

National Institute of Electricity and Electronics
INELEC - BOUMERDES
DEPARTMENT OF RESEARCH

THESIS

Presented in partial fulfilment of the requirements of the

DEGREE OF MAGISTER

In Electronic Systems Engineering

by

Rabah IMACHE

COMPUTER AIDED MEDICAL
TRAINING SYSTEM

Defended on September 19, 1994 before the jury:

President: Mr. A. BOULARAS, Professeur, USTHB.

Members: Mr. B. MEZHOUD, M. A. Master, INELEC

Mr. S. ACHOUR, PhD. C. C., U T O.

Mr. H. AZOUNE, T.U. C. C., USTHB.

Registration Number: 01/1994.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my research advisor Mr. Belkacem MEZHOUD for his assistance and guidance throughout the present work, and for his moral support and encouragements.

I am also grateful to Professor Mustapha MAAOUI for his collaboration and patience during the experimental development of the medical CAL application.

I would like also to thank Dr. Kamel HARICHE for his encouragements and for the research facilities necessary to the success of this project, he offered; all the research department members for their moral support, and all the library members for their kindness.

Finally, I would like to thank all INELEC students and staff who contributed to the achievement of this work.

ABSTRACT

In this thesis, an approach to software projects and products is suggested and investigated, then a medical Computer Aided Learning, CAL, application is developed.

The approach under study is named Structured Evolutionary Development Approach, or simply SEDA. It is a cocktail approach which consists of two components, the evolutionary prototyping concept, which involves developing the system in an incremental fashion, and the structured paradigm, which follows a phased life cycle. The choice of this approach is based on the evolutionary process of software engineering through time. That's, software engineering, which is the remedy to the software crisis, is the creation of systematic approaches or paradigms and tools such as the Waterfall approach, the evolutionary approach, the prototyping approach, and so on. Despite all the created and applied systematic software development approaches, the users and developers are still not satisfied. Each developed approach has its advantages and disadvantages.

The aim of the present work is to suggest a new approach in order to contribute in the improvement of software products and projects development. The idea behind this concept is to maximize

the advantages while minimizing the disadvantages of the cocktail approach. And this can be achieved by the combination of the evolutionary prototyping paradigm with the structured methodology which show high complementarity.

In order to check the efficiency of the proposed approach, an experiment is conducted. This experiment consists of developing a medical CAL application named, Computer Aided Medical Training System, or simply CAMTS. It is interactive and microcomputer-based training system designed to be used in a hospital by surgery medical personnel, physicians of all the degrees and medical students, in order to learn, review, self-evaluate their knowledge, and prepare examinations. This system will help the medical staff to be aware of the state of the art of their profession, and allow them to attain and maintain an acceptable level of competence which will improve patient care.

TABLE OF CONTENTS

THE ORGANISATION OF THE PRESENT WORK	viii
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 SOFTWARE ENGINEERING	5
2.1 History of software engineering	5
2.2 What is Software Engineering ?	8
2.3 Software Metrics	8
2.4 Software Qualities	9
2.5 The Software Life Cycle Models	11
2.5.1 The Slum Dunk Model	12
2.5.2 The Baroque Model	13
2.5.3 The Waterfall Model	13
2.5.4 The V-diagram Model	16
2.5.5 The Exploratory Programming Model	17
2.5.6 The Prototyping Model	17
2.5.7 The Evolutionary Model	19
2.5.8 The Spiral Model	21
2.5.9 The Fourth Generation Techniques ... Model	21

2.5.10 The System Assembly from Reusable ..	
Components Model	23
2.5.11 Transformation Models	24
CHAPTER 3 STRUCTURED METHODS	26
3.1 Introduction	26
3.2 Definitions	29
3.3 The structured Methods and the Life Cycles	29
3.3.1 Analysis Phase	30
3.3.2 Design Phases	31
3.3.2.1 System Design Phase	33
3.3.2.2 Detailed Design Phase	33
3.3.3 Implementation Phase	34
3.3.4 Testing Phases	35
3.3.4.1 Unit Testing Phase	36
3.3.4.2 Integration Testing Phase ...	36
3.3.4.3 System Testing Phase	36
3.3.5 Maintenance Phase	37
CHAPTER 4 MODELLING TECHNIQUES	40
4.1 Introduction	40
4.2 Data Flow Diagrams	43
4.3 Data Dictionary	46
4.4 Pseudocode	48
4.5 Structure Chart	50
4.6 Flowcharts	52

CHAPTER 5	THE COCKTAIL APPROACH : SEDA	56
5.1	Introduction	56
5.2	The Structured Approach	57
5.3	The Evolutionary Approach	60
5.4	The Cocktail Approach	63
5.5	Conclusion	66
CHAPTER 6	THE APPLICATION : COMPUTER AIDED MEDICAL	
	TRAINING SYSTEM (CAMTS) DEVELOPMENT	68
6.1	Introduction	68
6.2	Computer Assisted Learning	70
6.2.1	Integrating Conventional Teaching and ..	
	CAL Schemes	70
6.2.2	Factors Influencing Learning and CAL	71
6.2.3	CAL and the Computer Aided Medical	
	Training System	72
6.3	CAMTS Development Process	75
6.3.1	Problem Definition and Feasibility	
	Study Phase	78
6.3.1.1	Problem Definition	78
6.3.1.2	Feasibility Study	79
6.3.2	Requirements Analysis Phase	81
6.3.2.1	Data Flow Diagrams	83
6.3.2.2	Data Dictionary	35
6.3.3	Design Phases	89
6.3.3.1	System Design Phase	90
6.3.3.2	Detailed Design Phase	90
6.3.3.3	System Pseudocode	92

6.3.3.4 Data Dictionary	96
6.3.4 Implementation Phase	99
6.3.5 Testing Phases	100
6.3.5.1 Unit Testing Phase	100
6.3.5.2 Integration Testing Phase ...	101
6.3.5.3 System Testing and Decision	
Making Phase	101
6.3.6 System Delivery and Maintenance	101
6.4 Cost-Benefit Analysis	102
6.5 Results	103
CHAPTER 7 CONCLUSION	105
BIBLIOGRAPHY	108
APPENDIX A GUIDELINES FOR THE MODELLING TECHNIQUES ...	122
A.1 Data Flow Diagrams	122
A.2 Structure Charts	123
A.3 Data Dictionary	124
APPENDIX B SYSTEM FLOWCHARTS	125

THE ORGANISATION OF THE PRESENT WORK

The order of presentation in the present work is organised as follows: first software engineering is introduced from the point of view history, evolution, and maturing; secondly, the existing paradigms and modelling techniques are dealt with; thirdly, the proposed approach, namely Structured Evolutionary Development Approach, or simply SEDA, is studied theoretically, then computer assisted learning is introduced; finally, the proposed approach is applied to a medical CAL application development to make a comparison between the theoretical previsions and the experimental results.

The first chapter deals with the birth of software engineering, identifies the problem we are dealing with, and introduces the solution and the application domain, namely Computer Assisted Learning, or CAL. After considering a number of objectives and problems of software production, the software crisis is discussed then, software engineering is presented as a remedy to that crisis. The remedy is the creation of a number of systematic approaches that are introduced in chapter two. Chapter three deals with the structured methods throughout the life cycles. The

modelling techniques which are used to develop the specification documents throughout the development phases are discussed in chapter four. Whereas chapter five deals with the theoretical study of the suggested cocktail approach whose components are the structured approach and the evolutionary prototyping approach. The first sections of chapter six deal with Computer Aided Learning , CAL, then the approach under study, namely structured evolutionary development approach or simply SEDA, is studied experimentally in the remaining sections of chapter six in which it is tested by applying it to the development of a medical CAL application, namely computer aided medical training system (CAMTS). It is interactive and microcomputer-based training system designed to teach at the level of a hospital, the surgery medical personnel, students and physicians. And the implementation is done using the dBASE IV programming language. The purpose of this experiment is to evaluate the proposed approach to software products and projects development, and results are drawn. That is our objective is to contribute in the improvement of software projects and products development. Finally, a general conclusion is given in chapter seven.

INTRODUCTION

Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. It deals with software systems for which it uses engineering principles in the development of these systems, and is made up of both technical and non-technical aspects [12], [13].

During the sixties, computer scientists, systems analysts, designers, and programmers, were faced with one major problem which was their failure to develop reliable software products, or systems [5]. One reason for the difficulty of producing reliable software is the nature of software itself; that's, software is abstract rather than physical in nature. Another reason is that the computer hardware on which software products run has become cheaper and more sophisticated, giving rise to greatly increased expectations about what computer systems can achieve [14]. That's, as the hardware costs came down due to technological advances and the economics of mass production, the costs of software became more and more significant in most system development projects [9].

As a result, the size, capabilities and complexity of software systems are also increasing, putting further demands on the software developers ability to produce reliable software. A further reason lies in the way in which software is developed. The emphasis is on coding, with insufficient time given to requirements analysis and design [14]. All these problems in systems and software development resulted in a critical situation, the so-called "Software Crisis" [1], [3], [5]. The question that arises concerns how the software developers are to overcome these obstacles to the production of reliable and effective software. Software engineering is the response or remedy; that's, software engineering is intended to assist the development of good-quality software within budgets and timescales. A central aim is to overcome the software crisis [1], [11]. The process for constructing software systems contributes a great deal to the reliability and effectiveness of the end product [14]. There has been a growing emphasis on viewing software engineering as dealing with more than just "coding". Instead, the software is viewed as having an entire life cycle, starting from conception and continuing through design, development, deployment, and maintenance and evolution. The shift of emphasis away from coding has sparked the development of methodologies and sophisticated tools to support teams involved in the entire software life cycle [3]. Royce (in 1970) was the first to coin the phrase " The waterfall model " to characterize the series of software engineering stages [1], [15]. It is important, therefore, to look at how software production is organized, what activities are undertaken, what priorities are

set and where costs are incurred [14]. In other words, a major objective of software engineering is the search for an adequate development approach to guide the process of developing software products and projects; this will overcome the problems encountered by both the users and developers of the products. That is, the right development process will help produce the right product. There are many different paradigms described in the literature and used during the evolutionary process of software engineering development through time. These paradigms are discussed in the next chapter.

Our objective here is to improve software systems development. Hence, we suggest a cocktail approach that can be named Structured Evolutionary Development Approach, or simply SEDA, which combines the advantages of the structured approach and the advantages of the evolutionary prototyping approach, that are complementary. This proposed approach will be tested by applying it to the development of a medical CAL application, namely computer aided medical training system, or simply CAMTS. This software system will be used in a hospital by the surgery medical personnel, namely students and physicians of all degrees, in order to attend courses and self evaluate their knowledge in their field. Our motivation to use a medical CAL as the experimental domain is to contribute in this research area. The behavioral control teaching model [16] suits this application, hence integrated. Through the question-answer process, a physician may eliminate the doubt by checking his or her thoughts before taking any action when studying a patient. This system provides the user flexibility in

selecting either to perform a test, or attend a course, and in allowing him to progress at his own speed. Moreover, this system can be used by medical students to prepare their examinations.

SOFTWARE ENGINEERING

2.1 HISTORY OF SOFTWARE ENGINEERING

The product of software engineering is a program or software system which serves a function; the creation of a software requires engineering rather than manufacturing.

The birth and evolution of software engineering as a discipline within computer science is due to the maturing and development of the programming activity [3], [11].

In the early days of computing, the problem of programming was considered as how to place a sequence of instructions together; that's, to write a program, to get the computer to do something useful. The problem was just between the programmer and the computer, no other person is involved [3], [11]. Moreover, most programmers felt that while you could learn programming, there weren't any scientific ways of writing a correct program. A good program was simply a program that worked, and the way to know if the program worked was to test it many times.

As computers became cheaper and more common, more and more people started using them. High level languages were invented in the

late 1950s, this increased the level of abstraction in the programmer-computer interactions, in other words, HLLs made easier the communication with the machine. It was at this time that programming reached the status of profession, that's, you could ask a programmer to write a program for you instead of doing it yourself. This introduced a separation between the user and the machine. Now the user had to specify the task in a form different than the programming notation, the programmer then interpreted this specification and translated it into its corresponding equivalent set of instructions. This, sometimes resulted in the programmer misinterpreting the user's intentions, even in the small tasks.

At this time, the early 1960s, only few large software projects were built by computer pioneers who were experts.

In the middle to late 1960s, large software systems were attempted commercially, and the best documented of them was the OS 360 operating system for the IBM 360 computer family. The tentative to build large software systems were the source of the realization of the difference between the building of large systems and the building of small systems. In other words, the software systems developers and users were faced with various problems which are: the software fails to do what users want it to do, it is expensive, it cannot be transferred to another machine easily, maintenance is expensive, it is unreliable, and it is often delivered late. This situation was called the software crisis, and as a remedy to it "Software Engineering" was invented around this time. Many solutions were suggested and tried for improving that situation; there was no lack of ideas,

and the final consensus was that the problem of building software systems should be approached in the same way that other complex systems such as factories, ships, and airplanes, were built. The point was to view the final software system as a complex product and its building as an engineering job. The engineering approach required management, organization, tools, theories, methodologies, and techniques [3]. The solutions are not mutually exclusive, but they complement each other. Typically, the following ideas are considered: all the development phases must be carried out systematically; use of fourth generation languages (4GLs), software development environments, and Computer Aided Software Engineering (CASE); find out "exactly" what the users really want, demonstrating an early version of a system to its users using prototyping, using new programming languages, and try to ensure that the software is free of errors. And thus was software engineering born [3], [11].

Software engineering is intended to assist the development of good quality software within budgets and timescales [1], [11], [17]. It deals with software systems for which it uses engineering principles in their development, and it consists of both the technical and non technical aspects.

Actually, the main objective of software engineering is to look for an adequate development approach to guide the process of developing software systems in order to overcome the shortcuts resulted from the crisis and the developer-user communication problems [4], [18]. Many models are proposed and described in the literature. They are used during the evolutionary process of software engineering development through time; and these will be

- ### Discussion

1. *Journal of the American Medical Association*, 1997; 277: 1033-1036.

deans of

to use metrics [8], [19]. In other words, we must always measure what we are doing, and in practical we must measure the critical factors, not simply the easily available ones, such as space and time consumption which have been the major ones so far. Particularly, maintainability and portability are becoming the major factors because of their high cost compared to the cheap computer machine resources. An other important factor is reliability which is related to the complexity and documentation of programs [20].

Software development may be viewed as a continuous process in which the following types of change take place: refinement of the specifications through several levels and corrections to the software when bugs are detected. It is recognized that the absence of metrics leads to lack of control over systems, and finally to failure [19]. The software metrics are used in order to control the software development and to yield high quality software programs [21]. The next section deals with the major software metrics which are the qualities of a well engineered software.

2.4 SOFTWARE QUALITIES

Since the software production process deals with analysis, design, and implementation, rather than manufacturing, it has to meet some criteria to ensure the production of high quality software in the cost-effective way. Software quality measurements [22] help a lot in the evaluation of products.

There are many desirable software qualities; some of them apply

both to the product and to the production process. The user wants the software product to be reliable, efficient, and easy to use, whereas the developer wants it to be verifiable, maintainable, portable, and extensible [3], [5], [8]. Some quality attributes are subjective and difficult to assess. For a particular software project, the quality attributes which are more significant are identified and a plan how to judge them is set [5].

A well engineered software system has the following five attributes:

- * **Functionality:** (Modularity) it has three main objectives, capability of decomposing a complex system into pieces, of composing it from existing modules (reusability), and of understanding the system [2], [3], [19].

- * **Maintainability:** maintainability is divided into two separate qualities, repairability and evolvability. Software is repairable if it allows the correction of errors; it is evolvable if it allows modifications that enable it to satisfy new environmental requirements without undue costs [3], [5], [20]. And it is a function of design of the system, personnel, and support facilities [19].

- * **Reliability:** reliability is the probability that the software will operate as expected over a period of time, in other words, it is a measure of how well it provides the services that are expected by the user, particularly, it is the ability of the user to use the software and get correct results [3], [5], [19], [20].

* **Efficiency (or Performance):** a software system is efficient if it uses the computing resources (space, time, and people energy) economically [3], [5], [20].

* **User interface:** a software system is user friendly if it is tailored in such a way its users find it easy to use [3], [5]. Actually, the problem encountered by the software developers is the non linear relationships that exist between these quality attributes, that is they affect each other; for example, providing a better user interface may reduce the efficiency of the system, and any improvement of these qualities can be expensive. Thus, in order to attain the optimum level of the system under development, for each type of application, the trade-offs which are required must be made explicit early in the development process [3], [5]. In the case of medical applications, it is identified that the user interface and the reliability quality attributes are the most important for the software systems to be accepted by the medical personnel [20].

2.5 THE SOFTWARE LIFE CYCLE MODELS

After considering a number of objectives and problems in software development, there was a debate about the crisis in software production. The response to these problems is the creation of a number of systematic approaches to the software development. From the inception of an idea for a software system, until it is implemented and delivered to a user, and even after that, the system undergoes gradual development and evolution. The software

is said to have a life cycle composed of several phases that are: requirements analysis, design, coding, testing, and maintenance. Each of these phases results in the development of either a part of the system or something associated with the system, such as documents [0], [11], [20]. The life cycle is generally defined as follows: The life cycle may be regarded as a management and technical tool for organizing, planning, scheduling, and controlling the activities associated with a software project development and software maintenance efforts [1], [2], [3], [9], [17], [24], [25], [26].

A major objective of software engineering is the search for an adequate development approach to guide the process of developing software products and projects in order to overcome the problems encountered by both the users and developers of the products. There are many different paradigms described in the literature and used during the evolutionary process of software engineering since its birth. They vary according to the degree of detail being considered and the prevailing philosophy used to interpret the development task [1]. The next sections deal with the existing models of the life cycles.

2.5.1 The Slum Dunk Life Cycle Model

In this model, we begin coding (implementation) as soon as the project starts. The idea behind this approach is that the developer supposes the generation of many errors from the code, that's why he deals with coding from the beginning in order to minimize those bugs. Therefore, the system code is generated with

the system requirements analysis. In this model the different phases are confused or melted in one phase which is the coding phase [2]. This model is depicted in figure 2.1.

2.5.2 The Baroque Life Cycle Model

The concept of this approach is a response to the lack of discipline and structure exhibited by the Slum Dunk approach. The concept is that each stage of the development process will be completed before the next begins. Actually, this approach is not as great as it sounds, because we may face the situation in which one phase, say the analysis phase, is undeterminate; that's, the exit criteria is not reached. The fact may arise from refining the system requirements indefinitely, hence, neglecting the remaining phases. This life cycle model does not work for the simple reason that software development is not a deterministic activity [2]. This model is shown in figure 2.2.

2.5.3 The Waterfall Life Cycle Model

This model, which is the most widely known, is introduced by Royce in 1970 and popularized by Boehm [37]. It describes the generic process that many software developers follow, at least to some extent. This model attempts to correct the shortcomings of the Baroque approach by recognizing an advantage in having interactions among phases. By interaction is meant phase overlapping with respect to time; namely, the results of one phase are fed into the next, beginning with the analysis phase.

