AN EXPERIMENT IN COMPUTER AIDED INFORMATION SYSTEMS DEVELOPMENT

PER AANSTAD

TROND JOHANSEN

GEIR SKYLSTAD

COMPUTING CENTER OF THE UNIVERSITY

OF TRONDHEIM (RUNIT)

NORWAY

ARNE SØLVBERG

DIVISION OF COMPUTING SCIENCE

THE NORWEGIAN INSTITUTE OF TECHNOLOGY

THE UNIVERSITY OF TRONDHEIM

NORWAY

ABSTRACT

During the last decade there has been a growing interest in the use of computers to assist systems analysts and designers in the development of systems specifications and in the implementation of program systems consistent with these specifications. [1, 2].

An experimental software system, CASCADE/2, was developed at the Norwegian Institute of Technology in 1972-74 for this purpose. CASCADE/2 has been used in several software development projects. Experience with three cases is described. Further research and development are discussed both from a short term and long point of view.

1. THE SOFTWARE PROJECT PROBLEM

Development and operation of a software system proceed through phases, commonly called the system life cycle. Starting with a problem definition and analysis of user needs, crude requirement specifications are massaged into an information system design specification. Man/machine interfaces are agreed upon, equipment is ordered, software is produced, testing schemes are developed, manuals are written and users are trained. The resulting software product is put into operation, and the long, tedious process of changes, adjustment, tuning and user-correction proceeds.

The general charateristics of the software engineering project do not differ from those of engineering design projects in genereal. The project staff must be managed and the product specification reviwed and approved. The product must be tested for correctness and quality, costs must be controlled and the product must benefit the buyer.

Beneath the surface, however, there are distinct differences between software engineering and "engineering engineering". The skill and the knowledge of the software development staff are manifested, not in the shape of a physical structure like a bridge, an aeroplane or a telephone switchboard, but in a rather abstract set of rules governing the transfer and creation of data in a computer. This collection of rules, i.e. the computer program and its documentation, is the product.

The behaviour of physical structures is governed by the laws of nature. The behaviour of software products is governed by the laws of computer manufacturers (as manifested in operating systems, compilers, database systems, etc.) and by the behaviour of the persons who interact with the software.

The properties of physical structures can be specified in formal models which are consistens with the laws of nature. Formal calculations may be made to see if a proposed physical structure is feasible and to see if the proposed designs behave properly. If this was not so, our industrial societies would be based on craftmanship alone, not on technical science and craftmanship.

There is, so far, no generally accepted model which capture the relevant features of a software product, both with regard to the behaviour of computer resources, and with regard to the behaviour of the user community in which the software product is used.

There is an increasing volume of research concerned with the problem of information systems modelling. The modelling problem is attacked from different angles, such as data semantics, systems analysis, theorem proving, operating systems modelling, computer performance modelling, human engineering, etc.

In this paper, we shall describe one specific experiment in computer assisted information system development. The project has been carried out at the Norwegian Institute of Technology, Trondheim, Norway, in a joint venture between the univerity computing centre and the computing science division. The research effort started in 1969-1970.

We shall describe the system models which have been used and a systems documentation package (CASCADE/2) which has been developed to support these models [3,4,5]. We shall give an account of some real-life software development projects where CASCADE/2 has been used. Experiences with these experiments will be discussed.

2. MODELS FOR INFORMATION SYSTEMS SPECIFICATION

A complete specification model should provide concepts which enable the software engineer to capture and formally state the relevant features of any information system. The model should be so rich that all life cycle phases are covered, starting with the requirement definition and proceeding through design to implementation, operation and maintenance.

The information systems specifications are used for different purposes in different phases of the life-cycle. During requirement definition the most important use of the specifications is to enhance communication between the systems development staff and the user community so that the project group can get the requirements right before proceeding with implementation. The requirements specification is the information systems design at a crude level. The requirement specification therefore contains the basic design of the software system to be produced. During the subsequent phases of the lifecycle this basic software system specification is enriched by more and more detail. There is a shift en emphasis in the use of the systems specifications, from communicating with the user on the users terms, to communicating with the computer on the computers terms. This shift in emphasis is reflected by a change in the need for modelling concepts. The terminology and structure of programming languages, operating systems, communication networks etc., must be reflected in the systems model, if such a model is to enable the software engineer to specify the relevant properties of his software product.

