A SURVEY OF MOST IMPORTANT AND OUTSTANDING METHODS FOR SOFTWARE ENGINEERING

József Györkös
Ivan Rozman
Tatjana Welzer
University of Maribor

UDK 681.3.006

With the presentation of most important methods of software engineering we want to contribute to the better knowledge, application and development tools that demand a systematic procedure for the software design. We have mentioned basic methods of requirements engineering and structured design which were already developed in the seventies. It was not earlier than in the eighties when complex tools for computer aided design of software were formed from these methods. In their essence, these methods may be devided into those which solve the problems by analysing the data flow and those which solve the problem by decomposing the data structure. The most important factor to estimate the suitability of application of a method is the level of covering the phases in the software life-cycle. The behaviour in real-time environment and the possibility to create consistent entiti-relationship models for more complex information systems are important in dependence upon the system.

Z objavo pregleda pomembnejših metod programskega inženirstva želimo prispevati k boljšemu poznavanju, uporabi in razvoju orodij, ki narekujejo sistematičen pristop k snovanju programske opreme. Nanizali smo temeljne metode inženirstva zahtev in strukturnega snovanja, ki so nastale že v sedemdesetih letih. Šele v osemdesetih letih so se iz njih izoblikovala kompleksna orodja za računalniško podprto snovanje programske opreme. V osnovi se metode delijo na tiste, ki k reševanju sistema pristopajo z analizo toka podatkov in tiste, ki to izvajajo z razgrajevanjem strukture podatkov. Najpomembnejša postavka pri ocenjevanju primernosti uporabe neke metode je stopnja pokritosti faz življenjskega cikla programske opreme. V odvisnosti od sistema pa je pomembno tudi obnašanje v 'real-time' okolju in možnost kreiranja čvrstih entitetno-relacijskih modelov za kompleksnejše informacijske sisteme.

Ø. Introduction

A multitude of actions leading to a consistent, realiable and well documented software is called **Software engineering**. In the attempt to normalize the "multitude of actions", several methods have been developed from the sixties onwards. Regarding their applicability and above all their "usableness", certain methods did not reach their climax earlier than in the eighties and most often as a base for automatic tools for software design. Let's see a brief survey of methods of software engineering /Kell87/

i) the sixties may be called also the period of crisis in the software development because it was recognized that it is impossible to create effective programs without a sistematic procedure. In this period the philosophy of **Structured programming** was formed (important authors : Bohm, Jacopini, Dijkstra).

ii) in the seventies the previously mentioned recognition caused that various technologies were developed (structured analisys, design) and term **software life-cycle** was introduced. In this period, new methods lived only in scientific circles, therefore this period is called "the period of searching for panacea" (important authors : Constantine, Myers, Yourdon, Ross, DeMarco, Sarson).

iii) in the eighties, the methods of the seventies experience a revival. With the integration of some methods a series of effective and commercially successfull **automatic tools** for software development appeared.

In the introduction let us consider the term " software life-cycle". The software life-cycle covers six basic steps /Porc83/:

- Requirements Analisys
- Functional Specification
- Design
- Implementation
- Validation
- Maintenance

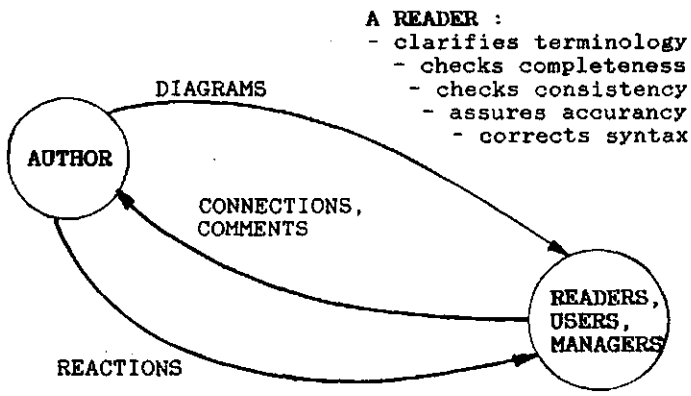
1. Requirements Engineering as a Part of Software Engineering

When we are solving problems by means of software we make a photo-copy of the real world in a logically clipped form. This form must substitute the reality in certain characteristics that are required. Therefore in the complex process of software engineering the requirements specification is of essential importance. As a rule, in this process are envolved the developer of the project and the customer. The process of requirements engineering, that is the cooperation mentioned, can be gathered into four principal points /Press87/ :

i) recognition of the problem
ii) development and synthesis
iii) specifications
iv) survey evaluation

The coordination between the developer of the project and the customer is made by the analyst (who is often called the system analyst), the system engineer, the programmer/analyst and the like. He must distinguish himself by his adaptability, ability to abstraction, communication, knowledge of the environment (hardware, software) in which the project will be realized.

