MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A164 289

THE DESIGN OF A STANDARD SOFTWARE
DEVELOPMENT METHODOLOGY
'OR THE BRAZILIAN AERONAUTICAL MINISTRY

THESIS

APARECIDO FRANCISCO DE OLIVEIRA
Lt Col, BRAZILIAN AIR FORCE

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

# AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC
ELECTE
FEB 1 4 1986
S D
D

86 2 14 015

AFIT/GCS/ENG/85D-13

DTIC
ELECTE
FEB 1 4 1986
S D
D

THE DESIGN OF A STANDARD SOFTWARE
DEVELOPMENT METHODOLOGY
FOR THE BRAZILIAN AERONAUTICAL MINISTRY

THESIS

APARECIDO FRANCISCO DE OLIVEIRA
Lt Col, BRAZILIAN AIR FORCE

AFIT/GCS/ENG/85D -13

Approved for public release; distribution unlimited.

AFIT/GCS/ENG/85D-13

THE DESIGN OF A STANDARD SOFTWARE DEVELOPMENT METHODOLOGY

FOR THE BRAZILIAN AERONAUTICAL MINISTRY

THESIS

Presented to Faculty of the School of Engineering of the

Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree  of

Master of Science in Computer Systems

Aparecido Francisco de Oliveira, B.S.

Lt Col, BRAZILIAN AIR FORCE

December 1985

Approved for public release; distribution unlimited

QUALITY INSPECTED 3

## Preface

The purpose of this thesis effort is to design and propose a standard software development methodology for the Brazilian Aeronautical Ministry.

This report is not intended to be a final answer, rather it should be taken as a single seed, a first step toward developing a software design methodology for the SIMAER.

To generate such a seed I have had a great deal of help from others. I wish to express my sincere appreciation to Capt Duard S. Woffinden, the advisor of this investigation, for his professional guidance and patience throughout the duration of this effort. I wish also to thank Dr. Gary B. Lamont for his important and worthy refinement.

Special thanks goes to my family, particularly my wife Vera. Her participation helped me to create an internal and isolated environment where I could put all my effort to generate the mentioned seed, while she took care of the external environment. For that she had to follow a hard path learning the language and culture of a foreign, but fortunately receptive country. My thanks to her also for typing most of the report. Without her support the seed could not have been germinated.

<div align="right">Aparecido Francisco de Oliveira</div>

## List of Figures

iii

List of Tables

## Abstract

This thesis proposes a standard software design methodology for the Brazilian Aeronautical Ministry.

The project matched the requirements of the Brazilian Aeronautical Ministry with the software life cycle models, methods, and techniques, which are currently available and most widely utilized.

Based on the analysis, a waterfall model was selected and integrated with some methcds, tools, and techniques, such as Gane's method, SADT, Data Dictionary, etc.

All of these recommendations were included in a proposed regulation for a software development methodology.

# Table of Contents

# THE DESIGN OF A STANDARD SOFTWARE DEVELOPMENT METHODOLOGY
# FOR THE BRAZILIAN AERONAUTICAL MINISTRY

## I.  Introduction

### 1.1  Background

The author worked successively as Chief of the Planning
Section of the Brazilian Aeronautical Ministry (MAer)
Information System, Chief of the Planning Section of the
Brasilia Data Processing Center, and finally as head of this
organization from 1980 to 1983. These positions offered him
the opportunity to become familiar with the many existing
automated data systems (ADS) in the MAer, and the opportunity
to follow them throughout their life cycles. During this time
he was impressed with both the lack of a standard methodology
for system development and the lack of a consistent
management approach to the software life cycle.

Generally speaking, few used modern, productivity-
increasing techniques such as structured design, top-down
analysis, or review sessions[26]. This condition has caused
several problems such as high system development cost,
interface difficulties between ADS designed by different
organizations, and constant training needs to name a few.

This situation is a matter of great concern for the
Centro de Informatica e Estatistica da Aeronautica (CINFE),
head of the Information System, as it is this organization

that has the overall responsibility for software development in the MAer[30]. A standard software life cycle and methodology for the MAer should reduce the systems development cost.

## 1.2  Problem Definition

Many authors recognize that there is an ongoing software crisis which, in essence, is that:

"it is much more difficult to build software systems than our intuition tell us it should be"[7].

Brazil and especially the MAer are not immune to this crisis, and the symptoms, enumerated by Fisher[7] and listed below, are also manifested in the MAer's system development:

1. Responsivenness. Computer-based systems often do not meet users needs.

2. Reliability. Software often fails.

3. Timeliness. Software is often late and frequently delivered with less-than-promised capability.

4. Transportability. Software from one system is seldom used in another, even when similar functions are required.

5. Efficiency. Software development efforts do not make good use of the resources involved (processing time and memory space).

6. Modifiability. Software maintenance is complex, costly, and error prone.

7. Cost. Software costs are seldom predictable and are often perceived as excessive.

A typical example from the Author's experience within the MAer, is the case of a system development where the manual activities were discontinued before the automated system was validated and accepted. As a result the automated system was not responsive causing some disastrous consequences.

Besides suffering the same worldwide common problems listed above, Brazil faces its own crisis, an economic one, evident in a high public deficit, which in turn imposes severe budgetary constraints. Also,it would be wise for Brazil, being a country in development, not to repeat the same errors experienced by more developed countries which have already passed through this process and found some of the answers to reduce the software crisis[7].

Embedded in this scenario is the MAer which currently faces high hardware and software demands to support both management as well operational systems with a limited amount of resources[30].

The problem is the high development cost, the lack of standards for software development in which modern techniques are used, associated with the need for making use of the available resources in as efficient and effective way as possible.

## 1.3 Scope

In order to solve the previous stated problem, some standards will be proposed. Such standards will cover areas such as: methodology, software life cycle, methods (including tools, techniques, modern programming practices), maintenance, High Order Languages (HOL), documentation, development management issues (personnel allocation, review sessions, development control), and cost.

A survey on software development in the MAer will be conducted, and based on its results and the author's past experience, SIMAER's requirements for software design will be defined, and objectives for standards to meet those requirements will be detailed. Also, parameters and criteria to evaluate and test the degree to which the tools, techniques, and  methodology recommended by the standard, support the requirements will be established. This will allow several alternatives to be examined and the recommended choices justified.

Finally an implementation plan covering cost, milestones, and training will be elaborated.

## 1.4 General Support

This project was developed using the author's previous experience, acquired background knowledge, academic support,

current literature reviews, advisor's orientation, and the MAer requirements.

## 1.5  Approach

In order to become more acquainted with the current software development situation in the MAer, and to determine its requirements, an overview of the MAer Information System Structure is presented and a survey of its organization related to software development was performed. Both, the MAer Information System Structure and the survey findings are presented in Chapter II.

Chapter III, Current Available Methodologies, consists of a literature review on all of the best known methods proposed by the academicians and software development professionals. Also, some methodologies used by American and European organizations were studied and compared. Finally an analysis was performed trying to highlight the methods' characteristics that best support the MAer requirements.

Next, in Chapter IV, a literature review on the available automated software development tools is discussed. Following this review, a preliminary study, looking for a future implementation of such tools in the MAer is performed. The study covers the MAer's requirements, as well the available resources.

In Chapter V, a software life cycle is designed and standards are proposed, based on the requirements determined

in Chapter II and on the available methods' characteristics
pointed out in Chapter III. Also an implementation plan,
including milestones, cost estimation, effort in training,
etc, is presented.

Finally, Chapter VI, summarizes the research findings
and states recommendations and conclusions.

## II. The Environment

### 2.1 Introduction

The design of a standard software life cycle and the selection of tools, techniques, and methods to be used during this cycle must support the requirements of the organization where such a standard life cycle will be observed.

This chapter presents an overview of the software system development environment in the MAer. The most common types of applications developed there, which tools are currently employed, and how the system's development is managed. All these factors together will compose the scenario which defines the requirements needed to design the proposed standard software life cycle.

### 2.2 Structure of the Brazilian Aeronautical Ministry

As stated in the Brazilian Constitution[33], the Aeronautical Ministry is the organization responsible for establishing the national aerospace policy and controlling the overall aeronautical activities. In war time its main objective will be to achieve and maintain air superiority over the Brazilian territory. This task is to be accomplished by the MAer's armed branch - the Brazilian Air Force. The MAer, as shown in figure 1, is composed of: (1) the Aeronautical Minister Office (GABAER); (2) the Aeronautical Staff(EMAER), the organization responsible for the planning and consulting activities of the Aeronautical Minister; (3)

7

the Civil Aviation Department (DAC), which is responsible for planning and controlling the civil aviation activities; (4) the Research and Development Department (DEPED) whose tasks are research, development, and fostering the industrial activities in the aerospace field; (5) the Personnel General Command (COMGEP), which is responsible for the management of the manpower activities; (6) the Training Department organization (DEPENS) responsible for the training activities within the MAer; (7) the Air General Command (COMGAR) organization whose mission is to perform the combat activities, and finally (8) the General Logistic Command (COMGAP), which has the responsibility of supporting all the MAer activities, mainly those related to flight.

```
                    ┌──────────────────┐
                    │   AERONAUTICAL   │
                    │     MINISTER     │
                    └──────────────────┘
                              │
                 ┌────────────┴──────┐
                 │              ┌──────────────┐
                 │              │    EMAER     │
                 │              └──────────────┘
     ┌───────┬───┴───┬───────┬──────┬────────┬───────┐
 ┌───────┐ ┌───────┐ ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐
 │  DAC  │ │ DEPED │ │ DEPENS │ │ COMGAR │ │ COMGEP │ │ COMGAP │
 └───────┘ └───────┘ └────────┘ └────────┘ └────────┘ └────────┘
```
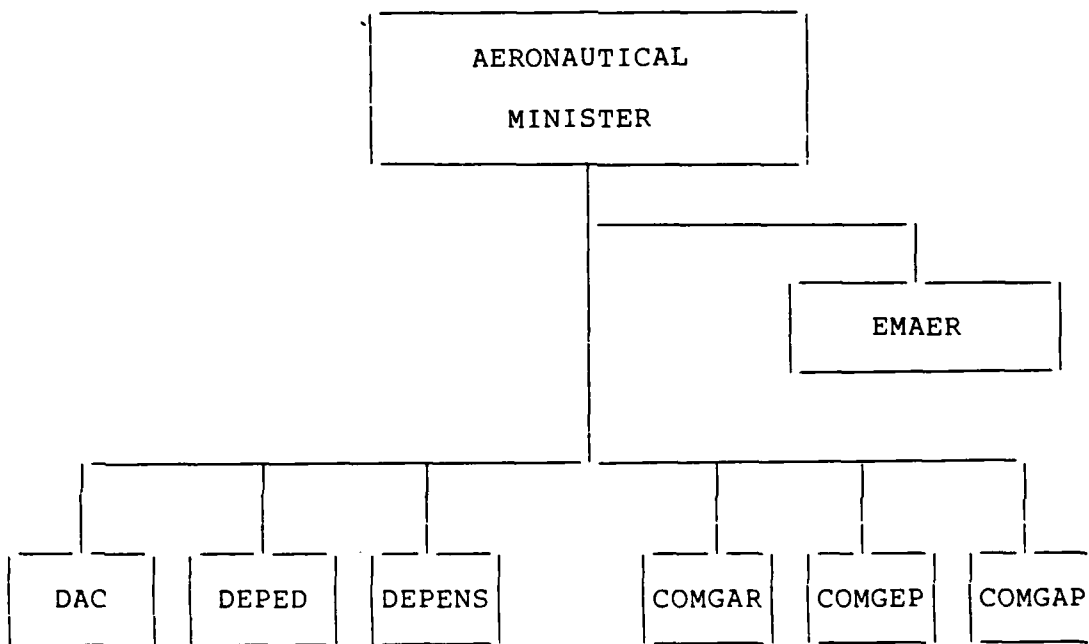
Figure 1 - The Brazilian Aeronautical Ministry

The automatic data processing activities in the MAer are organized as a system called SISTEMA DE INFORMATICA DO MINISTERIO DA AERONAUTICA (SIMAER)(Fig. 2) whose focal point is the CINFE. CINFE responsibilities include: the planning, coordination, and control of the data processing activities as well as support of system's members. The CINFE is subordinated to the COMGAP and headed by a Brazilian Air Force Colonel.
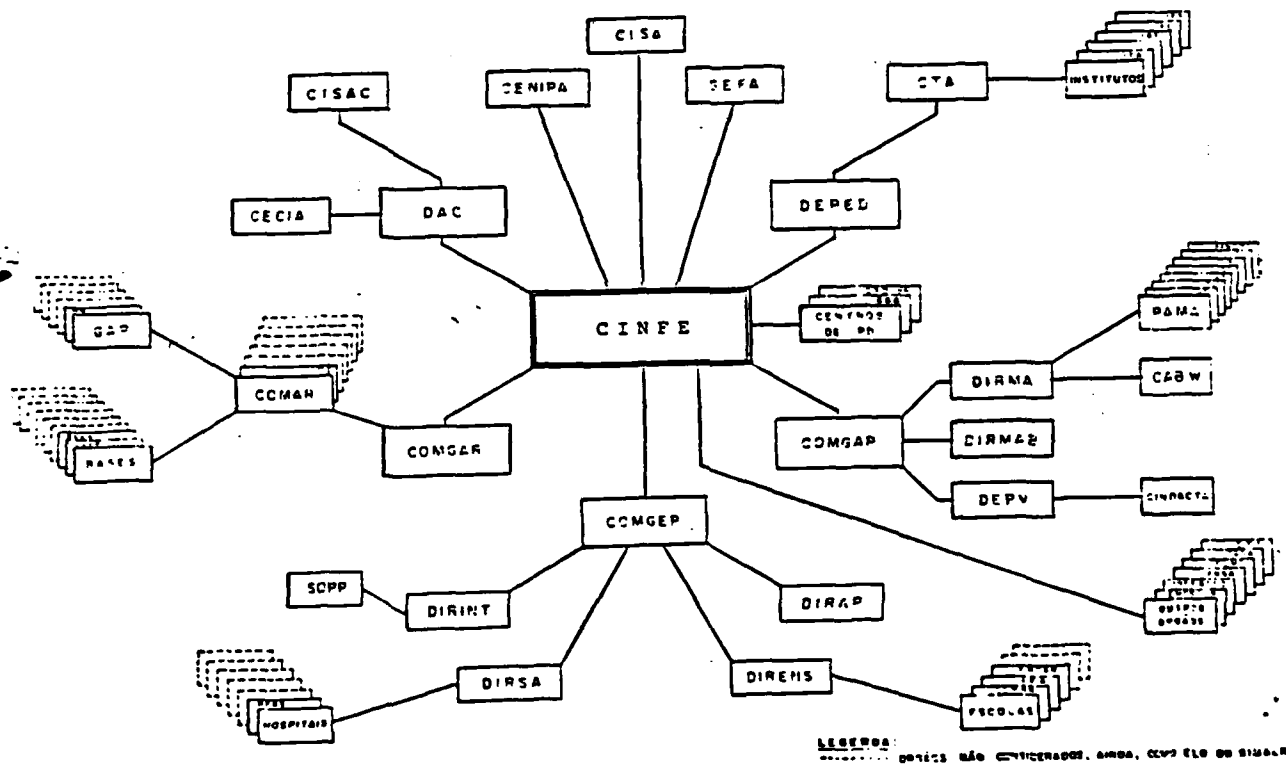


Figure 2 - The Aeronautical Ministry Information System (SIMAER)

## 2.3 Current Situation on Software Development

### 2.3.1 SIMAER's Overview

The SIMAER is currently undergoing a great expansion program. The MAer is starting to become involved in the activities of software acquisition for embedded systems, and there is a great demand for software design, mainly to support management and operational activities.

There is also a high demand for microcomputer acquisitions. This demand and its supply will require further studies to establish a policy on end-users development. Such studies should take into account several considerations and constraints related to user-developed systems. Davis[12] suggests the following disadvantages and advantages of user-developed systems: .

1. Disadvantages

   a. Low discipline of users

      Information Systems personnel generally accept and follow procedures and rules (a development discipline) that represent a codification of experience in application development. Users as new developers do not easily adopt this development discipline; they must obtain it through training, experience, and policies and procedures.

10

b. The risk from encouraging private information systems

The complete information system of an organization is composed of systems that are formal or informal and public or private. User developed systems, by promoting private formal systems, encourage information hiding by individuals. It is also difficult to transfer private systems to new persons taking over a position.

2. Advantages

a. Relieves shortage of system development personnel

A common user complaint is that there are not enough analysts and programmers to keep up with the demand for new systems. One of the alternative solutions to this problem is to transfer some of the development function to the users.

b. Eliminates the problem of information requirements determination by information systems personnel

One of the major problems in information systems development is the need to elicit a complete and correct set of requirements. Various methodologies have been proposed but it still

11

remains a difficult process. The problem is
made more difficult because the analyst is an
outsider who must communicate with a user
eliciting the requirements. Having users
develop their own systems eliminates the
problems of inadequate communication between
analyst and user.

c. Transfer the information system implementation
process to users

Poor implementation is one of the major reasons
systems are not utilized. Difficulties arise
from the interaction of the analysts and the
nontechnical users who are providing
requirements for the system. Users may develop
less sophisticated systems when they do their
own design and development, but they are more
likely to use them.

Other considerations include the implementation of
distributed processing, with its own implications, such as
the design of databases and efficient networks.

All of the considerations and constraints can only be
balanced by establishing appropriate policies.

Of course this high microcomputer demand is not a
Brazilian localized phenomena. According to a recent survey,
published in the Government Computer News[24], 38,000
microcomputers were purchased in fiscal year 1984, by US

governement agencies, a 450 percent increase over the previous fiscal year's acquisition. Of those 38,000 microcomputers, Department of Defense reported the most buys, 17,419, of which 4,009 were bought by the Air Force.

Currently SIMAER's, manpower and the financial resources are insufficient to face all the mission needs[30]. Thus there is an urgent requirement for making the use of existing resources as effective and efficient as they can be. One of the several ways to achieve this efficiency and effectiveness is by the establishing of standard policies with the following objectives:

1. Reducing software errors.

2. Requiring the SIMAER's professionals to adhere to the accepted principles of software engineering.

3. Providing a software design tutorial tool for novices.

4. Reducing the training cost.

5. Increasing the SIMAER's professional expertise in some specific tools and techniques for system's development.

6. Allowing for a more efficient use of the available resources.

7. Emphasizing the production of complete documentation.

8. Enforcing the employing of current managerial procedures for planning, development and control.

A well-known explanation of information system organizational change is the so-called Nolan's Stage Theory[25]. This model identifies four stages of information system organizations growth:

1. Initiation - early use of computers by small numbers of users to meet basic organizational needs, with decentralized control and minimal planning.

2. Expansion (or contagion) - experimentation with and adoption of computers by many users, proliferation of applications, and crises due to rapid rise in costs.

3. Formalization (or control) - organizational controls established to contain growth in use and apply cost-effectiveness criteria. Centralization and controls often prevent attainment of potential benefits.

4. Maturity (or integration) - integration of applications. Controls are adjusted. Planning is well established. Alignment of information system to organization.

The MAer has three distinct primary data processing areas: the Rio de Janeiro area that revolves around the Centro de Computação de Aeronautica do Rio de Janeiro-CCA RJ, the MAer's first data processing center, in which the most experienced people are located. This center may be reaching Formalization, the fourth development stage of the Nolan's stage model[25]. Besides several conventional files

14

applications, two databases are under final development in this Center. This organization is also a source of training for many MAer ADP professionals in the Rio de Janeiro area. Next is the Brasilia area that revolves around the Centro de Computação de Aeronautica de Brasilia-CCA, a new center that is still in the Expansion phase of the Nolan's stage model[25]. Finally there is the Instituto Tecnológico da Aeronautica-ITA, located in São José dos Campos, a similar organization to the United States Air Force Institute of Technology, where scientific and training applications are developed. It is at the same stage of the development as the Rio area.

### 2.3.2 The Survey

### 2.3.2.1 Introduction

As stated in Chapter I, and based on past experience, it seemed that the SIMAER's organization did not any use standard software methodology, and that just a few used some of the modern programming tools and techniques. In order to confirm those impressions, and get a appreciation of software developement and management, a survey of SIMAER's organizations was done. Another purpose of the survey was to gather suggestions for a future standard software life cycle design and a standard methodology selection, while allowing the ADP professionals participation. Thus a somewhat more valid product could be delivered.

The detailed survey, including procedures,

15

questionnaire, detailed answers, comments and conclusions is contained in appendix A of this report. A summary follows.

### 2.3.2.2 Summary of the Findings

The SIMAER has not established a standard software life cycle to be followed by its ADP professionals. The ones used individually do not consider the necessary and currently accepted reviews[5] which are performed at the end of each phase.

None of the SIMAER's organizations has developed or formally adopted a standard methodology, in which modern tools and techniques for system's analysis or software design are employed, to be followed by their ADP professionals during system's development. However, some professionals located in Rio do use, informally, some of the modern tools and techniques supported by Chris Gane[18] and the Jackson's methods[22].

While most of the SIMAER's professionals agree that a standard methodology would be helpful and cost-saving, a few showed some concern about having a standard methodology, arguing that the heterogenous training and the diversity of the applications will not make it practical! The survey showed that in a small but varied number of organizations in Rio a standard almost existed.

The heterogenous training can be considered one more reason for having an standard. Common training and practice

16

would help level off the degree of experience of all people involved in system development. Peters[28] stated in the conclusion of a survey which he did in 1976, that the use of some sort of method is likely to be better than using none at all and that the use of a defined software methodology that includes documentation standards is definitely increasing[28]. Also Davis[12] states that many installations have adopted a single development methodology to be used for all applications, giving training, supervision, and quality assurance benefits for the organizations.

There is not at any level, a MAer documentation standard establishing the minimun documentation that should be produced during a system development. The decision on what to document and how to do it is responsability of each system manager.

Also there is no standard graphical representation nor any regulation establishing which one should be used in each phase, however some professionals are familiar with and use Logical Data Flow[18], Jackson's method[22], HIPO[20], Chen-Entity[10] and Structure Charts[33].

Concerning system development management, the most common teaming approach is the classical method, and the most common control tool is the status report. This results in very little planning, and mainly just control. Amazingly the least used tool for planning and control was the PERT/CPM[5], however, at the same time, it was the technique

17

most suggested to be adopted[26]. Most of the professionals know but do not use any formal type of review technique.

For programming the most commonly used languages are: COBOL, FORTRAN, and PL1. The extensive use of languages that facilitate the use of modern programming practices is not enforced. When questioned, a few specialists showed some concern about the establishement of standard HOLs for the MAer, arguing that they could not apply to every application, and would also limit the professionals knowledge. Any standard should establish a number of languages sufficient enough to cover several types of applications. As far the limitation of knowledge no standard is supposed to be static, not allowing for modification to implement improvements in the field.

In general, many respondents suggested that the standards be established at top level leaving the details for each organization.

In conclusion, it could be seen that there is a lack of effective top and lower level management, manifested by the nonexistence of regulations for system development. The need for a standard software life cycle, tools, techniques, and methodology are highlighted as important issues.

## 2.4 SIMAER'S Requirements Definition

Based on past experience as well as in what was found by the survey, the SIMAER's requirements can be defined by:

18

1. A high system development demand for applications with the following characteristics:

   a. A large amount of applications are to support management information systems, which will replace manual systems.

   b. Many applications will be designed to run on microcomputers.

2. Two databases are under development and a few more are needed.

3. There is a need to develop Decision Support Systems.

4. The computer resources come from varied sources, which implies different technical characteristics.

5. There are a lot of novices, which implies that besides standards there is a need for some sort of tutorial material, and personnel training support.

6. Some of the modern tools and methodologies are already known and used by some SIMAER's professionals.

7. Embedded and real-time systems for military applications are gaining more and more attention from the MAer.

8. Software development management needs to be emphasized and enforced through the establishment of regulations and standards.

## III.  Current Available Methodologies

### 3.1  Introduction

This chapter is based on an extensive literature review of all current software life cycle models, best known methods, tools, and techniques, both proposed by academicians, as well as used by some organizations similar to the SIMAER. It highlights the characteristics, suitabilities, strengths, and weaknesses of these models, methods, tools, and techniques in order to allow the selection of the ones that are best suited for the SIMAER's needs. The presentation in this chapter is not intended to be a tutorial, rather, it is intended as an overview.

### 3.2  Literature

### 3.2.1 The Software Life Cycle

A common mechanism for planning, scheduling and controlling engineering projects is to subdivide the development process into several steps or phases[28]. One such mechanism used by ADP professionals is the so-called Software Life Cycle, which is defined as the period of time that starts when a software product is conceived and ends when the product is no longer available for use[21].

There are basically two software life cycle models, the Waterfall Model[5][28] and the Prototyping Model[28][12]. A description of these two models and two variations of them, suggested by Peters[28] and others follows.

20

### 3.2.1.1 The Waterfall Model

The Waterfall Model (Fig. 3) consists of the neat,



```
┌─────────────┐
│DATA         │
│COLLECTION   │
│(RAW DATA)   │
└─────────────┘
      ┌─────────────┐
      │SYSTEMS      │
      │ANALYSIS     │
      └─────────────┘
            ┌─────────────┐
            │REQUIREMENTS │
            │DEFINITION   │
            └─────────────┘
                  ┌─────────────────┐
                  │PRELIMINARY      │
                  │DESIGN           │
                  │(LOGICAL DESIGN) │
                  └─────────────────┘
                        ┌─────────────────┐
                        │DETAILED         │
                        │DESIGN           │
                        │(PHYSICAL DESIGN)│
                        └─────────────────┘
                              ┌─────────────────┐
                              │CODE             │
                              │(IMPLEMENTATION) │
                              └─────────────────┘
                                    ┌──────────┐
                                    │TEST      │
                                    └──────────┘
                                          ┌──────────────┐
                                          │INSTALLATION  │
                                          └──────────────┘
```
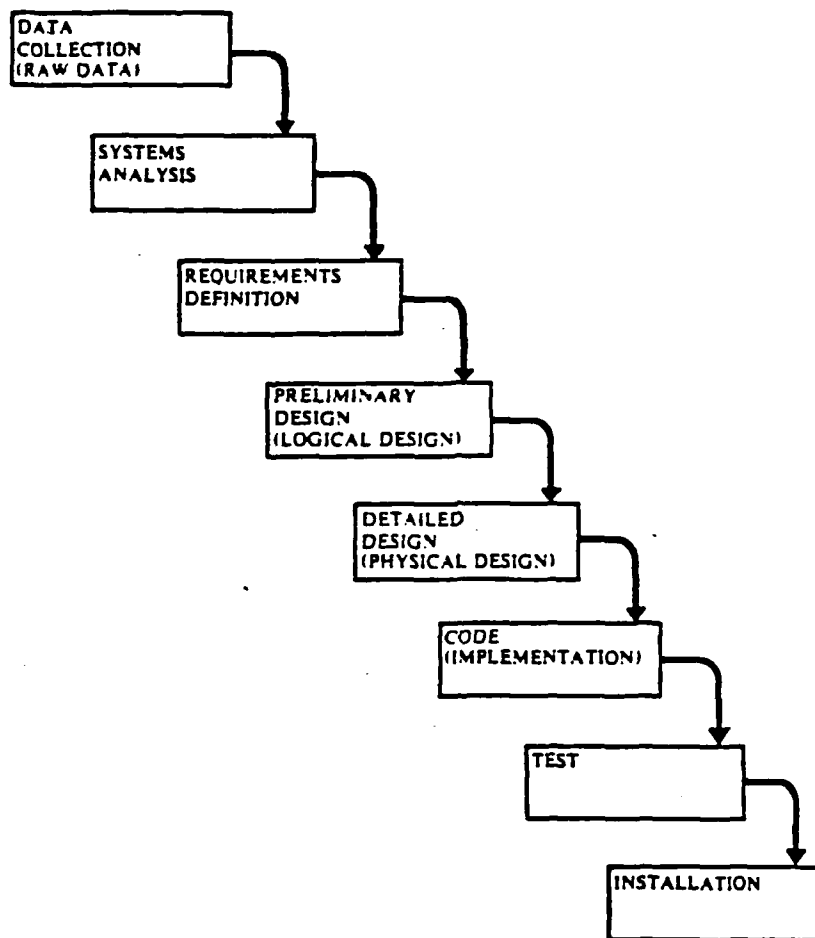
Figure 3 - The Waterfall Model

concise, and logical ordering of the series of obvious steps that must occur in order to obtain a product[28]. This is the most commonly used method and should be applied to large and highly structured application systems[12].