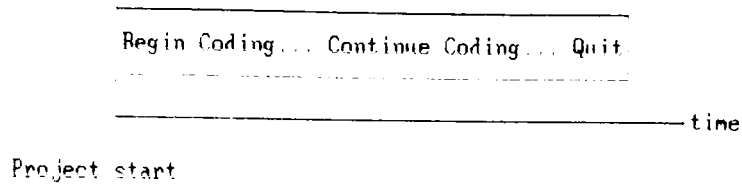


Figure 2.1 : The Slum Dunk life cycle model time line [5], sec 1.2, p 6

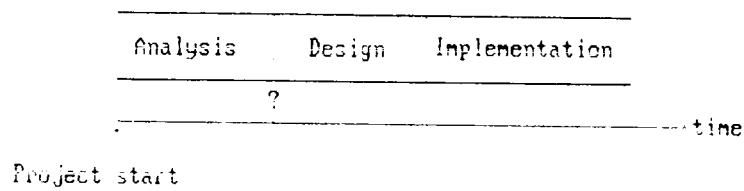


Figure 2.2 : Time Flow associated with the Baroque life cycle model [2], sec 1.3, p 7

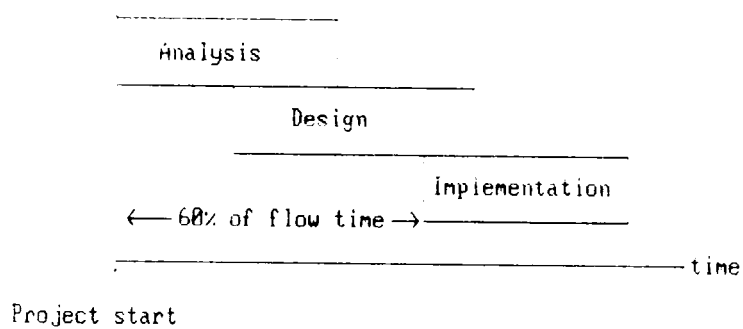


Figure 2.3: Time line associated with the Waterfall life cycle model [2], sec 1.4, p 8

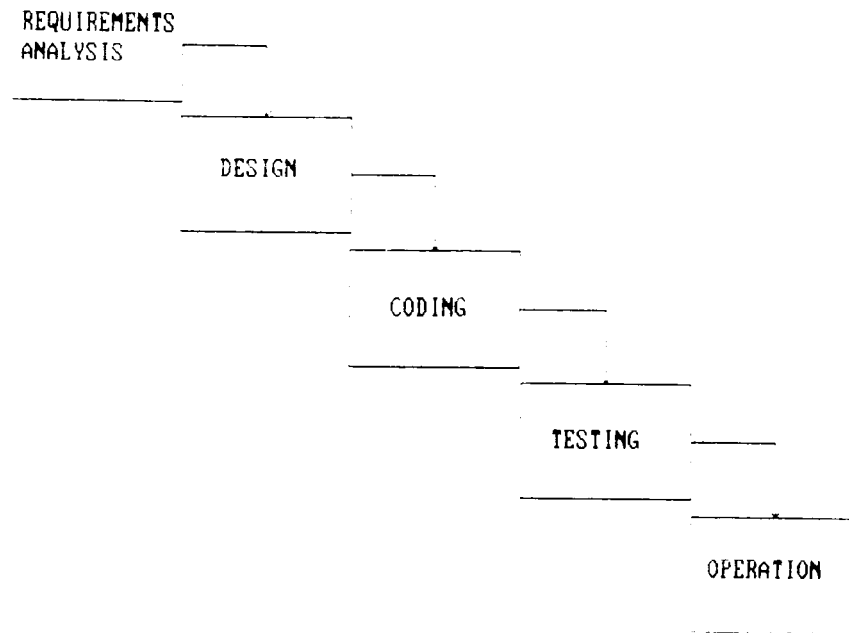


Figure 2.4(a): Classical Waterfall life cycle model [37] sec. 5.2.1, p83

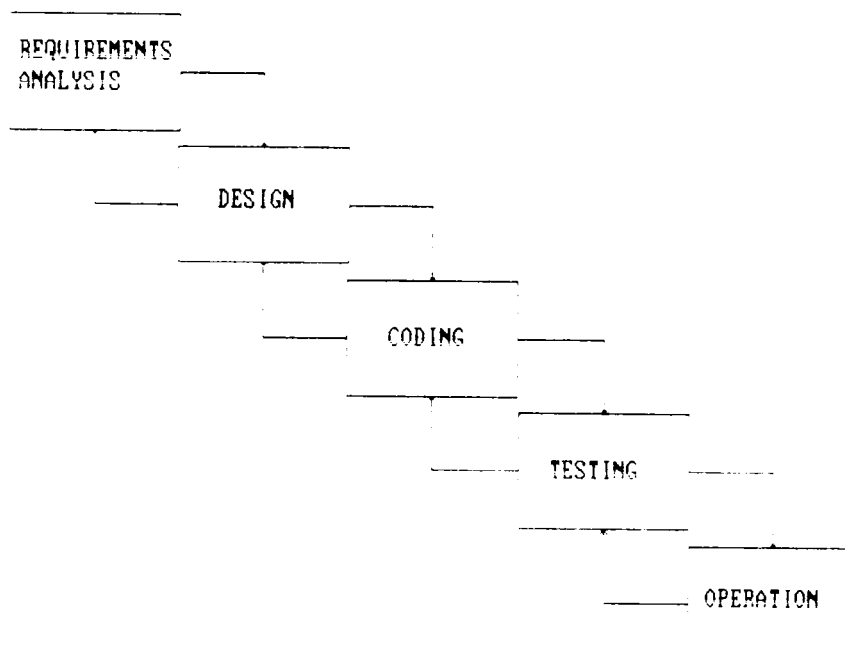


Figure 2.4(b): Standard Waterfall life cycle model [15], sec 1.1.2, p 8

Much criticism of the waterfall has ensued, however, which has resulted in more refined and enhanced models. The time line of this model is shown in figure 2.3 and two models of this life cycle are pictured in figure 2.4.

Note that this model follows almost the same steps involved in the other engineering disciplines, namely system analysis (data collection and system requirements), then software design (the blueprint) and finally, software implementation (construction). In figure 2.4.b, the arrows between two adjacent stages are bidirectional, because during any stage we may learn something causing us to return to the previous stage, to update its results. As useful as this approach, it has some shortcomings. The major one is that in a long term development effort, it is often more than a year before any working product or model of a product is available for the user to examine and critique [2], [3], [27].

2.5.4 The V-Diagram Life Cycle Model

It is recognized that the waterfall model fails to show the symmetry that exists between the earlier and later stages of the development life cycle, despite the importance of these relationships in determining activities, priorities and costs estimation. An alternating way of presenting the phased process is offered which shows these relations, the V-diagram life cycle model shown in figure 2.5.

The planning boxes point to the fact that it is inappropriate to wait until the testing stage to determine how your are going to

test.

In the 'V', the left side shows the development phases while the right side shows the validation phases. Each phase in the validation side validates its corresponding phase in the development side [15].

2.5.5 The Exploratory Programming

Exploratory programming is based on the idea of developing an initial implementation that will be deployed and used by the user. Then it is refined and enhanced through many stages until an adequate system is obtained. There is no system specification. Therefore, systems developed by this concept may be unspecifiable. This approach is depicted by figure 2.6

This concept suits for systems where it is difficult to establish a detailed system specification. The key to success is to use a rapid system iteration process so that changes may be incorporated and demonstrated quickly. This implies the use of a very-high level programming language [5].

2.5.6 The Prototyping Life Cycle Model

While the exploratory programming approach produces an unspecifiable system, the prototyping approach is intended to discover the system specification so that the end product will be specified [5].

Prototyping is a standard practice in almost all engineering professions. It is a useful assistant tool. Its main advantage

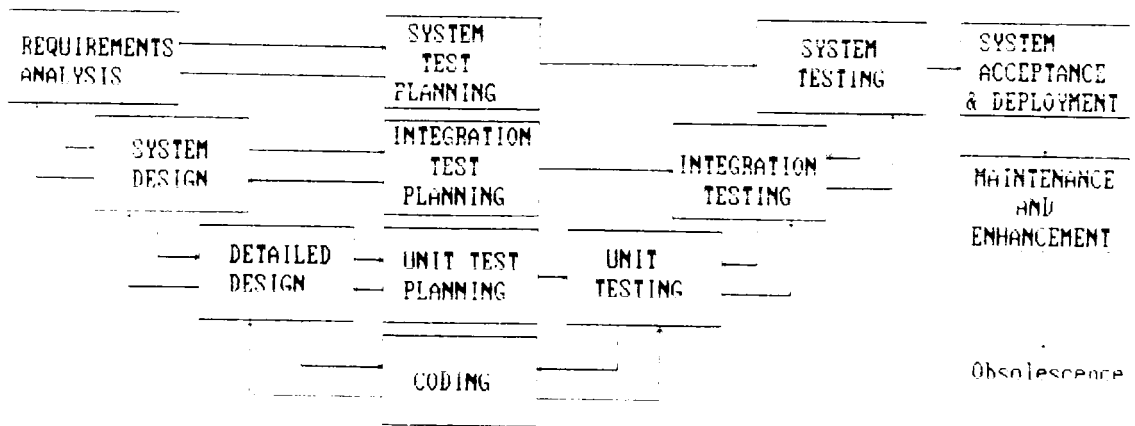
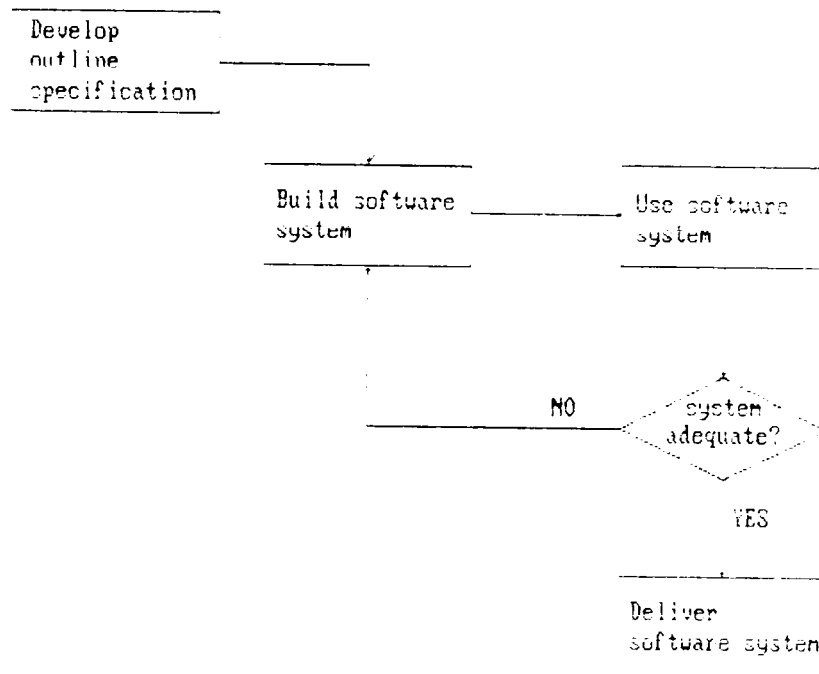


Figure 2.5: U-diagram life cycle model [15], sec 1.1.2, p 10



Exploratory programming approach [5], sec 1.2.2, p 12

so that the users have an early feel of the final system before it is built; moreover, prototyping is a powerful means of defining and refining the system requirements and improves communication with the end users [2], [5], [14], [15], [27], [28]. This model is pictured in figure 2.7.

This approach extends the requirements analysis with the intention of reducing overall life cycle costs. It is based on the assumption that the prototype is developed from the requirements, delivered for experiment and modified until the end user or client is satisfied with its functionality. Then a specification is delivered from the prototype and the system is re-implemented in a final version following the phased life cycle model [5], [15].

The prototyping approach is compatible with the traditional life cycle. A throwaway prototype can be built during any development phase of the life cycle [15], [29].

2.5.7 The Evolutionary Development Approach

The exploratory programming approach adopted to system development provides the user with an incremental incomplete system, then refining and augmenting that system as the user's requirements become apparent until the final system is obtained. The output of this approach is an executable system.

The evolutionary development model combines the advantages of exploratory programming with the control required for development, it involves developing the requirements and delivering the system incrementally. This model is depicted in

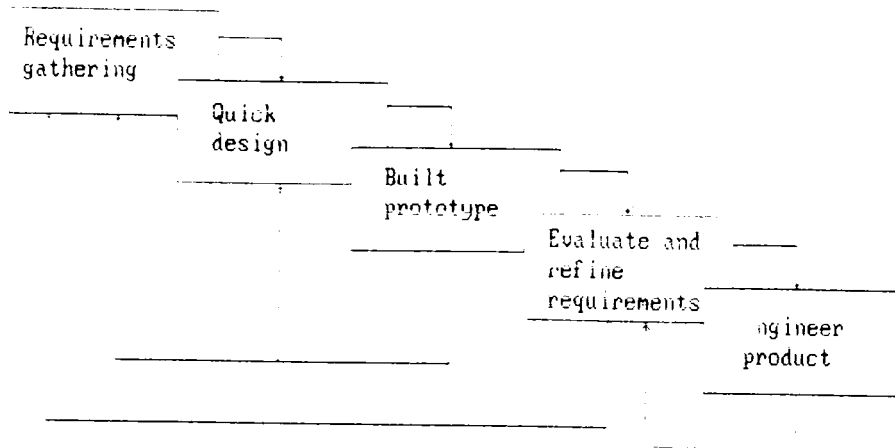


Figure 2.7: Prototyping [6], sec 1.5.3, p 23

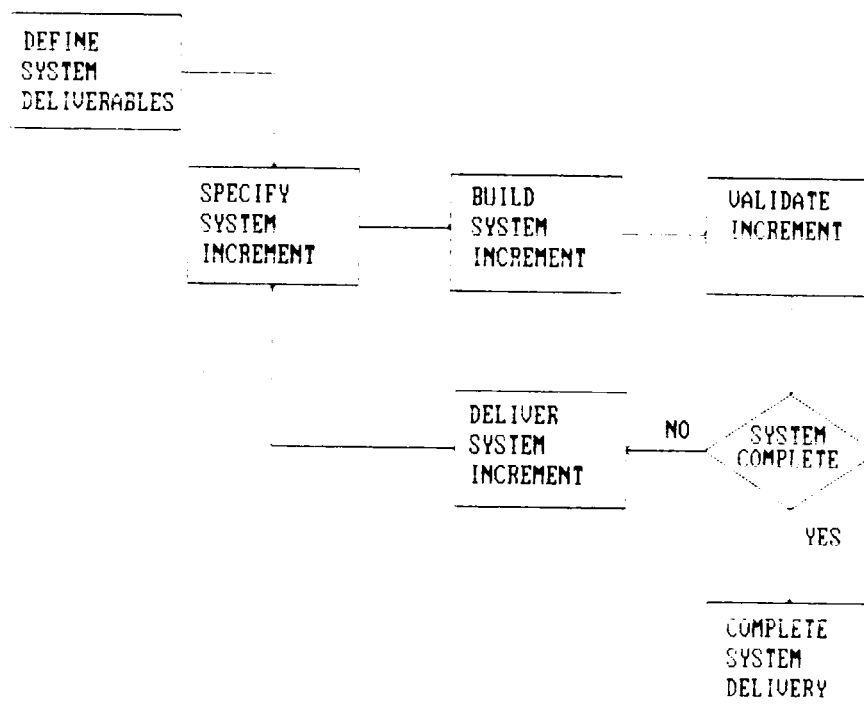


Figure 2.8: Evolutionary development model [5], sec 6.1, p 115

As part of the system is delivered, the user may experiment with it and provide suggestions and new requirements that will be taken into consideration in later parts of the system. However, this approach suits only mass production scheme which incurs the development cost [3], [5].

2.5.8 The Spiral Model

Even a well organised project needs to proceed iteratively, doing some analysis, then more design, then coding the first version of the project, then more design, and so on. The path of such project can be pictured as a spiral, as shown in figure 2.3.

The idea behind this approach is that at each phase we build a skeleton, first logical and then physical, see how well the skeleton works, and then go back to put the flesh on the bones. Prototyping is involved at each phase [3], [30].

The main characteristic of this model is that it is cyclic and not linear like the waterfall model. This approach reduces risks that may impair the development process and the quality of the product [3].

2.5.9 The Fourth Generation Techniques

They encompass a large set of software tools that have one thing in common; each enables the software developer to specify some characteristics of software at a high level. The tools then automatically generates source code based on the developer's specification. The higher the level at which software can be

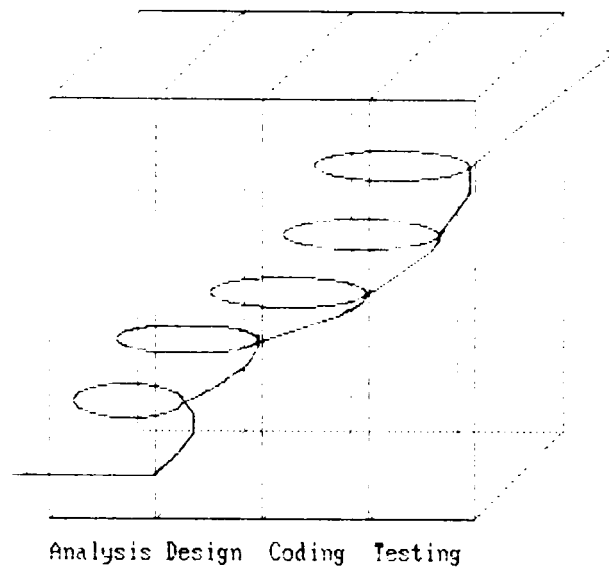


Figure 2.9: Spiral life cycle model [30], sec 10.1.1, p 225

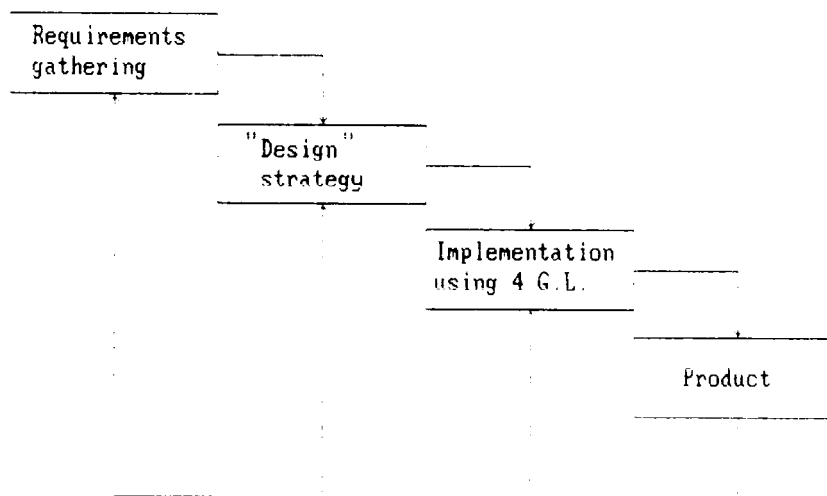


Figure 2.10: Fourth Generation Techniques [6], sec 1.5.4, p 24

specified to a machine, the faster a program can be built.

The fourth generation paradigm for software engineering specifies software to a machine at level that is close to natural language. A software development environment supporting the fourth generation techniques (4GT) concept includes some or all of the following tools: nonprocedural languages for database query, report generation, data manipulation, screen interaction and definition, and code generation, and high level graphics capability. This paradigm is depicted in figure 2.10.

This approach begins with the requirements gathering step. For small application, it may be possible to move directly from this step to the implementation step. However, for larger applications it is necessary to develop a design strategy. Without design the system will have the following disadvantages: poor quality, poor maintainability, poor user acceptance, etc. Implementation using a fourth generation language (4GL) enables the software developer to describe desired results which are translated automatically into source code to produce those results. The last step in this model is "product"; that's, after testing the system it is delivered to the user as a product [5], [6].

2.5.10 System Assembly From Reusable Components

This approach assumes that systems are made up of components which already exist and the system development process assembly rather than creation.

The advantage of software reuse is the costs reduction as the number of components that must be specified, designed, and

implemented in a software system is reduced. A systematic software reuse offers a number of advantages; such as reliability, risk reduction, development time reduction, and so on [3], [5], [31], [32], [33]. The problem is not the lack of reuse but a lack of systematic reuse. Software programmers use subroutines and algorithms since programming was invented; but they do all this informally. Thus, the reuse community is focusing on formalizing reuse because it recognizes that the software industry will achieve substantial quality and productivity payoffs only if reuse is conducted systematically and formally [33].

2.5.11 The Transformation Model

The idea behind this concept is the development of a formal specification of the software system and continue the development as a sequence of steps that gradually transform the specification, using correctness-preserving transformations, to a finished executable program. This approach is illustrated in figure 2-11.

The formal nature of the derivation may provide a mathematical check that one step is a correct transformation of the previous [3], [5].

A formal software specification is a specification expressed in a language whose vocabulary, syntax and semantics are formally defined; and this language is not based on natural languages but on mathematics, such as the specification language Z [34], [35], [36], that is used as a basis for describing model based

specifications [5].

The simplest form of formal specification is axiomatic specification where a system is represented as a set of functions and each function is specified using pre- and post-conditions. These conditions are predicates over the inputs and outputs of a function. And a predicate is simply a boolean expression which is true or false and whose variables are the parameters of the function being specified [5].

The transformation approach has been studied for small programs as a dual method for proving program correctness. Program correctness proofs represent an analytic, mathematically based approach. Transformations, instead, are a constructive, mathematically based approach. The transformation process requires skill and creativity from the programmer [3].

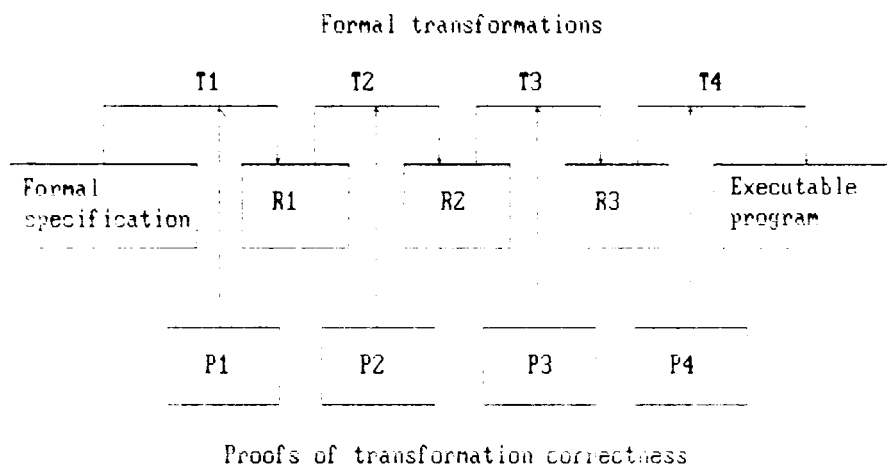


Figure 2.11: Transformational software development [5], sec 7.1, p 127

STRUCTURED METHODS

3.1 INTRODUCTION

A structured method refers to structured analysis [2], [3], [15], [24], [30], [37], [38], [39], [40], [41], structured design [2], [3], [11], [24], [38], [41], [42], [43], [44], [45], and structured coding [7], [11], [45], [46]. They can be used in conjunction with all the life cycle models we have discussed in the previous chapter [2].

