A DESIGN METHODOLOGY FOR DISTRIBUTED REAL-TIME CONTROL SYSTEMS BASED ON AUGMENTED PETRI NETS

Bernardus Rudolf Kruger

A thesis submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Doctor of Philosophy in Engineering

Pretoria, 1985

DECLARATION

I declare that this thesis is my own, unaided work. It is being submitted for the degree of Doctor of Philosophy to the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination to any other University before.

B.R.Kruger

29th day of August 1985

ABSTRACT

This thesis proposes the use of augmented Petri nets (A-nets) as a design methodology for distributed computer control systems. It uses the mathematical base of Petri nets to achieve the required ability of predicting the capabilities of the software design at an early stage of the life cycle.

A-nets differ from Petri nets in that multi-valued attributed A-tokens are used instead of the binary tokens of Petri nets. The advantage of A-tokens is that they may be used to model systems where not only binary values are required.

To be able to deal with A-tokens, various new transitions apart from the normal transfer transition of Petri nets are introduced. These transitions are used to operate on the attributes or the value of the attributes of the A-tokens. The merge transition is used to combine attributes to form a new A-token. The seperating transition is used to seperate an A-token into its different attributes. The selection of a specific A-token is achieved by using the selector transition. Other types of transitions are available to operate on the attribute value.

A hierarchical breakdown of the design is used to deal with complexity. Subnets are used to be able to

present understandable diagrams on the various levels of the hierarchical representation of a design.

The use of A-tokens is demonstrated by means of process control examples as well as a model of a communications channel for real-time distributed systems.

OPSOMMING

Hierdie tesis stel die gebruik van aangepaste Petri-netwerke (A-netwerke) voor vir gebruik in stelselontwerp vir verspreide rekenaarstelsels. Dit gebruik die wiskundige basis van Petri-netwerke om die vereiste vermoë te verkry om die eienskappe van die programmatuurontwerp vroeg in die lewenssiklus te kan voorspel.

A-netwerke verskil van Petri-netwerke in dié opsig dat 'n veelwaardige A-merk met attribute in plaas van die binêre merke van 'n Petri-netwerk gebruik word. Die voordeel van A-merke is dat stelsels gemodelleer kan word wat nie net binêre merke vereis nie.

Om A-merke te kan hanteer word verskeie nuwe merkgenerators gedefinieer. Hierdie generators word gebruik om die attribute of die waarde van die attribute van die A-merke te manipuleer. Die menggenerator word gebruik om attribute te kombineer om 'n nuwe A-teken te vorm. Die skeigenerator skei die verskillende attribute van 'n A-teken. Die seleksie van 'n spesifieke A-teken word met behulp van die kiesgenerator gedoen. Ander generators is beskikbaar om die waardes van attribute te wysig.

'n Hiërargiese ontleding van die ontwerp word gebruik om kompleksiteit te hanteer. Subnetwerke word gebruik

om verstaanbare diagramme vir die verskillende vlakke van die ontwerp te skep.

Die gebruik van A-merke word deur middel van prosesbeheervoorbeelde geïllustreer. 'n Model van 'n intydse kommunikasiekanaal vir verspreide rekenaarstelsels is ook met 'n A-netwerk gemoduleer om die effektiwiteit van hierdie modelbouproses te toon.

PREFACE

During the period 1968 to 1970 the author was involved in the automation of a slabbing mill. This was one of the first computer control installations in heavy industry in South Africa. This project involved the automation of the various functions of an existing slabbing mill. The sequencing of slabs on the roller tables, control of the screwdown as well as data logging was involved.

During 1972 to 1975 a new hot strip mill with a three-computer system was commissioned. The author received extensive training in the USA and was responsible for the hardware installation as well as fo. the direct digital control software. Sequence control of the roller tables as well as gauge control in the finishing mill were two of the functions performed.

During 1976 to 1978 a computer maintenance department was developed to maintain approximately 14 different computer control systems.

During 1978 a maximum demand control system, based on microcomputers, was developed. This system received pulses for kWh measurements from switchyards spread over approximately a 5 km^2 area.

During 1979 the author was involved in the planning

and design of a distributed PLC control system for the conveyor system of a large harbour complex. A paper regarding this system was read at the SACAC Users Forum in Pretoria in 1979.

The underlying theory of A-nets as well as the example of the conveyor system were presented at IFAC, DCCS-85 in California, USA.

A B.Sc. (Ing.) degree in electrical engineering was awarded in 1968, a B.Ing. (Hons) (cum laude) in electronic engineering in 1980 and a M.Ing. (cum laude) in electronic engineering in 1981. All of these degrees were awarded by the University of Pretoria, South Africa.

Opgedra aan:

- my ouers; wat die drome gewek het
- my leermeesters; wat die drome gevoed het
- my vrou; wat met die drome saamgeleef het
- my kinders; waarop die drome gebou is!

ACKNOWLEDGEMENTS

The author would like to thank the following persons for their assistance:

- My supervisor, Professor MacLeod, for his guidance, support and encouragement.
- Professor Rodd, for the initial guidance and the creation of the opportunities to do the study.
- Professor Dr. H. Kopetz, who introduced me to many interesting facets of distributed computer control systems.
- Mr. C.G. Theron of ISCOR Vanderbijlpark, where the initial experience on computer control was gained.
- Mr. Johann Pienaar for his assistance regarding some aspects of matrix algebra in appendix A.
- Connie de Klerk who typed this manuscript and generally assisted me.
- Rita Schlebusch who, once again, did the drawings perfectly.
- Ilse Bigalke for the language editing.

The author also wishes to acknowledge the provision of facilities by the AEC and the University of the Witwatersrand. The staff of the libraries are especially thanked for their help in locating the required background material.

CONTENTS page

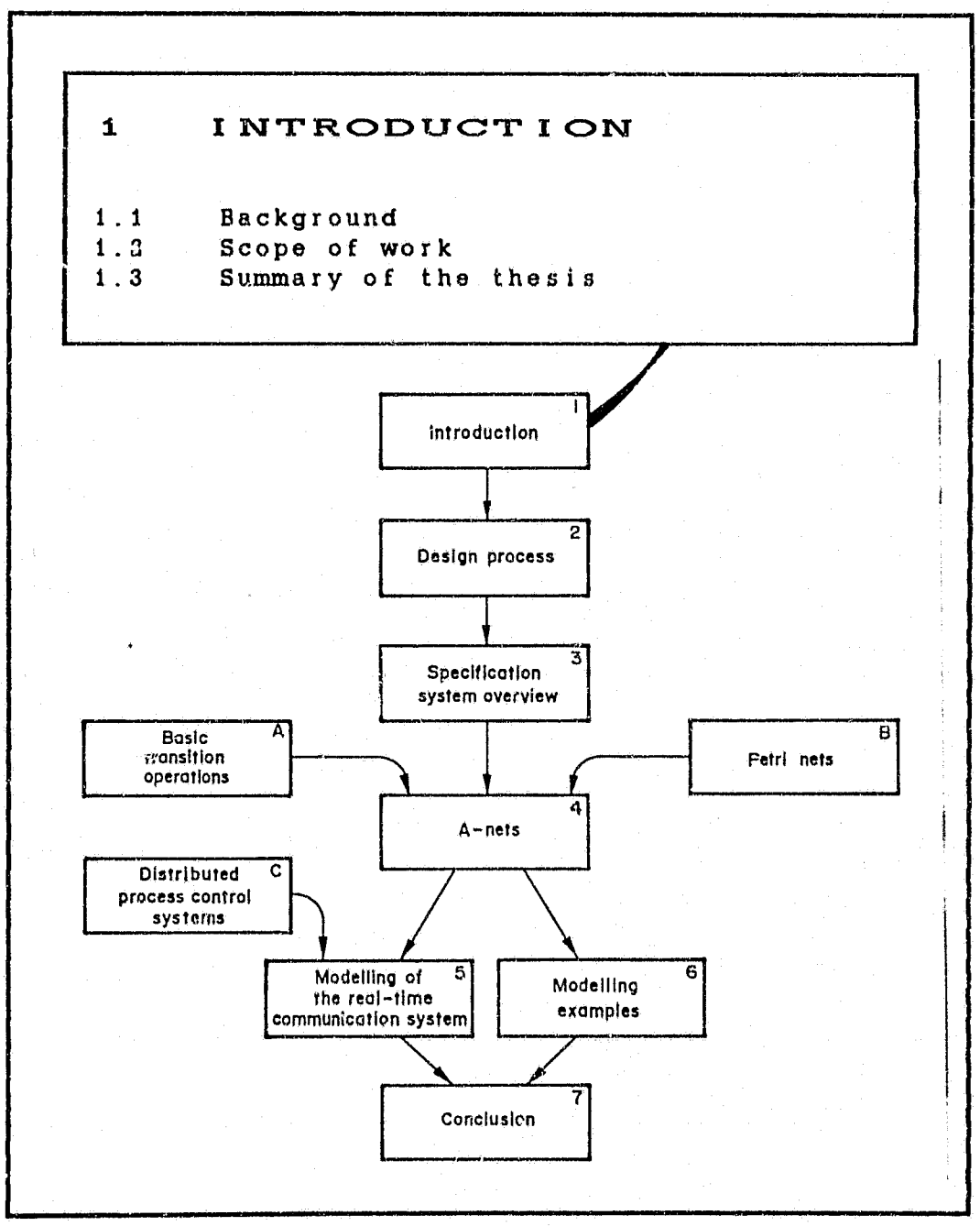
APPENDICES

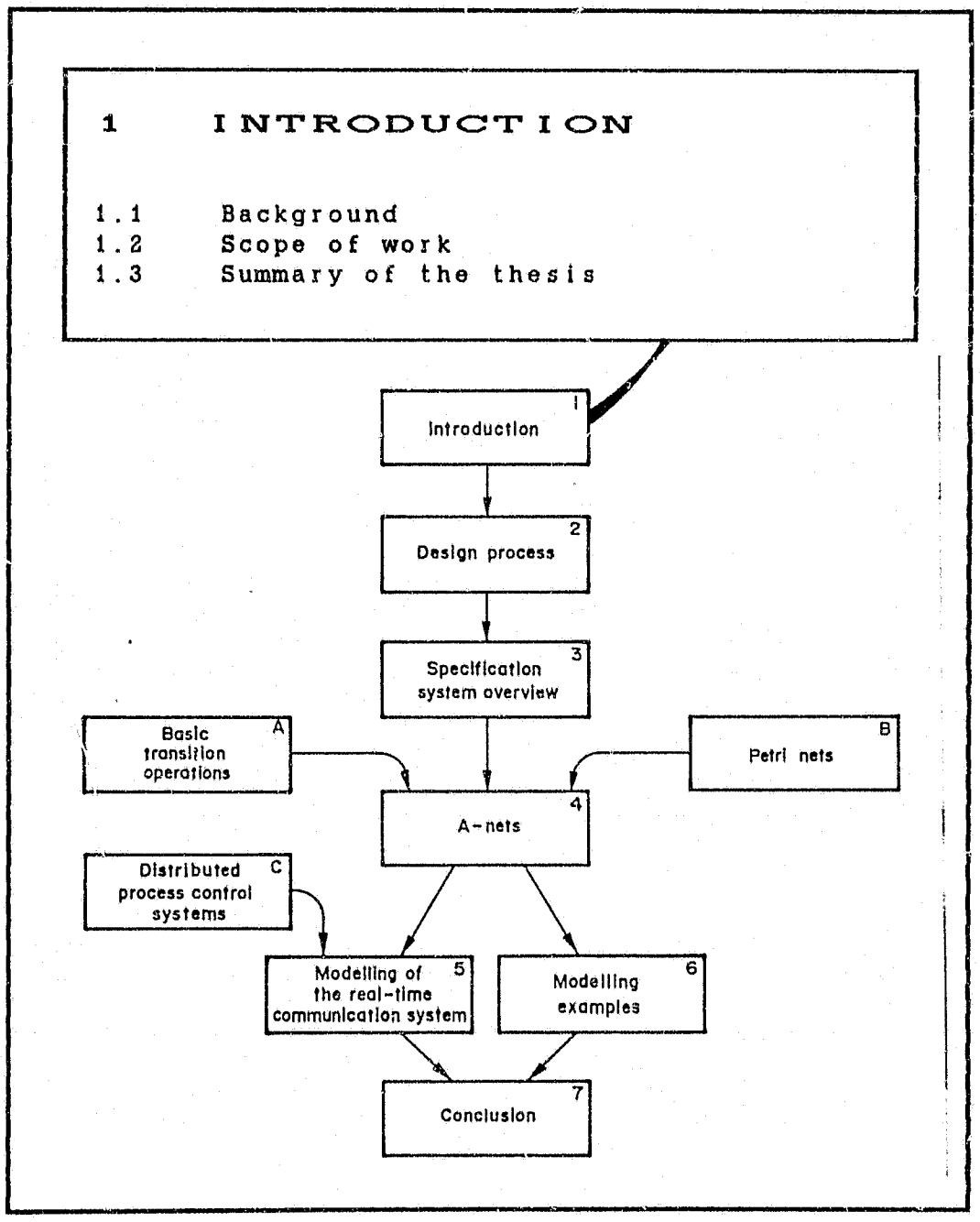
LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Places	$P=\{p_1,\ldots p_n\}$
Transitions	$T=\{t_1,\ldots t_m\}$
Input function	I
Output function	O
Tokens	μ
Duration time	$D=\{d_1,\ldots d_m\}$
Priority	$X=\{x_1,\ldots x_m\}$
Attributes of tokens	$A=\{a_1,\ldots a_q\}$
Token vector	$\lambda=\epsilon_1 a_1+\ldots+\epsilon_q a_q$
Value of attribute	$\epsilon_h=\alpha_{hh}$, β
- definer	α_h
- magnitude	∇_h
Unit value of attribute	ν
Nil value of attribute	ω
Don't care value of attribute	σ
A-token types	$B=\{B_1,\ldots B_v\}$
Transfer transition	T_N
Merge transition	T_M
Seperating transition	T_B
Arithmetic operation transition	T_A
Conditional firing transition	T_C
Selector transition	T_S


