# DATALOGI LINKÖPING

UNIVERSITETET I LINKÖPING
TEKNISKA HÖGSKOLAN

PROGRESS REPORT

## SOFTWARE SYSTEMS RESEARCH CENTER

Preliminary Report 1980

2nd (extended) edition

Software Systems Research Center
Linköping University
S-581 83 Linköping, Sweden

December 1980

ISSN 0348-2960

ISSN 0348-2960 December 1980.

# Software Systems Research Center

# 1980

This folder presents the background for the reseach activities, which are being performed at Software Systems Research Center at Linköping University. The presentation is intended to give an overview of the treated problem areas, together with an indication of the lines along which solutions are sought. Actual projects are not described in any detail. It is our intention to produce an up-to-date progress report each fall, presenting the current status of the work within the research center.

The research center is supported by the Swedish Board for Technical Development under contract no 80-3918.

*This 2nd edition of the 1980 status report is extended with a list of the personnel at the center, the plans for an AI project, and additional references to reports published during the fall.*

*Mailing address:*
Software Systems Research Center
Linköping University
S-581 83 Linköping
Sweden

*Postadress:*
Datalogicentrum
Tekniska Högskolan i Linköping
581 83 Linköping

# Personnel:

**Principal Investigator:**

Erik Sandewall, professor

**Laboratory Leadership and Coordination:**

*PELAB (Programming Environments Laboratory):*

Pär Emanuelson
Anders Haraldsson
Jerker Wilander

*ASLAB (Applied Software Systems Laboratory):*

Mats Andersson
Sture Hägglund
Erik Sandewall

*AI Project:*

Jim Goodwin
Uwe Hein

**Research Associates and Graduate Students:**

Anders Beckman
Kari Dubbelman
Kenth Ericson
Peter Fritzson
Arne Fäldt
Jim Goodwin
Christian Gustafsson
Hans Holmgren
Lennart Jonesjö
Jan Komorowski
Magnus Ljungberg
Hans Lunell
Anders Ström
Dan Strömberg
Ola Strömfors
Eva-Chris Svensson
Örjan Svensson
Erik Tengvald
Lars Wikstrand
Olle Willen

**Technical and Administrative Services:**

Anders Aleryd
Mats Andersson
Leif Finmo
Ulla Mathiasson
Sven-Tuve Persson
Katarina Sunnerud
Henrik Sörensen
Lillemor Wallgren

## 0. GENERAL OVERVIEW.

Within the research programme for *Knowledge development in Information Processing*, funded by the Swedish Board for Technical Development, an Experimental Research Center, called the *Software Systems Research Center*, has been formed at Linköping University. Work in the Research Center will be a continuation and extension of the research which has been conducted so far in the framework of the Informatics Laboratory in Linköping. The purpose of this status report is to describe the directions and plans for future work, as it is formulated in connection with the laboratory reorganization, and also to summarize shortly the previous and on-going projects, which we are building on. A catalogue with up-to-date project descriptions will be appended to this report later.

The activities within the center are in certain respects divided into two Laboratories, namely:

1. an Applied Software Systems Laboratory (ASLAB), oriented towards techniques which make computers easier to use for (non-expert) end users or as a tool in a given application; particularly techniques for dialogue between the user and the computer system ("man-machine dialogues") and for modelling (= description) of applications in the computer;

2. a Programming Environments Laboratory (PELAB), oriented towards the design of tools for the programmer, including both the general architecture of such tools (programming environments), specific functions which are needed in such tools (program manipulation), and the underlying, theoretical study of such specific functions, and the properties of programs which make them possible.

The organization of an *Experimental Research Center* is planned in accordance with ideas put forward in the Feldman Report (*Comm ACM 22*, 9) and later supplemented by Denning (*Comm. ACM 23*, 1), with a strong emphasis on methodology development in close interaction with implementation of experimental systems and joint projects with industry and government.

We assume a considerable homogeneity within the research center, both in terms of a common approach to issues of software architecture, such as the emphasis on programming environments and other interactive techniques for software development, and also in terms of a common organization for administrative matters and for technical support (equipment and basic

software). However, the conduct of projects, thesis supervision, industrial liaison, and related issues should to a large extent be handled within each laboratory. Such a division has already become necessary with the current size of our research group, and would become more necessary if additional activities and additional members are added to the group.

The two laboratories are similar to projects in the sense that they may have a shorter life-length than the research center, and since researchers at the center will shift between the laboratories for different activities, for example from one year to the next. Modification of the laboratory structure is foreseen as a way to adapt the research center's activities to new needs in the future.