The phases in the software life cycle are described

21

differently by different writers, but the differences are
primarily in the amount of detail and manner of
organization[12]. Peters [28]divides the software life
cycle(Figure 3) in the following phases:

1. Systems Analysis: This is sometimes referred to as the
   data collection phase. It is here that the problem is
   described, data gathered with which to gauge its
   magnitude, and a fundamental understanding of the
   problem obtained.

2. Requirements definitions: Also sometimes referred to as
   system specification, it involves the formalization of
   the data gathered during analysis into a concise,
   clear, and consistent statement of what the system is
   to do. As we are going to see later, this becomes one
   of the most important phases in the process. There is a
   great chance of errors being introduced in the system
   at this point through a lack of communication and the
   consequent difficulties of understanding between the
   user and analyst/designer about what the system is
   suppose to do.

3. Preliminary design:  This phase  produces a  high-level
   design  or  system  model  showing  how the system will
   accomplish its task, but without sufficient detail to
   implement it.

4. Detailed design: This is the refinement of the
   preliminary design to the point at which implementation

can begin.

5. Coding: This is the implementation of the refined
   design with the idiosyncracies of the programming
   language, operating system environment, and external
   (human and hardware) interfaces taken into account.

6. Testing: The ensuring that certain classes of errors do
   not exist within the system and that some predefined
   confidence in the system has been attained, is
   accomplished in this phase.

7. Installation: The actual introduction of the finished
   system into its intended environment, with continuing
   maintenance as required.

Boehm[5] and Davis[12] added to the Waterfall model a
Feasibility Phase (Fig. 4), in which an evaluation of
feasibility and cost-benefit of the proposed application is
done, and also an Operation and Maintenance Phase in which
day-to-day operation, modification and maintenance are
performed. It is worthwhile to mention that Boehm also
considered Verification and Validation activities at the
end of each phase. The objective of these evaluations is to
eliminate as many problems as possible in the products of
that phase. Davis[12] also points out that at the completion
of each phase, formal approval sign-offs are required from
the users as well as from the manager of project development,
and that each phase results in formal documentation.
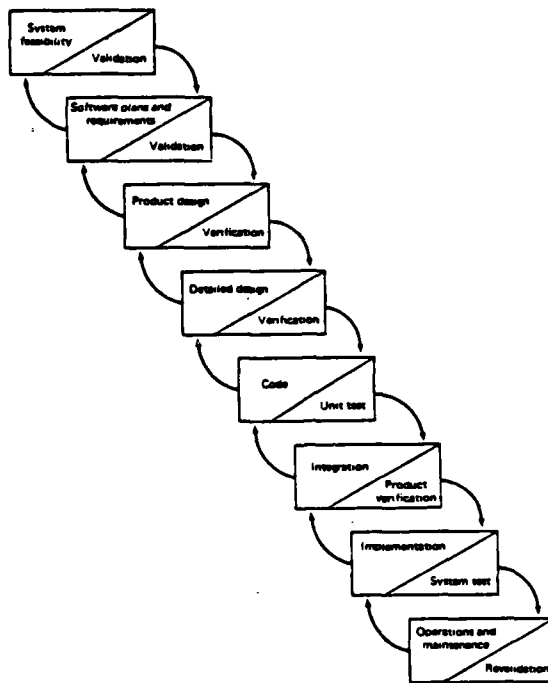
Figure 4 - Boehm's Version of the Waterfall Model

The system development usually follows an iterative

strategy since as pointed out by Peters[28] we may never be

able to stop discovering some new subtlely about the problem

or the approach to a solution in the requirements analysis

and development phases.

Experiences, not all of them positive, with the

waterfall model have indicated that considerable time and

much money have been spent in the coding, testing, and

maintenance phase to correct errors that were created during

the requirements definitions  and  design phases. This has

been figuratively represented by the so-called Error

Avalanche(Figure 5)[37].

24
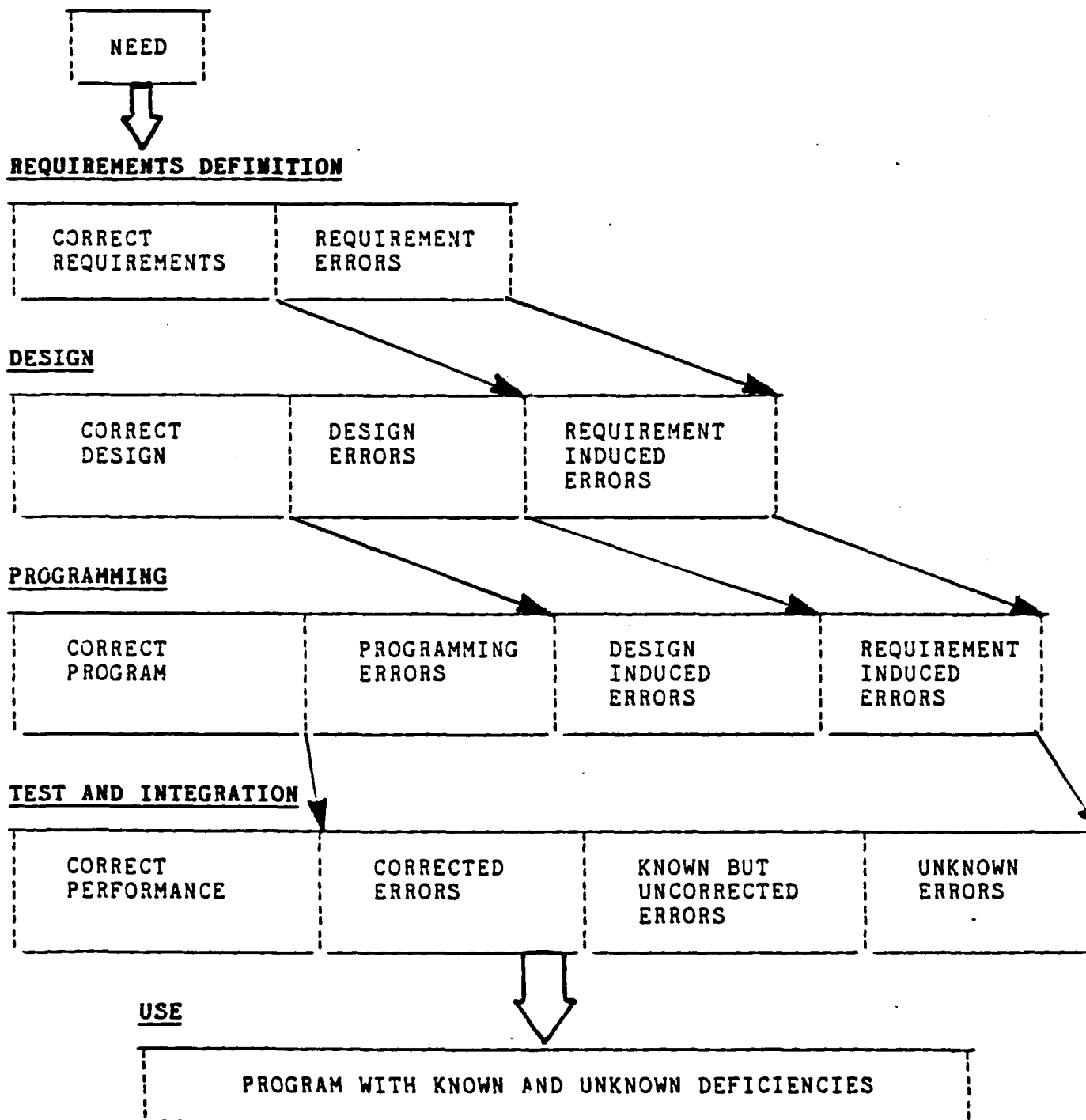
Figure 5 - The Error Avalanche

25

Studies conducted by Boehm[5], and represented in figure 6
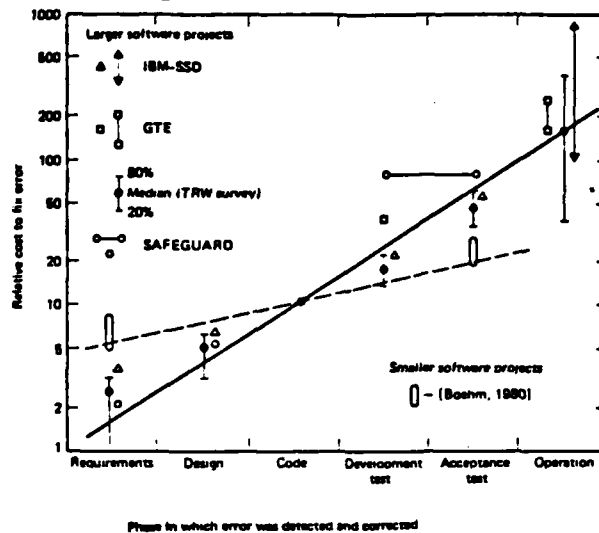


Figure 6 - Increase in Cost-to-Fix or Change Software
Throughout the Life Cycle

confirm that the later the errors are found the more costly
they are to fix.

The above aspects, along with the need of improving the
users/analyst communication and the necessity of increasing
the users' participation, led to the development of newer
models, such as the Logicalized Software Development Cycle,
the Structured Life Cycle and the Prototype Life Cycle[28].

3.2.1.2 The Logicalized Software Development Cycle

This model, using top-down decomposition and abstract
software design models, seeks to separate conceptual or
practical issues, identifying the clear dichotomy existing
between the logical design and the physical design. Although
this model has some apparent advantages over earlier models
by reducing the complexity for the designer, the problem of
assuring that the design fits the stated requirements still

remains. Since requirements are stated as what the system will do, and the design is how it will do it, it is difficult for these two to be compared. This task has been accomplished by using the life cycle model of structured analysis suggested by DeMarco[13].

### 3.2.1.3 The Structured Life Cycle Model

In this model, requirements definitions and logical design are linked or integrated into a single phase called structured analysis. Closer customer or user participation is also employed to ensure that the results of the analysis do reflect the customer's needs based on the present situation (current physical model), its abstract equivalent (current logical model), and the new system or solution model (new logical model). Some advantages of this model are:

1. Enhanced customer/contractor communication: This is accomplished by having the customer and contractor communities work together as a team, and by using written and graphic tools that they both understand.

2. Enhanced analyst/designer communication: DeMarco[13] proposes using the same notation in analysis as in design. This reduces information loss between the two camps of developers. The situation is aided even more when one group does both analysis and design, particularly, if group members maintain the mental discipline needed between phases.

27

3. Better overall quality in both analysis and design
   phases: The goals in each phase are limited,
   realistic, and objectively measurable. Although
   analysis or design could be refined ad infinitum,
   there is at least a minimum set of gcals that must be
   present[36]. Since the objective is defined and
   realistic, people work in a much more productive
   manner than when they have no way of knowing when they
   are done.

However some difficulties still remain. For example, it
is difficult for customers to visualize what the software
system will be like. This particular problem was addressed by
the Prototype Life Cycle Model[28].

3.2.1.4 The Prototype Life Cycle Model

Prototyping (Figure 7)[27] is used when requirements are
difficult to specify in advance or when requirements may
change significantly during development. Contrasting with the
waterfall model, prototyping should be applied to small and
less structured systems where a high degree of uncertainty is
present[12]. The basic notion is to provide the user with
some feedback early in the development cycle on what the
final system will be. Some characteristics of the prototype
model are:
   1. Emphasis is placed on speed of building rather than
      efficiency of operation.
   2. The user, rather than the designer, decides when

28

changes are necessary and thus controls the overall development time.

The greatest danger involved in employing this model is the user tendency to accept the prototype as the final product instead of basing acceptance on the fully specified design.

Step 1 — Identify the basic information requirements
- Basic needs
- Scope of application
- Estimated costs

Step 2 — Develop the initial prototype

Initial prototype

Use the prototype system and refine requirements

Is the user/designer satisfied?

yes — Operational prototype
- Use prototype as specifications for application development
- Use prototype as application

no — Working prototype — Revise and enhance prototype — Enhanced working prototype

Figure 7 - The Prototype Life Cycle Model

### 3.2.1.5 The Alavi Expirement

Maryam Alavi[1] presents the results of a two-phased research project comparing the prototyping approach with the more traditional life cycle approach. He finds that prototyping facilitates communication between users and designers during the design process, however, his findings also indicate that designers who used prototyping experienced difficulties in managing and controlling the design process.

### 3.2.1.6 Selection of a Software Life Cycle Model

In this section several software life cycle model approaches are addressed, their phases described, and the characteristics of each highlighted. According to the academicians, the waterfall model can be applied to a broad spectrum of applications. Its strengths resides in it being a powerful mechanism for management through the phases. Additionally the milestones and deliverables can be established for each phase. It was also found, as will be seen later, that the waterfall model is the most commonly used approach by other air forces.

Its weaknesses are the lack of flexibility and the users difficulties in understanding what the system will be like after it is designed. Despite those flaws, it appears that the combination of the waterfall model with the structured techniques, the so-called Structured Life Cycle Model, combines the strengths of a traditional, and indeed a well

30

known and comprehensive approach[28], with a modern software design technique making this combination the most powerful system development tool of those evaluated thus far.

Comparing the models with the SIMAER's requirements, it is possible to conclude that the Structured Life Cycle Model is the best approach. Some of SIMAER's requirements for which the Structured Life Cycle is well suited are:

1. Enforcement of software development management - The Structured Software Life Cycle model is considered a powerful mechanism for management through the phases

2. Varied types of applications are needed - The Structured Software Life Cycle model can be applied to a broad spectrum of applications[28].

Having decided for the Structured Life Cycle Model as the suggested standard to the SIMAER software development, in the next section, the complementary issues, methods, tools and techniques, to turn a model into a methodology will be covered.

### 3.2.2 Methods, Tools, and Techniques

A method is a regular and systematic way of accomplishing something, tools and techniques are instruments which help to implement methods, and a methodology is a set of methods and tools combined with an overriding management procedure[37].

In the later part of the 1960s, software engineering came of age with the realization that discipline was the

31

key to success in the software development[28]. Earlier, it was suggested that the software developer is often like an artist. However, Booch[7] points that when such artistry is relied upon in an engineering environment, the results are often not good. Booch goes further suggesting that organizations should adopt modern software methodologies supported by a high order language.

The objective of this section is to present some of the available methods, tools, and techniques described by Peters[28], Fairley[17], Chris Gane[18], and others.

### 3.2.2.1 System Architecture Techniques

1. Concept

The primary goal of an architectural representation scheme is to portray the software system in a such a way as to communicate these categories of information:

    a. Philosophical - that is the information providing the basis for the particular system organization chosen.

    b. Organizational - that is information about the structural properties of the system.

    c. Contractual - information about the ability of the system to meet or exceed the legal obligations of the contract with the user.

2. Approach

The goal of this class of representation techniques is reached by depicting major portions of functions of the

system, and their relationships to one another.

3. Tools

Peters[28] presents only two schemes for this category
of representation, the Leighton diagrams and HIPO (Hierarchy,
plus Input, Process, Output). Considering that the HIPO
diagram is the most known, used, and comprehensive of the
two representations, containing the information contained in
Leighton, and adding other important information (e.g. flow
of data and implied sequence of operation), only HIPO will be
discussed here.

a. HIPO

After the acceptance of the importance of
decomposition and hierarchical structures, IBM introduced
HIPO[20]. Colter[11] considers HIPO as a transition analysis
tool, standing between the traditional and the structured
methods.

(1). Concept

This software design representation scheme is
based on the view that software systems can be modeled as
processes with distinct inputs and outputs. It allows for
top-down decomposition, as well for data flow composition.

(2). Notation

HIPO is composed of two packages. The primary
package consists of the VTOC (Visual Table of Contents),
whose example is shown in figure 8, and a set of IPO (Input,
Process, and Output) diagrams (Fig. 9). The VTOC contains a

33

hierarchical system representation from a functional perspective. The IPO diagrams provide detail on individual functions in the system. HIPO is also supported by a set of appended materials, such as report and file layouts, input details, etc.
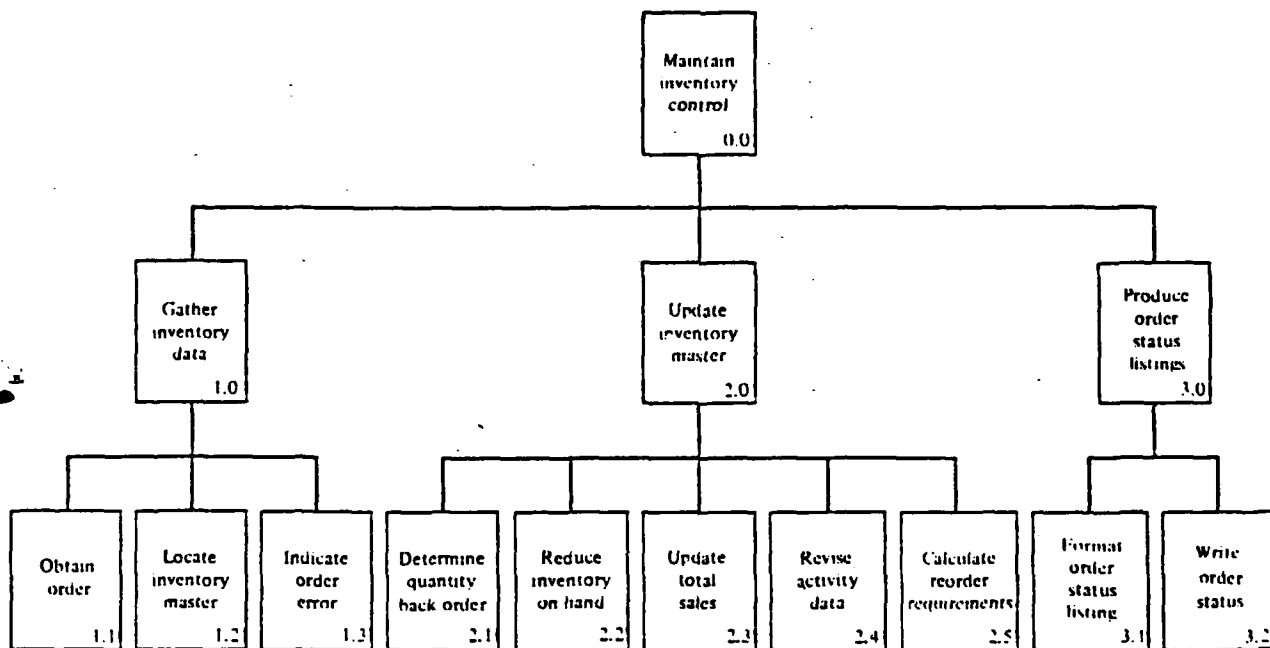


Figure 8 - HIPO Visual Table of Contents Diagram
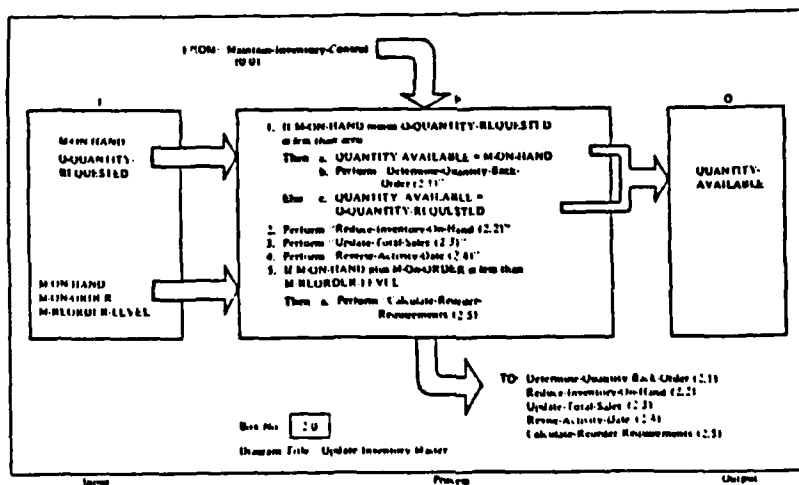


Figure 9 - HIPO  Input, Output, and Process Diagram

(3). Use

HIPO is claimed to be an easy to use scheme because it has few notational requirements[28].

HIPO diagrams were developed at IBM as design representation schemes for top-down software development, and as external documentation aids for released products. VTOC is primarily a high level representation, while IPO diagrams primarily oriented to lower level anlysis.

(4). Advantages

(a). Fairly easy for the user to understand.

(b). It is a top-down approach.

(c). Has a simple interface representation.

(d). Highlights missing information about inputs, processes, and outputs.

(5). Disadvantages

(a). Data structure and control structure are not addressed.

(b). Large system designs require multiple pages, which may become confusing and difficult to maintain.

(c). It is difficult to represent details of the design.

(d). Difficult to use in later phases of the system development.

HIPO has proved to be a useful tool in a variety of business applications. However it has some weaknesses such as

35

the ripple effect, when used for very large applications systems[2].

HIPO is a well known tool among the SIMAER's professionals, since many of them had attended courses at some of the IBM training centers.

4. Discussion

Neither of the two schemes of this category of representation technique, Leighton or HIPO, will meet all of the needs for a complete software design representation. However, each scheme has provided and will continue to provide, much of what is needed in a particular software development situation[28].

3.2.2.2 Data Flow-Oriented Methods

This is perhaps the most widely used approach. This approach seems natural when one is designing an automated system to replace a manual one. In such cases the first step is to build a model of the existing system using a data-flow oriented method.

The methods and tools to be presented here have been widely used for several years.

Peters[28] advocates that this approach can be applied to any software design due to the degree of refinement occuring since its introduction in 1974, and also due to the richness and utility of the evaluation scheme it includes.

36

### 3.2.2.2.1 <u>Structured Design</u>

This is the use of the systems concepts to decompose the information system and define the boundaries and interfaces of each subsystem[12]. It is based on concepts developed by Stevens, Myers, and Constantine[33]. In structured analysis the same notational schemes(data flow diagrams) and concepts are used to model problems and eventually to produce specification and a software design. A variety of tools such as structure charts, data dictionary, pseudocodes, decision tables can be used with this method along the several phases of the life cycle.

Structured design is one of the most used design methods[28]. The reasons for its popularity are: the ease with which it can be used, the evaluation criteria that it includes, the fact the software designer can express ideas in terms of data flow and transformations, and finally, the notation used is simple enough to be understood by management, customers, and the implementer[37].

1. Concept

Structured Design consists of three rationales[28]: One aims at the composition and refinement of the design, another separates issues into abstract issues and implementation issues, and the third enables the user of the method to evaluate the results of his efforts.

a. Composition Rationale

In this rationale, Structured Design views systems

37

in two complementary ways: One is flow of data, and the other is the transformation that such data undergo from input into output. Together, these views form a network model of a system showing data entering as input, undergoing a transformation, perhaps undergoing other transformations; joining, diverging, or being stored with other data, and finally becoming output. This model of software design may sound simple, but it generates several interesting dividends. Among them are the following:

(1) Absence of time in the data flow representation: Since movements and transformations are the only characteristics represented by the data flow diagram, the concept of the passage of time along one or more data flow paths is not present. The designer is free to concentrate on the clear establishment of what major or minor transformations must occur in order for the input data to be incrementally and correctly transformed into output.

(2) Lack of a classical functional decomposition: Top-down design has been described as showing only one path in a tree-like structure, because it assumes there is one problem to be solved. The use of the so-called data flow viewpoint reduces the effect of the designer's experience and biases on the results, and consequently retains the shape or structure of the system.

b. Abstract versus physical design rationale

One serious dilemma for a software designer is

38

the cycling between (abstract) high level design issues and
(physical) implementation details that usually occurs when
the designer tries simultaneously to understand the design
problem and to define a practical solution. It is a dilemma
because, while the designer is composing design solutions to
the problem even at a high level, he may shift his focus to
details of implementation. This increases the potential for
mistakes to be made. To avoid this problem, structured design
recommends a disciplined dichotomy between abstract or
logical design issues and physical design.

c. Design evaluation rationale

One of the most valuable aspects of the
structured design method is that it offers a set of non-
mathematical criteria for evaluating a software design. Two
classes of criteria are used: system level(coupling) and
module level(cohesion). A module is defined as a contigous
set of instructions that may be addressed by name[28].
Coupling is a means of evaluating the relationship between
modules in a system. The strength of their connection or
degree of interdependence will determine how easily the
system may be maintained or enhanced. The desire is to
minimize the coupling. Cohesion, on the other hand, is a
measure of the intramodule strength of connection. The desire
is to maximize the module's internal strength[37].

2. Approach

39

The basic strategy used in structured design is to identify the flow of data in the problem and to incorporate both detail and structure in an iterative fashion. A system specification that identifies inputs, desired outputs, and a description of the functional aspects of the system, should exist before design begins. The specification is used as a basis for the graph depiction, a data flow diagram, of the inherent data flows and data transformations. From the data flow diagrams, natural aggregates of these transformations and data flows are identified.

Following the structured design procedure(Fig. 10), this step eventually leads to definition and depiction
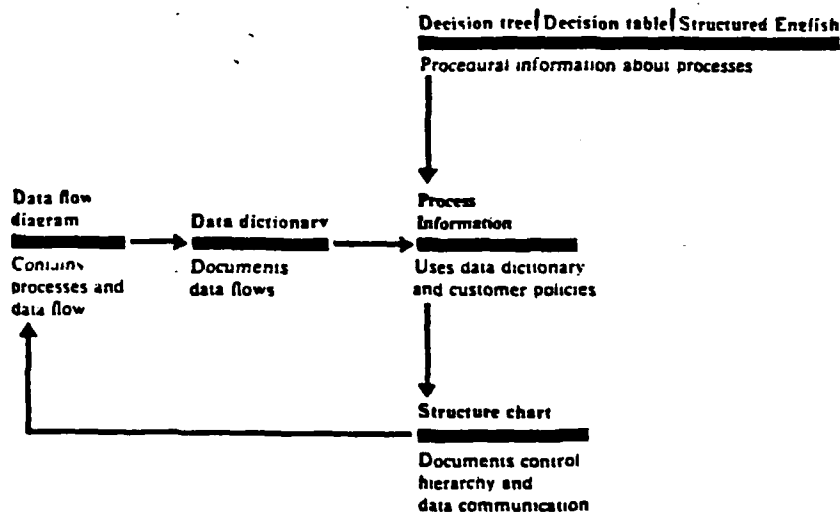


Figure 10 - Overview of the Structured Design Method

of modules and their relationships to one another and to various system elements, in the form of structure charts(fig. 11).
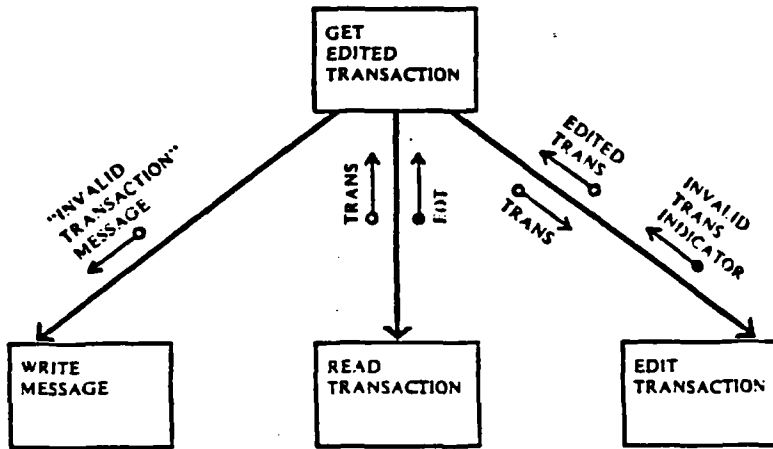
Figure 11 - Example of a Structure Chart

At this point, the system specification is re-examined, errors
or omissions remedied, and the process selectively cycled
through again (Fig.12).
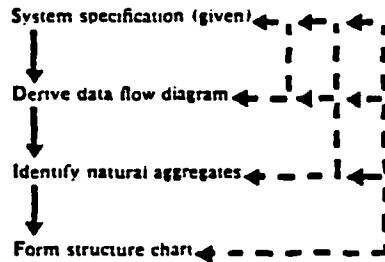


Figure 12 - General Flow of the Structured Design Method

3. Tools

a. Data Flow Diagrams

(1) Concept

Data flow diagrams consist of the
representation of individual functions and the flow of data
between those functions.