Picture 1a shows the process to accord and define the requirements, (see /Ross85/). This process, called the reader/author cycle, is a part of the method which will be considered in detail later (SADT - Structured Analysis and Design Technique).



Picture 1a "Reader/author" cycle of SADT Method

Besides SADT method several other methods for software analysis and implementation of specifications are developed. This is the requirement engineering. Each method posesses a specific procedure and therefore also a different notation. The methods pass the above mentioned four points and must meet the following three demands:

a) Understable presentation of informational and functional domain in order to analyse the requirements of the problem.
Information flow shows the way of data transformation when the data pass through the system. As the result of studying the information flow the data structure for the processed system is obtained.
The **functional** requirements /Roma85/ describe the dependence of the components upon each other and upon their environment. The whole system, the program or the element of hardware can appear as a component. A conceptional model is the result of fulfilment of functional requirements. Its level of understability should be adapted to the environment for which is determined. *Non-functional* requirements cause a problem because they can exert an essential influence on the complexity of design. Many difficulties cannot be known in the early phases of design (it is difficult to determine e.g. the influence of the required level of software reliability). The software reliability is closely connected with the powerful testing tool. It is difficult to foresee the influence of the "human factor" and the respondence of the finished product to the errors. It is impossible to formalize the process of maintenence completely.

b) Division of the problem into understandable and surveyable partitions.
The division of the problem into easier and more understandable parts is carried out by a vertycal hierarchical decomposition or with a functional decomposition in the horizontal hierarchy.

c) Logical and physical presentation of the system.
The task of the requirements engineering is to clarify **what** should be realized and not **how** it should be done. Logical presentation of the problem forms the base for software system design. The analyst will not include the phisycal presentation until the logical presentation is faultless. It is necessary to define e.g. exceptions in the hardware configuration, system for database management or specificity of the applied operating system.

In the process of analysing the user's requirements a document is formed which is often called *specificatons*. A standard is existing for this kind of document (*The National Bureau of Standards, IEEE Standard No. 830-1984*), but the automated methods of software engineering do not follow it entirelly.
It can be concluded that the task of the requirements engineering is to introduce a systematic and inform notation of the information and functional analysis to solve a software system. According to the software life-cycle, mentioned in the introduction the first two points are met by the requirements engineering. The development trends in the eighties support the development and application of computer aided integrated tools for software engineering : **CASE** Computer Aided Software Engineering. They are overbuilt and the most effective among them meet the development all phases of the software life-cycle. In the following chapters, some tools will be treated in detail.

2. Methods of Software Engineering

2.1. Basic Division of Methods

The basic division is taken from /Press87/. The selected division implies to the phase of system analysis, thus it is undependent from the design method. The latter is chosen according to the obtained functional decomposition :

i) *data flow* oriented methods
ii) *data structure* oriented methods

In literature the third group is called *automated methods*. It is true that in this group the methods were made as automated ones and the philosophy of SADT method (author D.T. Ross) served as the starting-point to design methods from the first two points. Nowadays many methods from the first two mentioned groups are automated in the form of CASE tools. SADT method will be described in detail in a separate, third chapter.
Our description of methods is limited (following our opinion) to most effective and

most popular ones. The criterion of effectiveness is how many phases of a software life-cycle are covered by the method.