# 1. THE PROGRAMMING ENVIRONMENTS LABORATORY

## 1.1 RESEARCH OBJECTIVES.

The central goal for the Programming Environments Laboratory (PELAB) will be to develop a theory for design of programming tools. Such a theory must prescribe two things, the content and architecture of the programming environment (PE). As for content, it must prescribe the choice of tools: what program manipulators should be provided. This requires a firm theoretical and practical knowledge of the various kinds of manipulators that exist, from program verifiers to flow analyzers to editors, and of the properties of programs which make them possible. As to architecture, it must prescribe the kind of communication that must occur between different tools and manipulators. To support such communication, it must also prescribe an appropriate shared representation for their knowledge about programs.

Our view of a programming environment is centered around a few main concepts:

1. *Program development as an incremental process.*
   The envisioned environment has a user interface intended to support program design, implementation and maintenance as an incremental process. Programming language issues are important but will not dominate the research.

2. *Integrated system.*
   The overall system design is an integrated system. Tools and other components of the PE are available from other tools, and communication between such tools is an important issue.

3. *Language oriented tools.*
   Tools, such as editors, debuggers, etc., are to be adapted to the programming language in use, e.g. by supporting concepts and structures from that language.

4. *Programs, program fragments and information about programs are stored in a database.*
   A program is no longer primarily represented as a character string on a file. It may be represented in several forms; an internal representation is a structured form to facilitate analysis, transformation and interpretation of the code. Information that is

included in the description of a program is verification assertions, test cases, program analysis results etc.

5. *Advanced program development and maintenance tools.*
The programmer should have advanced tools available when developing and maintaining programs. This includes tools for
   a. program specification
   b. program testing and debugging
   c. program presentation
   d. program analysis and transformations
   e. program verification
   f. program generation
   g. program documentation.

An important problem to study in this area is to find good strategies to transform program from an development/maintenance environment to a production environment, e.g., a program run on a micro computer or a program distributed to several computers in a network.

## 1.2 RATIONALE AND BACKGROUND.

The overall problem is to facilitate software production. In a programming environment advanced tools will be available for program development and maintenance. Today there is a lack of advanced tools. An editor, a compiler, a linkage editor and a simple debugger are often the only tools available, and there is no other advanced support for testing, debugging, program analysis and other aids to handle programs.

The first conventional language for which a requirement for a PE is discussed is for the new ADA language. There is, however, not yet any final specification available. From the language design and from early discussions of the PE it seems that the tools will be of standard nature. In our view, many advanced tools will be missing. Today Interlisp is the best example of an integrated PE. It has been used in artificial intelligence (AI) research for over 10 years and the system has continuously been growing and new facilities are extending the system.

After many years work with the Interlisp system, the implementations of PATHCAL and QLOG, and experiments with the program manipulation system REDFUN, we have created a stack of problems to attack.

Work needs to be done in the overall design of an advanced PE and we want to investigate questions as:
- what representations should a program appear in?
- what information about a program should be derived and maintained, and

how should such information be structured?
- what tools should be available and how should they be implemented and integrated?
- how to administrate program versions, test cases etc?
- how should the user interface be designed, a large system and advanced tools increases the complexity and makes its use more difficult.

Many problems are depending on finding appropriate ways to represent information about programs (e.g., declarative information, analysis results, verification assertions, test cases, specifications and performance statistics) and appropriate specifications of the language in use (e.g., semantics and other properties about the language)

Testing and debugging methods can be improved. More advanced testing methods may rely on symbolic evaluation and verification of assertions, and better handling of test cases and retesting are necessary. Debugging can be improved through better ways to supervise the execution, to present program code and through more advanced analysis tools.

We believe that generation of tools will be an important ingredient in a PE, and we can see the need of parser and code-generator generators, specialized analysis and manipulation programs, special tailored pretty printers etc. Preferably such tools should be easy to generate by the users.

The production of software will in the future be more oriented towards use of existing programs and change them to suit a new, but similar, application or to be adapted to a new system environment (e.g., computer system, programming language, operating system). We believe that it is important for a PE to be able to support this type of modifications with help from advanced program manipulation techniques.

The execution process of a program in a PE is an important issue and differs from the normal edit-compile-execute cycle. A program will preferably be developed interpretatively during the development and test phases and may be compiled for more efficient use. We can see different degrees of compilation. This will put new requirements on the programming language in use.

Many of the functions in the desired tools needs more formal treatment, and tools and methods in a PE should be based on formal results when appropriate implementations can be performed.

There exists a need for better program development tools for micro computers and distributed systems. We expect that very little program development in the future will be performed in assembly languages and thus we feel that the effort should be spent on general program development support. Programs will be developed in a development system and after that

transported to the micro computer system. Thus investigation of different transportation strategies will be necessary.

## 1.3 PREVIOUS WORK IN LINKÖPING.

The Linköping informatics lab has a strong tradition in both usage and development of methods for incremental languages and systems.The Interlisp system has been a tool widely used within the group and in addition to this it has been demonstrated how to export that technique to other languages. The group has also maintained a high level of competence in the field of "conventional" techniques. This will make it possible for the "Lisp-school" and the "Pascal-school" to exchange ideas.