The system begins with the user who has a need for technical support in order to computerize a part of his organization, if not the whole enterprise; but does not know enough about the computer and its capabilities and limitations. From the other side of the organization are the designers and programmers who know enough about the computer, but do not have a clear understanding of the user's needs. In other words, the user knows the problem but cannot solve it; while the software developers might be able to solve it, if they understand it. So the problem can be seen as being a communication gap between the users and developers since they speak different languages and each part with its way of conception [38].

Until the development of the structured systems, (analysis and design) tools, there was no means of showing a clear picture of the system and how its parts fit together to meet the user's needs. The problems encountered in the analysis can be summarized as:

1-The analyst has difficulties in understanding the needs as seen by the user.

2-The user community does not know enough about data processing to know what is feasible and what is not.

3-The analyst is overwhelmed with both the detail of the system and the technical detail of the new system.

4-The document of the details of a new system which forms a contract between the users and the system developers is generally not understood by the users because of its technical aspects.

5-If the specification document is written in a user's conception point of view, it may not make a sense to the physical developers who have to build the system [30].

The characteristics of a classical model are:

- Strong tendency toward bottom-up development of the system.
- Insistence on linear, sequential progression from one phase to the next [37].

The structured methods consist of an evolving set of tools and techniques which are resulted from the maturing of structured programming [7], [11], [46] and structured design [2], [15], [38], [39], [42], [43], [44]. The underlying approach is the building of a logical (non physical) model of a system, using graphical techniques and tools which enable users, analysts, and designers to get the " right " picture of the project and how its

parts fit together to meet the requirements or the users' needs. However, some of the problems are always with the developers because there is no way to know what is in user's mind without being told [30]. Hence, system development involves both technical and non-technical aspects [12], [13], [30].

The need for structured methods is evident from the many problems encountered with the classical methods of developing software systems; among these problems we find:

- The products do not meet the user's needs.

- High maintenance cost.

- Bottom-up approach which has the following difficulties:

 - Nothing is done until it is all done.

 - The most trivial bugs are found at the beginning of the testing period but the most serious bugs are found last.

 - Debugging tends to be extremely difficult during the final stages of system testing.

 - The requirement for computer test time usually rises exponentially during the final stages of testing [37].

 - The system is telling us just what it is structured like, which is the contrast of the top-down approach in which the developers impose their view of how the system will be structured [2], [47].

Actually, there are better ways of developing software programs and systems; if we understand logically what we want to do, that's, the problem, we can construct a software program (solution) to do it. In other words, there is a theory to guide the activities of the software development process. The idea behind this theory is to work all the logical possibilities in

detail and build the software program, or system, in a hierarchical logic fashion. This will yield correct software systems or products [84].

The structured methods improve the quality of the final product and decrease the maintenance cost. The characteristics of the structured methods are: top-down modeling, iteration, modularity, flexibility, low development cost, less complexity, maintainability, and reliability [13], [15], [24], [30], [38], [47].

3.2 DEFINITIONS

A methodology is a document set of practices and procedures that define the development life cycle and specifies how systems are to be developed [48].

When they were introduced first, structured methods were considered as being a system of management and technical practices and procedures [2].

3.3 THE STRUCTURED METHODS AND THE LIFE CYCLES

A structured method refers to structured analysis, structured design and structured coding. These are used in conjunction with all the life cycles [2].

Systems development involves a set of activities such as analysis, design, implementation, and testing throughout the life cycle. In this section, we will see the role of the structured methods in each phase of the software life cycle.

3.3.1 The Analysis Phase

Analysis is the study of a software problem prior to taking some action. The idea is that analysis binds only the solution space and sometimes it is necessary to do some design and implementation via prototyping to determine the solution boundary and to help the analyst-user communication. In other words the analysis phase is the organization of the information gathered by the analyst from the users, into a meaningful form [24]. This underlies the building of a logical (non physical) model, using graphical techniques and modelling tools, which represents a description of a system that will be built [2], [3], [24], [30], [37], [40], [41].

Analysis is a logical process and its objective is not to solve the problem, but to determine exactly WHAT must be done to solve the problem [2], [24], [38], [41]. Actually, the analysis phase goes through two major stages:

- a Physical model stage: This involves an examination of the current system through an analysis of its physical properties.

- b Logical model stage: This involves the abstraction or logicalization of the physical model.

The exit criteria of this phase is a logical model of a system showing all the functions, data, and relations that will be implemented. The graphical model of this phase is shown in figure 3.1. This logical model is subject to review by both the user and the developers.

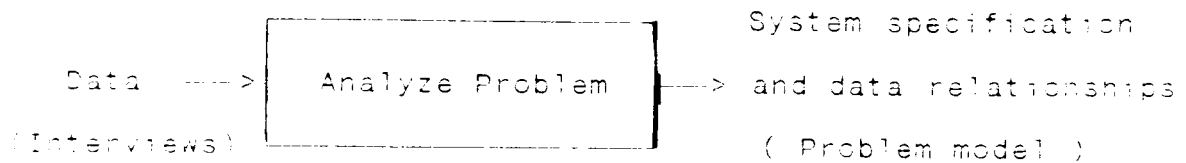


Figure 3.1(a): Graphical model of the analysis phase [2],
sec 1.6.1, p 11.

The analysis phase is also referred to as the requirements or specification phase. The results of this phase are the basis for the design phase which is the next in the development process [2]. The analysis phase is depicted graphically in figure 3.1 (a).

3.3.2 The Design Phases

Structured design is based on the research done by Larry Constantine in 1962 [49]. It is a program design technique that reduces the effort necessary to implement and maintain programs, and has the following characteristics: low development cost, ease of maintenance, flexibility, modularity, ease of use and reduces complexity [2], [7], [8], [24], [38], [39], [40], [44], [45], [47].

Structured design proceeds iteratively, it takes a logical model of a system, produced by the analysis phase, and produces the specification of a physical system that will meet the user's needs. In other words, the goal of this phase is the construction of a solution model, typically, a blueprint. Actually, this is only a model of the solution, not the solution itself since the latter refers to the delivered system [2], [3].

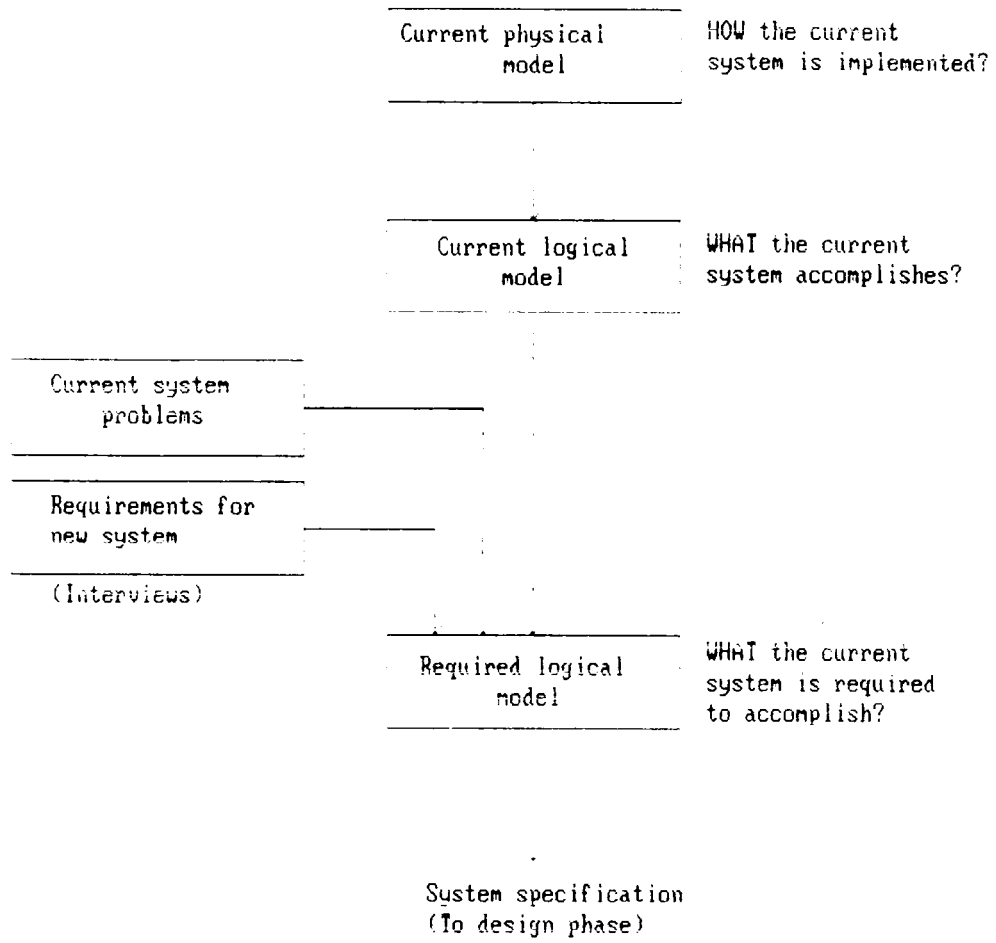


Figure 3.1(b): Analysis phase [24], sec 1.8.3, p 13

Inputs to this phase include the output of the analysis phase (specification documents, such as DFDs), experience, system knowledge, and the method(s) to be applied to arrive at a solution model.

To achieve high quality of design, the software designer must address two important related issues. first, a careful definition of the modular structure of the software system under development must be provided, identify the modules and their relationships. Second, appropriate criteria must be chosen for decomposing a system into modules [3].

We recall that we distinguish the work of analysis which is defining "WHAT" the system will do from the work of design which is defining "HOW" it will do it, recognizing that analysts often do design and designers often do analysis.

There are two design phases, the system design phase and the detailed design phase:

3.3.2.1 The System Design Phase

This phase, also referred to as logical design, preliminary design, architectural design, and high level design, creates a design which will satisfy what was specified in the analysis phase. It results in the identification of modules and corresponding control structure [2]. This design, however, does not include implementation considerations, constraints, programming language features etc.

3.3.2.2 The Detailed Design Phase

This stage, also referred to as physical design, produces the

final blueprint of the system [2], [8], [24], [38]. This phase takes the results of the logical stage and applies constraints, details of language and hardware, etc. The blueprint consists mainly of algorithms. The information flow through this phase is shown in figure 3.2 below.



Figure 3.2: Design Phase Information Flow [2], sec 1.6.2, p 12.

The output of this phase feeds the input of the implementation phase.

3.3.3 Implementation Phase

The goal of this phase is to implement the system according to the blueprint or solution model resulted from the detailed design stage; that's, this phase transforms the algorithms defined during the detailed design stage into a computer-understandable language [9], [15]. The programming languages contribute in software quality; a high level language program represents the easiest path to software quality as the software should be understandable [46]. In other words the system is physically created in a top-down fashion [7]. The output of this phase is the code of the system. The information flow associated with this phase is presented in figure 3.3 below.

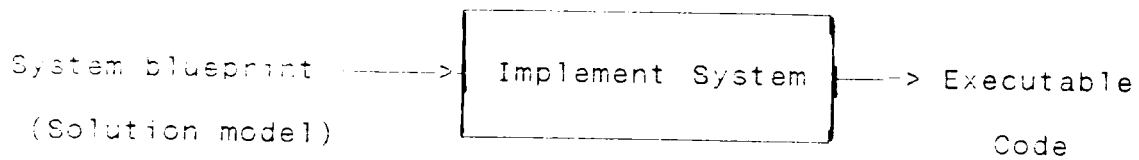


Fig 3.3: Implementation phase Information Flow [2], sec 1.6.3, p 13.

3.3.4 Testing Phases

If humans are perfect, the development process ends at this point, implementation. Unfortunately this is not the case; therefore, we need a testing process to correct the bugs.

Testing may show that the system does not exhibit the expected performance and functionality [19],[20], [22]. Once the code of any product is developed, program testing begins. Testing is to assure that for the defined input to the system under consideration will produce the expected output. Testing is a critical activity in software engineering, hence it should be performed in a systematic way by stating clearly what output one expects from the system and how one expects to obtain the output [3], [5], [6], [19], [20], [50], [51]. It is a means of analyzing the behavior of a system, and it enhances the developers confidence in systems qualities [3], [47]. Thus, testing:

- should be based on structured and systematic techniques so that, after testing, we may have a better understanding of the system's reliability.

- should help locate errors, not just detect their presence.

- should be repeatable, that's, repeating the same experiment by supplying the same input data to the same program, produces the same output. If the output is different just once, the system is not correct.

- should be accurate: this will increase the reliability of the testing process [3].

The structured testing process has generally three phases, and these are discussed in the following sections.

3.3.4.1 Unit Testing Phase

This phase is also called module testing and functional testing. During this phase, each individual module is tested in order to check if it behaves according to its specification defined during the detailed design phase and errors, if any, are corrected [3], [5].

3.3.4.2 Integration Testing Phase

This phase is also referred to as string testing and computer software component (CSC) testing [15]. This phase interconnects sets of the previously tested modules to ensure that the sets or subsystems behave as well as they did as independently tested modules and according to their specification defined during the system design phase. The integrated sets of modules should correspond to a component or subsystem in the design tree defined during the preliminary design stage of the design phase [3], [5].

3.3.4.3 System Testing Phase

This phase checks that the entire or fully integrated software

system put in its actual hardware environment behaves according to the software requirements specification (SRS) or user's needs [3], [5], [15]. Until now, all the testing activities are done only by the developers. We recall that the testing process is mainly concerned with detecting program errors and checking that the program conforms to its specification. Hence, the last stage or phase is the acceptance testing which is the process of testing the system with real data under the realistic conditions in the presence of the users; this type of testing is called alpha testing [3], [5], [8]. System testing generally shows errors in the system requirements definition.

In the case where a system is intended to be distributed as a software product, a testing process called beta testing is performed on the product under consideration. The beta testing process involves the delivery of the system to a number of selected customers or users. And on the basis of their feedback, the developers will determine whether any changes are necessary before the official release of the product [3], [5], [8].

3.3.5 Maintenance Phase

Maintenance is the enigma of software, large amounts of money are spent on it. It is the act of keeping a delivered software product functioning in a satisfactory way [20], [52].

As the system passes all the tests, it becomes operational and is delivered to the end-user then enters the maintenance phase. Any changes made to the system after its delivery are attributed to this phase [3], [3], [15], [20], [52]. This phase is actually

a full development life cycle; the reason for this is simple, whenever a need is identified, analyzed, then a modification is designed then implemented, after this the subsequent testing phases must be performed [2], [15], [20].

It is shown by [20] that maintenance is the dominant phase on a cost analysis basis, the design phase as the dominant phase on an error creation basis, and system testing and maintenance as the dominant phases on an error discovery basis.

The cost of the software maintenance depends on the phase where the error is detected. The error is detected sooner in the development process, the maintenance cost is smaller [20], [30], [37], [53]. Figure 3.4 shows the relative cost of fixing an error during system development, with a vertical logarithmic scale, depending on the phase of the system development life cycle where the error is detected. Thus an error which shows up in the requirements phase might cost one tenth of the same error when detected in the operation phase [30]. Therefore, the building of a logical model which clearly communicates to users what the system will and will not do is very important in terms of the cost of fixing errors later on. Moreover, it was shown that maintenance is not just the fixing of errors, but it has three activities [20]:

Perfective maintenance: it is the act of improving the function of the software by responding to user defined changes. This maintenance activity consumes 60 % of the software maintenance time.

Adaptive maintenance: it is the act of changing software to

adapt it to environmental changes; 18 % of maintenance is adaptive.

-Corrective maintenance: it is the pure correction of software errors; it consumes only 17 % of the maintener's time.

And the remaining 5 % of the maintener's time are allocated to others.

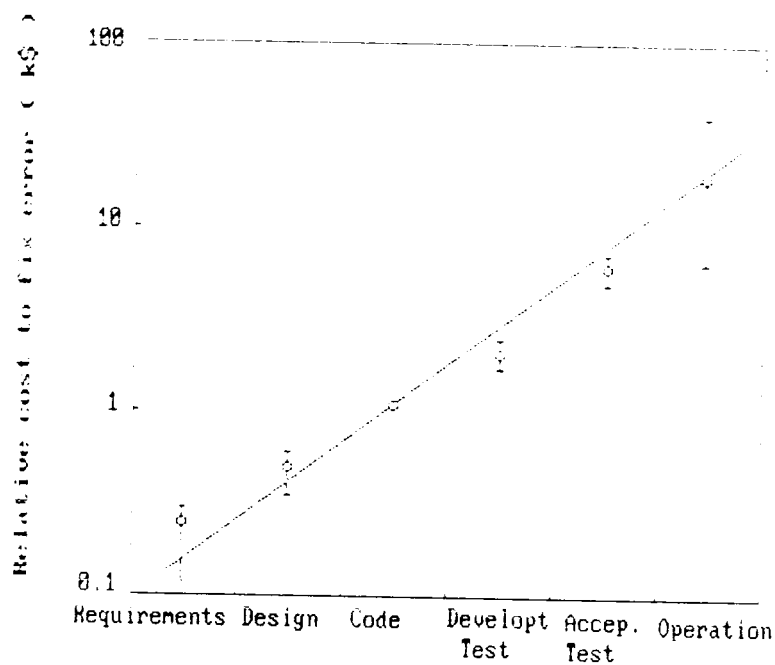


Figure 3.4: Relative cost to fix an error during system development [30] sec 1.3, p 7.

MODELLING TECHNIQUES

4.1 INTRODUCTION

We have seen that the software development process consists of phases. Each phase is designed to achieve a specific objective or set of objectives. The achievement of these objectives is engineered by using a set of tasks within each phase. Each task is based upon the most appropriate technique. A phase may therefore use several techniques to achieve its objective and a technique may be used in different phases [24]. During the analysis phase, the analyst performs an investigation; that's, the analyst talks to the user and constructs diagrams to record his understanding of the discussions. The diagrams will be augmented with other information that the analyst has found during his study. At this point, the following question may be asked: Why use diagrams ? The answer is that the problems of communication between the users and the developers was the inadequacy of an English narrative specification of a system. The diagrams have been shown to provide an excellent and unambiguous means of communication between users and developers [39].

Following are the characteristics of the diagramming techniques:

- 1-Make easier the communication between the users and developers.
- 2-Provide a means of defining the system boundary.
- 3-provide a means of defining partitions, abstractions, and projections.
- 4-Encourage the analyst to think and document in terms of the problem as opposed to the solution.
- 5-Allow for opposing alternatives but alert the analyst of their presence.
- 6-Make it easy for the analyst to modify the knowledge structure [15].

The diagrams, used initially for communication, fact finding, note taking and discussion, evolve as the work progresses to provide the base on which the system will be structured. They are supported by detail in other documents, but the correctness of the diagrams is the key to successful systems. A picture they say, is worth a thousand words. The diagrams convey a lot of information in a simple way [39]. Thus, modelling tools are used to:

- 1 focus on important system features while de-emphasizing less important features.
- 2-discuss changes and corrections to the user's needs with low cost and minimal risk.
- 3-validate the role of the analyst, as being the intermediate between the user and the designers and programmers [37].

Actually there are many different methods, and each one has its

view or views of the system to be developed; hence, appropriate modelling techniques are used according to the view that is under consideration. In fact, there are three views of the system, the process communication view, the data view, and the time view [39], [41], [24].

The modelling technique that is used in the process communication view is the Data Flow Diagrams (DFDs) [11], [24], [37], [39], [41], [48]; the techniques that are used in the data view are the Entity Relationship Diagrams (ERDs) [2], [37], [54], [55], [56], Logical Data Structure (LDS), Enquiry Access Path (EAP), Relational Data Analysis (RDA), and Enquiry Process Model (EPM); and the techniques that are used in the time view are the Entity Life Histories (ELHs), Effect Correspondance Diagrams (ECDs), and Update Process Models (UPMs) [24], [39], [41]. Other modelling and supporting techniques are Data Dictionary (DD) [37], [43], Structure Charts [2], [37], Flowcharts [45], and Pseudocode [39]. As already mentioned above, a phase may use several modelling techniques to achieve its objective and a technique may be used in different phases of the software life cycle.

The methods are spreadout throughout the world, the most popular method in the USA is Yourdon [37], whereas the European scene is very fragmented, MERISE method [57] in France, DAFNE method in Italy, SDM and NIAM methods in Germany, and SSADM method in the UK [24], [39], [41].

The nature of the application dictates the method and the view or views to be used. In the light of the present work, the Yourdon method and the process communication view, or functional view, of a system is being used, hence the techniques that will

be used are: The data flow diagrams and data dictionaries in the analysis phase; Structure charts, flow charts, data dictionary, and pseudocode in the design phases. Hence, only these techniques are introduced in this chapter, and guidelines to follow when developing data flow diagrams, structure charts, and data dictionaries are provided in appendix A.

4.2 DATA FLOW DIAGRAMS (DFDs)

4.2.1 Definition

A Data Flow Diagram (DFD) is a picture of the flows of data through a system of any kind showing the external entities which are sources or destinations of data, the processes which transform data, and the places where the data are stored [11], [30], [48].

The characteristics of a DFD are :

- 1 DFD consists of graphs or pictures and supporting textual modeling tools, they show the functions that the system must perform.

- 2-DFDs concentrate on the process communication view and are used in a top-down fashion with each level being decomposed into low level.

- 3 Users and developers find the notation easy to understand, this highlights the usefulness of the diagramming techniques.

4.2.2 Notation

The basic building blocks for DFDs are:

- 1-Terminators or external entities.