41

(2) Notation

The elements of the the data flow diagram (Fig. 13) are called "transforms" and are represented by small circles or "bubbles". As their name implies, the transforms represent transformation of data (which eventually will be accomplished by a module, a program, or even an entire system) from one form to another form. The data elements are represented by labeled arrows connecting one transform bubble to another.

(3) Use

This tool is suitable for both the analysis as well as for the preliminary design phase.

(4) Advantages

(a) Has ability to completely represent data flows.

(b) May be used in high and low level analysis, providing a good system documentation.

(5) Disadvantages

(a) Does not clarify I/O details.

(b) Does not provide the variety of structural mechanisms available in other tools, such as SADT, to be seen later.
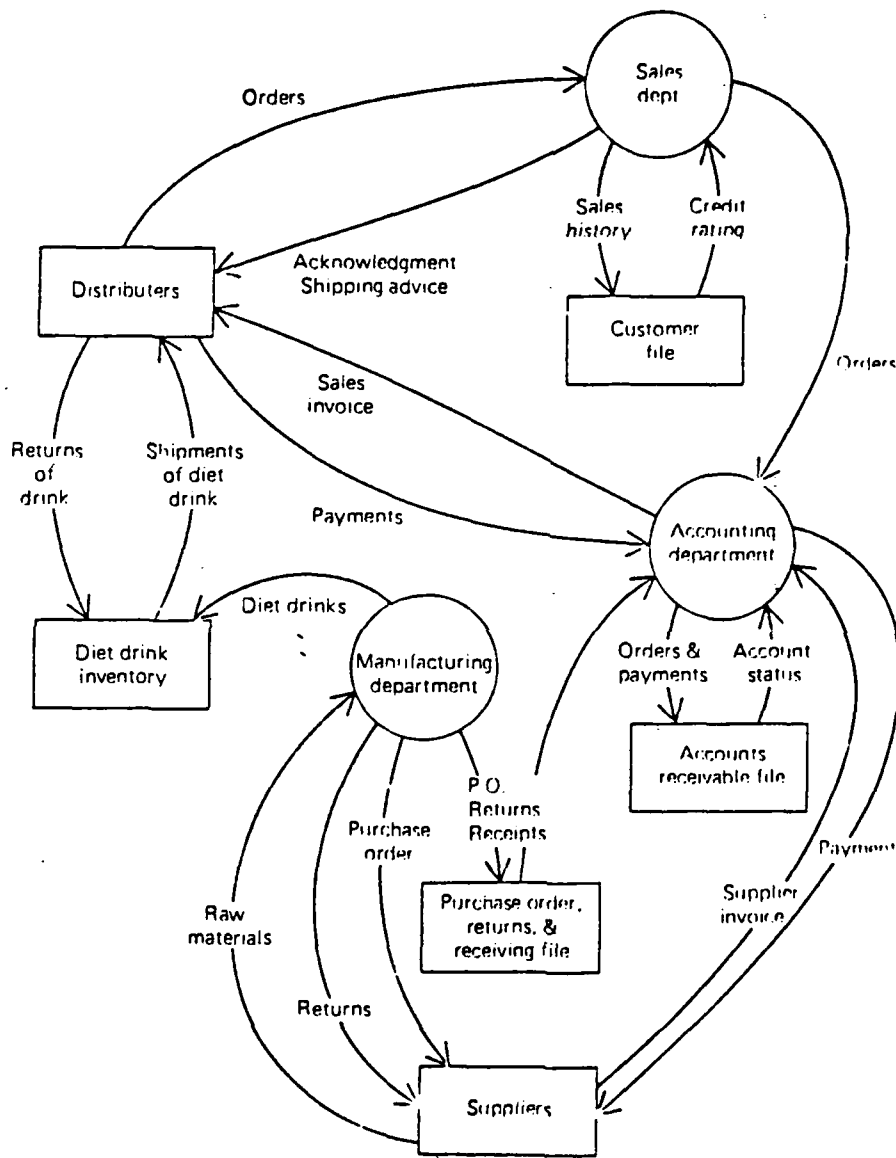
Figure 13 - Example of a Data Flow Diagram

b. Structure Charts

Structure charts were originally developed by Stevens, Meyers and Constantine to specify modular characteristics of software design[28]. The charts are integral part of the structured design method[13].

(1) Concept

The notational basis for the structure chart is the design tree. A module includes other modules and may invoke subordinate modules or be invoked by superordinate modules. A module is defined as a set of lexically contigous program statements that can be referred to by name[28]. The information communicated between modules is graphically depicted. The relationships and interactions depicted include data flow, activation, and communication of control parameters. This scheme specifically identifies modules that will compose the software system, but permits design quality to be evaluated by the criteria of coupling and cohesion.

(2) Notation

Structure charts can be drawn in different ways. The notational syntax proposed by Constantine et al. utilizes three basic graphic forms shown in figure 11:

(a) the rectangle, used to contain a module name or a module descriptor.

(b) the vector that highlights control relationships between modules (such as a call and subsequent

return).

(c) the arrow with a circular tail, used to depict
the transfer of data between two modules. The
arrow with a filled tail represents a control
flag.

(3) Use

This notational scheme is most suited    for
the design phase, which means that another tool should be
used for the analysis phase. Structure charts, in turn,
should also be complemented, in the design phase, by another
tool such as a pseudocode to give more accurate and extensive
information concerning how a module performs. This technique
seems to be adequate for a broad category of systems
applications such as: scientific, business, interactive,
real-time, batch, etc.

(4) Advantages

(a) Allows for iteration in the design
process.

(b) Is a relatively easy to use and understand
notation.

(c) Addresses both data and control.

(5) Disadvantages

(a) Does not allow for detailed decision
information without extra documentation.

(b) Even when decomposition is complete, the
modules may not have the detailed

information needed for implementation.

(c) If the design is for a new system, for
which there is not an existing preceding
system, it may be hard to identify data
flows, transactions and transformations.

c. Data Dictionary

A data dictionary, as the name implies, is a
repository of information about data[18]. Gane[18] points out
that the name gets stretched when we start to include the
details of processes, which strictly speaking are logic,
rather than data, and suggests that perhaps the data
dictionary should be called a project directory.

(1) Concept

Peters[28] states that the data dictionary is
a requesite tool in successful software design. It enables
the designer to establish the same sort of compositional
relationship for data that are employed for functions and
modules in the executable portions of the software.

Its role in analysis has been one of aiding
communication between analyst, customer, and user by ensuring
that they are speaking a common language. Its role in design
has been to clarify to the designer the flow and content of
data items through the system.

(2) Notation

Different authors have different schemes for

representing data dictionary information. Peters states that some schemes are inconvenient to use or require an automated tool. He suggests the representation shown in Fig. 14 below, which, according to the author, is easy to use manually, can be typed, or entered into a text processor.

| SYMBOL | DEFINITION | EXAMPLE |
|---|---|---|
| = | IS EQUIVALENT TO<br>IS COMPOSED OF | ORDER = CUPON AND PREPAYMENT<br>ORDER = CUPON + PREPAYMENT |
| + | AND | ADDRESSEE = CUST-NAME + ADDRESS |
| [] | EITHER-OR | ADDRESS =<br><br>PO BOX<br>STREET ADDRESS + STATE + ZIP |
| {} | ITERATIONS OF | PLAYER-ROSTER = {PLAYER-NAME +<br>PLAYER-NO} |
| () | OPTIONAL | PLAYER-NAME = SURNAME + (MIDDLE-<br>INITIAL) + FAM-NAME |

Figure 14 - An Example of a Data Dictionary Notation

(3) Use

A data dictionary can be used either automatically or manually, depending of the amount of information stored. The notational scheme presented in Fig. 14 can be employed in any one of several different ways. Many designers prefer to use notational variants that can be implemented on text processors or line printers.

A data dictionary can be applied in every phase of the software life cycle.

(4) Advantages

47

(a) Establishs a glossary of terms.

(b) Provides a standard terminology.

(c) Provides cross reference.

(d) Defines all terms associated with a system.

(e) Resolves problems associated with aliases and acronyms.

(f) Provides centralized control for systems changes.

(g) Provides reference guide for training and design evaluation.

(h) Helps minimize maintenance costs.

(5) Disadvantage

(a) Requires automation for large systems.

d. Pseudocode

Suggested as maybe the oldest software representation scheme still in widespread use today, it has undergone considerable changes with the advent of more modern software design and development practices. According to Peters[28], many forms of pseudocode have been suggested, but no standard or unique form of pseudocode has been widely adopted.

(1) Concept

Pseudocode is a program-like, but informal notation, containing natural language text, used to describe the functioning of a procedure or program[14].

48

Peters[28] states that the basic idea is to permit the designer to capture rapidly and conveniently the important elements of the design and to do so in a such way as to give designer maximum flexibility.

(2) Notation

There is no standard. Some specialists in software design have chosen to utilize the Pascal, Algol or other programming language syntax and semantics as a standard, while others have proposed their own. A normal practice is to select one that is well-matched to the target programming language[14].

Usually the designer describes system characteristics using short, concise, English language phrases that are structured by key words such as If-Then-Else, While-Do, and End. Key words and indentation describe the flow of control, while the English phrases describe processing actions.

(3) Use

Pseudocode can be used in both the preliminary and the detailed design phase. Like flowcharts, pseudocode can be used at any level of abstraction[17].

(4) Advantages

    (a) Is superior to flow diagrams since it allows the structure of a program to appear explicitly and facilitates top-down design[14].

(b) Does not involve specific syntax or
semantics.

(c) Can replace flowcharts and reduce the
amount of external documentation.

(5) Disadvantages

(a) The freedom of syntax or semantics can
lead to coding under the guise of
designing.

(b) The lack of universal standards can lead
to communication difficulties within the design
team.

e. Decision Tables

Decision Tables are a tabular method of describing
or specifying the various actions associated with
combinations of conditions. The method is tabular in that it
uses a special form of table to present the associations. The
actions specified are transformations to be done to data or
materials. The conditions are data variables that describe
the characteristics of the environment and the events that
happen in the environment[15].

(1) Concept

The basic notion is that, for each possible
combination of situations that a system can encounter, the
system's response is known. These situations are referred to
as actions. For every condition or set of conditions that can
occur, one and only one action or set of actions is possible,

the response of the system is known with certainty[28].

     (2) Notation

       As indicated by Fig. 15, the basic decision
table consists of two portions - the condition stub and the
action stub. Conditions are collected and optionally labeled
in the condition stub, while actions are collected, and
optionally labeled, in the action stub. Conditions and
actions are most often described horizontally. Vertical
columns in the condition stub are used to identify the
conditions in a given instance. A corresponding column in the
action stub describes the system's response.

**Condition stub**

| Condition 1 | Y | N | Y | — | — |
|---|---|---|---|---|---|
| Condition 2 | N | — | — | Y | Y |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| Condition n | Y | — | N | — | Y |

**Action stub**

| Action 1 | X | — | X | X | — |
|---|---|---|---|---|---|
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| Action n | — | — | X | — | X |

KEY

Y — Yes

N — No

— — Does not apply or is logically precluded due to other conditions

X — Action initiated

Figure 15 - Parts of a Decision Table

     (3) Use

       Decision tables find use in many phases of

51

system development, including system analysis, design, programming, debugging, and documentation. In system analysis, decision tables help analysts in identifying the significant control variables for the operation being studied. In systems design, decision tables help link the desired action to the control variables. In debugging, decision tables can help reduce time to locate "bugs" because they force a sharp distinction between control logic in a program and the actual production of the output data. In documentation, decision tables can concisely summarize the system or program in written form[15].

(4) Advantages

(a) Can serve as a compact means of describing or specifying operations.

(b) Provides a convenient way of tersely stating logically complex processing.

(5) Disadvantages

(a) Large decisions tables can become incomprehensible and difficult to be checked manually.

(b) Do not indicate the execution flow, data transfer, or database interaction.

4. Discussion

Structured design has gained wide popularity for two primary reasons. One is that it allows the software designer to express his perception of the design problem in terms he

52

can identify with, data flows and transformations. The notation with which he expresses these flows is simple, easy to use, and understandable by management, customer, and implementer.

The other primary reason for this method's popularity is that it provides the designer with a means of evaluating his (and other's) designs, serving as a sort of benchmark against which to measure his success or progress. This means of measure consists of nothing more than the concepts of coupling and cohesion.

However, there are some aspects that this system development method leaves unaddressed. For example, the internal working of the modules is not carefully attended, data flows and transformations are not easily identified with any degree of certainty, and neither does it provide a way of defining the places were the data should be stored whenever necessary between processes.

In conclusion, structured design has much to offer that is unique, tried, and tested, and although it does not address certain key issues such as data storage, it became the basis for the development of other system development methods. By incorporating the several tools and techniques described before, Structured Design can be applied to every phase of the software life cycle.

### 3.2.2.2.2 Structured Analysis and Design Technique

Structured Analysis and Design Technique(SADT), originated and promoted by the SofTech Corporation, is based on the results of studies into computer-aided manufacturing[28].

Under development since 1970, SADT has, according to SofTech, so far been applied principally to the planning and functional analysis of large, complex systems[32].

1. Concept

SofTech claims that SADT is a comprehensive method for performing functional analysis and design. By comprehensive they mean a coherent, integrated set of methods and rules that constitute a disciplined approach to analysis and design, built upon a foundation of closely inter-related concepts. These concepts are:

a. That precise models capable of providing an understanding of complex problems are the best means of obtaining effective solutions.

b. That analysis of any problem should be conducted in a top-down, structured, modular, and hierarchical fashion.

c. That differentiation must be made, as much as practicable, between the creation, first of a functional model of what the system must perform, and the creation of a design model of how the system will be implemented to perform those functions.

d. That the modeling approach must be able to depict both things (objects, documents or data) and happenings (activities performed by men, machines, computers, software, etc). The complete SADT model of a problem must show both aspects properly related.

e. That the system model should be represented graphically in such a way as to highlight the interfaces between component parts and the hierarchical structure that they compose.

f. That the analysis and design method must provide the discipline and coordination between participants, which is required in order to produce results which reflect the best thinking of a team.

g. That documentation and review of all decisions and feedback related to the analysis and design effort is essential.

2. Approach

SADT provides techniques and methods for:

a. Thinking in a structured way about large and complex problems.

b. Communicating analysis and design results in clear and precise notation.

c. Controlling accuracy, completeness, and quality by procedures for review and approval.

d. Documenting the system analysis and design history, decisions, and current results.

55

e. Managing development projects and assessing

        progress, and

        f. Working as a team with effective division and

        coordination of effort as shown in the table VII

        (Appendix B).

    3. Tools

        a. SADT Diagrams

        The   representation scheme includes a data model

and an activity model. However, most designers seem to use

only the activity model[28][17].

            (1) Concept

            The SADT scheme uses two graphic forms: One is

a design tree form which  is a road map for the system model

and the second is the system model used to describe a program

or a group of programs, and composed of one or more activity

charts or diagrams with their accompanying data charts if

used. The activity chart depicts the flow of data and control

information between activities and processes. The basic idea

is to provide the user of the technique with a means of

graphically portraying his analysis of an existing system or

his perception of a system under design.

            (2) Notation

            SADT diagrams employ labeled rectangular

boxes, arrows, labels, and a tree-like structure to maintain

decomposition traceability. Distinctions are made both in

what is represented and how it is represented. For example,

56

the basic distinction between data flow and activities is made, but data are also classified as input, output, or control (see figures 16, 17 and 18).
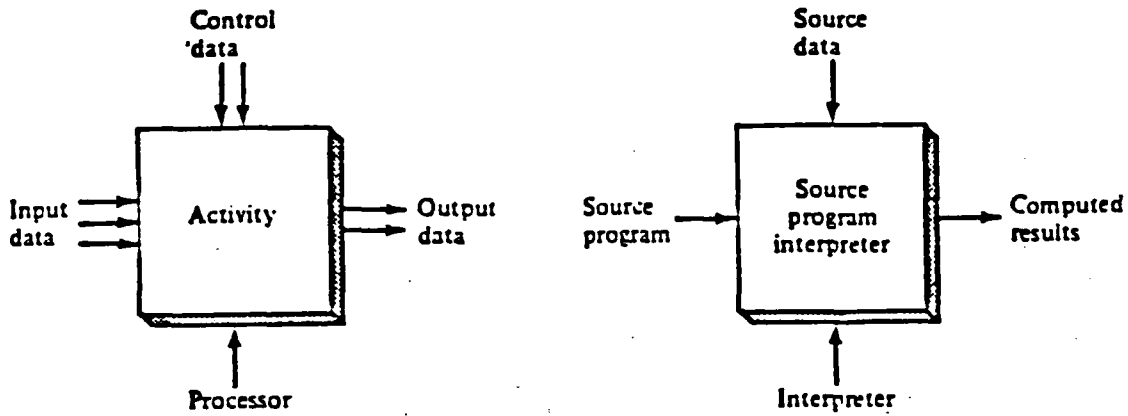
Figure 16 - Activity Diagram
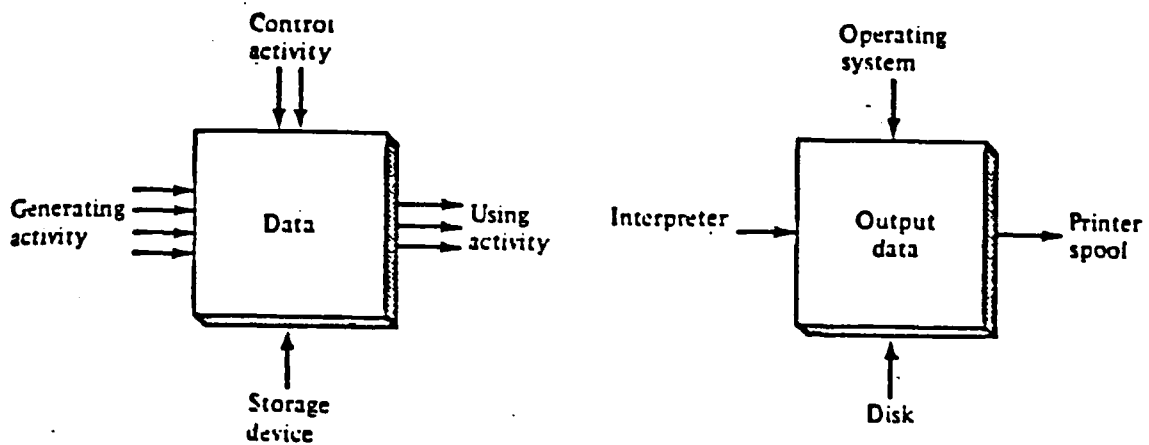
Figure 17 - Data Diagram
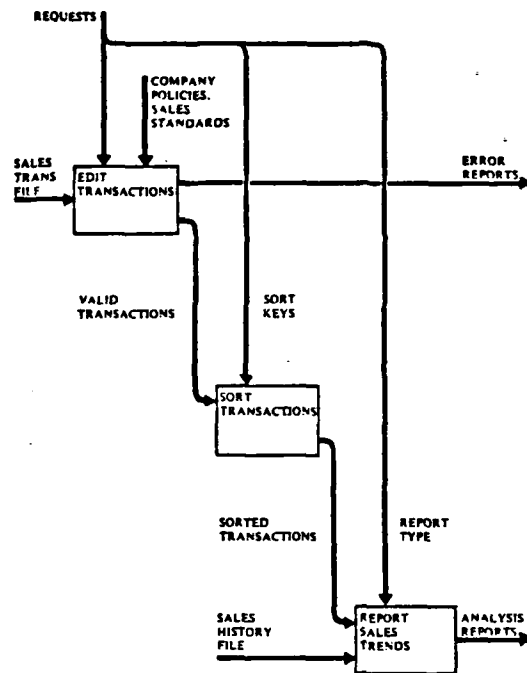
Figure 18 - Example of a Complete Activity Diagram

Note that some data flows in Fig. 18 split into separate

data flows, for example, the REQUESTS flow forks into SORT

KEYS and REPORT TYPE. This separation is known as a fork. The

case of two data flows becoming one is called a join. The

notation should also be complemented by text composed of a

few carefully chosen words.

(3) Use

SADT diagrams are claimed to be a powerful

tool to be used in every phase of the system development, but

it seems that its biggest strength is in the design phase. It

is somewhat difficult for a customer to learn and use in the

analysis phase where customer participation is more

intensive. SADT is considered very useful for real-time

systems design due to the clearly depiced interaction

between modules[28].

    (4) Advantages

        (a) It is a potential powerful notation scheme for analysis, provided that the customer understand how to interpret it correctly.

        (b) Does not require consistency at every level because it uses reviewers to resolve inconsistencies and interface problems.

        (c) Besides representing functions and data flow between functions, additionally shows the control under which each function operates and the mechanisms reponsible for implementation of the function.

        (d) Is complemented by a functional description and a complete data dictionary package.

        (e) There is a public domain version[30].

    (5) Disadvantages

        (a) Contains too much information which can confuse the user.

        (b) Is subject to wide latitudes of interpretation by the reviewers. Without direct participation of the author reviewers are forced to make assumptions.

        (c) Does not provide file or report details.

## 4. Discussion

As we have seen, the notational scheme distinguishs between control data and input data. In a system involving many such diagrams arranged in a hierarchical order, the usual comprehension problems for the design team and customers are compounded by the addition of the dimension of control to the diagrammatic scheme. Also, the advocated policy of permitting each designer or analyst to develop independent diagrams and resolving interface problems via the review cycle may cause additional difficulties. That is, interface problems between a designer's portion of the system and the rest of the system are only considered after he has developed his independent model.

*This method is proprietary of SofTech. However, there is* a public domain version[30].

### 3.2.2.2.3 Gane

Gane's method is a collection of specific tools and techniques described in the book: Structured Analysis: Tools and Techniques[18]. As the name suggests, the procedures and tools are based on the Structured Design Method, presented in Section 3.2.2.2.1. The primary design representation technique is a modified version of the data flow, enriched by the addition of decision tables, data dictionary, pseudocode, Chen entity-relationship, structure charts, etc.. All those tools are conceptually as well as pratically presented. The entire book shows, as an example,

60

the analysis and design process of an hypothetical system
using and integrating the mentioned tools.

1. Concept

Basically the concept is the same as Structured
Design, enriched and complemented by the other tools, as well
as by some features that allow the method to address some
areas the authors felt were not addressed by Structured
Design. Such features include database concepts, data
dictionary, and data store structuring techniques.

2. Approach

The authors start with a discussion of some of the
problems that are faced in analysis and then review the
graphical tools and how they fit together to make a logical
model. Then they take each tool in turn and treat it in
detail, starting with the key tool, the Logical Data Flow
Diagram. Later they sketch out a structured system
development method which takes advantage of the presented
tools. The use of such a  methodology involves the following
steps: build a logical model (nonphysical) of the
system using graphical techniques which enable users,
analysts, and designers to get a clear and common picture of
the system and how its parts fit together to meet user's
needs; build the system top-down using successive refinement;
emphasize the use of iterations for a good development; and
distinguish the work of analysis ("what") from the design
("how").

61

3. Tools

   a. Logical Data Flow Diagram

      (1) Concept

         Gane's method[18] uses one type of data flow
diagram which follows the same general principles of the one
described in Structured Design (Section 3.2.2.2.1). He states
that in analysis we  need to recognize  external entities and
data stores as well as data flows and transforms and process.
Gane's Logical Data Flow Diagram (LDFD) allow this to be
represented.

      (2) Notation

         In Gane's LDFD, functions are shown as
rectangles or boxes and data flows are represented as labeled
lines between the boxes (Fig. 19). He justifies the adoption
of a rectangle instead of a circle to represent the processes
because it is hard to get much legible writing inside of a
circle. Gane's LDFD diagrams are "leveled" so that each
diagram represents further decomposition of the higher level
functions. In addition, the diagrams show external sources
and sinks for data and location of files[11].

      Even without making any physical commitment during
analysis, the authors thinks that there are places where the
definition of the data storage location between processes is
necessary, and represents this location with a pair of

62
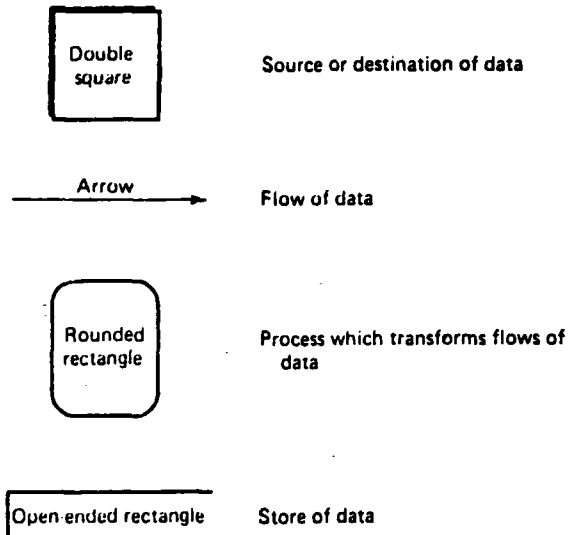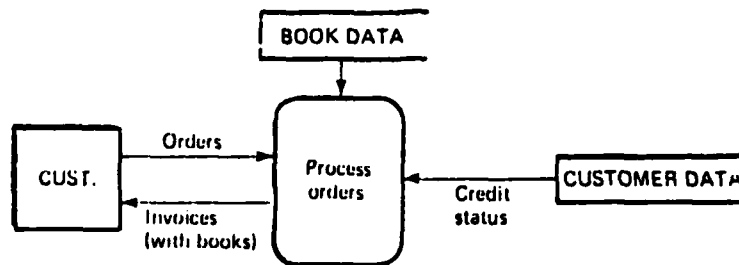
Figure 19 – Gane's Logical Data Flow Symbols

horizontal parallel lines, closed at one end, just wide
enough to hold the name. External entities, which represents
a source or destination of transactions, e.g. customers,
employees, aircraft, tactical units, suppliers etc, are
symbolized by a solid square, with the upper and left sides
in double thickness to make the symbol stand out from the
rest of the diagram. The entity can be identified by a lower
case letter in the upper left hand corner for reference.

Data flow is symbolized by an arrow,
preferably horizontal or vertical, with an arrowhead showing
the direction of flow. Each data flow should have a
description of its content alongside, as shown in Fig. 20.

Figure 20 - Data Flow Description

The function of each process is described with
imperative sentences using an active verb followed by an
object. The process has also an identification and a physical
reference to note how the function will be physically
implemented(Fig. 21).
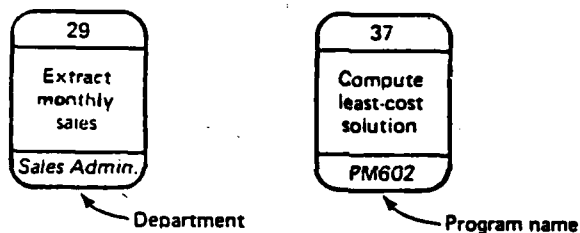


Figure 21 - Process Boxes with Physical References

(3) Use

This tool is suitable for both the analysis as
well as for the preliminary design phase. It is also most
often employed in the traditional data processing
environment.

(4) Advantages

(a) Has the ability to completely represent
data flows.

(b) Has the ability to show data stores.

64

(b) Has the ability to show data stores.

(c) May be used in high and low level analysis, providing a good system documentation.

(d) Is not proprietary, having plenty of documentation available at the cost of buying a book.

(5) Disadvantages

(a) Does not clarify I/O details.

(b) Does not provide the variety of structural mechanisms available in SADT.

b. Chen Entity-Relationship Approach

The Chen entity-relantionship approach[10] is one of several schemes to recognize that database as well as code needs to be designed in two stages - logical and physical.

(1) Concept

This design representation scheme contains three classes of things: entities, relationships, and attributes. Entities are objects that can be uniquely identified. Groups of entities may constitute an entity type, such as employee or automobile. Relationships are conceptual links that exist between or among entities. Attributes are properties possessed by entities and relationships, and have corresponding values.

The basic scenario in using the Chen scheme involves identifying and documenting entities, relations, and

65

their interaction, identifying and documenting attributes and

values, and, finally, combining these results into the form

of a data structure that may be implemented on any database

management system (DBMS). The process used to produce Chen

entity-relationship diagrams is shown in Fig. 22.

```
Identify: attributes------Form attitude-value
          values              diagrams
          entities
          relations------Form entity-relation
```

Figure 22 - The Process Used to Obtain a Chen Entity-
             Relationship Diagram


(2) Notation

Each concept used in Chen's approach is

represented by an individual building block in the diagram.