2.2. Data Flow Oriented Methods

All methods of this class have in common that they perform the structural analysis by means of *data flow* charts. So they give preference to the data transfer through the system over the information structure on which the system is based. The syntax of data flow charts is taken from /Gane79/ bubble chart and mainly from /DeMa79/.
Tom DeMarco describes very accurately the processing of analysed system until the functional decomposition is reached. The continuation of processing by means of structural design is taken from /Your79/. On the base of literature cited automated tools from the field of software engineering are constructed. The stress is laid on the system analysis by means of data flow charts. Nowadays the following products are availabile on the market :

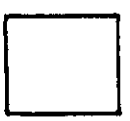
- ANALYST/DESIGNER TOOLKIT; product of the firm Yourdon Press, New York
- HP Teamwork SA/SD/RT; product of the firm Hewlett Packard
- TEK CASE; product of the firm Tektronix
- CASE; product of the firm Microtool, Berlin

SASD (Structured Analysis Structured Design) is the common name for the methods described. It is typical for SASD that it covers all phases of life cycle in the software development ("life-cycle modell" /Kell87/). SASD offers a good survey over the project and enables a team work. An effective functional decomposition from the top-down is the reason for it. As in the of this method a series of understandable tools and those being near to the man is used (a detailed description is given in the following chapters) the method permits immediate correction of the existing faults.
Automated tools always offer an up-to-date version to all members working on the project what is of special importance when individual members of groups do not work in the same place.

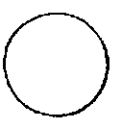
2.2.1. System Analysis of SASD
2.2.1.1. Data Flow Charts

The data flow charts can be used on every level of system abstraction. They consist of four basic building blocks (see Picture 2.2.1.1.a) from which the fundamental system model is constructed first. Then it is decomposed into several more detailed and understandable charts. The basic model is called level Ø1 of the data flow charts (further called DFC).

External entity represents the source or destination of system data. The system data can be the elements of hardware, interactive intervention of the usere or connectoin with other software systems.



Process : this sign is used when data are transformed in such a way that new data are obtained or the existing ones are converted.

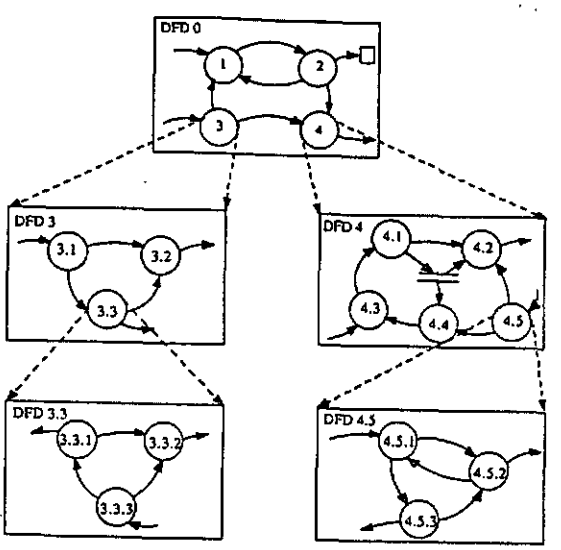


Data flow : serves to connect other basic elements of DFC. The arrow shows the direction of the flow. Each flow should possess its own uniform name.

Data storage ; the symbol means a file or an interface where the data are stored or from where the data are obtained.

Picture 2.2.1.1.a Basic elements of DFC

In order to follow the information flow best when analysis is carried out it is advisable to name the processes at the end. Data flows (information flows) that condition the necessary process form the base and not vice versa. Unfortunately, automated tools require to name the processes immediately. The picture 2.2.1.1.b shows the decomposition into the depth of an imaginary problem by means of DFC.



Picture 2.2.1.1.b Decomposition into the depth by means of DFC

2.2.1.2. Data Dictionary

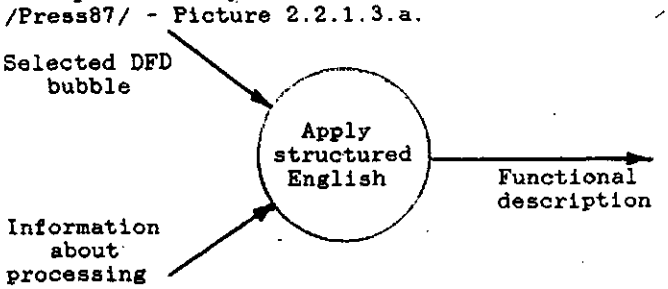
Each arrow in the data flow chart means one or more data elements of a piece of information. The data element /DEMA79/ cannot be decomposed into its components. Therefore it forms a basic element of the data flow.
In the data dictionaryevery data flow should be decomposed into elements. To do this a special notation is needed:

symbol	meaning
=	equals
+	logical and
[¦]	logical or
{ }^{n}	n iterations of bracket contens
( )	optional data
* *	comments