2-Processes.

3-Data stores.

4-Data flows.

The symbols used on DFDs differ from one method to another. Figure 4.1 shows the notations used in three methods SSADM, Yourdon, and Gane & Sarson.

4.2.3 Definitions

a-The terminator

A terminator shows the external entities with which the system communicates; typically, a terminator is a person, a group of people, or a department.

b-The processes

They represent the various individual functions that the system carries out. The functions transform data inputs into data outputs.

c-Data stores

They show collections of data, at rest; typically, they are files or databases.

d Data Flows

The data flows are used to describe the movement of data from one part of the system to another part.

The possible and legal connections between the DFD components using data flows are summarized in figure 4.2.

	SSADM [24], Chap. 10	YOURDON [37], sec. 9.1	GANE & SARSON [38], sec. 3.1
PROCESS	<div> <div>ID</div> <div>WHO</div> <div>WHAT</div> </div>	<div> <div></div> <div>IN</div> <div>R.T.S.</div> </div>	<div> <div>ID</div> <div>FUNCTION</div> <div>LOCATION</div> </div>
DATA STORE	<div></div>	<div></div>	<div>ID</div>
EXTERNAL ENTITY OR TERMINATOR	<div></div>	<div></div>	<div>a</div>
DATA FLOW	<div></div>	CONTROL FLOW <div></div>	<div></div>

Figure 4.1: Symbols used in DFDs in SSADM, Yourdon, and Gane & Sarson

	Process	Terminator or External Entity	Data Store
Process	YES	YES	YES
Terminator or External Entity	YES	YES	NO
Data Store	YES	NO	NO

Figure 4-2: Legal connections between DFD components
[24] chapter 10, p 53

Actually, both the one-line and two-line data flow digrams can be used, however the two-line DFD is more clear than the one-line DFD because it expresses the difference in time [24].

4.2.4 How Many Processes To Show On a DFD ?

It should be mentioned that the construction of DFDs is not a science but an art, and that one cannot give rules to be followed but guidelines. We recall that DFDs are used for communication between the users and the developers, thus clarity of communication should be ensured. In order to have good communication document, DFD, there must be a systematic relation between what goes in and what comes out. Hence, we have to take into consideration that there are limits on our capacity for processing information; this leads software developers to the use of the golden or magical number seven, plus or minus two. A research work, both technical and psychological, about this question is done by Miller [12]. And the guidelines to follow in developing the DFDs are given in appendix A.

4.3 DATA DICTIONARY (DD)

4.3.1 Definition

A DD is a data store that describes the nature of each piece of data used in a system [2], [30], [48].

4.3.2 Notation

A DD is a widely used technique for supplementing or documenting the graphical models that result from system analysis and system

design. The notation suggested for a DD is capable of representing the basic kind of relationships which exist between data items and elements. These relationships are:

1-Concatenation.

2-Iteration.

3-Selection.

Other types of information that enhance communication of definitions are:

4-Definition or composition.

5-Options.

6-Comments.

7-Values.

The symbols associated with the data dictionary are presented in figure 4.3 below.

Name	Symbol	Meaning
Composition	=	is composed of, consists of
Concatenation	+	and
Iteration	{ }	multiple occurrence of
Selection	[/]	select one of the alternative choices
Option	()	may or may not be present
Comment	* *	additional information
Discrete value	" "	the value of this variable

Figure 4.3: Data Dictionary Notation [2], sec 3.2, p 31.

4.4 PSEUDOCODE

A supplement tool to describe the process logic is the pseudocode presented in this section.

Pseudocode is used to describe the logic of a process. Pseudo means similar to; thus, pseudocode is similar to the programming code. It serves the two basic purposes: it bridges between natural and programming language and acts as a means of expressing thoughts about design and the definition of programs. Superficially, it looks like a program written in a high level language [8]. It is an alternative to structured English. When using structured English, details such as opening and closing files, initializing counters, and setting flags are often ignored; with pseudocode, they are coded. The idea is to describe the algorithm of the executable code in a form understood by the programmer. Pseudocode incorporates the three structured programming conventions: sequence, decision, and repetition [7], [38], [45]. We will use a pseudocode borrowed from the DBASE IV programming language [49], [58], [59] that is used in the present work.

* Sequence

The logic is executed in a simple sequence, one block after another. A block may consist of one or more instructions.

* Decision

```
IF < Condition >  
    < Commands >  
[ ELSE
```

```
        < Commands > ]  
ENDIF
```

If the < Condition > following IF is true, the subsequent commands are executed. If < Condition > is false, the commands in the ELSE clause are carried out. This continues until ENDF is encountered.

* Repetition

```
DO WHILE < Condition >  
    < Commands >  
[ LOOP ]  
[ EXIT ]  
ENDDO
```

The command statements between DO WHILE and ENDDO are repeated while the specified condition is true. EXIT and LOOP commands change the flow of control within the DO WHILE command.

* The CASE structure

```
DO CASE  
    CASE < Condition >  
        < Commands >  
[ CASE < Condition >  
    < Commands > ]  
[ OTHERWISE  
    < Commands > ]  
ENDCASE
```

It is a structured programming command that selects only one course of action from a set of alternatives.

4.5 STRUCTURE CHART

4.5.1 Definition

The structure chart is the most widely recognized tool used in structured design [2]. It is a logical model of a modular hierarchy, showing invocation, intermodular communication, data and control, and the location of major loops and decisions [30], [47]. That's, it shows the binding and the coupling characteristics of the modules [43], [47].

4.5.2 Notation

The structure chart uses the four primary symbols shown in figure 4.4 below; but other symbols are also used to describe specific conditions and system properties .

4.5.2.1 Definitions

a-Module

A module is a set of program statements that can be invoked by a name [43]. Examples of modules are: function, procedure, subroutine, etc.

b-The call

It is any mechanism that transfers control from one module to another.

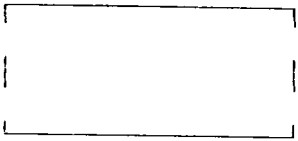
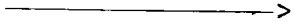


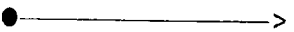

Symbol	Meaning
	Module symbol
 or 	Call symbol
	Data couple
 or 	Control couple

Figure 4.4: structure chart symbols [2], sec 9.2, p 155

c-The data couple and the control couple

Two types of information can be communicated between modules, data and control or flags.

Data couples can be accessed both by the user and the system (a module or a set of modules), whereas control couples or flags are inherent to the system and hidden to the user [2].

4.6 FLOWCHARTS

4.6.1 Definition

A flowchart is an effective graphical representation of program logic applicable to both hardware and software design [38], [45].

4.6.2 Notation

Using a set of geometric standard symbols and usage conventions, flowlines that show the sequence and direction of information flow, it is possible to describe specific operations and procedures in a concise manner [38], [45].

The notation of the symbols and their meaning is illustrated in figure 4.5.

4.6.3 Structure

The symbol configurations for three basic programming operations that are language independent form the basis for all structural programming; and these are: sequence, decision, and repetition. A program logic can be expressed as combinations of these basic patterns [38], [45].

a-The sequence

The sequence pattern implies that the logic is executed in simple sequence, one block after another, and a block may consist of one or more instructions [38]. This pattern is depicted in figure 4.6.1.



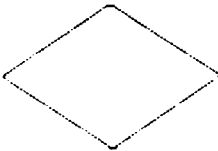


Symbol	Meaning	Explanation
	Terminal point	Marks the beginning or the end of a program or program segment.
	Process	Indicates any arithmetic or data copy operation.
	Decision	Indicates a Yes/No decision to be made by the program.
	Input/output	Indicates any input or output operation.
	Subroutine [45]	Indicates any sub program.

Figure 4-5: Basic Flowcharting Symbols [38], p 350 and [45].

The decision block implies IF-THEN-ELSE logic. This is illustrated in figure 4.6.2.

A condition (the diamond symbol) is tested. If the condition is true, the logic associated with the THEN branch is executed, and the ELSE block is skipped. If the condition is false, the ELSE logic is executed and the THEN logic is skipped [38].

c-Repetition

In the DO WHILE pattern, shown in figure 4.6.3, a test is performed at the top of the loop. If the condition tested is true, the logic of the loop is executed, and control is returned to the top of another test; if the condition is false, the logic of the loop is skipped, and control is transferred to the block following the DO WHILE [38].

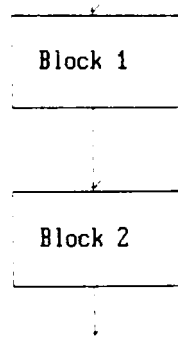


Figure 4.6.1: Sequence [38]

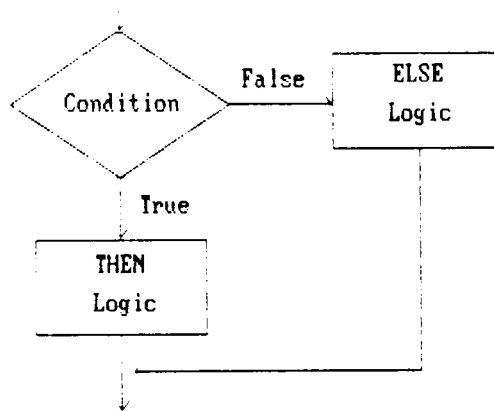


Figure 4.6.2: Decision or IF-THEN-ELSE logic [38].

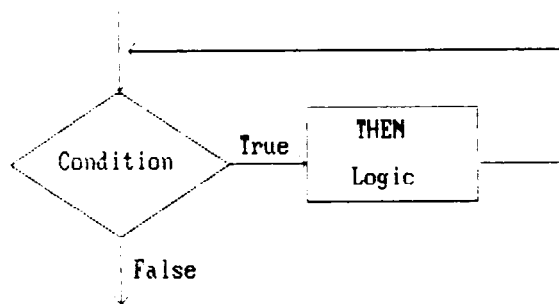


Figure 4.6.3: DO WHILE Logic [38].

THE COCKTAIL APPROACH : SEDA

5.1 INTRODUCTION

Software engineering is intended to assist the development of good quality software within budgets and timescales [1].

A major objective of software engineering is the search for an adequate development approach to guide the process of developing software systems in order to overcome the problems encountered by both the users and developers of the systems. The main purpose of a methodology should be to ensure that a system which meets the end user needs is produced.

Until now, many different life cycle models, approaches and many combinations are used, but still neither of them is ideal, each one or combination has its advantages and disadvantages. It has been shown that it is possible to implement different methodologies for different phases of the life cycle. O'Dell (1986), for example, considered how a structured methodology can be combined with prototyping to aid software production [1]. Nowadays, it is almost evident to build throwaway prototypes during the different development phases of the life cycle, namely

during the analysis phase and the design phase. But still the adequate approach to software development is not yet obtained.

In the present work, we suggest a cocktail approach which consists of the structured approach and the evolutionary prototyping approach. This combination, actually can be named Structured Evolutionary Development Approach, or simply SEDA. The proposed approach will be tested by applying it to the development of a medical Computer Aided Learning, CAL, application, namely computer aided medical training system, or simply CAMTS. The application to be developed will be used in a hospital by medical personnel, students and physicians, in order to learn and keep track of their knowledge.

This chapter deals with the derivation of the theoretical cocktail approach from its components; that's, we introduce the component approaches and show how the cocktail approach is derived, then we will discuss and apply its phases and advantages experimentally to a medical Computer Aided Learning, CAL, system development in the next chapter.

5.2 THE STRUCTURED APPROACH

A structured method refers to structured analysis, structured design and structured coding. These methods are used in conjunction with all the life cycles [2]. The five main phases or steps in the structured system life cycle are: analysis [2], [3], [15], [24], [30], [37], [38], [39], [40], [42], design [2], [24], [38], [39], [42], [43], [44], [47] implementation (coding), testing [3], [5], [7], [11], [21], [45], [47], [50], [60], and

maintenance [3], [20], [52], [53].

This approach is depicted in figure 5.1. The details of its phases are already discussed in chapter three, and will be seen in chapter six which deals with the experimentation or application of the cocktail approach under study in order to develop a methodology that can be used to validate a model or deduced from this experiment.

The structured approach has the following characteristics: top-down strategy, iteration, modularity, flexibility, low development cost, reduces complexity, maintainability, and reliability [2], [15], [24], [30], [38], [47]. This does not mean that the structured approach is ideal, it has its disadvantages mainly when dealing with complex systems, and these are listed below:

- 1-Requirements are often poorly understood.
 - 2-Requirements usually change during the development process.
 - 3-Current requirements remain only partially understood until the user (s) will have an opportunity to use a system [14].
 - 4-The users are not fully involved during the development process.
 - 5-Not enough importance is given to the psychological aspect.
- The third point appears to condemn this approach, but prototyping is compatible with all the development phases of any life cycle model [1], [15].

Following are the experimental advantages and disadvantages of the structured approach compared to the prototyping approach [27]:

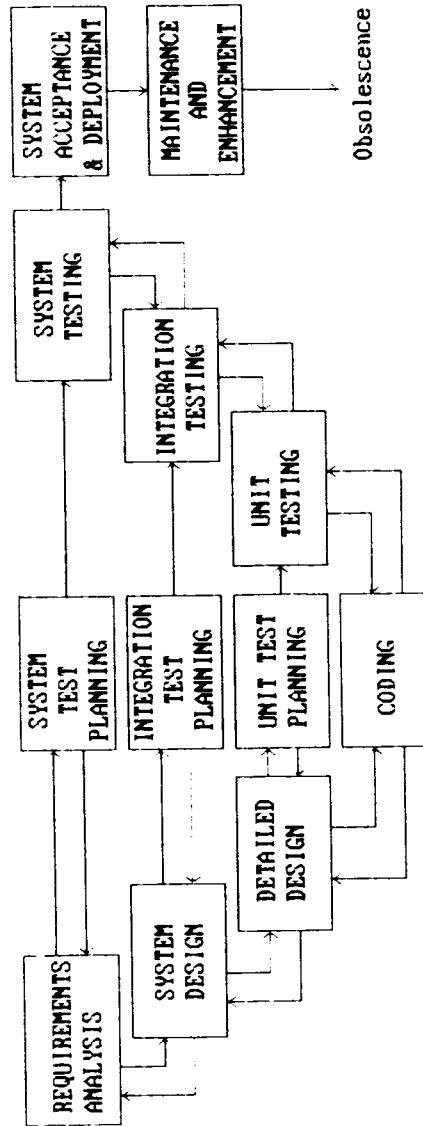


Figure 5.1: V-diagram life cycle model [15], sec 1.1.2, p 10

Advantages:

1-The structured approach enforces design documentation, and hence eases the tasks of future development and maintenance of the system.

2-The structured approach gives a wider and deeper understanding of the system.

3-It is more amenable to effective and "robust" planning. The stages of development are more visible.

4-It allows for the selection of the most appropriate software.

Disadvantages:

1-This approach is less robust to major changes.

2-The more vague the user requirements, the more difficult this approach is.

3-Implementation problems are not visible in advance, which, if we are not careful, can cause timescale overrun.

5.3 THE EVOLUTIONARY PROTOTYPING APPROACH

The evolutionary approach combines the advantages of exploratory programming, with the control required for development. It involves developing the requirements and delivering the system incrementally [5]. Prototyping offers several attracting advantages such as flexibility to adapt the software system to changing environmental characteristics or perceptions of user's needs [28].

Prototyping is used for exploring ideas, assessing potential markets, estimating costs, and for establishing feasibility and

performance limits [5], [6], [14], [27]. There are two types of prototyping, the throwaway and the evolutionary, and both are compatible with the development phases of a life cycle. The role of prototyping is very important in the communication between the user and the developer; mainly for the requirements validation and completeness. That 's, its role is to

- 1-determine the feasibility of a requirement.
- 2-validate that a particular function is really necessary.
- 3-uncover missing requirements.
- 4-determine the viability of a user interface.

In other words, this ensures that the right product is being specified and built [15].

During any development phase, the throwaway prototype should be quick and dirty. The most common way of developing a throwaway prototype calls for:

- 1-writing a preliminary system requirements specification (SRS),
- 2-implementing the prototype based on those requirements,
- 3-achieving user experience with the prototype,
- 4-writing the real SRS, and then
- 5-developing the real product [15].

As it may be deduced, the user is at a certain extent involved in the development process, and can be involved completely when using the evolutionary prototyping approach. The evolutionary prototype is different from the throwaway prototype; particularly, it is not built in a dirty fashion since it converges to the final product. Thus, it must exhibit the quality attributes through its evolution, and the development will not

be particularly rapid as compared with the throwaway prototype. The concept of the evolutionary prototyping approach is the building of an evolvable prototype to learn more about the problem or its solution and then expanded incrementally to become the final system.

Following are the experimental advantages and disadvantages of prototyping with respect to the structured approach [27]:

Advantages:

1-The prototyping approach is shown to be more robust to sudden and major changes and it is a demonstrable approach.

2-The prototyping approach provides a superior environment for knowledge elicitation through the mechanism of allowing the user to criticise working models of the system.

3-The prototyping approach allows for greater flexibility in project planning.

4-Testing in this approach is spread out throughout the project.

Disadvantages:

1-The prototyping approach tends to narrower and superficial knowledge domain.

2-Prototyping reinforces the human drive towards fast solutions on the screen and uses less documentation.

3-Prototyping allows for too much flexibility, which makes it difficult to control.

5.4 THE COCKTAIL APPROACH

As already mentioned early, the main purpose of software engineering which is still being developed, is to create an adequate approach to assist the development phases in order to get the right product and the product right.

In the present work, our objective is to improve software systems development; hence we suggest a cocktail approach which can be named Structured Evolutionary Development Approach, or simply SEDA, which combines the advantages of the structured approach and the advantages of the evolutionary prototyping approach, that are complementary. It is obtained by combining the structured approach with the evolutionary prototyping approach. Because of the complementarity of these two approaches, when combined, their advantages are additive, and this fact annihilates their disadvantages. The concept of this approach is the building of an evolutionary prototype, following the structured life cycle, which will evolve iteratively and gradually to the final system. Any life cycle model consists of a set of interrelated phases; these phases at their turn consist of tasks, and the tasks are built with the help of appropriate modelling techniques before their physical implementation. In the case study we are dealing with, namely the computer aided medical training system, we use the modelling techniques already discussed in chapter four; typically, data flow diagrams and data dictionaries in the analysis phase; structure charts, flowcharts, pseudocode, and data dictionaries in the design phase. These techniques are used with the help of software tools such as Computer-Aided Software

Engineering (CASE) [61], [62], [63].

The cocktail approach model is depicted in figure 5.2 below:

From this figure, we can notice the concept of this approach, that's, it is based on the building of a structured analysis and design prototype which evolves iteratively and incrementally or gradually to the end-product.

The idea behind this concept is illustrated in figure 5.3. In other words, at each iteration S_i , a defined prototype P_i , which is the next version of the system, is developed then deployed, after obtaining experience using it with the user (s), and according to that experience, we go back and redo the analysis, redesign, recode, retest and redeploy. After gaining more experience, the entire process is repeated again until the final version or the end-product is obtained. This ensures the creation of all the necessary documents and the incremental and iterative development process of the product involves a large participation of the user (s); typically, this ensures that we are solving the right problem and the product under development is right. In other words, the appropriate software metrics [19] must be set early and kept in mind throughout the development process. Theoretically, the advantages of this cocktail approach are the combination of the advantages of its components and they are complement to each other; that's, they are:

- 1-Enforcement of formal documentation.
- 2-Deeper understanding of the system.
- 3-Ensures effective and robust planning and the stages of development are more visible.
- 4-The development tool is chosen early in the project after

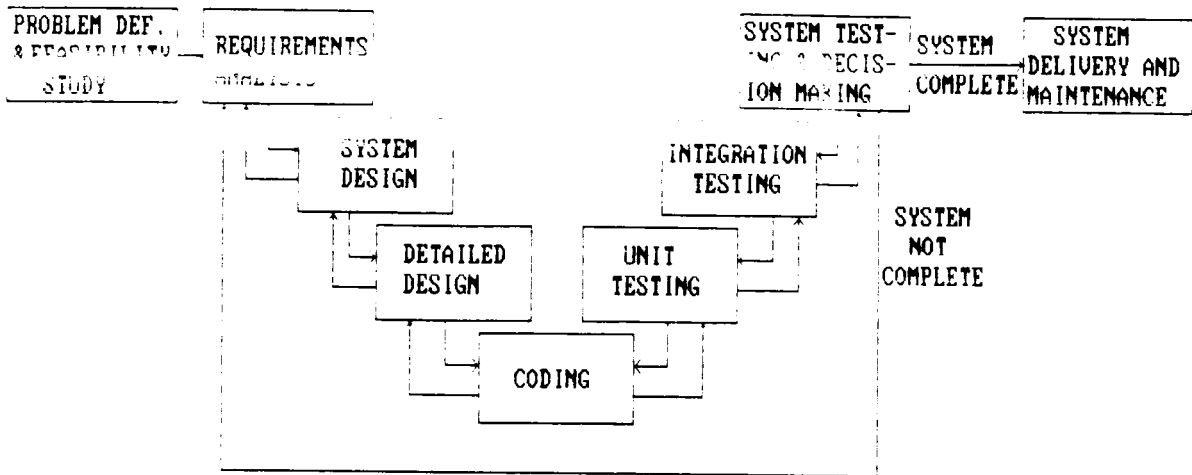


Figure 5.2: Cocktail approach (SEDA)

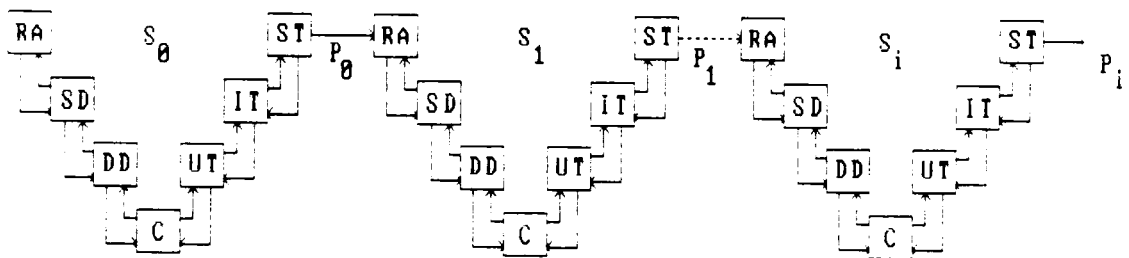


Figure 3: The software life cycle and Evolutionary prototypes [13]

understanding the domain knowledge.