Several blocks of the appropriate type may be combined

according to a set of conventions. The individual forms of

this notation are as follows (Fig. 23):
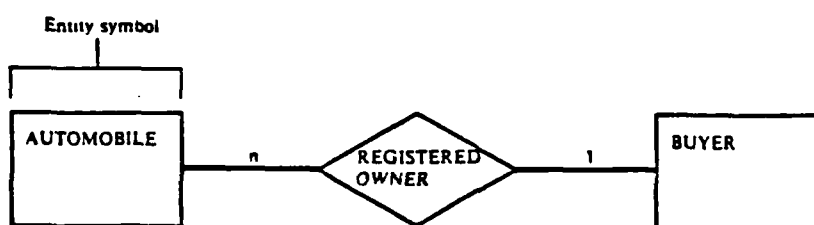


Figure 23 - Depicting Relationships Using Chen's Approach

(a) Entity types are depicted by rectangles.

(b) Relationship types are depicted by diamonds

    with lines connecting them to the appropriate

    entity types. Note the introduction of the

66

symbol n and 1, which document the nature of

the interaction between the relationship type

and the entity type.

(3) Use

The Chen entity-relationship approach

incorporates many of the features of the data dictionary

(such as data composition and organization) while providing

the software designer with a flexible means of depicting

information-based (as opposed to processing-based) problems.

The Chen approach forces the software designer

to view data not as a hierarchical arrangement, but as set of

entities, each possessing certain attributes, and each having

relationships of one kind or another with other entities.

This shift in view is somewhat analogous to the data flow

view of software systems. In the data flow view, design is

based on what is thought to be the most stable characteristic

of a system, the flow of data. Content may change, but

communication lines are stable. Similarly, in database

modeling, the relationships between and among entities will

be stable, although new entities, and new or modified

attributes may be incorporated over time. Such changes may

affect hierarchical structure more profoundly than relational

structure[28].

(4) Advantages

(a) Recognizes that code as well as database

needs be designed in the two stages -

67

logical and physical.

    (b) Is a well-suited tool to represent
database design.

  (5) Disadvantages

    (a) Assumes that flow of data will be stable
which may not always be the case.

    (b) Is not necessarily good for real-time
systems.

    (c) Does not show control of data.

The other tools used by Gane's method ere already covered under other methods.

   4. Discussion

As we see, Gane's method is a package of tools and techniques that draws on structured design and relational database theory, adding to them their own data flow representation technique, and a set of steps to be followed in the process of system analysis and design, covering all phases of the software development process. Though this set of representations fails to clarify I/O details, it is otherwise complete[11].

This method is publicly available through the acquisition of a number of books and is not proprietary.

### 3.2.2.2.4 Summary of Data Flow-Oriented Methods

These three software design methods - structured design,

68

SADT, and Gane have in common the data flow orientation approach. However, some conceptual differences exist. Next those relevant differences are discussed.

1. Communication Ability

SADT has a richer and more complex notation than Structured Design or Gane's method. However, experience with SADT has shown that its diagrams can often be misinterpreted, even by someone who is conversant with the method, but is not the author of the diagram.

Data Flow Diagrams, used by both Structured Design and Gane's method, are easy for the user to understand.

Gane's method turns out to be more complete by the adding of data storage and database representation *diagrams*.

2. Use

SADT is more suitable for large, complex, real-time systems, while Structured Design and Gane's methods are more processing oriented.

Of the three Gane's method is the only one that addresses database issues.

3. Mechanism Clarification

Like the data flow diagram used both by Structured Design as well as by Gane's method, SADT diagrams represent functions and data flow between functions. However, the SADT diagram additionally shows the control under which each function operates and the mechanism(s) responsible for the

implementation of the function.

Data flow-oriented methods are easier to use
when the software product is being designed to replace a
manual or previously used automated system. This happens
because paper document movement usually can be correlated to
data flow in such a way that you can trace this movement
between the source and destination. It is much more difficult
to use these methods if the system designed has no existing
predecessor.

Users of these methods should be cautioned of two
possible difficulties when applying these methods:

a. If a predecessor system (especially a manual one)
exists, there may be a tendency to try to automate the
predecessor on a one for one basis without exploring
alternatives which may lead to a better implementation.

b. If no system currently exists, then the designer
initially must try to understand the system before he uses
paper. This relies on the experience of the designer and any
information which he may be able to get from the customer,
user, fellow designers, etc. Any deficiencies in the mental
image formed will be passed on in the design and may result
in system which is unacceptable to the user[37].

3.2.2.3. Data Structure-Oriented Methods

The data-structure oriented method designers advocate
observing data at rest. The emphasis is on identifying and
observing logical relationships between discernible data

70

elements, for these relationships form the basis of the program itself.

Data structure approaches claim[28] that, given the same set of information, two experienced software designers would come up with the same design. The basic process is: The software designer first identifies the data needed by the program to do its job, then organizes it according to its natural hierarchy, and finally produces a program by following a translation procedure.

Due to its popularity and large usage, only one approach will be examined in this section, Jackson's method.

### 3.2.2.3.1 Jackson's Method

The basic approach and style of Jackson's method[22] makes it highly attractive to those working with commercial software design applications, such as finance, inventory, banking or insurance[28].

1. Concept

Jackson's method views software as a mechanism that transforms input data into an output report via set of coherent, synchronized operations. The problem for the designer is to determine what the operations and their sequence ought to be. Jackson attempts to overcome the lack of direction present in some top-down approaches by providing guidance to the designer  by restricting possible system sructures. The basic structure of the system  is determined

71

by the structure of the data it processes. According to
Jackson, the software designer's problem is that of matching
the structure of the input, output, and program. It is thus
assumed that using input and output structure as a guide to
program structure will result in a well-structured program.
Apparent in this approach is the ease with which problems
involving serial file structures can be solved. But,
according to Peters[28], the most important assumption
underlying this method is that the software designer knows
what the inherent data structure is or knows how to identify
it. However, Jackson's method does not tell the designer how
to structure data.

2. Approach

The basic process for using this method is: identify
the structure of the input data and output report, define a
program structure based on these structures, and identify the
discrete operations composing the program, and assign each to
a component of the design.

Several types of problems encountered by the software
designer are recognized and addressed by this method.
Foremost among them is the structure clash, which refers to
those instances where input and output data structures are
markedly different, violating Jackson's requirement of a
common program structure throughout all data structures.

The use of this method is complemented by the
specific notation described below, which enables the software

72

designer to depict iteration, selection, and sequence operations.

3. Tools

a. Jackson's Data Structure

Jackson's notational scheme represents a useful way of depicting database characteristics[22].

(1) Concept

The basis for Jackson's approach is the premise that a well-structured program design must parallel the structure of the data. Hence, this approach utilizes concepts from programming in order to depict data structure. The data structures are a model of the problem and the logic flows are limited to sequence, iteration, and selection.

(2) Notation

Jackson's uses rectangles to represent data (e.g., files, arrays, and individual data items), while connecting lines describe ownership or composition as shown in Fig. 24. An asterisk and a degree symbol are used to denote selection and iteration, respectively.

(3) Use

Fig. 25 presents an example of the use of Jackson's scheme. Notice that it is similar to other approaches, such as structure charts. It is good enough to represent the database characteristics associated with a single program; but it is not enough to represent systems of programs. This tool is more suitable to be used in the

design phase.
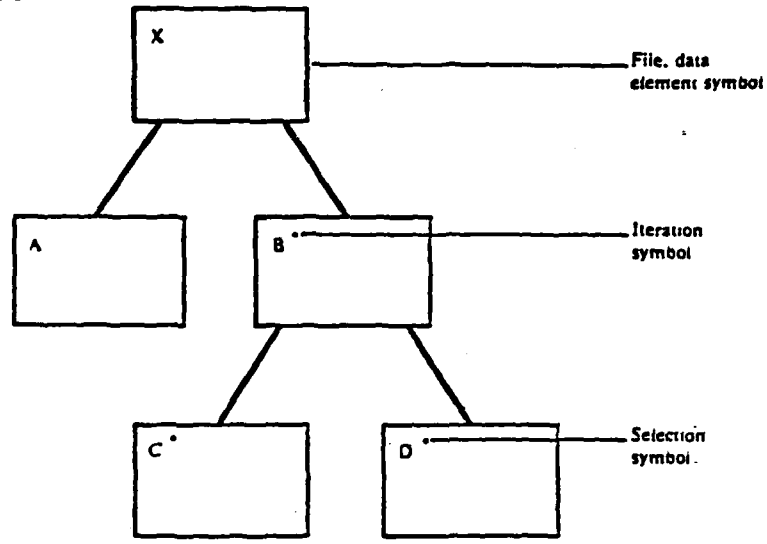


Figure 24 - Notation Used in Jackson's Approach
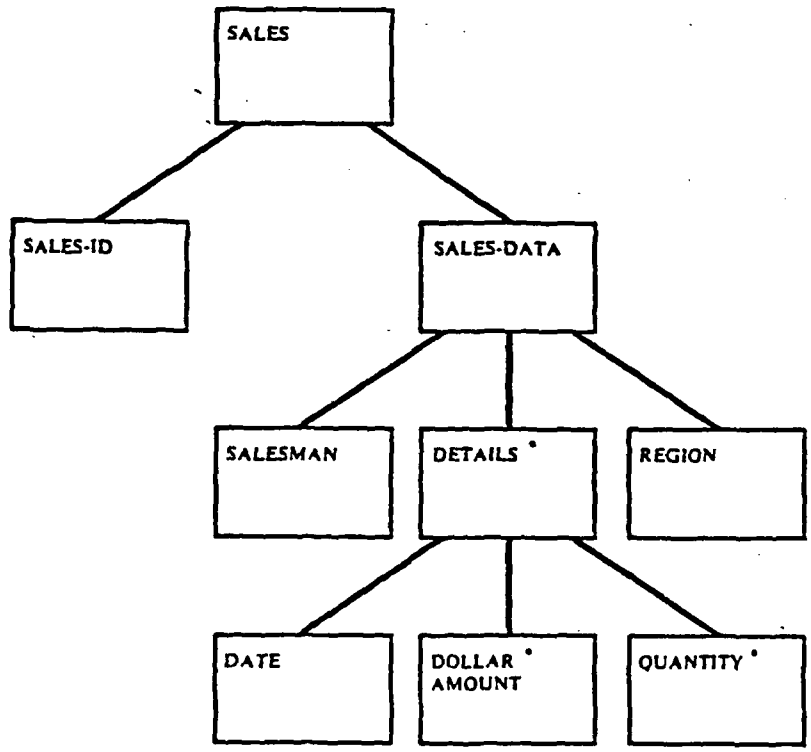


Figure 25 - An Example of the Use of Jackson's Approach to
Depict Data

(4) Advantages

    (a) Works well for file processing systems.

    (b) Works well for small systems or programs.

(5) Disadvantages

    (a) Little use of level of detail and partitioning since it is only a program design tool.

    (b) Not particularly top-down.

    (c) Not easily understood by the end-user of the system.

    (d) Less useful for non-file processing systems.

    (e) Data stream change may cause considerable change in hiearchy.

    (f) Not a system design tool[37].

4. Discussion

Jackson Structured Programming is widely used and is quite effective in situations where input and output data structures can be defined in a precise manner. It appears be to most effective in data processing applications[37].

3.2.2.4 Comparison of the Methods

In order to compare the most known methods, which have been reviewed in the literature, some criteria considered relevant for a system development method in general, as

well as to meet the SIMAER's specific requirements, were established. The establishing of these criteria is not meant to be exhaustive, but rather just a guideline to assist in evaluating the effectivennes of the methods as a software design tool as well as to support the SIMAER's requirements.

The phases of the Life Cycle where each method and tool can be applied is shown in table VIII in Appendix B.

Evaluating each method against the criteria led to the results indicated in table IX (Appendix B). The criteria are described after the table.

### 3.2.2.5 Selection of Software Design Methods

All methods reviewed showed some strengths in one or other area. This research is interested in finding the ones that besides having the maximum of the desirable characteristics of a method, at the same time satisfies as much as possible the SIMAER's requirements. Comparing the requirements with the characteristics of table VIII, Appendix B, thefollowing conclusion was reached.

1. Jackson's Method

Is the only one that does not provide capability to model an existing system, nor to permit the statement of requirements for an entire system. It is also not as easy as Structured Design or Gane for the user to understand. It is known by a few of the  SIMAER's professionals and is suitable only for the design phase.

2. SADT

Its strength lies in its suitability for design of real-time embedded systems through its data control ability. Unlike Gane, it does not incorporate database concepts[17]. Its level of proliferation in Brasil is low, and it is not known by the SIMAER's professionals.

3. Structured Design and Gane

The Gane method, being a derivative of the structured design, covers all aspects addressed by the later, and adds to it some other tools and techniques, such as data dictionary, storage representation, database issues, etc., which makes it a comprehensive and flexible methodology. Another advantage comes from the fact of being well known by some of SIMAER's professional, which will certainly reduce the time and cost necessary for its assimilation. Its main weakness comes from the fact that it is more data processing oriented.

Considering the above aspects, the incompleteness of all existing methods, and the varied scope of the SIMAER's requirements, the conclusion is not to establish a standard method, but rather recommend some, having their adoption induced by providing extensive and intensive training on them. The recommended methods will be the ones presented by Gane[18] for management information systems and SADT[32] for real-time and embedded systems.

The use of various methods during the phases of

development for various types of problems are discussed in
Chapter V.

The approach of not having a standard method is,
in essence as will be seen later, the same solution adopted
by similar organizations around the world. The tools and
techniques to be recommended are those included in the
suggested methods.

### 3.2.3 Documentation

Documentation is a vital part of a system development
cycle. Documentation of a system development falls into two
broad categories - development documentation and control
documentation. Development documentation records how a
software development is structured and what the software is
supposed to do, and gives the background information upon
which the design is founded. Control documentation, on the
other hand, serves an administrative function. It records the
resources used in developing and implementing the system.
This includes such documents as project plans, schedules,
resource allocation details, and progress reports[15].

### 3.2.3.1 Functions of Documentation

Documentation serves four main functions:

1. Intertask/interphase communication.

2. Historical reference for modification and
   correction.

3. Quality and quantity control.

4. Instructional reference.

a. Intertask/interphase communication - This operation records what has been done at each stage of the project so that instructions can be issued for the next phase of work, or so that all people involved in the project can agree what has been done before work proceeds to next step.

b. Historical Reference - The reference function is relevant to both commercial and scientific work. It is the documentation of how the system works that makes it easily changed after it is implemented. All systems are subject to change, with the sole exception of one-time problem-solving applications with limited amounts of data. A system can be maintained efficiently only if the existing operation of all procedures and programs is clearly known and understood. The documentation of the system provides this knowledge.

c. Quality/Quantity Control - As a system develops, various elements of documentation are completed as each step is finished. Management can use this documentation to evaluate the project progress and individual performance.

d. Instructional Reference - the development documentation can be reviewed during and after development for many general purposes. For example, documentation will enable trainees to study a system developed by experienced technicians. Another benefit of documentation is that an outside party can evaluate the system and its method of operation to determine if the package is suitable for use in

another environment. In this case, sufficient information must be given to enable the user to apply the software to other problems and requirements.

3.2.3.2 <u>Types</u> <u>of</u> <u>Documentation</u>

In the development of a system certains categories of documentation must be considered. These are:

1. Analytical documentation.

2. System documentation.

3. Program documentation.

4. Operations documentation.

5. User/management aids.

a. Analytical documentation - consists of all the records and reports produced when a project is initiated. These include: User requests that state the problem, a feasibility study that evaluates possible solutions, and a project plan that estimates the time and resources required to develop and implement the system.

b. Systems documentation - encompasses all information needed to define the proposed system to a level where it can be programmed, tested, and implemented. The major document is some form of system specification, which acts as a permanent record of the structure, its functions and work flow, and the control of the system. It is the basic means of communication between the systems design, programming, and user functions.

80

c. Program documentation - comprises the records of detailed logic and coding of the constituent programs of a system.

d. Operations documentation - Specifies those procedures required for running the system by operations personnel. It gives the general sequence of events for performing the job and defines precise procedures for data control and security, data preparation, program running, output dispersal, and ancillary operations.

e. User/management aids - consists of all the descriptive and instructive material necessary for the user to participate in the running of the operational system, including notes on the interpretation of the outputs results. It usually is contained in the User's Manual.

Brandon[8] points that every installation should establish documentation standards(i.e, rules for the completion of certain documents at certain times) that define the content, format, and distribution of the documents.

### 3.2.3.3 Summary

Documentation is a vital element in developing and running any computer project, either in a governement, business, academic, or military installation. It must not be handled in a haphazard fashion. Formal documentation standards must be laid down and enforced. These standards must cover all areas - users, systems, and programming, and

operations activities.

### 3.2.4 Software Development Management

The difference between software success and failure is often closely related to the quality of software management. The major sofware management problems have generally been the following:

1. Poor planning and coordination - This leads to large amounts of wasted effort and idle time because of duplication of tasks, tasks unnecessarily performed or overdone or poorly interfaced[5].

An important aspect of good planning is an efficient allocation of computing resources and personnel. Most problems occur at the interfaces of modules written by different programmers. The number of such interfaces grows as the square of the number of individuals involved, and the problem becomes unwieldy when the group grows to four or more[37]. The chief programmer team idea is one approach to the solution of this problem[36].

2. Poor Control - Even a good plan is useless when it is not kept up to date and used to manage the system development. One reason for poor control of software projects is the lack of surface visibility of the project. This means that it is difficult to assess the degree of completion of a project. The notion of a milestone[38] is useful, where a milestone is the specification of a

82

demonstrable event in the development of a system.

3. Poor Resource Estimation - without a firm idea of how much time and resources a task should take, the manager is in a poor position to exercise control. There is a lack of experience in software design in making accurate estimates when compared with the other activities. Some of the tools suggested by Zelkowitz[38] to increase the accuracy are:

     a. Compare the project to similar previous projects.

     b. Divide the project into units and compare each
        unit with similar units.

     c. Schedule work and estimate resources by month.

     d. Develop standards that can be applied to work.

4. Poor Accountability Strucuture - projects are usually organized and run with very diffuse delineation of responsibilities, thus multiplying all the above problems.

5. Inappropriate Success Criteria - minimizing development costs and schedules will generally yeild a hard-to-maintain product. Emphasizing "percentage of code written" tends to get people coding early and results in neglect of key activities such as requirements and design validation and test planning[5].

6. Procrastination on Key Activities - this is specially prevalent when reinforced by inappropriate success criteria

So far most of what has been presented was based on the current literature, at a more theoretical level. In the

next sections it will be seen how those theories have been applied to some ADP organizations similar to SIMAER.

## 3.3 U.S. Organizations

Looking to get more practical information on the current system development issues, it was decided to take an overview of what is being used by some organizations. For the purpose of this section some American organizations were evaluated. The United States Air Force was chosen as a reference due to the similarity of the mission, as well as the high degree of technological advancement reached by USAF in software development.

Most of the information presented here was taken from the pertinent regulations and standards established by the USAF, mainly from 300 and 800-series regulations.

## 3.3.1 Software Life Cycle

According to Air Force Regulation (AFR) 300-15 the USAF has established, although not mandatorily, a software life cycle (Fig. 26) aligned with the waterfall model and composed of the following phases:

1. Conceptual phase - where the mission and system requirements are determined by the user with data automation support and includes these tasks:

    a. Identifying the requirements.

    b. Analyze the requirements.

    c. Prepare the documentation.

A system/subsystem requirements review (SRR) ends the conceptual phase. Certain formal documents must be prepared during this phase such as: Data Automation Requirements (DAR), the Data Project Directive (DPD), the Data Project Plan (DPP) and the Functional Description (FD). These documents are described in the AFR 300-12.
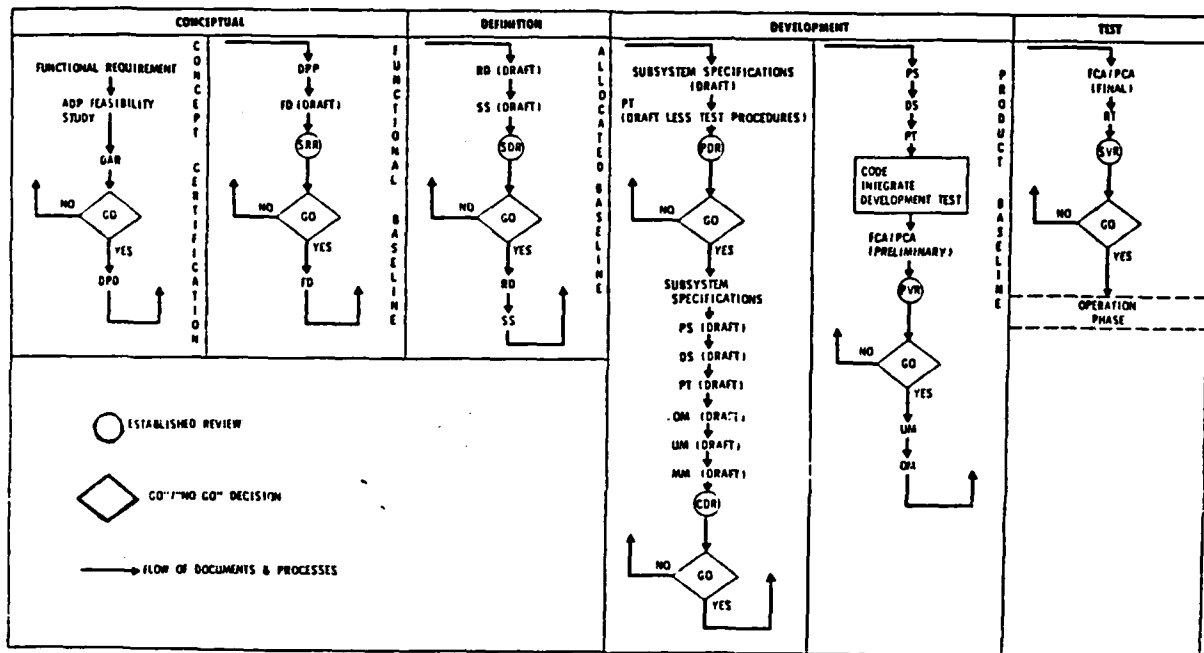


Figure 26 - USAF's Automated Data System Life Cycle (AFR 300-15)

2. Definition Phase

In this phase the developer(s) defines the design requirements for the major elements in the system. This includes:

a. Develop system interface control requirements i.e. define the interface between the operational functions that the system is to perform.

b. Expand system requirements, i.e make a full and

critical review of all performance and design requirements, and expand them.

3. Development Phase

Analysis and design, coding, debugging, integration, and development testing are done in this phase. It includes:

a. Preliminary Program Design.

b. Initiation of the supporting documentation composed of drafts of: the Program Maintenance Manual (MM), the Users Manual (UM), the Computer Operational Manual (OM), and the Test Plan (PT).

c. Detailed Function Design

In this step the flowcharts, HIPO, structure charts, or other logic designs, algorithms, and narrative descriptions are expanded. This must be done in enough detail to provide the basis for actual coding. Defining the database by giving the number, type, and structures of tables and a description of the items in the tables is finished at this time.

Users and developers conduct a Critical Design Review (CDR) to assure that the design meets its functional development requirements, and the design is defined fully enough to permit the start of the coding. After the program specification (PS) is approved during the CDR, the design of the ADS is finished.

Next the coding, compiling, and validation of the modules are performed. Development testing of coded modules

86

is done until complete programs and systems are developed and validated.

As far as could be concluded from the readings, there is not any software design representation technique established as a standard for the USAF. However, in Section E of AFR 300-15, Development Phase, a vague and generalized reference to flowcharts, HIPO and structure charts is made. The same comment holds for methodology.

### 3.3.2 Documentation

AFR 300-12 establishes three types of documentation to be used in a system development: requirements, management, and technical. Considering the scope of this research, it was felt that only the technical documentation needed to be discussed here. It includes:

1. Baseline documents

    a. Functional Description (FD)

    This is a basis for mutual understanding between the development group and the user group of a proposed ADS. It reflects the definition of the system requirements and provides the ultimate users with a clear statement of the operational capability to be developed.

    b. System/Subsystem Specification (SS)

    This is a technical document that governs the development of an ADS or subsystem of an ADS. These are the specifications for the performance, interface, and other

87
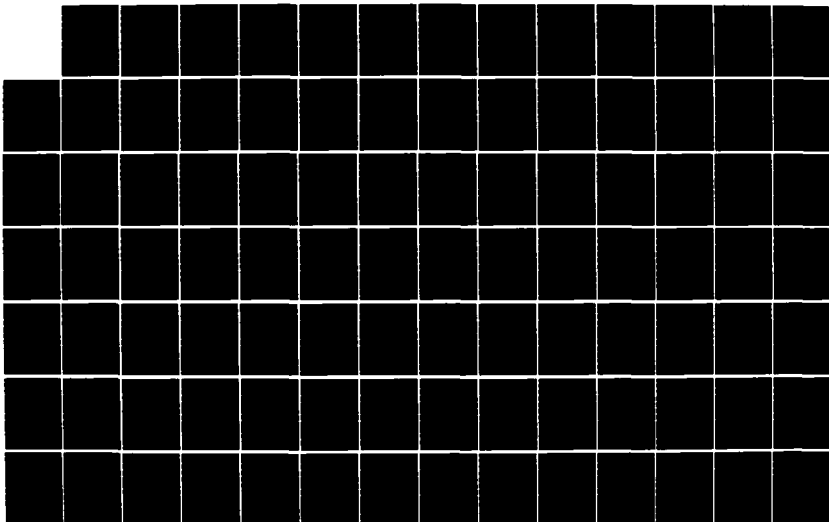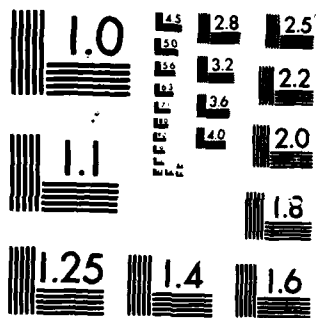
MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

technical requirements needed in designing the system.

      c. Program Specification (PS)

         This is written after SS to expand on its
requirements.

      d. Data Requirements Document (RD)

         This specifies the characteristics and limitations
of required data. It defines inputs required of the user,
procedures for providing this input to the system files and
expected outputs. It also defines the use of standard data
elements and codes, and any data limitations.

       (e) Database Specifications (DS)

           This document specifies the design of the
database and defines the interfaces.

   2. Support Documents

    a. Test Plan (PT)

       This is a tool for directing the ADS testing and
contains the orderly schedule of events and list of materials
to effect a comprehensive test of a complete ADS.

    b. Computer Operators Manual (OM)

       This contains precise and detailed information on
the control requirements and operating procedures to
successfully initiate, run, and terminate the system. It is
directed toward supervisory and operator personnel.

    c. Users Manual (UM)

       This tells in general, nontechnical terms, how to
use the system and its computer programs. It tells the user

about inputs and outputs, and other specific information is necessary for effective use of the system.

3. Internal Documents

   a. Development Test Plan (DT)

   This plan specifies the method to be used in development testing. It outlines test management reports, controls, manpower, acceptance, criteria, and test procedures.

   b. Program Maintenance Manual (MM)

   This is the manual to be maintained by the programmer (or team) that is responsible for each module.

### 3.3.3 System Development Management

Some of the principles that have been established by the USAF on system development management and the respective regulations that enforce them are:

1. Use or establish standard data elements and related features, as specified by AFR 300-15.

2. Use of HOL's, as specified in AFR 300-10.

3. Prepare system documentation according to DOD standard 7935.1-S.

AFR 300-12 establishes the types of reviews, objectives, contents, and when they should be performed during the system development cycle. This regulation also establishes some key milestones for control and its relationships with the reviews.

### 3.3.4 Summary

The USAF has established several standards related to system development, from a software life cycle to data elements, without losing the policy of descentralizing ADPS management, and at the same time providing for sufficient control and review at different levels appropriate to the scope of each system or application.

The success of such a policy is corroborated by the worldwide recognized high performance of the USAF where software development is a forefront technology.

### 3.4 European Organizations

In an attempt to get some information on how software is developed in this area of the world, representatives of German[29], France, Portugal, and Italy were contacted. Problems related with language, time, and distance made this search not as fruitful as it could be. The findings follow:

### 3.4.1 German

The German Air Force does have standards covering system development. The waterfall model is adopted as a standard system life cycle, being composed of: Requirements Definition Phase, Conceptual Phase, Definition Phase, System Development Phase and Implementation Phase.