1 INTRODUCTION
1.1 Background
1.2 Scope of work
1.3 Summary of the thesis
introduction 1
Design process 2
Specification system overview 3
Basic transition operations A
Petri nets B
A-nets 4
Distributed process control systems C
Modelling of the real-time communication system 5
Modelling examples 6
Conclusion 7


1 INTRODUCTION
1.1 Background
1.2 Scope of work
1.3 Summary of the thesis
Introduction 1
Design process 2
Specification system overview 3
Basic transition operations A
Petri nets B
A-nets 4
Distributed process control systems C
Modelling of the real-time communication system 5
Modelling examples 6
Conclusion 7

C H A P T E R 1

INTRODUCTION

1.1 Background

The availability of cheap digital computing hardware as well as of standardised local area networks makes the use of computer control feasible in a wider range of applications than before. However, one of the limitations to the wider application of computers remains the expense and the difficulty of software creation. This was demonstrated by Boehm (1976) in his well-known graph reproduced in Fig. 1.1.

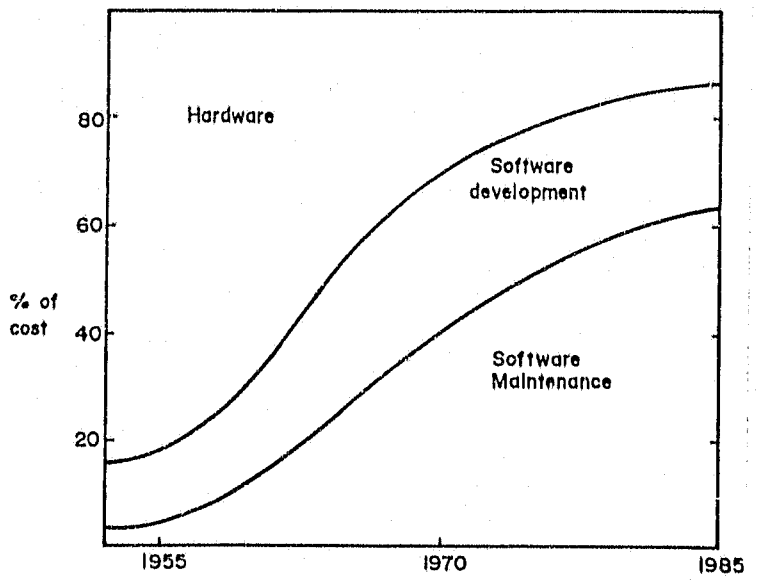


Fig. 1.1 Hardware - software cost trends

(Boehm, 1976)

-1.2- Introduction

Software development practice appears to be lacking solid scientific foundations. Writing a program which is to be used only by the original author, does not present a problem. However, the development of a large software system involving several individuals and where the life expectancy of the software produced is more than a decade, is substantially more difficult and is very similar to the development of other modern complex engineering products. Furthermore, software development is characterized by a much greater logical complexity than the development of most other engineering products, particularly in applications involving real-time control and automation. An overview of the many issues and problems encountered in the development of large software systems is provided by Brooks (1979).

The research reported in this thesis concerns the design, implementation and maintenance of large industrial control systems. A new viewpoint is proposed where the design is implemented by making use of an approach based on sound mathematical principles.

In this approach, the design is subjected to analysis. This allows one to be able to predict the correctness of the design at an early stage in the design process. Verification of the performance of a software design is currently done only in the testing phase and it continues into the operation phase. This is not

acceptable. Boehm (1976) has illustrated that it is more expensive to find an error in later phases of the system life cycle (Fig. 1.2). The errors introduced in the early phases of the life cycle are also more difficult to locate and costlier to rectify the later they are found. This problem is even more serious in the case of control systems, where once the operation phase is entered a production loss could occur when errors have to be rectified. It is therefore an aim of the research reported in this thesis to provide the required mathematical base for a design methodology for software systems for industrial process control. This mathematical base provides a means to the designer to predict the outcome of his design at an early stage. It also provides a discipline to be used in the design process.

1.2 Scope of work

The following elements are of importance during the design and development of an engineering product:

- the design of the architecture of the product. This is the determination of an arrangement of functions in such a way that the product characteristics are optimized with respect to some objective.
- a set of tools used to support the activities of conceiving, constructing, testing and evaluating the product.
- a set of scientific theories, methods and

disciplines that form the conceptual base accepted and used by the development personnel.

In modern engineering the last-mentioned element is of crucial importance for the following reasons:

- it provides the predictive power (possibly through a mathematical theory) to be able to analyse the design and predict the behaviour of the product before it is constructed.
- it plays a dominant role in guiding the choice of the architecture of the design as well as in the selection and development of the tools.

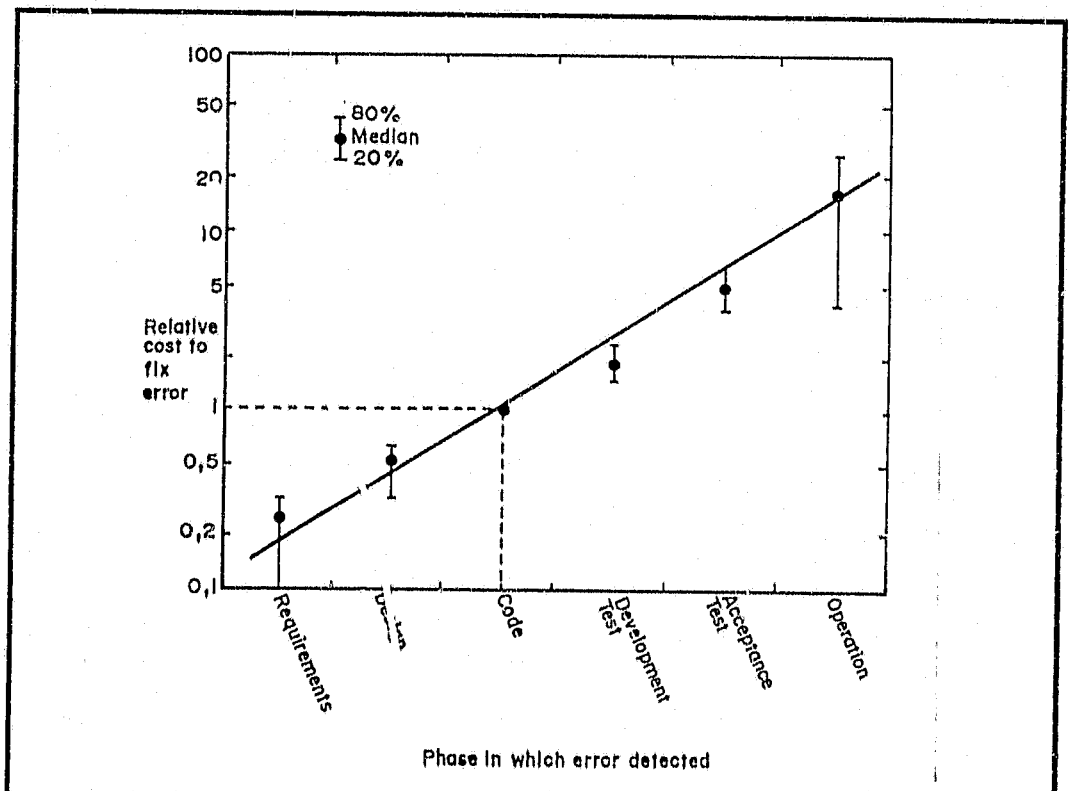


Fig. 1.2 Software cost v. phase detected

(Boehm, 1976)

-1.5- Introduction

The methodologies currently in use for the design of software are mainly based on methods and management techniques. Structured design methods (Stevens, 1974), for example, achieve a lot to design better software. These methods, however, lack the theoretical base that can be used to predict the behaviour of the software product. This lack of a theory that can be used to predict the outcome of the design can lead to unsuitable architectures for the product.

The present study was done to provide a design methodology for distributed computer control systems (DCCS) as part of a DCCS research program (Rodd, 1982). The programming primitives for this DCCS have been discussed in detail by MacLeod (1983).

It is argued in this thesis that a new approach using a design methodology based on augmented Petri nets (A-nets) provides a theoretical base for the software design process through all the system life cycle phases. The design philosophy relies on a model based on finite-state machines implemented with A-nets. These A-nets may be used to model the plant as well as the control system. The A-nets produced for the design provide a means of mathematical analysis and also prediction of certain properties of the design before it is implemented. Analysis is based on the methodology available for the analysis of Petri nets. A-nets also support a hierarchical design structure.

-1.6- Introduction

Finer detail of the design may be modelled as subnets.

1.3 Summary of the thesis

Chapters 4, 5 and 6 introduce the theory and use of A-nets and form the major part of the new proposal of this thesis. A-nets are developed from the basic theory of Petri nets presented in appendix B.

Chapter 2 consists of a discussion of different view points of the design process. The older models based on the life cycle phases of the design process do not supply enough information to be of great value. An information-gathering process over the total life cycle is proposed as a better model for designing a system. A common methodology over the total life cycle will therefore be beneficial. The limitations of the human designer are considered.

Chapter 3 provides an overview of existing design methodologies or specification systems. The following important issues emerge:-

- understanding between all parties concerned
- modelling of the environment or plant
- ways to provide quick prototyping
- a hierarchical breakdown.

A-nets are introduced in Chapter 4. A formal development of the theory is given. A-nets differ

from Petri nets in that the A-tokens may contain multi-valued attributes. A-tokens are described as vectors. A simple example is used to illustrate the extended modelling capabilities of A-nets. Some A-net detail is discussed in appendix A.

The real-time communication channel discussed in appendix C is modelled with the aid of A-nets in chapter 5. Subnets are created to be used in the modelling of distributed systems. The ability of A-nets for the design of a complex communication system is demonstrated.

Chapter 6 presents models of typical applications. The plants as well as the control system are modelled in each case. The ability of A-nets to create easily understandable models for industrial control systems is demonstrated.

A proposed automated modelling system is described in chapter 7. This work will be undertaken at a later stage as an implementation of the concepts proposed in this thesis.

Appendix B provides the basic Petri net theory required for the development of A-nets. Various useful extensions to Petri nets are also discussed. Petri nets are compared with finite-state machines. The modelling power of Petri nets is demonstrated by

means of a simple example.