One basic property of a high quality product is that is consistent with the specifications of that product. A complete information system specification model should provide opportunity for testing the consistency between the detailed software specification and the requirement specification. The modelling concepts of the software specification must be consistent with the modelling concepts of the requirement specification.

We have so far not managed to solve this modelling problem completely. Our models should therefore be regarded as a step in this direction, rather than as a porposal for a final solution.

Our basic modelling concepts are object classes, binary relations between object classes and attributes of object classes. A model is characterized by its object classes, relations and attributes. Three different kinds of models have been used: One software model, one information system model and one organisation model.

An information system is a part of a larger system called the total system. The information system model contains the specification of requirements to the software system. The organization model represents other relevant parts of the total system which might interact with the software system, i.e. the information system environ-

ment. The software system model represents an implementation which satisfies the requirement specification.

The organization model and the information system model are developed in collaboration with user representatives and must therefore reflect user terminology. The software model is developed by computing professionals and must reflect computer terminology.

2.1 The Information System Model

The basic object classes of our information system model are

INF - objects representing information which is produced and used in the total system, e.g. transactions, documents, archives.

IPS - objects representing information processing such as production, use, transmission, retrieval of information.

SIG - objects representing the flow of control in the information system, the sequencing of IPS-objects.

The basic model relations are

I - the input relation, $I \subset INF \times IPS$, relating information objects to those IPS-objects which use the information objects.

O - the output relation, $O \subset IPS \times INF$, relating IPS-objects to those INF-objects which are produced by the IPS-object.

C - the component relation,
$C \subset (IPS \times IPS) \cup (INF \times INF)$,
relating an object to its components. The C-relation is used to represent the hierarchical decomposition of processes and information respectively.

N - the entry relation, $N \subset SIG \times IPS$, relating the initiating control signal to the IPS-object to be activated.

X - the exit relation, $X \subset IPS \times SIG$, relating IPS-objects to those control signals which are produced when the IPS-object leave their active state.

The basic information system model is an 8-tuple

(INF, IPS, SIG, I, O, C, N, X)

The 4-tupe (IPS, INF, I, O) represents the flow of information between processes.

The 4-tuple (IPS, SIG, N, X) represents the flow of control between processes.

The tuple (IPS, C) represents the hierarchical decomposition of processes.

The tuple (INF, C) represents the hierarchical decomposition of information objects.

The leaf-nodes of an information tree are called TERMs (abbreviation: TR) analogously to the data item concept in database system terminology.

An auxiliary model concept is the information-type concept (abbreviation: INFTY). Information objects of the same INFTY appear several places in the information system description. The type-concepts is used to decrease the amount of writing associated with a systems description by permitting equivalent information structures to share the same structural definition, i.e. be declared to be of the same information type.

2.2 The Organization Model

The information system is part of a larger system, which we call the total system. To enhance the possibility of proper requirement definition the information system should be discussed in the total system context. The organization model is intended to describe those parts of the total system which do not belong to the information system but which are relevant to the requirements definition.

No general organization model, in terms of predefined object classes, relations and attributes, may be prescribed. This is so because the total system characteristics may be very different from case to case. The information system may in one case be a process control system for a chemical reactor, in another case it may be an accounting system in a retail business, or a project planning system in a shipyard.

The organization model is intended to describe the information system environment and must therefore be defined from case to case. The modelling concepts are object classes, binary relations and attributes.

One model that has been used to describe a civil service system <6>, consists of:

Object classes

LEGISLATION - laws and ruled that regulate the behaviour of public bureaucracy.

ORGANIZATION - bureaucratic units which are responsible for performing/supervising certain civil service tasks.

TASK - functions which the civil service have according to laws and regulations imposed by government/parliament.

Model relations

RESPONSIBILITY ⊂ ORGANIZATION X TASK
a certain organizational unit has responsibility for supervising, controlling, performing certain tasks.

LEGAL RIGHT ⊂ LEGISLATION X (ORGANIZATION U TASK)
an organizational unit exists because of some piece of legislation, a task is to be performed because of some piece of legislation.