Since the Linköping informatics lab was founded much of the research has been centered around PE's. A number of successful projects have been completed or are still in progress. Much interest has been centered around the design of interactive programming environments. The Interlisp system has been a common tool and has served as an inspiration and/or challenge to transfer ideas to other languages. Most work has attempted to show that languages are not central in the research but the interactive tools. Within the group PE's have been designed for such different languages as Pascal (Wilander), BCPL (Strömberg, Fritzson) and Prolog (Komorowski). The design issues and implementation problems are in many ways similar.

The programming methodology in an incremental system is very different compared to that of a "batch" oriented one. These new methodologies have been developed through experience with incremental systems. In these systems design methodologies like "structured growth" and "handles" (Sandewall) are now in practical use. Another effect of the incremental system design and the Lisp language has been that the view of a program has been shifted from a text file to a "data base". In an incremental system the program is a data structure of procedures and data. With such data structure (or database) it is possible to manipulate programs, make presentations of the program structure in different projections and to keep documentations associated with the program code.

Other important research within the group has been connected with program manipulation. From this work several PhD thesis projects have emanated (Haraldsson, 77) and (Emanuelson). One specific problem has been to perform optimizations on the source code level of programs. This research has shown among other things that it is not always necessary to develop a compiler instead of an interpreter merely for efficiency. Instead it is possible to use the interpreter as a compiler macro and thus get an fully compiled program without writing the special compiler. The problem of transporting a program from a development environment to a more production oriented system has also been studied (Strömberg, Fritzson). This requires

development of techniques for translating language constructs from an flexible extendible language into a fixed and efficient language. A complementary approach to this problem has been taken in the MEDICUS-project (Wikstrand, Hägglund), where program analysis techniques are used to produce a reduced system as a basis for program translation or semi-automatic program generation.

Implementation strategies have attracted substantial interest. Work on Lisp-implementations on low level has led to a dissertation (Urmi). Related work is in progress concerning the issue of generating compilers through description of the language and the machine. Work is performed in the field of automatic generation of code generators (Lunell) and parsers (Ericson). The latter part is in its finishing phases. The group has gained a great deal of knowledge in implementation techniques when implementing a Pascal compiler (Lunell, Börtemark), an Algol-68 compiler (Fäldt) and some Lisp systems (Urmi, Goodwin) in the past few years. This competence will be important to maintain.

Work on knowledge representation has been performed, both in Qlisp (Emanuelson) and with semantic networks (Goodwin).

## 1.4 PROJECT ORGANIZATION.

The organization of PELAB will be centered around a kernel project with a number of satellite activities, such as literature studies, theory development, minor implementations of experimental systems, thesis projects etc. The kernel project, aiming at a design for an integrated programming environment, will be more implementation and experimental oriented and closer to applications than the others. This project will have the dual aim to make it possible to transfer the expertise of the group out into industry and to give new PhD candidates a basic knowledge of the ideas and techniques used within the laboratory. The primary research objectives of this group will be implementation of components of PE's and experimental implementations and usage of integrated PE's. Program manipulation issues will be further studied and tools for that will be integrated in PE. This project is a collection of strongly related subprojects.

Implementations and experiments will mostly be carried out in the Interlisp-system and we will use Lisp as the implementation language. Pathcal, the implementation of Pascal in Lisp, may serve as a base for further work. The work will not be concentrated to a specific language, and different languages and language constructs will be considered.

This project is expected to show how a program implementer and maintainer could work and the type of tools he/she could have. This project will thus not attempt to discuss in detail the problems of specification and requirements

analysis. We will only require a connection to such systems and will in this experiment say that we have a simple specification tool that could be adapted to more formal techniques when they are explored.

The activities in this kernel project will, among other things, involve questions regarding:
- Design of program databases.
- Testing and debugging methodologies.
- Program manipulation.
- Documentation and specification.
- Design of programmers interface and program presentation.

The aim of the research will not be to develop *The* integrated programming environment, but to develop designs and components of such systems. The most important strategy for the research will be experimental system design. Total integration might occur in some cases but not necessarily in all.

As satellites to this kernel project there might be other project groups, studying a specific problem. Today we have plans for three such groups.

*Program representation*
This project will take input from the AI-school of knowledge representation. These techniques will be adapted to programming and programming methodology. Associative networks, e.g. semantic inheritance networks, and data abstraction techniques are some candidates to be considered.

*Formal methods for programming*
This group will mainly take input from the more conventional compiler construction school. The main thrust will be in the theory for program analysis using for example the monotonic framework analysis algorithms. Other areas of interest are methods for incremental analysis, interprocedural analysis and analysis of agenda-based programming languages (cf. Simula).

*Compiler writing tools*
The compiler tools part will be mainly oriented towards the development of tools for automatic code-generation. This group will create efficient means of implementing a language on a new machine.