5-Robustness to sudden and major changes and it is a demonstrable approach.

6-Provides a superior environment for knowledge elicitation through the mechanism of the demonstration of the working versions of the system.

7-Allows for greater and controllable flexibility in project planning.

8-Testing is spread out throughout the project development process.

9-Ensures the progress in the development of the product.

10-Reduces the cost/benefit ratio.

11-The user is involved throughout the development process.

5.5 CONCLUSION

The major causes of problems in software production is the fact that the initial design specifications are often incomplete and/or inconsistent. Hence, the iterative evolutionary cocktail approach we suggested and discussed allows, theoretically, the control of specifications for consistency and completeness.

After discussing the component approaches, we have seen the cocktail approach and its advantages; at this point the following question rises: What are the disadvantages of the approach under study ? The answer is that they cannot be deduced theoretically from the component approaches; the reason is that because of the complementarity of the component approaches, the disadvantages of each component approach are drawn in the advantages of the

other component approach. Hence, we hope that the number of disadvantages is as small as possible, and we let experiments answer the question.

Actually, we have seen the theoretical aspect of the suggested approach, the next chapter deals with the experimental aspect; typically, its application to the development of a medical CAL application, namely, computer aided medical training system, or simply CAMTS, that will be used in a hospital by surgery medical personnel, students and physicians, in order to learn, review, and self-evaluate their knowledge.

**THE APPLICATION:
COMPUTER AIDED MEDICAL TRAINING
SYSTEM DEVELOPMENT BASED ON THE
COCKTAIL APPROACH (SEDA)**

6.1 INTRODUCTION

The use of computers for teaching is a difficult application. The aim in teaching is to transmit knowledge to someone who will not be an expert in the application but will be just the opposite; that's, a student. The problem of encapsulating course material and successful teaching principles and of understanding students errors and lack of comprehension are redoutable. Some form of information retrieval is required [16].

Teaching can be analyzed into a process consisting of motive, perception, action, and consequence. First people want something, then they notice something, do something, and finally get something [64]. But, can teachers predict or estimate the responses of their pupils? The accuracy depends on the teacher's effectiveness in teacher-pupil relationships; that's, teacher's understanding of students is correlated with their success in

getting along with pupils. In order to get a better idea of how teachers should understand what aspects of their pupils, and how such understanding can be improved, we need to study cognitive aspects of both teachers and pupils [64], [65].

Through the medium of graphics terminals, the data can be displayed in ways which are informative and visually attractive. However, there are psychological factors which influence learning and CAL, furthermore, it is not easy to integrate the conventional teaching schemes in a computer [16]. In tutorial programs which teach the material through question-answer sequences, the student is active in his learning. The instruction is driven by the responses which the learner gives at the terminal, then the computer program can evaluate these responses, provide feedback, and makes appropriate decisions, which suit to his competence.

Computer Assisted Learning (CAL) systems could be viewed as an adjunct to the conventional teaching methods because they cannot supplant the teacher [16]. Typically, the modes of communication from the student to the computer are limited, therefore, the teaching steps must be relatively small, the approach directed, and the types of dialogue limited.

We recall that this application is intended to be used in a hospital by the surgery medical personnel in order to attend courses and perform tests typically, self-learning or -training and self-testing. The concept is the use of task-structure analysis and design [66] of a rule-based system [67] that is tailored to optimize a single rule-based program in which rules are stored as texts in different modules or files accordingly.

This chapter deals with the application of the cocktail approach, SEDA, already introduced previously, to the development of a computer aided medical training system. Hence, learning and CAL are first introduced, then the next sections deal with the application development process.

6.2 COMPUTER ASSISTED LEARNING (CAL)

The lack of individualized instruction is considered to be a major cause for the decline in the quality of education. In other words self study offers a rich potential for development and Computer Aided Instruction or Learning (CAI or CAL) offers some solutions to many problems of education [23], [68], [69]. However, the classical training consists of reading journals and books, attendance at formal courses and conferences and discussions with colleagues. This is useful because it keeps the professions aware of the state of the art, but these means are not always offered and the self study is neglected [16]. Moreover, students can use Computer Aided Instruction, Learning, or Training (CAI, CAL, or CAT) to prepare for examinations, since self-instructional methods allow flexibility [70].

6.2.1 Integrating Conventional Teaching and CAL Schemes

Integrating the conventional teaching scheme in a computer is not an easy task. Thus, the respective strengths of the teacher and machine-based programs can be considered complementary. The teacher has good general knowledge of the topic and easy modes

of communication with the student; and the computer assisted learning, CAL, can be a useful support when the teacher has neither the time nor the resources to give adequate group teaching [16].

6.2.2 Factors Influencing Learning and CAL

There are factors which influence learning and CAL. A considerable number of psychological and educational studies which are of interest, and since they are related to the design of CAL, emphasize the following points:

- 1-The need to provide the learner with informational feedback and the precise conditions under which it is most effective.

- 2-The influence of the relational structures within the teaching material, and its sequencing.

- 3-The importance of encouraging active learner control.

However, there is no agreed overall learning scheme, but Nuthall and Snook (1973) gave a teaching classification scheme in which they identify the following types of teaching:

- 1-Behavioral control models: These models stress complete control over student behaviour over the conditions of learning. Within CAL, the emphasis is on programs which are directive in style.

- 2-Discovery learning models: These models emphasise on the control which the learner has in building his knowledge structures. The teacher is not the primary source of information, but acts to simulate and monitor the learner and reveal the inadequacies of generalisations by producing counter examples.

It is maintained that such methods allow the student not only to arrive at more general conclusions, but to learn about the process of generalisation itself.

3-Rational models: These models stress the place of reasoning and dialogue in teaching.

Two approaches have been used to extend initiative and dialogue:

1-Use of simulation modules within author language tutorial programs.

2-Use of simulation programs which include dialogue facilities and programming languages through which students can express and demonstrate their solutions by designing, debugging, and running their own computer programs [16].

Actually, too much work on CAL systems, learning schemes, learning strategies and their integration is done, [64], [65], [68], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], and many successful results in improving the performance of learning using the computer are obtained.

6.2.3 CAL and the Computer Aided Medical Training System

Medicine is a field which requires extensive education to attain and maintain an acceptable level of functional competence [68]. And medical education is intended to help physicians learn new developments and review fundamental concepts in medicine on the assumption that knowledge gained will automatically improve patient care. However, the classical training consists of reading journals and books, attendance at formal courses and conferences and discussions with colleagues. This is useful because it keeps

the professions aware of the state of the art, but these means are not always offered and the self study is neglected [68], [76], [80]. The lack of individualized instruction is considered to be a major cause for the decline in the quality of education. In other words self study offers a rich potential for development and Computer Aided Instruction or Learning (CAI or CAL) offers some solutions to many problems of medical education [23], [68], [76]. The computer increases the availability of data and allows a flexible data organization in its memory [82]. The application of computer technology to the teaching of medicine is attempted with varying success over the years, but newer technology and development tools suggest that Computer Assisted Learning, CAL, must be introduced in Medical schools and hospitals [68], [78], [79], [80], [81]. Computer technology supplies knowledge and guidance on specific problems at the time the physician is studying a patient, hence diminishing reliance on memory [76]. And is now an integral part of patient care in many health care systems. It helps in improving the effectiveness of health and medical personnel in planning decision making and problem solving [77].

Moreover, medical students can use Computer Aided Instruction, Learning, or Training (CAI, CAL, or CAT) to prepare for examinations, since self-instructional methods allow flexibility [16], [70].

According to the previous sections, we can conclude that the model of teaching to be incorporated in a CAL system depends on the application domain which dictates its corresponding cognitive aspects and nature. The application we are dealing with is a

computer aided medical training system. Taking into consideration the non deterministic nature of the medical field and the user needs, the behavioral model suits as the teaching scheme to be built within the present application. In this scheme, the teacher is the primary source of information, and he is a manager who seeks to accomplish specific objectives as quickly and as efficiently as possible. Thus, he controls the selection and the arrangement of content and task so that the responses are provoked. He controls also the feedback and other reinforcing stimuli which are used to maintain and regulate effort, and to shape more complex learning behaviours by building up response chains composed of small steps. In other words, the system is built in such a way to expose the students misunderstandings and provide feedback to enable him to correct them. A cheat condition is avoided, that is, presentation by computer ensures that feedback is not available until the student responds. This shows the informational role of feedback, and the benefits of control over learning activity. A third point is that of sequencing learning material, particularly in subject areas which can be arranged in a hierarchical fashion, such as the medical application we are dealing with. Typically, decision rules are constructed which only allow the learner to proceed to further sections of the material in the hierarchy. Gagne and Paradise (1961) and Gagne and al. (1962) reported studies which showed the importance of such decision rules. All these research work approve for decision rules based on specific information about the individual learner's knowledge and performance. For this reason, adaptive CAL programs should in principle be efficient

aids to learning [16].

In behavioural control model situations, management of learning is an important aspect of CAL programs [16]. Thus, in the medical CAL application development which will be seen in the next sections, the courses and tests are organized in a way to maximize the benefits of the behavioral teaching model. The user-machine communication has a high level of abstraction; this is one of the characteristics of the programming language that's used, namely DBASE IV [49], [58], [59]; in other words, the users will not be bothered by a command language training before using the system. And in the case of medical applications, it is identified that the user interface and the reliability quality attributes are the most important for the software systems to be accepted by the medical personnel [23]. Hence, the development is carried out keeping in mind mainly these metrics.

6.3 THE COMPUTER AIDED MEDICAL TRAINING SYSTEM (CAMTS) DEVELOPMENT PROCESS

The intent of this system is for training surgery medical personnel. Its use will be at the level of a hospital by the surgery personnel.

The approach used to build this system as already mentioned and discussed early, is the cocktail approach that is formed by the combination of the evolutionary prototyping approach with the structured approach. This concept, we named Structured Evolutionary Development Approach, or simply SEDA, is depicted by figure 6.1.

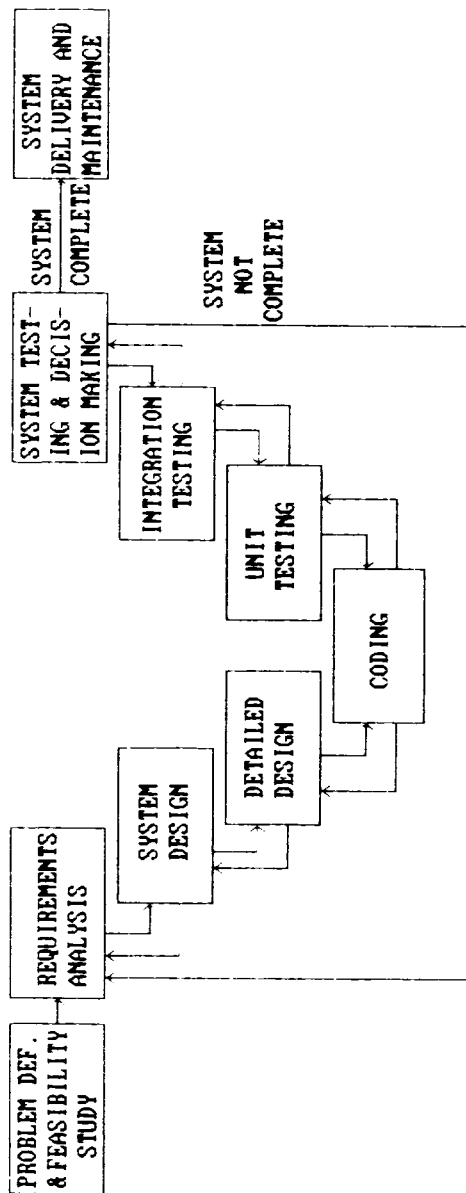


Figure 6.1: Cocktail approach (SEDA)

cocktail approach in order to draw conclusions with respect to the theoretical study that is done in the previous chapter. As already mentioned in the previous section, for medical applications, it is identified that the user interface and the reliability quality attributes are the most important for the software systems to be accepted by the medical personnel [23]. Hence, the development is carried out keeping in mind mainly the following metrics: ease of use, reliability, and maintainability. The nature of the application and the development approach, namely functional approach using the Yourdon methodology, dictate the appropriate notations and modelling techniques to be used; typically, data flow diagrams, data dictionary, pseudocode, structure charts, and flowcharts, which are already introduced in chapter four. The application is developed in the DBASE IV programming language environment. This later is easy to use and has a high level of abstraction, moreover it suits to the entreprise or hospital environment.

The development is based on the identification of several prototypes of the system. Each prototype is the enhanced version of the previous one; this allows a rapid development of the end product. In other words, the cocktail approach that is used allows the visibility of the attributes required by the users; and the rapid development allows the building of a prototype that will go through a process of successive iterations. Moreover, it allows knowledge elicitation through the mechanism of allowing the users to evaluate and criticise each working version of the system.

All the phases which form the approach under study are described by applying them to the application development, referring to figure 6.1.

6.3.1 Problem Definition and Feasibility Study Phase

During this phase, first the problem is identified and defined, then, the feasibility study of the software system under study is carried out.

6.3.1.1 Problem Definition

We begin with a brief overview of the organization, (hospital) where the system will be used, from the point of view medical personnel then introduce the problem. The output of this stage is a statement objectives.

In Thenia hospital (Algeria), there is a lack of experts in medical fields. For example in surgery one professor is not enough for explaining to his colleagues a subject; and a professor might not be available for different reasons. Therefore, there is a need for building a software system that can help those physicians, at any time, to learn and to self-evaluate their knowledge. The computer increases the availability of data and allows a flexible data organization in its memory [82]. In other words, the motivations behind this application are:

- Not many experts in the hospital.
- Very hard to get in touch with very few experts that exist in Algeria.
- A large number of medical students.

This leads us to the following statement objectives: the computer-aided medical learning or training system is needed in the surgery service of the hospital and will be used by the medical personnel, students and physicians, in order to attend courses and self-evaluate their knowledge individually or collectively under the control of an expert.

6.3.1.2 Feasibility Study

Now that the objectives are clarified, an investigation concerning the feasibility of this project is conducted. The purpose is to find the answers to the following questions which consist this stage:

- 1-Can the system be implemented using current technology ?
- 2-Can the system be implemented in the hospital ?
- 3-Do benefits outweigh costs ?

As may be noticed, each question deals with one aspect, and these are technical, operational, and economic respectively.

In order to find the answers, we have first to see the extent of the project. The only way to do this is by refining the problem definition, that's, getting more details about the project, and the only source for this purpose is the user. let's see what the user wants !

During this stage all possible user's needs are gathered through interviews that are psychologically directed; and for the user-developer communication means, data flow diagrams are used. Actually, the user's needs are depicted in figure 6.2.

An important point about this figure is that, it is a logical data flow diagram; it is easy to make changes and identify other

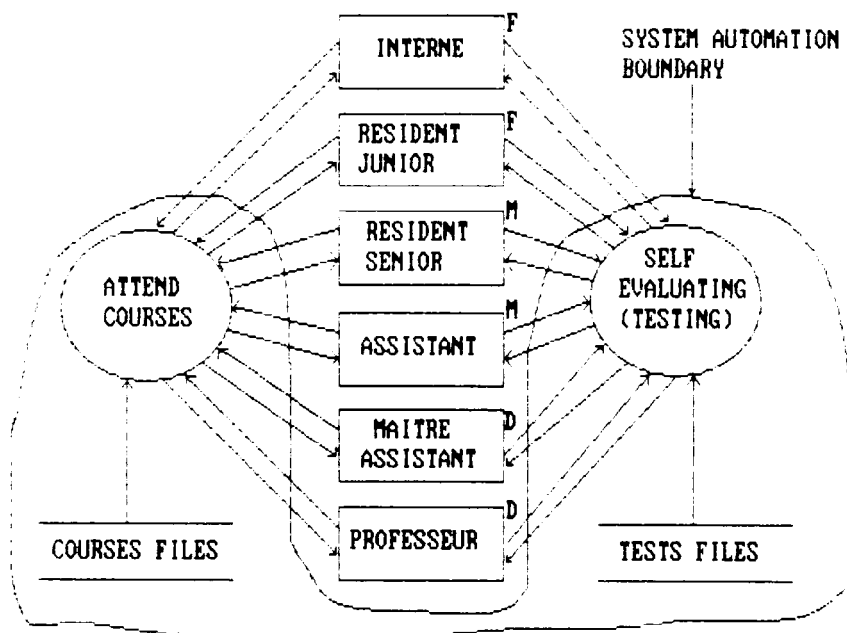


Figure 6.2: System automation boundary

alternatives of the automation area. The system automation boundary shows all the components of the software system to be built. To be confident in answering any question, after few discussions with the users, a rapid throwaway prototype was built using a computer software tool called Automate-Plus which is used to assist software developers in the analysis and design phases [62], [63]. This prototype allowed us to get the main frame of the whole system; that's, the preliminary requirements of the system under study. And helped a lot in getting precisions and more requirements.

The answers to the above questions are all affirmative; that's, the system can be built using the current technology, the system can be implemented in the hospital, and benefits outweigh costs. In fact, this is my first experiment in such software product development; hence, thanks are addressed to my advisor B. Mezhoud whom guided me and approved this feasibility study which requires experience.

6.3.2 Requirements Analysis Phase

At this point, we know that the problem can be solved, the intent of this phase is to determine at a logical level WHAT the proposed system must do. The input to this phase is the documentation developed during the feasibility stage, that is the data flow diagram. The objective of analysis is to answer the question: WHAT must the system do ? This question is partially answered during the feasibility study, but it is not enough. Hence, during this phase we develop a complete functional

understanding of the proposed system. The intent is not to determine how the system will work, this is the job of the design phase, but what it must do. In other words, during this phase, the necessary detailed specification documents namely, data flow diagrams, and data dictionaries are developed. The DFDs are used to communicate with the users and then refined after gathering more information. Additional penetration or insight is gained by exploding the DFD to lower levels; step by step, the logical detailed system is more fully defined. In other words, the logical levels of the system are identified then their corresponding specification documents are delivered incrementally and iteratively to the next phases of the development process. A specification increment may be one logical level of the system, hence, it will correspond to one prototype or one version of the system. And since the structured approach involves top down strategy, the development will start from the high level, namely the context diagram, and iteratively refined and enhanced as the system being developed. The idea behind this strategy is to build first the system skeleton then fill it with the flesh.

We begin with the DFD document of the feasibility study, this document identifies a number of functions or processes that must be performed by the computer assisted learning system. A new and well structured logical system is being developed. Since the cocktail approach, formed by the structured approach and the evolutionary prototyping approach is used, its advantages are under consideration. Typically, the structured approach dictates the top-down development [47] and iteration, while the prototyping approach allows us to develop a prototype.

The exit criteria of this phase include a complete DFD and DD documents.

Following are the data flow diagrams of the system under development.

6.3.2.1 Data Flow Diagrams

The proposed automated system is pictured in the logical DFDs of the following figures. The processes show the activities that will be processed or executed by the system.

6.3.2.1.1 Context Diagram

Called also level 0, represents the very high level view of the system. It consists of only one process. The context diagram of this system is shown in figure 6.3.

This diagram shows the overall logical system as the circle labeled CAMTS. The terminator USER is not a part of the system but the system user. Transactions consist of information fed into the system or extracted from it; its details are given in the data dictionary.

6.3.2.1.2 Level one DFD

This level is obtained from the explosion of the contex diagram. The data flow diagram shown in figure 6.4 represents the level one for the logical system. It depicts two main processes labeled GET DATA and OPEN MODULE.