In the design of information systems that are aided by powerful data bases the data dictionary and data store /Gane79/ are needed to model the relational shema in a normal form (Codd's normal forms) /Show87/. The fact that the relation shema of the data base of the system designed can be formed by the structured analysis / Gane79/ essentially contributes that this method can be used. Here we find a linking point with methods that are concentrated on the data structure.
E.g. the automated tool TEK CASE alone forms the data dictionary to such an extent that the necessity to describe the undefined data flows is shown. In the data dictionary the syntax of the record is controlled automatically.

2.2.1.3. Functional Decomposition

With the data flow chart and the data dictionary it is satisfied to the information domain of the problem analysis. The transformations (processes) in the product are described as a functional domain. For this purpose structured natural language, most often " Structured English ", is used. Such clipped language is called Program Design Language - PDL.
Let's show the formulation of the functional decomposition by means the data flow chart /Press87/ - Picture 2.2.1.3.a.



Picture 2.2.1.3.a. The Process of Functional Decomposition

The dictionary of the language used for the description of processes should contain english words in the imperative form, expressions from the data dictionary and reserved words for the description of hidden logic (for this purpose the usage of decision tables is recommended). The sintax of the language used for the description of the processes permits simple sentences such as : PUT, GET, REQUEST, etc. and obligatory includes activity based constructions for the description of sequence, choice and repetitions (IF, CASE, REPEAT, WHILE, etc.).

2.2.2. System Design SASD

System design of the SASD method is taken from the source /Your79/. The transition from the data flow to the first phase of structured design, which is called structured charts, can be made in two ways. The first and most often used way is the **transform analysis** The second way, called the **transaction analysisis** because of its exaggerated formalization less often used.
The nucleous of Transform analysisis that the data flow charts are devided into the afferente part, central or transformation part and efferent part. The basic three components of the structured design are described. In the chapter 2.2.2.4. the example of the transition from the data flow charts into the structured chart is described by means of transform analysis.

The design by means of operation analysis is more effective from the transform analysis only in those cases when the centre of the transformation cannot be uniformly identified.

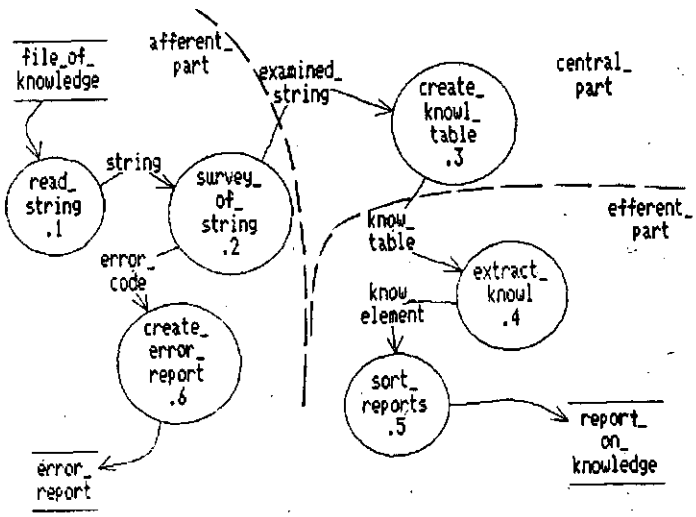
This is the case in split data flow charts (e.g. several output flows). The process that splits the input flow is called transaction centre. Around this centre a suitable structured chart is organized. Let's see the basic tree phases of structured design :

a) Structured Charts
The data flow chart shows a network of processes that needn't be connected in the final program in the same way. The model of the system in the form of software modules is ensured by structured charts. The term 'module' dendes procedures or functions in the target programming language. Thus, the structured chart directly shows the hierarchical structure of software.
More about the syntax of structured charts can be found in the literature mentioned /Your79/. The most important characteristics are described by the words :
- *rectangles* are used to denote modules. External modules, e.g. libraries are denoted by double vertical lines.
- *arrows* connect modules into various hierarchical leves.
- *arrows with circlets* denote communication interfaces between modules; the direction of the arrows shows the direction of the control paths and data items.
- *special symbols* give information on the procedurality of the system; these are iterations, conditional choices between modules, common blocks.

b) Data Dictionary
The data dictionary used in the structured design is *the same* than that used in the structured analysis. Here it is completed with new data obtained from the structured chart.