COMPONENT ⊂ LEGISLATION X LEGISLATION
U ORGANIZATION X ORGANIZATION
U TASK X TASK
the component relation is used to represent the hierarchical decomposition of legislation, organization and tasks.

The organization model and the information model are interrelated by two binary relations:

SOLUTION ⊂ TASK X IPS
information processing systems represent the solution of tasks defined by legislation.

SUPERVISION ⊂ IPS X ORGANIZATION
an information system is supervised and controlled by an organizational unit.

The organization model and the information system model give the formal framework for requirement specifications development.

2.3 The Software Model

The basic object classes are

PROGRAM - objects representing computer programs.

SUBR - objects representing subroutine-type programs.

FILE - objects representing data files.

RCL - objects representing record classes.

EM - objects representing error messages from PROGRAM-objects.

The basic model relations are

REF ⊂ PROGRAM X SUBR U SUBR X SUBR
which is used to represent the reference structure (subroutine call-structure) in the software system.

MEM ⊂ RCL X RCL, the membership relation,
which represent the owner/member relationship in a database network structure.

DBOP = FIND U GET U DELETE U STORE U MODIFY,
the database operation relations,
DBOP ⊂ SUBR X RCL, which represent the kind of database operations the SUBR-objects perform on RCL-objects.

The model contains additional facilities for representing how records consist of data items. The software model facilities also contain an object class called PROC, which represents chunks of declaration statements for subroutines, and one object class INLINE which represents chunks of active statements for subroutines. Object classes PROC and INLINE are related to SUBR-objects by REF-relations.

Several attributes, e.g. program size, number of records, record size, are defined in the software model and are used to represent properties of software objects.

The software model and the information system model are interrelated by the relation.

IMPL ⊂ IPS X PROGRAM U INF X FILE which represents how IPS-objects are implemented by programs and how INF-objects are implemented by files.

3. THE SOFTWARE PACKAGE CASCADE/2

The CASCADE/2 program system has functions for systems description, system presentation and computer program generation. CASCADE/2 is designed for interactive use, but can also be operated in batch mode. It was developed for the UNIVAC 1100 series in Fortran IV. A Honeywell Bull 6000 Series version is now also available.

3.1 The most Important Design Criterion was Flexibility

The CASCADE/2 software package was developed to support research in the area of systems analysis and design. A major design criterion has been flexibility, to prevent rigidity in the software support making impossible experiments with new system model propositions.

A system is represented as a set of interrelated objects. Properties of objects and relations are described by attributes. CASCADE/2 has functions for storing, manipulating and presenting system descriptions based on this "object-relations-attribute" model. New kinds of objects, relations and attributed can be introduced ad lib. Consequently, CASCADE/2 is a very flexible tool for the investigations of systems models.

3.2 Free Format Input Language

CASCADE/2 has a free format input command language which reflects the object-relation-attribute concept.

In figure 3-1 is shown a very simple system where a process P (IPS) has A and B as input related objects (INF) and Q as output related object (INF).

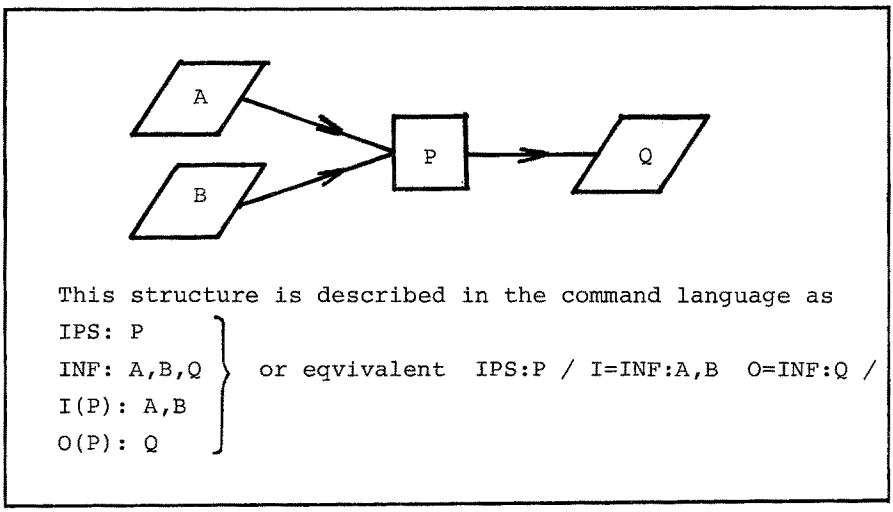

Figure 3-1. A SIMPLE SYSTEM.