It is expected that these projects and the kernel project will have much in common. The main flow of information will probably come from the satellite projects to the kernel project. Although the projects differ in technique one could view the projects as attacking partial problems for a total integrated programming environment.

# 2. APPLIED SOFTWARE SYSTEMS LABORATORY

## 2.1 RESEARCH OBJECTIVES.

The research activities in this laboratory deal primarily with application or end user oriented aspects of software systems. This is to be interpreted as the study of computer support for specific classes of applications as well as the design of flexible end user interfaces. Research problems concerned involve methods for specification, implementation and maintenance of such systems.

One practical goal for research in the Laboratory is to make computers easier to use for people lacking special training and interest in programming or other forms of computer mastery. Computer systems today are often hard to access: they are not flexible enough in day-to-day operation, and changing their behaviour requires reprogramming by a specialist, who is a middleman in short supply. This situation is a considerable disadvantage from the point of view of public interest: as a general rule, information in computer systems should be made as widely available as possible (except of course for cases where privacy is at stake). It is also often a disadvantage for the use and usefulness of the computer system. Finally, it is probably a contributing factor for the common perception, correct or incorrect, that computer professionals have a considerable 'power' by virtue of their expertise.

We identify three technical areas which should be approached in order to remedy the situation:

1. Modelling of applications, dealing with how a computer system may be oriented towards a specific application area, with respect to how the applications are described to the computer, and how the processing within the computer is organized. Good techniques for modelling of applications are necessary in order to develop computer software which is well adapted to the application.

2. Dialogue techniques, dealing with various ways of carrying out the dialogue between the user and the computer, for example interactive graphics, written natural-language I/O, and so forth.

3. Development of very-easy-to-use systems for specific purposes. An example of a system which is intended to fall in this class is the Viewdata system.

We believe that the third of these approaches is a question of product development rather than build-up of technological knowledge, and therefore restrict the continued proposal to the first two areas. The following more specific issues are involved there.

### 2.1.1 Modelling of application areas.

The classical way to develop an application for a computer, is to write a program that does the job. Textbooks of computer programming and on software engineering still often suggest, explicitly or implicitly, that that is the way to do things.

In practice, one usually prefers to have specialized tools that service a class of applications. This is a natural approach since many software companies select a limited number of specialized application areas, and deliver similar systems within these areas, to many customers whose needs are similar but not identical. The specialized tools are not restricted to subroutine libraries, but may also be:

- specialized programming languages
- program generators, which accept a description of an application and generate a corresponding, tailor-made program
- general-purpose programs ('superroutines') which are highly parameterized, and which enable the user to develop an application by selecting appropriate parameter structures.

These methods may be used both for classes of applications (e.g. systems for financial planning), or for specific functions that recur in a wider class of applications (e.g. report-program generators, and support for form-like layouts on screens). They have in common that they introduce a specialized language for describing a class of applications, or a specific function, and various kinds of tools for manipulating that description.

More advanced examples of the same techniques may be found in the research literature for some application areas, for example in systems for describing office procedures , in compiler-compilers, or in systems for describing process control systems. Similarly, the trend towards putting more and more information in the 'declarative part' or the 'representation of knowledge', in artificial intelligence systems, seems to be a parallel development.

In many cases, it will be natural to separate the computer-based knowledge about the application situation, from the software that is actually used in that situation. If such a separation is done, we obtain as a consequence a method for application development where a model of the application is gradually built up in a data base, tested as a pilot system in that environment, and then used to generate the production programs (and parameters) for the

application, to be used by the end user. This approach should encourage one to build a tool-box of programs for operating on such models, for example as documentation support.

Systems of this kind may sometimes be used by a computer specialist, for reasons such as:

- to increase the productivity of the specialist;
- to reduce the time required for completing the application;
- to improve the quality of the software and/or its documentation;

but it may also be designed so that it enables a specialist in some other profession to develop his own 'programs' without the nuisance of conventional programming, and without having to work indirectly, through a programmer.

In other application situations, and particularly in a longer perspective, one may instead wish to keep the knowledge about the application present at all times. 'Expert' problem-solving systems in artificial intelligence research are an early example of that possibility, although we believe that it is more fruitful to try to embed A.I. techniques in current computer applications, rather than inventing new applications. Building systems which contain and use an explicit description of their application environment, is an approach to ultimately obtain systems which

- allow task solving in a truly interactive, cooperative fashion;
- are able to explain to the user, the reasons for their behavior or response in specific situations;
- allow the user to request (in the language of the application) exceptions from the standard procedure usually performed by the system, and have it performed correctly and consistently
- are able to handle 'new' situations adequately, and request the user's help only when really needed.

### 2.1.2 Dialogue techniques

It is obvious that the computer systems of tomorrow will, to a very large extent, be interactive to their nature. Experience shows that developing interactive parts of a program system is usually a very time consuming task, especially with regard to subsequent program maintenance. It is to be noted that *interactive* does not only imply entering of parameters in a dialogue mode, but also the organization of programs to support cooperative, incremental "problem solving".