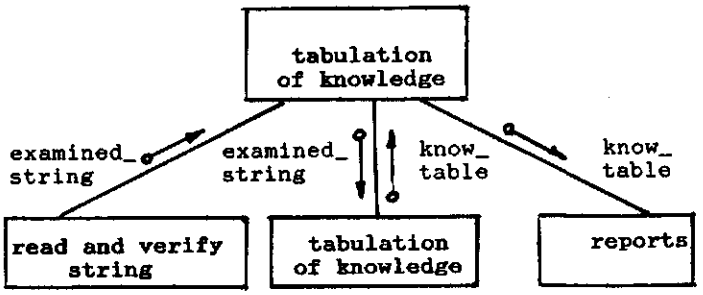
c) Description of Modules
The task of the description of modules is to explain the activity based characteristics of the modules. A pseudocode is obtained which is the direct input into the program implemented in the selected program language.

A simple example to create and follow the knowledge base is given. The picture 2.2.2.a shows the data flow chart of the problem that was devided into three basic parts, according to the *transformation* analysis.



Picture 2.2.2.a Division of the data flow chart into three parts

Each part in the data flow chart has a corresponding functional part in the tree of structured chart. The root of the tree is a new functional component which connects parts (picture 2.2.2.b). The central part of the division of data flow chart can also play this role. The starting hierarchical structure of the problem is obtained. As the processes are not described acurately by the individual functional components a further factorizing within individual divisions in the data flow charts is needed. This is done by top-down design.



Picture 2.2.2.b Structured Chart

2.2.3. Implementation and Testing in SASD

Implementation of the system is based on the top-down procedure and adding (incrementation) of new modules. Incremental procedure demands that each module should be developed and tested separately and then in combination with other modules. In such procedure the member of possible faults is decreased and the testing costs of the system are lowered.
The optimization of the implemented system should be done as late as possible and only in cases when performance tests not uncet the requirements. In this case the optimization should be concentrated on selected modules because it exerts a bad influence on important characteristics such as performance, applicability, reliability and simple maintenance of a well designed system.
In the field of testing and evaluation of software reliability the tendency is to automate the tools with uniform criteria. Additional information on practical access to this problem is found in literature /Rozm87/.

2.2.4. Application Maintenance and Owerbuilding in SASD

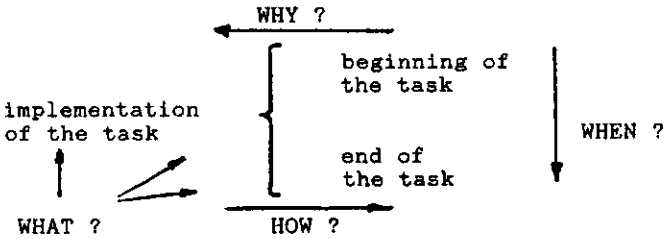
If SASD is formed through all phases into a concluded form, an exhaustive reference-book is given. The documentation on original design is given by data flow charts and structured charts, the definition and organization of data is given by data dictionary; an accurate description of processes and module structure is given, too.
Eventual corrections in testing modules should be stored as a certificate on suitability of the system and as an aid to overbuilding.
The maintenance of the system is simple. The influence of hardware changes or the modified requirements of the user can be thorougly studied in the documentation through all development phases of the system.

2.3. Data Structure Oriented Methods

As we learn from the title of this chapter these methods lay a greater stress on the construction of regular data structure than to the course of data structure. Two methods are mentioned that specifically solve the problem. They have four common points:
i) each method demands that the key information object, that is the entity, and the operators or the process should be determined at the beginning;
ii) the second common point of these methods is that their starting-point is a hierarchical structure of information;
iii) the data structure should be described by basic procedural constructs - these are the sequence, decision and repetition;
iv) these methods contain effective tools to convert the problem from the hierarchical data structure into the suitable software structure.

