, A. C., Madey, G. R., & Smith, R. D. (1992). Selection criteria for expert system shells. *Communications of the ACM*, *35*(10), 30–48.

Tyrväinen, P. (1991). DMG—Object-oriented iterative knowledge acquisition for the generation of diagnostic hypertext systems. *LISP Pointers*, *4*(1).

Tyrväinen, P., Saarinen, P., & Hätönen, K. (1993). Domain modelling for technical documentation retrieval. In H. Kangassalo, H. Jaakkola, K. Hori, & T. Kitahushi (Eds.), *Information modelling and knowledge bases IV* (pp. 388–399). IOS Press.