No standard graphical representation has been established as mandatory, although HIPO is prefered.

PERT/CPM and timetables are the main instruments of planning and control.

There is not a standard concerning programming languages. The most used are PL1, COBOL, and FORTRAN.

### 3.4.2 Portugal

The Portuguese Air Force has established a regulation[27] which covers generically the system life cycle phases as well as planning, organization, control, and documentation.

There are ongoing studies to adopt a structured analysis methodology based on Chris Gane[18] and Tom DeMarco's books[13].

### 3.4.3 European Space Agency

While trying to obtain some methodology or standard established by organizations in France and Italy, a publication called Software Engineering Standards[16], published by the European Space Agency (ESA), was obtained. Such standards were developed by ESA's Board for Software Standardization and Control (BSSC). The standards started as recommendations, presented and discussed with the Agency's software specialists and with representatives of the Aerospace and software industry currently involved in the development of the Agency's projects. The standards also took into account the content of certain documents published by the IEEE Software Engineering Standards Subcommittee[21].

91

One interesting point observed in these standards is
that they were classified in three classes of practices:

1. Mandatory - those applied to all software, either
developed or used by ESA, where the term "shall" is used.

2. Recommended - not mandatory, but strongly
recommended. A justification to the appropriate level in the
Agency's hierarchy is needed if they are not followed. The
term "should" is used in these rules.

3. Guidelines - which are all other items contained in
the documents, after excluding mandatory standards and
recommended practices; guidelines are to be considered only
as useful practices and no justification is required if they
are not followed.

The ESA Software Engineering Standards are structured in
nine chapters. Considering the general similiraties of these
standards with most of what has been presented in this
research, only the most important and peculiar issues that
seem worthwhile to comment on will be addressed below.

1. The Software Life Cycle

The ESA standard adopts a waterfall model as showed
in Fig. 27. A peculiarity of the life cycle is that the
definition of the user requirements is not considered part of
the software life cycle, although constituting a necessary
step before a software project can begin. The software life
cycle begins with the notification and acceptance of  user
requirements.

At the end of each of the first three phases of the software life cycle, the plan foresees a separate activity consisting of review and formal acceptance of the deliverable items of the phase concerned. These activities are indicated by the same abbreviation which distinguish the phase followed by a "/R" shown along the top row.

For each of the phases, Fig. 27 indicates the major activities involved, the deliverable items of the phase with an indication of the items which should be under change control during the life cycle, and the major review and acceptance activities.

| PHASES ITEMS | UR USERS REQUIREMENTS DEFINITION | SR SOFTWARE REQUIREMENTS DEFINITION | SR/R | AD ARCHITECTURAL DESIGN | AD/R | DD SOFTWARE DETAILED DESIGN & PRODUCTION | DD/R | TR TRANSFER | OM OPERATIONS AND MAINTENANCE |
|---|---|---|---|---|---|---|---|---|---|
| MAJOR ACTIVITIES | • IDENTIFICATION OF USERS REQUIREMENTS • FEASIBILITY ASSESSMENT | • IDENTIFICATION OF SOFTWARE REQUIREMENTS • DEVELOPMENT PLAN AND COST ESTIMATES (±30%) • PA | | • DEFINITION OF THE GENERAL ARCHITECTURE OF THE SYSTEM • DETAILED DEVELOP-MENT PLAN & COST ESTIMATES (±12%) • PA | | • DETAILED DESIGN • CODING • UNIT TESTING • SYSTEM TESTING • PA | | • PERFORMANCE OF PROVISIONAL ACCEP-TANCE TESTING • PA | • FINAL ACCEPTANCE • OPERATION OF THE SOFTWARE SYSTEM • MAINTENANCE OF CODE AND DOCUMENTATION • PRODUCTION OF THE SOFTWARE LIFE CYCLE HISTORICAL REPORT • PA |
| DELIVERABLE ITEMS THE → IMPLIES UNDER CHANGE CONTROL | USERS REQUIREMENTS DOCUMENT [URD] | SOFTWARE REQUIREMENTS DOCUMENT [SRD] | | ARCHITECTURAL DESIGN DOCUMENT [ADD] → | | [CODE] DETAILED DESIGN DOCUMENT [DDD] → SOFTWARE USERS MANUAL [SUM] → | | SOFTWARE TRANSFER DOCUMENT [STD] → | PROJECT HISTORY DOCUMENT [PHD] ⋮ |
| REVIEWS | | | ↑ | | ↑ | INTERMEDIATE REVIEWS ⌐─┬──┬─┐ | ↑ | | |
| ACCEPTANCE | | | | | | | | | |
| MAJOR MILESTONES | | USERS REQUEST ACCEPTED ▽ | | SRD APPROVED ▽ | | ADD APPROVED ▽ | CODE/DDD/SUM APPROVED ▽ | PROVISIONAL ACCEPTANCE ▽ FINAL ACCEPTANCE △ | END OF LIFE CYCLE ▽ ISSUE PHD △ |

Figure 27 - ESA's Software Life Cycle[16]

93

Finally, Fig. 27 indicates the major milestones of a software development project from the inception of the development to the end of the life cycle, when the software is dismissed.

2. The Users Requirements Definition Phase

    a. Requirements Specification Languages

The standards suggest that requirements should be written in a natural language because of the advantage of introducing no additional barriers between the people of different disciplines who are involved in the project. ESA does not define "natural language". However, the suggestion is made with the caviat that it may introduce ambiguity, imprecision, and inconsistency to the specifications. Supposing that most of ESA's projects are of the type that requires a high degree of reliability (space activity), and high cost, this recommendation is somewhat of a contradiction with the actual academic thinking that a formal type of definition should be used in such cases[37].

3. The Architectural Design Phase

    a. Decomposition  of the System and Definition
       of High Level Design Standards

While performing these activities the standards recommend the top-down and modularization approaches. To achieve this, they strongly recommend the adoption of a consistent methodology, some of which are listed in one of the annexes. They are: PDL, SADT, Structured Design, High

Order Software, Jackson's, PSL/PSA, Flowcharts, and Warnier's.

The using of the modules relatedness measurements, coupling and cohesion[33] is also recommended.

b. The standards generally recommend the use of a HOL without specifying any particular type.

c. Development Plan and Cost Estimates

According to the standards during the Architectural Design Phase one of the main activities will be to produce a development plan which allows control of the project and which indicates the cost involved. This plan should cover: work breakdown structure, team structure, work schedule, and a milestone chart, which should include, reviews and a planning network, showing the relationships of programming and testing activities.

4. The Software Detailed Design and Production Phase

a. Principles

The standards are based on three known principles:

(1) Top-down construction.

(2) Structured Programming.

(3) Concurrent design, programming, and documentation.

b. Tools and Techniques

Under the above cited principles, some known tools and techniques are recommended to be adopted such as: Project leader, teaming, project librarian, documentation

support, and review sessions. No specific recommendation is made with respect to methodology or graphical representation technique to be used, the only suggestion is to use a "formal" technique intended to promote communication between people.

c. Documentation

Detailed documentation should grow with the system, becoming available in its final form at the end of the phase.

5. Operations and Maintenance Phase

One important aspect of the maintenance phase is the classification of the problem according to degree of regression in the life cycle to fix the error. Following this process, problem repair can be classified as:

a. User misunderstanding.

b. Design does not conform to requirements.

c. Requirements not appropriate.

The standards state that the naming of problem classification is to avoid problems associated with blame for "errors", which only hinders an efficient maintenance process.

Among the outputs from this phase is the Project History Document (PHD) which summarizes the main events and outcome of the project. This becomes a useful tool and guide in the process of estimating effort for future projects, in setting up the organization of a new project, in trying to

avoid repeating mistakes made in the past, and in re-applying
successful methodologies.

### 3.4.4  Summary

Taking in account its recency (1984), completeness, and
currency, the European Space Agency's Software Engineering
Standards can be considered a valid representative of the
current software development stage in Europe. Besides its
complete adherence to the modern principles of Software
Engineering it seems to have solved the problem of
establishing standards without inhibiting the creative
process by establishing levels of observance of the
standards.

Having decided on the Waterfall model as the desirable
approach for a software life cycle, and on Gane and SADT as
the recommended methods, in the next Chapter a literature
review on automated tools for software development will be
performed, and a preliminary  study aiming at a future
implementation in the SIMAER will be done.

## IV.  Automated Software Development Tools

### 4.1 Introduction

The acceptance of the fact that the  design of any reasonably large software system cannot be cost-effectively kept up to date through manual means is spreading. It is widely accepted that some degree of automation is necessary. The greater the use of automation, the higher the probability that system design documents will be kept up to date[28].

In this chapter a literature review of the available automated tools, which in the author's view are the most important, will be performed as a first step for further studies of the feasibility of employing one or more such tools in the SIMAER's software development.

### 4.2 Literature Review

### 4.2.1 Definition

A software development environment is defined as an integrated set of automated and interactive software tools which aid the software engineer in developing quality software products and documentation[19]. The software products and documentation that are developed with the use of a software development environment include requirement definitions, design specifications, source and executable program code, test plans, procedures and results, as well as other associated documentation such as guides and manuals for

operations and maintenance of the software.

Most of the currently available tools, although supporting each phase of the life cycle, are disjoint and often do not interface to tools of the other phases of the life cycle[19]. A description of the most known tools follows.

### 4.2.2 PSL/PSA

The Problem Statement Language (PSL) was developed by Professor Daniel Teichrow at the University of Michigan[35]. The Problem Statement Analyzer (PSA) is the PSL processor. PSL is based on a general model of systems. This model describes a system as a set of objects, where each object may have properties, and each property may have property values. Objects may be interconnected and the connections are called relationships. The general model is specialized to information systems by allowing only a number of predefined objects, properties, and relationships.

The objective of PSL is to permit expression of as much of the information that commonly appears in a Software Requirement Specification as possible. In PSL, system descriptions can be divided into eight major aspects:

1. System input/output flow.
2. System structure.
3. Data structure.
4. Data derivation.

5. System size and volume.

6. System dynamics.

7. System properties.

8. Project management.

PSL contains a number of types of objects and relationships to permit description of these eight aspects. The system input/output flow aspect deals with the interaction between a system and its environment. System structure is concerned with the hierarchies among objects in a system. The data structure aspect includes all the relationships that exist among data used and/or manipulated by a system, as seen by the users of the system. The data derivation aspect of the system specifies which data objects are involved in particular processes in the system. Data derivation describes data relationships that are internal to a system. The system size and volume aspect is concerned with the size of the system and those factors that influence the volume of processing required. The system dynamics aspect of a system description presents the manner in which the system behaves over time. System properties are the objects that compose the system along with its characteristics; PSL allows them to be described. The project management aspect requires that project-related information, as well as product-related information, be provided. This involves identification of the people involved, their responsabilities, schedules, cost estimates, etc.

The Problem Statement Analyzer (PSA) is an automated analyzer for processing requirements stated in PSL. The structure of PSA is illustrated in Fig. 28. PSA operates on a database of information collected from a PSL description.

```
                        COMMANDS
                           IN
                        COMMAND
                        LANGUAGE
                           |
                           v
        +----------------------------------------+
        |      OPERATING          SYSTEM          |
        |                                          |
   STATEMENTS    +----------------------+         REPORTS
   IN THE        |                      |         AND
   PROBLEM  -->  |     PROBLEM          |  -->    MESSAGES
   STATEMENT     |     STATEMENT        |
   LANGUAGE      |     ANALYZER         |
   (PSL)         |                      |
        |        |     (PSA)            |         |
        +--------|                      |---------+
                 +----------------------+
                        |     ^
                        v     |
                    +---------------+
                    |   ANALYZER    |
                    |   DATA        |
                    |   BASE        |
                    +---------------+
```
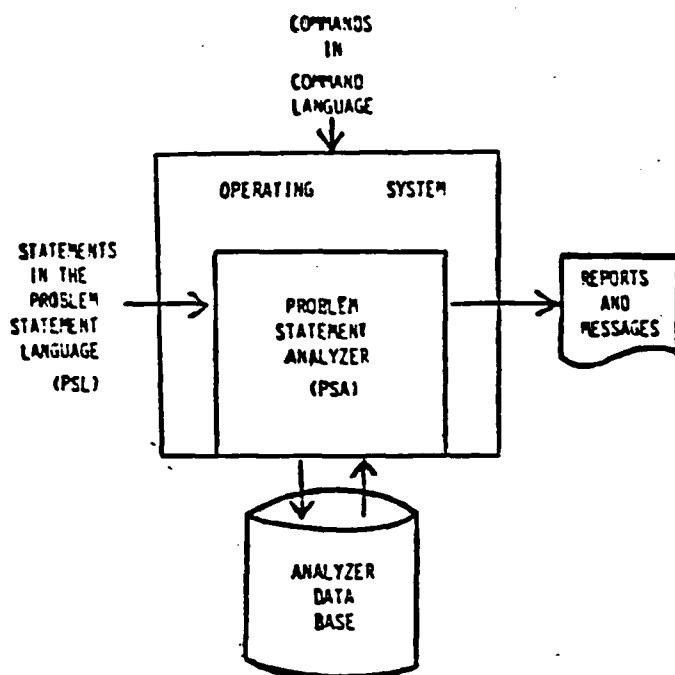
Fig. 28 - The Problem Statement Analyzer

The PSA system can provide reports in four categories: database modification reports, reference reports, summary reports, and analysis reports.

Database modification reports list changes that have been made since the last report, together with diagnostic and warning messages. These reports provide a record of changes

for error correction and recovery. Reference reports include the Name List Report, which lists all the objects in the database with types and dates of last change. The Formatted Problem Statement Report (Fig. 29) shows properties and relationships for a particular object. The Dictionary Report provides a data dictionary.

Summary reports present information collected from several relationships. The Data Base Summary Report provides management information by listing the total number of objects of various types and how much has been said about them. The Structure Report shows complete and partial hierarchies, and the External Picture Report depicts data flows in graphical form.

Analysis reports include the Contents Comparison Report which compares the similarity of inputs and outputs. The Data Processing Interaction Report can be used to detect gaps in information flow and unused data objects. The Processing Chain Report shows the dynamic behavior of the system.

PSL/PSA is a useful tool for documenting and communicating software requirements. According to Fairley[17] PSL/PSA not only supports analysis, but also design. This, in his view, may not be entirely beneficial. It makes it easy for the PSL user to fall into the trap of becoming too involved with design details before high-level requirements are completed.

```
PROCESS                     hourly-employee-processing
   DESCRIPTION:

                            this process performs those actions needed
                            to interpret time cards to produce a pay
                            statement for each hourly employee;

GENERATES:                  pay-statement, error-listing,
                            hourly-employee-report;

RECEIVES:                   time-card;

SUBPARTS ARE:               hourly-paycheck-validation,
                            hourly-emp-update,
                            h-report-entry-generation,
                            hourly-paycheck-production;

PART OF:                    payroll-processing;

DERIVES:                    pay-statement

USING:                      time-card, hourly-employee-record;

DERIVES:                    hourly-employee-report

USING:                      time-card, hourly-employee-record;

DERIVES:                    error-listing

USING:                      time-card, hourly-employee-record;

PROCEDURE:
    1. compute gross pay from time card data.
    2. compute tax from gross pay.
    3. subtract tax from gross pay to obtain net pay.
    4. update hourly employee record accordingly.
    5. update department record accordingly.
    6. generate paycheck.

note: if status code specifies that the employee did not work
this week, no processing will be done for this employee;

HAPPENS:                    number-of-payments TIMES-PER pay period;

TRIGGERED BY:               hourly-emp-processing-event;

TERMINATION-CAUSES:new-employee-processing-event;

SECURITY-IS:                company-only;
```

Figure 29 - Example of a PSL Formatted Problem  Statement[36]

103

PSL/PSA has been used in many different situations ranging from commercial data processing applications to air defense systems.

It is operational on most larger computing environments which support interactive use, including IBM 370 series (OS/VS/TSO/CMS) and CDC 6000/7000 series.

### 4.2.3 RSL/REVS

The Requirements Statement Language (RSL) was developed by the TRW Defense and Space Systems Group to permit automation of specifications for real-time software systems[4]. The Requirements Engineering Validation System (REVS) processes and analyzes RSL statements. Both RSL and REVS are components of the Software Requirements Engineering Methodology (SREM). Many of the concepts in RSL are based on PSL. For example, RSL has four primitive concepts: elements, which name objects; attributes, which describe the characteristics of elements; relationships, which describe binary relations between elements; and structures, which are composed of nodes and processing steps. "Data" is an example of an RSL language element. "Initial Value" is an attribute of the element Data, and Input specifies a relationship between a data item and a processing step.

The fundamental characteristic of RSL is the flow-oriented approach used to describe real-time systems. RSL models the stimulus-reponse nature of process-control

104

systems. Each flow originates with a stimulus and continues to the final response. Specifying requirements in this fashion makes explicit the sequences of processing steps required. A processing step may be accomplished by several different software components, and a software component may incorporate several processing steps. In fact, a sequence of processing steps may involve hardware, software, and people components.

The flow approach also provides for direct testability of requirements. A system can be tested to determine that responses are as specified under various stimuli, and performance characteristics and validation conditions can be associated with particular points in the processing sequence.

Flows are specified in RSL by requirements networks (R-NETS), which have both graphical and textual representations, as illustrated in Figs. 30 and 31.

RSL incorporates a number of predefined element types, relationships, attributes, and structures. Pre-defined elements include Alpha, Data, and R-Net. An Alpha specifies the functional characteristics of a processing step in an R-Net.
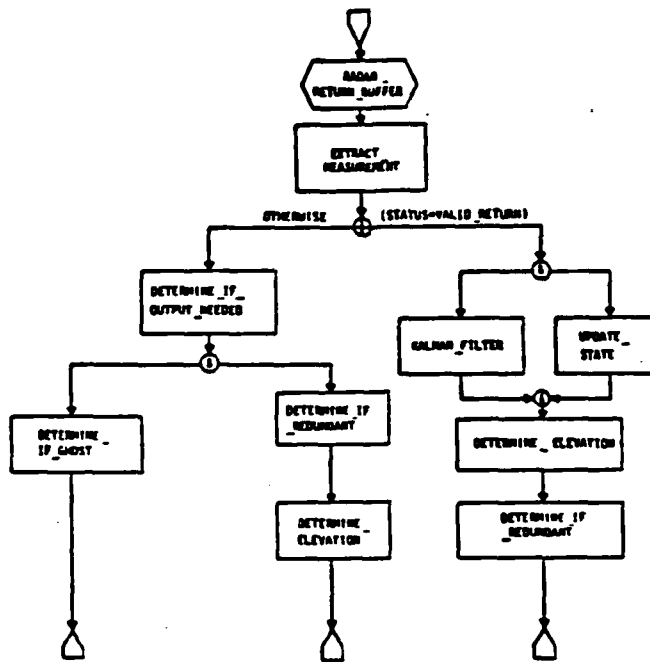
Fig. 30 - Flow Graph of a Sample R_Net

```
R_NET: PROCESS_RADAR_RETURN.
     STRUCTURE:
          INPUT_INTERFACE RADAR_RETURN_BUFFER
          EXTRACT MEASUREMENT
          DO (STATUS = VALID RETURN)
               DO UPDATE_STATE AND KALMAN_FILTER END
               DETERMINE_ELEVATION
               DETERMINE_IF_REDUDANT
               TERMINATE
          OTHERWISE
               DETERMINE_IF OUTPUT NEEDED
               DO  DETERMINE_IF REDUNDANT
                    DETERMINE_ELEVATION
                    TERMINATE
               AND DETERMINE_IF_GHOST
                    TERMINATE
               END
          END
     END
END.
```

Fig. 31 - Sample R_Net in RSL

106

In addition to the predefined elements, types, relationships, and attributes in RSL, new elements, relationships, and attributes can be added to the language using "Define". After the new items have been defined, they can be used to specify other attributes of the system.

The Requirements Engineering and Validation System (REVS) operates on RSL statements. REVS consists of three major components:

     1. A translator for RSL.

     2. A centralized database,the Abstract System Semantic Model (ASSM).

     3. A set of automated tools for processing information in the ASSM.

A schematic diagram of REVS is presented in Fig. 32 below.
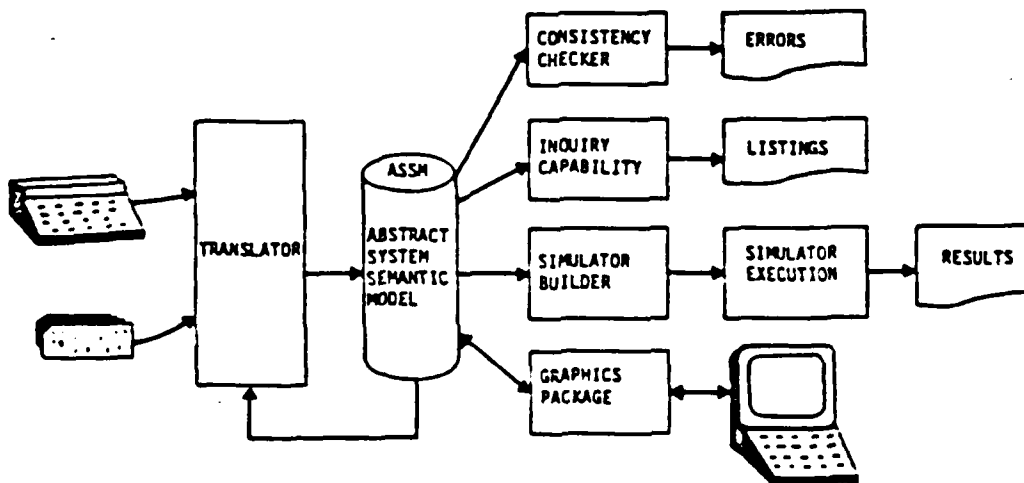


Fig. 32 - REVS' Schematic Diagram

107

The ASSM is a relational database similar in concept to the PSL/PSA database. Automated tools for processing information in the ASSM include an interactive graphics package to aid in specifying flow paths, static checkers that check for completeness and consistency of the information used througout the system, and an automated simulation package that generates and executes simulation models of the system. In addition to the standard displays and reports, REVS provides a capability for defining specific analyses and reports that may be needed on particular projects.

REVS is a large, complex software tool. Use of the REVS system is cost-effective only for specification of large, complex real-time systems.

### 4.2.4 The Software Development Workbench

Currently this development environment is undergoing a great deal of change. Lots of efforts are going on and it is not clear what the final product will be. As a consequence, in this preliminary study only the ones currently available will be addressed.

The Software Development Workbench (SDW) is being developed at AFIT, in concert with and with support from the Air Force Materials Laboratory/Integrated Computer-Aided Manufacturing Division. The SDW has achieved an initial implementation of an integrated software development environment under VAX-11/780[19]. The SDW concept supports

the development and maintenance of software from conception to termination by using automated and interactive tools that enforce the principles of software engineering.

The fundamental characteristic of SDW is the concept of integration of tools that compose the environment. In SDW, integration is realized in two distinct levels. The first level deals with the access and usage mechanisms for the interactive tools, and the second level concerns the preservation of software development data and the relationship between the products of the different software life cycle stages. The first level requires that all of the SDW component tools be resident under one operating system and be accessible through a common user interface. SDW has achieved the first level of integration through the SDW Executive (Fig. 33 and 34). It is the primary interface and controller of the components tools. The second level dictates the need to store development data (requirements specifications, design, code, test plans and procedures, manuals, etc.) in an integrated database that preserves the relationships between the products of the different life cycle stages. SDW achieves this second level integration through the project databases (Fig. 33), which are the integrated data storage areas.

SDW was designed based on five primary objectives:

1. Reduction of software errors.

   This is to be achieved by supporting and enforcing

109

the use of accepted software engineering
principles, as well as by using the computer
to augment different testing procedures.

2. Responsive to changes
   Considering that software is a dynamic entity,
   the SDW must be able to support changing
   requirements for its operations.

3. Rapid assessment of design alternatives through
   the use of simulation models and prototyping.

4. Providing interactive and automated support.
   Emphasizing the production, recording, and
   maintenance of all software development associated
   data.

5. Assisting the  software manager in planning and
   tracking software development efforts.

The current implementation of the SDW is an initial
version composed of software development tools that support
the pre-implementation activities of software  development,
as well as the common capabilities found in most
implementation-oriented development environments such as
editing, compiling, linking, and debugging.

This is the initial implementation of the SDW and a
number of enhancements are planned, such as: extension and
refinement of the SDW tool set to provide a full array of
capabilities to support the entire life cycle; the Pre-Fab

Software Description and Product Data Bases (Fig. 34) will be completely developed and populated to support the pre-fabricated programming concept; finally, the concept of a syntax-directed editor will be extended to a consistency-directed editor.

The SDW is currently installed on both the AFIT Information Systems Laboratory VAX-11/780 and the Central ICAM Development System. According to the designers, users at both installations have found the environment very easy to learn and use. Also, they point out that SDW has proven to be a very effective aid in the development, production, and maintenance of computer software and its related documentation[19].
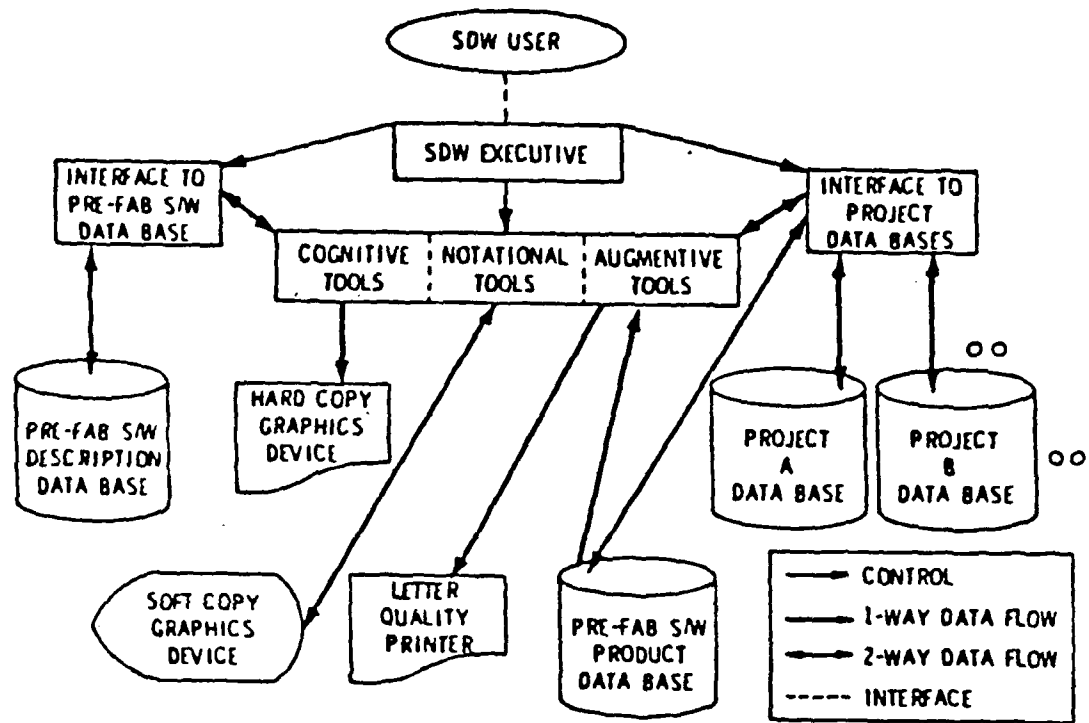
Figure 33 - SDW Configuration Model
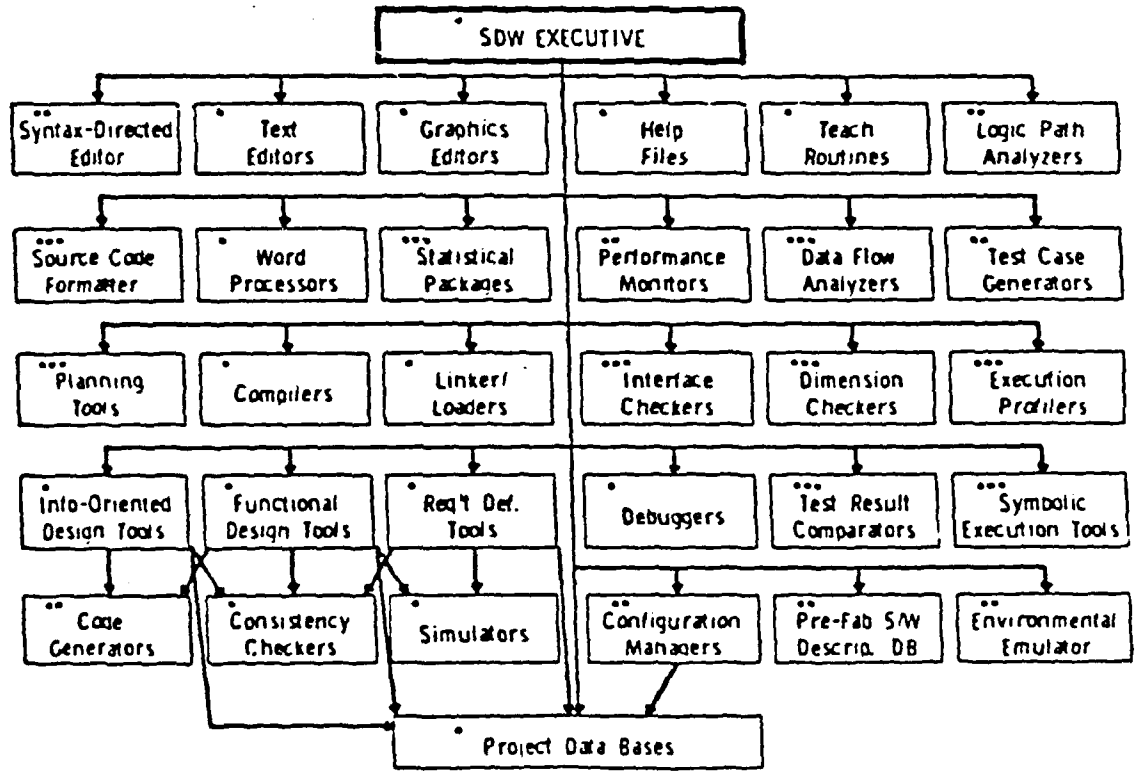
Figure 34 - SDW Structural Model

112

## 4.3 Summary

The analysis presented here is intended to be a first approach to a study for using an automated software development tool in the SIMAER's system development activities.