Development of dialogue management techniques appropriate for various kinds of applications and using different types of equipment is one way to improve the end user interface. A number of techniques to this end exist,

such as e.g. command languages, tree-structured menu selection, screen forms, restricted natural language etc., but many aspects of these techniques still remain to be explored. Another way to deal with the problem of dialogue development is to look at the architecture of the software:

- Design of an end user system ("end user facility"), i.e. a system which serves as a user's workbench, and allows him to perform local operations on his work data, and serves as an interface in his dealings with an assortment of other, often remote D P systems. The user interface may then help to provide uniformity in notation and concepts, to facilitate combined use of several systems, and to support locally operations not provided by the remote systems.

- Development tools for dialogue techniques. The section on modelling of application areas, above, discussed the use of special-purpose languages and other similar tools for developing applications. Similar techniques are being used for supporting the development of programs for specific functions within a system, and to a large extent for supporting dialogues. Report-program generators and screen-control program generators are examples which are already on the market.

In particular, we notice here the interdependence of the 'application modelling' and 'dialogue techniques': on one hand, application modelling needs the support of dialogue techniques for entering models into the computer, and on the other hand, dialogues need the support of the same kind of special-purpose languages as are used for expressing application models.

## 2.2 TECHNICAL APPROACH.

We proceed now to the alternative, technology-oriented characterization of the Laboratory's activities. From this perspective, research in the Laboratory is concerned with methods and tools for the development and modification ("maintenance") of software, with an emphasis on application software rather than e.g. compilers. The basic technical concept may be stated as: *software production through design environments and production environments for restricted classes of applications*

Software is usually developed in a classical framework which is organized around the compiler, and which also includes text editors, program generators, and various documentation tools 'before' the compiler, and a linkage editor (for connecting separately compiled modules) and a host operating system (within which programs are executed) 'after' the compiler. We are presently questioning this classical framework, and our current hypothesis for an alternative framework will guide the research in the laboratory, at least initially.

We propose instead to think of two major types of tools, development systems and production systems. Development systems are CAD systems for software, i.e. they are interactive systems which provide an integrated set of services for the programmer (in a broad sense of the word, i.e. the person who impresses a desired pattern of behavior on a computer). In particular, the development system provides services for editing of programs (again in a broad sense of the word), for test execution and debugging (both in a conventional sense, i.e. to cure reasons for functional breakdown, and in the pilot-system sense, i.e. to allow end users to check out the system's behavior and specify modifications), for documentation support, and for generation of 'object code'. In particular, the development system contains a data base in which the parts of the program, and the corresponding descriptive information (declarations, documentation) is stored.

The production system is usually an end-user system. It implements a fixed framework of communication with the user. In addition it typically contains a number of general-purpose services such as text editing and 'help' support and it contains a number of exchangeable modules for performing tasks which are specific to the particular user or installation. Each production system will receive such modules as contributions from one or more development systems. The development systems contain in their data bases, the information that enable them to furnish the right production systems with the right modules at the right times.

The 'programs' that are maintained in the development systems, need not be expressed in general-purpose programming languages. Instead, the development environment should support a variety of different special-purpose languages which are oriented towards either a specific technical function (such as for describing forms for use on screens or printouts, or for describing dialogues), or a set of concepts that are useful for a restricted set of applications (such as 'budgeting' or 'information-flow applications'). In particular, this means that the use of the development environment is not restricted to professional programmers or other computer experts; it may also be a tool for the application expert.

The strategy encourages software design for classes of applications, rather than for single applications. Each class of applications would be supported by one or a few, appropriate development systems, including e.g. special purpose languages for describing instances of the class to the development system, as well as a basic production system which can be used throughout the class. Each instance within the class can then be created by specifying it to the development system(s), debugging it there, and then having the appropriate modules sent to copies of the basic production system.

This mode of working enables a software manufacturer to arrange its production similarly to manufacturers of other complex technical systems, i.e. he specializes in a limited number of 'products' (which is what we have called

'classes of applications'), and sets up his operation so that he can easily provide each customer with a custom-made variant of the product. As a consequence, modularity is emphasized for the purposes of manufacturing and maintenance, and not only for facilitating the design work.

The following are the significant areas of professional interest that we have identified so far:

*Architecture of the development environment and the production environment*
This includes issues of how to represent programs and other knowledge in the data base of the development environment, generally useful support for the repertoire of special-purpose languages; issues of modularity and other global structures in in the data base and the programs; and the structure of the supporting software for the two kinds of environments

*Dialogue techniques*
Good methods for dialogues are essential in both development systems and production systems. Dialogue techniques include special-purpose languages for describing dialogues; techniques for mixed-initiative dialogue and natural-language dialogue; etc.

*Facilities in the production environment*
This includes 'fancy terminal' techniques such as graphics, multiple windowing and multiple screen techniques, non-standard keying devices, etc.