In general, the user gives names to the system objects, and indicates to which object class they belong. Two (or more) objects are related by specifying the name of the (binary) relationship, and the name of the objects which participate in the relation.

The standard repertoire of CASCADE/2 contains 49 different object classes, 36 types of relationships and 10 types of attributes.

Relations are directed and the inverse relationships are automatically maintained by CASCADE/2.

3.3 The User May Defind His Own Models

CASCADE/2 is designed such that the user is free to define his own models. This is done by introducing the new object classes, relationships and attributes in the CASCADE/2 database.

Objects are denoted by different naming systems: By unique (global) names or by qualified names. Three different naming mechanisms give the user a wide choice in selecting object names.

3.4 Simple Report Generator

A simple report generator has been developed to present the system description, which is stored in the CASCADE/2 database, in different ways.

The report specification language is closely related to the object-relation-attribute concept. The user specifies the contents of a report by referring to the relevant object classes, relations and attributes. The report formats are lists, cross reference lists, structure lists and tables.

There is also a facility for "navigating" through the stored description. The "route" is specified by a sequence of objects/relations/attributes selections. This navigation facility may be used to answer questions like "Which of the outputs from subsystem A have any influence on system B?".

3.5 Model-dependent Functions

The facilities which are mentioned so far are independent of the actual systems description models which are being used. Some features of CASCADE/2 are model-dependent:

3.5.1 Diagram Generator

An automatic digramming facility is available for system descriptions based on the information system model of chapter 2.1. The user specifies those parts of the description of which he wants a diagram. There is a choice of six different types of diagrams. The diagrams focus attention on different aspects of the structure by ignoring others. Examples of diagrams are systems flowcharts, process flowcharts and activity charts.

3.5.2 Consistency Test

A top-down decomposition means that the same system is described on different levels of detail. If the information system model (chapter 2.1) is used, CASCADE/2 contains facilities for testing the consistency of descriptions on two different levels of detail, as shown in figure 3-2.

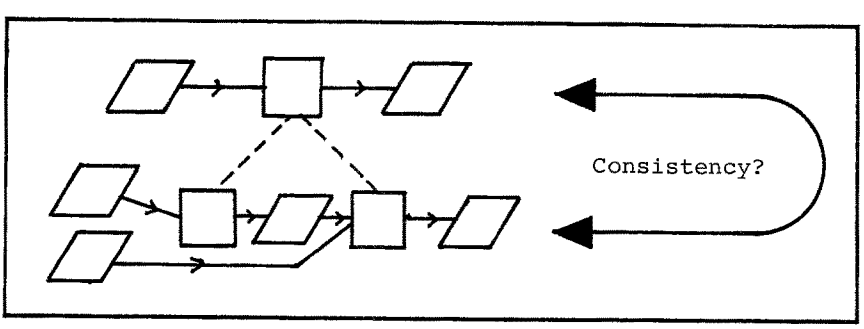

Figure 3-2. CONSISTENCY TEST.

3.5.3 Automatic Software Generation

By using the information system model, a system may be specified to a level of detail where processes are described either by Fortran code or decision tables. From this type of specifications, CASCADE/2 offers the possibility of generating Fortran source code from the detailed requirement specifications. The facility is called APG (= Automatic Program Generator).

4. EXPERIMENTS OF USE IN SOFTWARE PROJECTS

CASCADE/2 has been used in several projects of different character, ranging from projects concerned with organizational analysis and problem definition alone, to projects with the main emphasis on software implementation aspects. We shall in this paper concentrate on application projects at our own organization. This is partly because we know the projects very well, but also because a major emphasis in these projects has been on developing operational software, based on formal requirement specifications. Three software development projects have given especially valuable experience.