From this preliminary study it was possible to conclude that PSL/PSA and SDW, covering a broad range of applications are the ones that best match the SIMAER's requirements.

PSL/PSA has the advantage of being able to run on IBM and CDC computers, which are the currently available equipments in the MAer's inventory. On the other hand, SDW has the advantage of covering and joining all phases of the life cycle. Considering the availability of expertise, any study for evaluation of the feasibility of using an automated tool in the SIMAER, should be done at ITA, a similar organization to AFIT. This will facilitate the SDW evaluation, due to the similarity of the missions of the two organizations.

In the previous chapter it was concluded that the waterfall model is the most adequate based on the SIMAER's needs. In the next chapter, a standard software life cycle will be presented.

113

## V. Proposed SIMAER Standard Software Development Methodology

### 5.1 Introduction

It was inferred by the findings in the previous chapters that the SIMAER does not have a standard methodology (methods, tools and techniques plus a software life cycle) to be used in its system's software development. It was also concluded that this is needed in order to reduce the difficulties currently faced in this area of activity in the MAer.

In this chapter an overview of a proposed standard software life cycle aligned with the structured life cycle model will be presented, and some of the discussed methods, tools, and techniques will be recommended to be used in each phase of such life cycle. The general idea is to provide the specialist with a set of tools in such a way that he or she can choose the ones that best apply to a specific situation. The recommendation of using some yet unknown method by the SIMAER may present some drawback in the early stages until that the skill in using each tool is reached by the professionals. However, in the long run, with continued usage, the knowledge acquisition, the standardization, and the skill reached will bring benefits that will compensate the effort.

In chapter III, it was seen that there are lots of models available, and that some organizations similar to the

114

SIMAER have even designed their own. Considering the completeness, currency, and adherence to the principles of software engineering, as well as the broad coverage of ESA's Software Engineering Standards, the proposed methodology will be based in part on those standards. In addition the findings from the literature research and the knowledge acquired with the Software Engineering Lectures[37] will be used.

Aspects related to the previous activities of software development, such as flow and format of documents for request of a design, and channels of command, are considered out of the scope of this research, and will not be addressed here.

This proposal does not intend to be the actual regulation text, but an outline, which includes a short description of the main parts that compose the regulation.

## 5.2  The Proposed SIMAER Software Development Methodology Regulation

### 5.2.1 Structure

Having reviewed many regulations and metodologies, and having chosen a waterfall software life cycle model, the next step was to relate the parts of the methodology with the life cycle model.

The proposed SIMAER Software Development Methodology Regulation is composed of 8 chapters as shown in Fig. 35 below.
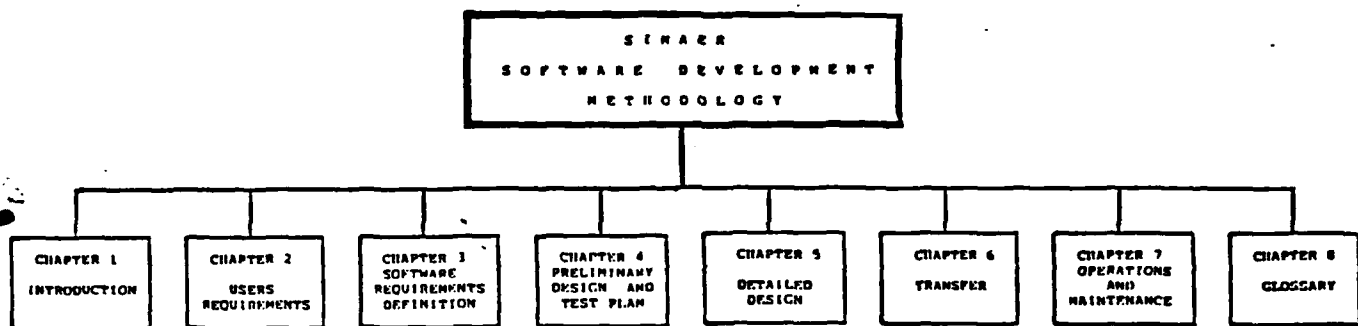


Figure 35 - Structure of SIMAER's Software Development
Methodology Regulation

Chapter 1 defines classes of standards, scope, and purpose of the regulation. In addition, it defines the overall software life cycle.

Chapter 2 through 7 describe the phases as seen in Fig. 35.

Chapter 8 presents a glossary of terms used in the regulation.

### 5.2.2 Classes of Standard Practices

There are 3 different types of standard practices:

1. Mandatory Standards. These apply to all software either developed or used at SIMAER. Two asterisks (**) at the beginning of the paragraph will highlight them in the context.

2. Recomended Practices. These are not mandatory but strongly recommended. A justification to the CINFE is needed if they are not followed. One asterisk (*) at the beginning of the paragraph will highlight them in the context.

3. Guidelines. All the other items contained in the documents, after excluding mandatory standards and recommended practices, are to be considered only as useful practices. No justification is required if they are not followed.

The management responsible can obviously always enforce stricter standards in a software development project, provided that they encompass the standard practices presented here.

### 5.2.3 Scope

### 5.2.3.1 Purpose

This regulation outlines the phases to be followed for software development as well as recomends methods, tools and techniques to be used for software development within the SIMAER. It provides guidance on organizing, planning,

117

developing, and maintaining an ADS project.

## 5.2.3.2 Application

Procedures described in the regulation are to be applied to all MAer activities responsible for planning, designing, developing, maintaining, and managing ADS projects.

The methodology applies regardeless of the size, the type of application (scientific or administrative, real-time or batch, etc), the hardware, basic software and language used, or the nature of the developers (in-house staff or industry). Each of these peculiarities of course has an influence on the way in which the development is performed, and on the formal aspect of the deliverable items described, but conceptually the phases of the life cycle and the related deliverable items described are to be considered valid for any software development project.

## 5.2.4 The Software Life Cycle

## 5.2.4.1 Phases

The SIMAER's Software Life Cycle (Fig. 36) is composed of six phases:

1. Users Requirements Definition (UR)

2. Software Requirements Definition (SR)

3. Preliminary Design and Test Planning (PD)

4. Detailed Design (DD)

5. Transfer (TR)

6. Operations and Maintenance (OM)

118
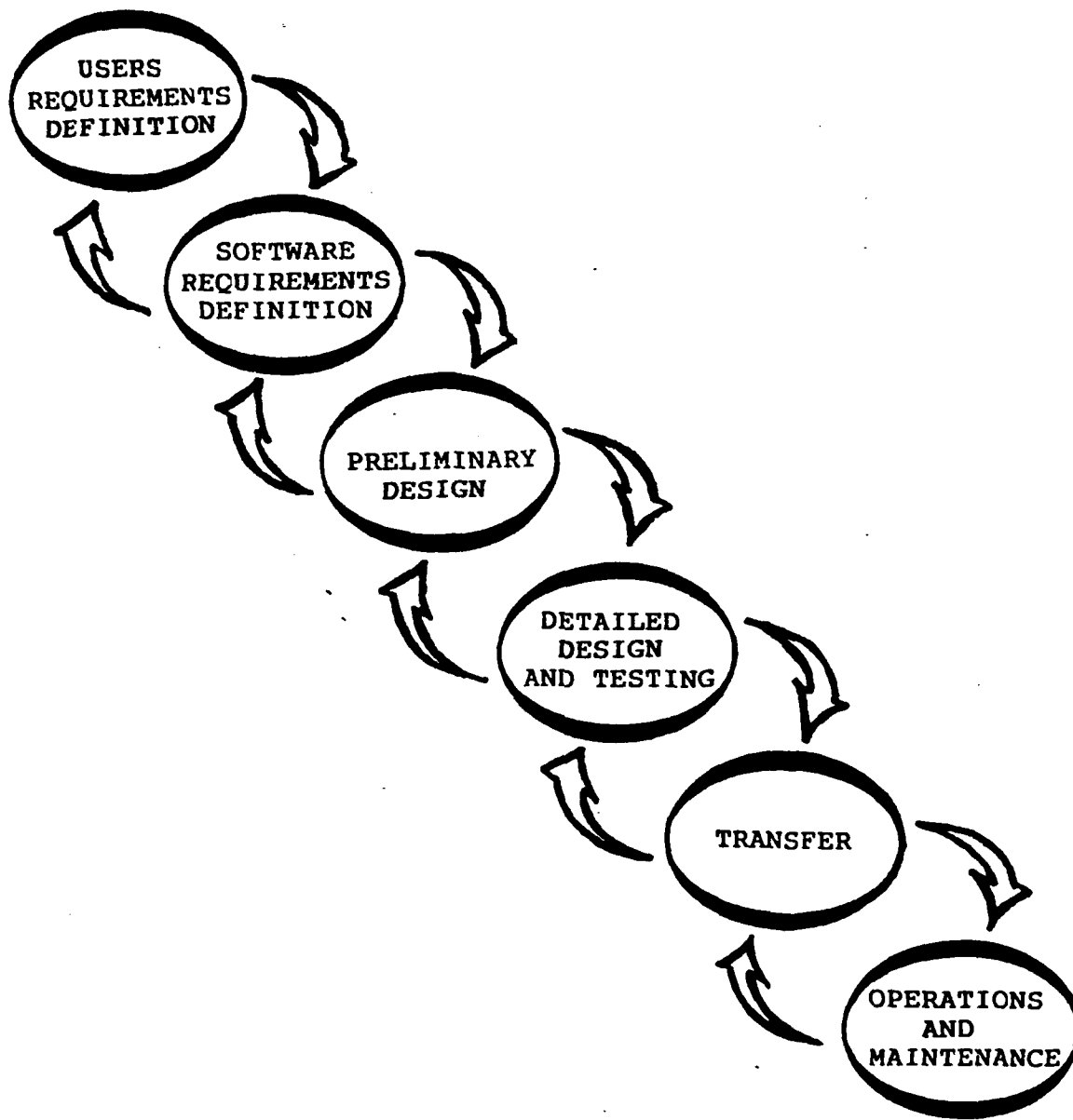
Figure 36 - SIMAER's Software Life Cycle

The software life cycle begins with the definition of a

users requirements.

At the end of the UR, SR, PD, and DD, there is a

separate activity consisting of the review and the formal

acceptance of the deliverable items of the phase concerned.

These activities are indicated by the same abreviation which

distinguishes the phase followed by a "/R" (for review,e.g.

DD/R is the review activity for the DD phase).

For each of the phases, Fig. 37 indicates the major

Figure 37 - SIMAER's Software Life Cycle Management Scheme

120

activities involved, the deliverable items of the phase with the major review and acceptance activities and, the method, tools, and techniques recommended to be used in the phase.

* Fig. 37 also shows the major milestones of a software development project between the inception of the development and the end of the life cycle, when the software is dismissed. The major milestones, which should always be present, even in a small project, to allow the progress of development to be monitored are:

1. the acceptance of the user's request.

2. the approval of the Software  Requirements Document (SRD)

3. the approval of the Preliminary Design Document (PDD) and the Test Plan Document (TPD).

4. the statement of readiness for provisional acceptance testing, i.e. the acceptance of the Detailed Design Document (DDD), of the Users Manual (UM), and of the code for provisional acceptance testing.

5. the statement of provisional acceptance.

6. the statement of final acceptance.

7. the issue of the Project History Document (PHD).

It should be noted that while all the other major milestones fall at predefined moments of the life cycle, the moment of the final acceptance can be determined by the final acceptance criteria. Also the moment of the issue of the project history document is not fixed, but a good practice

121

would be to issue the experience recorded during the
development immediately after the final acceptance and
complete it at the end of the life cycle.

1 . Users Requirements Definition (UR)

a. Introduction

In this phase the initiator begins with a general idea
of a task to be perfomed using computing equipment, and
refines this general requirement into a definition of what is
expected from the computer system and by what means its
correctness and acceptability can be assessed.

b. Inputs to the phase

The functional manager submits a request document, which
describes and justifies the "needs" for a system to do a
certain function or to carry out a certain operation.

c. Major Activities

** Users Requirements should never be expressed in terms of
implementation details and design of the software. On the
other hand, it is frequently necessary to consider
implementation possibilities and implications before
achieving a final requirements document. This will normally
entail discussions between the initiator of the requirement
and qualified software experts, as well as other interested
parties such as  operations personnel. However, the Users-
Requirements shall stand as a logical and complete
specification after such considerations are removed. This
iterative procedure for the definition of the Users

Requirements goes hand in hand with the study and
confirmation of feasibility and eventually the harmonization
of the new requirements with existing software.

(1) Determination of Operational Environment

Expression of users requirements goes together with a
statement of boundary conditions, or definitions of the
environment in which the software ultimately has to operate.
In the case of software to be run on an existing
installation, this may be readily available, but in the event
of a combined hardware and software procurement, the original
requirement may allow some functions to be achieved either in
hardware or in software and thus calls for a choice among a
wide variety of system architectures. In such a case it is
necessary to carry out studies to define the general system
design and to identify the parts of the system which will
require software packages. Each system component must be
sufficiently identified to enable the production of a
specific SRD for that component, and this SRD must include the
details of the interface with the rest of the system and the
necessary communication protocols. While  users will not
complete this process they should provide all available
relevant information to assist the subsequent preparation of
detailed system and software requirements.

(2) Classification of Requirements

** Having considered the environment in which the software
will be operated, the user shall proceed to classify his

123

requirements into those features which are essential and those which are merely desirable. If a feature is described as essential, then there shall be a test proposed to determine whether the resulting software is acceptable. Statements of the kind, "it is essential that the system be optimized for speed of execution", are not admissible. Instead, a means must be stated of determining whether the operational speed achieved is sufficient.

(3) Man/Machine Interface

This category of requirement will vary in importance according to the type of system under consideration. In some cases it will be sufficient to indicate which parameters may need to be varied and which alternatives may be required for the output medium, but in real-time systems considerable work may be needed to define procedures for data input and system control as well as for data presentation and archiving of data. This may include the definition of a comand language and interactive dialogue.

(4) Feasibilty Assessment

Feasibility will have to be checked in respect to:

(a) Available memory

(b) Real-time perfomance

(c) Suitability of programming environment both hardware and software.

(d) Availability of resources for software requirements definition, design, and programming.

124

\*       Feasibility studies will need to be undertaken in
every case in which the above points cannot be confirmed
within adequate boundries by direct enquiry. Users should
give any available information which helps to determine which
points may be critical as well as details which may aid the
ensuing studi?s.

(5) Management Information

Information will be required at an early date regarding
resources, cost, and schedule in order to enable management to
authorize work to proceed. This will include:

(a) budgetary estimates for the total development

(b) identification of resources required

(c) provisional schedule for completion of work

While users may not be in a position to furnish this
data as part of their requirements, they should give any
available information to assist the task of making these
estimates.

d. Outputs from the phase

(1) Users Requirements Document (URD).

This document includes:

(a) Essential Requirements

\*\*       These shall be stated in such a way as to
indicate how the resulting software may be demonstrated.

(b) Desirable Requirements

\*       These should have included a weighting
indicating degree of desirability.

(c) Man/Machine Interface

**         This section will vary in importance according
to the type of software, but shall always be included.

(d) Operational Environment

- Comments concerning hardware on which the
software will have to operate, and hardware on which software
will be designed, coded, and tested.

- Statement of related tasks which may affect the
applicability of local standards and the re-use of existing
software.

(e) Feasibility

This section contains users' inputs identifying
areas in which feasibility may require study, and reference
material which may be useful in the determination of
feasibility in any of the categories mentioned in the
feasibility assessment.

(f) Management Information

*         Preliminary information should be given to
assist in prcducing estimates of time and cost for the
Software Requirements, Preliminary Design, and Detailed
Design Phases, and for total software costs tc delivery.
Dates on which the working software system will be required
should also be given where approppriate.

e. Methods, Tools, and Techniques

The recommended methods, tools, and techniques for
various types of design projects include:

126

Gane - for data processing systems;

SADT - for real-time and embedded systems;

LDFD - for data flow representation;

Chen Entity-relationship - for database systems
representation;

Data Dictionary - for documentation;

PERT/CPM , GANTT charts and status Report for planning
and development control.


2. Software Requirements Definition (SR)

   a. Introduction

     The objective of this phase is to come to a complete,
validated specification of the required functions,
interfaces, and performance for the software product.

\*      This phase should establish what should be done and not
how it is done.

   b. Inputs to the Phase

     The input to this phase is the Users Requirement
Document, supplemented by any further information.

   c. Major Activities

    (1) Identification of software Requirements

     In this step the requirements are collected. To help
this process some types are established, such as: Functional
requirements, perfomance requirements, interface
requirements, operational requirements, resource
requirements, safety requirements, reliabilty requirements,

127

and maintainability requirements.

Apart from these types of requirements, there are other requirements based on economic considerations and scheduling constraints. These too will reflect back into the above mentioned types of requirements and detailed trade-offs have to be made in almost all cases.

Other considerations related to requirements include the establishing of attributes to allow guidance of requirements in such a way to avoid ambiguitity, make them complete, consistent, and testable.

(2) List of Acceptance Tests

** During this phase a list of acceptance tests shall always be collected. This will be used to generate the software Test Plan Document (TPD).

(3) Cost and Schedule Estimates

At this stage of the software development cycle no design yet exists and cost and schedule estimates cannot be based on number of modules or on number of lines to be coded. Therefore, the estimates can be based on comparison with similar systems or use parametric models based on analogy base database.

d. Review

** The SRD shall be formally reviewed through the software Requirements Review (SR/R). Participation should include the user, the operations personnel, the developers (hardware engineers and software designers), and the managers

128

concerned.

The Software Requirements Definition Phase terminates with the formal approval of the updated SRD after the SR/R.

e. Output from the Phase

** The deliverable items which constitute the output from this phase shall be the Software Requirements Documents (SRD)

** The SRD shall always be produced for any software project.

* In terms of software requirements this document should be independent of any implementation detail. In other words, at this stage, the project should still be open to various and distinctly different possible architetures and implementations. The analysts and designers will have to evaluate them in the following phase and finally select one. The SRD should include a development plan and cost estimates.

3. Preliminary Design and Testing Plan (PD)

a. Introduction

The aim of this phase is to design the general architecture of the system fulfilling the requirements laid down in the SRD and, to detail the implementation plan in response to the SRD. The system design should be represented as the composition of the solution to subproblems displayed in a hierarchical structure of components. The preliminary design is complete when the project leader can split the subsequent project work between teams or individual team

129

members. The individuals should then be able to continue with the Detailed Design phase, working almost independent of each other and using the interface definition given in the Preliminary Design Phase.

This phase may involve several iterations based on alternative assumptions. Particularly in the case of important, critical, or highly interactive systems, the implementation of prototype software to verify the correctness and the impact of the basic assumptions should be considered.

Another objective of this phase is to elaborate the Test Plan which is based in the requirements definition and specifies the test conditions for acceptance testing of a computer program.

b. Inputs to the Phase

The input to the Preliminary Design Phase is the Software Requirements Document (SRD).

c. Major Activities

During this phase the following activities shall be performed:

(1) Decomposition of the System;

(2) Functional definition of the components;

(3) Definition of the data structures;

(4) Computer resource utilizitation study;

(5) Test Plan;

(6) Development Plan and Cost Estimates;

130

(7) Choice of programming language.

* The choice of the programming language should be done at the very end of this phase.

d. Review

* The PDD should be reviewed formally by the users, computer hardware and software designers, and by the managers concerned during the Preliminary Design Review (PD/R). Its approval constitutes one of the major milestones of the project. The TPD should also be reviewed at this point.

e. Outputs from the phase

** The formal outputs of the PD phase shall be the preliminary Design Document (PDD) and the Test Plan Document (TPD).

* The PDD should include a detailed development plan and cost estimates for the DD, TR, and the OM phases.

f. Methods, Tools, and Techniques

Top-down and modularity should be the approach used in this phase.

The recommended methods, tools, and techniques for this phase include:

Gane - for data processing systems.

SADT - for real-time and embedded systems.

LDFD - for data flow representation.

Structure Charts - for preliminary design.

Chen Entity - for database systems representation.

Data dictionary - for documentation.

*     The programming language, HOL, should be chosen at this point. Assembler languages should be selected only for very specific and justified reasons.

4. Detailed Design (DD)

    a. Introduction

        In this phase the main components of the software system are defined. They can be baselined and the project can proceed to the detailed design of software.

*       After several iterations in the SR and PD phases, the requirements should be, a this stage of the software life cycle, completely defined and baselined and the architecture of the system definitely designed in terms of hardware and software structure.

        During the DD phase the lower level components defined in the PD phase are further decomposed until reaching the module level. Modules are then designed, coded and module tested by individual team members. As each team member declares himself satisfied with any particular module, he passes it to the project librarian for inclusion in a "Tested Module Library". From this library, verification personnel select modules to build into System Versions to be verified. As particular Systems Versions are verified they are passed to the project librarian for inclusion in a "Verified Version Library".

        It should be noted that any change in requirements or in the architecture of the system while, still possible up to

this stage, become increasingly difficult and expensive to deal with from now on. It is therefore extremely important not to start this phase if there are still doubts, major open points, or uncertainties in the requirements or the architectural design.

There is no point in starting code and testing activities if the computer, the operating system, and the system software are not available and sufficiently reliable and stable.

b. Inputs to the Phase

Inputs to this phase are the Software Requirements Document (SRD), the Preliminary Design Document (PDD), and the Test Plan Document (TPD).

c. Major Activities

The SIMAER methodology is based on the three following principles: top-down construction, structured programming, and concurrent design, programming, and documentation.

The activities in the DD phase are driven by these principles, which affect both the organization of the work and its actual implementation.

(1) Organization

The functions described hereafter are necessary irrespective of the size of the system to be implemented and of the team organization involved in the implementation.

The functions may all be performed by one person in

133

small systems or be assigned to different members of the team in larger systems.

(a) Project Leader

The project leader should have a good understanding of all the parts of the system, of the external interfaces, and of all the internal interfaces between the various sub-systems. The project leader should take care of the relationships with the external world, and organize, plan, and control the activity of the team.

(b) Team Leaders

Teams of 3-4 persons should be composed. Each team should be concerned with one or more sub-systems and should have a team leader.

(c) Project Librarian

Essential steps to ensure the quality of the software are the reviews carried out at various stages of the development and version control of the code and documentation produced. The coordination of the reviews, i.e. the distribution of the documents, collection of the comments, preparation and distribution of the change notices, and the version control should be assigned to the librarian.

The librarian should be given responsibility to maintain the master copies of all design documents, source and object program files, source listings, and load module files. Any update to any of these files should be done exclusively by the librarian according to project

134

configuration control procedures. Other members of the team may copy elements from the various files but are not allowed to update any element of the project files.

(d) Documentation Support

Another centralized function is the documentation support.

** A project shall have appropriate secretarial support to produce the documentation required.

A typical structure for a development team with 3 or 4 subsystems could be of the type shown in Fig. 38.

For larger or smaller projects, the structure may have to be modified as a consequence of the number of people involved.

It is a good practice for each team leader to be capable of backing-up the team members, at least for critical functions. The same applies to the project leader with respect to the management functions of each of the team leaders.

Once the organization of the development team is defined, the project leader should proceed to define the design, coding standards, the naming conventions, the error handling procedures, and the development and operational procedures.

```
        --------------------
        |      Project     |
        |      Leader      |
        --------------------
                  |
---------------------         |
|Project Librarian|         |
|amd Documentation|---------|
|Support Functions|         |
---------------------         |
        _____|_____
        |                     |                          |
------------------   ------------------    ------------------
|     Team       |   |     Team       |    |     Team       |
|    Leader      |   |    Leader      |    |    Leader      |
|                |   |                |    |                |
| Sub-System 1   |   | Sub-System i   |    | Sub-System n   |
------------------   ------------------    ------------------
        |                    |                      |
     ----------           ----------             ----------
     |Members |           |Members |             |Members |
     |  of    |           |  of    |             |  of    |
     | Team   |           | Team   |             | Team   |
     |  1     |           |  i     |             |  n     |
     |_____|           |_____|             |_____|
```
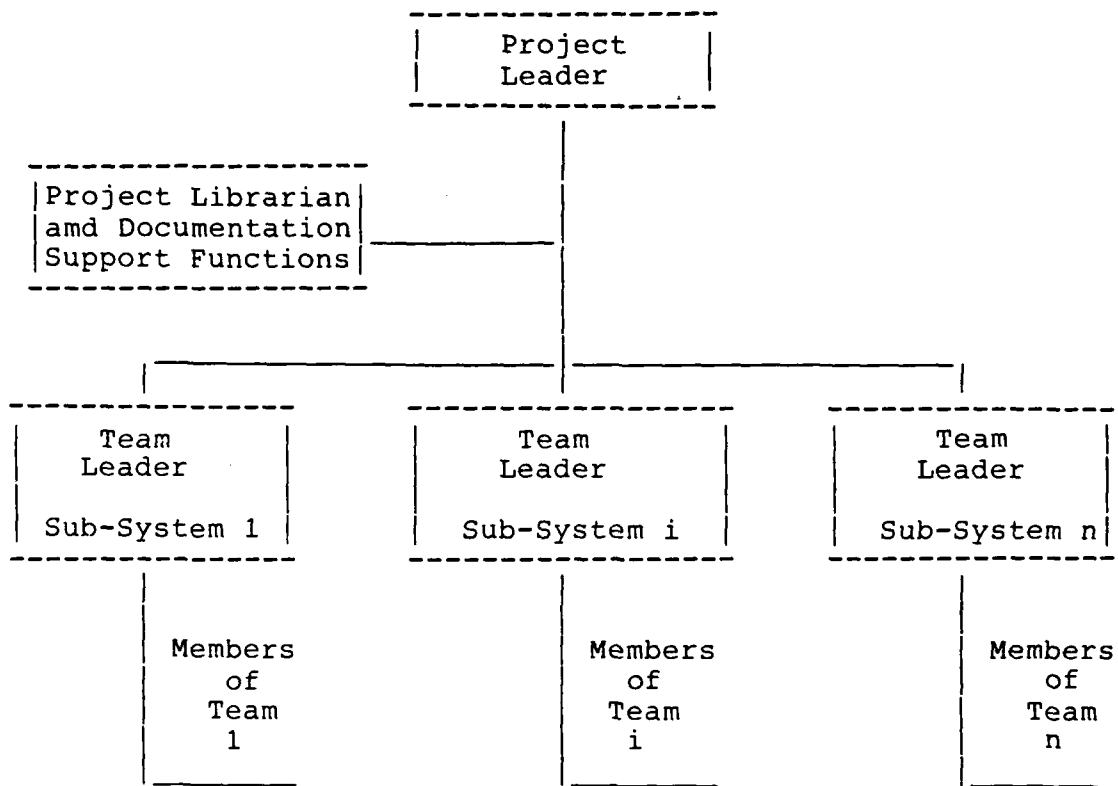
Figure 38 - A Typical Structure of a Development Team

(2) Implementation

The PDD defines the function of the various sub-systems. Starting from this definition, the design proceeds to lower levels.