The individual processes refer to the system functions, that's, the activities it will perform. Actually, the activities depicted

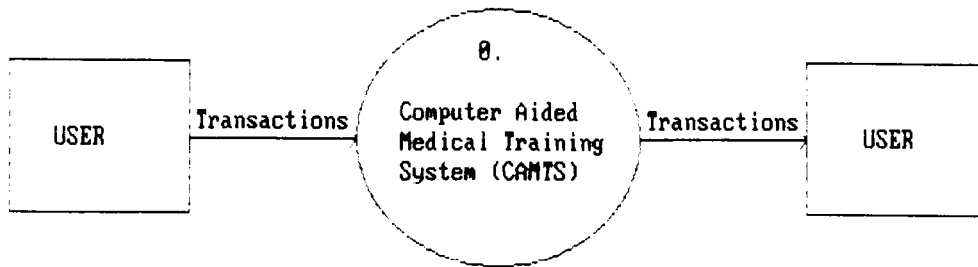


Figure 6.3: Level 0., Context diagram

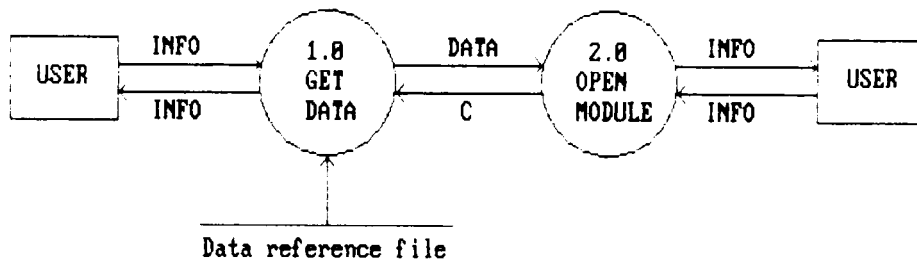


Figure 6.4: Level 1 DFD for the logical system
Explosion of the context diagram.

by the individual processes are not basic ones, but consist of a set of other activities that will be shown in the next section which deals with the explosion of the level one DFD processes to obtain level two DFD processes.

6.3.2.1.3 Level Two DFD

The two processes shown in the level 1 DFD of figure 6.4 do not refer to basic or elementary tasks as said; so the elementary processes, level two in this application, are obtained by exploding each individual process of the level 1 DFD. The goal is to get another level of detail of the system to understand its overall operation. The level 2 DFD of the logical system is depicted in figure 6.5.1 and figure 6.5.2.

Having obtained these pictures, we can consider them as being a representation of the details of the system. Namely, each process refers to an elementary system function.

The description of these different activities are dealt with in next section which is the data dictionary.

6.3.2.2 Data Dictionary

All the labels involved in the logical DFDs are defined in the following dictionary:

CAMTS = * Abbreviation of the system's name : *

* Computer Aided Medical Training System *

User = [Intérne/Résident Junior/Résident Senior/Assistant/Docent/
Professeur]

Transactions = [Input Info/Output Info]

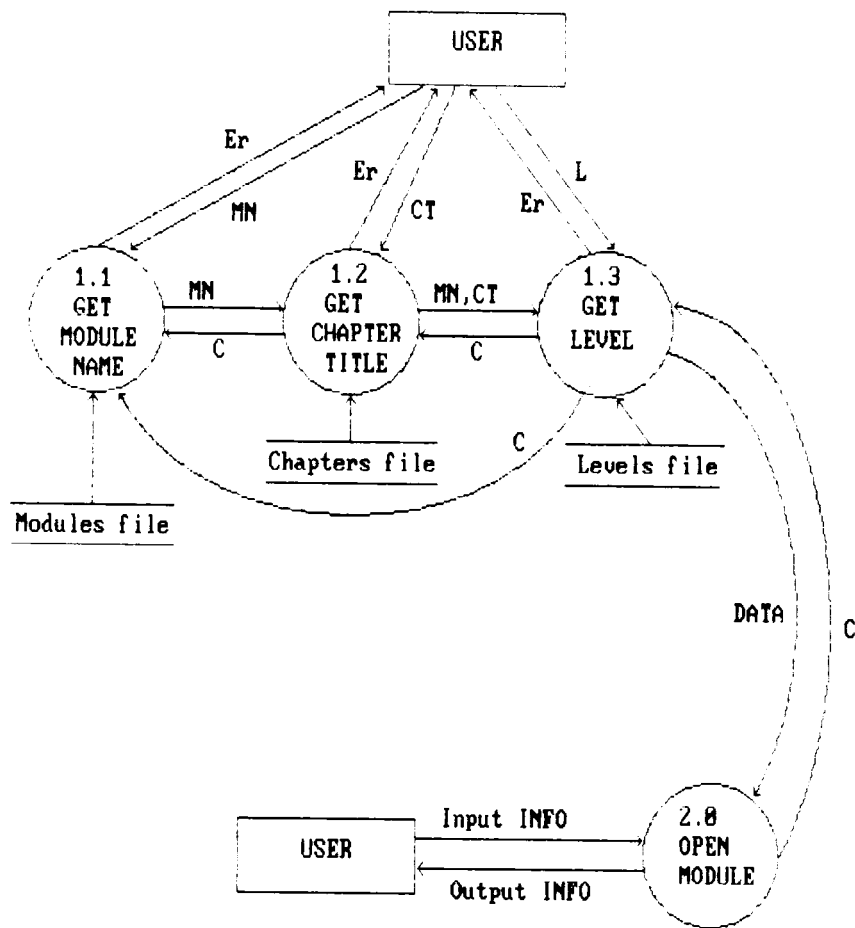


Figure 6.5.1: Level 2 DFD, Explosion of Get Data Process

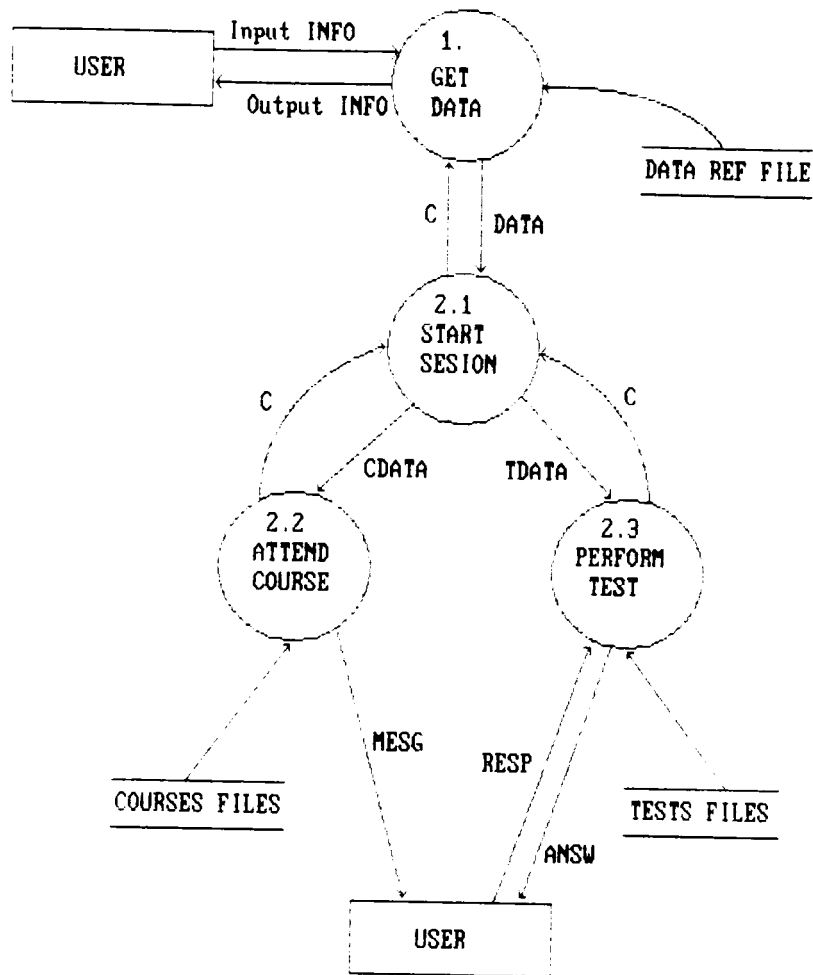


Figure 6.5.2: Level 2 DFD, Explosion of Open Module Process

Info = * Abreviation for information *
 Input Info = Module Name + Chapter Title + Level + Responses
 Output Info = [Courses/Tests] + Messages + Answers
 Get Data = * Input Data Process *
 Data = Module Name + Chapter Title + Level
 Module Name = [Courses Module/Tests Module]
 Chapter Title = [Chirurgie Générale/Chirurgie Digestive/Urologie/
 Orthopédie]
 Level = [Facile/Moyen/Difficile]
 Facile = Interne + Résident Junior
 Moyen = Résident Senior + Assistant
 Difficile = Docent + Professeur
 Data Reference File = * Reference Data Lists *
 Get Module Name = * Process which handles the list of the
 modules *
 Modules File = * The file containing the reference list of
 modules *
 MN = * Abreviation for Module Name *
 Er = * Abreviation for Error Message *
 Chapter File = * The file containing the reference list of
 chapters *
 CT = * Abreviation for Chapter Title *
 Levels File = * The file which contains the reference list of
 levels *
 L = * Abreviation for Level *
 Open Module = * Dispatcher module which routes the transaction
 to the appropriate or selected destination
 according to the input data *

Start Session = * The process which opens the selected destination, that's, to attend a course or perform a test *

Attend Course = * The selected course is displayed on the screen of the computer *

CDATA = * Courses DATA *

TDATA = * Tests DATA *

MESG = * Abbreviation for Message(s) *

RESP = * Abbreviation for Responses *

ANSW = * Abbreviation for Answer *

Perform Test = * The chosen test is displayed on the screen of the computer *

Courses Files = * Data store containing the courses *

Course = Summary + Bibliographical references

Tests Files = * Bank containing the tests *

Test = Q.C.M. test format + Comments

6.3.3 Design Phases

Continuing the development, this phase describes the system design. The objective of this phase is to determine HOW the system will be implemented. Using the data flow diagrams obtained during the analysis phase as input, the main objective of system design is to develop a blueprint for the physical system. The output of the analysis phase is a functional logical model for the proposed system. The design phase consists of translating this functional model into a hierarchy of modules that will

determine the overall operation of the system. And these modules will be followed by their corresponding algorithms. This is done through two phases:

6.3.3.1 System Design Phase

This phase is also referred to as high level design, preliminary design, and logical design. The objective is to plan a logical model within the context of a complete system. The data flow diagram document is transformed into a structure charts document. Using the transform-centered hierarchy structure [30], the structure chart is derived directly from the data flow diagram of the system. That's, the transform-centered structure chart system is derived from the data flow diagrams in which all the transactions follow the same path though all the identified levels, and the corresponding hierarchy of modules is obtained. Figure 6.6.1 and figure 6.6.2 below, show the modules that are identified and their relationships.

As may be seen, this stage answers the question : HOW the system will be built ?

The details of the modules will be seen in the next section which is the detailed design phase.

6.3.3.2 Detailed Design Phase

This phase, also referred to as physical design, produces a blueprint of the system; that's, it determines HOW, specifically, the system should be implemented. This phase uses as input the results of the previous phase, for which it applies constraints, details of language and hardware, then produces the blueprint,

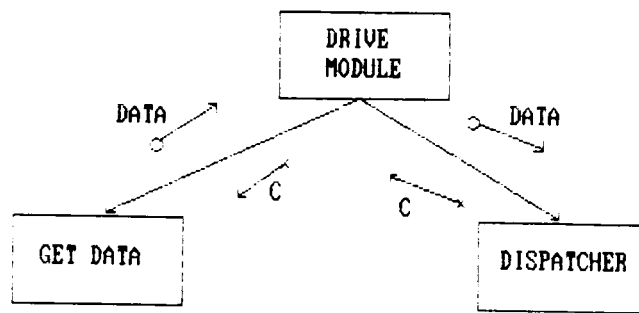


Figure 6.6.1: System design structure chart

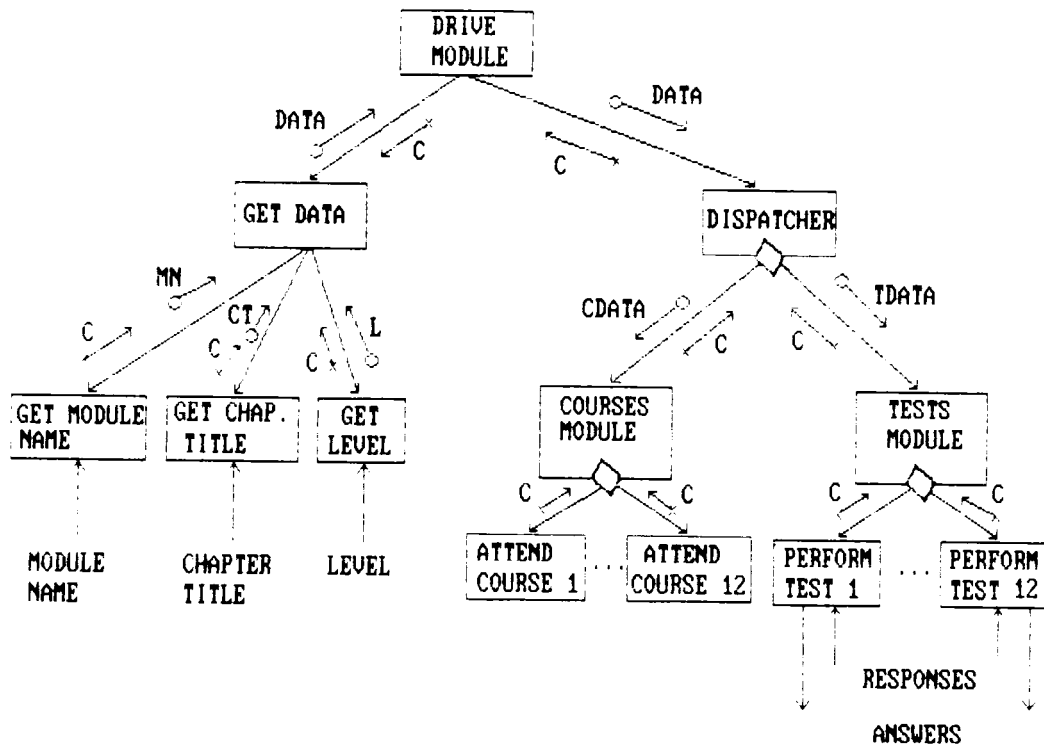


Figure 6.6.2: Detailed design structure chart

or solution model. That's, the output is a set of algorithms developed using the flowcharting technique. The pseudocode of the application program is given in the next section and the flowcharts are depicted in appendix b.

6.3.3.3 System Pseudocode

Pseudocode is a tool that is used to describe the logic of a process. It is an alternative to structured English which, when used, ignores details such as opening and closing files, initializing counters, and setting flags. In other words, pseudocode attempts to correct the shortcomings of the structured English.

For the present application, we use a pseudocode borrowed from the DBASE IV programming language that is used to implement the software under development.

```
Begin drive_module (Data;Course,Test)
    Do get_data_module (C;Data)
    Do dispatcher_module (Data;C)
End drive_module
```

```
Begin get_data_module (C;Data)
    Do get_module_name (MN;C)
    Do get_chapter_title (CT;C)
    Do get_level (L;C)
End get_data
```

Begin get_module_name (MN;C)

Input MN

If MN correct

Continue (exit)

Else

Loop

Endif

End get_module_name

Begin get_chapter_title (CT;C)

Input CT

If CT correct

Continue (exit)

Else

Loop

Endif

End get_chapter_title

Begin get_level (L;C)

Input L

If L correct

Continue (exit)

Else

Loop

Endif

End get_level

```

Begin dispatcher_module (Data;Cdata,Tdata,C)
  If Data=Cdata
    Do courses_module
  Else (Data=Tdata)
    Do tests_module
  Endif
End dispatcher_module

```

```

Begin courses_module (Cdata;C)
  Do case
    case Cdata=CHGNLEFC_code
      Attend course CHGNLEFC
    case Cdata=CHGNLEMC_code
      Attend course CHGNLEMC
    case Cdata=CHGNLEDC_code
      Attend course CHGNLEDC
    case Cdata=CHDIGFC_code
      Attend course CHDIGFC
    case Cdata=CHDIGMC_code
      Attend course CHDIGMC
    case Cdata=CHDIGDC_code
      Attend course CHDIGDC
    case Cdata=UROLOGFC_code
      Attend course UROLOGFC
    case Cdata=UROLOGMC_code
      Attend course UROLOGMC
    case Cdata=UROLOGDC_code
      Attend course UROLOGDC
  
```



```

        case Cdata=ORTHOPFC_code
            Attend course ORTHOPFC
        case Cdata=ORTHOPMC_code
            Attend course ORTHOPMC
        case Cdata=ORTHOPDC_code
            Attend course ORTHOPDC
    Endcase
End courses_module

```

```

Begin tests_module (Tdata;C)

```

```

    Do case
        case Tdata=CHGNLEFT_code
            Perform test CHGNLEFT
        case Tdata=CHGNLEMT_code
            Perform test CHGNLEMT
        case Tdata=CHGNLEDT_code
            Perform test CHGNLEDT
        case Tdata=CHDIGFT_code
            Perform test CHDIGFT
        case Tdata=CHDIGMT_code
            Perform test CHDIGMT
        case Tdata=CHDIGDT_code
            Perform test CHDIGDT
        case Tdata=UROLOGFT_code
            Perform test UROLOGFT
        case Tdata=UROLOGMT_code
            Perform test UROLOGMT
        case Tdata=UROLOGDT_code

```

```

        Perform test UROLOGDT
    case Tdata=ORTHOPFT_code
        Perform test ORTHOPFT
    case Tdata=ORTHOPMT_code
        Perform test ORTHOPMT
    case Tdata=ORTHOPDT_code
        Perform test ORTHOPD
    Endcase
End tests_module

```

The next section is a Data Dictionary of the new labels that are introduced throughout the design phases.

6.3.3.4 Data Dictionary

The names of the modules and their meanings and the data couples are given in this section.

```

Drive Module = * The module which contains the main program *
Get Data Module = * Input Data Module : MN + CT + L *
Get Module Name Module = * Input MN *
Get Chapter Title Module = * Input CT *
Get Level Module = * Input L *
Dispatcher Module = * On the basis of the input DATA, it routes
                        the transaction to the preselected
                        destination *
Courses Module = * This module opens the selected course file *
Attend Course Module = * Elementary module which contains the
                        contents of the course *

```

* The present system contains 12 courses *

Tests Module = * This module opens the selected test file *

Perform Test Module = * Elementary test module which contains the
content of the test *

* The present system contains 12 tests *

CHGNLEFC = * Abbreviation for ' Cours de Chirurgie Générale,
Niveau Facile ' *

CHGNLEMC = * Abbreviation for ' Cours de Chirurgie Générale,
Niveau Moyen ' *

CHGNLEDC = * Abbreviation for ' Cours de Chirurgie Générale,
Niveau Difficile ' *

CHDIGFC = * Abbreviation for ' Cours de Chirurgie Digestive,
Niveau Facile ' *

CHDIGMC = * Abbreviation for ' Cours de Chirurgie Digestive,
Niveau Moyen ' *

CHDIGDC = * Abbreviation for ' Cours de Chirurgie Digestive,
Niveau Difficile ' *

UROLOGFC = * Abbreviation for ' Cours d'Urologie, Niveau
Facile ' *

UROLOGMC = * Abbreviation for ' Cours d'Urologie, Niveau
Moyen ' *

UROLOGDC = * Abbreviation for ' Cours d'Urologie, Niveau
Difficile ' *

ORTHOPFC = * Abbreviation for ' Cours d'Orthopédie, Niveau
Facile ' *

ORTHOPMC = * Abbreviation for ' Cours d'Orthopédie, Niveau
Moyen ' *

ORTHOPDC = * Abbreviation for ' Cours d'Orthopédie, Niveau

Difficile ' *

CHGNLEFT = * Abbreviation for ' Test de Chirurgie Générale, Niveau
Facile ' *

CHGNLEMT = * Abbreviation for ' Test de Chirurgie Générale, Niveau
Moyen ' *

CHGNLEDT = * Abbreviation for ' Test de Chirurgie Générale, Niveau
Difficile ' *

CHDIGFT = * Abbreviation for ' Test de Chirurgie Digestive, Niveau
Facile ' *

CHDIGMT = * Abbreviation for ' Test de Chirurgie Digestive, Niveau
Moyen ' *

CHDIGDT = * Abbreviation for ' Test de Chirurgie Digestive, Niveau
Difficile ' *

UROLOGFT = * Abbreviation for ' Test d'Urologie, Niveau
Facile ' *

UROLOGMT = * Abbreviation for ' Test d'Urologie, Niveau
Moyen ' *

UROLGDT = * Abbreviation for ' Test d'Urologie, Niveau
Difficile ' *

ORTHOPFT = * Abbreviation for ' Test d'Orthopédie, Niveau
Facile ' *

ORTHOPMT = * Abbreviation for ' Test d'Orthopédie, Niveau
Moyen ' *

ORTHOPDT = * Abbreviation for ' Test d'Orthopédie, Niveau
Difficile ' *

6.3.4 Implementation Phase

This is the last phase of the development process of the structured approach. This phase translates, in top-down fashion [62], the algorithms developed in the design phase into a programming language. In other words, implementation means coding or creation of the physical system.

In order to create a physical system we need a software environment or a software tool. It has been shown by [44] that high level languages contribute to the quality and understandability of the software under development. For the application and the approach we are dealing with, the fourth generation language, namely DBASE IV [49], [58], [59], which offers a high level of abstraction, suits both the approach by allowing a rapid development, and the application domain, that's it accepts a lot of data.

DBASE IV is a compiler language that is developed by the ASHTON-TATE company. It is developed with the C language from which it inherits the following characteristics: C code is small, fast, portable, and flexible [59]. The characteristics of the DBASE IV language are:

- High speed with which software can be developed.
- High abstraction language:
 - Relatively close to English.
 - Easy to learn.
 - Easy to use.
- Ideal to rapid prototyping [6].
- Easy creation of user interface and dialogs [6].