4.1 Three Development Projects

The experience presented in this paper are mainly based on the following three software development projects carried out by RUNIT.

BIB-SYS: Automation of university libraries.
This is a multi user, CRT-terminal oriented transaction system. It contains subsystems for document acquisition, cataloguing and borrowing and lending transactions. Ca. 60.000 program statements.

SAPO: An accounting and planning system.
This system was developed for a research organization with project-oriented activities. It contains subsystems for project accounting, invoicing, resource and financial accounting. Ca. 40.000 program statements.

DOLS: A desentralized banking and information distribution system.
This is a pilot project aiming at a decentralization of the dp tasks in the accounting and personnel section of a nation-wide banking and information distribution system. The system contains subsystems for off-line registration of bank and money orders, accounting system, and payroll system for regional offices. Ca. 40.000 program statements.

The three projects have been under tight budgetary and performance control, so the experimental use of CASCADE/2 has been restricted by cost/benefit considerations.

The first project was initiated in 1971, the last project in 1975. The CASCADE/2 software has been substantially developed over this period, so there has been different degrees of sophistication in the projects' use of CASCADE/2.

4.2 The Use of CASCADE/2 for Requirement Specification

The use of CASCADE/2 has been somewhat different in the three projects. In the library project CASCADE/2 was to be used for requirement specification purposes only. No software specification facilitites had been developed when the library project started. The Automatic Program Generator (APG) was developed in parallel with the library project. Because APG was not operational at the start of the proramming phase of that project, it was only used to a limited extent. The libraries, represented by librarians with little or no knowledge of data processing, played an active role in the development of the formal requirement specifications.

In the SAPO project, the use of CASCADE/2 was advanced another step towards automatic software production. The crude design specifications were decomposed to the program statement level. The APG facilities were used for software production as a matter of routine.

In the DOLS-project a new software specification model (chapter 2.3) was used. This model reflects computer programming terminology more accurately than the information systems model which was used in the two other projects. The requirement specification in the DOLS project was perceivable, and a manual documentation within the framework of the information systems model was used instead of computer assisted documentation.

4.3 Software Specification in the DOLS-project

The need for computer assisted documentation increased in the programming phase. Up to that point, the amount of "manual" documentation concerning the requirement specification was managable.

More persons were added to the project staff. The number of programs and subroutines increased rapidly. It became necessary to have up-to-date documentation to answer questions like:

- Is the name of the subroutine which is to be programmed used before?
- If I change the layout of a record in a file, which programs are then affected and who is to be informed of this?
- Which subroutine(s) writes a certain error message?

Development of dialogue systems add some special demands for documentation. Because many seprate transaction programs operate on the database, it was necessary to have detailed documentation on the database command level.

To store, maintain and present this kind of documentation, data about the software must be entered into the CASCADE/2 database: Descriptions of programs, subroutines, record-classes and so on. The project's programming secretary performed this operation.

The reports which were produced by CASCADE/2 supplementd the "manually" produced requirement specification and software specification, which consisted of text and flow diagrams.

Three main groups of report types was produced: overview lists, cross reference reports and deviation reports. A detailed specification of the three groups of reports is as follows:

- Overview lists
  transaction orders, error messages, transaction system tables, source programs, data fields, data field-types, file structure, record class descriptions.
- Cross reference reports between
  subroutines and transaction orders, PROCs and subroutines, record classes and subroutines, data field and record class, data field-type and data fields.
- Deviation reports
  not referenced subroutines, transaction orders not attached to a sub-system, record classes only read (not created/modified), record classes not referenced.

The project library contained an updated set of all CASCADE/2 documentation reports.

5. EXPERIENCES AND GENERAL CONCLUSIONS

Our objective in this research project has been to develop a basis for an integrated software development tool for all phases of the system lifecycle.

We have been reasonably successful in mastering the requirement definition part of the development project. We are not, however, satisfied with our results in the formal interfacing of the requirement specification to the programming and operation phases of the system lifecycle. Even if we can use our computerized documentation package in each separate phase we have not been able to integrate the formal specifications of the different project phases.

Because our research objectives have been ambitious, and because we have very thoroughly tested our software support tools, we feel that our experiences can be of general value for further research in this area.