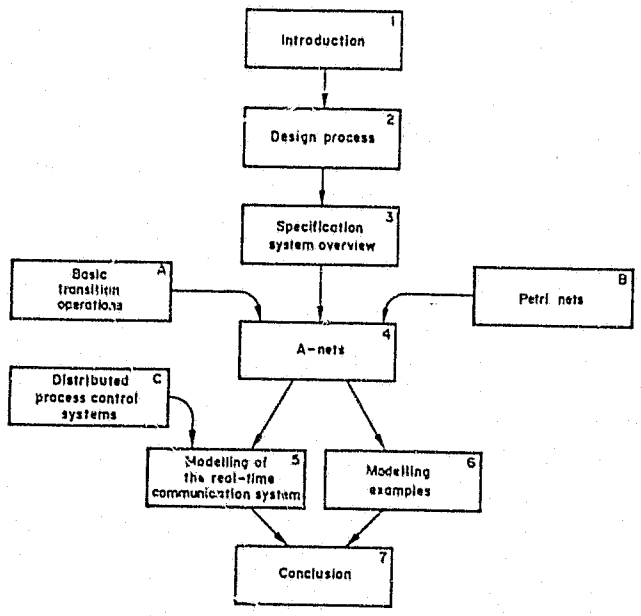
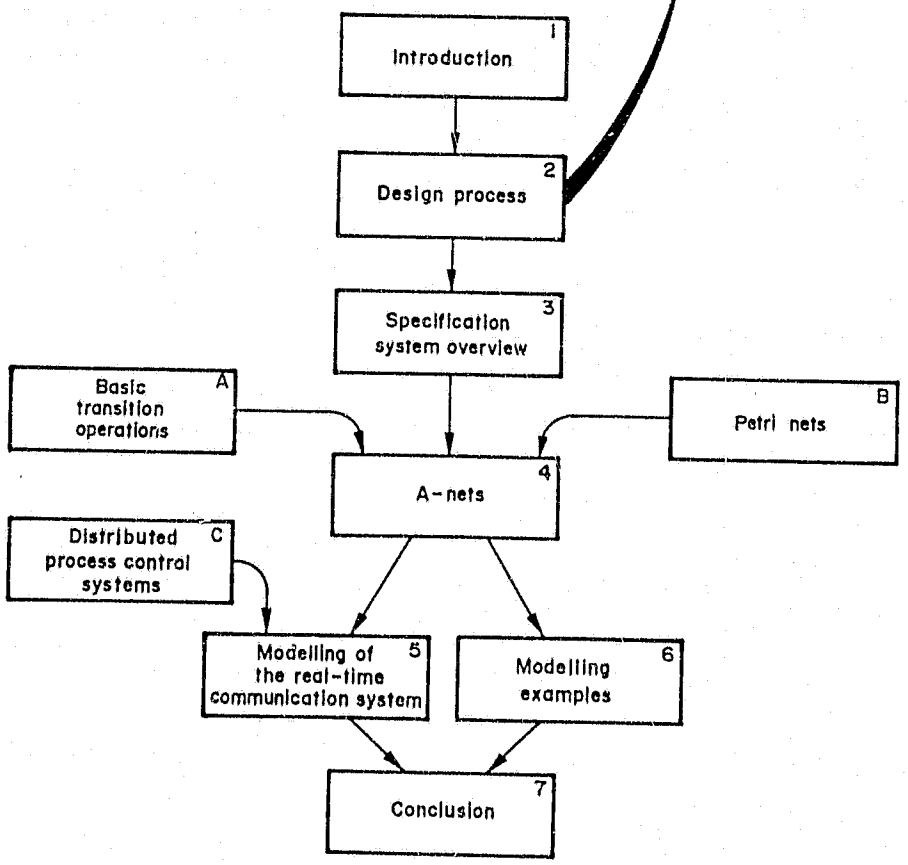


Fig. 1.3 Thesis layout

Fig. 1.3 presents a diagrammatical layout of the thesis.

2 THE DESIGN PROCESS



C H A P T E R 2

THE DESIGN PROCESS

2.1 Introduction

Software engineering is still in its infancy if for instance compared to civil engineering. The design process, however, is a problem-solving process and this process is similar for all engineering design endeavours. This chapter, therefore, looks at the design process not only from the software engineering side, but from a wider perspective to obtain the required principles.

The design process includes all aspects of the transformation of the user's needs into a scheme so that an implementation can be carried out leading to the fulfilment of the needs. The product of the design process is a design.

This chapter explains the design process and the design models that play a role in the selection of a design methodology for software systems for large industrial process control. This chapter further illustrates that the design process is a difficult

undertaking. It shows how complexity is dealt with and it discusses the limitations of the human being as a designer. Problem-solving methods are discussed as an introduction to design models.

2.2 The design process

A design process is required for the creation of a software product and it is therefore important to look at what exactly the design process consists of. In this section the nature of the design process, the human being as designer and the problem-solving process are discussed.

The transformation of the user's needs into a design (the design process) is difficult for various reasons. The design process

- is a wicked problem
- is complex
- is ill-defined at the start
- is a process carried out by human beings
- deals with unique problems.

These factors complicate the design.

2.2.1 The nature of the design problem

It is not easy to find an answer to the question "What is design?" Jones (1969) suggested the following

-2.3- The design process

definitions and descriptions:

- A creative activity - it involves bringing into being something new and useful that has not existed previously (Reswick, 1965)
- The optimum solution to the sum of true needs of a particular set of circumstances (Matchett, 1968)
- The imaginative jump from present facts to future possibilities (Page, 1966)
- Engineering design is the use of scientific principles, technical information and imagination in the definition of a mechanical structure, machine or system to perform prespecified functions with the maximum economy and efficiency. (Fielden, 1983)
- A goal-directed, problem-solving activity (Archer, 1965)
- Decision making, in the face of uncertainty, with high penalties for error. (Asimow, 1962)
- Simulating what we want to make (or do) before we make (or do) it as many times as may be necessary to feel confident in the final result (Booker, 1964)
- Relating products with situation to give satisfaction (Gregory, 1966)
- Finding the right physical component of a physical structure (Alexander, 1963)
- The conditional factor for those parts of the product which come into contact with people (Farr, 1966)

-2.4- The design process

- The performing of a very complicated act of faith (Jones 1966)

It is evident from the above quotations that design is a complex activity and a single definition will not satisfy everybody. It is therefore not suprising that different design methodologies exist.

The various factors of design that are stressed are:

- problem-solving activity
- simulation
- human interface
- the use of scientific principles
- economy and efficiency
- customer satisfaction
- creative activity.

The design of large systems normally has the following properties:

- requirements are ill-formulated
- confusing information exists
- there are many clients and decision makers with conflicting values.

Design as a wicked problem

The above factors led Rittel to describe design as a wicked problem. The following are some properties of wicked problems taken from Bazjanac (1974):

-2.5- The design process

- Wicked problems have no definitive formulation. Every time a formulation is made, additional questions can be asked and more information can be requested.
- Every formulation of the wicked problem corresponds to the formulation of the solution (and vice versa). The information needed to understand the problem is determined by one's idea or plan of a solution. In other words, whenever a wicked problem is formulated there must already be a solution in mind.
- Wicked problems have no stopping rule. Any solution that is formulated can be improved or extended. One can stop only because one has run out of resources, patience, etc.
- No wicked problem and no solution can be tested in totality. In other words, even if a test is "successfully" passed, it is still possible that the solution will fail in some other respect.

The above points clearly indicate that it is impossible to first define the wicked problem and then look for a solution. Solutions are generated while the problem is formulated. It is easy to produce a better design at the second attempt, while testing of the solution remains a problem. In finding a model for the design process, this is an important factor. One can however not implicitly state that all designs are wicked. A degree of wickedness may be attributed

to a specific design. All designs must therefore be handled with care in this regard.

Dealing with complexity

The situation is, however, not as bad as it looks. Techniques to deal with the complexity of large systems have been studied and evaluated.

Many ways of describing systems exist. Klir (1969) uses the following constant traits of a system:

- The set of external quantities together with the resolution level
- A given activity
- Permanent behaviour
- Real UC structure (structure of universe and couplings)
- Real ST structure (state-transition structure).

The UC and ST structures are important to us. The UC structure is the base for system analysis and consists of a set of elements and a set of couplings between these elements. The ST structure of a system is the states of a system with the possible transfers and conditions for transfers between the states. State space methods are extensively used to deal with complex systems.

Another method to deal with complexity consists of

decomposition in hierarchical levels. Mesarovic (1970) distinguishes between three types of hierarchical systems, namely:

- the level of description or abstraction
- the level of decision complexity
- the organizational level.

All three types are used to decompose a complex system. Simon (1962) used hierarchical decomposition on systems like biological evolution. He shows that the description used for complex systems naturally tends to be hierarchical. He also concludes that complex systems in nature tend to be hierarchical.

This section has illustrated that the design process is complex and that a single satisfactory definition does not exist. Design is a wicked problem and there is a need for a methodology to deal with complexity. Complexity is dealt with by describing the system dynamics with the various states that it may be in. Another method of dealing with complexity consits of dividing complex systems into subsystems with lesser complexity through which a hierarchical breakdown is achieved. These methods are well-proven and form the base of the proposed design methodology for software generation for industrial control systems.

-2.8- The design process

2.2.2 The human being as designer

The limitations and characteristics of the human designer are also to be taken into consideration. The following factors are important:

- complexity-handling capability
- generation of innovative ideas.

Dealing with complex systems

In the design process, the understanding of complex ideas by the human designer as well as by all the other people involved, is important. According to Chestnut (1970) the information required to understand a complex system should describe its structure, its distinguishing qualities, and the magnitude, probability and time of the variables and parameters. He summarizes various methods of representing the three attributes. He concludes that "The great amount of information required to describe typical large-scale systems can be decreased considerably by the use of graphic structures as well as suitable names for their distinguishing qualities and their magnitude, probability, and time characteristics". The use of graphical representation is therefore important.

Another method of dealing with complexity is to decompose the system into smaller subsystems. Hierarchical decomposition has already been discussed.

-2.9- The design process

The amount of information that a human being can receive, process and remember is limited. Miller (1956) gave this as "The magical number seven, plus or minus two". In his article he has compared results of various researchers concerning the information channel capacity of human beings regarding various stimuli. He then concluded that with the limitation of our span of immediate memory, the span of absolute judgement or unidimensional judgement is usually somewhere around seven. There are ways of side-stepping this number, like

- relative, rather than absolute judgements
- increasing the number of dimensions along which the stimuli can differ
- making a sequence of several absolute judgements in a row.

He differentiates between the span of absolute judgement and the span of immediate memory. Absolute judgement is limited by the amount of information. Immediate memory is limited by the number of items. Miller differentiates between bits of information in the first case and chunks of information in the second. The span of immediate memory is independent of the number of bits per chunk.

This allows one to deal with complexity in a top-down decomposition of a problem by limiting the number of

items to seven and allowing for "chunking" of the information.

Complexity is basically dealt with in two ways:

- By structuring a complex system in a hierarchical way. This is the base of the top-down design strategy as reported by Bergland and Gordon (1981).
- By chunking of the information presentation to allow for the capabilities of the human being.

Generation of innovative ideas

Creating innovative ideas is important. Himmelblau (1974) states that "innovation, creativity, and other words are used to describe that characteristic of design that leads to novel designs rather than conventional results". Fig. 2.1 shows this process diagrammatically. How is this creativity jump accomplished?

Rubenstein (1982) states that problem solving in general involves search activities. The brain consists of two parts. The one side controls analytical, logical and sequential thinking, while the other controls orientation in space, identification and recognition of patterns, faces and sites, and in general the more artistic functions consisting of holistic and less analytical thinking. These two parts of the brain are connected and can work

together. By using both sides one can accomplish the required jump for a real innovative design.

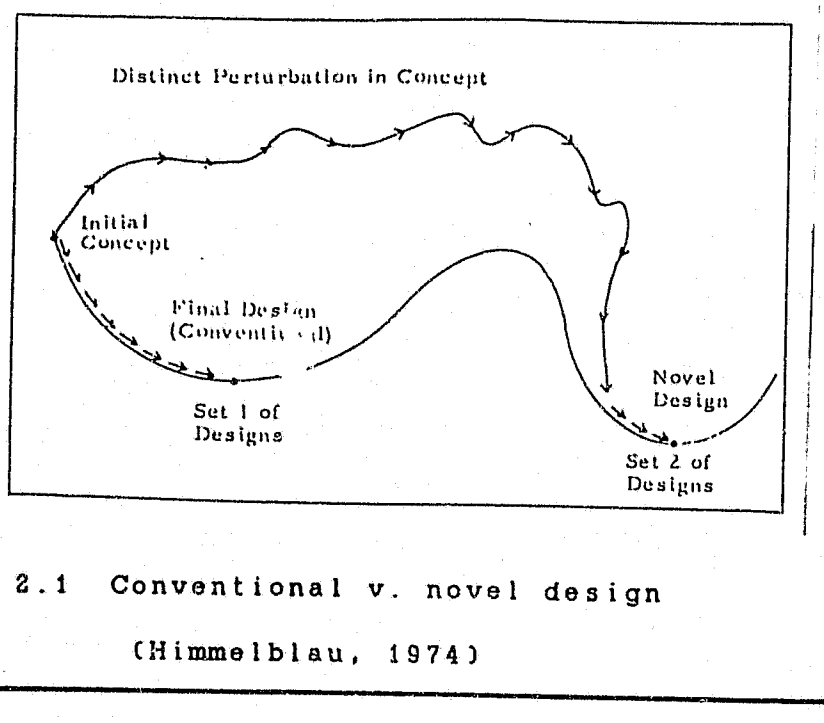


Fig. 2.1 Conventional v. novel design

(Himmelblau, 1974)

Jones (1969) presents methods for searching for ideas that force people to make use of both sides of their brain. Methods mentioned are:

- brainstorming
- synectics
- removing mental blocks
- morphological charts.

This section has considered the human being as designer. The automation of the design process can relieve the designer of the routine tasks. It can help him in the evaluation of vast amounts of data and of alternatives. Innovative design, however, still

remains a human activity. A design methodology should therefore assist the designer in dealing with complexity and in generating innovative ideas. This can be achieved by allowing the designer to use both sides of his brain. A graphical structure using a hierarchical breakdown, to limit the items to approximately seven, is a necessary prerequisite for the proposed design methodology.

2.3.3 The problem-solving process

The design process may be looked upon as a problem-solving process. The problem-solving process has been examined by many psychologists. Basically two types of models for the problem-solving process emerge, namely phase models and mechanistic models. The phase models endeavour to explain the various phases in a problem solving situation, while the mechanistic models look at problem solving as being constructed out of many basic operations.