- Interactive environment [6].

- Industry standard.

The fourth generation language is perceived as relevant to the development and production of application systems. A high level language contributes in the quality of software [46].

The physical development reflects the physical iterative and incremental implementation of each system version until the completion of the system development.

If we, as humans, are perfect, the development process ends at this point, implementation; unfortunately, this is not the case.

Therefore, we need a testing process to correct the bugs.

In fact in the approach we are using, testing is spread out throughout the project development process, this is already seen theoretically and actually it is proved experimentally. Hence the following phases deal with testing, they are also referred to as validation phases.

6.3.5 Testing Phases

The software testing process is a critical activity of software quality assurance and represents the review of specification, design and coding when an error is detected. For the present application, the bottom-up strategy [47] is applied and the different phases of the testing process that are performed are discussed below.

6.3.5.1 Unit Testing Phase

This phase is also called module testing, black-box testing, and

functional testing.

During this phase each individual module is tested against its specification derived during system design, then errors if any are corrected.

6.3.5.2 Integration Testing Phase

During this phase, the modules belonging to the same branch of the structure chart are interconnected to form subsystems and tested against their corresponding branch specification, then bugs if any are corrected.

6.3.5.3 System Testing and Decision Making Phase

Finally, the version of the system is interconnected or integrated then tested against the user needs. During this phase the software product version is deployed and validated by the user. This means that the alpha testing process is performed for each version of the system. And if the system is not yet complete, we redo the development process for the next software requirements increment that might be a level as suggested. In each demonstration, the user interface and reliability of the system version is evaluated by the users. This iterative evolutionary process is carried out until the valid right product is obtained. When the system is "complete", we proceed with the next phase which is described in the next section.

6.3.6 System Delivery and Maintenance Phase

This is the last phase of the cocktail approach that we named

structured evolutionary development approach, or simply SEDA. Once the system is, for the present "complete" and valid, it is delivered to the user. Knowing that there is no perfect system, maintenance is required and begins when the system enters productive use. Maintenance is mainly required to adapt the system to the changes of the environment with time. The problem is that the maintainer is rarely the author of the code; hence, he lacks an understanding of the program [52].

Maintenance is expensive, a solution to reduce this cost is to design the system with ease of maintenance in mind. One key is functional modularization [38], [47], [52]. This is one characteristic of the structured approach, hence of the cocktail approach, we used to develop the present product.

Many studies about software inspection techniques [50], measuring software quality [51], and software complexity and maintenance costs [47], [53] are carried out by the software community in order to reduce the costs of maintenance and increase the quality of the software products and many succesful results are obtained. Actually, we hope that the present approach will contribute a lot in developing high quality software projects and products, in fact, it is our purpose from the beginning.

6.4 COST/BENEFIT ANALYSIS

Identifying and resolving software problems early in the development process, often in the phase in which they first occur, has been shown to contribute significantly to the reduction of risks and cost in software development. In other

words, the cost of the software maintenance depends on the phase where the error is detected. The error is detected sooner in the development process, the maintenance cost is smaller [30], [37], [53]. That's, an error which shows up in the requirements phase might cost one tenth of the same error when detected in the delivery phase [30]. Therefore, the building of a logical model which clearly communicates to users what the system will and will not do is very important in terms of the cost of fixing errors later on.

For the present experiment, in order to increase the quality assurance, hence decreasing the costs, a continuous communication between the developer and the users is established. Moreover, with all the iterative testing and validation process and specification documents that are offered by the approach under experiment, we can conclude that the cost-benefit ratio is reduced.

6.5 RESULTS

What is said for the theoretical proposed model [83], namely the cocktail approach, is applied to the computer aided medical training system development. All the necessary specification documents are developed. Actually, the results are promising; that's, the experiment showed that the cocktail approach or SEDA, combines the advantages of both the structured approach and the evolutionary prototyping approach, that is the two combined approaches are complementary and when combined, then applied to develop a software product or project they yield a product with

high performance, robustness, functionality and ease of use, and ease of learning. That's, the characteristics of the approach under study are:

1-Enforcement of formal documentation.

2-Deeper understanding of the system.

3-Ensures effective and robust planning and the stages of development are more visible.

4-The development tool is chosen early in the project after understanding the domain knowledge; typically during the feasibility study.

5-Robustness to sudden and major changes and it is a demonstrable approach.

6-Provides a superior environment for knowledge elicitation through the mechanism of the demonstration of the working versions of the system.

7-Allows for greater and controllable flexibility in project planning.

8-Testing is spread out throughout the project development process.

9-Ensures the progress in the development of the product.

10-The user is involved throughout the development process.

11-Reduces the cost/benefit ratio.

The last point is not really experienced since the work is done for the sake of research, but deductions can be drawn from the present study and from the characteristics of the component approaches which consist the cocktail approach that is under development.

CONCLUSION

Software systems development is not a mechanical process but a human activity; and requires clear thinking, work, and rework to be successful. Despite all the research work that is done during the last three decades, the battle is not yet ceased and it is not for tomorrow. The software community and the software users are still not satisfied, they are always looking for a better tool. In order to get a better software product, we need to use a better approach for its development; but the problems change with time and time is not waiting for us, that's, we are always late with respect to time.

In the present work, it is our turn to suggest an approach. After considering a number of objectives and problems of software production, the software crisis and the existing paradigms. We are suggesting a cocktail approach whose components are the structured approach and the evolutionary prototyping approach. This approach, namely structured evolutionary development approach, or simply SEDA, is actually discussed theoretically then applied to the development of a medical CAL application,

namely computer aided medical training system, or simply CAMTS. It is interactive and microcomputer-based training system designed to teach surgery medical personnel in a hospital, students and physicians. Its menu-driven structure allows the user to learn easily and has a high level of abstraction, these characteristics are inherited from the DBASE IV environment which offers a very high level language and accepts a lot of data. As may be noticed, the section 6.5 deals only with the advantages of the approach under study. Does this mean that this approach has no disadvantages ? Or shall we deduce them from the component approaches ? The answer for the second question is no, because the disadvantages of each component approach are drawn in the advantages of the other component approach when combined. And for the first question, on the basis of one experience, we cannot give a clear and complete answer. However, we can say that the disadvantages of the proposed approach will be deduced by applying it to the development of software products and projects in different application domains and let the users conclude after using the systems. The purpose of this experiment is to evaluate the promising approach, named SEDA, to software products and projects development. That is our objective is mainly to contribute in the improvement of software projects and products development; then to contribute also in CAL systems development which is under research. We hope that such learning or CAL systems will be developed and used in hospitals, schools, and enterprises in order for everyone to update his knowledge and to be aware of the state of his profession. The product is put under acceptance testing, namely alpha testing

process before its final delivery, and this is done for several times, that's, for each version of the system. Both parts, the developers and the users are satisfied. This does not mean that this approach is ideal and that the developed application is free of errors, well that's what we want, particularly we will be happy. Only experience can tell. But still a famous question is not yet answered: Is this approach apply for all types of software applications ? The answer can only be obtained by statistics, hence, we invite the software community to use this approach in different application domains to develop software projects and products, then conclusions will be drawn, that's, experience in time will tell.

B I B L I O G R A P H Y

- [1] Simons, G.
"Introducing software engineering", 1987, NOCL publications.
- [2] Peters, L.
"Advanced structured analysis and design", 1989, Prentice-Hall International.
- [3] Ghezzi, C.; Jazayeri, M.; and Mandrioli, D.
"Fundamentals of software engineering", 1991, Prentice-Hall International.
- [4] Andriole, S. J. and Freeman, P. A.
"Software systems engineering: the case for a new discipline"; Soft. Eng. Journal, May 1993, vol. 8, No 3, p 165-179.
- [5] Sommerville, I.
"Software engineering", 1989, Third edition, Addison Wesley.

[6] Pressman, R. S.

"Software engineering: a practitioner's approach"; 1987,
second edition, McGraw-Hill International.

[7] Alagic, S. and Arbib, M. A.

"The Design of Well-Structured and Correct Programs", 1978,
Spring-Verlag International.

[8] Macro, A.

"Software engineering: concepts and management", 1990,
Prentice-Hall International, Ltd.

[9] Dwain, S. W.

"New techniques in software project management", 1987, John
Willey and Son International.

[10] Jensen, R. W. and Charles, T. C.

"Software Engineering", 1979, Prentice-Hall International.

[11] Bell, D.; Morrey, I.; and Pugh, J.

"Software Engineering: A programming approach", 1992, Second
edition, Prentice-Hall International (UK) Ltd.

[12] Miller, G. A.

"The magical number seven, plus or minus two: some limits
on our capacity for processing information", Psychol. Rev.,
1956, vol. 63, no 2, p 81-96.

- [13] Schwartz, J. T.
"The practical and the not-yet-practical in software engineering" in Computing tools for scientific problem solving, 1990, Academic press limited, p 23-38.
- [14] Alexander, H. and Jones, V.
"Software design and prototyping using mee too", 1990, Prentice-Hall International.
- [15] Davis, A. M.
"Software requirements: analysis and specification", 1990, Prentice-Hall International
- [16] Smith, H. T. and Green, T. R. G.
"Human interaction with computers", 1980, Acccademic Press.
- [17] Bruce, P. and Pederson, S. M.
"The software development project planning and management", 1982, John Wiley and sons, International.
- [18] Wolff, J. G.
"Towards a new concept of software", Software Engineering Journal, January 1994, vol. 9, no 1, p 27-38.
- [19] Gilb, T.
"Software Metrics", 1977, Windrop Publishers, International.

- [20] Glass, R. L. and Noiseux, R. A.
"Software maintenance guidebook", 1981, Prentice-Hall,
International.
- [21] Tarek, K. A. H.; Sengupta, K.; and Ronan, D.
"Software Project Control: An Experimental Investigation of
Judgment with Fallible Information", IEEE Transactions on
Software Engineering, June 1993, vol. 19, no 6, p 603-612.
- [22] Grady, R. B.
"Practical results from measuring software quality",
Communications of the ACM, November 1993, vol. 36, no 3,
p 62-68.
- [23] Shortliffe, E. H.
"Consultation systems for physicians: the role of AI
techniques", in readings in AI, Webber, B, L. and Nilsson,
N. J. (Editors), 1981, Tioga Publishing Co., p 323-333.
- [24] Cutts, G.
"Structured systems analysis and design methodology", 1988,
Paradigm publishing Ltd.
- [25] Claybrook, B. G.
"File management techniques", 1983, John Willey and Sons,
International.

- [26] Buckle, J. K.
"Software configuration management", 1982, Macmillan
Education Ltd.
- [27] Hilal, D. K. and Soltan, H.
"To prototype or not to prototype ? That's a question",
Soft. Eng. Journal, November 1992, p 388-392.
- [28] Boehm, B. W.; Gray, T. E.; and Seewaldt, T.
"Prototyping versus Specifying: A multiproject
Experiment", IEEE Transactions on software engineering, May
1984, vol. SE-10, no 3, p 290-302.
- [29] Anderson, W. L. and Grocca, W. T.
"Engineering practice and codevelopment of product
prototype", Communications of the acm, june 1993, vol. 36,
no 4, p 49-66.
- [30] Gane, C. and Sarson, T.
"Structured systems analysis: tools and techniques", 1979,
Prentice-Hall International.
- [31] Haulibaugh, R.
"Application of reusable software components at the SEI",
Software reuse issues, Dec. 1989, p 135-145.
- [32] Whittle, B. and Ratcliffe, M.
"Software component interface description for reuse", Soft.

Eng. Journal, Nov. 1993, vol. 8, no 6, p 307-318.

[33] Prieto-Diaz, R.

"Status Report: Software Reusability", IEEE Software, May 1993, p 61-66.

[34] Azni, M.

"A model based formal specification of a medical expert system", 1994, Magister thesis, Advisor: Mezhoud, B., INELEC, Algeria.

[35] Spivey, J. M.

"An introduction to Z and formal specification", Software Engineering Journal, 1989, p 40-50.

[36] Spivey, J. M.

"The Z notation: a reference manual", 1989, C. A. R. Hoar Series Editor.

[37] Yourdon, E.

"Modern structured analysis", 1989, Prentice-Hall International.

[39] Longworth, G.

"A user's guide to SSADM: getting the system you want", 1989, NCC publications.

- [40] DeMarco, T.
"Structured analysis and system specification", 1978,
Yourdon Press.
- [41] Downs, ED; Clare, P.; and Core, I.
"SSADM: Application and Context", 1992, Second edition,
Prentice-Hall International (UK) Ltd.
- [42] Millington, D.
"Systems analysis and design for computer applications",
1981, Ellis Horwood Ltd.
- [43] Stevens, W. P.
"Using structured design", 1988, John Wiley and sons
International.
- [44] Wayne, S.
"Software design: concepts and methods", 1991, Prentice-Hall
International.
- [45] Byces, B. B.
"Flowcharting: Programming, software designing, and
computer problem solving", 1975, John Willey and sons,
International.
- [46] Wichmann, B. A.
"Contribution of standard programming languages to software
quality", Software engineering Journal, January 1994, vol.9,

no 1, p 3-12.

[47] Diaz-Herrera, J. L.

"The importance of static structures in software construction", IEEE software, May 1993, p 75-87.

[48] Kowal, J. A.

"Analyzing Systems", 1988, prentice-Hall International.

[49] Delannoy, C.

"Maitriser DBASE IV", 1983, Berti editions (Paris).

[50] Knight, J. C. and Ann Myers, E.

"An improved inspection technique", Communications of the ACM, November 1993, Vol. 36, no 11, p 51-61.

[51] Grady, R. B.

"Practical results from measuring software quality", Communications of the ACM, November 1993, Vol. 36, no 11, p 62-68.

[52] Harrold, M. J. and Malloy, B.

"A Unified Interprocedural Program Representation for a Maintenance Environment", IEEE Transactions on Software Engineering, June 1993, vol. 19, no 6, p 584-593.

[53] Banker, R. D.; Datar, S. M.; Kemerer, C. F.; and Zweig, D.

"Software complexity and maintenance costs", Communications

[54] Polack, F.

"Integrating formal notations and system analysis: using entity relationship diagrams", Soft. Eng. Journal, Sept. 1992, vol. 7, no 5, p 363-371.

[55] Pin-Shan Chen, Peter

"The Entity-Relationship Model: Toward a unified view of data", ACM Transactions on database systems, March 1976, vol. 1, no 1, 1976, p 9-36.

[56] Davis, C. G.; Jajodia, S.; NG, P. A.; and Yeh, R. T.

"Entity-Relationship approach to software engineering", 1983, Elsevier science publisher B. V.

[57] Banos, D. and Malbose, G.

"MERISE PRATIQUE: Les points-clé de la méthode", 1990, Troisième Edition, Editions Eyrolles (Paris).

[58] DBASE IV, 1988, Ashton-Tate corporation.

[59] Lilien, H.

"DBASE IV: Le guide complet de l'utilisateur", 1989, Editions Radio (Paris).

[60] Hordeski, M. F

"Control system interfaces: design and implementation using

personnal computers", 1992, Prentice-Hall International.

[61] Gane, C.

"CASE: the methodologies, the products, and the future",
1990, Prentice-Hall International.

[62] LBMS

"Automate-Plus release notes version 3.02, vol. 1-3",
1987, Copyright (c) LBMS, International.

[63] The start guide, 1987, sec. edit., vol. 2; NCC publications.

[64] Gage, N. L.

"Explorations in teachers' perceptions of pupils", in
reading in Social perception, Hans, Toch and Henry Clay,
Smith, 1968, D. Van Nostrand Company International, p 219-
226.

[65] Davis, W. S.

"Systems analysis and design: a structured approach", 1987,
Addison Wesley.

[66] Pask, G.

"Styles and strategies of learning", Br. J. educ. Psychol.,
1976, 46, p 128-148.

[67] Chandrasckaran, B.; Johnson, T. R.; and Smith, J. W.

"Task-Structure Analysis for Knowledge Modelling",

Communications of the ACM, September 1992, vol. 35, no. 9,
p 124-136.

[67] Ozejdo, B.; Eick, C. F.; and Taylor, M.

"Integrating Sets, Rules, and Data in Object-Oriented
Environment", IEEE Expert, February 1993, p 59-66.

[68] Bahr, J. P.

"Computer-Aided Instruction in Perinatal Education",
American journal of perinatology, April 1986, vol. 3, no 2,
p 147-150.

[69] Irani, K. B.; Cheng, J.; Fayyad, U. M.; and Qian, Z.

"Applying Machine Learning to Semiconductor Manufacturing",
IEEE Expert, February 1993, p 41-46.

[70] Matheny, J. L.

"Comparison of different Approaches to pharmacological
Instruction of Medical Students", Health Comm. Informatics,
1979, vol. 5, p 9-13.

[71] Haton, M. C.

"L'Ordinateur pedagogique", La recherche 246, sept. 1992,
vol. 23, p 1014-1022.

[72] Lincoln, M. J.; Turner, C. W.; Haug, P. J.; Warner, H. R.;

Williamson, J. W.; Bouhadaddou, O.; Jessen, S. G.;
Sorenson, D.; Cundick, R. C.; and Grant, M.

"Iliad training enhances medical students: diagnostic skills"; Journal of medical systems, 1991, vol. 15, no 1, p 93-110.

[73] Daly, D. W.; Dunn, W.; and Hunter, J.

"The CAL project in mathematics at the university of Glasgow", Int. J. Math, Educ. Sci. Technolo., 1977, vol. 8, no 2, p 145-156.

[74] Pask, G; and Scott, B. C. E.

"Learning strategies and individual competence", Int. J. Man-machine studies, 1972, vol. 4, p 217-253.

[75] Shortliffe, E. H.; Davis, R.; Axline, S. G.; Buchanan, B.G.; Green, C. C.; and Cohen, S. N.

"Computer-based consultations in clinical therapeutics: explanation and rule acquisistion capabilities of the MYCIN", Computers and biomedical research, 1975, vol. 8, p 303-320.

[76] Manning, P. R. and Hoagland, P. I.

"Continuing Medical Education: the next step", AAMSI Congress 83 proceedings, p 339-344.

[77] Karmakar, N. L.

"Use of Information Technology in Health and Medical Education at Australian Universities", Interactive Learning International, 1991, Vol. 7, p 265-266.

- [78] Mansour, A. A. H.; McGregor, J.; Franklin, M.; & Poyser, J.
"Intelligent Medical Multimedia-Based Tutoring Systems:
Design issues", IEE Colloquium on 'Intelligent Decision
Systems and Medicine', 1992, Digest no 143, p 101-103.
- [79] Mansour, A. A. H; Poyser, J.; McGregor, J. J.; & Franklin,
M. E.
"An intelligent tutoring system for the instruction of
medical students in techniques of general practice",
Computers Education, 1990, vol. 15, no 1-3, p 83-90.
- [80] Ellis, L. B. M. and Fuller, S.
"Computers in Medical Education: A cooperative approach to
planning and implementation", Proceedings of the 12th annual
symposium of computer applications in medical care, 1988,
IEEE cat. no 88 CH6161, p 323-327.
- [81] Kunstaetter, R.
"Intelligent Physiologic modelling: An application of
knowledge based systems technology to medical education",
Proceedings of the 10th annual symposium on computer
applications in medical care, 1986, cat. no 86 CH2341-6,
p 381-393.
- [82] Martin, J. M.; Jabot, F.; and Marrel, P.
"How to organize the medical data of chronically ill
patients in the computer", Methods of Information in
Medicine, 1985, vol. 24, no 1, p 5-12.

[83] Imache, R.; Mezhoud, B.; and Maaoui, M.

"Computer aided medical training system", To be published
in Modelling, Measurement and Control, C, Vol. 48, No 4,
1995, p 51-63.

[84] ORR, K. T.

"Structured systems development", 1977, Yourdon Press
International.

MODELLING TECHNIQUES GUIDELINES

The present appendix is a complement to the techniques presented in chapter four. However, the guidelines provided in order to develop or draw data flow diagrams [29], [80], structure charts [5], and data dictionaries [5] are presented.

A.1 Guidelines For Drawing Data Flow Diagrams

- 1-Identify the external entities involved.
- 2-Identify the scheduled inputs and outputs that you can expect.
- 3-Specify which information is given to the system and the one required from the system.
- 4-Take a large sheet of paper. Start on drawing the DFD from the left hand side with the external entity that seem to be the prime source of inputs, then the data flows, processes, and the data stores that you think will be required.
- 5-Draw the first draft freehand.
- 6-Accept that you will need at least three drafts of the high

level data flow.

7-Check back the first draft if it has included all the listed inputs and outputs.

8-Produce a clear second draft with minimum number of crossing data flows.

- Duplicate external entities if necessary.

- Duplicate data stores if necessary.

- Allow data flows to cross if there is no alternative [29], sec 3.4, p 34.

9-Agree the boundary of the system.

10-Draw a context diagram or level 0 DFD.

11-Draw a level 1 DFD by exploding the context diagram.

12-Decompose down to levels 2,3, etc.

13-Stop the decomposition process at suitable points [30]

14-Support each bottom level process with pseudocode.

A.2 Structure Charts

1-The symbol for a module is a rectangle.

2-Each module must have a label.