The break-down process proceeds from the top. It should be noted that, although the design should rigorously proceed from the top levels down to the lowest, there are some activities in a software development which are better accomplished by not following this approach. Typically service packages such as high level device drivers, or access methods or routines, which are used by various members of the

136

development team, ought to be implemented first. This is also useful in view of gaining experience with the programming environment before starting a massive coding activity.

* Each level of design should be formally reviewed and approved before proceeding deeply into the next level of ³ɘsign.

** The structure of the design shall always be reflected in the identification system of the components and in the structure of the Detailed Design Document (DDD). In other words, the content of the DDD shall have a one to one correspondence with the levels and components into which the system is broken down.

(3) Coding

* As the design of each module is completed, reviewed, and approved, the module can be coded following the coding conventions. These conventions should include in particular:

(a) coding standards for ˋl the languages to be used

(b) rules for definition of constants

(c) rules for common code

(d) rules for insertion of comments, including references to the DDD

(e) naming conventions for programs, sub-programs, files, variables, data.

** In particular, each module shall always have a standard header with essential information (at a minimum,

identification number, title, function, date-coded, last
update and any other information as decided by the project
management, will be included). It is a good practice to have
this standard header in a file in a suitable form to be
edited, completed, and then inserted at the head of each
module.

\*      Once the coding is finished, the module should be tested
by the author to show that it performs correctly all the
tasks specified. Data and results of each test should be kept
for further tests and comparison whenever a module has to be
modified.

\*      Module design documentation should be produced
concurrently with detailed module specifications, module
coding, and testing. In other words the Detailed Design
Document (DDD) grows with the system and it becomes available
in its final form at the end of the phase.

A fully tested module should be passed to the project
librarian in the form of code and related documentation to be
inserted formally in the tested module library.

Upon delivery of a module the librarian should formally
check its compliance with coding and documentation standards. .

\*\*      For progress control purposes, a module shall only be
considered completed when it has been formally accepted by
the project leader or person designated by him.

(4) System Integration and Verification

Following the unit testing and the implementation of

138

special software and data sets which have been identified in earlier phases in preparation for the Transfer Phase, modules are integrated into sub-systems, and eventually into a complete working system. The results of the module tests and verification are collected in a test file which is made available for review during the Transfer Phase. When substantial subs-systems are completed and verified, evaluation of the correctness of many of the decisions made in the requirements and design phases can be made.

(a) Integration

Where possible, integration should proceed in a top-down, function by function sequence. This means that the complete system should be integrated at the highest modular level using "stubs" to represent lower level module. As modules are completed they replace the stubs. This approach lessens the impact of problems associated with system generation and configuration. It is useful to use the project librarian as part of the integration team to generate the system, since this minimizes configuration identification problems.

Implementation of a system function by function allows end users to gain experience with significant parts of the system at the earliest possible date. This in turn increases management confidence that the project is progressing satisfactorily.

(b) Verification

## Purpose of Verification

In all but the smallest systems, it is unrealistic to
expect that a set of tests can be carried out in a few hours
or days and give a complete verification of system
performance.
Verification should start as soon a system version with at
least one verifiable function becomes available. Sufficient
time should be allocated to this activity.

The purpose of module testing is to demonstrate that the
individual modules meet the design specification. The purpose
of verification is to show that:

(1) Modules work together in the manner foreseen in the
design specification. This aspect may often be left to module
design, code and test personnel.

(2) The as-built software satisfies all the formal
requirements as expressed in the software requirements
documents.

(3) The User's Manual and the software agree.
This aspect of verification is often ignored because of
poor understanding, thus resulting in bad relations with the
end users.

## Organization of Verification Personnel

For large systems it is highly recommended that the
personnel who perform system verification be independent of

140

those responsible for design, code, and unit test, at least within the software project. This technique is useful to avoid the mental distortion of requirements and operational interfaces which is inevitable from personnel who have been working closely with the code for months, or even years. In particular, the alternative point of view brought to the product will illuminate areas which the system designers have thought unimportant, or even in the worse, ignored.

(c) Preparation for the Acceptance Testing

The Test Plan Document (TPD), which has been prepared in outline form during the PD phase, is completed by the addition of full operating instructions and data references needed to carry out the acceptance test procedures.

d. Methods, Tools, and Techniques

The recommended methods, tools and techniques for this phase are:

Gane - for data processing systems;

SADT - for real-time and embedded systems;

LDFD - for data flow representation;

Chen Entity-relationship for database systems representation;

Data Dictionary - for documentation;

Pseudocode - for representing the logic of modules;

HOLs that allow for structured programming;

Classical teaming -  for personnel allocation; and

141

PERT/CPM, GANTT Charts for development control.

e. Review

*       In the DD phase, the design proceeds in a top-down manner. Whenever the design of one level is completed, there should be a formal review of it.

The purpose of these reviews, to be held at each level of the system, is to verify that the design of the level being discussed is correct, and its documentation contains sufficient information to proceed to the implementation of the components belonging to that level, and to the design of the lower level.

*       The project leader should participate in these reviews, together with the team leader and members of the subsystem teams concerned.

*       In addition, each team leader should organize internal walk-throughs to check module specifications and code.

**       The code, tested at module level and verified at sub-system levels, the DDD, and the UM, in their final versions, shall be subject to a final review (DD/R). Upon satisfactory completion on this final review the system can be declared ready for provisional acceptance to be performed in the Transfer (TR) phase.

This is the major milestone which concludes the DD phase.

e. Outputs from the Phase

The outputs from the phase shall be the Code, DDD and

UM .

(1) Code

*    Each verified system version, including
procedures, is delivered to the project librarian, who puts
them into the verified version library. There may be various
versions of the system available at certain times before the
system enters into operation, and the project librarian
should control all of them, i.e. it should always be possible
to establish a relation between a version and a particular
set of requirements.

(2) Detailed Design Document (DDD) and Users Manual
(UM)

The DDD and the UM are evolving documents. They
initially contain the sections corresponding to the top
levels of the system. As the design and code proceeds down
through increasing levels of detail, the related sections are
added.

The DDD should contain the following information:
project standards, conventions, procedures, detailed design
specifications, etc.

The UM contains information needed for the users
of the system, and the information needed to operate the
system. The latter sometimes is called the Operator's Manual.
The two categories, users and operators, may coincide in some
systems, but the two distinctions should exist in any case.

143

5. Tranfer Phase (TR)

    a. Introduction

\*         The main purpose of this phase is to establish that the system fulfills the requirements laid down in the SRD. Tests are performed according to a test plan, and should also include a check of the quality of the programming and software documentation. The test plan should have been prepared during the DD phase. The test plan and the results of the tests should be compiled in the Software Transfer Document (STD).

        Since acceptance tests are based on the users' requirements, there are a wide variety of tests possible. It is the responsibility of the author of the user requirement document to ensure that it will be possible to ascertain the correctness of the final product and to lay down the principles by which this will be achieved during the Users Requirements Definition (UR) phase.

    b. Inputs to the Phase

    (1) Description of the principles on which the acceptance tests are based.

    (2) The TPD including:

    (a) The definition of the hardware on which the acceptance tests are to be run.

    (b) The full list of tests to be run, i.e. the tests procedures.

    (c) The software test files covering the results

144

of module test and verification.

    c. Major Activities

        (1) Performance of provisional acceptance testing.

        (2) Correction of errors found in the acceptance testing.

        (3) Record of reception of all deliverable items in the Software Transfer Document (STD).

    d. Outputs from the Phase

        (1) Statement of provisional acceptance.

        (2) The provisionally accepted software system on computer support ready for submission to operations staff for final acceptance.

        (3) The Software Transfer Document (STD).

6. *The Operations and Maintenance Phase (OM)*

    a. Introduction

Once the system has been provisionally accepted and is entered provisionally into operation, it should still undergo a final acceptance test with real data, to demonstrate that it meets the reliability and availability requirements defined in the SRD.

The period during which the final acceptance testing is made is called the validation period. It begins immediately after the provisional acceptance has been pronounced and finishes when the system can be demonstrated to run stably with a defined minimum level of performance.

At a certain moment in this phase, related to how the

reliability and availability requirements are defined, the system can be declared finally accepted.

   b. Inputs to the Phase

   Input to the phase is a full set of the following documents (where applicable) together with at least one version of the product which they describe:

      (1) Users Requirements Document (URD);

      (2) Software Requirements Document (SRD);

      (3) Preliminary Design Document (PDD);

      (4) Detailed Design Document (DDD);

      (5) Users Manual/Operators Manual (UM); and

      (6) Software Transfer Document (STD).

The Software Transfer Document (STD) will record all tests carried out on the product which have led to provisional acceptance.

   Provisional acceptance is the formal milestone marking the start of the Operations and Maintenance Phase.

   c. Major Activities

      (1) Maintenance

** Every software product which has not been dismissed shall have at least one person designated as maintenance programmer.

** All projects shall follow a formal, written problem identification and working procedure. A normal medium for establishing such a procedure would be a software problem report (SPR). The objective of the SPR is to identify the

problem and if necessary to initiate the mechanisms for updating the documentation.

The maintenance activity can cause regression to any phase of the life cycle. For example, a problem may be caused by a failure to implement the software detailed design in the code, it may be a new user requirement, or it may simply be a misunderstanding on the part of the user.

The maintenance organization shall classify problem reports according to the degree of regression in the life cycle. This in turn implies which documents need to be changed, and what testing needs to be performed.

(2) Operations

Reponsibility for operation of the product usually. lies outside the organization responsible for the product development.

Following problem repair, software shall be re-released to operational use.

d. Outputs from the Phase

Outputs produced during the phase are:

(1) A final acceptance certificate;

(2) A Project History Document;

(3) One or more sets of software documentation relating to the current versions/releases;

(4) One or more set of source and binary code corresponding to the released versions of the product; and

(5) A record of problem identification and change

147

activity with respect to the product.

### 5.3 Implementation Plan

The implementation of this methodology should be executed in four steps. First, the CINFE should review this proposal, next the two most experienced organizations in software development within the SIMAER, CCA-RJ for management information systems, and ITA for embedded and real-time systems, will train their specialists in the standard software life cycle and in the recommendend methods and tools for six months. Such training should be done not only at a theoretical level, but also at practical level by means of actually developing a system. After that will come an evaluation period and, eventually, the necessary corrections. Once the methodology is corrected and/or improved, there should follow a gradual and regional extension of training and regular utilization by the entire SIMAER.

A tentative implementation plan schedule could be the one shown in Fig. 39.

### 5.4 Cost

Most of the cost for the implementation of this methodology will be absorbed in the regular personnel wages. However, some indirect cost, such as the time spent to learn a yet unknown method and tools, like SADT, should be considered, although difficult to quantify. On the other hand, one cannot lose sight of the consequent benefits once

the methodology is learned and used all over the SIMAER.

| PERIOD ORGANIZATION | 8 6 | | | | | | | | | | | | 8 7 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OUT | NOV | DEC |
| CINFE | | • | • | | | | | | | | | | | | | | | | | | | | | |
| CCA-RJ | | | | # | # | # | # | # | # | # | e | e | X | X | | | | | | | | | | |
| ITA | | | | # | # | # | # | # | # | # | e | e | X | X | | | | | | | | | | |
| DEPV | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| DAC | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| DIRAP | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| DIRINT | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| DIRENG | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| DIRMA | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| CCA-BR | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| CPO | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| CISA | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| CINDACTA | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| CENIPA | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| COMGAR | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| COPAC | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| SEFA | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |
| AFA | | | | | | | | | | | | | | | # | # | S | S | S | S | S | S | S | S |

LEGEND

••• - Review
### - Training
eee - Evaluation
XXX - Corrections / Improvements
$$$ - Regular Use

Figure 39 - SIMAER's Software Life Cycle Implementation Plan

## 5.5 Conclusion

In this chapter a standard methodology for the SIMAER software development was proposed. Peters[28] points that methods are important, but their successful application occurs only in supportive environments. The proposed methodology is patterned after the ESA approach, which

149

establishes mandatory actions, recommendations, and guidelines. This should allow maximum flexibility for the methodology, and will hopefully help establish a supportive environment in which the methodology can evolve.

In the next chapter some actions that help to establish this supportive environment are recommended.

## VI. Conclusions and Recommendations

### 6.1 Conclusion

After more than six months of research, defining the environment, searching for the requirements, diagnosing the most common problems, studying and comparing the most common methods and tools designed by the academecians and used by several organizations, a SIMAER's Software Development Methodology Regulation was created and proposed. Its acceptance and effective utilization, will require some special ways of doing things, as well as some complementary activities. These are listed as recommendations.

### 6.2 Recommendations

This research did not intend to be exhaustive. At the same time, some complementary issues that should be a matter of CINFE's concern were found. They are:

1. Research should continue beyond this thesis effort in order to:

(a) Select and implement interactive, automated software development tools for the SIMAER.

(b) Select and adopt standard HOLs that allow for structured programming to be used within the SIMAER.

2. The SIMAER should:

(a) Select and train its specialists in software cost estimation methods as appropriate.

(b) Create a board for software standardization and

control.

(c) Evaluate this methodology and hopefully adopt it as a standard for the SIMAER.

(d) Update the current regulations related to software development requests and approval to conform with the new standard to be adopted.

(e) Establish a policy for microcomputer selection, acquisition, and use within the MAer.

It is a well known fact that software development is an activity hard to grasp. The main purpose of this work is to make it less painful by giving training, combining the efforts, and sharing the resources and knowledge through the establishing of standard policies in the ADP activities within the SIMAER.

152

## Appendix A

CURRENT SOFTWARE DEVELOPMENT SITUATION IN THE BRAZILIAN AIR
MINISTRY: A SURVEY OF  METHODOLOGY,  DOCUMENTATION, GRAPHICAL
REPRESENTATION, MANAGEMENT, PROGRAMMING, TESTING, MAINTENANCE
AND SUGGESTIONS

MAY 85

153

FOREWORD

This appendix presents the results of a survey of
seventy-nine MAer ADP professionals assigned to seventeen
*SIMAER* organizations. It includes information about software
development from management, planning, and control to
execution.

My sincere thanks are due to the busy respondents for
taking the time to complete the questionnaire and for the
many comments that they added. The information and advice
that they provided should be most helpful to my work.

Aparecido F.de Oliveira

INDEX

# THE SURVEY

## I. Introduction

As stated before, and based on past experience, it seemed that few organizations in the MAer had a standard methodology or used modern programming techniques for software development. In order to figure out how ADS were actually designed in the MAer, and to gather suggestions to design a standard methodology, a survey was done.

## II. Letter of Transmittal

Each questionnaire was accompanied by a letter of transmittal. This letter defined the purpose of the survey and requested the individual's collaboration in answering the questions.

## III. Population Polled

Cost, distance, and time considerations, allied with previous knowledge, lead to the choice of the subjective sampling technique[2]. Sixty-three system analysts/designers and sixteen programmers that were felt to better represent the population were hand-picked. The intention was to draw upon the experience and knowledge of the most experienced professionals in the SIMAER. All of the seventeen organizations within the SIMAER had at least one representative. The professionals, experience on data

processing varies from one to twenty-one years . Of the
seventy-nine questionnaires delivered forty-two replies were
received (fifty-three percent return).

IV.    Procedure

The data and comments were gathered through a mail
survey of the ADP specialists. The subject areas covered are
presented below:

### Questionnaire

The five-page questionnaire was composed of nine subject
areas with a total of twenty-one questions. Blank spaces were
provided at the top of the first page to be filled with the
name, organization, function, and answerer experience on data
processing. Following that there was a short set of
instructions on answering the questionnaire, after which came
the questions.

### Questions

The questions were classified by subject as follows:

1.1 to 1.3 Methodology-Existed, Which, Standardization

2.1 to 2.4 Documentation-Types used, Phase where used

3.1 to 3.3 Graphical Representation-Types used, Phases

4.1 to 4.3 Management-Personnel Allocation, Control, Review
Methods

5.1 to 5.3 Programming-Languages used, Standardization,
Thechniques Used

6.1 to 6.3 Testing-Plan, when and How done

7.1 to 7.2 Maintenance-Types, Formalization

157

8.        Cost

9.        General Suggestions

V.  Results

As we are going to see below not every professional
answered or gave suggestions to every question. Concerning
the suggestions, most of them were more comments rather
suggestions. It was felt that by in large, people seemed not
to want to commit themselves, rather taking the approach of
"let the boss decide". However, some of the comments were
very useful.

The responses received to each questions are presented
in the sequence below:

1.   Method

1.1  Does your organization use any standard method for
ADS development?

No organization has established any standard method. The
most commonly used are shown in Table I below.

TABLE I

Methods Used by SIMAER'S Professionals

| METHOD | REPLY |
|--------|-------|
| No reply | 5 |
| Gane | 9 |
| HIPO | 2 |
| Jackson | 2 |
| James Martin (BD) | 4 |
| No method | 20 |
| TOTAL | 42 |

1.2 In any answer describe which method is used
mentioning: phases, what is done in each of them, documents
generated, advantages and disadvantages of the used
method, and source.

This question should be divided into two other
questions: One referring to development phases, i.e. life
cycle, and the other on the method itself. Concerning the
software life cycle, it was possible to conclude that there
is not any standard, and also that the ones used do not
consider the execution of reviews at the end of each phase.

The question concerning the method itself, standard or
not, was already answered in 1.1.

1.3 What method do you think will best fit the MAer

159

requirements? Why?

This question was intended to give a chance for participation by the specialists, and to gather suggestions for a method selection. The answers are shown in Table II below:

TABLE II

The Most Suggested Methods for the SIMAER

| METHOD | REPLY |
|---|---|
| No reply | 23 |
| Gane | 14 |
| Jackson | 2 |
| James Martin (BD) | 3 |
| TOTAL | 42 |

There were also some comments rather than real suggestions. They are:

- The one that best fits the developing organizations needs, the difficulty will be to make them follow it.

- Some structured method such as Gane, Yourdon, or Jackson. The reason is because low experienced personnel can do a good job with them.

- I did not study them comparatively. I hope your research brings us the answer.

- Top-down since it allows to have an overview of the system.

- I do not believe that the MAer can have just one method applicable to all sorts of system development. It has to be limited to a management level.

- I do not have an overview of all the problems which have influence on the several organizations in order to suggest one specific strategy, though I think the adoption of one standard method would be very helpful.

- The system structured method of Chris Gane, since it is the most readable in order to have a general understanding of the system.

- It would depend on the application area, for each a specific method should be used considering the peculiarities of the Brazilian Air Force.

- We should use the traditional waterfall model and over it apply the techniques described in the Chris Gane and Trish Sarson's book: Structured Systems Analysis.

- We are using the James Martins's method for database design and I think this method would be good for the MAer.

- Structured Systems Analysis - Chris Gane, because a lot of analysts of the MAer know it.

- Structured Systems Analysis - Chris Gane, because this is the most known among the academia at Rio de Janeiro.

- Considering the system development decentralization within the MAer and the programmers and analysts' heterogenous training I do not think that would be advisable to adopt one standard method for the MAer.

161

- Clearly it is necessary to discipline this matter. I see with some concern to think that just one standard method can be ajusted to a so vast field of applications. Indeed I think that would be advisable to divide the subject in two levels, one broad originated at the information system focal point (CINFE), establishing policies and rules for each step, and another sectorial, allowing some flexibility in accordance to each specific involved area.

- I think we should have one method produced by a consensus of a working group composed with user's and the CINFE representatives.

## Summary of the Answers on Methodology

It was seen that most of the suggestions were to adopt some of the tools and techniques suggested by Chris Gane[18]. This can be explained by the fact that the people who made such a suggestion have graduated from the Pontificia Universidade Católica - PUC in Rio de Janeiro where this is a text book for Computer Science Courses.

While most people think that a standard method would be helpful, others showed some concern about having a standard method due to the heterogeneous training and the diversity of applications. However, it may be noted that this standardization, in Rio de Janeiro area, essentially already exists informally, using the tools and techniques suggested by Chris Gane [18]. The heterogenous training can be considered one more reason to have a standard method with

skill in using it given by a common training.

2. Documentation

2.1 Which are the documents used in system
development?(manuals, reports, program documents etc).

- Each system is documented in a different way.

- We are using some of the CINFE's forms for program
documentation. In some cases the user's manual is produced.
This is the maximum we could get.

- As  I said before since we do not have a standard, it
varies from person to person. Someone document the programs,
other develop the manuals, a third does the report, so
nothing is predetermined.

- SIMAER's forms, System Manual, Operations Manual,
User's Manual, Gantt's Charts and Logic Data Flow Diagram.

- For programming and operation the standards SIMAER's
forms, and manuals.

- None as an obligation, it is up to each designer.

- The Structured Design System's Manual and the User's
Manual.

- The system's Manual, Program's Manual and User's
Manual.

- In all phases are used non-standard documents.

2.2 In which phase each document is elaborated ?

- Generally after the system is implemented.

- Generally the documents are elaborated after the

programmimg phase when the system is already implemented.

- Usually at the end.

- Usually the documents are done after the system or the programs are ready.

- Usually at the end.

2.3 Who is responsible for elaborating, updating and filing the documents during and after the system implementation?

- During the development the project manager is responsible for the documentation. After the implementation, it is the user responsibility or it is responsibility of the system development sector of the organization.

- The Development Section.

- The Analyst is responsible for elaborating; filing is responsability of the project manager.

- During the implementation it is the system analyst/designer . After the implementation it is the system manager.

- The system manual is elaborated and updated by the system analyst responsible for the system.

- The system manual is elaborated and updated by the system analyst responsible for the system. The program's manual is started by the analyst and passed to the programmers which should complete and update it. The user's manual is elaborated by the analyst together with a user

representative.

- The system analyst is responsible for maintaining the system documentation.

- The system analyst.

- The programmer in the programming phase and the Development Section for the user's manual.

- The programmer himself, however if sometimes the system or program lacks documentation another programmer will document.

- The database administrator.

- Each analyst is responsible for his system.

- The analyst before implementation. After that the user if his organization has an analyst.

- All the information will be stored in a data dictionary and the DBA will be responsible.

- The system analyst.

- The system analyst with collaboration of the programmers.

- The project manager.

2.4 In your opinion which would be the ideal documentation for the MAer?

- One that could be used both for conventional and for database system.

- Documentation automatically generated through utility software.

- Such answer could only be given by a centralized

165

CINFE's organization and method group which would analyze the problem during a significative amount of time (around one year) and for a specific type of system (management for example).

- A system structured manual, describing all the system's development phase in such a way to allow easy of maintaining and a user's manual explaining how to operate and utilize the system.

- I have not enough knowledge to make such suggestion. I have chosen this for being practical and simple.

- The one that should be chosen by the CINFE.

- A system manual and a operation manual.

- Data Dictionary.

- A documentation that represents the general consesus of a users group of several organizatiions.

- The one that each programmer feels more comfortable to elaborate.

- We should take the same reasoning as for question 1.3 (method), i.e., at a higher level we should standardize and at a lower level we should leave up to each sector without losing the centralized orientation.


## Summary of the Answer on Documentation

The CINFE has printed and supplies some standard forms for the coding and operation phases which are being used.

However, besides these forms no standard has been established for the entire system development. Most of the professionals know and produce the traditional system, users, and program manuals. Some people suggested the using of a data dictionary and only one organization is currently designing it.

Many people think that a standard documentation should be the product of a consensus, and defined only at the top level, leaving the details up to each organization.

3. Graphical Representation

Does your organization employ any type of graphical representation for system development (HIPO, Jackson, SADT, SREM, SAMM etc) ?

Answers:

Block Diagram - CCA BR, CISA, CPO, CENIPA,COMGAR, SEFA

Data Flow Diagram - CCA RJ, DIRAP, DIRMA, DIRINT, DEPV

System Modularization Diagram - CCA BR, ITA, DIRENG

Structured Charts (Constantine) - DEPV, DIRINT, DIRAP

Jackson - DIRINT, DIRMA, CCA RJ

HIPO - DEPV, DIRMA, CCA RJ

Skiner - DIRAP

Chen Entity-Relationship Diagram - CCA RJ

Note: The answers, although computed by organization, were based on individual responses. It was found that in a same organization different specialists use the same and/or different graphical representation.

167

The results, by individual response is shown in the Table III below.

TABLE III

Graphical Representation Techniques Used by SIMAER

| TECHNIQUE | REPLY |
|---|---|
| LDFD-Gane | 11 |
| Jackson | 5 |
| HIPO | 3 |
| Chen Entity-Relationship | 4 |
| Yourdon | 2 |
| Skiner | 1 |
| Block Diagram | 3 |
| Structure Chart | 4 |
| System Modularization Diagram | 3 |
| No Reply | 11 |
| TOTAL | 47 |

Note: More than one technique is usually used in one project, thus more than 42 responses are shown.

Suggestions for Question 3.3

- Considering the system development decentralization in the MAer I do not see any advantage in standardizing a graphical representation.

- Chris Gane and Jackson because is easy to learn and use.

- I believe Jackson, Yourdon and Gane being some of the most easy to build and interpret.

- Any since we give training.

- I suggest HIPO because it allows the documentation

168

parallel with the development.

- I think that Jackson would be a good suggestion since the five phases that compound it are enough for a quick and easy understanding of the system. I am happy with it.

- Chris Gane due to the easy of understanding.

- Jackson allows easy program maintenance.

- Jackson for program definition because it limits the programmer creativity, making the program easy to maintain. DFD for a logic system overview, makes it easy to the user understand the system allowing him to participate even without having ADP knowledge. HIPO for the system operation flow, because it is easy to document and maintain.

## Summary of the Answers on Graphical Representation

Again there is no established standard. It is up to each analyst/designer or programmer to choose. The most commonly used and suggested to be used by the SIMAER were: Gane's Data Flow Diagram for system analysis, Jackson for programming, HIPO for the operation phases, and Chen Entity-Relationship for database.

The main reasons for using and suggesting those were: Gane's DFD - easy to use and interpret both by the analyst and the user.

Jackson - allows easy program maintenance.

HIPO - because it is easy to document and maintain.

4. Development Management

4.1 Personnel Allocation

4.1.1 As you know there are several ways to organize the programming team. Which one is adopted in your organization?

All organizations use the classical teaming.

4.1.2 Considering the MAer peculiarities and its implications (military, hierarchy, duties, TDY etc) which would be the best teaming to be adopted ?

Answers are shown in the Table IV below.

TABLE IV

Suggested Teaming for the SIMAER

| TEAMING | REPLY |
|---|---|
| Classical | 26 |
| Specialist | 4 |
| Democratic | 3 |
| Chief-Programmer | 2 |
| No Reply | 7 |
| TOTAL | 42 |

Comments on questions 4.1.1 and 4.1.2

- In our organization we use the classical teaming mainly due reasons related to hierarchy, however this kind of allocation brings a lot of problems, mostly when the natural evolution of a programmer occurs, sometimes when reaching the same level of an analyst, he stays with his capacity limited due to his seniority. Here comes the questions; subutilize him, or utilize in functions incompatible with his grade ?

170

- For the military organization I think the classical is the most suitable.

- The classical teaming is giving good results in the database development.

## Summary of the Answers on Teaming

All of the organizations use the classical teaming which is obviously linked to the hierarchical reasons. At the same time some people pointed out that this teaming approach is also a source of trouble whenever a lower rank specialist reaches a skill level of an analyst. Even so, by far, the classical organization was the most suggested teaming type, even recommended by some sergeant programmers.

4.2 Development Control

4.2.1 Does your organization use any type of control development (Status Report, PERT/CPM, Gantt Charts etc) ? Which ?

Answers:

Status Report - CCA BR, CISA, CPO, CENIPA, COMGAR, SEFA, DEPV, DIRENG, DIRINT, CCA RJ, DIRAP

PERT/CPM - ITA

GANTT CHARTS - ITA, DIRMA , DIRAP

4.2.2 What is your suggestion for the MAer ? Why ?
Answers in the Table V below.

TABLE V

Suggested Development Control Tools to be Used within SIMAER

| TOOL | REPLY |
|------|-------|
| Status Report | 3 |
| Gantt Chart | 6 |
| PERT/CPM | 8 |
| No Reply | 25 |
| TOTAL | 42 |

Suggestions on Question 4.2.2

- Status report for being more objective, easy to do and understand and because it gives best results.