2.3.1. Warnier - Orr Method

From the theory of Warnier-Orr charts the DSSD method (Data Structured System Development) was developed. Warnier - Orr charts implement a hierarchical analysis of information domain. The global picture presented as multitude is decomposed. It is further decomposed into a series of submultitudes. Everithing is interconnected by basic procedual constructs. Braces are used to devide hierarchical levels. A quick and effective explanation of the meaning of the actions which are possible by Warnier-Orr notation is shown in picture 2.3.1.a /Higg79/.



Picture 2.3.1.a Directions in Warnier-Orr charts

We can see that the reading of the events from top of the chart complex downwards gives a sequence of temporal course of events. In the left direction the explanation of events is 'globalized'. To the right, there is a micro-level of described system.
The first step in the system analysis is to *define the exit of the process*. Then the logical data structure and later the corresponding physical structure are defined. In this phase the entity-relationship model can be formulated. The design of physical model follows from bottom-up but we should not think that Warnier-Orr method is based on botton-up design because the whole logical composition was performed from top down. Practical experiences of design with this method are described in literature /Gyor86/ and /Rese86/.

2.3.2. The Jackson System Development Method

The JSD (Jackson System Development) method /Jack83/ ensures a methodical way to design complex problems. The expected design result is an objective and repeatable system which is independent from the creativity of the designer. The designing process can be adjusted and is heavily influenced by the user.
The *basic idea* of this method is that it is possible to design for the user an interesting part of the real world by means entities and actions. The entities are basic elements that can be recognized by the user. Actions are implemented over the entities described and can be changed from one state to the other (the term 'states' means various phases in the life cycle of entities).
With the definition of entities and actions and with the formulation of structural shema the design phase of a model is concluded.
In the following phase a suitable distribution and definition of individual functions is made. The functions are included into the existing model at a definite moment and under definite circumstances which ensure a correct operation of the functional model.

JSD has a very rigid delimination between design and implementation. The application of the existing hardware and program languages by means of which the designed system will be realized in a classical way or by JSP (Jackson System Programming) will be not included earlier that in the last phase, that is in the phase of implementation.
JSP is also a data-oriented method. It is meant for program design and it is based on the structure of input and output data. The basic ideas of program design by means of JSP are transfered into the design of greater problems and systems. This means that JSD forms an extension of JSP. Everithing what is used in program design according to JSP can be used in system design according to JSD. The common points and differences between the two methods are shown in the following table (Picture 2.3.2.a).

action	JSP	JSD
description of the problem	terms of sequence processes	terms of sequence processes
the model consists of	structured charts	a multitude of unlinked processes for the description of temporal behaviour of entities
description of the problem is concluded by	implement.of uniform proc. structure	adding and impl. of processes which form the problem described
implementation	problem can be described immediatelly	description of the problem is converted into suitable form

Picture 2.3.2.a Comparision between JSD and JSP

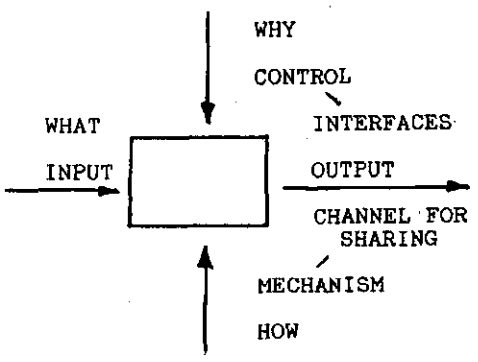
The JSD method is meant to design system in which special attention should be paid to temporal condition. JSD is used to design a wide spectrum of applications based on 'on-line' or 'batch' manner. These applications are taken from the real world and temporal extension is of supreme importance.
From the literature /Came86/ it is seen that this method is most often used in Great Britain where many systems realised by means of JSD are in operation. The method is widely used in Western Europe and less often in North America.