3-Module labels are composed of a transitive, or action, verb and an object noun.

4-Modules may communicate information only via CALLs.

5-A module may not call or invoke another module which is higher in the hierarchy than it is.

6-No two modules, data couples, or control couples may have the same name.

7-No data couple may have the same name as a control couple.

8-The operation which takes place in a module must be described with pseudocode.

9-All data couples, control couples, pseudocode, etc., must be defined in the data dictionary [5], sec 9.5 p 169.

A.3 Data Dictionary

A set of conceptual standards that form a basic dictionary consistent with the methods presented are listed in the following guidelines:

1-All data and other information that comprise the analysis, design, and code must be defined in the data dictionary.

2-All data and information that are contained in the data dictionary must be used in some part of the system.

3-The information in the data dictionary should be internally consistent, complete, nonredundant, and correct.

4-The data dictionary is intended to be the primary project information resource.

5-The DD may comments regarding data the software developer deems necessary to understanding and using the information contained in it.

6-The names used for data items within the DD and the names used on the data flows must be identical.

7-The names used for data items within pseudocode must be identical to those used on the DFDs and the DD, [5] sec 3.4, p40.

SYSTEM FLOWCHARTS

This appendix is a complement to the design phase of the medical CAL application, namely Computer Aided Medical Training System, or simply CAMTS, that is developed in chapter six. However, the application flowchrts are given in the following pages.

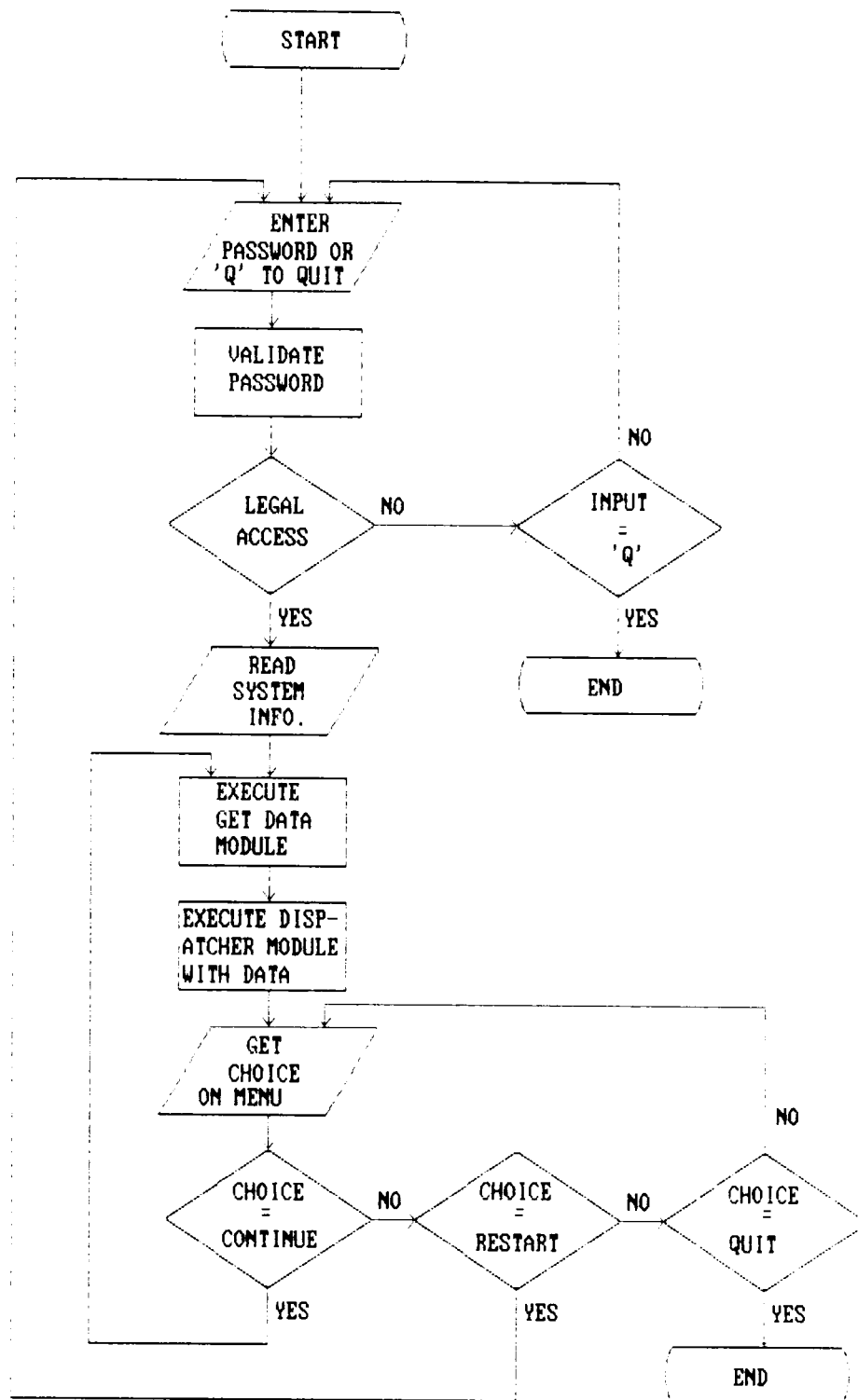


Figure B.1: Main program (Driver) module flowchart

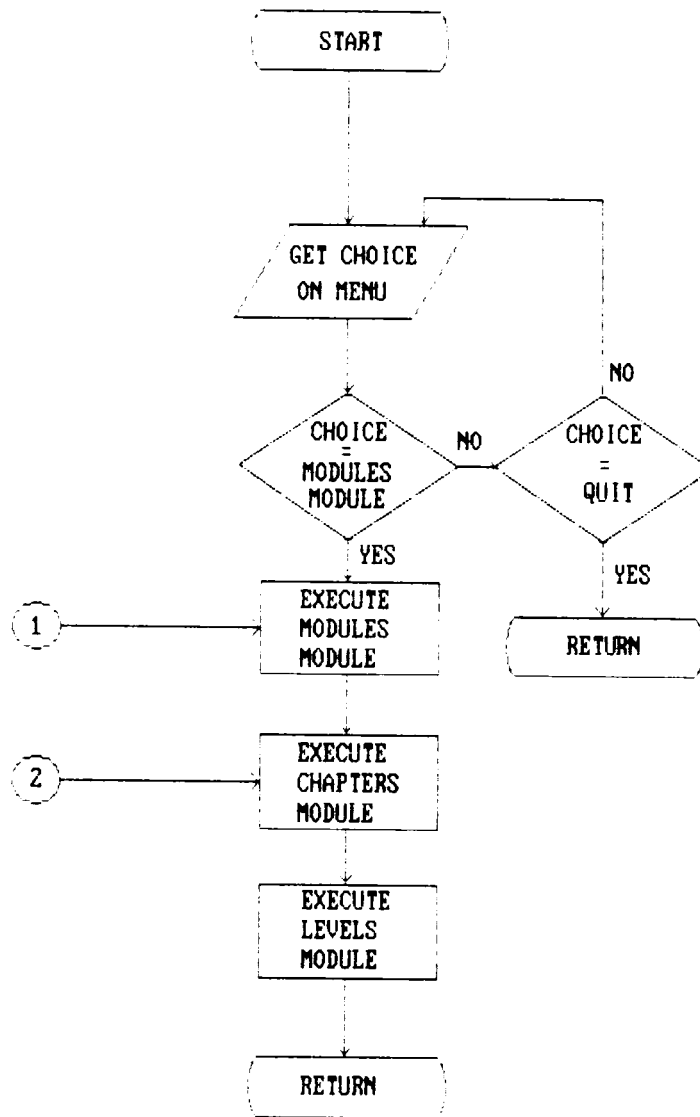


Figure B.2: Data module flowchart

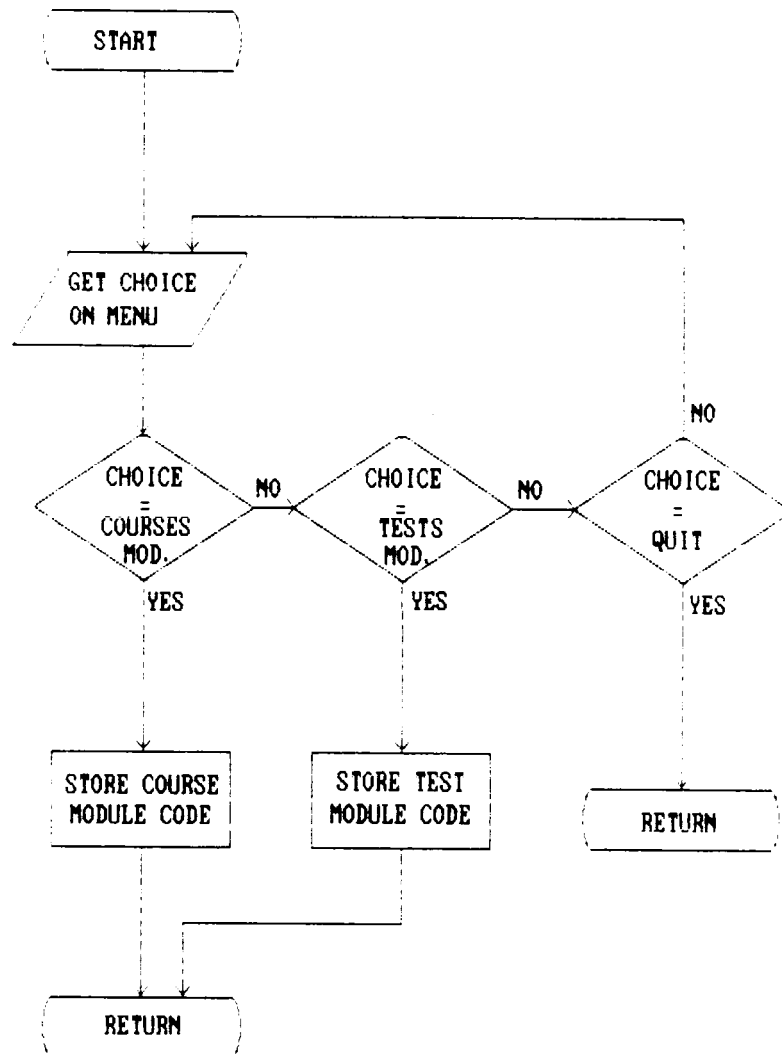


Figure B.3: Modules module flowchart

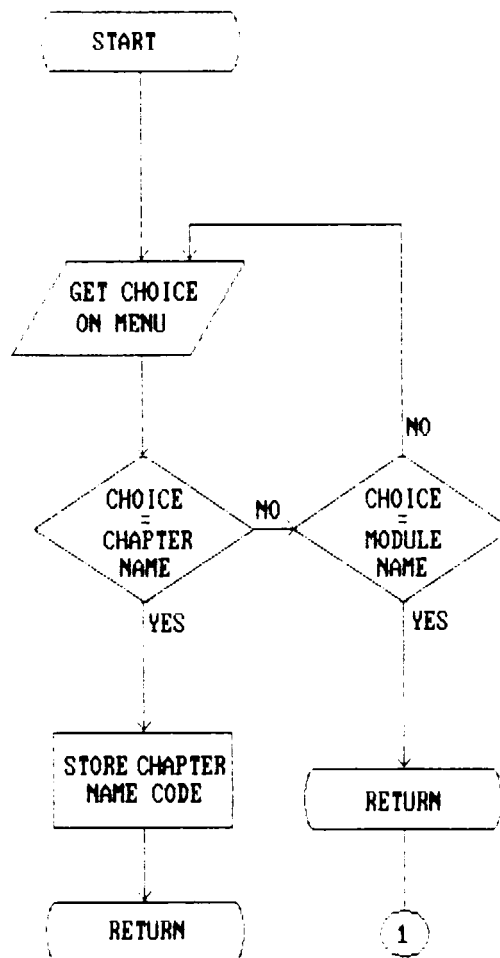


Figure B.4: Chapters module flowchart

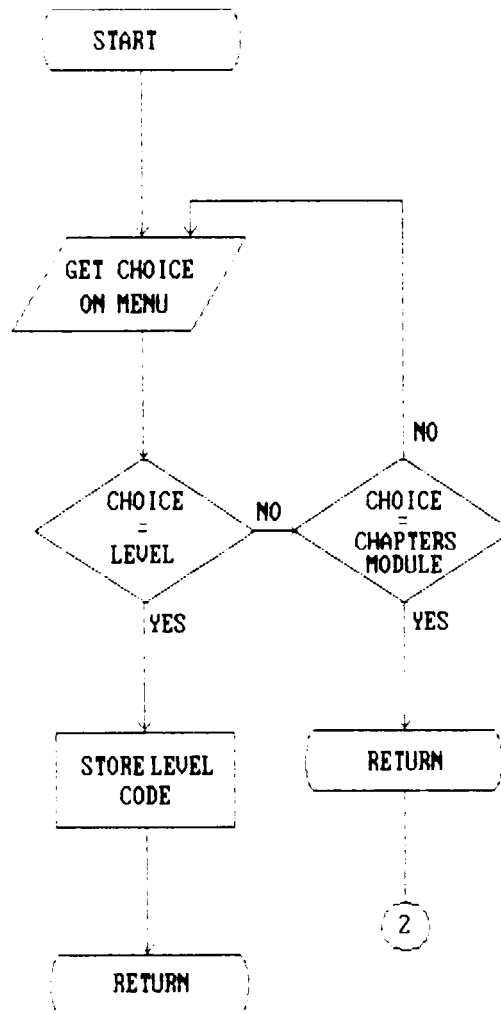


Figure B.5: Levels module flowchart

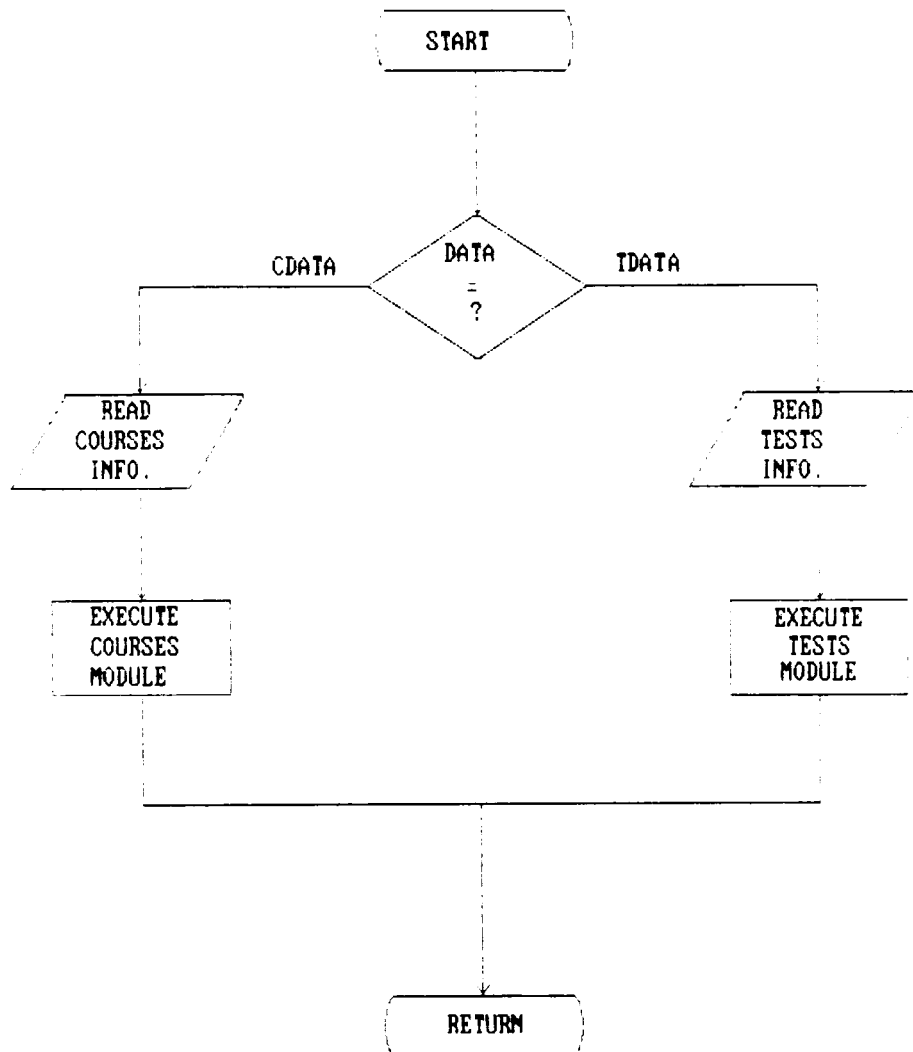


Figure B.6: Dispatcher module flowchart

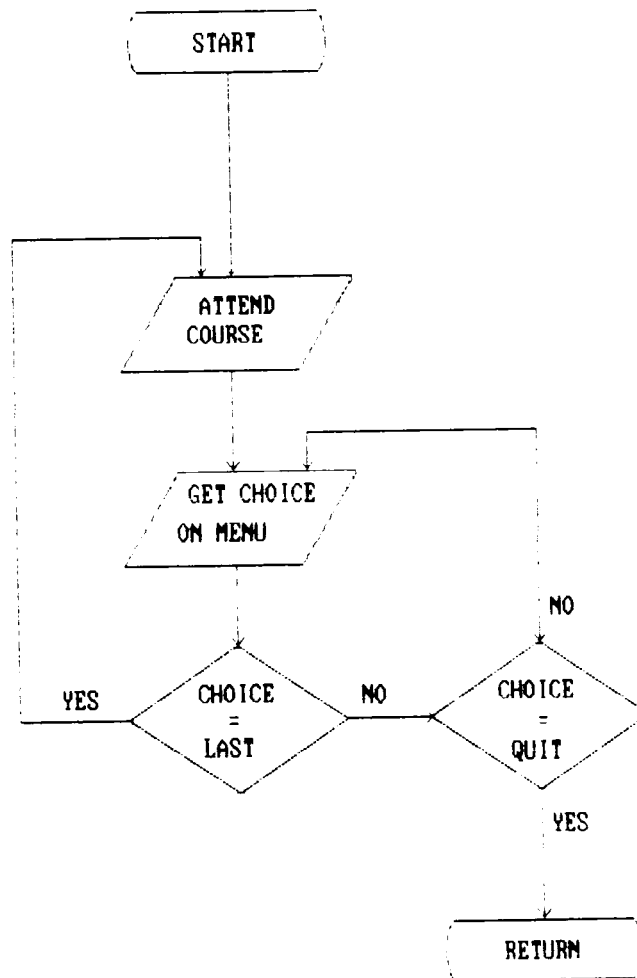


Figure B.7: Courses module flowchart

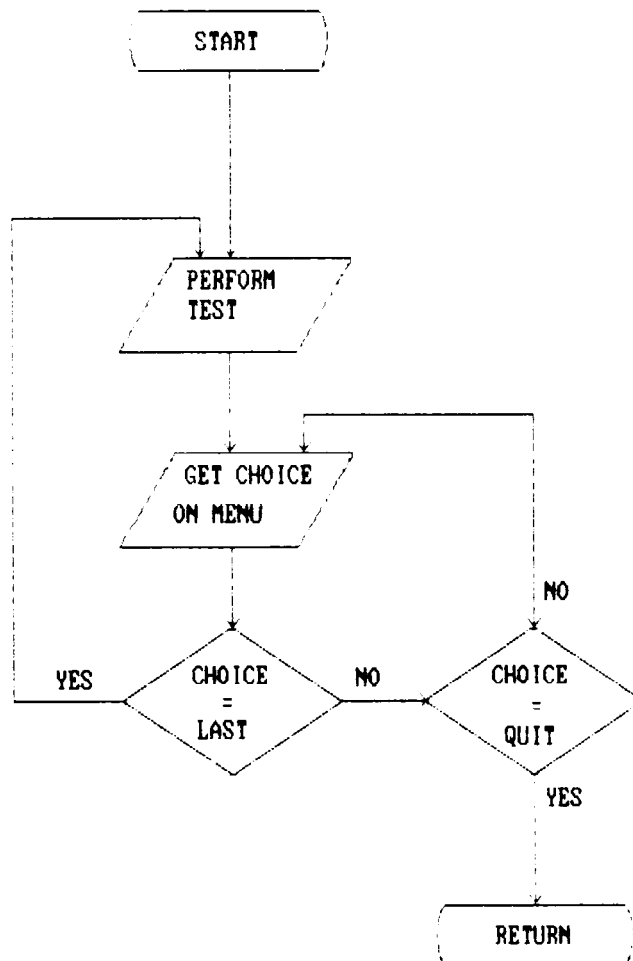


Figure B.8: Test module flowchart

A N N O U N C E

Le Centre de Recherches et d'Etudes de l'Institut National de l'Electronique (I.N.E.)

est ouvert au public pour l'Ingenierie des Systemes Electroniques

(I.N.E.) C. INGENIERIE. Rabah.....

PRESIDENT /: MR A. BOULARAS, (Ph, U.S.T.H.B)

RAPPORTEURS/: MR HOLCOMBE , (Ph, U.Sheffield (U.K))

MR NE. BOUC (Ph, U.S.T.H.B) - MR T. BOUC

MEMBRES /: MR S. ACHOUR, (Phd, C.C - U.T.O)

MR H. AZZOUNE, (T.U, C.C - U.S.T.H.B)