- PERT/CPM since it allows for corrections during the project development.

- A standard should be adopted by the MAer and its accomplishment enforced.

- We use the status report. We tried to use the PERT/CPM but it was hard to come up with a reliable time estimate for the tasks.

- For large systems, mainly if done by contractors, I suggest the PERT/CPM; for medium or small systems Gantt Chart.

- PERT/CPM because it is well known by most of analyst and programmers of the MAer.

- Each type of development requires a specific control.

172

In some cases, several could be utilized at the same time, one complementing the other.

- Status Report because it does not require any specific knowledge for understanding or elaboration.

- PERT/CPM to make it possible to follow up the development as a function of expected results.

## Summary of the Answers on Development Control

Most of the organizations just use the status report because they feel it is more objective, and easy to do and understand. However the most suggested for adoption by the SIMAER was the PERT/CPM, because it was considered to be well known, and also it makes it possible to follow up the development as a function of expected results.

4.3 Review Sessions

4.3.1 As you know there are several types of review sessions such as: Inspection, Walkthrough, Circulating Review etc. Does your organization use any type?

Answers:

INSPECTION - ITA, DIRMA, CCA RJ

WALKTHROUGH - DEPV

CIRCULATING REVIEW - None

4.3.2 Who participates of the reviews? How often are they done?

- The development team

173

- The manager, analyst and programmers

- Users, analysts and programmers

- Weekly

- As necessary

Summary of the Answers on Review Sessions

Most of the organizations do not use any formal type of review session nor have them scheduled in a regular fashion, they just do it as the need arises.

When asked to give suggestion there was a weak reponse. Only one suggested the walkthrough.

5 . Programming

5.1 Which programming languages are used in your organization?

Answers:

COBOL - All of organizations

Fortran - CCA BR, CINDACTA1, COPAC, AFA, ITA, DEPV, DIRENG, CCA RJ

PL1 - ITA, DEPV, CCA RJ

PASCAL - CCA RJ, ITA

ALGOL - ITA

BASIC - ITA

CORAL - ITA

LTD - DIRINT

5.2 Do you think advisable to standardize some HOL for the MAer? If so which?

Answers in the table VI below.

174

TABLE VI

Hols Suggested to be Used by SIMAER'S Organizations

| LANGUAGE | REPLY |
|----------|-------|
| C | 1 |
| BASIC | 1 |
| FORTRAN | 2 |
| COBOL | 2 |
| PASCAL | 3 |
| No standard | 4 |
| No reply | 29 |
| TOTAL | 42 |

Comments on Question 5.2

- It will depend on the hardware and basic software. Standardize is always good. The hard part is to make people follow the standard.

- It will depend on the CINFE.

- I do think advisable to standardize, but without too much rigidity.

- No, the languages should be adequate to the equipments and the system involved.

- Yes, because standardize means to reduce cost.

- Standardization is a form of obstruction to knowledge increasing, we ought just to advise on some languages for certain applications.

- The language standardization always bring benefits to

175

the users.

## Summary of the Answers on Programming Languages

The most common language used by all of organizations was COBOL, followed by FORTRAN and PL1.

When asked if it would be advisable to standardize some HOL for the MAer the answers vary. The main reasons favoring standardization were cost reduction and benefits for the users. The cons were limitations imposed on knowledge and applications characteristics.

The most suggested languages in case of standardization were: COBOL, FORTRAN, PASCAL and BASIC.

5.3 Which types of modern programming techiniques does your organization utilize?

Answers:

MODULAR DECOMPOSITION - CCA BR, CCA RJ, CISA, DIRMA

PSEUDOCODE - CCA RJ, DEPV, DIRENG, DIRINT

TOP-DOWN - DEPV, DIRINT, DIRMAS

STRUCTURED PROGRAMMING - CCA RJ

## Summary of Answers on Modern Programming Techniques

This question did not have a strong response, maybe because of the way it was stated, however it could be infered from the answers that the using of modern techniques is viewed as being more a programmer option rather than an organization guideline. It could be seen that the CCA RJ is a focal point for training in the Jackson methodology for

structured programming.

6. Testing

6.1 Is a formal testing plan prepared for this phase?

It was found that in only three organizations a formal test plan is prepared for testing. In the others tests are done without a pre-established plan.

7. Maintenance

7.1 What type is more common in your organization? corrective or for improvement?

The most common, as expected was the corrective maintenance; only two organizations stated they were also working on improvement maintenance.

8. Cost

Does your organization use any method for cost calculation?

Only one organization stated that they are testing a homemade method base on an IBM standard. No further details about the method were provided.

9. General Comments

- The harder part will be to make people follow the methodology. Usually analysts want to go straight to the design and coding phase.

- Your work will be of great value for the data processing future in the MAer.

- ... I think the MAer should implement developing methods.

- Should exist one standard methodology for the MAer.

- I think that the adoption by the SIMAER of a standard methodology covering documentation, graphical representation, personnel allocation, review methods, languages, verification and validation, maintenance, cost estimation and other issues, should be studied. However, considering the subject relevance and complexity, I think that any definition in such area should be the object of a working group study composed of several representatives of the SIMAER's members.

## VI. Conclusion

The SIMAER has not adopted a standard software life cycle model to be followed by its ADP professionals. The models used and followed individually do not consider the necessary and current accepted practice of doing reviews[6] at the end of each phase.

None of the SIMAER's organizations has developed or formally adopted a standard methodology, in which modern methods, tools, and techniques for system's analysis or software designed are employed, to be followed by their ADP professionals during system's development. However, as shown in the Tables I and III, some professionals located in Rio are familiar with and do use, informally, some of the modern methods, tools and techniques supported by Chris Gane[18], Jackson[22], and James Martin[24] for database.

When asked to suggest some method to the MAer there was a heavy concentration on Gane, followed by James Martin and Jackson, as shown in the Table II.

While most of the SIMAER's professionals think that a standard methodology (method + life cycle) would be helpful and cost-saving, a few showed some concern about having a standard arguing that the heterogenous training and the diversity of the applications would not make it practical. The survey showed that in a small but varied number of organizations in Rio a standard almost existed. The heterogenous training can be considered one more reason for having a standard. A common training and practice would help level off the degree of experience of people involved in system development.

There is not at any level, a MAer documentation standard establishing the minimal documentation that should be produced during a system development.

Concerning system development management, the most common teaming approach is the classical method, and the most common control tool is the status report. This results in very little planning, and mainly just control. Amazingly the least used tool for planning and control was the PERT/CPM. However, at the same time, it was the technique most suggested to be adopted. Most of the professionals know but do not use any formal type of review technique. The

179

statistics related to those findings are shown in Tables IV
and V.

For programming the most commonly used languages are:
COBOL, Fortran, and PL1. The extensive use of languages that
facilitate the use of modern programming practices is not
enforced. When questioned, a few specialists showed some
concern about the establishment of standard HOLs for the
SIMAER, arguing that they could not apply to every
application, and also would limit the professionals
knowledge. Any standard should establish a number of
languages sufficient enough to cover several types of
applications. As far as the limitation of knowledge no
standard is supposed to be static, not allowing for
modification to implement improvements that certainly will
arise in this field. The statistics for this topic is shown
in Table VI.

In conclusion it could be seen that the problem is the
incomplete adherence to the modern principles of software
engineering. All the survey findings reveal the need for
training, policies, and standards that enforce the observance
of the referred principles in the SIMAER's systems software
development.

Dayton, march, 18, 1985.

Dear Friend

As you probably know I am attending a course in order to obtain a master degree in Computer Science.

Such achievement depends on several requisites, among others to do a research on a topic of the Air Force interest and subsequently to present it as a thesis.

Looking for to accomplish that requirement, as also to propose a solution to an Aeronautical Ministry problem, I decided to design and suggest a standard Data Processing System Development Methodology to our ministry.

Considering the special condition in which I find myself (far and with a short time) I decided to elaborate the annex questionnaire looking for to obtain:

(1) **Information on the current stage on data processing system development in the MAer.**

(2) **Suggestions in order to design a standard methodology.**

Finally, due to all the considerations above I ask the colleague corroboration in answering, until the second of may, the annex questions and return them to Col Victorio Baptista da Silva-CCA RJ, which will redirect them to me.

Looking forward for your corroboration I send my sincerely thank you.

_____
Aparecido Francisco de Oliveira - Lt Col


181

**SURVEY ON DATA PROCESSING SYSTEM DEVELOPMENT IN THE MAER**

I. GENERAL DATA

   1. Name/Rank:_____

   2. Organization:_____3. Function:_____

   3. Time in ADP activity:_____

II. INSTRUCTIONS:

   1. Answer in this sheet and complement with annex.

   2. Whenever possible annex examples.

   3. All questions refer to your organization. If in it no system is being developed use your previous experience **FOR SUGGESTIONS.**

   4. Do not refrain of presenting suggestions because they will be of great value for my work.

III. QUESTIONNAIRE:

   **1.Methodology**

   1.1 Does your organization use any standard methodology for data processing system development?

YES |_____|          NO |_____|

1.2 Describe which methodology(ies) is used including: phases, what is done in which of them, documents generated, advantages and disadvantages of the used methodology and source.

1.3 What kind of methodology do you think would be good for the MAer? Why?

## 2.Documentation

2.1 Which are the types of documents employed in the system development? (manuals, reports, program documents etc). Annex samples if possible.

2.2  In which phase each document is elaborated?

2.3 Who is reponsible for elaborating, up-dating and filing of the documentation during and after the system implementation?

2.4 In your opinion which is the ideal documentation for the MAer?

## 3. Graphical Representation

3.1 Does your organization use any type of graphical representation for system development? (HIPO, Leighton, Jackson, SADT, SREM, SAMM etc)

YES |_____|         NO |_____|

END

FILMED

DTIC
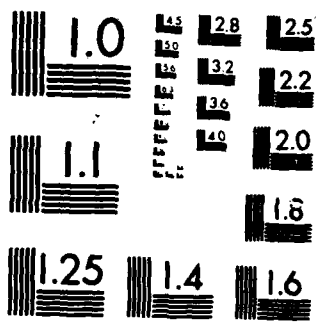
MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

3.2 If so, which, in which phase and why was such technique chosen?

3.3 Which is your suggestion for the MAer? Justify.

## 4. Development Management

4.1 Personnel Allocation

4.1.1 As you know there are several ways of personnel allocation for system development. Which of the types below is used in your organization?

4.1.1.1 - Classical (Manager, Analyst and Programmer)

4.1.1.2 - Chief-Programmer (Chief-Programmer, Administrator, Documentation Editor, Program Librarian, Toolsmith, Test Specialist etc ).

4.1.2 Considering the MAer pecularities and its implications (militarity, hierarchy, duty, TDY etc) which would be the best teaming to adopt?

4.2 Development Ccntrol Instruments

4,2,1 Does your organization utilize any type of development control ( status Report, PERT/CPM, Gantt Chart etc)?

YES [          ]          NO [          ]

Which:_____

### 4.3 Review Sessions

4.3.1 As you know there are several review methods for system design such as: Inspection, Walkthrough, Circulating Review etc. Does your organization employ any method?

YES |_____|     NO |_____|

4.3.2 Who participates of the sessions? How often are they done?

### 5. Programming

5.1 Which programming languages are utilized in your organization?

5.2 Do you think that it would be advisable to standardize some HOL for the MAer? If so which for management, scientific and embeded systems applications?

5.3 Which modern programming techniques (chief-programmer, Librarian, top-down development, modular decomposition, structured programming, pseudocode, structured walkthrough) does your organization utilize? Which are your suggestions for the MAer?

## 6.Testing

### 6.1 Are plans made?

YES [      ]     NO [      ]

### 6.2 When are they done?


## 7. Maintenance

### 7.1 Which type of maintenance is more frequent?

Corrective_____          For Improvement_____

### 7.2 Is there a formal document for request?

YES [      ]     NO [      ]

## 8. Cost

Does your organization use any method for software development cost estimation?

YES [      ]     NO [      ]

Which?_____

What is your suggestion for the MAer?

9. The word is free. Sorry for making you so tired and thank you for the collaboration.

## Appendix B

### TABLE VII

Personnel Titles and Descriptions within SADT

---------------------------------------------------------------

| Title | Description |
|-------|-------------|
| Author | One who performs data gathering and analysis tasks and organizes this material using SADT models |
| Commenter | One who reviews models by authors and who comments in writing; usually is another author |
| Reader | One who reads SADT diagrams constructed by others but is not required to document (write) these comments; in a sense, one who receives the models for his own information or verbal comment only |
| Expert | One who provides technical guidance to authors concerning the resolution of unique or troublesome problems |
| Technical Committee | A group of expert personnel who review the results of the analysis effort on a level-by-level basis; they can identify or resolve technical problems and coordinate with project management |
| Project Librarian | One who archives and controls versions, releases, updates, and feedback from reviewers |
| Project Manager | One who has overall responsability for the project |
| Monitor (or Chief Analyst) | One who provides technical assistance and guidance in SADT use |
| Instructor | One who trains author and commenters to use SADT |

---------------------------------------------------------------

TABLE VIII

Phase of the Life Cycle/Methods, Tools and Techniques Use

| METHOD TOOL TECH-NIQUE | SOFTWARE LIFE CYCLE | | | | | | |
|---|---|---|---|---|---|---|---|
| | SYSTEM ANALYSIS | REQMT DEFINIT | PRELIMIN DESIGN | DETAILED DESIGN | CODE | TEST | OPERATION MAINTENANCE |
| HIPO | | X | X | X | | | X |
| SADT | X | X | X | X | X | X | X |
| STRUC DESIGN | X | X | X | X | X | X | X |
| GANE | X | X | X | X | X | X | X |
| JACK-SON | | | | X | X | X | X |
| STRUCT CHART | | | | X | X | X | X |
| DFD | X | X | X | | | | |
| LDFD | X | X | X | | | | |
| CHEN ENTITY | X | X | X | | | | |
| DATA DICTI-ONARY | X | X | X | X | X | X | X |
| DECI-SION TABLE | X | X | X | X | X | X | X |
| PSEUDO CODE | | | X | X | X | | |

188

## TABLE IX

### Comparison of Methods

| CHARAC- TERISTICS | METHOD | | | |
|---|---|---|---|---|
| | STRUCTURE DESIGN | SADT | GANE | JACKSON |
| CURRENT SYSTEM MODELING | YES | YES | YES | NO |
| SYSTEM SPECIFICATION | YES | YES | YES | NO |
| SYSTEM ARCHITECTURE | YES | SOMEWHAT | YES | YES |
| LOGICAL DESIGN | YES | YES | YES | SOMEWHAT |
| PHYSICAL DESIGN | YES | POTENTIALLY | YES | YES |
| AVAILABILITY OF TRAINING COURSES IN BRAZIL | YES | NO | YES | YES |
| EASE OF USE (HIGH = EASE) | HIGH | LOW | HIGH | MODERATE |
| PROLIFERATION LEVEL IN BRAZIL | HIGH | LOW | HIGH | HIGH |
| LEARNING EFFECTIVENNES | HIGH | LOW | HIGH | MODERATE |
| COMMUNICATION WITH CUSTOMERS | HIGH | LOW | HIGH | LOW |
| HIERARCHICAL IN NATURE | YES | YES | YES | YES |

TABLE IX - Comparison of Methods (Cont'd)

| | | | | |
|---|---|---|---|---|
| PROVISION OF OBJECTIVE EVALUATION CRITERIA | YES | NO | YES | SOMEWHAT |
| BASIS OF METHOD | CONCEPT | CONCEPT/ PROCEDURAL | CONCEPT/ PROCEDURAL | CONCEPTUAL/ PROCEDURAL |
| DEGREE OF TECHNICAL ISSUE COVERAGE | 4 OUT OF 4 | 3 OUT OF 4 | 4 OUT OF 4 | 3 OUT OF 4 |
| SUPPORT BY AN AUTOMATED TOOL | NO | YES | NO | NO |
| SUPPORT BY QUALIFIED CONSULTANTS | YES | YES | YES | YES |
| MOST PORTABLE FEATURE (IF ANY) | COUPLING/ COHESION | DATA/ CONTROL MODELING | COUPLING/ COHESION | DATA STRUCTURE MODELING |

### Evaluation Criteria of the Methods

1. Current System Modeling - the ability of the method to provide users with a way to model an existing system. The system may include manual tasks, physical objects, and geographic locations as well as the more classical functional needs and processes. This is a desirable software development method feature.

2. System Specification - the extent to which the method provides the necessary semantic and conceptual framework to permit the statement of requirements for an entire system, not just the software. This is an important and desirable method feature.

190

3. System Architecture - the ability of the method to allow flexibility in laying out the overall interface between the major system elements. This is a desirable software development method feature.

4. Logical Design - whether the method includes a clear, explicit recognition that an abstract, conceptual solution must be formulated and refined prior to the introduction of implementation issues. This is a desirable method feature.

5. Physical Design - whether the method explicitly addresses implementation issues apart from conceptualization of the logical design solution. This is a desirable method feature since it allows the designer to separate the "what" from the "how"

6. Availability of Training Courses in Brazil - the degree to which public courses are available in Brazil. This is a SIMAER requirement.

7. Ease of use - the ease with which a designer can effectivelly use the method; reduced by unique requirements such as templates and pre-printed forms. A desirable method feature and a SIMAER requirement.

8. Learning Effectiveness - the absence of subtleties in the method that might confuse a novice; intended to alert designers to the amount of care they must exert in order to avoid unforeseen difficulties. A desirable method feature and a SIMAER requirement.

9. Communication with Customers - the degree to which the method provides open communication between customer and designer (for example, through understable diagramming techniques). One of the most desirable method feature since communication is a key issue in software development.

10. Hierarchical in Nature - the extent to which the method provides a convenient scheme for controlling complexity via the organization of the design (and system) into ordered chunks that can be examined separetely from the rest of the system. A desirable method feature that will allow for modularization.

11. Proliferation Level in Brazil - the degree to each a method has spread as an indicator of its relative effectiveness. A SIMAER requirement.

12. Provision of Objective Evaluation Criteria - whether the method has a measure of design that would yeild approximately the same result if used by two different (unbiased) designers. A desirable method feature.

13. Basis of Method - whether the method is based on some rationale; prescribed set of rules, or a combination of both.

14. Degree of Technical Issue Coverage - the relative importance of any one or several of the four technical issue classes present in any software design effort: data structure, data flow, control structure, and control flow.

15. Supported by an Automated Tool - whether the method is supported by a computer-aided scheme to make changes,

identify inconsistencies, and do clerical tasks, thereby

enhancing the designer's effectiveness. A desirable method

feature for the development of large systems.

16. Supported by Qualified Consultants - the availability

of experienced advisers to reduce the instances of misuse and

of unsatisfactory results. A desirable method feature and a

SIMAER requirement.

17. Most Portable Feature - that part of the method, if

any, that could be used totally apart from the original

method (for example, using the coupling and cohesion

characteristics for evaluation with some method other than

structured design). A desirable method feature.

# Bibliography

1. Alavi M. "An Assessment of the Prototyping Approach to Information Systems Development," <u>Communications</u> <u>of</u> <u>the</u> <u>ACM</u>, <u>27</u> : 556-563 (June 1984).

2. Bailey, R. W. <u>Human</u> <u>Performance</u> <u>Engineering:</u> <u>A</u> <u>Guide</u> <u>for</u> <u>System</u> <u>Designers.</u> Englewood Cliffs, NJ: Prentice-Hall Inc., 1982.

3. Baker, F. T. "Chief Programmer Team Management of Production Programming, "<u>IBM</u> <u>Systems</u> <u>Journal</u>, <u>11</u> : 415-421 (1972).

4. Bell, Thomas E., David C. Bixler, and Margaret E. Dyer, " An Extendable Approach to Computer - Aided Software Requirements Engineering". <u>IEEE</u> <u>Transactions</u> <u>on</u> <u>Software</u> <u>Engineering</u>, SE-3: 6-15 (January 1977).

5. Boehm, Barry W. <u>Software</u> <u>Engineering</u> <u>Economics</u>. Englewood Cliffs, NJ: Prentice-Hall Inc., 1981.

6. Bohm C. and G. Jacopini. "Flow Diagrams, Turing Machines and Languages with Only Two Formatior Rules," <u>Communications</u> <u>of</u> <u>the</u> <u>ACM</u>, <u>9</u>: 366-371, (May 1966).

7. Booch, G. <u>Software</u> <u>Engineering</u> <u>with</u> <u>Ada</u>. Menlo Park: The Benjamin/Cummings Publishing Co., 1983.

8. Brandon, D. <u>Management</u> <u>Standards</u> <u>for</u> <u>Data</u> <u>Processing</u>. Princenton, NJ: Van Nostrand, 1963.

9. Chapin,N. " Flowcharting with the ANSI Standard : A Tutorial, " <u>ACM</u> <u>Computing</u> <u>Surveys</u>, <u>2</u>: 119 - 146 ( June 1970 ).

10.Chen, P. "The Entity-Relationship Approach to Logical Data Base Design" The <u>Q.E.D.</u> <u>Monograph</u> <u>Series</u> <u>on</u> <u>Data</u> <u>Base</u> <u>Management</u>, <u>6</u>: 15-21 (March 1977).

11.Colter, A. M. "A Comparative Examination of Systems Analysis Techniques," <u>MIS</u> <u>Quarterly</u>, 51-64 (March 1984).

12.Davis, B.G. and Margrethe H. Olscn. <u>Management</u> <u>Information</u> <u>Systems:</u> <u>Conceptual</u> <u>Foundations,</u> <u>Structure,</u> <u>and</u> <u>Development</u> New York: MacGraw-Hill, 1985.

13.DeMarco, T. <u>Structured</u> <u>Analysis</u> <u>and</u> <u>System</u> <u>Specification</u> New York: Yourdon Press, 1978.

14.<u>Dictionary</u> <u>of</u> <u>Computing</u>. Oxford Science Publications.

14. Dictionary of Computing. Oxford Science Publications. New York, 1983.

15. Encyclopedia of Computer Science and Engineering (Second Edition). New York: Van Nostrand Reinhold Company, 1983.

16. European Space Agency - "Software Engineering Standards". 1984.

17. Fairley, R. Software Engineering Concepts. New York: Mac-Graw-Hill, 1984.

18. Gane, C. and Trish Sarson, Structured Systems Analysis: Tools and Techniques. Englewood Cliffs,NJ:Prentice-Hall Inc., 1979.

19. Hadfield, S.M. and Gary B. Lamont. "The Software Development Environment" Proceedings of the Digital Equipment Computer User Society , 171-177 (October 1983).

20. HIPO-A Design Aid and Documentation Technique. IBM Corp. Manual # GC20-1851. White Plains, NY: IBM Data Processing Division, 1974.

21. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1983.

22. Jackson, M.A. Principles of Program Design. London: Academic Press, 1975.

23. Law, E. "GSA Reports Huge Increase in Micro Buys", Government Computer News, 4 (June 1985).

24. Martin, James. Managing the Data Base Environment. New York: MacGraw-Hill, 1984.

25. Nolan, Richard L. "Managing the Crises in Data Processing" Harvard Business Review. 115-126 (March-April 1979).

26. Oliveira, A. F. Current Software Development Situation in the Brazilian Air Ministry: A Survey of Methodology, Documentation, Graphical Representation, Management, Programing, Testing, Maintenance, and Suggestions. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, May 1985.

27. Portuguese Air Force Regulation RFA 45-6. Lisbon, 1984.

28. Peters, Lawrence J. Software Design: Methods and Techniques New York: Yourdon Press, 1981.

29.Rodewal Hans, Capt FRG. Personal correspondence. German Military Representative USA and Canada. Washington DC, 15 July 85.

30.Rubey, Raymond J. Technical Director. Telephone Interview.Softech, Inc. Ohio,17 October 1985.

31.Silva, Victorio B. da "The Informatic and the Development of Its Applications in the Brazilian Air Force," Brazilian Air Force Staff Command School Project, 203: 7-8 (December 1982).

32.SOFTECH Structured Analysis and Design Technique, SADT Manual 9022-78.

33.Stevens, W.P., Meyers G.J., and L.L. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design(Second Edition). New York: Yourdon Press, 1978.

34.The Brazilian Federative Republic Constitution. Rio de Janeiro: National Press, 1949.

35.Teichroew, D. and Hersey E.A. "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis," IEEE Transaction on Software Engineering, SE-3: 211-218 (January 1977).

36.Weinberg,V. Structured Analysis. New York: Yourdon Press 1978.

37.Woffinden, Duard S., Instrutor. Software Engineering Class Notes. EE 5.93. School of Engineering, Air Force Institute of Techonology (AU), Wright-Patterson AFB OH (Spring 85).

38. Zelkowitz, M.V. "Perspectives on Software Engineering," Computing Surveys, 10: 197-216 (June 1978).

VITA

Aparecido Francisco de Oliveira was born on 26 july in
Bauru, Sao Paulo, Brazil. He graduated in 1966 with a
Bachelor of Science (B.S.) degree from Academia da Força
Aérea Brasileira (Brazilian Air Force Academy) when he also
received his pilot's wings. In 1977 he graduated with a B.S.
degree in Business Administration from the Centro de Ensino
Unificado de Brasilia (United Educational Center of
Brasilia).

As an officer and pilot in the Brazilian Air Force
(BAF), he has worked at the operational level since 1968,
accumulating more than five thousand flying hours, in both
training and operational missions . As he progressed in rank,
his responsability slowly shifted more to management
activities.

Getting involved in the management activities of
aircraft maintenance and supply, he started using computers
for inventory controls. This was the incentive to study the
ADP area.

Having taken several courses in data processing, his
last assigment was head of the Brazilian Data Processing
Center.

Permanent Address:

SHIS QI 27 Conj 14 Casa 13

BRASILIA - DF 70016

BRASIL

197

# REPORT DOCUMENTATION PAGE

| Unclassified | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION | 1b. RESTRICTIVE MARKINGS |

| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT |
|---|---|
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | Approved for public release; distribution unlimited |

| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
|---|---|
| AFIT/GCS/ENG/85D-13 | |

| 6a. NAME OF PERFORMING ORGANIZATION | 6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION |
|---|---|---|
| School of Engineering | AFIT/ENG | |

| 6c. ADDRESS (City, State and ZIP Code) | 7b. ADDRESS (City, State and ZIP Code) |
|---|---|
| Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433 | |

| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION | 8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
|---|---|---|
| Brazilian Air Force | BAF | |

| 8c. ADDRESS (City, State and ZIP Code) | 10. SOURCE OF FUNDING NOS. | | | |
|---|---|---|---|---|
| | PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT NO. |
| Brazil, Brasilia, D. F. | | | | |

| 11. TITLE (Include Security Classification) |
|---|
| See Box 19 |

12. PERSONAL AUTHOR(S)
Oliveira, Aparecido Francisco, B.S., Lt Col, BAF

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Yr., Mo., Day) | 15. PAGE COUNT |
|---|---|---|---|
| MS Thesis | FROM _____ TO _____ | 1985 December 13 | 197 |

16. SUPPLEMENTARY NOTATION

| 17. | COSATI CODES | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB. GR. | Software Life Cycle    Methodology |
| | | | Life Cycle    Software design |
| | | | Method    Standard |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title:  The Design of a Standard Software Development Methodology for the Brazilian Aeronautical Ministry.

Thesis Chairman:  Duard S. Woffinden, Captain, US Army.

Approved for public release: IAW AFR 190-1/.

*[signature]* 16 JAN 86
WN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AFIT)
Wright-Patterson AFB OH 45433

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐ | Unclassified |

| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE NUMBER (Include Area Code) | 22c. OFFICE SYMBOL |
|---|---|---|
| Duard S. Woffinden, Captain, US Army | | |

**DD FORM 1473, 83 APR** EDITION OF 1 JAN 73 IS OBSOLETE.

Abstract

This thesis proposes a standard software design methodology for the Brazilian Aeronautical Ministry.

The project matched the requirements of the Brazilian Aeronautical Ministry with the software life cycle models, methods, and techniques, which are currently available and most widely utilized.

Based on the analysis, a waterfall model was selected and integrated with some methods, tools, and techniques, such as Gane's methods, SADT, Data Dictionary, etc.

All of these recommendations were included in a proposed regulation for a software development methodology.

# END

# FILMED

3-86

# DTIC