In all of the three reported projects of chapter 4, formal requirement specifications were developed. The requirement specifications were developed in several steps:

- Systems knowledge was gained through analysis and description of the existing administrative systems in all three cases.
- Decisions were made regarding which parts of the exisiting system to reorganize and automate.
- An overall design of the new organization was made, including specifications of response times, man/machine interaction, implementation cost, computer system architecture.

So the requirement specification included both organizational design and baseline software system design.

In the formal specification, main empasis was put on the use of the information system model of chapter 2. The organizational environment was not formally specified in any of the projects.

5.1 Experiences in Automatic Programming

In two of the projects, basic software system design specifications were detailed to a program statement level using the information system model. Suitable "chunks" of code (on the average 15-30 statements for each "chunk") and decision tables were linked together by the Automatic Program Generator (APG) of CASCADE/2, and program modules were produced. Computer programs were produced from the design specification. During debugging we found, to nobody's surprise, that programmers want to do their "firefighting" on the programs where the bugs are discovered. They resisted going back to the design specification, correcting the bug, producing a program with a slightly unfamiliar pattern, and starting all over again to get accustomed to the pattern of the new program.

One basic problem is to keep track of the changes in the software product and the software specifications at the same time, so that a change in the software product (e.g. because of debugging), is reflected in the software specifications, and vice versa. If the software specifications do not reflect the changes of the software product (i.e. the program system), the value of the specifications are rapidly so seriously degraded that the benefit of developing a detailed specification may be seriously questioned when comparing the benefits with the costs involved.

5.2 Separation of Requirement Specification and Software Specification

Because we did not succeed in solving the automatic software generating problem in our first trial we chose to separate the requirement specification and program specification in our last project (DOLS). The basic reason for this decision was

that we at that time, did not have any facilities to ensure consistency between the program product and the software specification.

The software system model of chapter 2.3 was used for the program specification and the information system model of chapter 2.1 was used for requirement specification. For the reasons mentioned above, we did not try to relate the two specifications such that the software specification could be formally tested against the requirement specification for completeness and consistency. The requirement specification therefore tended to serve as a body of basic systems knowledge, rather than a formalization of system constraints and objectives against which the software could be formally tested.

The use of computer assistance during the requirement definition had consequently to be decided on different criteria than in the SAPO-project. Because the volume of the requirement specification was moderate, and the project staff experienced, the project staff chose to use manual methods for documentation purposes. Computer assistance was, however, used for software specification purposes, where the documentatiom volume is larger.

5.3 Experiences from the Requirement Specification Phase

One of the most pleasant experiences has been the way that the projects have managed to get the users involved. Use of the information systems model of chapter 2.1, combined with functional decomposition to handle systems complexity, has had a profound diciplining consequence on the project staff and their user contacts and collaborators. Of special value was the hierarchical consistency test of CASCADE/2 (chapter 3.5.2).

The automatic flowcharting facilities of CASCADE/2 proved to be of considerable value in enhancing communication between project staff and users. The data dictionary facilitites proved to be useful also from a project management point of view, both concerning integration aspects and control of project vocabulary.

Most of the positive effects can be obtained by using manual documentation methods. The decision to use a computer tool like CASCADE/2 is dependent on two points. The first point is the size and complexity of the information system. CASCADE-like tools tend to be more useful for large and complex systems than for small systems, even if computer assistance is used only for requirement definition as a separate task. The second point is the degree of formal integration between the requirement specification and the software product via the software specification. A high degree of formal integration means that formal requirement specifications can be used directly for checking the consistency and completeness of software specifications and software products. Consequently the benefits of a thorough requirements specifications might easier outweight the cost of establishing the formal specifi-

cation.

5.4 Experiences from the Software Specification Phase

We have earlier in this paper pointed out that we did not succeed in establishing a workable automatic programming environment from a cost/benefit point of view. In the DOLS-project, where the software specification was separated from the requirement specification, CASCADE/2 was used mainly as a simple data dictionary system. Our general experience is that the CASCADE/2 produced documentation is valuable for development and especially for maintenance of the software product. Cross reference reports have been very valuable. These give a satisfactory overview of the consequences of specification changes.