*Operations on application models*
By virtue of the special-purpose languages and the dialogue techniques, models of the application are built in the development environment. Some significant operations on these models are for simulation and other performance evaluation, for generating data base schemata for use in a data base for the production system, and for non-trivial generation of program modules for the production environment.

These issues have a lot of commonality with the research issues which were identified, from a different perspective, in the previous section.

## 2.3 REFERENCE APPLICATIONS.

Research in the area of application modelling must clearly achieve a balance between working with specific reference applications, and working with the general problems that occur when this approach is used. We foresee a mode of working where some projects concentrate on the needs of their respective

applications, while paying marginal attention to new general-purpose techniques, and other projects concentrate on the general-purpose problems while using one or a few applications as test examples and toy problems. One main area of interest in this respect has so far been computers in office information systems, hospitals and university education, and combinations of these.

The following four separate efforts are presently in progress in our group:

1. The MIL project (Medical Information system in Linköping). The topic of this project is customized medical information systems, and the purpose is to develop an explorative system which can be tried in a few applications at university hospitals. Important techniques developed and tested in this environment concern e.g. methods for automatic derivation of database structures from end user forms, information flow modelling facilities, pilot system implementation techniques and program transfer to the production environment.

2. The LOIS project (Linköping Office Information System). The purpose of this project has been to develop a prototype system for text processing, communication between users, and handling of structured data, in a paperless-office type environment. The emphasis of this project has been on integrating various kind of services while still maintaining a high degree of programmability and adaptability to end user demands.

3. The MEDICUS project (Medical Education with Interactive Computer Usage for Simulation). In this application, the task is to create a number of programs for simulation of patient management situations, to be used in medical education. Simulation programs are to be created directly by physicians using an on-line terminal. This project demonstrates the feasibility of allowing a professional user, who is not trained to use computers, to develop programs within a certain class of programs, using only the terms and concepts of his application, and providing much less detail than if he were to write a conventional program.

4. The TPV project (terminal software and end user facilities). This project studies methods for development of software systems for programmable terminals and especially the distribution of processes and data between the terminal and the host computer. A related topic is the design of end user facilities in a system of the "electronic desk"-type. Specific topics addressed are:
- Design of a generalized editor, which can handle structured texts as well as graphic information, tables etc.
- Implementation of window handling on an intelligent terminal, with associated functions for text editing, mail services, notes scribbling etc.
- Design of a language for manipulation of an electronic desk top.

Work on tools in these reference applications has raised a number of significant architectural issues, having to do with program structuring, modularity, the roles of compilers and operating systems for program structure, and so forth. We expect to continue the analysis of these issues, based on the continued experience from using the tools.

Further, an increasing number of specific methodological problems are being identified in the work on the tools and the reference applications. It will be worthwhile at this time to increase the attention on such problems, particularly since we have already built up a reasonable repertoire of tools which make interesting reference applications possible. Work is in progress in the following areas:

- derivation of data base structure from specification of user information needs expressed as forms (in progress, thesis almost completed).
- development of generalized conceptual frameworks for dialogue languages (cf. programming languages, which are basically monologue languages), e.g. for text editing, database query, job control etc.
- methods for transfer or semi-automatic generation of application programs from a specialized development environment to a target system.
- mapping of operations on the surface presentation of data base contents, e.g. as structured text, onto corresponding operations of the data base itself.

Work on these and similar problems are undertaken by individual members in the group, as thesis project or as a post-graduate project.

## *3. ARTIFICIAL INTELLIGENCE PROJECT*

The artificial intelligence project was initiated in May 1980 with the objective to organize research in the area of artificial intelligence within the Software Systems Research Center. Accordingly, we spent most efforts during the last year in finding out how existing AI interests, former AI work performed by members of the SSRC and new ideas about artificial intelligence could be unified into a common framework which would fit into and contribute to the overall objectives of the SSRC.

Our activities resulted in a proposal to establish an artificial intelligence laboratory besides the already existing laboratories for programming environments and application software within the SSRC. In the following we will shortly describe the goals and objectives for such an AI laboratory. A more detailed account has been given in a separately published research proposal.

Artificial intelligence is a steadily growing discipline which makes important contributions to a great number of different areas of research, because:

- AI explores new and important areas for computer applications, such as expert systems, computer vision systems and robots.
- AI explores new and more powerful techniques for the interaction with computer programs, such as natural language front ends.
- AI provides a new framework and new formalisms for procedural theories of human cognition which are and will be of great interest for psychologists, linguists and philosophers.
- AI explores new techniques and tools which can stimulate computer science, such as very high-level languages, knowledge representation systems, production systems.

The main objective for the AI laboratory must be to build up expertise and to conduct qualified research in the field of artificial intelligence. In order to accomplish this task several activities have to be performed within the laboratory, including careful analyses of existing programs and theories as well as the implementation of new programs which can illustrate theoretical concepts and by which hypotheses can be tested. Furthermore, the laboratory should provide a good educational environment for graduate and

post-graduate research students who want to specialize in the field.