3. Structured Analysis and Design Technique (SADT)

This method was developed in the seventies, in the firm SofTech, Inc. when they searched for an effective tool to describe the software architecture of large systems. Douglas T. Ross /Ross77a/ and /Ross77b/ was the first who wrote on SADT. The latter source deals with the applicability of this method for extremely activity based and data-strong systems. SADT is cited as a reference for methods such as Yourdon, Jackson, Warnier-Orr, Petri nets, The multitude of PDL-s (Program Definition Language), pseudo languages and for typical data-oriented tools Codasyl, Entity Relation Attribute and others.
SADT is very effective in the early and late phases of software life-cycle. The detailed design mekes a bottle neck. SADT can overcome it with the inclusion of poverful languages for the definition of process (PDL). The most effective tool of the SADT method is treated in detail. These are 'box-and-arrow' charts which describe activity based and data aspect of the analized system. These charts are called graphic language of the structured analisys (in SADT, of course).

3.1. Graphic Language of Structured Analysis SADT

The basic guideline in the structured analysis is *understability* and *simplicity*. Therefore this method uses the decomposition from top down to the most simple basic elements. The graphic language is based on the *structured analysis boxes* (further called SAB). In the picture 3.1.a the extensiveness of such basic element is shown. The features of individual SAB are described with ICOM (input, control, output, mechanism) codes. Each SAB has its own number. On every level SAB can be decomposed into hierarchical lower components (the relationship parent - child; like in data flow charts). The decomposition cannot be done on the lowest level (atomic level) because shown system must be entirely understandable.



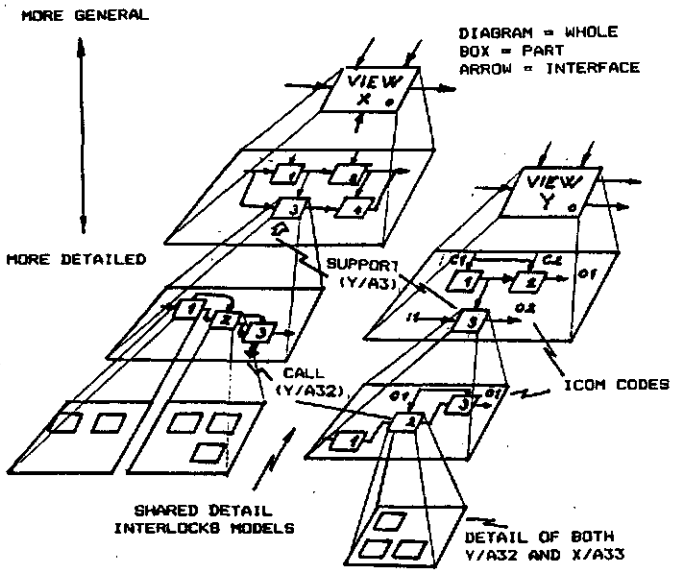
BOX IS
FORM = DATA
OR { DUAL OPPOSITES }
TRANSFORM = ACTIVITY

Picture 3.1.a The extensiveness of SAB

The *model* is called a collection of interconnected charts. the model /Ross85/ defines: M is the model of A if M can be used to answer the questions that are put about A. The quality of the model is determined by the extent of questions and suitability of answers".
Models consist of a multitude of SAB by which the decomposition of system is made. Two kinds of konnections, shown in the picture 3.1.b are known : support connections (support arrow) which are actually global nests in both models and call connections (call arrow) which can be imagined as a call of subprograms, in the terminology of program languages. The model syntax permits also that the recursion is shown.
The graphic form of requirements definition can be formalized by RML (requirements modelling language).

3.2. Applicability of SADT

Besides having influenced the development of very effective methods the SADT idea is successfully used in the development of projects dealing with various fields (formulation of requirements in various fields of multinational societies, design of complex business systems, development of telecomunication systems and unfortunately many complex military programs). It cannot be said on which field is SADT most successful. With



Picture 3.1.b Decomposition and multiple model

combined activity/data models and thorough hierarchical decomposition the method covers a vide sphere of requirements. The author of the method, D.T. Ross, aknowledges that the design and automation of design of detailed analysed system present is bottle-neck. A very accurate requirements definition and excellent documentation made the method popular to design large systems.

4. Conclusion

In this paper an informative presentation of some methods that proved themselves in the development of software systems is given. In spite of the fact that the methods like PSL/PSA (Problem Statement Language/ Problem Statement Analyser), TAGS (Technology for the Automated Generation of Systems), IORL language (Input/Output Requirements Language) and SREM (Software Requirements Engineering Methodology) are not treated we should be aware that these automated methods are widely used in the USA.