Phase models

A multiphase model was presented by Dewey (1910). He divides the thought process into the following parts:

- a felt difficulty
- its location and definition
- suggestion of a possible solution
- developing by reasoning of the bearings of the

-2.13- The design process

suggestion

- further observation and experiment leading to its acceptance or rejection; that is the conclusion of belief or disbelief (Dewey, 1910).

The important factors emerging from this thought process are:

- the generation of a proposed solution and
- the evaluation of that solution.

Johnson (1955) identifies three types of productive thoughts:

- trial and error
- insight
- gradual analysis.

He proposed the following three-phase model for problem solving:

- preparation
- production
- judgement.

It seems that the problem-solving process always includes some form of synthesis and evaluation. The synthesis may involve different subphases like trial and error.

-2.14- The design process

Mechanistic models

The mechanistic view on problem solving consists of subdividing a problem into basic transforms. The parallel between the human thought process and digital computers provides a different way of looking at problem solving.

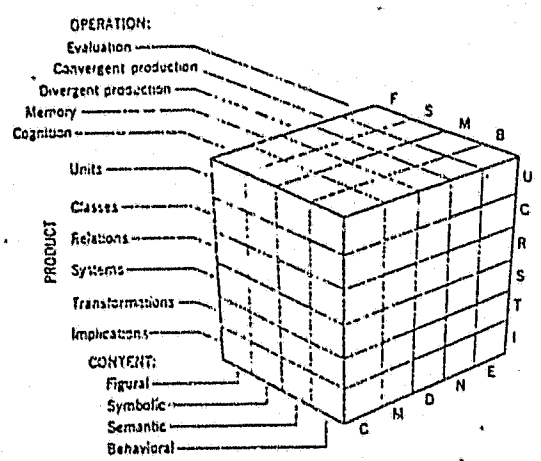


Fig. 2.2 The structure-of-intellect model (Guilford, 1967)

Newell, Shaw and Simon (1958) considered problem solving as a series of elementary information processes. The solution to a problem is a search in a very large space for possible solutions. This implies a search algorithm and a manner of judging.

The following model for problem solving was developed by Guilford (1967). A structure-of-intellect model is presented in Fig. 2.2. The three aspects operation,

product and content are used to construct this three-dimensional morphological model. The problem-solving model is shown graphically in Fig. 2.3. This is an operational model with inputs from the environment, the soma and the memory.

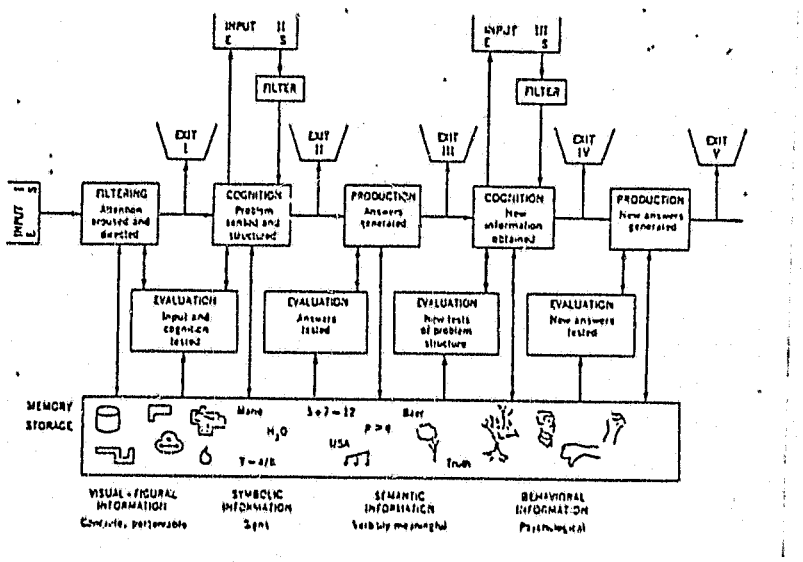


Fig. 2.3 Model of problem solving

(Guilford, 1967)

Important in the structure-of-intellect model are the following operations:

- Evaluation
- Convergent production
- Divergent production.

These operations are also shown in the model in Fig. 2.3. The evaluation operation in the model is used to make a decision on whether to continue, exit or redo.

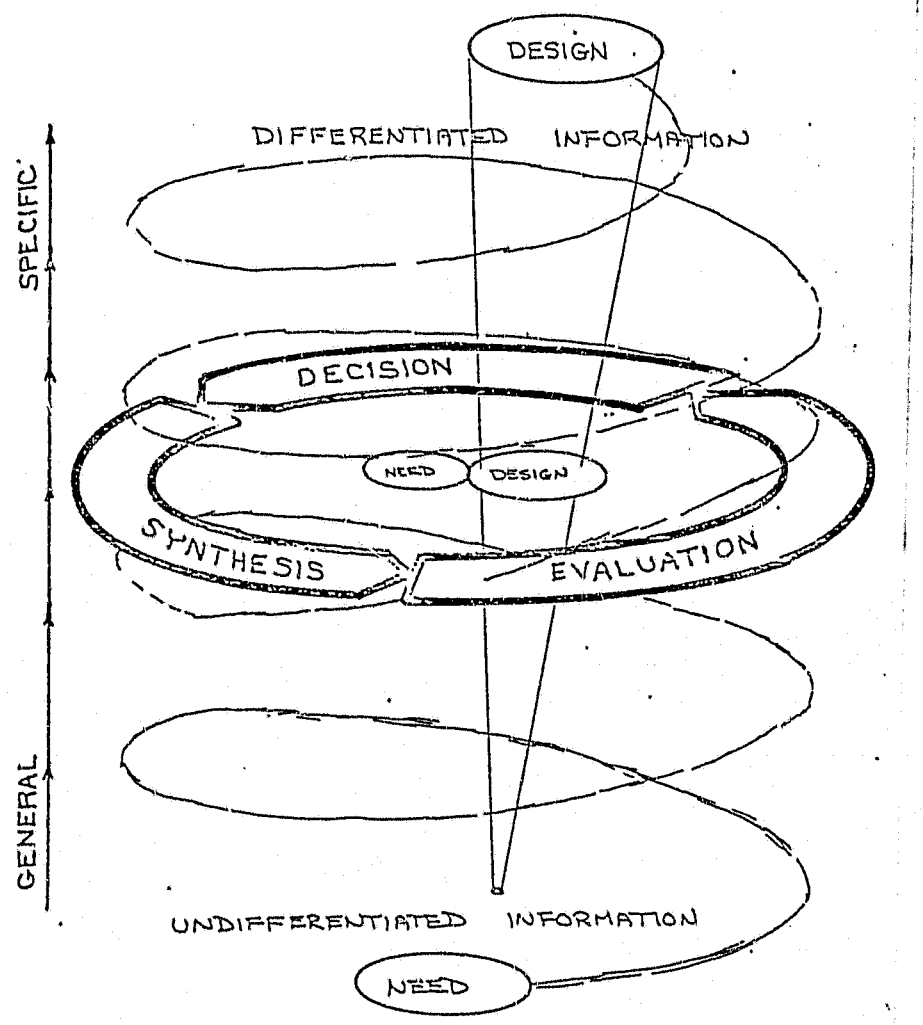


Fig. 2.4 The problem-solving spiral
(Becker, 1973)

Becker (1973) proposed a problem-solving "spiral that rises from a large body of undifferentiated, general, information to the specifics of a detailed solution. The process is initiated by the recognition of a need and terminated by the acceptance of a solution". (Fig. 2.4) This model illustrates that the design process starts with a general and terminates with a specific solution. This is then a transition from

informal to more formal information, or as is stated by Becker, a progress from undifferentiated to differentiated information. Decision, synthesis and evaluation operations are required to complete the design process.

This section has indicated that the modelling of the problem-solving process is difficult. However, it provides the base for the design models in the following paragraph. This again serves as the base for the model of the software engineering process.

2.3 Design Models

According to Jones (1969), most writers agree that design includes the following stages:

- analysis
- synthesis and
- evaluation.

He calls it

- divergence
- transformation and
- convergence.

Divergence is necessary to extend the boundary of the design situation to create a large search space. The use of both sides of the brain in this phase may lead

-2.18- The design process

to innovative designs.

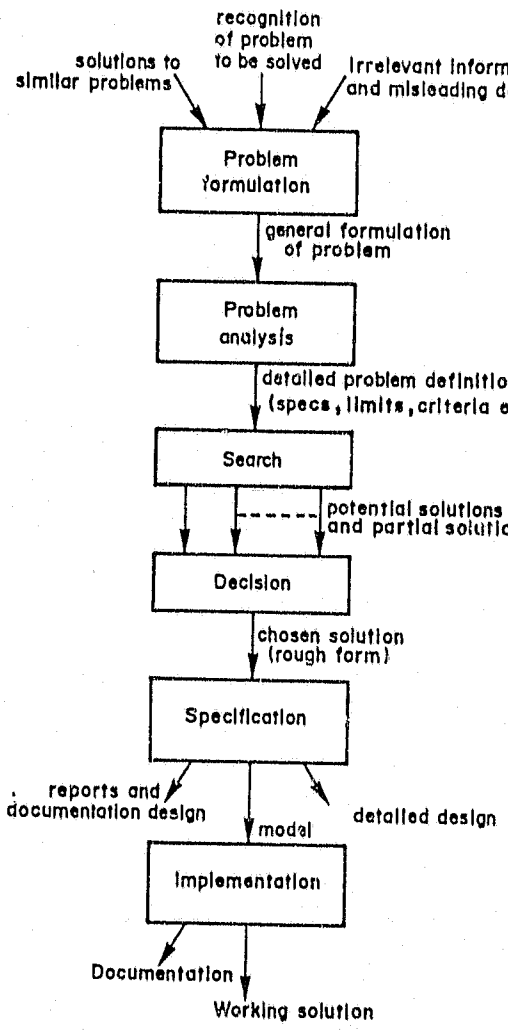


Fig. 2.5 Engineering design process

(Jensen and Tonies, 1979)

Transformation is the phase where the problem space is transformed into one or more solutions in the solution space. The architecture of the solution is selected in this phase.

Convergence is the selection process during which one

superior design emerges as the winner amongst all the available solutions. The analytical process during this phase is of importance.

Jensen and Tonies (1979) present a model of the engineering design process in Fig. 2.5 which illustrates the three phases as was discussed by Jones.

Divergence is indicated by problem formulation, problem analysis and search. The transformation then takes place and convergence takes place in the decision, specification and implementation blocks.

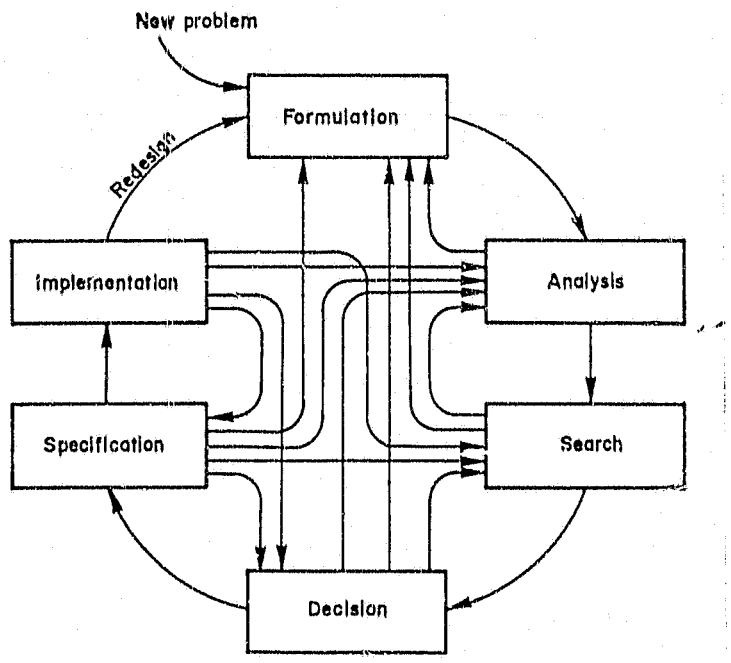
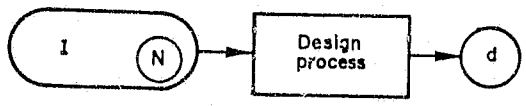


Fig. 2.6 Realistic engineering design process (Jonsen and Tonies, 1979)

-2.20- The design process



I Input to design process
N user's need
d design

Fig. 2.7 The general design process

(So, 1979)

This linear model, however, is not a good approximation of the real design process. Fig. 2.6 shows a realistic engineering design process by Jensen and Tonies which endeavours to illustrate the complex relooping of the iteration process of the design process.

So (1979) presents a model based on the work of Becker (1973) in Fig. 2.7. This model is based on the following fundamental elements:

- the design space, D
- the performance space, P
- the context, C
- the evaluation function, E
- the decision function, Δ and
- the conception function, X.

All these elements and their interrelationships are discussed by So and Becker. So (1979) then relates the model to the well-known software design models.