Since AI has become a rather large discipline by now, we deemed it essential to find meaningful limitations for the AI laboratory both in terms of topics to be investigated and domains to be explored for the exemplification of theoretical concepts. These limitations have been chosen with respect to practical and theoretical reasons. According to our own competence and specialization we thus propose that research in the AI laboratory will be concerned with the following major issues of artificial intelligence:

- communication in natural language
- knowledge representation
- learning
- common-sense reasoning

We thus exclude areas such as computer vision, robotics and automatic deduction (within logical formalisms), because serious research in each of these areas would be too resource demanding.

With respect to the domain we have chosen to concentrate on office worlds, since there exist the strongest interactions with other research activities in the SSRC. This is one application area where we expect to have a mutually fruitful exchange of ideas and experiences with the other laboratories, e.g. on problems regarding dialogue technique and knowledge representation. The office domain includes such tasks as document preparation, travel planning, budget expertise, room allocation, scheduling of meetings etc. This domain can furthermore be shown to possess all of the features required for theoretical reasons.

# *LIST OF PUBLICATIONS.*

## *PhD theses:*

(Linköping Studies in Science and Technology Dissertations.)

| | |
|---|---|
| No 14 | **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977. |
| No 18 | **Mats Cedvall:** Semantisk Analys av processbeskrivningar i naturligt språk, 1977 |
| No 22 | **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978 |
| No 33 | **Tore Risch:** Compilation of Multiple File Queries in a Meta-database System, 1978. |
| No 51 | **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980. |
| No 54 | **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980. |
| No 55 | **Pär Emanuelson:** Performance Enhancement in a Well-structured Pattern Matcher through Partial Evaluation, 1980. |

## *Research reports published 1976-1980.*

| | |
|---|---|
| LiTH-MAT-R-76-8 | **Erik Sandewall:** Some observations on conceptual programming. Also in Elcock, E.W., Michie, D. (eds), *Machine Intelligence 8*, Edinburgh University Press, 1977. |
| LiTH-MAT-R-76-9 | **Erik Sandewall:** Programming in an interactive environment: The LISP experience. Also in *ACM Comp. Surveys, vol. 10,* no 1, pp 35 -71, march 1978. |

LiTH-MAT-R-76-11 **Anders Beckman:** Programvarukvalitet, En nordisk förstudie.

LiTH-MAT-R-76-13 **Sture Hägglund, Östen Oskarsson:** En teknik för utformning av användardialoger i interaktiva datasystem.

LiTH-MAT-R-76-14 **Erik Sandewall:** Personal data bases and the design of man computer dialogues.

LiTH-MAT-R-76-17 **Jaak Urmi:** String to string correction.

LiTH-MAT-R-76-18 **Jaak Urmi:** A shallow binding scheme for fast environment changing in a "spaghetti stack" LISP system.

LiTH-MAT-R-77-26 **Anders Haraldsson:** A partial evaluator, and its use for compiling iterative statements in Lisp. Also in *Proc. of the Fifth Annual ACM Symposium on Principles of Programming Languages*, Tucson, 1978.

LiTH-MAT-R-78-1 **Anders Beckman, Sture Hägglund, Gunilla Lönnemark:** A program package supporting run time variation of text output from interactive programs.

LiTH-MAT-R-78-8 **Kenth Ericson:** Pascal i Sverige.

LiTH-MAT-R-78-19 **Erik Tengvald, Sture Hägglund:** En metadatabas för Cobolsystem.

LiTH-MAT-R-78-20 **Jerker Wilander:** Interaktiv programutveckling i Pascal - Programmeringssystemet Pathcal.

LiTH-MAT-R-78-21 **Erik Sandewall:** Programmeringsteknik för flexibilitet.

LiTH-MAT-R-78-22 **Erik Sandewall:** LOIS - An Overview of Facilities and Design. Also in *DATA, vol 9*, nr 1-2, February 1979. Revised version as "Provisions for Flexibility in the Linköping Office Information System", *Proc. of the National Comp. Conf.*, Los Angeles, 1980.

LiTH-MAT-R-78-23 **Claes Strömberg, Henrik Sörensen:** Beskrivning av fontsystemet och erfarenheter vid systemutvecklingen.

LiTH-MAT-R-78-24 **Erland Jungert:** Generering av databasstrukturer från en formulärbaserad datamodell.

LiTH-MAT-R-78-25 **Pär Emanuelson:** A case study of Qlisp: Representing knowledge taken from medical diaries.

LiTH-MAT-R-79-3 **Arne Börtemark, Hans Lunell:** Implementering av Pascal på minimaldatorn: en tillbakablick.

LiTH-MAT-R-79-5 **Jim Goodwin:** Taxonomic Programming with Klone.