Automated methods - each automated method is based on certain formalism. The requirements for automated methods must be very strict because possible errors in early development phases are in exponential increase in the following phases. It is said that those methods are good that with incorrectly presented requirements give no final results and the designer can not be mislead. In such cases the procedures within method should give results and this effect should be increased by automation. The creativity of the designer and his ideas still represent the dominant tool. The method should only orient him correctly, warn against the faults and formulate his ideas.

Trends of CASE tools development

Besides the fact that the tool should be based on a proved formal design method which, supports the development of a program in all its phases, other requirements should be taken into account as well. These are dependent on the environment in which tool will be used.

In general, we can speak about three imortant trends in the development of CASE tools.
First we speak about the suitability of tools for the *real-time* system development. Methods deriven from data flow charts often include the real time extension. For the present state of the development of CASE tools it is tipical that too little attention is paid to the requirements of mathemeatical modelling and simulation. The necessity for immediate and detailed knowledge of hardware makes trouble the real-time systems are interrupt or event driven.
The second trend is to develop such tools that can *model the data base* by means of which the system is aided symultaneously with the development of the system.
The development trend of CASE tools aided by the *artificial intelligence* is gaining in importance. Expert systems can be used to control syntactical and mainly semantical errors of automated tools. In order to fulfill the communication between the designer and the tool as much as possible object - oriented methods are developed. They enable a natural connection between the designed model and the reality described by the model /Borg85/.
An interesting problem arises when we try to design unstructured AI software systems by conventional methods of software engineering. Ideas to solve this problem are found in literature /Part86/. The author warns us that for the present it is impossible to offer an accurate and formalized method to design AI software systems.

LITERATURE :

/Borg85/ A.Borgida,S.J.Greenspan, J.Mylopoulos, "Knowledge Representation as the Basis for Requirements Specification", IEEE Computer, april 1985

/Came86/ J.R. Cameron, "An Overview of JSD", IEEE Software Engineering, 1986/2

/DeMa79/ T. DeMarco, "Structured Analysis and System Specification", Prentice Hall, N.J., 1978

/Gane79/ C.Gane, T.Sarson, " Structured System Analisys ", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979

/Gyor86/ J.Gyorkos,T.Dogsa,I.Rozman," Warnier-Orr Diagram Application Experiences", 8th International Symposium Computer At The University, Cavtat, 1986

/Higg79/ D. A. Higgins, " Program Design and Construction",Prentice-Hall,Inc., New Jersey, 1979

/Jack83/ M.A. Jackson, "System Development", Prentice Hall International, 1983

/Kell87/ G.Kellner (CERN Geneva), Papers from " CASE Seminar Tektronix", Wien, 1987

/Part86/ D.Partridge, " Arifical Intelligence Applications in the future of software engineering", Patridge/Ellis Horwood Ltd., Chichester, 1986

/Porc83/ M.Porcella, P.Freeman,"Ada Methodology Questionnaire Summary", ACM Software Engineering Notes Vol 8 No 1 Jan 1983

/Pres87/ Roger S.Pressman,"Software Engineering ", McGraw-Hill Book Company, New York, 1987

/Rese85/ I.Rozman and others: Research report on Informatyon systems and Artificial Intelligence, C2-0522/796-85, Faculty of Technical Sciences Maribor, 1986

/Roma85/ G.C. Roman, " A Taxonomy of Current Issues in Requirements Engineering ", IEEE Computer, April 1985

/Ross77a/D.T. Ross, "Structured Analysis: A Language for Communicating Ideas",IEEE Trans.Software Eng.,Vol SE-3, Jan 1977

/Ross85/ D.T.Ross,"Applications and Extensions of SADT ", IEEE Computer, April 1985

/Rozm87/ I.Rozman, T.Dogsa,"Empirical software Reliability Models", The 2nd Beijing International Symposium On Computerized Information Retrieval, Beijing China, 12/1987

/Shov87/ P. Shoval, M.Even-Chaime, "Data base shema design: An experimental comparision between normalization and information Analisys", DATA BASE, vol. 18, 1987