Automatic control of consistency between the documentation and the software system is impossible with the present system. It is left to the user to ensure that all alterations of the program code also involve the corresponding updata of the CASCADE/2 documentation. This is a weak point with our present system. To have the full benefit of the software specification, the consistency between product and specification must be maintained. A part of a solution might be to develop a "Data Division" generator such that any change in the datatype specifications would initiate a genereation of new Data Division and a subsequent recompilation of the program which used those Data Divisions. We are aware that some Data Dictionary systems, which are currently marketed, provide this kind of facility.

5.5 Conclusions

CASCADE/2 is a prototype. It was designed to support research primarily in the systems analysis and design phases of the life cycle. We had reasona success in supporting the development of requirement specifications. Our appetite grew and we tried to use CASCADE/2 for software specification and software production by adding new facilities to our package.

CASCADE/2 is general in the way that it supports a wide variety of systems models. This generality has to be paid for in terms of computational efficiency. If we abandon the idea of integrating the documentation of the different life cycle phases, tailormade software support for each type if specification might bring down computer costs to a more pleasant level. The drawback of aiming at separate specifications is that what is documented in one phase will not be directly usable in subsequent phases except as a general body of knowledge about the system.

There is a considerable cost associated with a formalization of systems specifications. Without short term benefits from the formalized system specification which are comparable to the costs of formalizing the project staff will be reluctant accept the idea of developing formal specifications. Only if a development project is so large that controlling and maintaining the requirement specification becomes

a serious problem will the idea of computer aided systems development be appealing to the project management.

Despite the difficulties we have mentioned in this paper, we want to point out that the production rate in the projects which have used CASCADE/2, measured in lines of code/man year, competes very favourably with production rates reported in the literature [7]. What we do not know is if this should be attributed to the use of CASCADE/2, to the use of formal systems models, to high competence in the project staff or to a combination of these factors.

We also want to make explicit, that incompetent project management can blow any project, despite the quality of development techniques which are used.

We have in this research project shown that information systems specifications can be formalized, that the formal specifications can be handled by a computer and that there is some benefit associated with doing so.

There is room for substantial improvements in this technique. One of the major long term problems will be to interface the different kinds of specifications to the software product such that automatic programming is realized on a practical scale and such that product changes are automatically reflected at the software- and requirement-specification level.

The alternative of developing tailormade software tools for each phase seen in isolation has already been mentioned.

There is a lack of theoretical knowledge in the field of systems development. We believe that a substantial improvement of the state-of-the-art is dependent on a solution of some problems which still belong to basic research. We still lack a workable definition of the concept of information. We also have problems concerning concepts like consistency, completeness and flexibility, just to mention a few. A theory of software design still seems to be quite far away.

The state-of-the-art of information systems development, based on formal specification technique, is dependent on the level of knowledge of these subproblems. When comparing with the progress in the solution of important subproblems over the last few years, we are convinced that our general framework for computer aided information systems development will prove to be valid in the years to come.

REFERENCES

[1] 2nd International Conference on Software Engineering, 13-15 Oct. 1976. San Fransisco, California, Proceedings.

[2] Bubenko, Langefors, Sølvberg (eds):
"Computer Aided Information Systems Analysis and Design",
Lund, Sweden, 1971.

[3] Aanstad, Skylstad, Sølvberg:
"CASCADE - A Computer Based Documentation System", In [2]

[4] Auglænd, Sølvberg:
"A Technique for Computerized Graphical Presentation of Information Systems to be Used in Systems Design",
In: "Approaches to Systems Design", National Computing Center, England.

[5] Sølvberg:
"The Use of Models and Associated Software in the Design of Wicked Systems"
In: Grochla, Szyperski: "Information Systems and Organizational Structure",
de'Gruyter, Berlin, 1975.

[6] Fredriksen, K:
"Brukererfaringer CASCADE",
In: Proceedings, Nord Data 75, Oslo 23-26 juni 1975, (in Norwegian)

[7] C.E. Walston, C.P. Felix:
"A Method of Programming Measurement and Estimation",
IBM Systems Journal, No. 1, 1977, pp 54-72.