-2.21- The design process

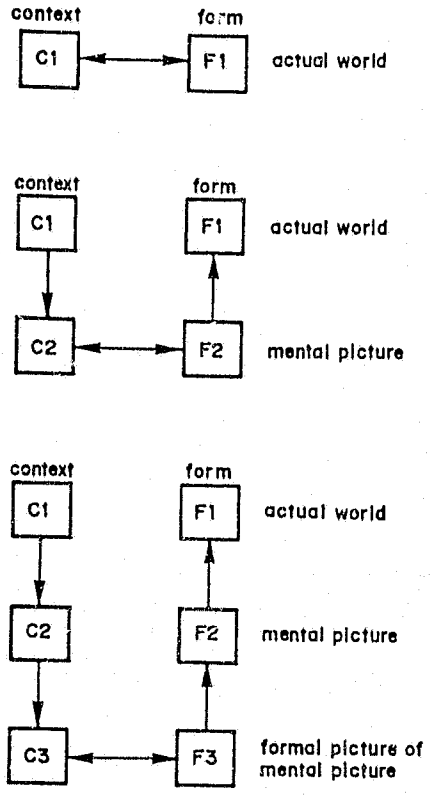


Fig. 2.8 Synthesis of form

(Alexander, 1967)

Alexander (1967) states that "every design prob m begins with an effort to achieve fitness between tw entities: the form in question and its context. The form is the solution to the problem; the context defines the problem". He explains how misfits between the form and context lead to a closed-loop process which gets rid of the misfit. He then explains three models of the design process as illustrated in Fig. 2.8. Fig. 2.8(a) shows the unselfconscious process. The shaping of the context C1 and the form

-2.22- The design process

F1 takes place in the actual world. The human being is an agent in this process and reacts by changing misfits. Fig. 2.8(b) shows where the shaping takes place in the mental picture of the designer. Conceptual interaction between the conceptual picture of the context (C2) and ideas, diagrams and drawings which stand for the forms (F2), takes place to eliminate the misfits. Fig. 2.8(c) shows a further abstraction. C3 is a formal mathematical picture derived from C2. F3 is derived from C3, - it may be intuitive but must be well understood. Alexander then illustrates the importance of decomposition as well as of hierarchical structures for the design process. Misfits imply that the designer and client have something to compare the requirements against. One could add an additional level to Alexander's model by making a prototype. This prototype may for example be C4, derived from the mental picture C3. Studying this prototype increases the knowledge base of the problem which is fed back to C3. Prototyping is therefore of value in the design process.

Models of the software life cycle follow the general models that were introduced up to now. The model introduced by Boehm (1976) is probably the oldest and best known example and it is shown in Fig. 2.9. It is based on the phases of the life cycle and rework is emphasized. Tonies (1979) shows a similar model based on the phases of a project. This model, shown in

-2.23- The design process

Fig. 2.10, uses the concept of entropy. Some of the resources (energy) are wasted in the process (entropy). This entropy can surface in a later phase and cause reworking - i.e. the use of extra energy.

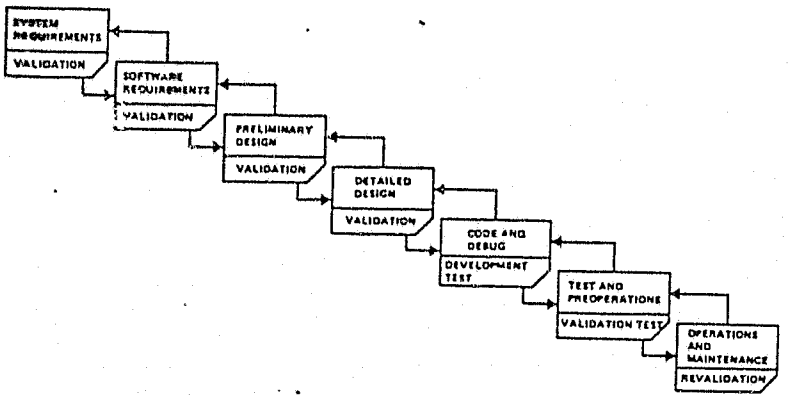


Fig. 2.9 Software life cycle

(Boehm, 1976)

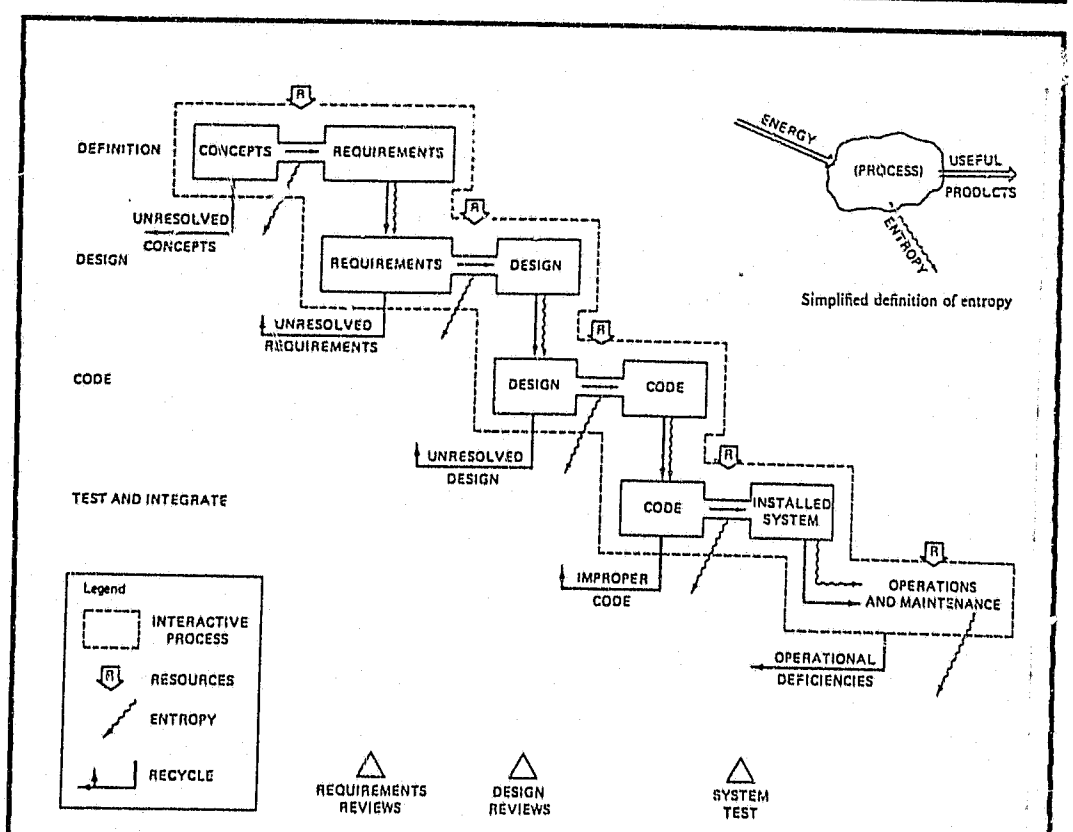


Fig. 2.10 Software life cycle interactive process

(Tonies, 1979)

The design process

Fig. 2.10, uses the concept of entropy. Some of the resources (energy) are wasted in the process (entropy). This entropy can surface in a later phase and cause reworking - i.e. the use of extra energy.

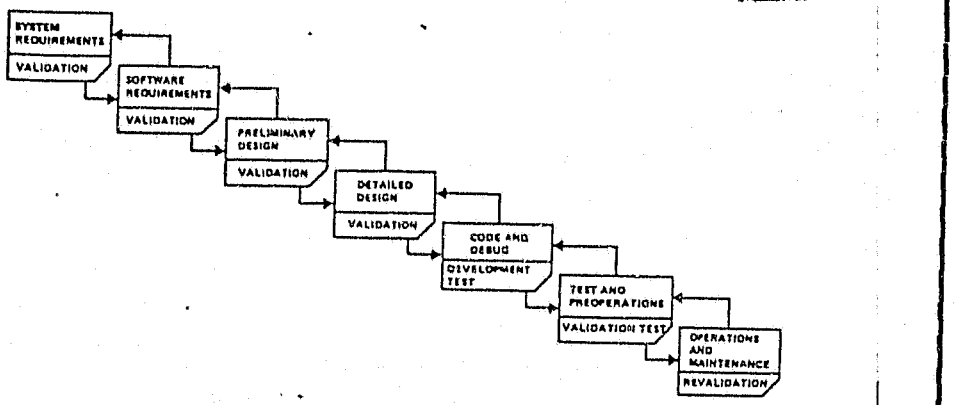


Fig. 2.9 Software life cycle

(Boehm, 1976)

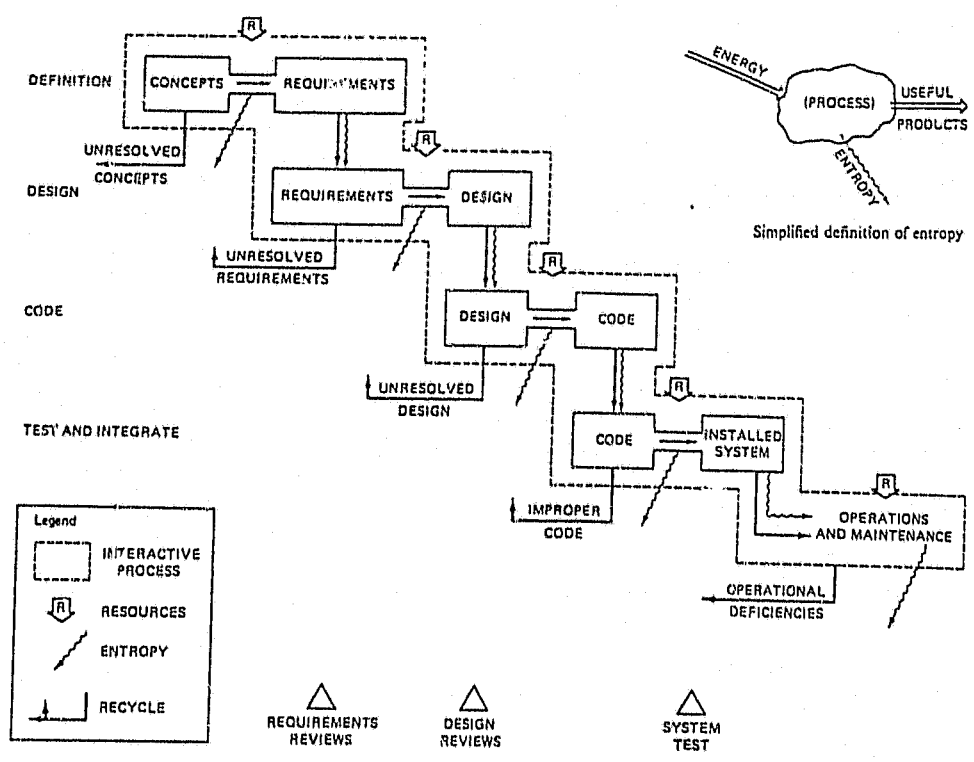


Fig. 2.10 Software life cycle interactive process

(Tonies, 1979)

-2.24- The design process

The life cycle models based on the phase of the project were mainly developed by persons interested in the project management side of software engineering. Some people working with formal specification systems don't agree with these models. Ludewig (1982), Lauber (1982) and Hesse (1981) use a different modelling method, namely the degree of formality required to specify the system at any time during its life time as criterion. At the start of the project, when only vague ideas exist, an informal specification will help with the developing of ideas. Eventually, the system is fixed in a very formal implementation language. This same idea is also mentioned by Teichroew and Hershey (1977): "The process by which the initial concepts evolve into an operational system consists of a number of activities each of which makes the concept more concrete". Fig. 2.11 shows the increase in formality required as the idea becomes more concrete and is transformed into code. Different viewpoints exists regarding the way to continue. Theoretists immediately want to formalize the information as the ideas are produced. In current practice the formalizing is delayed until the coding phase. It is proposed that a better solution exists somewhere between these two extremes. Innovative ideas are destroyed by formalizing too early. Formalizing too late produces a result that may not be the required result. Early formalizing allows one to predict certain properties of the design.

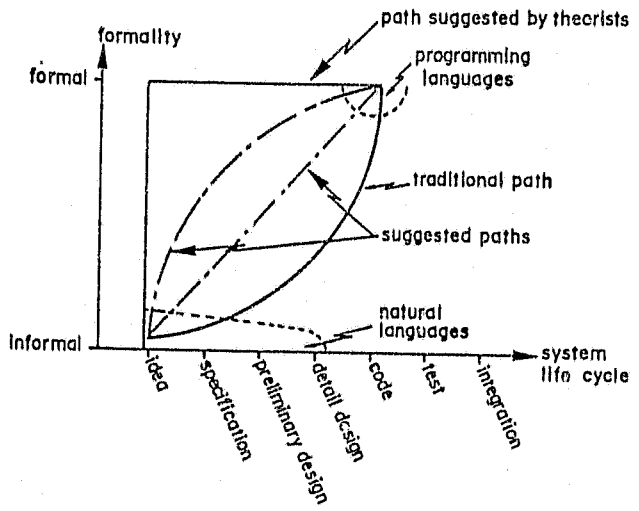


Fig. 2.11 Formality v. life cycle phase

(Adapted from Ludewig, 1982)

According to Bazjanac (1974), design is a learning process. The designer documents his view of the problem and the solution as he sees it at any point in time. He keeps on learning about the problem in his search for a solution. This may then alter his view of the problem and the solution. The designer keeps on refining and documenting his new formulation of the problem until he is satisfied or he has run out of time.