LiTH-MAT-R-79-7 **Pär Emanuelson:** A comparative study of some pattern matchers.

LiTH-MAT-R-79-10 **Arne Börtemark:** Felbekämpningsmedel, en första översikt.

LiTH-MAT-R-79-18 **Hans Lunell:** Automatic generation of code for conceptual machines: A problem discussion.

LiTH-MAT-R-79-19 **Jan Komorowski:** QLOG interactive environment - the experience from embedding a generalized Prolog in INTERLISP.

LiTH-MAT-R-79-21 **Erik Sandewall:** Biological Software. Also in *Proc. of the 6th Int. Joint Conf. of Artificial Intelligence*, Tokyo, 1979.

LiTH-MAT-R-79-22 **Erik Sandewall:** Self-description and reproduction in distributed programming systems.

LiTH-MAT-R-79-23 **Erik Sandewall:** A description language and pilot-system executive for information-transport systems. Also in *Proc. of the 5th Int. Conf on Very Large Data Bases*, Rio de Janeiro, 1979.

LiTH-MAT-R-79-24 **Erik Sandewall, Erland Jungert, Gunilla Lönnemark, Katarina Sunnerud, Ove Wigertz:** A tool for design and development of medical data processing systems. Also in *Proc. of the 2nd Congress on Medical Informatics, Europe*, West Berlin, 1979.

LiTH-MAT-R-79-28 **Erik Sandewall:** Why superroutines are different from subroutines.

LiTH-MAT-R-79-37 **Jerker Wilander:** An interactive programming system for Pascal. Also in *BIT, vol 20*, 2, 1980.

LiTH-MAT-R-79-39 **Sture Hägglund:** An Application of Lisp as an Implementation Language for the Domain Expert's Programming Environment.

LiTH-MAT-R-79-42 **Anders Ström:** Experiment med partialevaluering.

LiTH-MAT-R-80-01 **Johan Elfström, Jan Gillqvist, Hans Holmgren, Sture Hägglund, Olle Rosin, Ove Wigertz:** A Customized Programming Environment for Patient Management Simulations. Also in *Proc. of the 3rd World Conf. on Medical Informatics*, Tokyo, 1980.

LiTH-MAT-R-80-08 **Dan Strömberg, Peter Fritzon:** Transfer of Programs from LISP to BCPL Environments: An Experiment.

LiTH-MAT-R-80-18 **H. Jan Komorowski:** QLOG - The Software for Prolog and the Logic Programming.

LiTH-MAT-R-80-20 **Pär Emanuelson, Anders Haraldsson:** On Compiling Embedded Languages in Lisp. Also in *Proc. of the 1980 LISP Conf.*, Stanford, Calif, 1980.

LiTH-MAT-R-80-22 **Erik Sandewall, Henrik Sörensen, Claes Strömberg:** A System of Communicating Residential Environments. Also in *Proc. of the 1980 LISP Conf.*, Stanford, Calif, 1980

LiTH-MAT-R-80-23 **Uwe Hein:** Interruptions in Dialogue. Also in D. Metzing (ed), *Dialogmuster und Dialogprozesse.* Hamburg, Buske, 1980 (to appear).

LiTH-MAT-R-80-24 **Anders Haraldsson:** Experiences from a Progam Manipulation System.

LiTH-MAT-R-80-27 **Erik Tengvald:** En Intuitiv Förklaring till Kildalls Algoritm

LiTH-MAT-R-80-28 **Erik Tengvald:** A Note Comparing Two Formalizations of Dataflow Algorithms.

LiTH-MAT-R-80-29 **James W. Goodwin:** Why Programming Environments Need Dynamic Data Abstractions. (Presented at the Schlumberger Workshop on Programming Environments, Ridgefield, Connecticut, 1980. Partial proceedings to appear).

LiTH-MAT-R-80-30 **Lars Wikstrand, Sture Hägglund:** A System for Program Analysis and its Application as a Tool for Software Development and Program Transfer.

LiTH-MAT-R-80-35 **Erland Jungert:** Deriving a Database Schema from an Application Model Based on User-defined Forms.

LiTH-MAT-R-80-36 **James W. Goodwin and Uwe Hein:** Artificial Intelligence and the Study of Language.

LiTH-MAT-R-80-37 **Sture Hägglund:** Towards Control Abstractions for Interactive Software. A Case Study.

LiTH-MAT-R-80-38 **Sture Hägglund, Johan Elfström, Hans Holmgren, Olle Rosin, Ove Wigertz:** Specifying Control and Data in the Design of Educational Software.

LiTH-MAT-R-80-39 **Kenth Ericson, Hans Lunell:** Redskap för kompilatorframställning

LiTH-MAT-R-80-41 **Hans Lunell:** Some notes on the terminology for Compiler-Writing Tools

LiTH-MAT-R-80-42 **Östen Oskarsson, Henrik Sörensen:** Integrating Documentation and Program Code