2.4 Conclusion

This chapter has shown that the design process is complicated to understand and to model. Older models based on the phases of the process are of very little use. The total life cycle of a system, including the

use and maintenance thereof, has received very little attention. The following informal model is therefore proposed (Fig. 2.12). The project phases are retained. The important facet of each phase, however, is the gathering of information. This information pool exists for the lifetime of the system. Information loss may occur at any time. This information loss is equivalent to the raising of the entropy of the system. Borgida (1985) deals with this information-gathering process in the requirements phase from an artificial intelligence point of view. This is an important viewpoint that needs further examination.

Any design methodology should therefore use a common basic system throughout all the life cycle phases of the project to be of optimum use.

The following important factors about the design process were mentioned in this chapter:

- a hierarchical design reduces the complexity to a situation that can be dealt with by the human being
- innovative ideas require unorthodox methods
- phase models on their own are not sufficient to describe the design process
- the design process is an iterative process
- prototyping is an aid to detect misfits
- graphical representation helps to understand a design

-2.27- The design process

- formalizing a problem is critical
- design is an information-gathering process

The large number and complexity of the above factors explain why it is so difficult to propose a satisfactory tool, theory or technique for software design. This chapter has indicated why the design process is complex, the limitations of the designer and the problem-solving process culminating in design models.

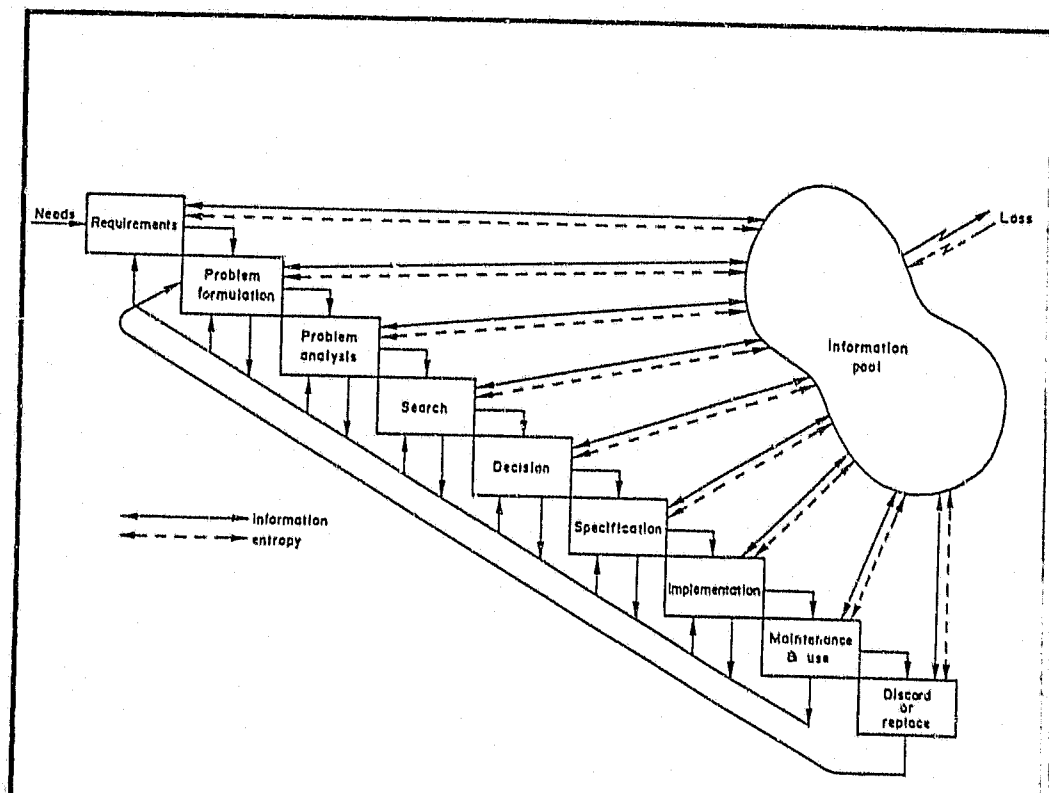
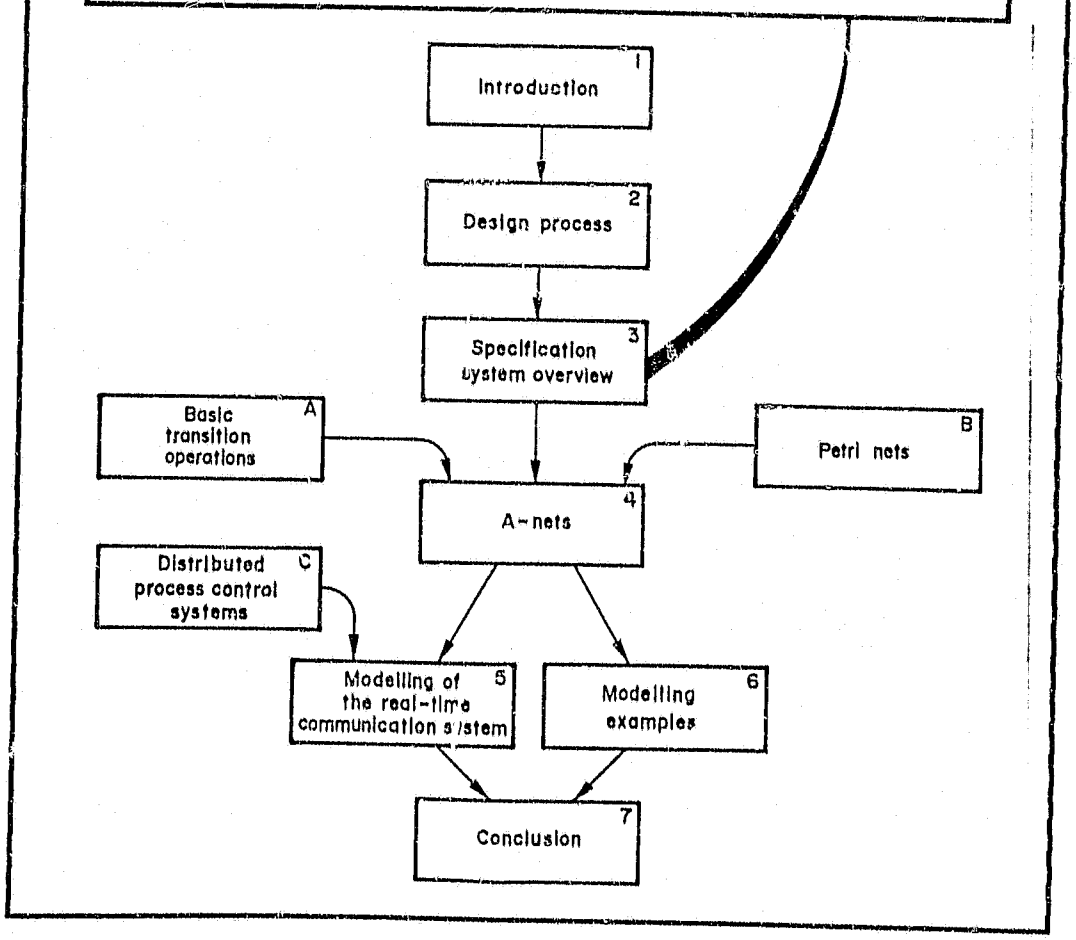


Fig. 2.12 An information-gathering process

3 SPECIFICATION SYSTEMS OVERVIEW

3.1 Introduction
3.2 Examples of specification systems
Data flow structures
Program logic structure
3.2.1 Structured analysis and design technique
3.2.2 Problem statement language/problem statement analyser
3.2.3 Software requirements engineering methodology
3.2.4 A modular approach to software construction, operation and test
3.2.5 Applicative language
3.2.6 Finite-state machine based system
3.2.7 ESPRESO
3.2.8 EPOS
3.2.9 Requirements language processor
3.2.10 Motus-Quirk system
3.2.11 Grafcet
3.3 Conclusion



C H A P T E R 3

SPECIFICATION SYSTEMS OVERVIEW

3.1 Introduction

The life cycle of a software product was mentioned in the previous chapters. It was, however, not defined or discussed. The life cycle consists of the following phases:

- requirement specification
- architecture design
- detail design
- implementation
- verification
- maintenance and operation
- discard or replace.

Difference of opinion exists on the various phases, for example the viewpoint of Boehm (1976) as shown in Fig. 2.9 or that of Tonies (1979) in Fig. 2.10. The above-mentioned phases are, however, a fair representation of the phases of the software design process.

Various important documents are created during the design process, namely:

-3.2- Specification system overview

- a requirement specification. This is a document that can be understood by both the user (client) and the designer of the system. It states the needs of the user and tells what is to be done. This document is represented by N in Fig. 2.7 and it forms part of the input I to the design process.
- a design specification. This document is a product of the architecture and detail design phases. It specifies how the system is to be implemented. It is represented by d in Fig. 2.7.
- a test specification. This document is generated during the design phase and is to be used during the verification phase.

A specification is therefore the outcome of some of the phases in the life cycle to be used as an input document for the next and/or subsequeni phases. A specification system is a tool to assist the system designer, preferably during as many phases of the life cycle as possible.

Various reasons exist why the specification process could be emphasized:

- some authors see at it as the starting phase of automatic programming
- some see it as a quick prototyping system (Zave, 1982)
- to others this is the phase where errors are easily rectified. Errors introduced in this phase are,

however, difficult and expensive to find in a later phase. (Boehm, 1984)

Fig. 1.2 (Boehm, 1976) illustrates that errors made in the earlier phases are costlier to rectify. A 100:1 saving is possible by locating and solving problems early in the life cycle. This implies that it is important to verify the design at an early stage in order to detect requirement problems.

Consensus of opinion has, however, not been reached on what requirements specifications should be.

Heninger (1979) presented the following objectives:

- specify external behaviour only
- specify constraints on the implementation
- be easy to change
- serve as a reference tool
- record forethought about the life cycle of the system
- characterize acceptable responses to undesired events.

Davis (1979) mentioned the following important points that a requirements specification should answer to:

- it should specify the functions to be performed by the system, from the viewpoint of the user or the external environment.
- it should specify the performance to be achieved

by the system from the viewpoint of the user or the external environment.

- it should state design constraints.
- it should have the following audiences in mind: the customer, the management team, the designers, the system testers and the requirements writers.

Balzer (1979) presents the following criteria for judging specifications. Specifications must be:

- understandable by both parties
- testable
- maintainable.

The above discussion shows that a uniform view of requirements specifications does not exist. Important factors seem to be:

- it should state the functions
- it should be understandable
- it should be maintainable.

The criteria for requirements and design specifications are presented by Boehm (1984) and are summarized in Fig. 3.1. The major criteria are:

- completeness
- consistency
- feasibility
- testability.

-3.5- Specification system overview

Meyer (1985) presents the case for formality in specifications, but he wants to retain a natural language requirements document.

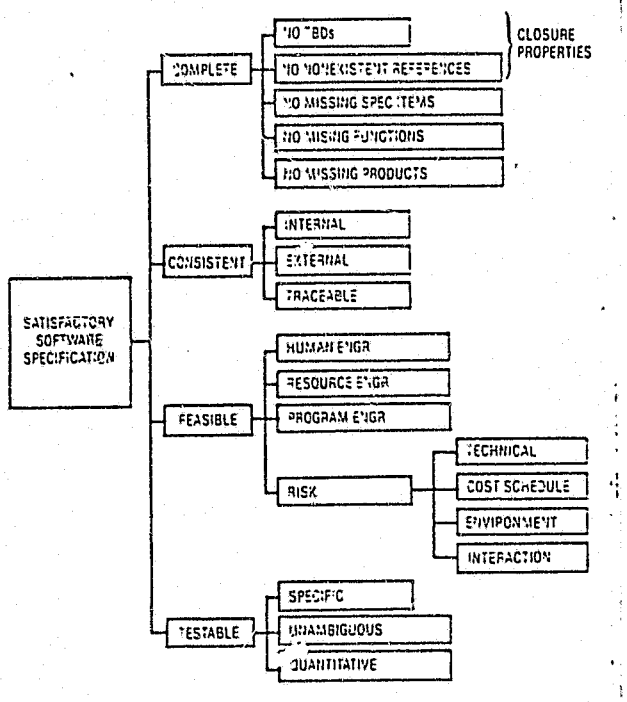


Fig. 3.1 Taxonomy of a software specification (Boehm, 1984)

3.2 Examples of specification systems

Various surveys were done on the available specifications systems, (Ramamoorthy, 1978), (Hesse, 1981) and (Ludewig, 1978), and a comparison was made between various systems on a specific example of a package sorter. (Hommel, 1980)

The various methods used and the different starting points show the state of the art of specification

-3.6- Specification system overview

systems.

Fig. 3.2 (Ludewig, 1978) shows the components of a very elaborate system in block form. The system is built around a data base where all the information pertaining to the system is stored. The input to the data base is from the:

- requirements definition
- design description
- implementation specification
- test specification.

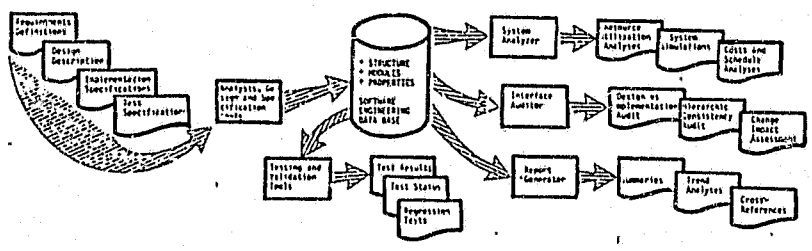


Fig. 3.2 Design system block diagram

(Ludewig, 1978)

These documents are analysed and operated on by the analyser and by the design and specification tools. These tools extract the structure and properties to build the data base.

-3.7- Specification system overview

The following main functions are performed by means of the data base:

- system analysis
- interface auditing
- report generation
- testing and
- simulation.

Not all the discussed methods are as elaborate as indicated, in fact, some systems are manual and can only be used on small projects.

Systems may be divided into two groups, namely:

- data flow structure and
- program logic structure methods.

Data flow structure

Various basic methods devised around the data flow structure have been designed:

- The Jackson design method (Jackson 1976)
- Structured system design (Gane 1979).

The following methodology is normally used:

- draw a logical data flow diagram
- compile a data dictionary
- define the logic of the processes
- define the data stores.

The program structure is then built around the data

-3.7- Specification system overview

The following main functions are performed by means of the data base:

- system analysis
- interface auditing
- report generation
- testing and
- simulation.

Not all the discussed methods are as elaborate as indicated, in fact, some systems are manual and can only be used on small projects.

Systems may be divided into two groups, namely:

- data flow structure and
- program logic structure methods.

Data flow structure

Various basic methods devised around the data flow structure have been designed:

- The Jackson design method (Jackson 1976)
- Structured system design (Gane 1979).

The following methodology is normally used:

- draw a logical data flow diagram
- compile a data dictionary
- define the logic of the processes
- define the data stores.

The program structure is then built around the data

structure.

These methods can be used in a normal data-processing environment where the handling of information is very important.

Program Logic Structure

Various methods based on the program logic structure have been devised:

- structured analysis and programming (Stevens et al 1974)
- HIPO (Stay 1976)
- top-down design (Van Leer 1976)

A hierarchical structure of modules is set up. Each module is designed in a structured way. The system is decomposed into modules. There are certain guidelines for this decomposition, but there are no firm rules. Parnas (1972) presented some criteria to be used in decomposing systems into modules of which the most important is the interaction between the modules. This binding of the modules can be:

- coincidental
- logical
- temporal
- communicational
- sequential
- functional.

-3.9- Specification system overview

The aim is to make the coupling between modules simple and obvious. Each module can then be understood individually.

Existing specification systems

A short description is given of the four oldest specification systems and thon of some that were designed with real-time systems in mind.

3.2.1 Structured Analysis and Design Technique (SADT)™

NODE INDEX AND CORRESPONDING
DECOMPOSITION STRUCTURE OF DIAGRAMS

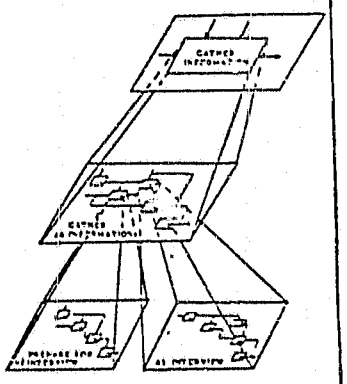
Node Index		Corresponding Decomposition Structure
A -0	Gather Information (Context)	
A0	Gather Information	
A1	Coordinate and Monitor	
A2	Prepare for Interview	
A21	Confirm Data	
A22	Determine Eligibility	
A23	Prepare Notifications	
A24	Notify Unit and Member	
A3	Interview	
A31	Confirm Preparedness	
A32	Instruct Member re Options	
A33	Gather Member Choices and Data	
A34	Prepare Forms and Instructions	
A35	Instruct Member	
A4	Prepare Items	
A5	Perform Final Actions	

Fig. 3.3(a) SADT - node index

(Ludewig, 1978)

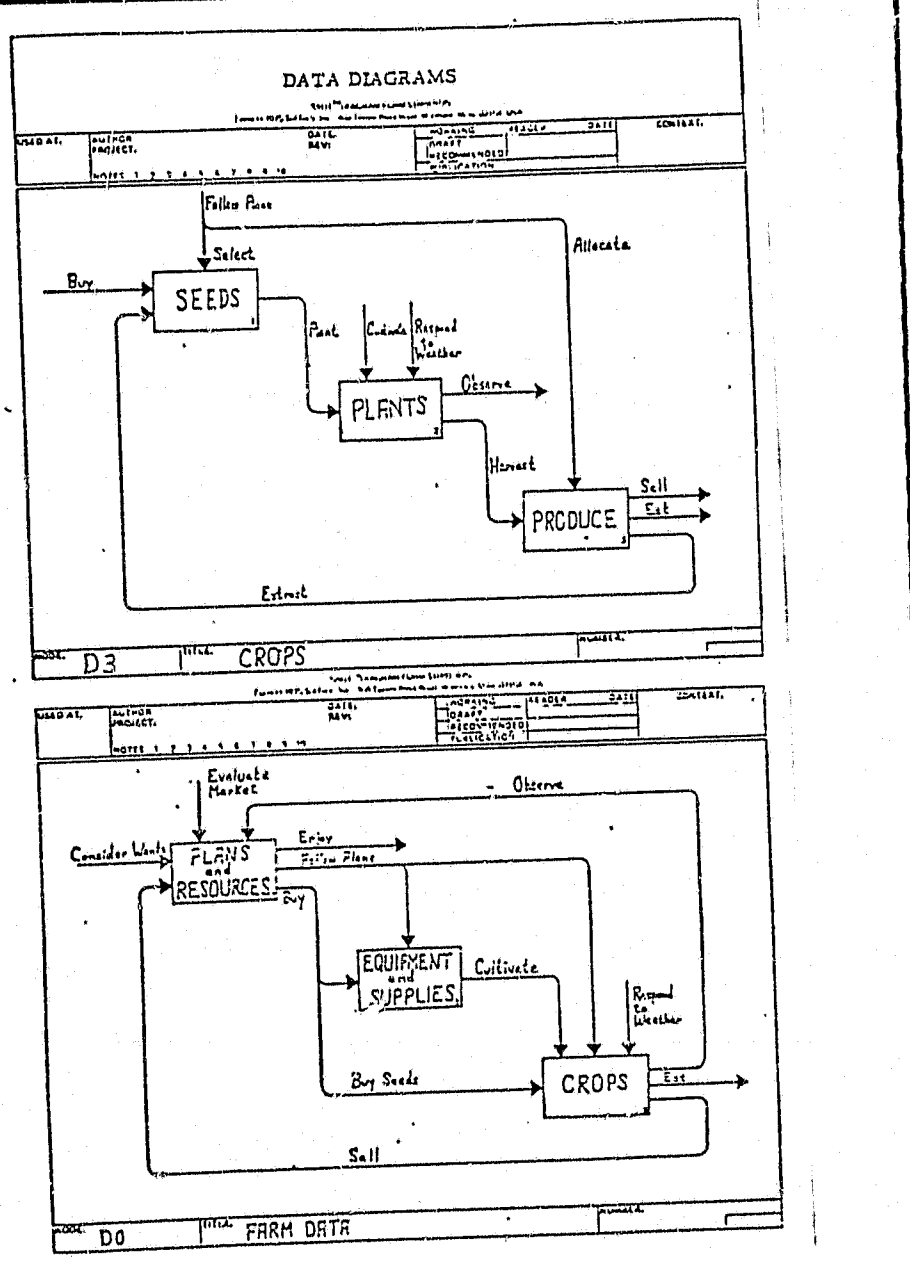


Fig. 3.3(b) SADT - data diagram

(Ludewig, 1978)

This method supports functional analysis and was designed by SoftTech (Ross 1977a, 1977b and 1985) in the early seventies. This system uses hierarchical diagrams to describe a system. Two sets of diagrams are used - the first to describe the control flow and

the second for the data flow. Fig. 3.3(a) shows the hierarchical structure of the diagrams, Fig. 3.3(b) illustrates the data and Fig. 3.3(c) the activity diagrams. As can be seen, the diagrams are kept simple. A guideline is that there should not be more than 5 to 7 blocks on a single diagram, when keeping in mind the ability of the human being as was discussed in chapter 2.

System analysis as well as design is supported by SADT. It is a manual system. To keep all the drawings up to date may require a lot of effort in a large system. An automated system is described by Bernus and Hatvany (1979).

The graphical representation is easy to understand and the communication between the system supplier and user can only benefit by its use.

3.2.2 Problem statement Language/Problem statement Analyser (PSL/PSA)

This system, developed by Teichroew (1977), is most probably the pioneer amongst the automated systems. The software system design is done in the PSL language and it is used to compile a data base. PSA is then used to analyse the system and various reports are generated. PSL/PSA was developed for data processing and it has been used with limited success for real-

time systems or process control.

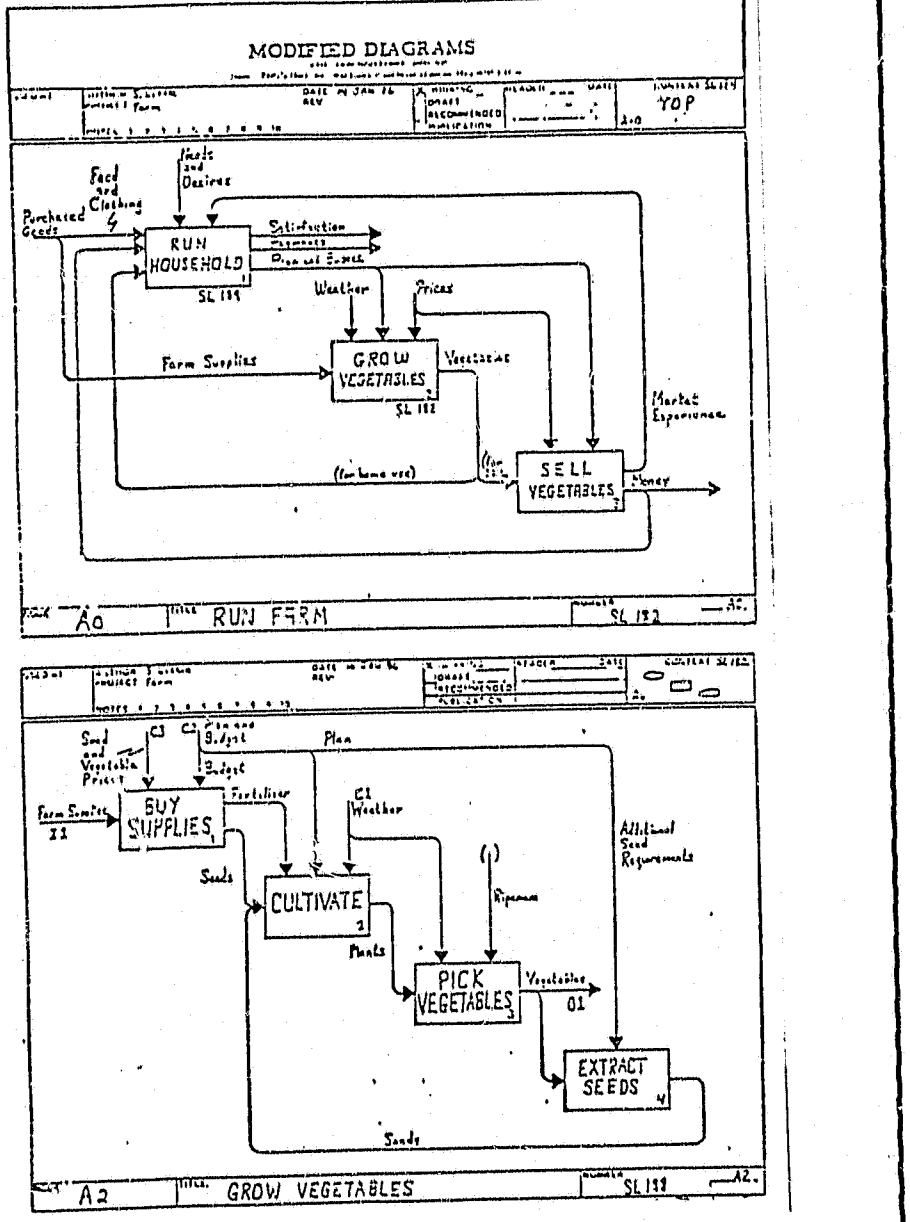


Fig. 3.3(c) SADT - activity diagram

(Ludewig, 1978)

3.2.3 Software Requirements Engineering Methodology (SREM)

SREM was developed by TRW for the Ballistic Missile

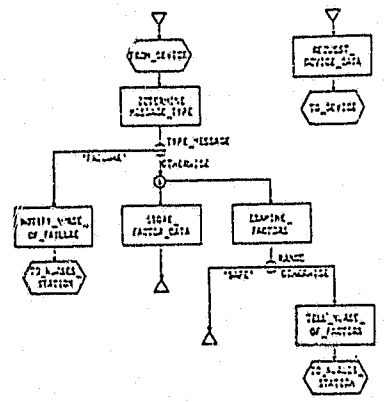
-3.13- Specification system overview

Defense Advanced Technology Center (Alford, 1977 and 1985), (Davis, 1977), (Bell 1976) and (Scheffer, 1985). SREM is one of the most elaborate systems available for real-time systems.

R-NET

The problem is the following:

"(1) A patient monitoring program is required for a hospital. (2) Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood-pressure, and skin resistance. (3) The program reads these factors on a periodic basis (specified for each patient) and stores these factors in a data base. (4) For each patient, safe ranges for each factor are specified (e.g., patient X's valid temperature range is 98 to 99.5 degrees Fahrenheit). (5) If a factor falls outside of the patient's safe range, or if an analog device fails, the nurse's station is notified".



ORIGINATING_REQUIREMENT: SENTENCE_3,
DESCRIPTION: "DEFINES ANALOG DEVICE MEASUREMENTS",
TRACES TO: MESSAGE DEVICE_REPORT.

MESSAGE: DEVICE_REPORT,
PASSED THROUGH: INPUT_INTERFACE FROM_DEVICE,
MADE UP: DATA DEVICE_NUMBER, DATA TYPE_MESSAGE,
DATA DEVICE_DATA,
TRACED FROM: SENTENCE_3.

DATA: DEVICE_DATA,
INCLUDES: DATA PULSE, DATA TEMPERATURE,
DATA BLOOD_PRESSURE,
DATA SKIN_RESISTANCE.

ENTITY_CLASS: PATIENT,
ASSOCIATES: DATA PATIENT_NUMBER,
DATA SAFE_FACTOR_RANGE FILE
FACTOR_HISTORY.

DATA: SAFE_FACTOR_RANGE,
INCLUDES: DATA [illegible],
DATA [illegible],
DATA SKIN_RESISTANCE,
DATA MI_SKIN_RESISTANCE,
TRACED FROM: SENTENCE_4.

FILE: FACTOR_HISTORY,
CONTAINS: DATA [illegible],
DATA [illegible],
DATA MINIMUM_RESISTANCE,
TRACED FROM: SENTENCE_5.

ALPHA: EXAMINE_FACTORS,
INPUTS: DATA DEVICE_DATA, DATA SAFE_FACTOR_RANGE,
OUTPUTS: RANGE,
DESCRIPTION: "THIS PROCESSING STEP FIRST RETRIEVES THE SAFE FACTOR DATA ASSOCIATED WITH THE DEVICE, COMPARES THE DEVICE DATA TO THE SAFE FACTOR RANGES FOR THE PATIENT BEING MONITORED, AND DETERMINES WHETHER THE FACTORS ARE IN_BOUNDS (RANGE_SAFE) OR OUT_OF_BOUNDS",

DATA: PATIENT_DEVICE_NUMBER,
INCLUDED IN: DATA SAFE_FACTOR_RANGE,

Fig. 3.4 SREM - example

(Ludewig, 1978)

-3.14- Specification system overview

SREM consists of:

- a requirements statement language (RSL)
- a requirements evaluation and validation system (REVS)
- a data base management system.

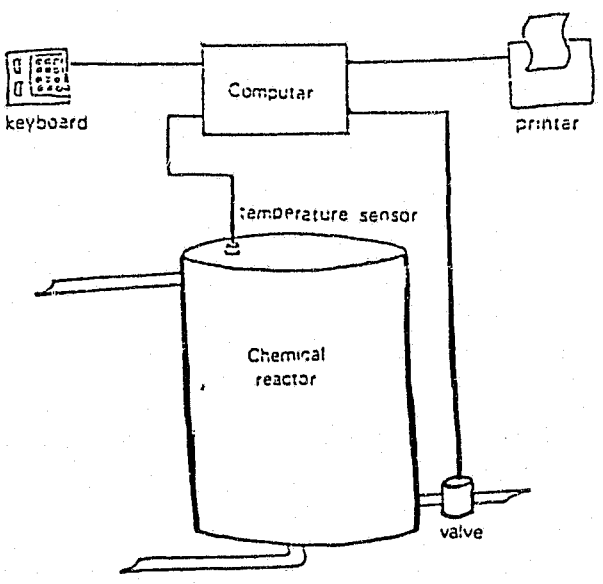
It is based on stimulus-response or requirement networks (R-nets). A graphical representation of the R-net is supplied. A simulation package is also available.

SREM is used to define the complete data-processing portion of the system. An example of the R-net as well as of the RSL language is shown in Fig. 3.4.

3.2.4 A Modular Approach to Software Construction, Operation and Test (MASCOT)

MASCOT was developed by the Royal Radar Establishment to deal with the problems of the real-time data processing of the radar systems. (Jackson, 1975) and (Jackson and Harte, 1976).

It is based on a network of cooperating parallel processes and is represented in a graphical way. The communication is asynchronous. An example is presented in Fig. 3.5.



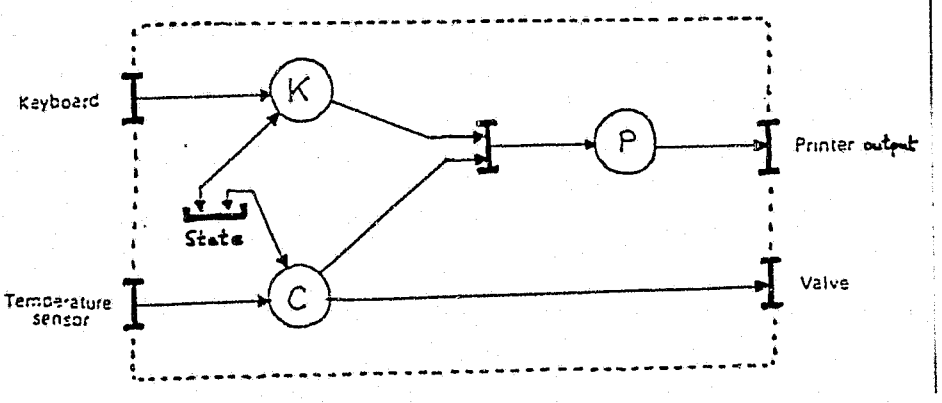


Fig. 3.5 MASCOT - example

(Ludewig, 1978)

3.2.5 Applicative Language

Zave (1982) has a different approach - in her system, the environment as well as the control system is modelled. This combined model is then executed to debug the specification. An applicative language, PAISLey, is used.

-3.16- Specification system overview

The advantages of the modelling of the total system are:

- it assists in the decomposition of the complexity
- a document exists for the environment as seen by the user and producer of the system
- future changes in the environment can be modelled
- the analyst is forced to think about how the environment is going to act and react.

3.2.6 Finite-state machine based system

Vitins (1981) uses an extended finite-state machine in his model for requirements specifications for industrial real-time automation systems.

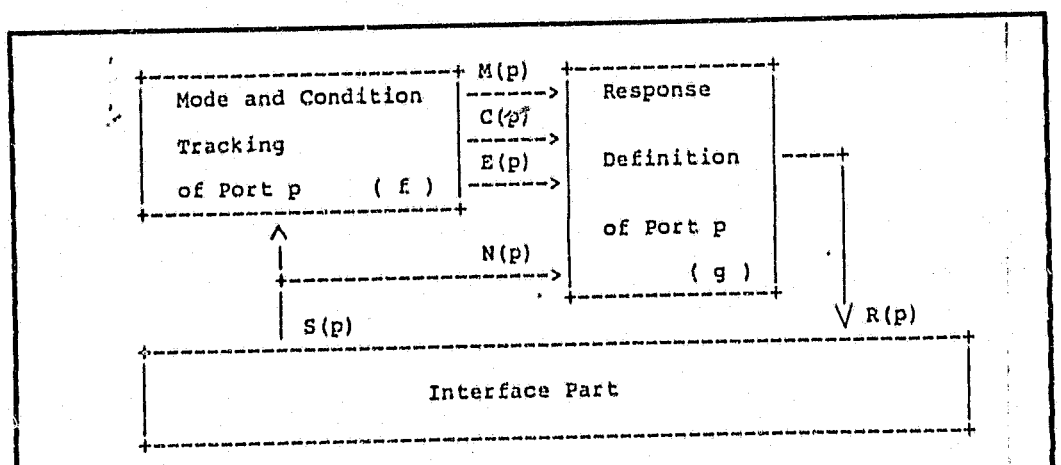


Fig. 3.6 Extended finite-state machine

(Vitins, 1981)

-3.17- Specification system overview

A schematic diagram of the extended finite-state machine is shown in Fig. 3.6. It consists of an interface part to describe all stimulus and response signals of the total system, a mode-tracking section to describe the mode transitions and a response definition section where the output functions are described. A formal description of the specified system is made in terms of state machine concepts.

3.2.7 ESPRESO

ESPRESO was developed from experience gained with PSL/PSA and a PSL/PSA derivative PCSL (Ludewig, 1980). ESPRESO is reported by Ludewig (1981a), (1981b), (1981c), (1982a), (1982b).

ESPRESO is based on the following concepts:

- as early as possible documentation (informal if necessary)
- support the designer in formalizing his problem
- hinder the designer in stating detail to early
- minimize clerical work
- create a centralized data base of the specification
- provide tools for error detection
- allow various representations.

It is difficult to implement all of the above concepts in the same system. ESPRESO is therefore a compromise and it consists of:

- ESPRESO-S, a block-oriented, non-procedural language with emphasis on data flow
- ESPRESO-W, a tool consisting of a converter, a deconvertor and a report generator.

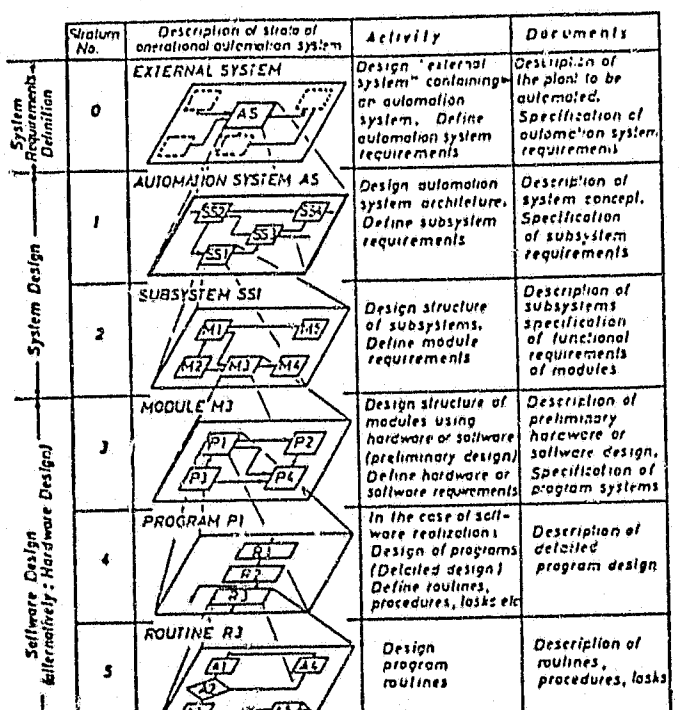
According to Ludewig, the separation of design and specification is an Utopia. Because the design process is a wicked problem, the specification must give an indication of how the design should be done. For example, the architecture of the design should be indicated by the specification.

3.2.8 EPOS

The EPOS system is reported by Biewald (1979) and Lauber (1982). It covers the complete life cycle of a system starting with the requirements and ending with coding. It supports a code generated in PEARL, a process control language. EPOS is based on a hierarchy of abstraction strata as is shown in Fig. 3.7.

It consists of:

- EPOS-R, to describe the automation system
- EPOS-S, to describe the design
- EPOS-A, to analyse and check
- EPOS-D, to generate reports
- EPOS-C, to communicate between EPOS and the user.



Definition of abstraction strata of the industrial automation system.

Fig. 3.7 EPOS - example

(Biewald, 1979)

3.2.9 Requirements Language Processor (RLP)

Davis (1982) stresses the point of readability (by the user) of the specification. Various users have different applications and therefore require different vocabularies to state their requirements.

This problem is dealt with by having a common system model and various language definition tables for the